TU Wien

Business Informatics Group
Institut für Softwaretechnik und Interaktive Systeme

Java Enterprise Anwendungen und UML 2.0 - ein starkes Team?

Diplomarbeit zur Erlangung des akademischen Grades eines Magister (Mag. Rec. Oec. Soc)

eingereicht bei o. Univ.-Prof. Mag. Dipl.-Ing. Dr. Gerti Kappel
mitbetreuender Assistent: Mag. Manuel Wimmer

Christian Sokop
Wien, Oktober 2007

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

<Ort, Datum> <Eigenhändige Unterschrift>

Danksagung

Mein besonderer Dank geht an meine gesamte Familie, insbesondere an meine Eltern, Eva und Alfred Sokop, die mich zuerst finanziell und in weiterer Folge Emotional unterstützt haben.

Weiters möchte ich mich bei Mag. Manuel Wimmer für die fachliche Betreuung dieser Arbeit, und Daniel Stevens für die Hilfestellung bei der englischen Übersetzung bedanken.

Zum Abschluss möchte ich mich bei all meinen Freunden und Kollegen für die schönen, kurzweiligen, intensiven, feucht-fröhlichen und besten Stunden während meiner Studienzeit bedanken, und die Hoffnung aussprechen, dass es für immer so lustig weitergeht!

Abstract

Today not only the world but also the requirements for computer-based applications change permanently. In addition to these requirements, the development of technologies and tools also continues. Modern and object-oriented technologies, such as UML and Java meet today's high standards.

Furthermore, the object-oriented programming language Java, or more precisely the Java Enterprise Edition, is the perfect tool to develop distributed systems including web applications. Also the object-oriented modelling language UML is able to keep up with the rapid development.

This thesis evaluates the interaction between UML 2.0 and Java Enterprise applications.

The language UML 2.0 offers a total of 13 different classes of diagrams. Each of these diagrams presents some strengths but also some weaknesses in relation to Enterprise applications.

This thesis shows which parts of an Enterprise application can be modelled using which kind of UML 2.0 diagram as well as their incompatibilities. Besides the interaction between UML 2.0 and a web application developed by means of Java Enterprise technology, this thesis also presents the possibilities to model general aspects of an Enterprise application, such as design patterns and system architectures. In this process, all available diagrams offered by UML 2.0 for a given example are modelled.

In addition, an overview of the various diagrams in UML 2.0 in the Rational Unified Process is a further objective of this work. In this realm, the question of which kind of diagram is applicable in which phase of the unified process is evaluated. The unified process will be examined from an Enterprise application point of view.

Kurzfassung

In der heutigen Zeit ändern sich Anforderungen an Anwendungen laufend. Neben den Anforderungen haben sich natürlich auch die Hilfsmitteln und Technologien weiterentwickelt. Moderne objektorientierte Sprachen wie z.B. UML und Java werden den heutigen Ansprüchen gerecht.

Die objektorientierte Programmiersprache Java, oder besser gesagt die Java Enterprise Edition, eignet sich hervorragend um verteilte Systeme, zu denen auch Webanwendungen zählen, zu entwickeln. UML 2.0 ist mit ihren aktuellen Diagrammarten ebenfalls sehr gut für die Modellierung von verteilten Anwendungen geeignet.

Diese Arbeit evaluiert das Zusammenspiel zwischen UML 2.0 und Java Enterprise Anwendungen.

In UML 2.0 gibt es insgesamt 13 verschiedene Diagrammarten. Jedes einzelne Diagramm hat Stärken und Schwächen in Bezug auf Enterprise Anwendungen. Diese Arbeit zeigt, welche Teile einer Enterprise Anwendung sich mit welchem Diagramm in UML 2.0 modellieren lassen bzw. welche nicht.

Zusätzlich geht diese Arbeit auch auf die Möglichkeiten ein, allgemeine Aspekte einer Enterprise Anwendung zu modellieren, wie z.B. Design Patterns und Systemarchitekturen. Dabei werden für ein bestehendes Beispiel die einzelnen Diagramme, die UML 2.0 zur Verfügung stellt, modelliert.

Ein weiteres Ziel der Arbeit ist, einen generellen Überblick von UML 2.0 Diagrammarten im Rational Unified Process zu geben. Dabei wird evaluiert, welches Diagramm in welcher Phase des Unified Process in Bezug auf Enterprise Anwendungen hilfreich sein kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Das folgende Kapitel beschreibt als Einleitung die Motivation und die Ziele der vorliegenden Arbeit.

1.1 Motivation

In der heutigen Zeit ändern sich Anforderungen an Anwendungen laufend. Neben den Anforderungen, haben sich natürlich auch die Hilfsmitteln und Technologien weiterentwickelt. Moderne objektorientierte Sprachen wie z.B. UML und Java werden den heutigen Ansprüchen gerecht.

Die objektorientierte Programmiersprache Java, oder besser gesagt die Java Enterprise Edition, eignet sich hervorragend um verteilte Systeme, zu denen auch Webanwendungen zählen, zu entwickeln.

Auch die objektorientierte Modelliersprache UML, bei der Strukturdiagramme und Verhaltensdiagramme unterschieden werden können, passt sich der schnellen Entwicklung an. Das kürzlich erschienene UML 2.0 bietet neben Änderungen bei bestehenden Diagrammen, auch Neuheiten wie das Kompositionsstrukturdiagramm.

Wie mehrere Jahre praktischer Erfahrung gezeigt haben, werden Java Enterprise Anwendungen sehr technologiegetrieben entwickelt. In den meisten Fällen beginnen Programmierer mit der Implementierung zu einem Zeitpunkt im Projekt, zu dem nur wenige Anforderungen existieren und die existierenden Anforderungen wenig bis gar

nicht analysiert wurden. Im Laufe der Zeit werden mehr und mehr Anforderungen bekannt, die sofort umgesetzt werden. Dadurch ist, vor allem in großen Projekten, ein ständiges neu organisieren der Teilprojekte notwendig und die Implementierung wird immer unübersichtlicher. Diese Form der Implementierung ist mit hohen Zeit- und Ressourcenkosten verbunden.

Als unterstützendes Werkzeug, um dieses Szenario zu vermeiden, dient die objektorientierte Modellierungssprache UML. Sie bietet Werkzeuge, in Form von Diagrammen an, um Softwaresysteme zu modellieren.

Mit UML modellierte Projekte sind leichter wartbar und übersichtlicher. Die modellierten Diagramme dienen auch zur Kommunikation zwischen den Beteiligten in einem Projekt. Als Beispiel kann hier das Anwendungsfalldiagramm genannt werden. Diese Diagrammart ist sowohl für den Kunden, der meist über wenige EDV-Kenntnisse verfügt, wie auch für den Softwareentwickler lesbar und verständlich. Auch der Einstieg in das Projekt für neue Mitarbeiter wird durch die Modellierung, die auch gleichzeitig als Dokumentation dient, erleichtert.

Weiters befinden sich zahlreiche Frameworks auf dem Markt, die bei der Implementierung einer Java Enterprise Anwendung unterstützen. Diese sind unter anderem:

- Das Framework *Hibernate* ermöglicht die Abbildung von Objekten der Enterprise Anwendung auf relationale Datenbanken (OR/Mapping).
- Das Framework *Spring* erlaubt es unter anderem Java Enterprise Beans über Konfigurationsdateien zu erzeugen, sodass die Business Logik als reine POJO (*Plain Old Java Objects*) - Entwicklung realisiert werden kann.
- *Apache Struts* bzw. *Java Server Faces* bieten eine gute Möglichkeit die Präsentationsschicht zu abstrahieren.

Diese genannten Frameworks machen die Modellierung in UML 2.0 noch komplexer. Theoretisch müsste man, neben dem eigenen Projekt, auch die verwendeten Frameworks in den Modellen unterbringen.

Das Problem ist die Modellierung von Enterprise Anwendungen mit der Modellie-

rungssprache UML 2.0 ohne Verwendung von UML Profilen. Die Sprache UML lässt Erweiterungen und Anpassungen mit so genannten Profilen zu, diese sollen in der vorliegenden Arbeit nur am Rande erwähnt werden.

In UML 2.0 gibt es insgesamt 13 verschiedene Diagrammarten. Jedes einzelne Diagramm hat Stärken und Schwächen in Bezug auf Enterprise Anwendungen. Es gibt keine Zusammenfassung für welche Teile einer Enterprise Anwendung welche Diagramme zu verwenden sich bzw. welche Diagramme sich nicht für die Modellierung einer Enterprise Anwendung eignen.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, das Zusammenspiel von UML 2.0 und einer mittels Java Enterprise Technologie entwickelten Webanwendung zu evaluieren. Hier sollen insbesondere die Neuigkeiten von UML 2.0 beachtet werden.

Zusätzlich wird versucht allgemeine Aspekte einer Enterprise Anwendung in UML 2.0 umzusetzen. Zu allgemeinen Aspekten einer Enterprise Anwendung zählen insbesondere die Modellierung von Design Patterns und die Modellierung der Architektur. Hier gilt es herauszufinden, welche Diagrammart sich für die Modellierung welcher Schicht einer mehrschichtigen Enterprise Anwendung, und welche Diagrammart sich für die Modellierung des Zusammenspiels der Schichten eignet.

Dies soll anhand eines schon bestehenden Beispiels geschehen, welches die oben genannten Frameworks verwendet. Für dieses Beispiel werden beispielhaft die einzelnen Diagramme, die UML 2.0 zur Verfügung stellt, modelliert. Dadurch soll evaluiert werden, welches Diagramm sich für welchen Abstraktionsgrad der Anwendung anbietet.

Ein weiteres Ziel der Arbeit ist einen Überblick von UML 2.0 Diagrammarten im Rational Unified Process zu geben. Die Modellierungssprache UML 2.0 stellt keinen Softwareentwicklungsprozess zur Verfügung, sondern nur die einzelnen Diagramme. Es soll evaluiert werden, welches Diagramm sich für welche Phase im Unified Process hilfreich sein kann. Der Unified Process soll in Bezug auf Enterprise Anwendungen betrachtet werden.

In Bezug auf das konkrete Beispiel, welches eine mehrschichtige Enterprise Anwendung ist, sollen folgende Fragen beantwortet werden:

- Welches Diagramm ist für welche Schicht zu verwenden bzw. welche Schicht kann mit welchen Diagrammen modelliert werden?
- Mit welchem Diagramm kann das Zusammenspiel der Schichten modelliert werden?

Aus Sicht einer Enterprise Anwendung sollen folgende Fragen beantwortet werden:

- Mit welcher Diagrammart kann die Architektur einer Enterprise Anwendung beschrieben werden bzw. wie können Designpatterns in UML 2.0 modelliert werden?
- Welche Aspekte der Enterprise Anwendung können mit welcher Diagrammart modelliert werden?

Für den Softwareentwicklungsprozess Unified Process soll folgende Frage beantwortet werden:

- Welche Diagrammart erweist sich in welchen Arbeits- bzw. Teilarbeitsschritt im Unified Process als hilfreich?

Alle vorliegenden Diagramme sind als Beispiel zu verstehen. Es wird versucht für jedes Diagramm in UML 2.0 ein passendes Beispiel zu finden. Nach der Modellierung wird versucht das Diagramm nach den oben genannten Punkten zu bewerten.

Kapitel 2

Technologien

Das folgende Kapitel soll eine Einleitung über die behandelten Technologien geben.

2.1 UML 2.0 - Unified Modelling Language

Als kurze Definition von der Unified Modelling Language gibt [Rupp05] eine *„verbreitete Notation, um Softwaresysteme zu analysieren und zu entwerfen“* an. Die neue Version 2.0 wurde von der Object Management Group OMG im April 2005 offiziell verabschiedet [Rupp05]. Die Object Management Group ist ein internationales Konsortium, welches sich um die Entwicklung von Standards kümmert. Neben der objektorientierten Modellierungssprache UML wurde unter anderem der verbreitete Standard CORBA[1] von OMG entwickelt [OMG07].

Neben einigen Änderungen und Erweiterungen, werden in UML 2.0 auch einige neue Diagramme beschrieben.

2.1.1 Grundlagen von UML 2.0

UML gilt als Modellierungssprache und kann über den gesamten Softwareentwicklungsprozess hinweg verwendet werden. Dadurch ist UML unabhängig von Vorge-

[1] Common Object Request Broker Architecture (plattformunabhängige Architektur für verteilte Anwendungen [OMG07a])

hensmodellen für die Softwareentwicklung und hat auch nicht das Ziel ein eigenes, neues Vorgehensmodell zu definieren. Weiters soll UML unabhängig von Entwicklungswerkzeugen und Programmiersprachen, sowie mit verschiedensten Anwendungsbereichen kompatibel sein [Hitz05]. Diese Punkte spiegeln auch die Vorteile von UML wieder. Die Einsatzmöglichkeiten reichen von der Analysephase bis hin zur Implementierungsbeschreibungen eines Projektes. UML kann sowohl bei Echtzeitsystemen wie auch für verteilte Systeme angewendet werden.

Praktische UML Modellierung mit StarUML

Für die vorliegende Arbeit wurden die meisten Modelle mit dem frei verfügbaren Tool StarUML [Star07] modelliert. Dieses Produkt wurde gewählt, da es eine Vielzahl der zu modellierenden Diagramme unterstützt, und im Gegensatz zu vielen anderen Modellierungstools frei verfügbar ist [Reic06].

2.1.2 Diagrammübersicht

Das folgende Kapitel soll einen kurzen Überblick über die 13 Diagrammarten von UML 2.0 geben. In weiterer Folge wird sich die vorliegende Diplomarbeit mit jedem dieser Diagramme in Bezug auf Java und insbesondere auf Java Enterprise Anwendungen beschäftigen.

Prinzipiell kann in UML zwischen Verhaltens- und Strukturdiagrammen unterschieden werden. Mit Verhaltensdiagrammen wird, wie der Name schon sagt, das Verhalten eines Systems modelliert. Mit Verhaltensdiagrammen werden die dynamischen Aspekte eines Systems beschrieben. Mit Hilfe von Strukturdiagrammen werden die statischen und strukturellen Aspekte eines Systems modelliert [Hitz05].

Zu den Strukturdiagrammen zählen in UML 2.0 Klassen-, Objekt-, Paket-, Kompositionsstruktur-, Komponenten- und Verteilungsdiagramme. Bei Verhaltensdiagrammen wird zwischen Anwendungsfall-, Aktivitäts-, Zustands- und Interaktionsdiagrammen unterschieden. Interaktionsdiagramme werden noch in Sequenz-, Kommunikations-, Zeit- und Interaktionsübersichtsdiagramme eingeteilt.

Die Grenze zwischen den Diagrammen ist fließend. So ist es z.B. möglich Teile eines Klassendiagramms in einem Paketdiagramm oder umgekehrt zu modellieren [Hitz05].

Zusätzlich ist es in UML 2.0 möglich so genannte Profile zu erstellen. Ein Profil ermöglicht die Anpassung von UML-Standardmodellelementen auf spezifische Projektumgebungen [Rupp05].

Die folgende Abbildung zeigt eine Übersicht über die Diagrammarten in UML 2.0.

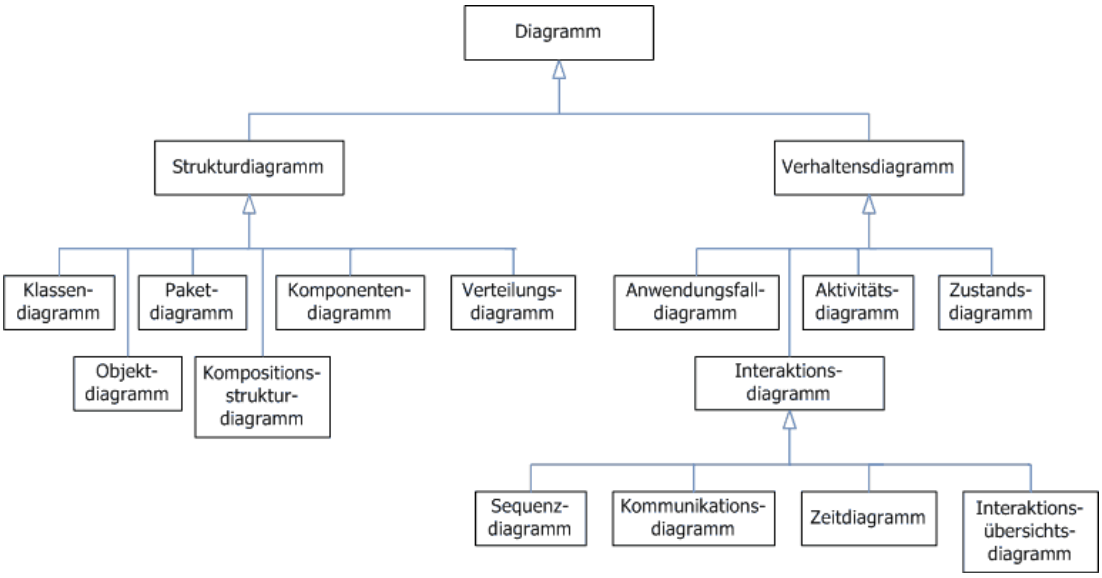

Abbildung 1: Übersicht der Diagramme in UML 2.0 [Hitz05]

Verhaltensdiagramme in UML 2.0

UML 2.0 beschreibt folgende Verhaltensdiagramme:

- *Anwendungsfalldiagramm*: Das Anwendungsfalldiagramm beschreibt die Anwendung in einer hohen Abstraktionsebene. In diesem Diagramm wird der Zusammenhang zwischen Aktoren und Anwendungsfällen beschrieben. Ein Anwendungsfall ist eine *„abgeschlossene, zusammenhängende Einheit, welche einen Teil der Funktionalität des Systems repräsentiert*“ [Zuse01].
- *Aktivitätsdiagramm*: Das Aktivitätsdiagramm dient für die Modellierung von Prozessen und Software [Hitz05]. Diese Diagrammart wurde zum Vergleich zu älteren UML Definitionen enorm verändert und erweitert. Es beschreibt die gesamten (Teil-)Abläufe in einem System mit Hilfe der zeitlichen Abfolge von Aktivitäten, die im System passieren.
- *Zustandsdiagramm*: Beim Zustandsdiagramm werden Zustände und Über-

gänge von Zuständen im System modelliert. Eine Änderung des Zustands kann vom System oder von einem Aktor außerhalb des Systems bestimmt werden [Rupp05].

- *Interaktionsdiagramm*: Das Interaktionsdiagramm ist kein eigenes Diagramm, sondern der Überbegriff für vier weitere Diagramme.

Folgende Interaktionsdiagramme, als Teil der Verhaltensdiagramme werden in UML 2.0 unterschieden:

- *Sequenzdiagramm*: Mit Sequenzdiagrammen werden Interaktionen zwischen Objekten modelliert. Dabei steht die zeitliche Abfolge der Nachrichten im Vordergrund [Hitz05]. Mit dem Sequenzdiagramm soll die Frage „*Wie läuft die Kommunikation in meinen System ab?*“ [Rupp05] beantwortet werden. Das Sequenzdiagramm ist das meistverwendete unter den Interaktionsdiagrammen [Rupp05].
- *Kommunikationsdiagramm*: Das Kommunikationsdiagramm beschreibt, wie auch das Sequenzdiagramm, Interaktionen zwischen Objekten. Bei dieser Diagrammart stehen allerdings die strukturellen Beziehungen der Objekte zueinander im Vordergrund [Hitz05].
- *Zeitdiagramm*: Das Zeitdiagramm dient zum Modellieren von Änderungen der Zustände von Objekten. Besonders geeignet sind diese Diagramme für die Modellierung von Systemen mit zeitkritischem Verhalten (Echtzeitsysteme) [Hitz05].
- *Interaktionsübersichtsdiagramm*: Das Interaktionsübersichtsdiagramm ist eine Variante des Aktivitätsdiagramms. Es soll den Zusammenhang zwischen Interaktionen zeigen. Das Interaktionsübersichtsdiagramm soll die Frage „*In welcher Reihenfolge und unter welchen Bedingungen finden Interaktionen statt?*“ [Rupp05] beantworten.

Strukturdiagramme in UML 2.0

Folgende Diagrammarten werden in UML 2.0 der Strukturmodellierung zugeordnet:

- *Klassendiagramm*: Das Klassendiagramm beschreibt die strukturellen Aspekte des zu entwickelten Systems in Form von Klassen und Interfaces. Außerdem werden Beziehungen zwischen Klassen und Interfaces modelliert [Hitz05]. Ein Klassendiagramm kann stark durch seine Abstraktionsebene unterschieden werden. So ist es möglich einen groben Überblick über das Gesamtsystem zu geben. Ein Klassendiagramm kann aber auch eine detaillierte Darstellung der Klassen im System geben und dadurch als Implementierungsgrundlage dienen. Das Klassendiagramm soll die Frage „*Wie sind die Daten und das Verhalten meines Systems im Detail strukturiert?*“ [Rupp05] beantworten.

- *Paketdiagramm*: Das Paketdiagramm dient zur strukturellen Darstellung des Systems. Auch bei diesem Diagramm können verschiedene Abstraktionsebenen unterschieden werden. Mit Hilfe dieses Diagramms soll die Komplexität des Systems reduziert werden, indem es in mehrere Pakete aufgeteilt wird [Hitz05].

- *Objektdiagramm*: Das Objektdiagramm ist dem Klassendiagramm sehr ähnlich und unterscheidet sich dadurch, dass in diesem Diagramm nicht die Klassen, sondern Objekte (Instanzen von Klassen) modelliert werden. Das Objektdiagramm bietet einen Ausschnitt eines prototypischen bzw. exemplarischen Systems [Hitz05].

- *Kompositionsstrukturdiagramm*: Das Kompositionsstrukturdiagramm wurde für die UML Version 2.0 neu eingeführt. Es bietet die Möglichkeit die interne Struktur einer Klasse und Beziehungen einer Klasse zu anderen Teilen des Systems darzustellen [Rupp05].

- *Komponentendiagramm*: Das Komponentendiagramm bietet die Möglichkeit Komponenten und deren Zusammenhänge zu definieren. Eine Komponente kann dabei sowohl eine Klasse, aber auch eine systemspezifische Konfigurationsdatei sein. Es soll die Frage „*Wie ist mein System strukturiert und wie werden diese Strukturen erzeugt?*“ [Rupp05] beantworten.

- *Verteilungsdiagramm*: Mit dem Verteilungsdiagramm kann die eingesetzte Hardwaretopologie und das zugeordnete Laufzeitsystem, sowie eine Verteilung der Komponenten modelliert werden [Hitz05].

2.2 Designpattern

Das folgende Kapitel soll die prinzipiellen Designpattern einer Enterprise Anwendung erläutern. Ein Pattern (Entwurfsmuster) ist die „best practice“ zum Lösen eines bestimmten, wiederkehrenden Problems. Das heißt ein Pattern beschreibt ein wiederkehrendes Problem und stellt eine Möglichkeit für eine Lösung dar [Mari02].

Bekannt in diesem Bereich wurde die so genannte „Gang of Four“ (GoF), die in ihrem Buch [Gamm95] 23 verschiedene Patterns vorstellt. Eines der bekanntesten Entwurfspatterns ist das so genannte Model-View-Controller (MVC) Pattern, welches eine Trennung zwischen Domainobjekten (Model), der Anzeige (View) und der Businesslogik (Controller) vorsieht.

Weitere Designpattern, vor allem Pattern in der Softwarearchitektur, werden in [Busc96] beschrieben.

2.2.1 Grundlegende Architektur einer Enterprise Anwendung

Die grundsätzliche Architektur einer Enterprise Anwendung ist die so genannte *4-Tier-Architektur* (4-Schichten-Architektur). Laut [Lang06] gilt die 4-Tier-Architektur seit dem Internet Boom Mitte der 90er Jahre als „*State-of-the-art-Architektur*“ für webbasierte Anwendungen.

Die vier Schichten (Ebenen) werden wie folgt eingeteilt [Lang06]:

- *Visualisierungsebene (Web-Client)*: Diese Ebene nimmt Eingaben des Benutzers entgegen und gibt diese an die nächste Schicht weiter. Nach der Verarbeitung werden Daten von der nächsten Schicht entgegen genommen und für den Benutzer gerecht dargestellt.
- *Präsentationsebene (Web-Server)*: Diese Ebene verarbeitet und prüft die vom Benutzer eingegebenen Daten und schickt sie an die dritte Schicht weiter. In die andere Richtung werden die Daten von der nächsten Ebene empfangen und für die Visualisierungsebene vorbereitet.

- *Geschäftslogik (Anwendungsserver)*: Die Geschäftslogik stellt das Herzstück der Enterprise Anwendung dar. In dieser Ebene werden die Daten verarbeitet und alle Prozesse, die die eigentliche Anwendung betreffen, implementiert. Zu speichernde Daten werden an die nächste Schicht weitergegeben. Benötigte Daten von der vierten Ebene gelesen.
- *Datenbankebene (Datenbankserver)*: In dieser Ebene werden die Daten, die in der Geschäftlogik entstehen, gespeichert, bzw. Daten, die von der Geschäftslogik benötigt werden, geholt.

Folgende Abbildung zeigt eine graphische Darstellung der 4-Tier-Architektur:

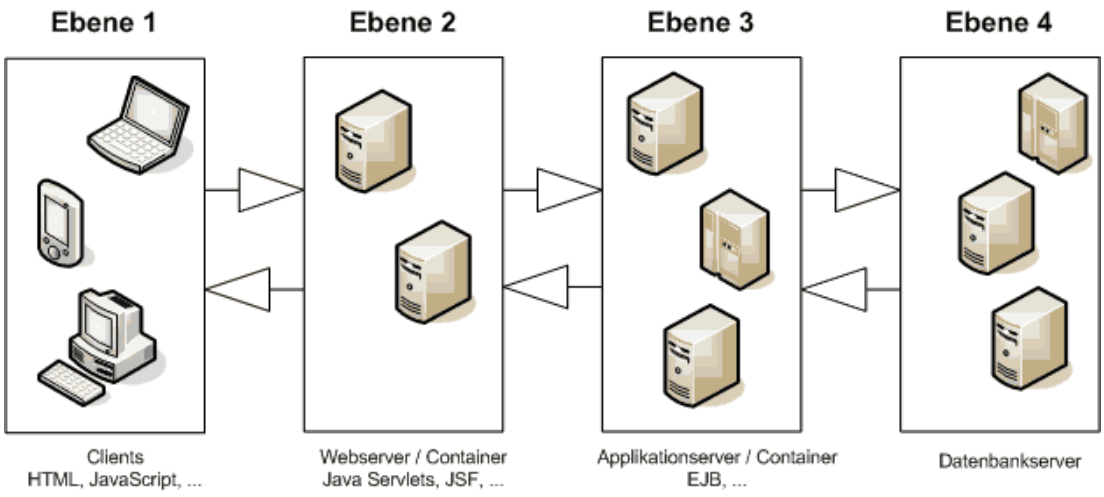

Abbildung 2: Übersicht 4-Tier Architektur (basiert auf [Lang06])

Der Vorteil einer mehrstufigen Schichten Architektur liegt vor allen an der Skalierbarkeit und der leichten Wartung der Anwendung. So kann jede Schicht für sich auf einen eigenen physischen Rechner laufen und auch einzeln gewartet werden [Schä02].

Auch im Zusammenhang mit dem Entwicklungsprozess stellen mehrschichtige Architekturen einen Vorteil dar. Entwickler können unabhängig von anderen Schichten ihren Teil implementieren.

2.2.2 Weitere Designpattern einer Enterprise Anwendung

Das folgende Kapitel soll weitere Designpattern vorstellen, die in dieser Arbeit verwendet werden. Die Firma Sun Microsystems stellt eine Liste von den meist verwendeten Patterns in Java Enterprise Anwendungen zur Verfügung [Sun07].

„Data Access Object“ (DAO)-Pattern

Das DAO-Pattern kapselt den Datenzugriff vom aufrufenden Objekt. Das „Data Access Object“ Interface enthält Zugriffs- bzw. Schreibemethoden für den Datenbestand. Die Implementierung dieses Interfaces kann auf verschiedene Arten vorhanden sein. Der Zugriff auf die Daten kann z.B. mittels SQL Abfragen implementiert werden. Wird in einer späteren Phase im Projekt entschieden, als Datenquelle XML zu verwenden, so muss nur die Implementierung des Zugriffobjektes und die Konfiguration ausgetauscht werden. Die, meist komplexe, Business Logik bleibt hingegen unverändert. In der Praxis wird häufig ein Austausch der Implementierung für das Testen der Businessmethoden vorgenommen.

Intercepting Filter-Pattern

Das Intercepting Filter-Pattern wird angewendet, um die Anfrage („request“) und die Antwort („response“) von/an einem Web Client vor/nach der eigentlichen Verarbeitung zu modifizieren [Sun07].

In der Praxis wird das Intercepting Filter-Pattern für die Authentifizierung bzw. die Autorisierung verwendet. Weitere Einsatzmöglichkeiten sind unter anderem Logging, Tracking, Performanzmessungen, Datenkomprimierungen oder Verschlüsselungen [Wang03].

2.3 (Java) Enterprise Anwendungen

Als Definition von Java Enterprise Anwendungen, kann man sagen, dass es sich um Anwendungen handelt, die mit der Java Enterprise Edition (Java EE) implementiert sind. Java EE könnte man in einem Satz als das *„am weitesten verbreitete „Betriebssystem“ für Webanwendungen“* [Star05] beschreiben.

2.3.1 Java EE - Java Enterprise Edition

Bei der Suche nach einer einfachen und kurzen Definition von der Java Enterprise Edition (Java EE) wird man in der freien Internet-Enzyklopädie Wikipedia fündig:

Java Plattform, Enterprise Edition, abgekürzt Java EE oder früher J2EE, ist die Spezifikation einer Softwarearchitektur für die transaktionsbasierte Ausführung von in Java programmierten Anwendungen.

Es handelt sich also um eine Spezifikation einer Softwarearchitektur. Die offizielle, standardisierte Beschreibung der Java Enterprise Edition [Jend06] von der Firma Sun umfasst in etwa 1.300 Seiten.

Einen kurzen Einblick über Java EE Architektur soll die Einteilung der Kapitel zeigen [Jend06]:

- *Web-tier Technologien*: Dieser Teilbereich von Java EE umfasst Standards wie z.B. Java Servlets, Java Server Pages oder Java Server Faces.
- *Web Service Technologien*: In diesem Teil werden alle Modelle beschrieben, die sich mit Web Services befassen.
- *Enterprise Java Beans (EJB)*: Dieser Teil beschreibt, wie man die Business Logik einer Java Enterprise Anwendung entwickelt und umfasst Basistechnologien, wie *Session beans* und *Message-driven beans*.
- *Persistenz Technologien*: Diese Kapiteln beschäftigen sich mit dem Datenbankzugriff von einer Java Enterprise Anwendung.
- *Plattform Services*: Das letzte Kapitel beschäftigt sich mit Themen, die von vielen anderen Komponenten verwendet werden. Hier werden unter anderem die grundlegenden Technologien wie das Transaktionshandling oder die Sicherheit einer Java Enterprise Anwendung beschrieben.

Java EE Anwendungen sind meist webbasiert und werden vom Benutzer über einen Webbrowser aufgerufen. Der große Vorteil in dieser Architektur liegt darin, dass

jede Schicht plattformunabhängig bleibt.

Zur Ausführung einer Java Enterprise Anwendung wird ein so genannter Anwendungsserver verwendet. Die gängigsten sind das Open Source Projekt JBoss[2], sowie die kommerziell vertriebenen Anwendungsserver von Bea (Bea Weblogic[3]) und IBM (IBM Websphere[4]). Es existieren allerdings auch weitere Produkte am Markt.

Motivation und Ziele

Java Enterprise Anwendung stellen eine Vereinfachung für die Entwicklung von Client-Server Anwendungen dar. Folgende Ziele werden dabei verfolgt [Schä02]:

- *Wiederverwendbarkeit*: Die Implementierten fachlichen Komponenten sind unabhängig von technischen Konzepten (*Business Objekte*).
- *Standardisierte Schnittstellen*: Standardisierte Schnittstellen sollen die Unabhängigkeit zu anderen Systemen ermöglichen, die Enterprise Anwendung wird dadurch herstellerunabhängig.
- *Trennung zwischen fachlichen und technischen Aspekten*: Jeder Entwickler hat eine spezielle Aufgabe und ist durch die standardisierten Schnittstellen unabhängig(er) von anderen Entwicklern.

Vor- und Nachteile

Der größte Vorteil an der Java Enterprise Edition bzw. an Java allgemein ist die Plattformunabhängigkeit und die laufende Weiterentwicklung und Anpassung der Programmiersprache.

In der heutigen Zeit haben modern entwickelte webbasierten Systeme hohe Anforde-

---

[2] http://www.jboss.org

[3] http://www.bea.com

[4] http://www.ibm.com/websphere

rungen, wie Skalierbarkeit und Wartbarkeit. Dadurch bietet sich eine mehrstufige Systemarchitektur an, um Benutzersicht, Geschäftslogik und Datenbank zu trennen [Schä02]. Mit Hilfe der Java Enterprise Edition wird genau diese Trennung innerhalb einer Anwendung ermöglicht.

Einer der größten Nachteile von Java Enterprise Anwendungen ist, dass es zwar einen Standard gibt, in der Praxis dieser Standard allerdings (leider) von jedem Anwendungsserver im Detail etwas anders implementiert wird [Schä02].

Ein weiterer Nachteil ist die Performanz einer Java Enterprise Anwendung. In vielen Fällen erfolgt bei jedem Aufruf ein Zugriff auf eine entfernte („*remote*“) Komponente. Dadurch müssen alle Objekte die verschickt werden zuerst serialisiert werden, um danach beim Empfänger wieder zu einem vollständigen Objekt zusammengebaut zu werden. Dieser Vorgang benötigt einiges an Zeit und Rechenleistung [Schä02].

2.3.2 Das Java Framework Spring

Das folgende Kapitel soll das Java Framework Spring näher beschreiben. Als Framework versteht man eine Ansammlung von Klassen bzw. Paketen, die wiederkehrende Probleme in einem Projekt lösen sollen.

Kurz gesagt bietet Spring-Framework die Möglichkeit, Java Enterprise Anwendungen zu entwickeln. Das Framework soll unterstützend im Implementierungsprozess dienen und dem Programmierer Arbeiten abnehmen. Das Spring Framework wird von einer großer Community unterstützt und weiterentwickelt. Die Anzahl der Downloads übersteigt eine Million. Deshalb wurde das Spring Framework de facto zu einem Standard [John07].

Das Spring Framework, dessen Ideen ursprünglich aus dem Buch „Expert One-on-One: J2EE Design und Development“ von Rod Johnson [John03] entstanden sind, wird als Open Source Framework realisiert.

Das Framework Spring wird für serverbasierte Enterprise Anwendungen eingesetzt. Ziel von Spring ist es, dem Entwickler eine Möglichkeit zu bieten, unabhängig von der eingesetzten Technologie seine Geschäftslogik, also das Herzstück des Projekts, zu implementieren. Dabei werden so genannte POJO’s (Plain Old Java Objects) und POJI’s (Plain Old Java Interfaces) verwendet. Unter einem POJO wird eine Java Klasse verstanden, die nur Methoden, die sich mit der Geschäftlogik beschäftigen,

beinhalten und keine systemabhängige oder plattformabhängige Methoden.

Als weiteres Werkzeug stellt Spring XML-Konfigurationsdateien zur Verfügung, die den Zusammenhang zwischen Klassen beschreiben.

Durch die Tatsache, dass Spring, von der Idee her, vollkommen unabhängig von Java Enterprise Anwendungen ist, wird es ermöglicht einzelne Teile der Software, unabhängig vom Container (Anwendung Server) zu testen [Oate07].

Ziele von Spring

Die Hauptziele von Spring lassen sich wie folgt definieren [John07]:

- Es soll die Entwicklung eines unabhängigen Programmiermodels ermöglicht werden. Der implementierte Quellcode soll so weit wie möglich von der verwendeten Architektur getrennt sein.
- Durch diese angestrebte Lösung wird es Entwicklern ermöglicht, sich auf die Kernprobleme des Projektes (Businesslogik) zu konzentrieren. Allgemeine Probleme im Projekt (wie die z.B. die Verteilung der Komponenten oder die Authentifizierung) werden vom Spring Framework implementiert.
- Das Spring Framework soll die Entwicklung von Enterprise Anwendungen einfach halten, allerdings nicht auf Kosten der Performanz.

Das eigentliche Ziel von Spring ist es also die Java Enterprise Edition zu vereinfachen und einen guten Programmierstil zu fördern [John05].

Architektur und Philosophie des Frameworks Spring

Das Spring Framework verwendet mehrere bereits bestehende Frameworks (z.B. für Logging oder Connection pools). Ein Grundsatz von des Frameworks Spring ist es, offene Schnittstellen zu definieren und dadurch eine leichte Einbindung von anderen Technologien zu ermöglichen [John05].

Die beiden Hauptarchitekturteile des String Frameworks sind „Inversion of Control“ und „Dependency Injection“. Der Grundsatz von „Inversion of Control“ ist das Prinzip „Don’t call me, I’ll call you“. Dass heißt, das Framework ruft die implementier-

ten Klassen auf, und nicht umgekehrt [John05].

Der Grundsatz von „Dependency Injection“ ist es, Objekte unabhängig von ihrer Umgebung zu machen. Im Spring Framework funktioniert dies über die so genannten „Setter-Methoden“, die es erlauben bestimmte Abhängigkeiten bei Laufzeit (über eine XML Konfigurationsdatei) zu bestimmen. Eine alternative Möglichkeit wäre es, beim Anlegen des Objekts die Abhängigkeiten mitzugeben [John05]. Durch diese Eigenschaft können Abhängigkeiten von Objekten geändert werden, ohne den Quellcode zu verändern.

2.3.2 Weitere Java Frameworks

Das folgende Kapitel soll einen Überblick über weitere Java Frameworks geben, die in der vorliegenden Arbeit zu einem späteren Zeitpunkt verwendet werden.

Das Framework „Spring Webflow“

Das Spring Webflow Framework ist ein Teil des oben beschriebenen Spring Frameworks und ein Java Web Framework der nächsten Generation. Dieses Framework erlaubt es Entwicklern, Aktionen des Benutzers in Form von getrennten Modulen zu implementieren [Dona07].

Durch die Aufteilung der Benutzerschnittstellen in Modulen ist eine hohe Wartbarkeit und Wiederverwendbarkeit gegeben. Die eigentliche Controllerlogik wird an einer Stelle implementiert, die öfters wieder verwendet werden kann. Die Definition des gesamten Ablaufes wird mit Hilfe einer XML-Konfigurationsdatei dargestellt.

Das „Spring Webflow“ Framework stellt kein selbständiges Framework dar, sondern soll als „Abflussmanager“ für andere Frameworks dienen [Dona07].

Das Framework „Struts“

Beim Framework „Struts“ handelt es sich um ein Open Source Projekt, welches die Entwicklung von Webprojekten unterstützt. Die Grundlagen dieses Frameworks basieren unter anderem auf bekannte Technologien wie Java Servlets, Java Server Pages oder XML [Haig05].

Als Grundlage dient das MVC-Pattern. Die Modelelemente dienen als Datenhalter und erlauben im Framework „Struts“ eine einfache Validierung der Benutzereingabe.

Für die Präsentation („View“) werden Java Server Pages (JSP) verwendet. Als Controller werden so genannte „Actions“ eingeführt, die eine Schnittstelle zwischen der Präsentationsschicht und der Modelelemente darstellen.

Das Herzstück einer Struts-Anwendung ist eine XML-Konfigurationsdatei (struts-config.xml), welche die Komponenten miteinander verbindet und den Workflow der Anwendung definiert [Star05].

Das Framework „Hibernate“

Beinahe hinter jeder Enterprise Anwendung steht eine relationale Datenbank, welche der Speicherung von Daten dient.

Das Framework „Hibernate“ ist eines der bekanntesten Open Source Frameworks zur Implementierung eines Objekt/Relationalen-Mappings (OR-Mapping). Unter einem Objekt-relationalen Mapping versteht man das Erzeugen von Objekten aus einer relationalen Datenbank und das Speichern von Objekten in eine relationale Datenbank.

Das Framework „Hibernate“ verwendet dazu „Hibernate Mapping Files“, welche die Darstellung der Objekte in der relationalen Datenbank beschreiben. Für Datenbankabfragen wurde in diesem Framework eine eigene Abfragesprache (HQL) entwickelt, welche der bekannten Abfragensprache SQL ähnelt. Die Verwendung des Frameworks „Hibernate“ ermöglicht das Austauschen einer Datenbank, ohne die bestehende Anwendung zu verändern. Das Framework wird von vielen gängigen relationalen Datenbanken unterstützt [Oate07].

2.4 Begleitendes Beispiel

Die vorliegende Diplomarbeit soll von einem Beispiel begleitet werden, welches auf Basis von Spring [Demo06] implementiert wurde. Für diese Beispielanwendung soll untersucht werden, welches Diagramm von UML 2.0 in welcher Abstraktionsebene passend ist.

Das Beispiel soll einen einfachen Bestellvorgang und eine einfache Administration eines Shopsystems simulieren [Demo06].

Es kann zwischen zwei Gruppen von Benutzern unterschieden werden:

- *Administratoren*: Ein Administrator kann Kunden und Produkte verwalten

(hinzufügen, ändern, löschen) sowie Bestellungen administrieren.

- *Kunde*: Registrierte Kunden können Bestellungen durchführen.

Der folgende Screenshot (Abbildung 3) zeigt die Kundenübersicht für den Administrator. Der Anwender hat die Möglichkeit, durch klick auf die ID des Kunden, diesen zu ändern. Durch Klick auf „Delete“ wird der Kunde aus der Datenbank gelöscht. Nach Klick auf den Button „New Customer“ wird die Eingabemaske zum anlegen eines neuen Kunden angezeigt.

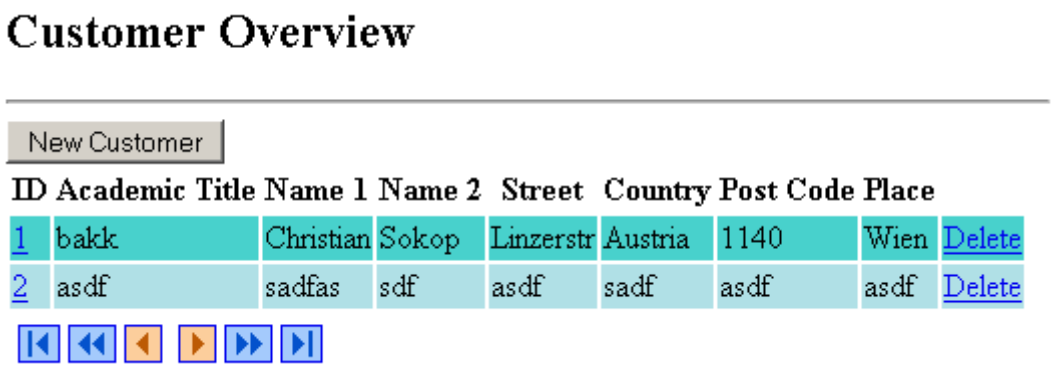

Abbildung 3: Kundenübersicht [Demo06]

Die Kundenverwaltung wurde in diesem Beispiel mit Hilfe von zwei verschiedenen Frameworks implementiert. Die eine Version verwendet das Framework „Struts“, welche in weiterer Folge mit Hilfe von UML 2.0 Diagrammen näher beschrieben wird. Die zweite Version wurde mit Hilfe von JSF (Java Server Faces) implementiert. Diese Version findet in dieser Arbeit keine weitere Betrachtung.

Der zweite Teil der Anwendung erlaubt es Kunden, Bestellungen durchzuführen.

Bereits registrierte Kunden können Bestellungen durchführen. Der Bestellvorgang besteht im wesentlichem aus folgenden Schritten:

- *Kunden anmelden*: Durch die Eingabe von Benutzername und Passwort wird der Kunde am System angemeldet.

- *Produkte wählen*: Der Kunde hat die Möglichkeit aus einer Liste von Produkten zu wählen (siehe Abbildung 4).
- *Kundendaten ändern*: Dieser optionale Schritt wird angezeigt, nachdem der Kunde seine Produkte ausgewählt hat.
- *Bestellung abschicken*: Nachdem der Kunde seine Daten bestätigt hat, wird die Bestellung abgeschickt.

Folgende Abbildung zeigt die Produktliste. Auf dieser Seite kann der Benutzer seine Produkte auswählen und in den Einkaufswagen hinzufügen.

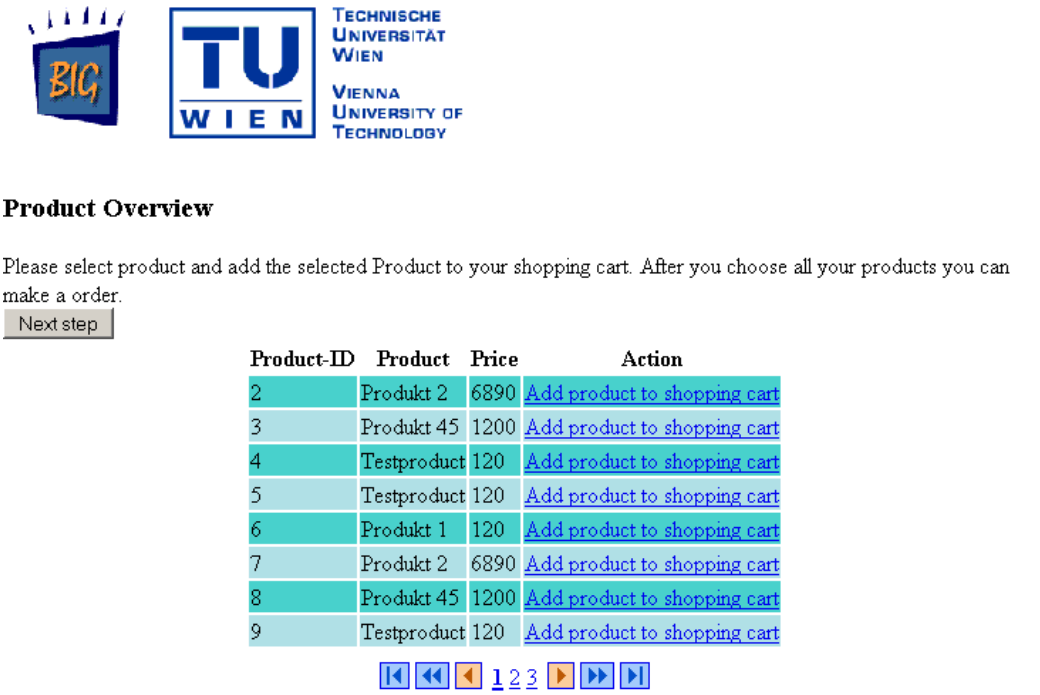

Abbildung 4: Liste der Produkte im Bestellvorgang [Demo06]

Der detaillierte Aufbau der Anwendung wird in den nächsten Kapiteln in Form von UML 2.0 Diagrammen beschreiben.

Kapitel 3

UML 2 - Verhaltensmodellierung

Bei der Verhaltensmodellierung stehen die dynamischen Aspekte des Systems im Vordergrund [Hitz05]. Verhaltensmodelle können in jeder Phase des Projektes zum Einsatz kommen.

3.1 Anwendungsfalldiagramm

Das Anwendungsfalldiagramm abstrahiert das zu entwickelte System und stellt es in Form vom Anwendungsfällen aus Sicht des Benutzers dar [Hitz05]. Mit einem Anwendungsfalldiagramm soll die Frage „*Was soll mein geplantes System eigentlich leisten*“ [Rupp05] beantwortet werden. In älteren Versionen von UML wurde das Anwendungsfalldiagramm als Strukturdiagramm geführt und nicht als Verhaltensdiagramm.

3.1.1 Anwendungsfalldiagramm und Enterprise Anwendungen

Beim Anwendungsfalldiagramm soll die Benutzer spezifische Ebene modelliert werden. Die in der Einleitung beschriebene 4-Schichten Architektur findet dabei keine Bedeutung. Das Anwendungsfalldiagramm ist eine sehr abstrakte, wenig detaillierte Sicht auf eine Enterprise Anwendung.

Ein Anwendungsfalldiagramm ist ein guter Einstieg, um einen Überblick über das gesamte System zu geben, und wird deshalb sehr früh, wenn nicht sogar als erstes, in einem Projekt modelliert. Das Anwendungsfalldiagramm wird in der Praxis öfters in Absprache mit dem Kunden erstellt.

In einem Anwendungsfalldiagramm werden prinzipiell zwischen Aktoren und Anwendungsfälle unterschieden. Aktoren interagieren mit dem System, sind aber klar außerhalb des Systems angesiedelt [Hitz05]. Dabei kann es sich um Benutzer, aber auch um andere Systeme handeln. Ein Anwendungsfall ist eine abgeschlossene, zusammenhängende Einheit, welche einen Teil der Funktionalität des Systems repräsentiert [Zuse01].

3.1.2 Anwendungsfalldiagramm im begleitenden Beispiel

Wie schon beschrieben stellt ein Anwendungsfalldiagramm eine graphische Übersicht über das System und seine Aktoren dar. Im begleitenden Beispiel sind Kunden und Administratoren die Aktoren der Enterprise Anwendung.

Das Suchen von Anwendungsfällen geht meistens ein genaues studieren einer textuellen Beschreibung voraus. Verben in den Sätzen sind dabei mögliche Tätigkeiten und Ansätze für einen Anwendungsfall [Zuse01]. In unserem Beispiel sind die ersten Kandidaten „Produkte verwalten“, „Kunden verwalten“, „Bestellungen verwalten“ und „Bestellung durchführen“.

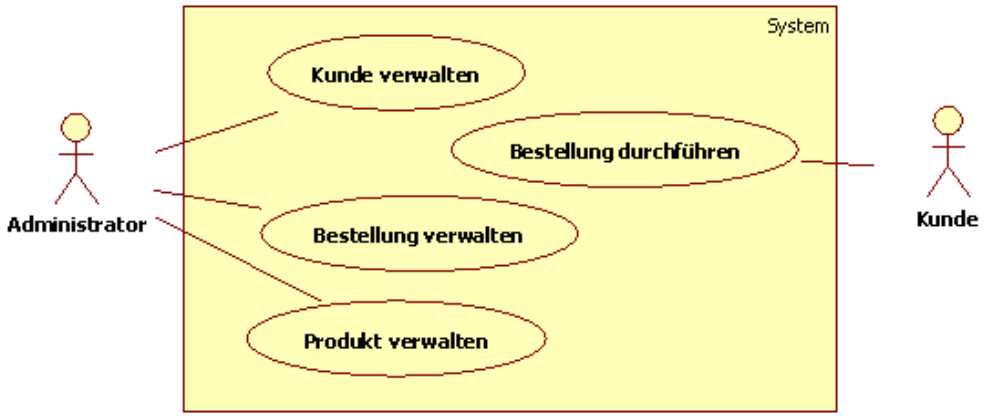

Abbildung 5: Anwendungsfalldiagramm Übersicht ([Demo06])

Ein Anwendungsfalldiagramm wird meist in mehreren Iterationen entworfen. Das erste Diagramm gibt eine grobe Übersicht über das Gesamtsystem. In einem weiteren Schritt werden die Anwendungsfälle „Kunde verwalten“ und „Bestellung durchführen“ verfeinert.

Seit UML 2.0 können Anwendungsfälle in Pakete gegliedert werden [Rupp05]. Für

die Modellierung bedeutet das, dass schon auf dieser sehr abstrakten Ebene eine Einteilung in Pakete erfolgen kann. In Bezug auf eine Enterprise Anwendung könnten die Pakete, die zu implementierenden Services sein. In Fall des Beispieles wären z.B. ein Kundenservice und ein Shop Service zu implementieren.

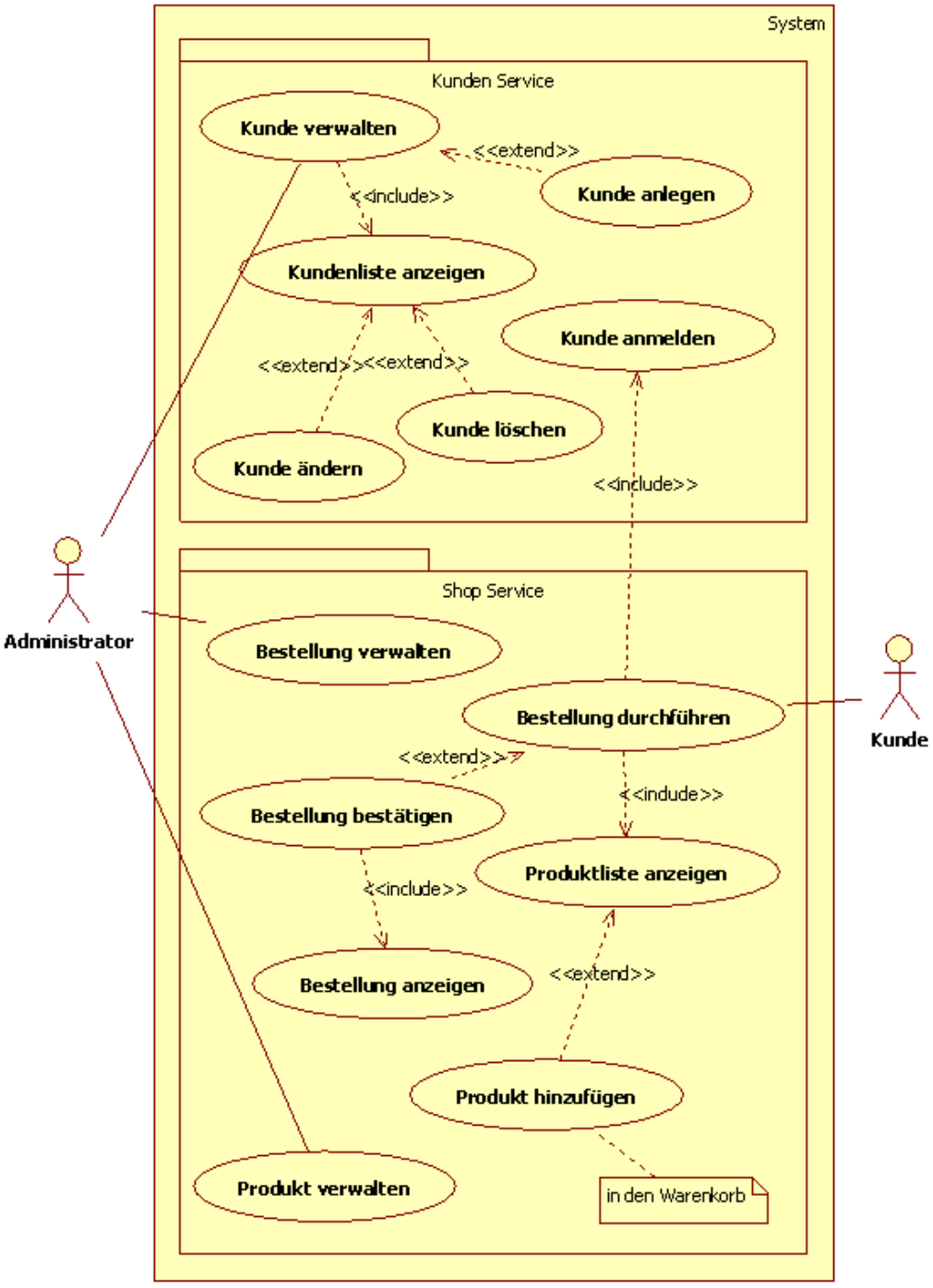

Abbildung 6: Anwendungsfalldiagramm detailliert

In vielen Fällen, und auch von der Literatur empfohlen ([Zuse01], [Hitz05], …), ist es sinnvoll eine so genannte Anwendungsfallbeschreibung durchzuführen. Diese Beschreibung soll eine detaillierte Übersicht über den Anwendungsfall geben, und enthält unter anderem eine Kurzbeschreibung, Vorbedingungen, Beschreibung des Ablaufs und Auswirkungen [Zuse01]. In UML wird keine Anwendungsfallbeschreibung vorgeschrieben, deshalb wird sie auch hier nicht durchgeführt. Anwendungsfälle können in UML 2.0 mit anderen Diagrammen beschreiben werden. Diese Möglichkeiten werden in den folgenden Kapiteln gezeigt.

3.2 Aktivitätsdiagramm

Aktivitätsdiagramme sind mit der Version 2.0 von UML komplett überarbeitet worden. Sie spezifizieren den Kontroll- und Datenfluss zwischen verschiedenen Arbeitsschritten [Hitz05]. In früheren Versionen stellten Aktivitätsdiagramme eine spezielle Form von Zustandsdiagrammen dar. Mit der Version 2.0 von UML wurden neue Elemente, wie z.B. Strukturierte Knoten, Entscheidungsknoten oder Schleifenknoten eingeführt [Rupp05].

Aktivitätsdiagramme können in verschiedenen Abstraktionsebenen modelliert werden. Das ist auch der Grund, warum sie in vielen Bereichen im Projekt einsetzt werden können. Die drei wichtigsten Einsatzbereiche sind die Geschäftsprozessmodellierung, die Beschreibung von Anwendungsfällen und die Implementierung einer Operation [Rupp05].

Damit bietet das Aktivitätsdiagramm jegliche Form der Abstraktion an und reicht in seiner Vielfalt von der Modellierung von Geschäftprozessmodellen, welche eine sehr abstrakte Sicht auf ein Unternehmen gibt, bis hin zur Modellierung eines Algorithmus, welcher einen kleinen Teil in einem Programm darstellt.

Gegliedert nach diesen drei Anwendungsbereichen sollen die folgenden Kapiteln Aktivitätsdiagramme in Bezug auf Enterprise Anwendungen beschreiben. Die Modellierung einer Operation wird im nächsten Unterkapitel anhand eines Beispiels gezeigt. Die Beschreibung von Anwendungsfällen wird anhand des begleiteten Beispiels gezeigt.

3.2.1 Aktivitätsdiagramme und Enterprise Anwendungen

Aktivitätsdiagramme sind für allgemeine Architekturbeschreibungen nicht geeignet. Ein Einsatzgebiet von Aktivitätsdiagrammen ist die Modellierung von Operationen, wie das nächste Kapitel anhand eines Beispiels demonstrieren soll.

Implementierung einer Operation

Eine Operation ist „*der kleinste Verarbeitungsschritt einer Transaktion in einem Transaktionssystem zur Veränderung des Datenbestandes*“ [Wiki07]. Ein Aktivitätsdiagramm bietet die Möglichkeit Algorithmen bzw. Operationen zu modellieren [Rupp05].

Im begleitenden Beispiel wird kein komplexer Algorithmus implementiert, da in Java sehr viele gängige Algorithmen schon ausprogrammiert sind. Deshalb wird hier ein anderes kleines Beispiel vorgestellt und als Aktivitätsdiagramm modelliert. Dabei soll gezeigt werden, wie das Modellieren eines Algorithmus mit Aktivitätsdiagrammen möglich ist.

Als Beispiel wurde hier der Sortieralgorithmus „Bubble Sort“ gewählt. Dieser Sortieralgorithmus durchläuft ein Array bis zum Ende, vergleicht zwei Zahlen, und tauscht sie, wenn die vordere Zahl größer ist. Dieser Vorgang wird so lange wiederholt, bis das Array sortiert ist.

Der Quellcode des Algorithmus sieht in Java wie folgt aus [Möss05].

```
// Sort a (using bubble sort)
static void sort(int[] a) {
   for (int i = a.length - 1; i > 0; i--) {
      for (int j = 0; j < i; j++) {
         if (a[j] > a[j+1]) {
            int h = a[j]; a[j] = a[j + 1]; a[j + 1] = h;
         }
      }
   }
}
```

Listing 1: Quelltext „Bubble Sort“

Sehr hilfreich bei der Modellierung des Aktivitätsdiagramms erwies sich, für Kenner des Nassi-Shneiderman-Struktogramms[5], ein Vergleich zwischen Shneiderman-Diagrammen und Aktivitätsdiagrammen [Soph05].

StarUML bietet die in Abbildung 7 gezeigte Notation nicht an [Reic06].

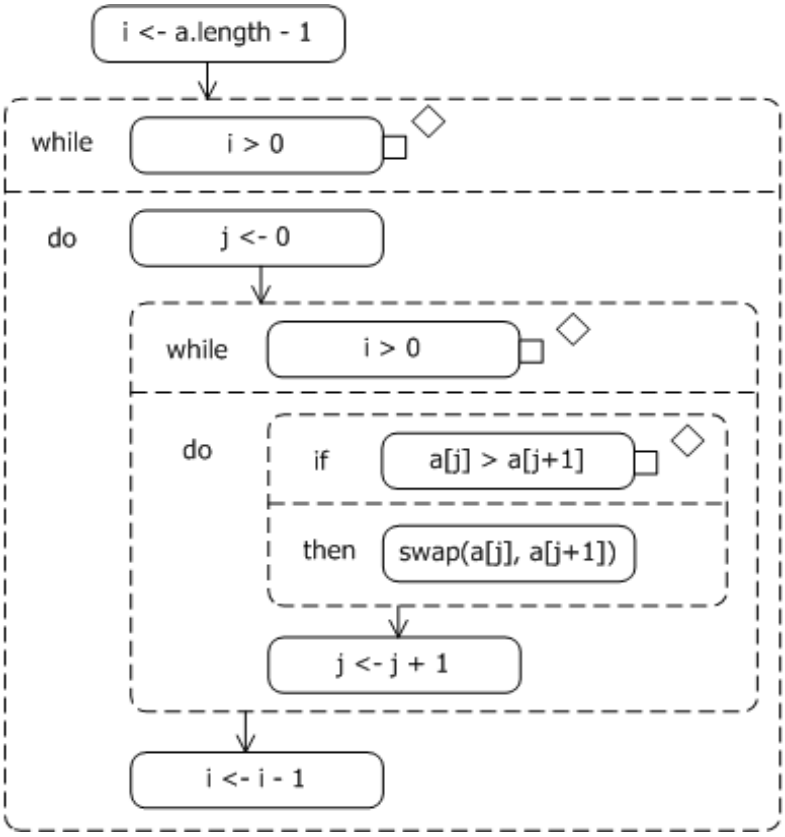

Abbildung 7: Aktivitätsdiagramm Bubble Sort

Das Aktivitätsdiagramm für dieses einfache Beispiel ist noch einigermaßen übersichtlich, allerdings sagt UML 2.0, dass die Verwendung von anderen Modellierungstools durchaus in Betracht gezogen werden soll [Rupp05], sobald komplexere Problemstellungen modelliert werden sollen.

Auch das Aktivitätsdiagramm des relativ einfachen Sortieralgorithmus streckt sich schon über eine halbe Seite. Bei komplexeren Algorithmen sind Pseudocode oder andere Methodiken vorzuziehen, die nicht so platzintensiv sind.

[5] http://de.wikipedia.org/wiki/Nassi-Shneiderman-Diagramm

Meiner Meinung nach wurde in UML 2.0 diese Möglichkeit geschaffen, um auch Algorithmen darzustellen zu können, was mit älteren Versionen von UML nicht möglich war.

3.2.2 Aktivitätsdiagramme im begleitenden Beispiel

Im folgenden Kapitel werden die Einsatzmöglichkeiten der Geschäftsprozessmodellierung und der Anwendungsfallbeschreibung im begleitenden Beispiel mit Hilfe des Aktivitätsdiagramms gezeigt.

Geschäftsprozessmodellierung

Die Modellierung eines Geschäftsprozesses macht in Bezug auf dieses Beispiel keinen Sinn, da im begleitenden Beispiel kein Geschäftsprozess realisiert wird. Bei Anwendungen, die den Businessprozess unterstützen sind Aktivitätsdiagramme in dieser Abstraktionsebene durchaus sinnvoll und können gerade in der Anfangsphase eines Projekts eine gute Übersicht geben.

Anwendungsfallbeschreibung

Im folgenden Kapitel soll der Anwendungsfall „Bestellung durchführen“ aus dem begleiten Beispiel modelliert werden.

Dieses Beispiel gibt einen guten Überblick über die Einsatzmöglichkeit von Aktivitätsdiagrammen in Enterprise Anwendungen. Der Anwendungsfall wird in Bezug auf Benutzer- und Systemaktivitäten dargestellt.

Als zweites Bespiel wurde versucht der Anwendungsfall „Kunde verwalten“ zu modellieren. Dabei konnte kein brauchbares Ergebnis erzielt werden. Bei der Modellierung dieses Anwendungsfalls war das Hauptproblem das Finden von Aktivitäten und das bestimmen von deren Reihenfolge.

Anwendungsfälle, die mittels Aktivitätsdiagramme beschrieben werden sollen, sollten einen chronologischen Ablauf haben. In diesem Punkt sehe ich auch einen kleinen Nachteil beim Aktivitätsdiagramm. Es können nicht alle Anwendungsfälle aussagekräftig modelliert werden.

Die Modellierung von Aktivitätsdiagrammen wird von StarUML nicht ausreichend unterstützt. Deshalb stimmen einige in diesem Diagramm gezeigten Elemente nicht

zu 100% mit der UML 2.0 Notation überein [Reic06].

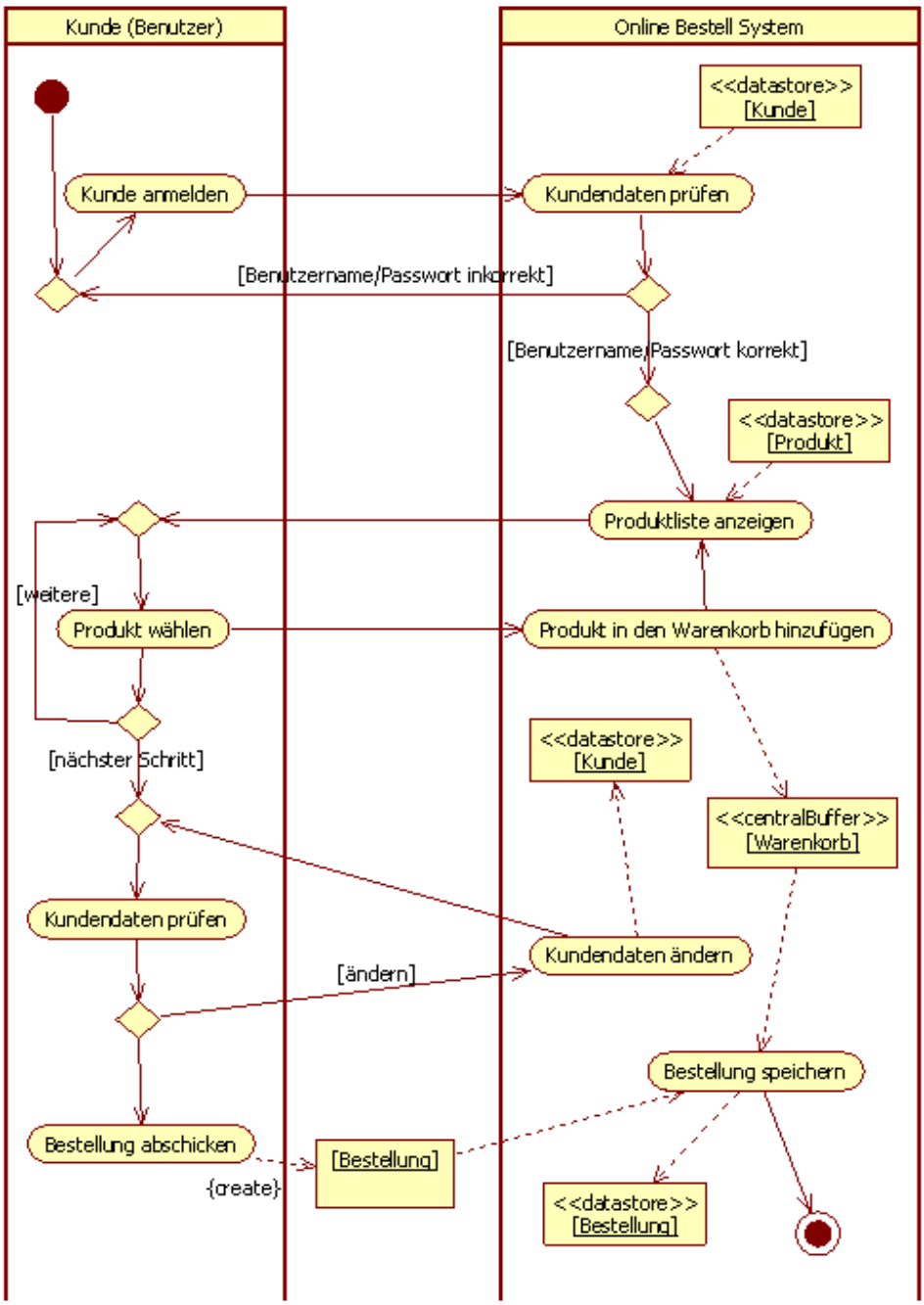

Abbildung 8: Aktivitätsdiagramm des Anwendungsfall „Bestellung durchführen“

Das Aktivitätsdiagramm eignet sich prinzipiell sehr gut für die Modellierung der Präsentationsschicht einer mehrschichtigen Architektur. Hier kann sehr gut beschrieben werden, welche Aktivitäten der Benutzer vornimmt, und welche Aktionen (Aktivitäten) das System macht, unabhängig davon, in welcher Ebene die eigentliche Lo-

gik implementiert ist.

Zusammenfassung

Zusammengefasst kann gesagt werden, dass das Aktivitätsdiagramm eine gute Möglichkeit darstellt, um komplexe Anwendungsfälle zu beschreiben. Ein komplexer Anwendungsfall durchläuft mehrere Aktivitäten und kann möglicherweise auch mehrere alternative Aktivitäten durchlaufen.

Allgemein betrachtet ist auch das Aktivitätsdiagramm sehr abstrakt und allgemein gehalten, und bietet dadurch eher einen Überblick über die Enterprise Anwendung, ohne dabei in Details zu gehen.

3.3 Zustandsdiagramm

Zustandsdiagramme sind eine weitere Möglichkeit um das Verhalten eines Systems zu modellieren. Dabei werden die Zustände, die das System einnehmen kann und Änderungen von Zuständen modelliert. Änderungen von Zuständen können durch interne oder externe Ereignisse ausgelöst werden [Rupp05].

Ein Zustandsdiagramm soll die Frage „*Wie verhält sich das System in einem bestimmten Zustand bei gewissen Ereignissen?*“ [Rupp05] beantworten. Zustandsdiagramme erlauben auch die Modellierung von parallel ablaufenden Zustandsautomaten, was vor allem bei der Modellierung von verteilten Systemen zum Einsatz kommen kann [Rupp05].

3.3.1 Zustandsdiagramme und Enterprise Anwendungen

Prinzipiell haben Zustandsautomaten mehrere Einsatzgebiete, die sich wie folgt definieren lassen [Rupp05]:

- *Anwendungsfälle und Zustandsautomaten*: Zustandsdiagramme eignen sich, neben der textuellen Beschreibung und den Aktivitätsdiagrammen (siehe voriges Kapitel), zur Beschreibung von Anwendungsfällen [Rupp05]. Der Unterschied zum Aktivitätsdiagramm, ist die Sicht auf das Gesamtsystem. Während beim Aktivitätsdiagramm das Zusammenspiel zwischen Aktoren und dem System modelliert wird, werden im Zustandsdiagramm nur Systeminterne Zustände und deren Änderungen modelliert.

- *Klassen und Zustandsautomaten*: Klassen und Attribute von Klassen können bestimmte Zustände einnehmen. Ist die Anzahl der Zustände endlich, können auch diese mit einem Zustandsdiagramm dargestellt werden (z.B. Enumerationen) [Rupp05].
- *Protokollzustandsautomaten*: Diese spezielle Art von Zustandsdiagrammen dient zur Beschreibung eines Protokolls. Unter einem Protokoll wird hier ein abgeschlossenes System (z.B. eine Klasse) verstanden. Innerhalb dieser Klasse werden mögliche Zustände und deren Änderungen beschrieben. In dieser Form des Zustandsdiagrams dürfen nur bestimmte Protokollzustände und Protokolltransitionen verwendet werden. Unter Protokollzuständen versteht man Zustände ohne Aktivitäten. Protokolltransitionen haben folgenden Aufbau: *„[Vorbedingung] Operation / [Nachbedingung]“* [Rupp05]. Im nächsten Kapitel wird ein Beispiel eines Protokollzustandsautomaten gezeigt.

Mittels Zustandsdiagrammen können unter anderem Anwendungsfälle beschrieben werden. Damit kann eine gute Übersicht über mögliche Zustände der Präsentationsschicht gegeben werden. Ein Zustand entspricht dabei einer Seite, die der Benutzer sieht. Eine Änderung des Zustands entspricht entweder einer Benutzeraktivität oder das Anzeigen einer anderen Seite. Das heißt Zustandsdiagramme sind, wie auch Aktivitätsdiagramme, in dieser Abstraktionsebene für die Modellierung der Präsentationsschicht geeignet.

Auf einer anderen Abstraktionsebene können mittels Zustandsdiagrammen die internen Zustände eines Objektes modelliert werden (vgl. Protokollzustandsautomaten). In diesem Fall bietet das Diagramm eine Übersicht über ein Objekt im gesamten System, ohne Beachtung der Schichten in der Enterprise Anwendung.

3.3.2 Zustandsdiagramme im begeleitenden Beispiel

Die Zustandsdiagramme im begleitenden Beispiel werden in die Unterkapitel Anwendungsfallbeschreibung und Protokollzustandsdiagramm eingeteilt.

Anwendungsfallbeschreibung

Im folgenden Kapitel werden die beiden Anwendungsfälle „Bestellung durchführen“ und „Kunde verwalten“ modelliert.

Der Anwendungsfall „Bestellung durchführen“ setzt sich aus den Zuständen „Log-in“, „Produktauswahl“, „Adresse prüfen“ und „Bestellung abschicken“ zusammen. Diese Zustände entsprechen den unterschiedlichen Webseiten in der Implementierung des Systems.

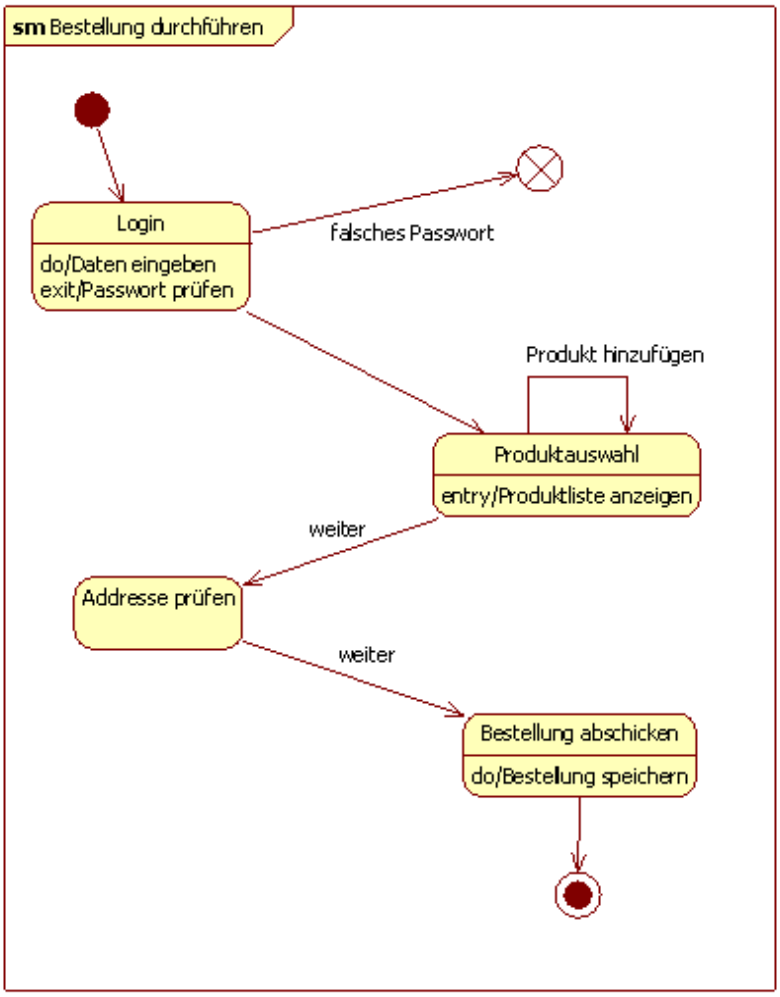

Abbildung 9: Zustandsdiagramm „Bestellung durchführen“ (basiert auf [Demo06])

Das Diagramm bietet einen guten Überblick über den Anwendungsfall „Bestellung durchführen“. Besonders beachtenswert ist ein Vergleich zwischen diesem Zustandsdiagramm und der Implementierung. In der Beispielanwendung [Demo06] wurde

dieser Teil mit Hilfe des Frameworks Spring Webflow implementiert, welche folgende XML Konfigurationsdatei verwendet (in Deutsch übersetzt von [Demo06]).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE flow PUBLIC "-//SPRING//DTD WEBFLOW 1.0//EN"
    "http://www.springframework.org/dtd/spring-webflow-1.0.dtd">

<flow start-state="Login">

  <view-state id="Login" view="/login">
    <transition on="submit" to="checkLogin"/>
  </view-state>

  <decision-state id="checkLogin">
    <action bean="loginAction"/>
    <transition on="success" to="produktauswahl"/>
    <transition on="error" to="end"/>
  </decision-state>

  <view-state id="produktauswahl" view="/produktauswahl">
    <transition on="add" to="produktauswahl"/>
    <transition on="finish" to="addresse_pruefen"/>
  </view-state>

  <view-state id="addresse_pruefen" view="/addresse_pruefen">
    <transition on="submit" to="bestellung_abschicken"/>
  </view-state>

  <view-state id="bestellung_abschicken"
      view="/bestellung_abschicken">
    <transition on="commit" to="end"/>
  </view-state>

  <end-state id="end" view="/intro"/>
</flow>
```

Listing 2: „Bestellung durchführen“ Konfigurationsdatei [Demo06]

In diesem Beispiel kann man gut den Zusammenhang zwischen dem modellierten Zustandsdiagramm und der Implementierung sehen. In dieser XML Konfigurationsdatei werden die einzelnen Zustände mit Hilfe der Elemente „*view-state*“, „*decision-state*“ und „*end-state*“ definiert. Die gleichen Zustände finden sich im modellierten Zustandsdiagramm wieder. Änderungen von Zuständen werden in der Konfigurationsdatei mit dem Element „transition“ definiert, die wie im Zustandsdiagramm auch, bei einem Zustand beginnen, und in einem anderen Enden.

Zusammengefasst kann gesagt werden, dass die XML Konfigurationsdatei des Frameworks „Spring Webflow“ eine andere Darstellungsform eines Zustandsdiagramms ist. Auch im Hinblick auf eine modellgetriebene Entwicklung stellen Zustandsdia-

gramme eine gute Möglichkeit dar, um die Präsentationsschicht einer Enterprise Anwendung zu modellieren.

Aber auch einfachere Anwendungsfällen lassen sich mit einem Zustandsdiagramm gut modellieren. Hier wird der Anwendungsfall „Kunde verwalten“ modelliert.

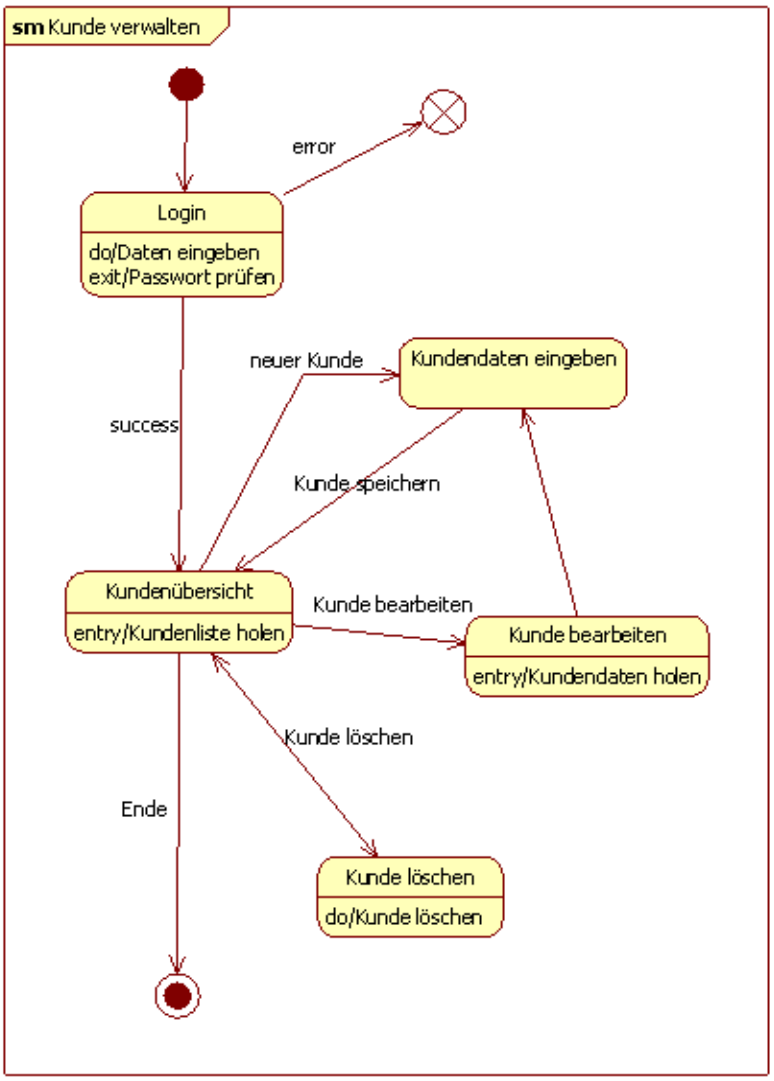

Abbildung 10: Zustandsdiagramm „Kunde verwalten“

Besonders beachtenswert ist, auch in diesem Beispiel, ein Vergleich zwischen dem modellierten Zustandsdiagramm und der Konfigurationsdatei der Implementierung. In der Beispielanwendung [Demo06] wurde dieser Teil mit Hilfe des Frameworks „Struts“ implementiert. Auch in „Struts“ gibt es eine XML-Konfigurationsdatei („*struts-config.xml*“), welche das Element „*actions-mapping*“ beinhaltet. Innerhalb

dieses Elements werden Aktionen der Web Anwendung definiert. Dieser Teil der Konfigurationsdatei spiegelt den modellierten Zustandsautomaten gut wieder (in Deutsch übersetzt von [Demo06]):

```
<?xml version="1.0"?>
<!DOCTYPE struts-config PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
  "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
  ...
  <action-mappings>
    <action path="/login" name="..." type="..." scope="session">
      <forward name="error" path="/pages/error.jsp"/>
      <forward name="success" path="/kunden_uebersicht.do"/>
    </action>
    <action path="/kunden_uebersicht" name="..." type="...">
      <forward name="neuer_kunde"
          path="/pages/kundendaten_eingeben.jsp"/>
      <forward name="kunden_uebersicht"
          path="/kunden_uebersicht.do"/>
    </action>
    <action path="/kunde_bearbeiten" name="..." type="...">
      <forward name="kundendaten_eingeben"
          path="/pages/kundendaten_eingeben.jsp"/>
    </action>
    <action path="/kunde_speichern" name="..." type="...">
      <forward name="kunden_uebersicht" redirect="true"
          path="/kunden_uebersicht.do"/>
    </action>
    <action path="/kunde_loeschen" name="..." type="...">
      <forward name="kunden_uebersicht" redirect="true"
          path="/kunden_uebersicht.do"/>
    </action>
    ...
  </action-mappings>
  ...
</struts-config>
```

Listing 3: „Kunde verwalten“ Konfigurationsdatei [Demo06]

Wie wir in diesem Beispiel sehen können, werden Zustände mit dem Element „*action*“ definiert. Änderungen von Zuständen werden mit dem Element „*forward*“ definiert.

Die Konfigurationsdatei des Frameworks „Struts“ hat im Vergleich zur oben gezeigten Konfigurationsdatei des Frameworks „Spring Webflows“ einen entscheidenden Nachteil. Das Element „*forward*“, welches Änderungen von Zuständen beschreibt, besitzt das Attribut „*path*“. Dieses Attribut gibt die folgende Seite an. Leider wird hier kein eindeutiger Identifier verwendet. Dadurch haben Änderungen von Zustän-

den keinen fix definierten Endzustand.

Protokollzustandsdiagramm

Als weiteres Beispiel wird ein Protokollzustandsdiagramm des Objektes „Warenkorb“ modelliert. Mittels Zustandsdiagrammen kann eine übersichtliche Darstellung der Zustände in einem Objekt modelliert werden.

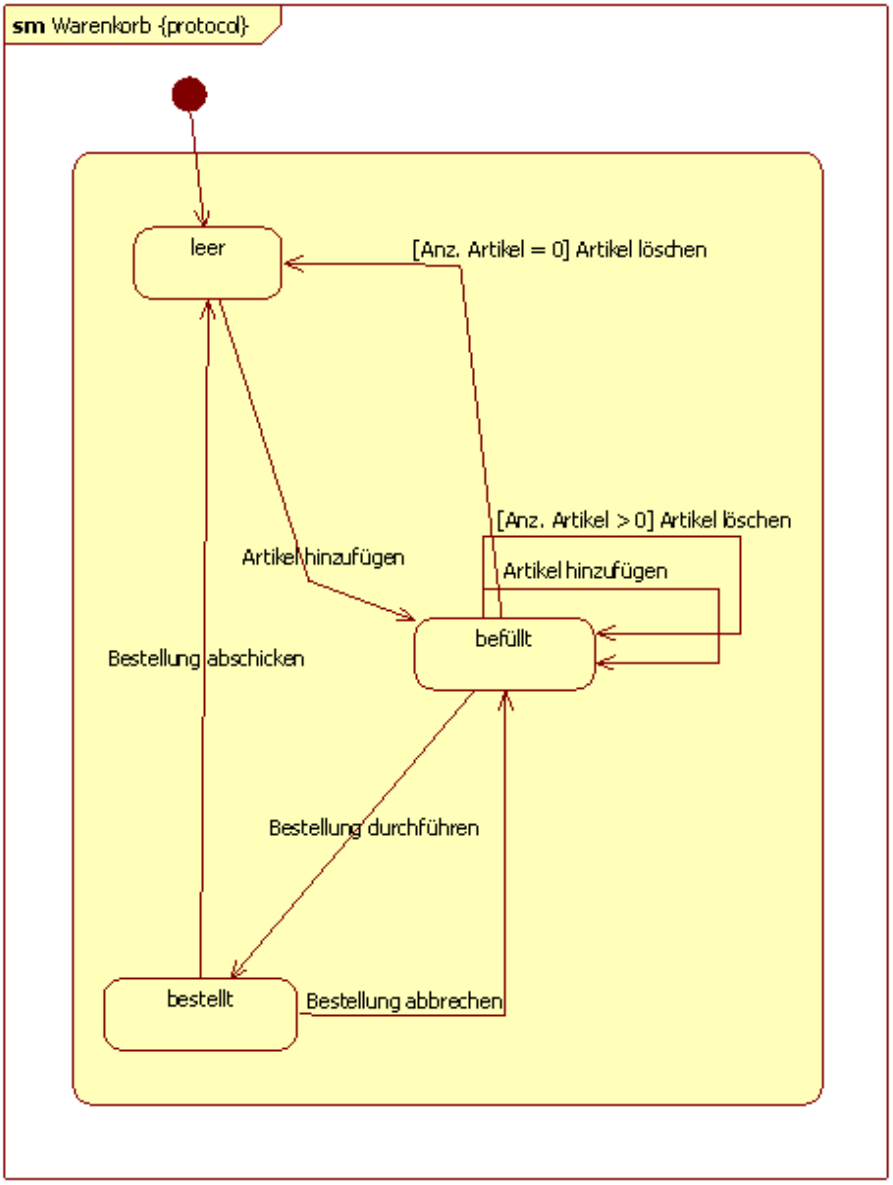

Abbildung 11: Protokollzustandsdiagramm Warenkorb

Mögliche Zustände eines Warenkorbes in einer Online Anwendung sind üblicher-

weise „leer“, „befüllt“ und „bestellt“.

Der Ausgangszustand eines Warenkorbs in einer Online Anwendung ist üblicherweise „leer“. Nach dem Hinzufügen des ersten Artikels ist der Warenkorb im Zustand „befüllt“. Werden weitere Artikel in den Warenkorb hinzugefügt bleibt der Zustand des Objektes gleich. Wird ein Artikel aus dem Warenkorb gelöscht, ändert sich der Zustand des Objektes nicht, außer wenn alle Artikel aus dem Warenkorb gelöscht wurden, dann ist der Warenkorb wieder im Zustand „leer“.

Ist das Objekt Warenkorb im Zustand „befüllt“ kann die Bestellung durchgeführt werden. Nachdem die Bestellung durchgeführt wurde, nimmt das Objekt Warenkorb den Zustand „bestellt“ ein. In diesem Zustand kann die Bestellung abgebrochen werden. Wird die Bestellung abgebrochen nimmt das Objekt Warenkorb wieder den Zustand „befüllt“ ein.

Befindet sich das Objekt im Zustand „bestellt“, kann die Bestellung abschickt bzw. bestätigt werden. Bei dieser Änderung des Zustandes werden die Produkte im Warenkorb bestellt und das Objekt Warenkorb nimmt wieder den Zustand „leer“ ein.

3.4 Sequenzdiagramm

Sequenzdiagramme zählen in UML 2.0 zu den Interaktiondiagrammen und dienen zur Modellierung von Interaktionen. Eine Interaktion spezifiziert die Art und Weise, wie Nachrichten und Daten zwischen verschiedenen Interaktionspartnern ausgetauscht werden [Hitz05].

Ein Sequenzdiagramm soll die Frage „*Wie läuft die Kommunikation in meinem System ab?*“ beantworten [Rupp05].

Das Sequenzdiagramm stellt ein sehr mächtiges Werkzeug für die Modellierung in UML 2.0 dar. Mittels Sequenzdiagrammen lassen sich Interaktionen auf verschiedensten Abstraktionsebenen modellieren. Diese Diagrammart erlaubt sowohl die Modellierung des Systemkontexts, wie auch die Modellierung eines detaillierten Operationsablaufs. Bei der Modellierung des Operationsablaufs werden die Aufrufe einzelner Methoden modelliert. Diese Art von Sequenzdiagrammen stellt einen detaillierten Blick auf das System dar.

Das ist auch der Grund, wieso Sequenzdiagramme in vielen Phasen des Projekts

sinnvoll eingesetzt werden können [Rupp05].

Sequenzdiagramme bieten auch eine gute Möglichkeit, um verwendete Architekturen in einem System zu beschreiben. In der Praxis werden des Öfteren verschiedene Designpattern mit Hilfe eines Sequenzdiagramms erklärt (vergleiche z.B. [Sun07]).

Einen guten Überblick über die verschiedenen Einsatzmöglichkeiten von Sequenzdiagrammen in Bezug auf die Phasen in einem Projekt soll die folgende Abbildung zeigen.

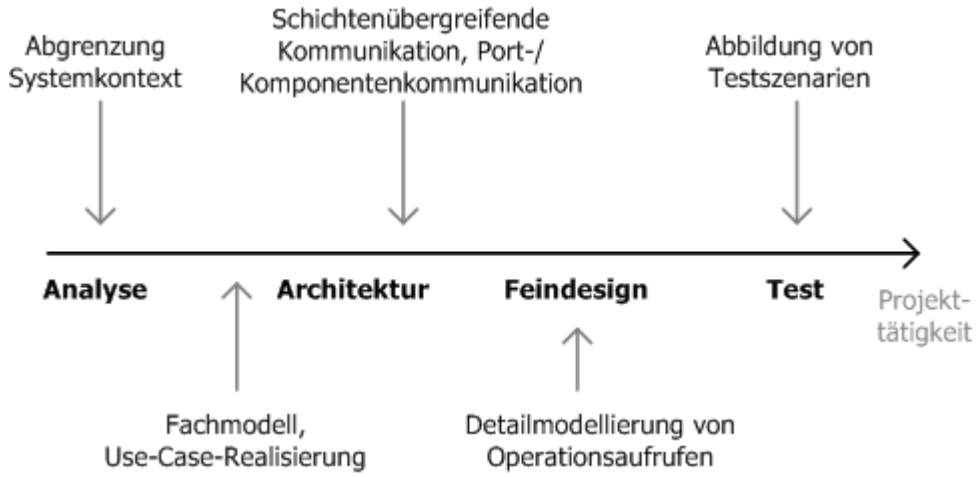

Abbildung 12: Anwendungen von Sequenzdiagrammen im Projekt [Rupp05]

Der Nachteil am Sequenzdiagramm ist, dass die Modellierung relativ aufwendig ist. Deshalb kann kein Projekt im angemessenen Zeitrahmen komplett mit Sequenzdiagrammen modelliert werden [Rupp05]. Ziel von Sequenzdiagrammen soll es eher sein, komplexe Abläufe in einem System zu beschreiben.

3.4.1 Sequenzdiagramme und Enterprise Anwendungen

Wie oben beschrieben, gibt es in beinahe jeder Phase eines Projektes eine gute Einsatzmöglichkeit für Sequenzdiagramme. Mit Hilfe eines Sequenzdiagramms kann sowohl die Architektur des Systems, wie auch Teile eines konkreten Anwendungsfalls modelliert werden.

Im folgenden Kapitel werden zwei Patterns einer Java Enterprise Anwendung modelliert. Zunächst soll das DAO-Pattern modelliert werden. Als zweites Architekturkonzept wird das Intercepting Filter-Pattern mittels Sequenzdiagrammen beschrieben.

Zusätzlich wird die 4-Schichtenarchitektur und der Lebenszyklus eines Java Servlets

mit Hilfe des Sequenzdiagramms modelliert. Weitere Architekturbeschreibungen mittels Sequenzdiagrammen befinden sich in [Sun07].

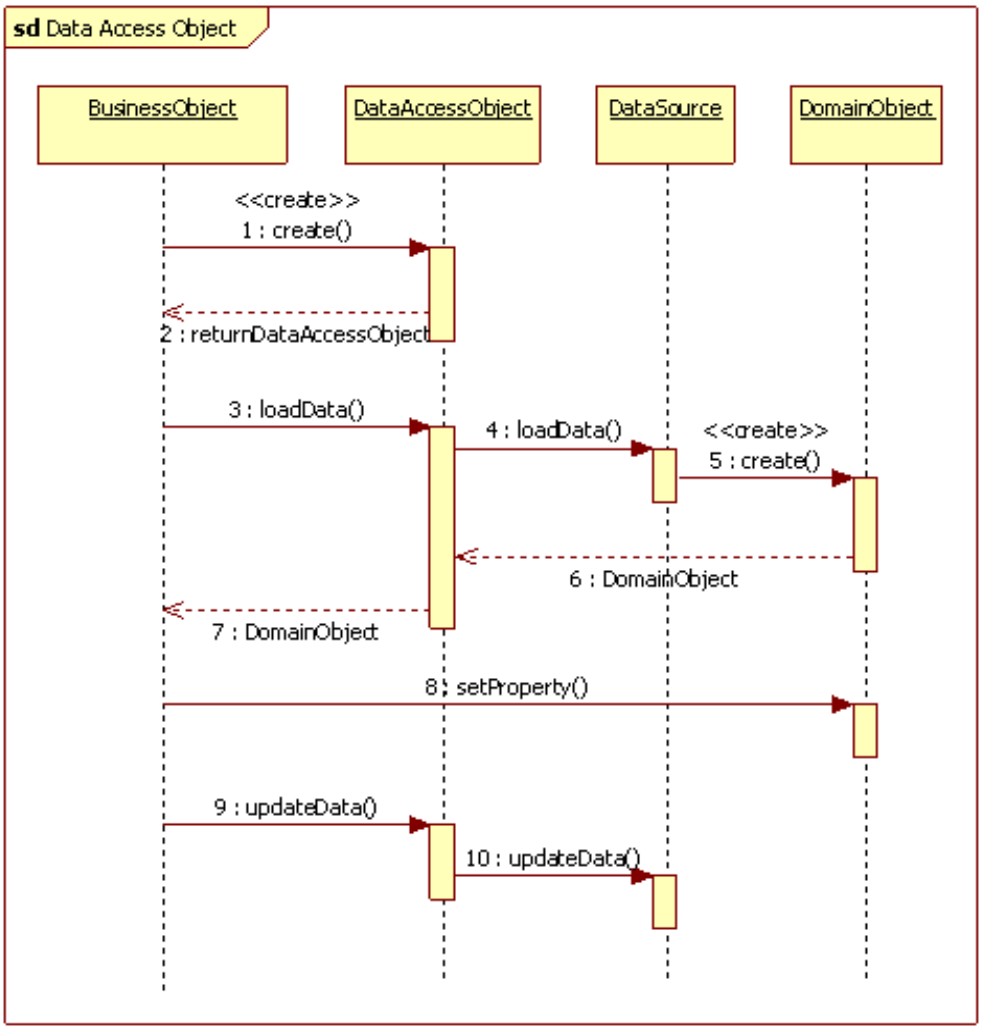

Abbildung 13: Sequenzdiagramm DAO-Pattern (basiert auf [Sun07])

Dieses Beispiel soll das Laden eines Objekts demonstrieren, welches in weiterer Folge geändert, und wieder in die Datenbank geschrieben werden soll.

Wie in der Abbildung gut ersichtlich kapselt das DAO-Pattern die Zugriffe auf die Datenspeicherungsschicht. Das Business Objekt greift nicht direkt auf die Datenquelle zu, sondern nur auf das Datenzugriffsobjekt. Das ist der große Vorteil dieses Patterns, da die Datenquelle leicht austauschbar bleibt (z.B. von einer Datenbank zu XML Dateien). Bei einer Änderung der Datenquelle sind keine Änderungen an der Business Logik notwendig. Das Framework „Spring“ bietet hier einen weiteren Vor-

teil, indem das Zugriffsobjekt in einer Konfigurationsdatei festgelegt wird und so keine Notwendigkeit besteht, den Quellcode zu verändern.

Als zweites Architekturpattern soll das Intercepting Filter-Pattern mittels Sequenzdiagramm modelliert werden.

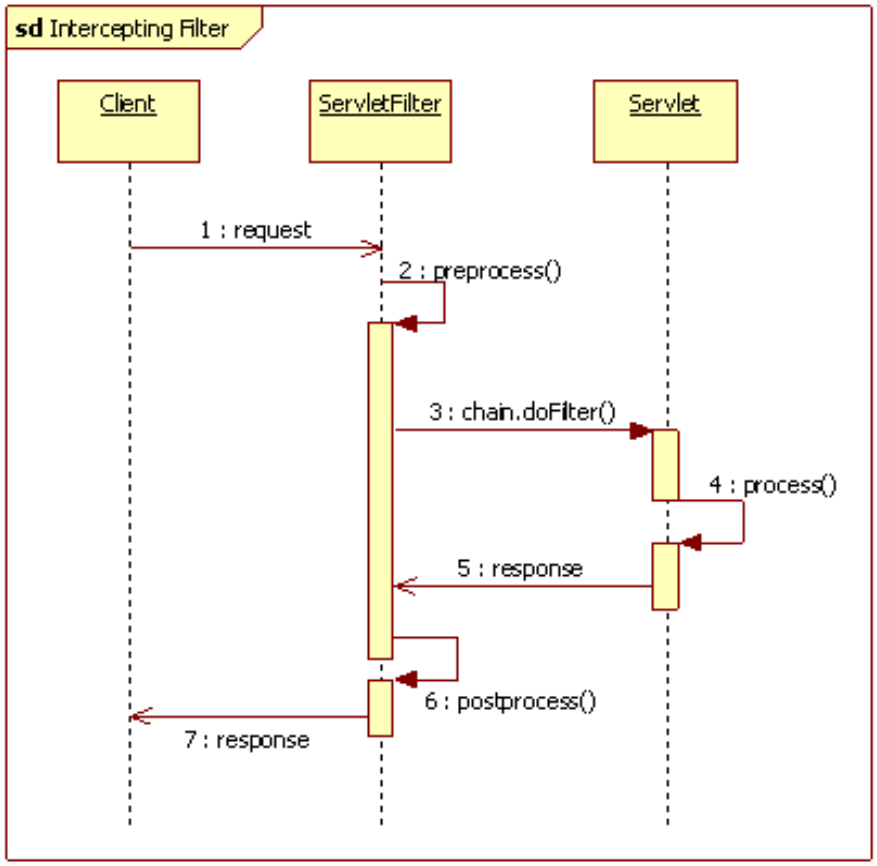

Abbildung 14: Sequenzdiagramm Intercepting Filter-Pattern [Wang03]

Das Intercepting Filter-Pattern erlaubt die Verarbeitung von Daten einer Anfrage („Request“) bzw. einer Antwort („Response“) vor bzw. nach der eigentlichen Verarbeitung. Damit können Daten manipuliert werden oder die Anfrage an ein anderes Servlet weitergeleitet werden. Ein Filter kann zum Beispiel die Überprüfung übernehmen, ob ein Benutzer am System angemeldet ist bzw. ob ein Benutzer die Berechtigung besitzt dieses Servlet aufzurufen. Wenn der Benutzer angemeldet ist werden die Daten des Benutzers aus einer Datenbank gelesen, sonst wird die Anfrage an ein anderes Servlet weitergeleitet, welches den Benutzer auffordert sich anzumelden.

In der Praxis werden Servlet Filter meistens für die Authentifizierung bzw. Autorisierung verwendet.

Weitere Einsatzmöglichkeiten sind unter anderem Logging, Tracking, Performanzmessungen, Datenkomprimierungen oder Verschlüsselungen [Wang03].

Als weiteres Beispiel soll ein allgemein gehaltenes Sequenzdiagramm modelliert werden, welches den Aufruf der einzelnen Schichten darstellt. Mit Hilfe eines Sequenzdiagramms kann das 4-Schichten Modell und seine Datenflüsse gut und übersichtlich dargestellt werden.

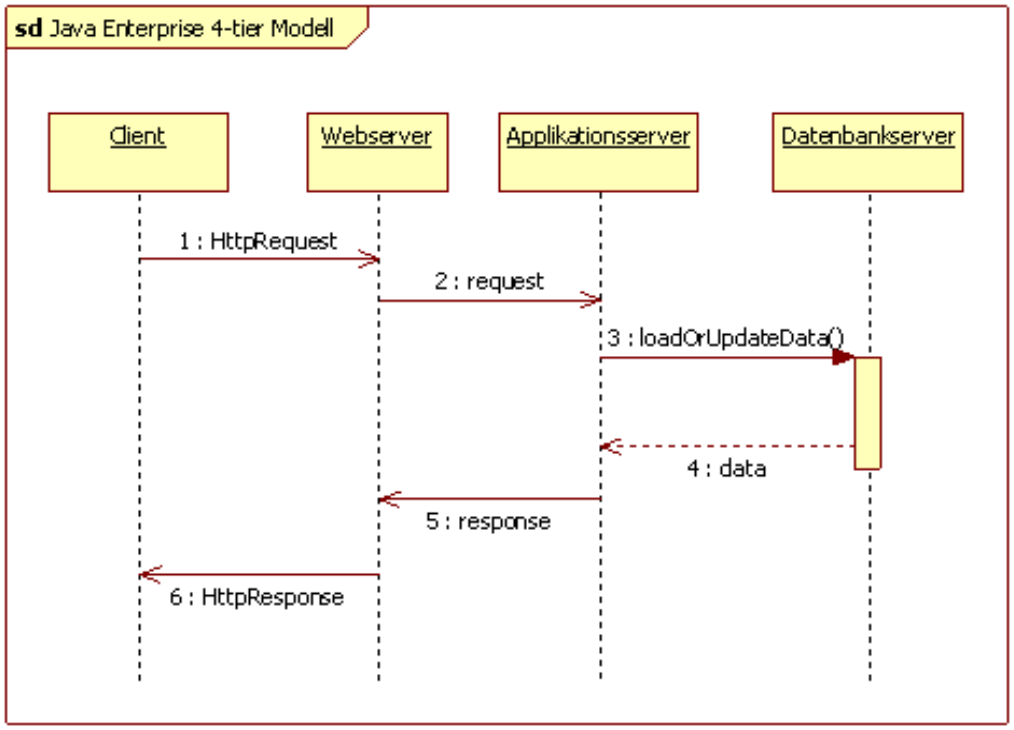

Abbildung 15: Sequenzdiagramm 4-Schichtenmodell

Die vier Schichten sind Client, Webserver, Anwendungsserver und Datenbankserver. Ein Client schickt eine Anfrage („*HttpRequest*“) an den Webserver. Dieser verarbeitet die Daten, und leitet eine entsprechende Anfrage an den Anwendungsserver weiter. Der Anwendungsserver, welcher die Businesslogik implementiert, speichert die empfangenen Daten in die Datenbank bzw. liest die Daten von der Datenbank und schickt eine entsprechende Antwort an den Webserver zurück. Der Webserver verarbeitet diese Daten und schickt die Antwort an den Client zurück („*HttpResponse*“).

Als weiteres Beispiel soll der Lebenszyklus eines Java Servlets mit Hilfe eines Sequenzdiagramms modelliert werden.

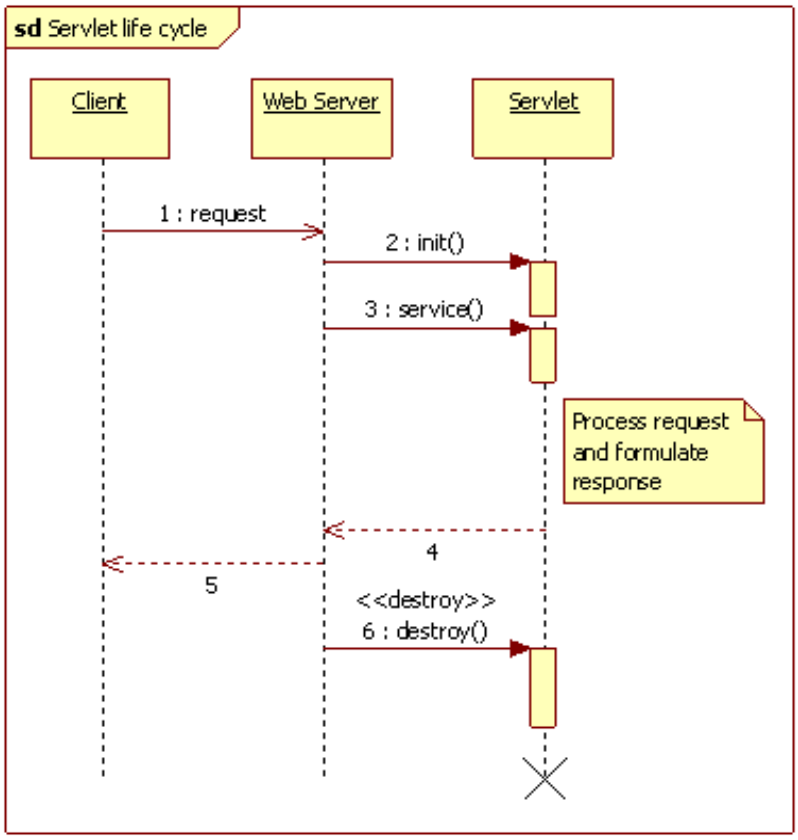

Abbildung 16: Sequenzdiagramm Lebenszyklus Java Servlet [Ahme02]

Der Client schickt eine Anfrage („*request*“) an den Webserver. Dieser legt das Java Servlet an und initialisiert es. Nach dem Aufruf der „*init()*“ Methode, wird die Servicemethode aufgerufen. Hier wird nach der Requestmethode unterschieden („*HTTP GET, POST, ...*“). In der Servicemethode findet die eigentliche Verarbeitung statt, und das Antwortobjekt („*response*“) wird erzeugt. Das Java Servlet gibt das Antwortobjekt an den Webserver weiter, welcher dieses Objekt wieder zurück an den Client schickt. Danach wird das Java Servlet vom Webserver zerstört, nachdem die Methode „*destroy()*“ aufgerufen wird.

Dieses Beispiel soll demonstrieren, dass mittels Sequenzdiagrammen auch detaillierte technische Spezifikationen modelliert werden können.

3.4.2 Sequenzdiagramme im begleitenden Beispiel

Als Sequenzdiagramm im begleitenden Beispiel wird ein Analysemodell des gesamten Systems modelliert. Als zweites Beispiel wird der Anwendungsfall „Kunde anlegen“ im Detail modelliert.

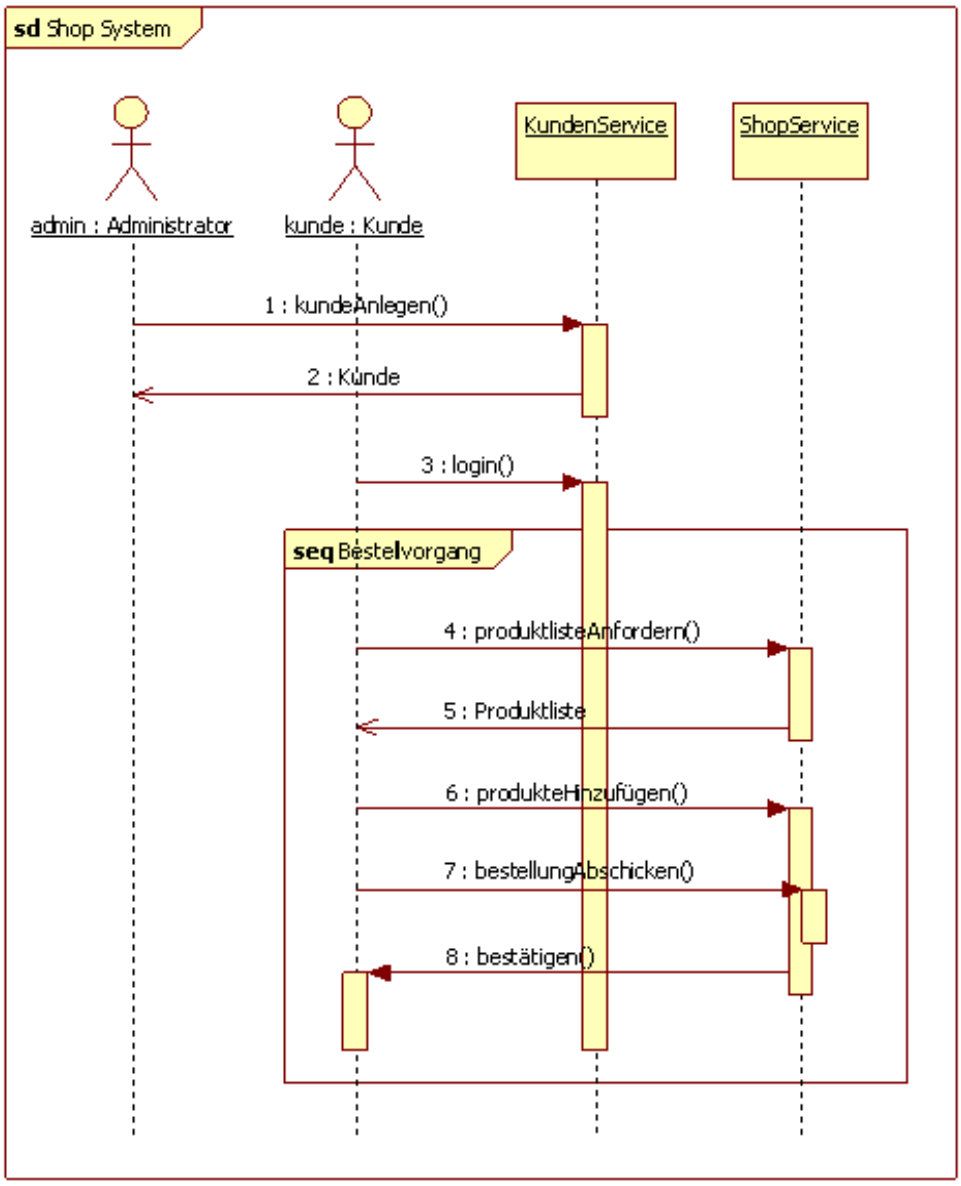

Abbildung 17: Sequenzdiagramm Gesamtsystem

Das Sequenzdiagramm des gesamten Systems soll zeigen, dass Sequenzdiagramme in Bezug auf deren Abstraktionsebene sehr weit gestreut sind. Die Gesamtübersicht zeigt das Zusammenspiel der verschiedenen Aktoren und Komponenten. Ein Administrator legt einen Kunden an. Dieser kann sich danach mit seinen Benutzerdaten am

System anmelden und mit dem Bestellvorgang beginnen. Für den Bestellvorgang muss der Kunde zunächst die Produktliste anfordern. Danach können Produkte aus dieser Liste in seinem Warenkorb hinzugefügt werden. Nachdem die Bestellung abgeschickt wurde, bekommt der Kunde eine Bestätigung von seiner Bestellung.

Als zweites Beispiel soll das implementierte Teilprojekt „CustomerStruts“ und das Zusammenspiel mit dem Teilprojekt „CustomerService“ [Demo06] gezeigt werden.

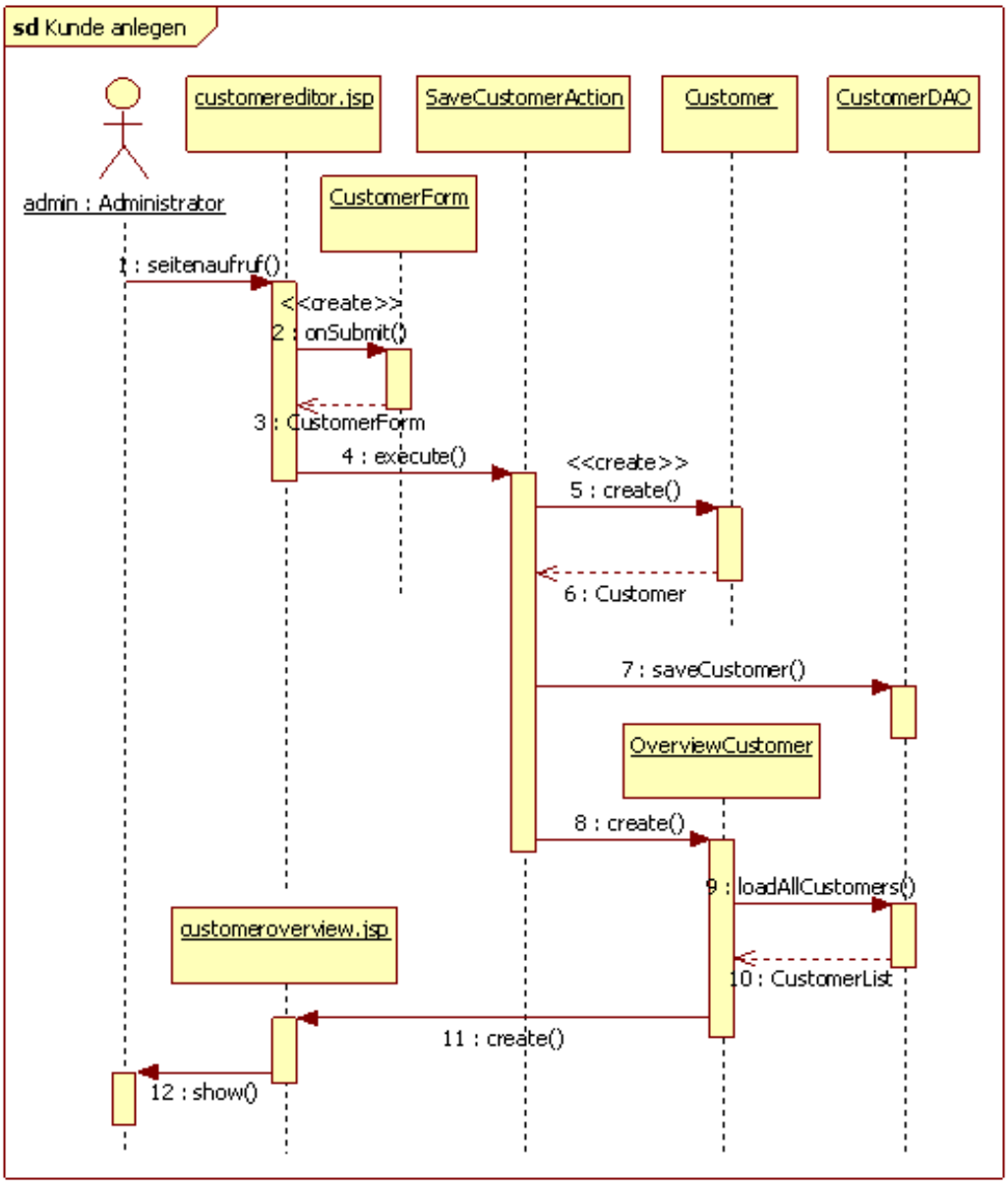

Abbildung 18: Sequenzdiagramm „Kunde anlegen“

Das Framework „Struts“ hält sich strikt an das MVC-Pattern. Als Model werden so

genannte Form-Klassen verwendet, die im wesentlichem die Daten halten. Als Controller werden Action-Klassen verwendet, die sich um die eigentliche Logik kümmern, und für die Präsentation werden üblicherweise JSP-Seiten verwendet.

Das modellierte Sequenzdiagramm spiegelt die Implementierung wieder. Ein Administrator ruft die Seite „*customereditor.jsp*“ auf und gibt die Daten des anzulegenden Kunden ein. Diese Eingabe wird als Anfrage an den Webserver gesendet. Die gesendeten Daten werden vom Framework „Struts“ in ein „*CustomerForm*“ Objekt umgewandelt. Mit diesem Objekt als Parameter wird die Klasse „*SaveCustomerAction*“ aufgerufen. Diese Klasse dient im Framework „Struts“ als Controller des MVC-Patterns. Dieser Controller übernimmt die Verarbeitung der Daten. Es wird ein, für die Businesslogik lesbares, Objekt erstellt und das Datenzugriffsobjekt wird mit diesem erstellten Objekt als Parameter aufgerufen. Das Datenzugriffsobjekt speichert die Daten in die Datenbank. Der Controller „*SaveCustomerAction*“ erstellt die Folgeseite („*OverviewCustomer*“). Diese Seite holt sich die Liste aller Kunden vom Datenzugriffsobjekt und leitet die Daten an die Anzeigeseite weiter. Der Administrator sieht als Ergebnis die Seite „*customeroverview.jsp*“, welche die Liste aller Kunden anzeigt.

In diesem Beispiel sieht man das Zusammenspiel und die Trennung der einzelnen Schichten des 4-Schichtenmodells. Allerdings gilt es hier zu beachten, dass die Objekte schon vom Framework Spring über „Dependency Injection“ gesetzt und angelegt werden.

Wie man in diesem Beispiel schon sehen kann, benötigt der relativ einfache Anwendungsfall „Kunde anlegen“ schon einige Objekte. Bei komplexeren Anwendungsfällen wird das Sequenzdiagramm schnell unübersichtlich. UML 2.0 bietet als alternative das Kommunikationsdiagramm, welches im nächsten Kapitel beschrieben wird.

Zusammenfassung

Zusammengefasst kann gesagt werden, dass Sequenzdiagramme sowohl die Modellierung abstrakter Architekturen erlauben, wie auch einen konkreten Anwendungsfall. In der Literatur werden Architekturen sehr oft mittels Sequenzdiagrammen beschrieben. Der Nachteil an Sequenzdiagrammen ist, dass sie sehr schnell unübersichtlich werden und relativ aufwendig in der Modellierung sind.

3.5 Kommunikationsdiagramm

Das Kommunikationsdiagramm stellt das System auf einer relativ abstrakten Ebene dar. Es soll die Frage „*Welche Teile einer komplexen Struktur arbeiten wie zusammen, um eine bestimmte Funktion zu erfüllen?*“ [Rupp05] beantworten.

Ein Kommunikationsdiagramm bildet auf mittlerer Abstraktionsebene das Zusammenspiel von Kommunikationspartner zur Lösung einer gemeinsamen Aufgabe ab [Rupp05].

Bei diesem Diagramm steht, im Vergleich zum Sequenzdiagramm, die Übersicht im Vordergrund. Details und die zeitliche Abfolge sind beim Kommunikationsdiagram nicht entscheidend [Rupp05].

Deshalb sind Kommunikationsdiagramme leichter zu modellieren und leichter zu lesen als Sequenzdiagramme.

In älteren Versionen von UML wurde das Kommunikationsdiagramm unter dem Namen Kollaborationsdiagramm geführt [Hitz05].

3.5.1 Kommunikationsdiagramme und Enterprise Anwendungen

Kommunikationsdiagramme werden unter anderem für folgende Problemstellungen verwendet:

- Kommunikationsdiagramme können verwendet werden, um die *Implementierung eines Anwendungsfalls* zu modellieren [Hitz05].
- Außerdem können Kommunikationsdiagramme für die Modellierung der *Implementierung einer Operation* verwendet werden [Hitz05].
- Das Kommunikationsdiagramm dient zur Darstellung von *komplexen Strukturen* auf eine übersichtliche Art und Weise [Rupp05].
- Mit Hilfe eines Kommunikationsdiagramms kann die *grundsätzliche Struktur* eines Systems modelliert werden [Rupp05].

Das heißt das Kommunikationsdiagramm ist, wie auch das Sequenzdiagramm, in Bezug auf die Abstraktionsebene sehr weit gestreut und kann auch in vielen Phasen im Projekt eingesetzt werden.

Der Vorteil am Kommunikationsdiagramm, im Vergleich zum Sequenzdiagramm, ist dass neben den Interaktionen auch strukturelle Aspekte modelliert werden können. Dadurch ist, im Vergleich zu Sequenzdiagrammen, kein zusätzliches Diagramm notwendig um strukturelle Aspekte zu modellieren [Hitz05].

Aus den genannten Eigenschaften ergibt sich, dass Kommunikationsdiagramme eine gute Übersicht über die Schichten einer Enterprise Anwendungen geben. Die folgende Abbildung soll den Aufbau des 4-Schichtenmodells einer Enterprise Anwendung in Form eines Kommunikationsdiagramms zeigen.

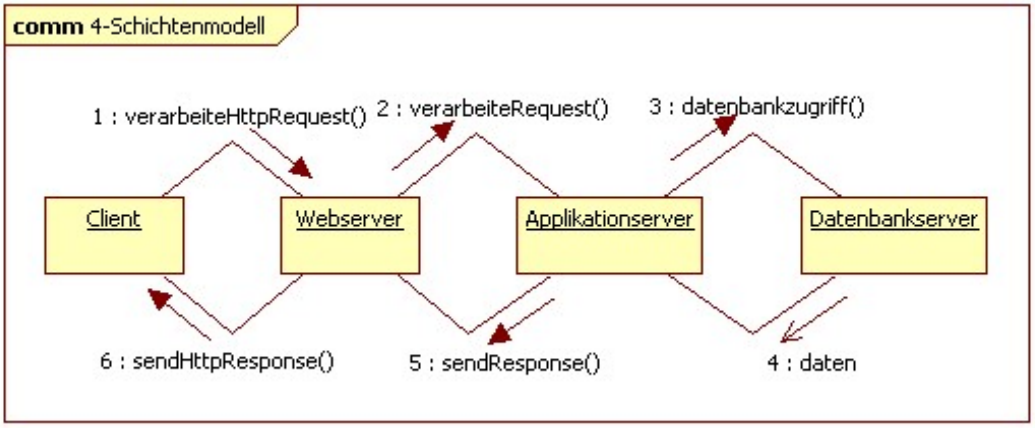

Abbildung 19: Kommunikationsdiagramm 4-Schichtenmodell

Das Kommunikationsdiagramm bietet, im Vergleich zum Sequenzdiagramm, eine bessere Übersicht über das 4-Schichtenmodell.

3.5.2 Kommunikationsdiagramme im begleitetem Beispiel

Im folgenden Beispiel wird der Anwendungsfall „Kunde anlegen“, der schon im vorigen Kapitel als Sequenzdiagramm dargestellt wurde, modelliert.

Dieses Diagramm hat die gleiche Aussage, wie das Sequenzdiagramm im vorigen Kapitel. Ein Kommunikationsdiagramm bietet allerdings mehr Übersicht, als es ein Sequenzdiagramm erlaubt.

Das Kommunikationsdiagramm bietet ähnliche Möglichkeiten wie das Sequenzdiagramm in einer vereinfachten Form. Es soll der Übersicht dienen, und auch komplexe Strukturen, die im Sequenzdiagramm unübersichtlich wirken, sollen mit Hilfe des

Kommunikationsdiagramms dargestellt werden können.

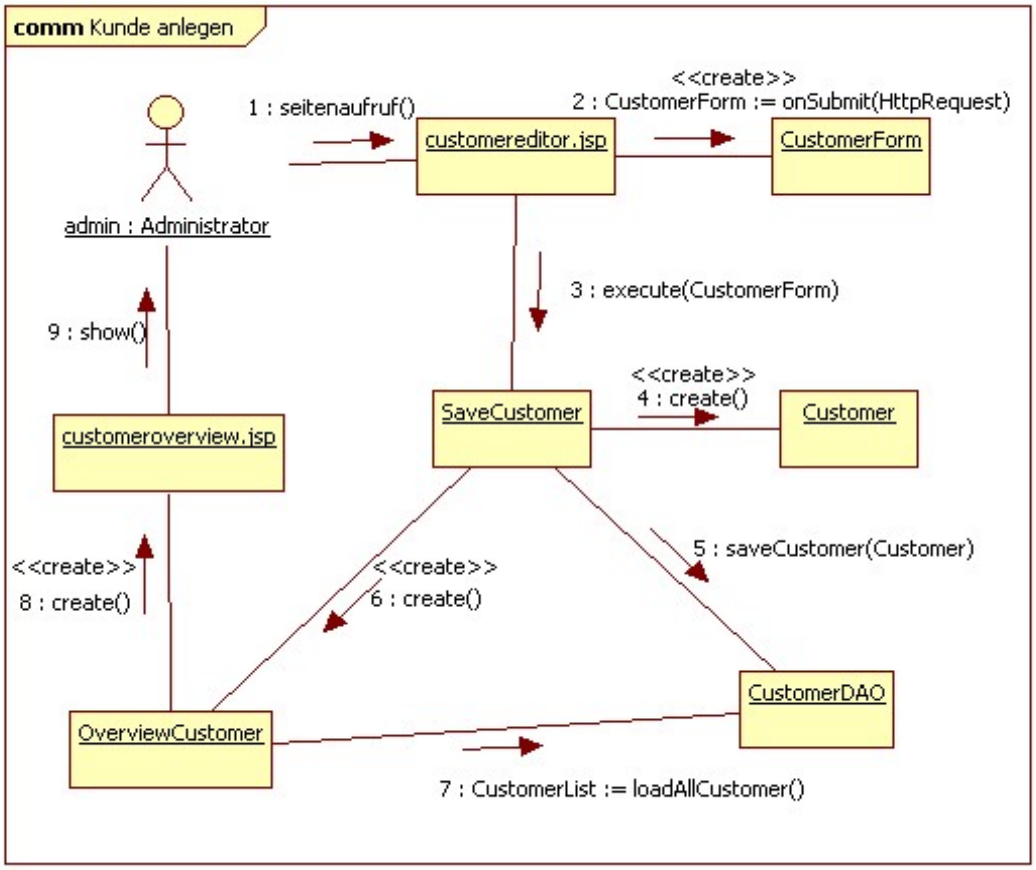

Abbildung 20: Kommunikationsdiagramm „Kunde anlegen“

Wie diese Abbildung zeigt, gibt das Kommunikationsdiagramm einen guten Überblick über den Anwendungsfall „Kunde anlegen“.

3.6 Zeitdiagramm

Das Zeitdiagramm zeigt das zeitliche Verhalten in einem System. Es wurde in UML 2.0 neu eingeführt und soll die Frage: „*Wann befinden sich verschiedene Interaktionspartner in welchem Zustand?*“ beantworten [Rupp05].

Zeitdiagramme werden für die Modellierung von zeitkritischen Systemen, wie es Echtzeitsysteme sind, verwendet [Hitz05]. Im Unterschied zu Sequenz- bzw. Kommunikationsdiagrammen können mittels Zeitdiagrammen der Ablauf nur exemplarisch darstellt werden, da laut Standard keine Kontrollstrukturen zur Verfügung stehen [Hitz05].

3.6.1 Zeitdiagramme und Enterprise Anwendung

Zeitdiagramme sind unter anderem unter folgenden Bedingungen einzusetzen [Rupp05]:

- Bei einem stark modularisiertem System mit stark abhängigen Zustandsdiagrammen, um die Kommunikation zwischen den Automaten darzustellen.
- Wenn beim dem zu implementierenden System genaue zeitliche Übergänge wichtig sind.
- Wenn für den zu modellierenden Sachverhalt lokale und globale Daten weniger interessant sind.
- Wenn bei der Anwendung Interaktionen einfach gestrickt sind und Nebenläufigkeiten oder Kontrollelemente unnötig sind.

Alle oben beschrieben Punkte treffen bei einer Enterprise Anwendung nur sehr wenig zu, deshalb ist es relativ schwierig ein passendes Zeitdiagramm zu finden.

Als Beispiel wurde versucht ein Zeitdiagramm für die Benutzersession zu modellieren. Das Ergebnis zeigt folgende Abbildung:

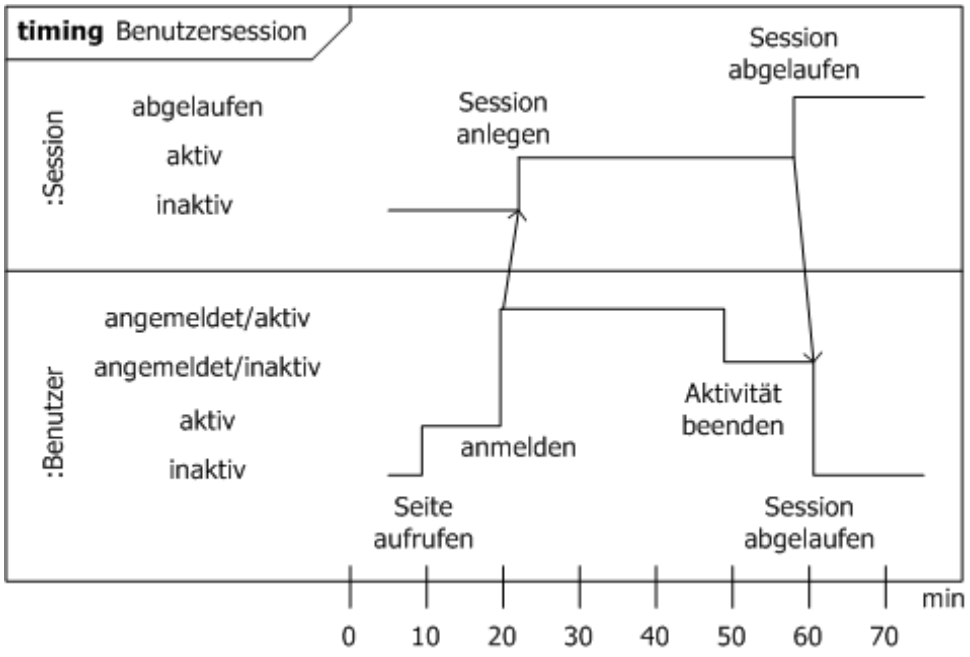

Abbildung 21: Zeitdiagramm Benutzersession

Nachdem sich der Benutzer auf einer Internetseite eingeloggt hat wird die Benutzersession angelegt. Diese bleibt solange aktiv, solange der Benutzer auch aktiv ist (z.B. durch klicken auf links). Ist der Benutzer für 10 Minuten (dieser Wert ist konfigurierbar) nicht aktiv, gilt die Benutzersession als abgelaufen. Wird der Benutzer wieder aktiv, so muss er sich erneut anmelden.

Die Aussage dieses Zeitdiagramms kann auch mit anderen Diagrammen, wie z.B. dem Sequenzdiagramm, dargestellt werden. Dabei können Zeit und Zustandsinformationen mittels Notizen modelliert werden [Hitz05].

Leider kommt in diesem Diagramm die zu modellierende Aussage, dass eine Session nach 10 Minuten abgelaufen ist, nicht gut zum Vorschein. Ein Sequenzdiagramm oder ein Kommunikationsdiagramm mit einer entsprechenden Notiz eignen sich mit Sicherheit besser, um das hier zu modellierende Problem darzustellen.

3.6.2 Zeitdiagramme im begleiteten Beispiel

Da der zeitliche Ablauf im begleitenden Beispiel in diesem Abstraktionsgrad nicht relevant ist, kann hier kein passendes Beispiel modelliert werden.

3.7 Interaktionsübersichtsdiagramm

Das Interaktionsübersichtsdiagramm zeigt das Zusammenspiel verschiedener Interaktionen. Die Darstellung erfolgt in Form einer Abwandlung des Aktivitätsdiagramms [Rupp05]. Ein Interaktionsübersichtsdiagramm soll die Frage „*In welcher Reihenfolge und unter welchen Bedingungen finden Interaktionen statt?*“ beantworten [Rupp05].

Ein Interaktionsübersichtsdiagramms verwendet verschiedene Komponententypen aus anderen Diagrammarten. Deshalb wurde es öfters kritisiert, soll aber dennoch einen guten Überblick über die Ablaufreihenfolge verschiedener Interaktionsdiagramme geben [Hitz05].

3.7.1 Interaktionsübersicht und Enterprise Anwendungen

Interaktionsübersichtsdiagramme sollen Zusammenhänge zwischen anderen Interaktionsdiagrammen modellieren, um die Übersicht über das System zu behalten [Rupp05].

Werden viele Interaktionsdiagramme, die im direkten Zusammenhang stehen, modelliert, ist es zielführend Interaktionsübersichtsdiagramme zu erstellen. Betrachtet man eine Enterprise Anwendung, so stehen die modellierten Architekturen kaum in einem direkten Zusammenhang.

3.7.2 Interaktionsübersichtsdiagramm im begleiteten Beispiel

Im begleitenden Beispiel stehen Interaktionsdiagramme nur sehr wenig im direkten Zusammenhang, sondern sind eher eigene abgegrenzte Modelle.

Als Beispiel wird hier ein Interaktionsübersichtdiagramm über das Teilsystem „Artikel bestellen“ modelliert. Die dazu verwendeten Interaktionen sind „Benutzer anmelden“, „Warenkorb befüllen“, „Adresse eingeben“ und „Bestellung abschicken“.

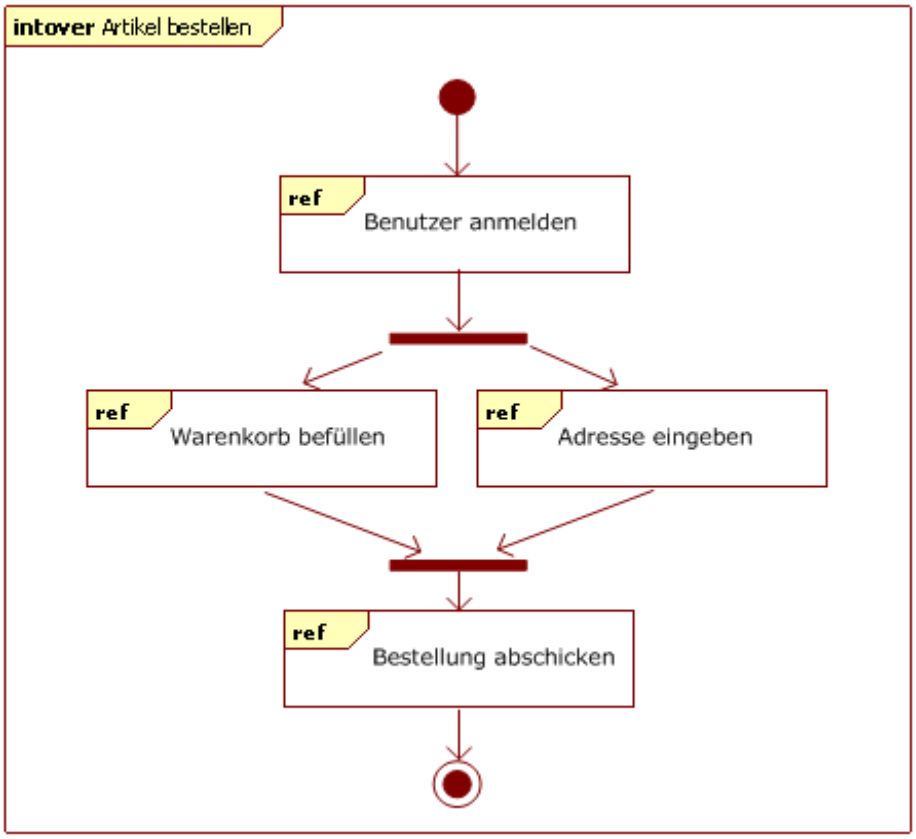

Abbildung 22: Interaktionsübersichtdiagramm „Artikel bestellen“

Dieses Beispiel soll demonstrieren, dass die Verwendung von Interaktionsübersichtsdiagramme sinnvoll ist, um den Zusammenhang zwischen anderen Interaktionsdiagrammen zu zeigen.

Kapitel 4

UML 2 - Strukturdiagramme

Strukturdiagramme beschreiben - wie der Name schon sagt - die Struktur eines Systems. Strukturdiagramme bieten eine Vielzahl von Möglichkeiten, die von der Aufbaustruktur einer Klasse bis hin zur Gliederung vollständiger Architekturen und Systeme reichen [Rupp05].

4.1 Klassendiagramm

Ein Klassendiagramm zeigt die statische Struktur des Systems, indem es die statischen Eigenschaften einer Klasse und deren Beziehungen zueinander beschreibt. Das Klassendiagramm bildet den Kern der Modellierungssprache UML 2.0 [Rupp05]. Es soll die Frage „*Wie sind die Daten und das Verhalten meines Systems im Detail strukturiert?*“ beantworten [Rupp05].

Klassendiagramme sind die zentrale Diagrammart in fast jedem Projekt. Sie können in jedem Schritt der Softwaremodellierung in unterschiedlicher Abstraktionsebene verwendet werden.

Prinzipiell können zwei Arten von Klassendiagrammen unterschieden werden, wobei der Übergang fließend ist [Rupp05]:

- *Konzeptuell-analytische Modellierung*: Die Konzeptuell-analytische Modellierung wird in der Analysephase eines Projekts eingesetzt. Das heißt ein Klassendiagramm soll in diesem Fall einen abstrakten Überblick über das System geben. Ziel auf dieser Ebene ist es, die korrekten Zusammenhänge

im Projekt zu definieren. Das Klassendiagramm soll die Fachkonzepte und deren Beziehungen zueinander abbilden, dabei stehen konkrete Attribute und Operationen im Hintergrund [Rupp05].

- *Logische, design-orientierte Modellierung*: Die logische, design-orientierte Modellierung findet in einer späteren Phase im Projekt statt. Sie soll als Vorbild für die Implementierung verwendet werden. Ein Klassendiagramm könnte/sollte es sogar erlauben den Quelltext automatisch generieren zu lassen. Hier werden alle Attribute und Operationen, mit deren Datentypen bzw. Rückgabewerte und deren Sichtbarkeit beschrieben [Rupp05].

An dieser einführenden Beschreibung lässt sich schon erkennen, wieso Klassendiagramme ein sehr beliebtes Werkzeug für die Modellierung sind. Klassendiagramme sind gut lesbar, in jeder Phase des Projektes einsetzbar und erlauben große Unterschiede im Hinblick auf ihre Abstraktionsebene.

4.1.1 Klassendiagramme und Enterprise Anwendungen

Die Modellierung mittels Klassendiagrammen ist sowohl für jede einzelne Schicht des 4-Schichtenmodells, wie auch für das Zusammenspiel der Schichten geeignet.

In einer sehr hohen Abstraktionsstufe kann ein Klassendiagramm einen Überblick über die verwendete Architektur im System geben. In einer sehr detaillierten Abstraktionsebene gibt das Klassendiagramm einen Einblick in die Implementierung der Anwendung.

Konzeptionelle / Analytische Modellierung

Eine gute Anwendung für eine Konzept-analytische Modellierung des Klassendiagramms ist die Darstellung von Patterns. In der Literatur werden Patterns sehr oft mit einem Klassendiagramm und einem Sequenzdiagramm (vgl. Kapitel 3.4) beschrieben (vgl. z.B. [Sun07]).

Das folgende Kapitel soll zwei Bespiele geben, wie Designpatterns mit Hilfe eines Klassendiagramms modelliert werden können. Das erste Bespiel zeigt das DAO-Pattern, das zweite Beispiel zeigt, wie das Intercepting Filter-Pattern mittels Klas-

sendiagramm beschrieben werden kann.

Wie in vorigen Kapiteln schon beschrieben kapselt das DAO-Pattern die Datenzugriffslogik von der Businesslogik. Beide Klassen verwenden so genannte Transferobjekte um miteinander zu kommunizieren.

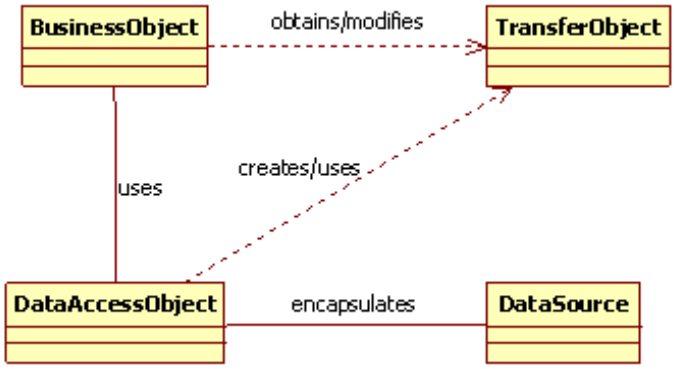

Abbildung 23: Klassendiagramm DAO-Pattern [Sun07]

In diesem Klassendiagramm lässt sich gut der Sinn dieses Pattern erkennen. Das Businessobjekt greift nicht direkt auf die Datenquelle zu, was diese leicht austauschbar macht. Um die Datenquelle zu ändern, muss nur die Datenzugriffsklasse angepasst werden, die Businesslogik kann hingegen unverändert bleiben.

Als weiteres Beispiel wird das Intercepting Filter-Pattern als Klassendiagramm modelliert.

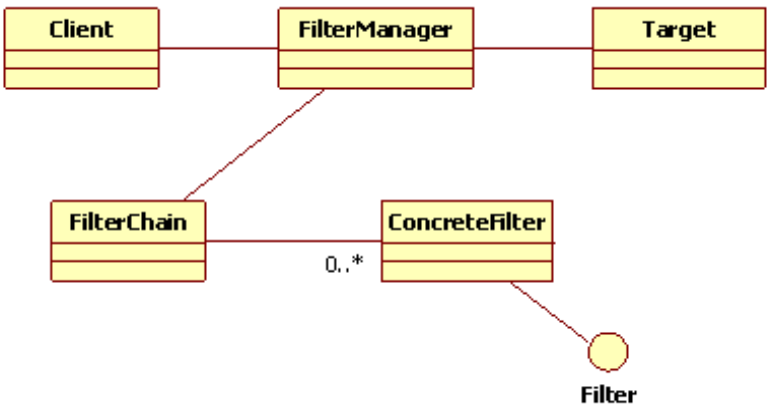

Abbildung 24: Klassendiagramm Intercepting Filter-Pattern [Sun07]

Dieses Pattern erlaubt die Manipulation von Anfragen („Request“) bzw. Antworten („Response“) vor bzw. nach der eigentlichen Verarbeitung. Es wird in der Präsentationsschicht unter anderem für die Authentifizierung bzw. Autorisierung von Benutzern verwendet.

Bevor die Daten zum eigentlichen Ziel kommen, prüft der Filtermanager, ob Filter vorhanden sind. Sind Filter vorhanden, werden diese nach der Reihe abgearbeitet.

Dieses Klassendiagramm alleine ist nicht aussagekräftig, gibt allerdings in Verbindung mit dem Sequenzdiagramm (siehe oben) eine gute und vollständige Beschreibung dieses Patterns. Mit dem Sequenzdiagramm kann nicht modelliert werden, dass mehrere Filter im System vorkommen können.

Zusammengefasst kann gesagt werden, dass zur Modellierung von Designpattern, Klassendiagramme in Verbindung mit Sequenzdiagrammen am besten geeignet sind, wie es auch in der Literatur öfters verwendet wird (vgl. z.B. [Sun07]).

Logische / Design-orientierte Modellierung

Im Rahmen der logisch design-orientierte Modellierung wird versucht einen allgemeinen Überblick über den Aufbau einer Enterprise Anwendung zu geben, wie folgende Abbildung zeigen soll:

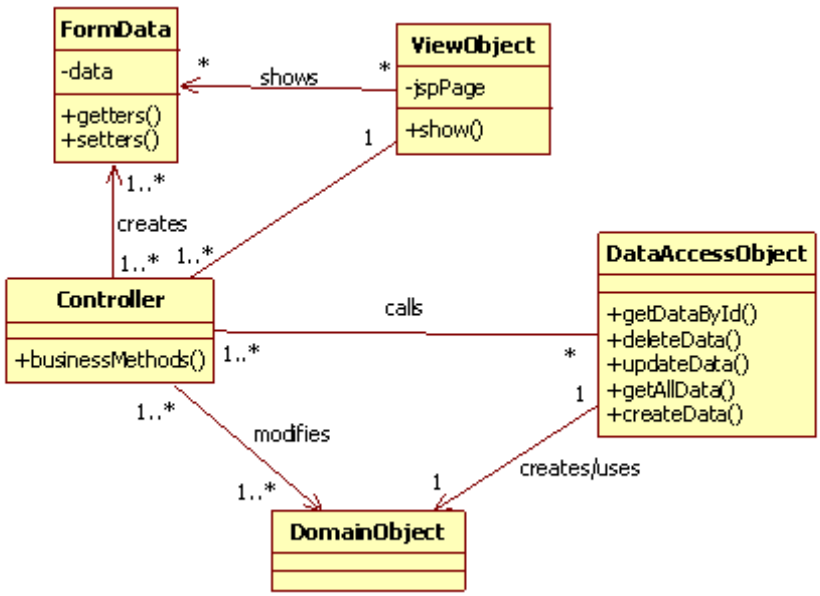

Abbildung 25: Klassendiagramm Enterprise Anwendung (nach [Demo06])

Dieses Beispiel zeigt das Zusammenspiel zwischen der Präsentationsschicht und den Datenzugriffsmethoden, wie sie im begleiteten Beispiel [Demo06] implementiert wurden.

Das Anzeigeobjekt zeigt Daten an und ruft den Controller auf. Im Controller befinden sich die Businessmethoden, welche Datenzugriffsobjekte verwenden, um Daten aus der Datenbank zu lesen und in die Datenbank zu schreiben. Der Controller und das Datenzugriffsobjekt verwenden Domainobjekte als gemeinsame Basis.

4.1.2 Klassendiagramme im begleitenden Beispiel

Hier modellierte Klassendiagramme sollen nach der Abstraktionsebene gegliedert werden. Als erstes Beispiel soll eine Übersicht über das System in Form des Domainmodells gegeben werden. Das zweite Beispiel zeigt die Implementierung des Pakets „Kundenverwaltung“.

Das erste Beispiel zeigt das Domainmodell in Form eines Klassendiagramms:

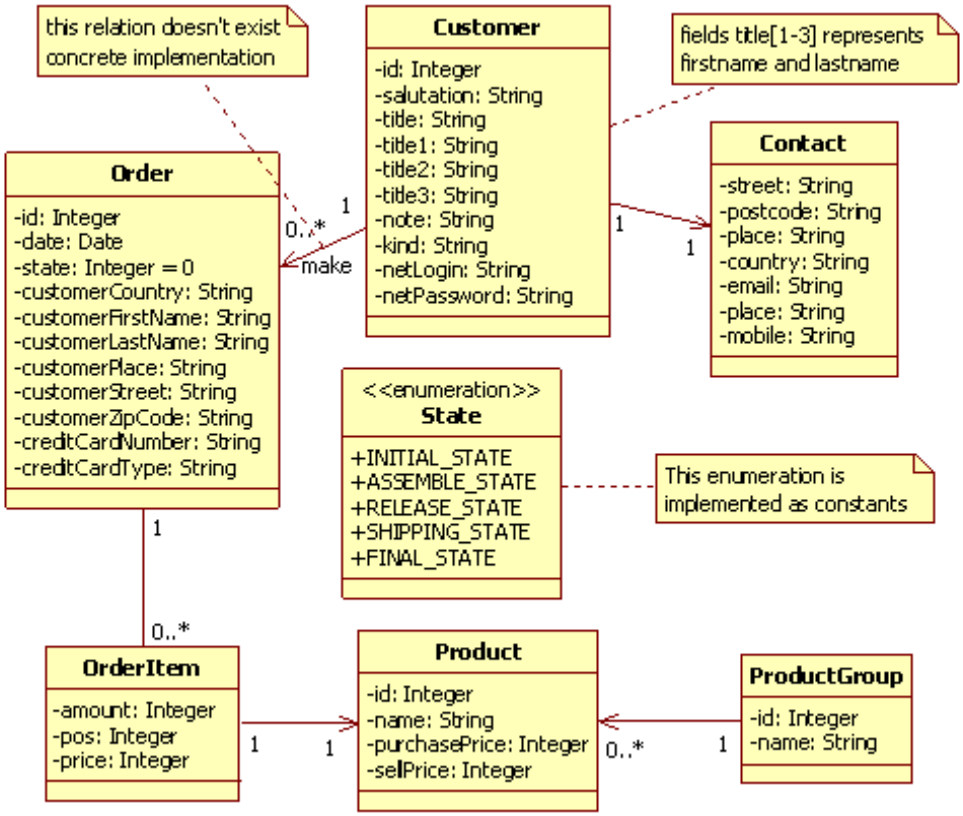

Abbildung 26: Klassendiagramm Domainmodell (basiert auf [Demo06])

Ein Domainmodellklassendiagramm gibt einen guten Überblick über das System und seine Meta-Daten. Es werden alle Klassen mit deren Attributen, Typen von Attributen und auch mögliche Defaultwerte modelliert.

Jede dieser modellierten Klassen hat als Operatoren „getter“ und „setter“ Methoden, die zum Lesen und Setzen der Attribute dienen. Diese werden in der Praxis oft weggelassen, um die Übersicht zu bewahren.

Aus diesem Diagramm könnten im Rahmen einer modellgetriebenen Softwareentwicklung Klassen generiert werden und auch teilweise Datenbankskripts zum Erstellen von Datenbanktabellen. Leider wird in UML 2.0 nur der Typ der Attribute angegeben und keine weiteren Informationen, wie z.B. die Feldlänge.

Jeder Kunde („Customer“) hat ein Objekt der Klasse Kontakt („Contact“), und tätigt mehrere Bestellungen („Order“). In der konkreten Implementierung gibt es keine Verbindung zwischen der Klasse „Order“ und der Klasse „Customer“, sondern die Attribute des Kunden werden neu abgelegt, da sich die aktuelle Versandadresse von der tatsächlichen Adresse des Kunden unterscheiden kann [Demo06]. Hier liegt eigentlich ein Fehler in der Modellierung vor, besser wäre es, dem Kunden mehrere Adressen zuzuweisen, von dem eine die aktuelle Versandadresse darstellt. Um mit der Implementierung im Einklang zu bleiben, wurde die aktuelle Struktur übernommen und die Verbindung nur in Form einer Notiz modelliert.

Eine Bestellung („Order“) besteht aus mehreren Teilen („OrderItem“). Jeder Teil einer Bestellung ist genau einem Produkt zugewiesen. Diese Zwischenklasse wurde angelegt, um ein Produkt öfters bestellen zu können. Ein Produkt ist in genau einer Produktgruppe, wobei eine Produktgruppe mehrere Produkte beinhalten kann, aber auch leer sein kann.

Als weiteres Beispiel soll das Paket „Kundenverwaltung“ modelliert werden. Dabei erfolgt noch keine Einteilung in Pakete (siehe Kapitel 4.2 - Paketdiagramm).

Dieses Modell wird wegen der besseren Übersicht in zwei Teile geteilt. Zuerst werden die Datenobjekte und die JSP-Seiten modelliert (Präsentationsschicht), im zweiten Schritt werden die Controllerklassen und Datenzugriffsobjekte modelliert (Businesslogikschicht).

Klassen die aus den Frameworks „Struts“ oder „Spring“ kommen werden andersfärbig dargestellt.

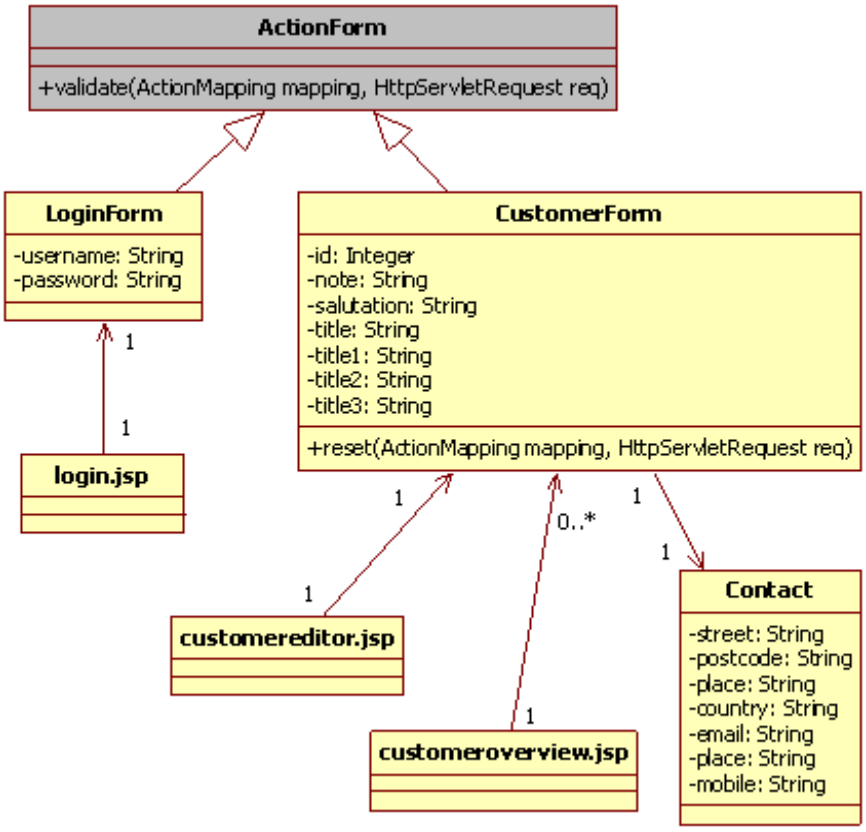

Abbildung 27: Klassendiagramm Kundenverwaltung I

Wie aus diesem Klassendiagramm ersichtlich werden die Datenobjekte („LoginForm“ und „CustomerForm“) von der Klasse „ActionForm“ aus dem Framework „Struts“ abgeleitet. In der Basisklasse befindet sich auch die Methode „validate“, welches zum Prüfen der Datenobjekte dient. In der konkreten Implementierung wurde dieses Feature nicht verwendet.

Die JSP-Seiten zeigen Datenobjekte an bzw. erzeugen neue Datenobjekte. Die Seite „customereditor.jsp“ dient zur Bearbeitung von Objekten der Klasse „CustomerForm“, die Seite „customeroverview.jsp“ dient zum Anzeigen einer Liste von „CustomerForm“ Objekten.

Als zweiter Teil werden die Controller- und Datenzugriffsklassen mit Hilfe des Klassendiagramms modelliert.

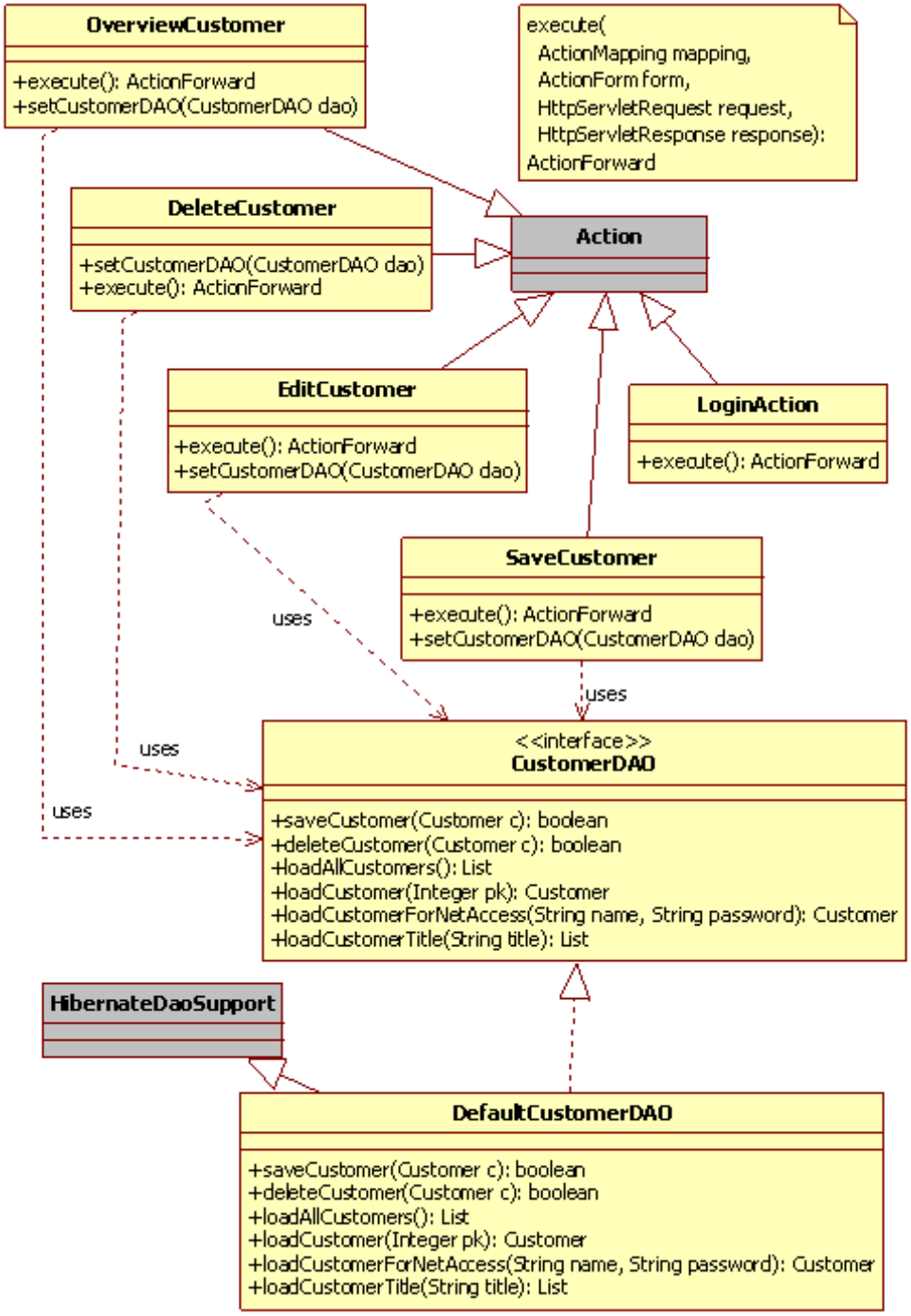

Abbildung 28: Klassendiagramm Kundenverwaltung II

Jede Controllerklasse („LoginAction“, „OverviewCustomer“, „DeleteCustomer“, „EditCustomer“ und „SaveCustomer“) erbt ihre Grundfunktionalität von der Basisklasse aus dem Framework „Struts“ („Action“). Jede dieser Klassen überschreibt die Methode „execute“, in der die eigentliche Businesslogik implementiert ist. Die zwei-

te Methode, die in jeder Controllerklasse vorhanden ist („setCustomerDAO“) dient zum setzen des Datenzugriffsobjekts. Das Setzen des Datenzugriffsobjekts wird vom Framework „Spring“ übernommen. In der Konfigurationsdatei des Frameworks wird festgelegt welche Implementierung verwendet wird. In diesem Fall wird ein Objekt der Klasse „DefaultCustomerDAO“ verwendet.

In der konkreten Implementierung werden die Datenzugriffsobjekte direkt angesprochen, und nicht, wie es eigentlich sein sollte, über die Business Logik. In der Modellierung wurde diese Logik übernommen.

Dieses Beispiel soll zeigen, dass ein Klassendiagramm eine gute Vorlage für die Implementierung sein kann.

4.2 Paketdiagramm

Das Paketdiagramm dient zur Gliederung des Gesamtsystems in Pakete. Es soll die Frage „*Wie kann ich mein Modell so darstellen, dass ich den Überblick behalte?*“ beantworten [Rupp05].

Das Paketdiagramm bietet eine gute Möglichkeit, um einen Überblick des Gesamtsystems zu modellieren. Nachdem größere Softwareprojekte meist aus sehr vielen verschiedenen Klassen bestehen, müssen diese in Pakete zusammengefasst werden, um die Übersicht zu behalten.

4.2.1 Paketdiagramme und Enterprise Anwendungen

Das Paketdiagramm kann einen guten Überblick über die einzelnen Schichten einer Enterprise Anwendung geben. Außerdem kann ein Paketdiagramm eingesetzt werden, um eine funktionale Gliederung vorzunehmen.

Allgemein können Paketdiagramme in zwei unterschiedliche Anwendungsgebiete eingeteilt werden [Rupp05]:

- *Funktionale Gliederung*: Bei einer funktionalen Gliederung werden alle Teile, die funktional oder logisch im Zusammenhang stehen, zu einem Paket zusammengefasst. Diese Einteilung wird meist zu einem frühen Zeitpunkt im Projekt durchgeführt, da das modellierte Paketdiagramm einen guten Überblick gibt, welche Teile der Enterprise Anwendung zusammenhängend imp-

lementiert werden können [Rupp05].

- *Definition von Schichten*: Die Definition von Schichten dient zur Darstellung eines mehrschichtigen Systems, wie es auch Enterprise Anwendungen sind. In dieser Form von Paketdiagrammen werden die Pakete meist verschachtelt dargestellt.

Eine ähnliche Einteilung nach welchen Kriterien Pakete gefunden werden können gibt [Hitz05]:

- *Funktionale Kohäsion*: vgl. funktionale Gliederung.
- *Informationskohäsion*: Hier werden Elemente (Klassen) zusammengefasst, die untereinander stark gekoppelt sind und nach außen hin schwach gekoppelt sind.
- *Verteilungsstruktur*: Hier werden Elemente auf demselben physischen Knoten zusammengefasst.
- *Erfahrung*: Hier werden schwierig zu implementierende Elemente (die Erfahrung bei der Implementierung benötigen) zusammengefasst.

Bei einem Paketdiagramm sind auch verschiedene Abstraktionsebenen zu beachten. So kann ein Paketdiagramm einen sehr groben Überblick über die verwendete Architektur geben, aber auch als Gliederung eines Klassendiagramms und dadurch als Implementierungsgrundlage dienen.

Die folgenden Kapitel sollen Beispiele für eine funktionale Gliederung und Definition der Schichten in Enterprise Anwendungen geben.

Funktionale Gliederung

Die funktionale Gliederung hängt stark von einer konkreten Implementierung ab. Deshalb wird dieser Punkt in Kapitel 4.2.2 Paketdiagramme im begleiteten Beispiel näher erläutert.

Definition von Schichten

Paketdiagramme bieten eine gute Möglichkeit, um eine Einteilung der Schichten zu modellieren [Hitz05]. Die folgende Graphik zeigt ein Systemmodell einer Enterprise Anwendung.

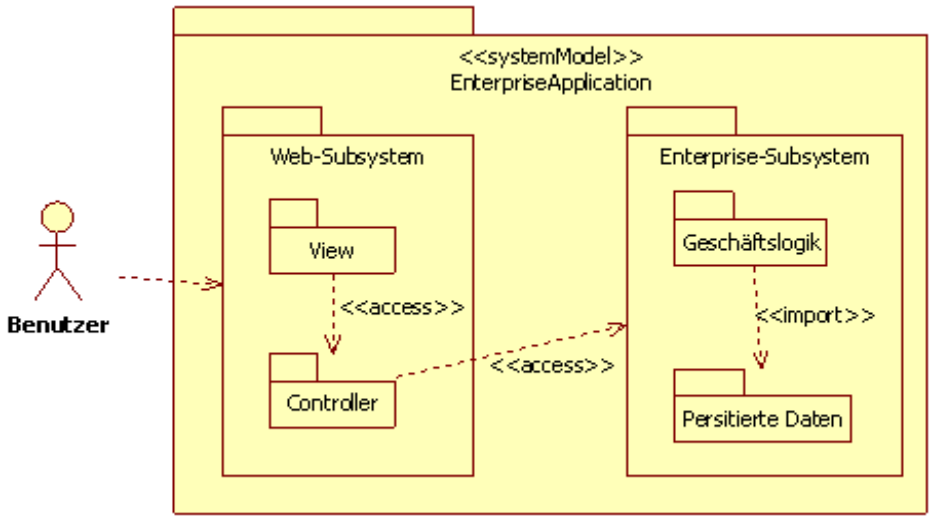

Abbildung 29: Paketdiagramm Übersicht Enterprise Anwendung

Der Aktor „Benutzer“ greift auf das „Web-Subsystem“ zu. In diesem Teilpaket befinden sich die Elemente der Präsentationsschicht und die Controllerelemente. Die Controllerelemente greift auf das Enterprise Subsystem zu. In diesem Paket ist das Unterpaket Geschäftslogik, welches auf das zweite Subpaket „persistente Daten“ zugreift, um Daten aus der Datenbank zu lesen bzw. Daten in die Datenbank zu schreiben.

4.2.2 Paketdiagramme im begleiteten Beispiel

Paketdiagramme sind ein gutes Werkzeug, um die Struktur eines Projektes darzustellen. Dies kann in verschiedenen Abstraktionsebenen geschehen.

Das erste Paketdiagramm zeigt eine funktionale Einteilung der gesamten Anwendung. Das nächste Beispiel zeigt das Teilsystem Kundenservice. Das dritte Beispiel zeigt ein Paketdiagramm, in dem des Teilsystem Kundenservice noch eine Ebene weiter aufgeteilt wurde und aus einer technischen Sicht betrachtet wird. Bei der Auswahl der Beispiele wurde auf die verschiedenen Abstraktionsebenen geachtet.

Das erste Diagramm soll einen groben Überblick über die funktionale Einteilung in Paketen geben:

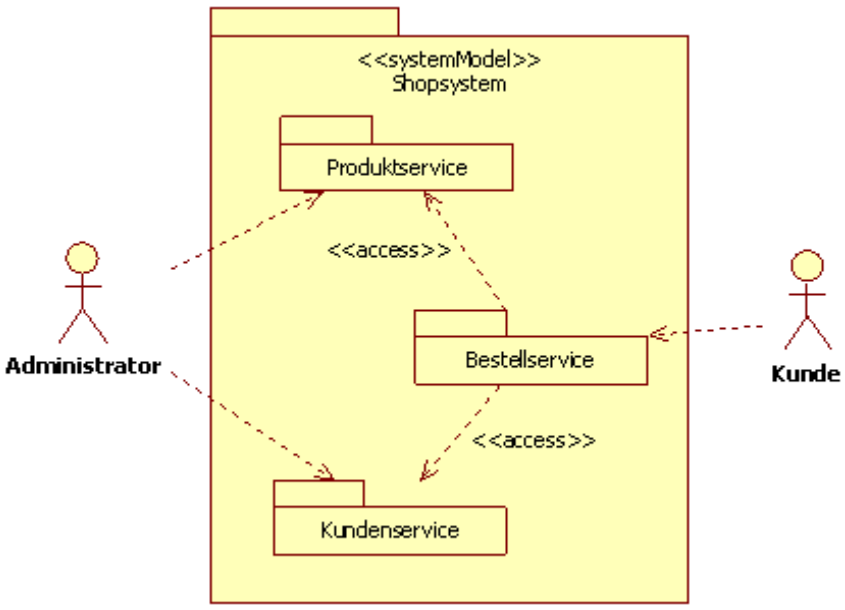

Abbildung 30: Paketdiagramm Shopsystem Übersicht

Dieses Diagramm zeigt eine grobe Übersicht über das Gesamtsystem, ohne die Pakete näher zu beschreiben. In weiterer Folge werden die in diesem Diagramm gezeigten Pakete auszugsweise einzeln modelliert.

Die gesamte Enterprise Anwendung teilt sich in drei Pakete:

- *Kundenservice*: Das Kundenservice beinhaltet die gesamte Benutzerverwaltung. Zu diesem Service zählen Anwendungsfälle wie unter anderem „Kunde anlegen“, „Kunde ändern“ oder „Kundenliste anzeigen“.
- *Bestellservice*: In diesem Paket sind alle Teile der Bestellung enthalten. Zu diesem Paket zählen unter anderem die Anwendungsfälle „Bestellung verwalten“ oder „Bestellung bestätigen“.
- *Produktservice*: Dieses Paket enthält die Funktionalität, um Produkte verwalten zu können. Dies sind insbesondere die Funktionalitäten „Produkt hinzufügen“ oder „Produkte löschen“.

Ein Administrator hat Zugriff auf das Kundenservice und das Produktservice, um administrative Tätigkeiten in der Enterprise Anwendung durchführen zu können. Ein Kunde kann Bestellungen durchführen. Dieser Anwendungsfall ist im Paket „Bestellservice“ implementiert. Das Bestellservice greift auf das Kundenservice und das Produktservice zu.

Das nächste Paketdiagramm teilt das im vorigen Diagramm modellierte Kundenservice weiter in seine Pakete auf und zeigt die Übersicht über das Teilsystem Kundenservice:

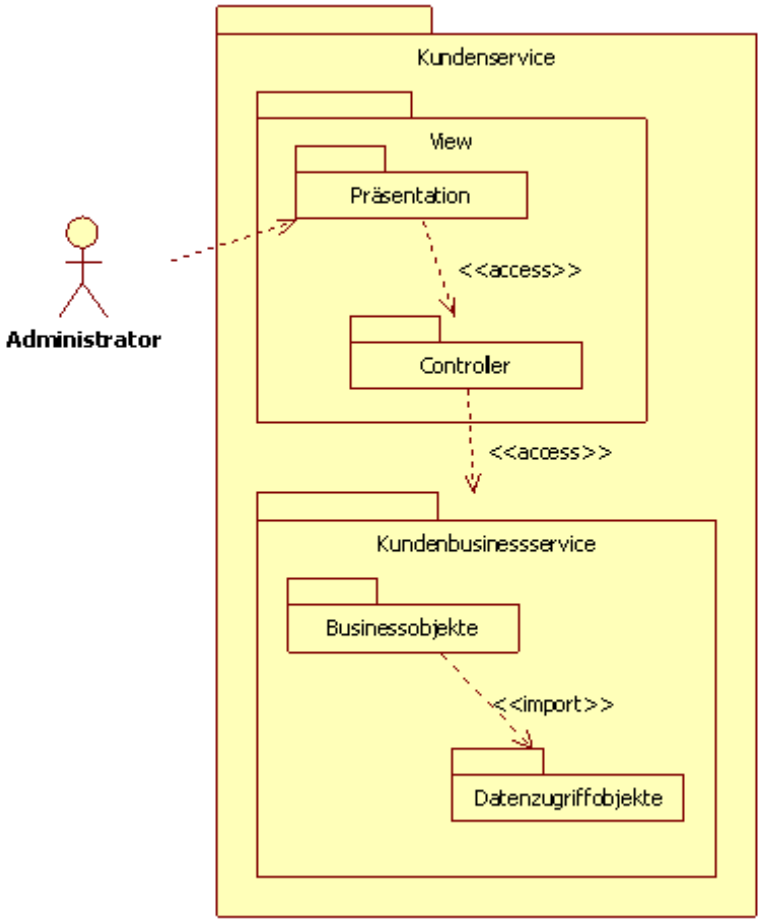

Abbildung 31: Paketdiagramm Kundenservice

Der Administrator greift auf die Präsentationsschicht zu. Diese Schicht befindet sich, wie auch die Controllerelemente im Paket View. Die Controllerelemente rufen Elemente aus dem Kundenbusinessservice auf. Im Paket Kundenbusinessservice befin-

den sich die Subpakete Businessobjekte und Datenzugriffsobjekte.

Als letztes Beispiel soll eine weitere Verfeinerungsstufe modelliert werden, wie es in der konkreten Implementierung verwendet wird. Es ist die letzte Verfeinerungsstufe eines Paketdiagramms und zeigt eine Einteilung des in Kapitel 4.2. modellierten Klassendiagramms.

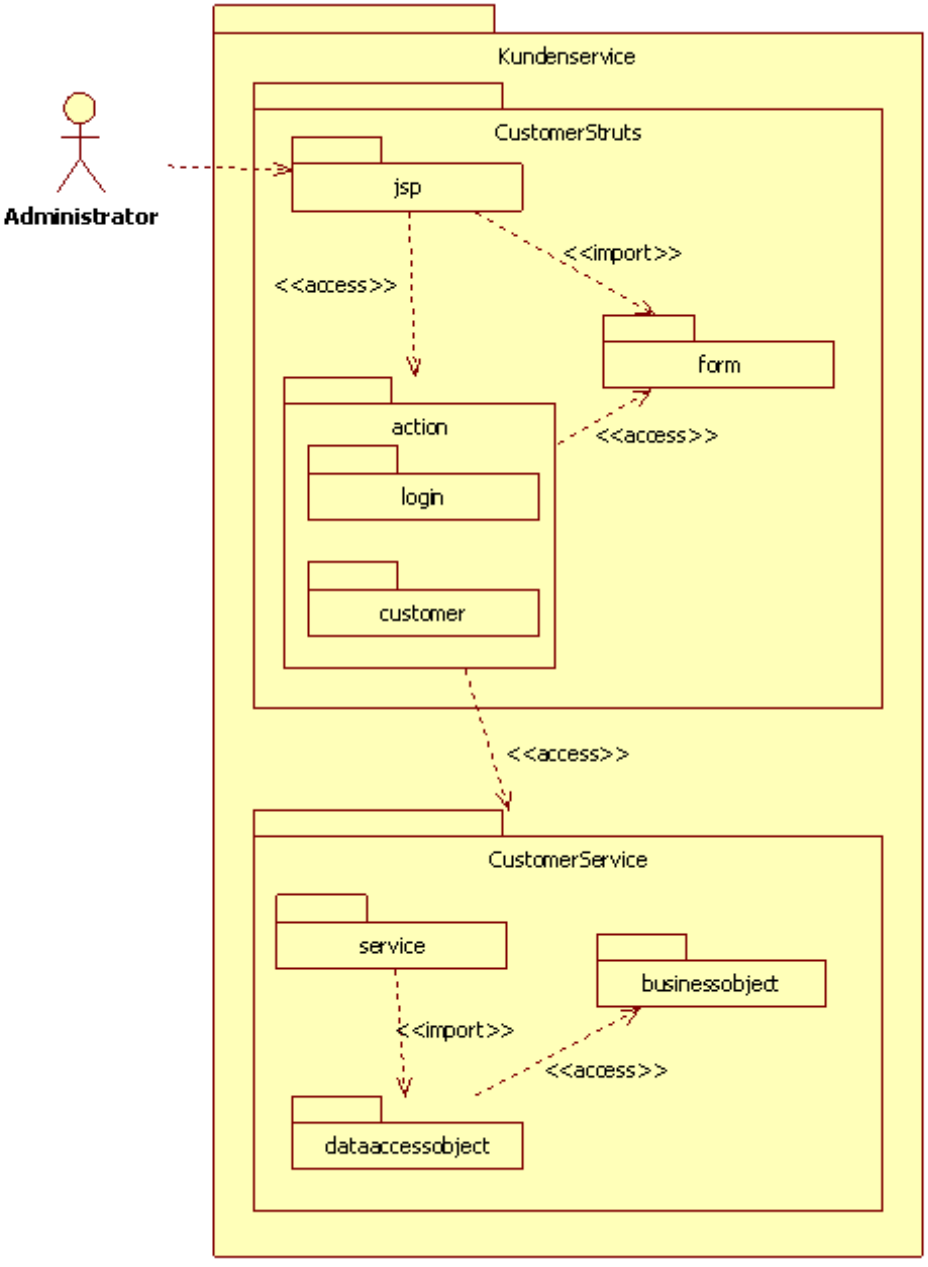

Abbildung 32: Paketdiagramm Kundenservice detailliert (technisch)

Für das oben gezeigte Paketdiagramm wurde die Einteilung des Teilprojektes, wel-

ches mit dem Framework „Struts“ realisiert wurde, gewählt. In der konkreten Implementierung gibt es zwei Teilprojekte „CustomerStruts“ und „CustomerService“, welche in diesem Diagramm als Pakete modelliert wurden. Die hier modellierten Pakete entsprechen den in der Implementierung verwendeten Paketen. Die hier verwendeten Paketnamen unterscheiden sich von den in der Implementierung verwendeten. Zum Beispiel hat das modellierte Paket „businessobject“ in der Implementierung den Namen „at.demolsky.customer.bo“ [Demo06].

In der Praxis werden Paketdiagramme und Klassendiagramme sehr oft in Kombination verwendet. So können die im vorigen Diagramm dargestellten Pakete die modellierten Klassen aus dem Klassendiagramm enthalten.

Das UML-Modellierungstool StarUML sieht keine eigene Diagrammart „Paketmodell“ vor, sondern die Elemente sind im Klassendiagramm inkludiert [Reic06].

4.3 Objektdiagramm

Ein Objektdiagramm bietet die Möglichkeit Instanzen von Klassen, Komponenten, Knoten, Assoziationen und Attributen zu modellieren. Es soll die Frage: „*Wie sieht ein Schnappschuss meines Systems zur Ausführungszeit aus?*“ beantworten [Rupp05]. Eine Klasse wird als Objekt dargestellt, ein Attribut einer Klasse als konkreter Wert, und eine Assoziation zwischen Klassen als Link.

Der Aufbau eines Objektdiagramms, ist dem Klassendiagramm sehr ähnlich, mit dem Unterschied, dass bei Objektdiagrammen Instanzen von Klassen modelliert werden.

Ein Vorteil des Objektdiagramms im Vergleich zum Klassendiagramm ist, dass sich Attribute von Klassen besser beschreiben lassen. Das Klassendiagramm erlaubt nur die Modellierung eines Defaultwertes, während beim Objektdiagramm konkrete Werte modelliert werden [Rupp05].

Das Klassendiagramm bietet einen abstrakten Überblick über Objekte. Im Vergleich dazu werden mit dem Objektdiagramm konkrete Anwendungen modelliert. Das Objektdiagramm bietet aus diesem Grund eine detaillierte Sicht auf ein Teilsystem.

4.3.1 Objektdiagramme und Enterprise Anwendungen

Objektdiagramme haben im Projekt breit gestreute Anwendungsmöglichkeiten. Die Anwendungsmöglichkeiten im Projekt sind unter anderem [Rupp05]:

- Objektdiagramme dienen zur *Illustration rekursiver Strukturen* im Klassendiagramm. Das Objektdiagramm zeigt die tatsächlichen Strukturen zwischen Instanzen gleicher Klassen. Diese rekursiven Strukturen sind im Klassendiagramm nicht ersichtlich.
- Objektdiagramme können zur *Überprüfung von Klassendiagrammen* eingesetzt werden. Wird ein konkreter Fall in Form eines Objektdiagramms modelliert, können auch in einer frühen Phase des Projekts Fehler im Klassendiagramm aufgedeckt werden.
- Zum *Finden von Klassen* durch die Modellierung von Objektdiagrammen. Wird zuerst das Objektdiagramm modelliert, so kann aus diesem ein mögliches Klassendiagramm abgeleitet werden.
- Für die *Dokumentation von Architekturen*, in denen Objekte durch abstrakte oder generische Fabriken erzeugt werden. Diese Objekte sind in einem Klassendiagramm nicht sichtbar und werden erst durch die Darstellung des konkreten Objekts sichtbar.
- Für die *Modellierung von Konfigurationen*, falls diese dem Modell entnommen werden. Auch Konfigurationsdateien können mit Hilfe des Objektdiagramms dargestellt werden. Dies macht vor allem dann Sinn, wenn Konfigurationsdateien aus dem Modell generiert werden sollen.
- Ein weiterer Anwendungsfall für Objektdiagramme ist die Modellierung von Testdaten. In diesem Fall werden konkrete Testdaten in Form eines Objektdiagramms modelliert.

Objektdiagramme bieten für Enterprise Anwendungen eine gute Übersicht. In Enterprise Anwendungen gibt es meistens eine Vielzahl von Interfacedefinitionen und Implementierungen, dadurch ist es durchaus sinnvoll, eine konkrete Abbildung des Klassendiagramms in Form eines Objektdiagramms zu modellieren.

Ein weiterer Anwendungsfall ist die Modellierung eines Objektdiagramms für Domainobjekte. Es stellt eine gute Überprüfung für das Klassendiagramms dar und der gleiche Arbeitsschritt ermöglicht die Modellierung von Testdaten.

4.3.2 Objektdiagramme im begleiteten Beispiel

Im folgenden Kapitel soll für das Klassendiagramm aus Kapitel 4.1.1 eine konkrete Instanz modelliert werden. Es könnte sich dabei um Testdaten des Domänenmodells handeln.

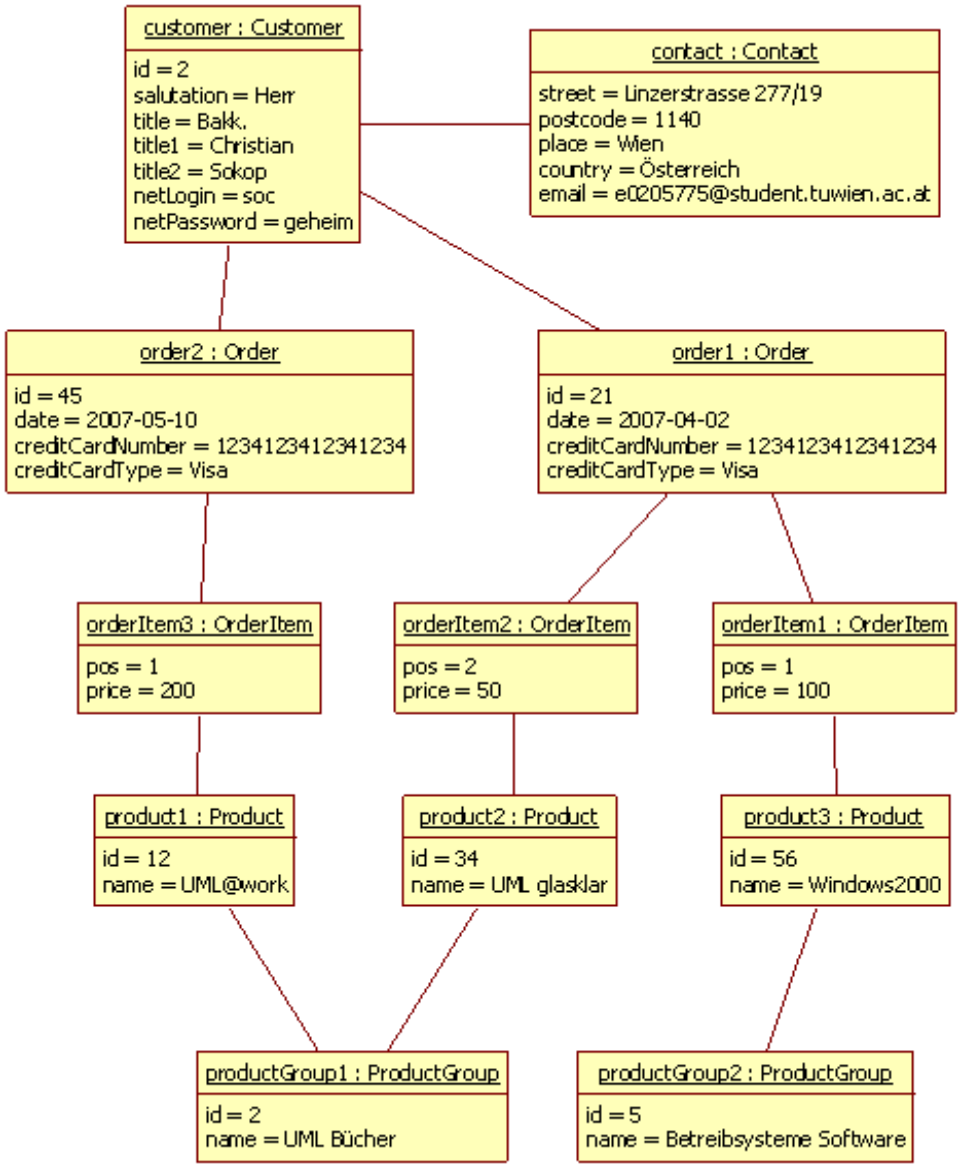

Abbildung 33: Objektdiagramm Testdaten Domänenmodellklassendiagramm

Anhand dieses Objektsdiagramms kann man gut erkennen, dass das vorher modellierte Klassendiagramm korrekt ist und in dieser Form angewendet werden kann.

Durch die Modellierung des konkreten Beispiels wurde auch klar, dass es wohl nicht vorkommt, dass ein Kunde zur gleichen Zeit mehrere Bestellungen hat. Die 1:n Beziehung macht allerdings sehr wohl bei der Administration von Bestellungen Sinn.

4.4 Kompositionsstrukturdiagramm

Mit dem Kompositionsstrukturdiagramm lassen sich interne Strukturen von Komponenten und deren Interaktionen zu anderen Komponenten darstellen. Es soll die Frage „*Wie sind die einzelnen Architekturkomponenten strukturiert und mit welchen Rollen spielen sie dabei zusammen?*“ beantworten [Rupp05].

Das Kompositionsstrukturdiagramm wurde in UML 2.0 neu eingeführt [Rupp05].

Prinzipiell kann zwischen zwei Arten von Kompositionsstrukturdiagrammen unterschieden werden [Hitz05]:

- *Kompositionsstrukturdiagramm für Klassen:* Das Ziel des Kompositionsstrukturdiagramms für Klassen ist es, eine detaillierte Beschreibung der internen Struktur einer Klasse zu modellieren. Der Vorteil gegenüber dem Klassendiagramm ist es, dass im Kompositionsstrukturdiagramm die Zusammenhänge der Klassen aus Sicht einer bestimmten Klasse (Kontextklasse) modelliert werden. Dadurch ist ein Kompositionsstrukturdiagramm in vielen Fällen übersichtlichter als das Klassendiagramm, hat aber die gleiche Aussage.
- *Kompositionsstrukturdiagramm für Kollaborationen:* Hier wird das Zusammenspiel der Komponenten modelliert. Das Hauptaugenmerk liegt dabei nicht auf das „wie und wann“, sondern wer mit welcher Komponente kommuniziert [Hitz05]. Diese Art des Kompositionsstrukturdiagramms eignet sich besonders gut, um Designpattern zu beschreiben, die in der Implementierung angewendet werden [Rupp05].

In den folgenden Kapiteln sollen für diese beiden Arten von Kompositionsstrukturdiagrammen Beispielen modelliert werden. Das Kompositionsstrukturdiagramm für

Kollaborationen wird in Zusammenhang mit Enterprise Anwendungen gezeigt, das Kompositionsstrukturdiagramm für Klassen anhand des konkreten Beispiels.

4.4.1 Kompositionsstrukturdiagramm in Enterprise Anwendungen

Kompositionsstrukturdiagramme für Kollaborationen bieten eine gute Möglichkeit um Designpattern zu beschreiben [Rupp05]. Die Beschreibung von Designpattern ist ein wichtiger Punkt in einer Enterprise Anwendung. In weiterer Folge können die modellierten Patterns in konkreten Beispielen angewendet werden. Kompositionsstrukturdiagramme für Klassen finden ihre Anwendung hingegen nur bei konkreten Beispielen, und werden im nächsten Kapitel beschrieben.

Kompositionsstrukturdiagramm für Kollaborationen

Das Kompositionsstrukturdiagramm für Kollaborationen ist für die Darstellung von Pattern sehr gut geeignet [Rupp05]. Die folgenden zwei Diagramme zeigen, die im Rahmen dieser Arbeit schon des Öfteren modellierte DAO- und MVC-Pattern.

Im Kompositionsstrukturdiagramm für Kollaboration werden die benötigten Elemente und die Zusammenhänge zwischen den Elementen modelliert. Das Kompositionsstrukturdiagramm soll in dieser Abstraktionsebene eine Übersicht über ein Pattern geben, und nicht den genauen Aufbau eines Patterns beschreiben. Für die Beschreibung des Aufbaus eines Patterns sind Klassen- bzw. Sequenzdiagramme besser geeignet.

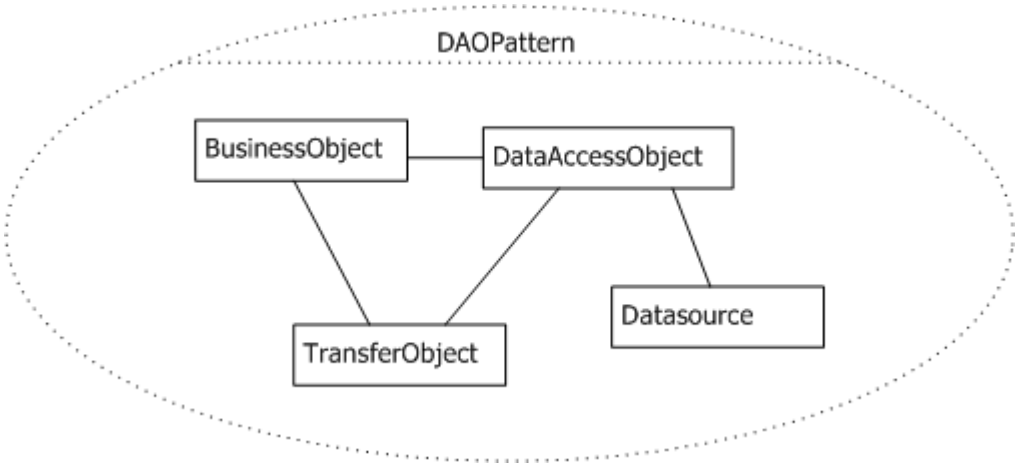

Abbildung 34: Kompositionsstrukturdiagramm DAO-Pattern

Im Falle des DAO-Patterns werden das Businessobjekt, das Datenzugriffsobjekt, das Transferobjekt und die Datenquelle modelliert. Diese Modellierung des Patterns erlaubt in weiterer Folge die Modellierung eines konkreten Anwendungsbeispiels, wie es das nächste Kapitel demonstrieren soll.

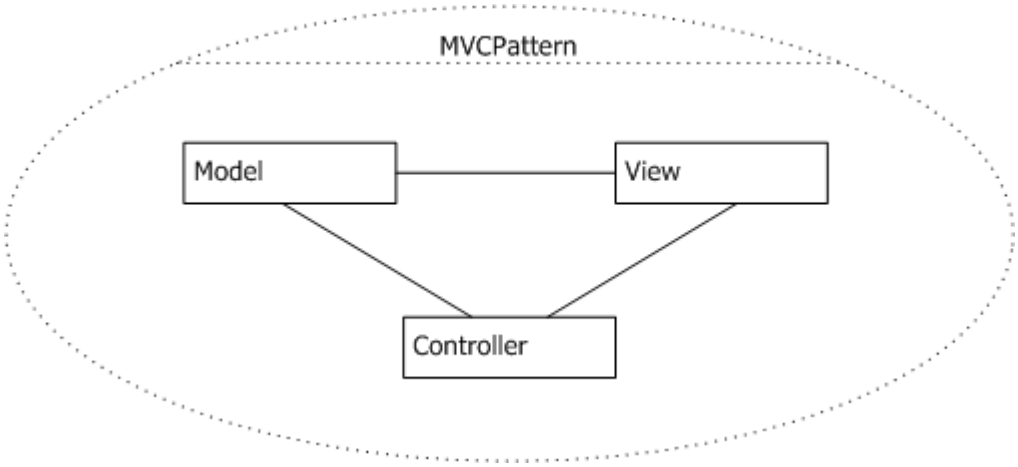

Abbildung 35: Kompositionsstrukturdiagramm MVC-Pattern [Rupp05]

Das Kompositionsstrukturdiagramm des MVC-Patterns sagt aus, dass eine Model-, eine Controller- und eine Viewkomponente benötigt werden, um dieses Pattern zu realisieren.

Da das Kompositionsstrukturdiagramm in UML 2.0 neu definiert wurde, wird es von StarUML noch nicht ausreichend unterstützt [Reic06].

Die beiden modellierten Diagramme werden im nächsten Kapitel eine konkrete Anwendung finden.

4.4.2 Kompositionsstrukturdiagramme im begleiteten Beispiel

In einem konkreten Beispiel eignet sich das Kompositionsstrukturdiagramm sowohl für eine übersichtliche Darstellung von Klassen, wie auch für die Anwendung von Designpatterns.

Kompositionsstrukturdiagramm für Kollaborationen

In diesem Kapitel werden die beiden oben definierten Designpattern in einer konkreten Anwendung verwendet. Dazu wird das DAO-Pattern im Kundenservice und der

Anwendungsfall „Kunde anlegen“ als Anwendung für das MVC-Pattern modelliert.

Wie in der oben gezeigten Modellierung des DAO-Patterns werden ein Businessobjekt, ein Transferobjekt, ein Datenzugriffsobjekt und eine Datenquelle benötigt. Die Datenquelle wird in diesem konkreten Beispiel nicht modelliert.

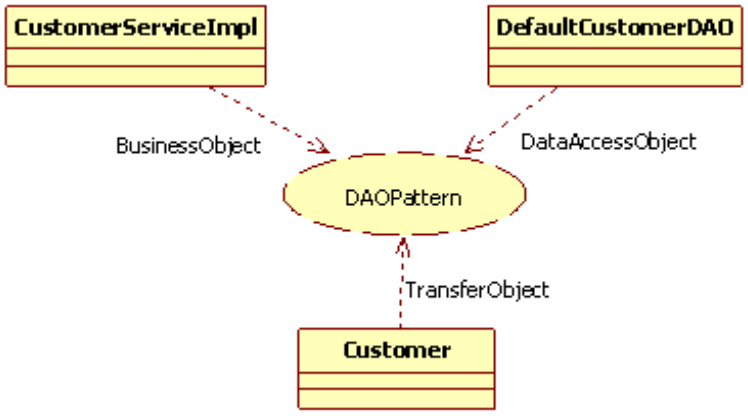

Abbildung 36: Kompositionsstrukturdiagramm Anwendung DAO-Pattern

Dieses Kompositionsstrukturdiagramm zeigt die Anwendung des DAO-Patterns im begleiteten Beispiel. Das Businessobjekt wird in Form der Klasse „CustomerServiceImpl“ dargestellt. Als Datenzugriffsobjekt wird die Klasse „DefaultCustomerDAO“ verwendet und als Transferobjekt die Klasse „Customer“. Dieses Diagramm soll verdeutlichen, dass hier das DAO-Pattern verwendet wird.

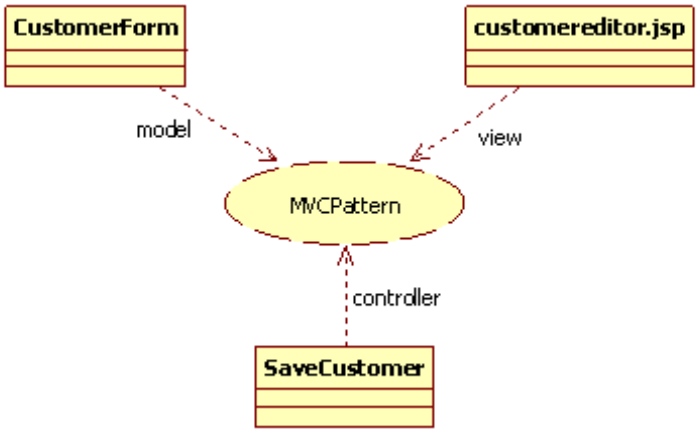

Abbildung 37: Kompositionsstrukturdiagramm Anwendung MVC-Pattern

Dieses Kompositionsstrukturdiagramm zeigt eine Anwendung des MVC-Patterns am Beispiel „Kunde anlegen“ der konkreten Implementierung. Als Model wird die Klasse „CustomerForm“ verwendet. Der Controller ist in der Klasse „SaveCustomer“ implementiert, und die View wird mit Hilfe der JSP-Seite „customereditor.jsp“ angezeigt. Dieses Beispiel soll verdeutlichen, dass diese drei Komponenten im begleitenden Beispiel das MVC-Pattern verwenden.

Kompositionsstrukturdiagramm für Klassen

Mit Hilfe des Kompositionsstrukturdiagramms kann auch eine Art Klassendiagramm modelliert werden. Dazu wird eine Kontextklasse bestimmt und ausgehend von dieser Klasse werden die anderen Klassen und deren Zusammenhänge modelliert. In diesem Kapitel wird das Klassendiagramm aus Sicht des Kunden modelliert.

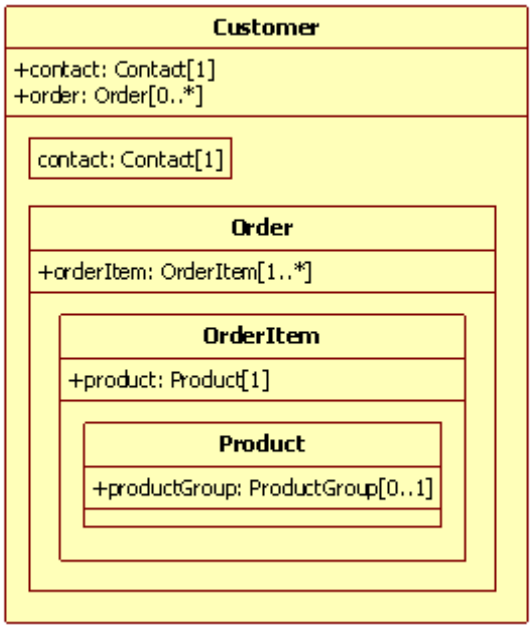

Abbildung 38: Kompositionsstrukturdiagramm Kontextklasse „Customer“

Dieses Kompositionsstrukturdiagramm gibt eine Übersicht über die einzelnen Klassen im Klassendiagramm aus Sicht der Klasse Kunde („Customer“). Ein Kunde hat einen Kontakt („Contact“) und kann mehrere Bestellungen („Order“) haben. Eine Bestellung wiederum besteht aus mehreren Teilen („OrderItem“). Diese werden genau einem Produkt („Product“) zugewiesen, welches einer Produktgruppe („Pro-

duktGroup“) zugeordnet sein kann.

In diesem Diagramm kommt die Navigation durch die Klassen gut zum Vorschein. Üblicherweise werden, wie hier gezeigt, weitere Attribute der Klassen nicht modelliert, sondern nur die Attribute, die eine Verbindung zu einer anderen Klasse definieren.

Diese Diagrammart gibt einen guten Überblick über das Zusammenspiel der Klassen. Im Vergleich zum Klassendiagramm lassen sich mit Hilfe des Kompositionsstrukturdiagramms die Zusammenhänge der Klassen übersichtlicher modellieren, da das gesamte Diagramm von einer Kontextklasse aus betrachtet wird.

4.5 Komponentendiagramm

Das Komponentendiagramm bietet die Möglichkeit, die Struktur eines Systems zur Laufzeit darzustellen. Es soll die Frage „*Wie ist mein System strukturiert und wie werden diese Strukturen erzeugt?*“ beantworten [Rupp05].

Der Begriff Komponente ist in UML 2.0 „*ein modularer Teil eines Systems, der zur Abstraktion und Kapselung einer beliebig komplexen Struktur dient, die nach außen wohldefinierte Schnittstellen zur Verfügung stellt*“ [Hitz05].

4.5.1 Komponentendiagramm in Java Enterprise Anwendungen

Mit Hilfe des Komponentendiagramms lassen sich „*sowohl physische als auch logische Modellierungsaspekte abdecken*“ [Hitz05]. Aus diesem Grund sind Komponentendiagramme ein wichtiges Modellierungswerkzeug für verteilte Systeme, wie es auch Enterprise Anwendungen sind. Das Komponentendiagramm kann auch das Zusammenspiel zwischen physischen und logischen Komponenten darstellen.

Das Komponentendiagramm stellt eine gute Möglichkeit dar, die einzelnen Dokumente und deren Zusammenspiel zu beschreiben. Vor allem bei der Verwendung von Java Enterprise Frameworks ist der richtige Einsatz von Konfigurationsdateien und deren Zusammenspiel von großer Bedeutung.

In einer Enterprise Anwendung werden sehr viele Interfaces und Implementierungen verwendet. Das Komponentendiagramm bietet eine gute Möglichkeit, um diese Interfaces darzustellen.

Das Komponentendiagramm erlaubt *„sowohl die interne Struktur einer Komponente abzubilden als auch die verschiedenen Wechselbeziehungen zwischen Komponenten*“ [Rupp05].

Eine Java Enterprise Anwendung besteht aus mehreren Komponenten, die im folgenden Diagramm dargestellt werden:

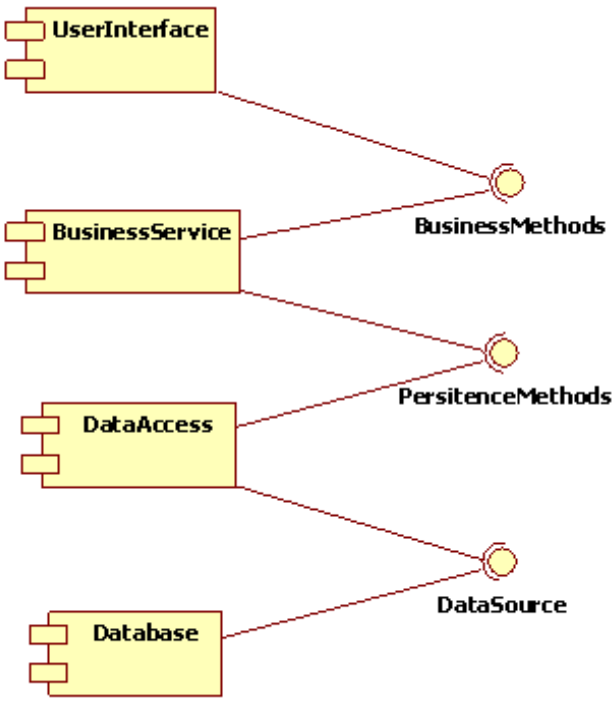

Abbildung 39: Komponentendiagramm Java Enterprise Anwendung

Das Komponentendiagramm zeigt einen üblichen Aufbau einer Java Enterprise Anwendung. Die Datenbank stellt eine Datenquelle zur Verfügung, welche von den Datenzugriffsobjekten verwendet wird. Diese stellen wieder ein Interface zur Verfügung (Persistenzmethoden), welches vom Businessservice verwendet wird. Die vom Businessservice zur Verfügung gestellten Businessmethoden werden vom Userinterface aufgerufen. Dieses Komponentendiagramm zeigt eine Übersicht über eine Java Enterprise Anwendung. Es kann sich dabei sowohl um physische, wie auch um logische Komponenten handeln.

Weiters bieten Komponentendiagramme die Möglichkeit Artefakte und deren Inhalte darzustellen. Das folgende Diagramm soll das Zusammenspiel der Komponenten einer Java Enterprise Anwendung darstellen.

Mit Komponentendiagrammen kann eine exakte Deploymentumgebung modelliert werden. Das ist bei Java Enterprise Web Anwendungen relativ schwierig, da es, selbst bei sehr kleinen Projekten, eine genaue Anordnung der Komponenten vorsieht.

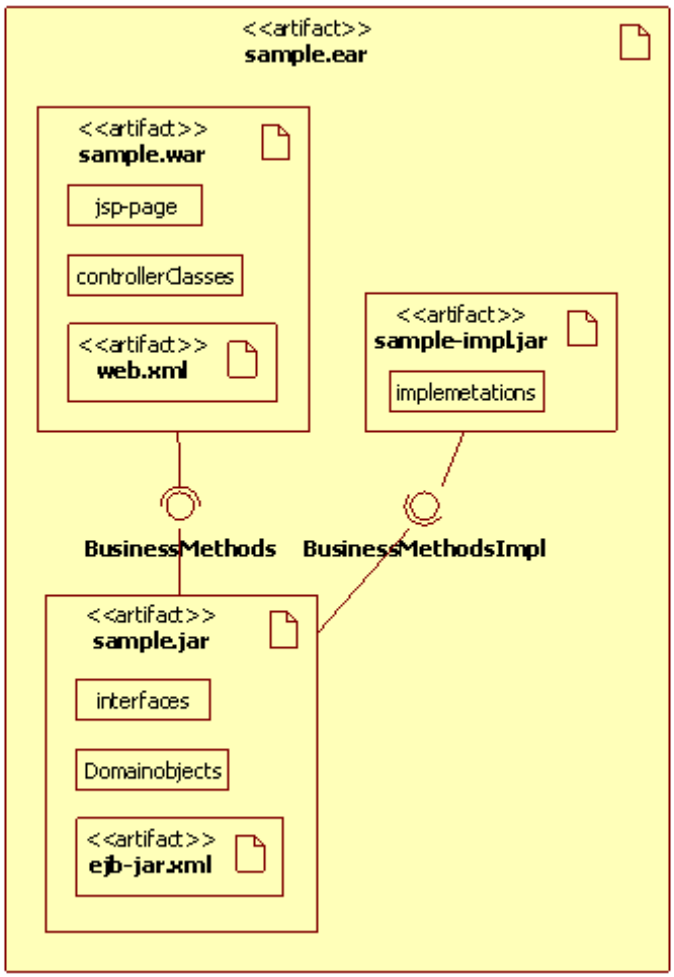

Abbildung 40: Komponentendiagramm Java EE Web Anwendung

Das Komponentendiagramm der Java EE Web Anwendung zeigt den Aufbau der „ear“ Datei. Eine „ear“ (Enterprise Archive) Datei enthält alle Komponenten, die in einer Java Enterprise Anwendung vorhanden sind. Das sind insbesondere eine „war“ Datei, die Interfaces der Businessmethoden (Enterprise Java Beans) und die Implementierung der Business Methoden. Die „war“ (Web Archive) Datei enthält alle Komponenten, die für die Realisierung des Webteils benötigt werden. Die Interfaces enthalten neben den Definitionen der Businessinterfaces selbst, die gemeinsam verwendeten Domainobjekte und einen Deploymentdeskriptor („*ejb-jar.xml*“).

Die beiden modellierten Diagramme zeigen einen allgemeinen Überblick über eine Enterprise Architektur.

4.5.2 Komponentendiagramm im begleiteten Beispiel

Mittels Komponentendiagrammen ist es auch möglich konkrete Implementierungen zu modellieren. Das folgende Kapitel soll zunächst die Komponente „CustomerService“ im Detail modellieren. Dabei soll gezeigt werden, dass dieses Service nach innen abgeschlossen ist, und verschiedene Schnittstellen zur Verfügung stellt.

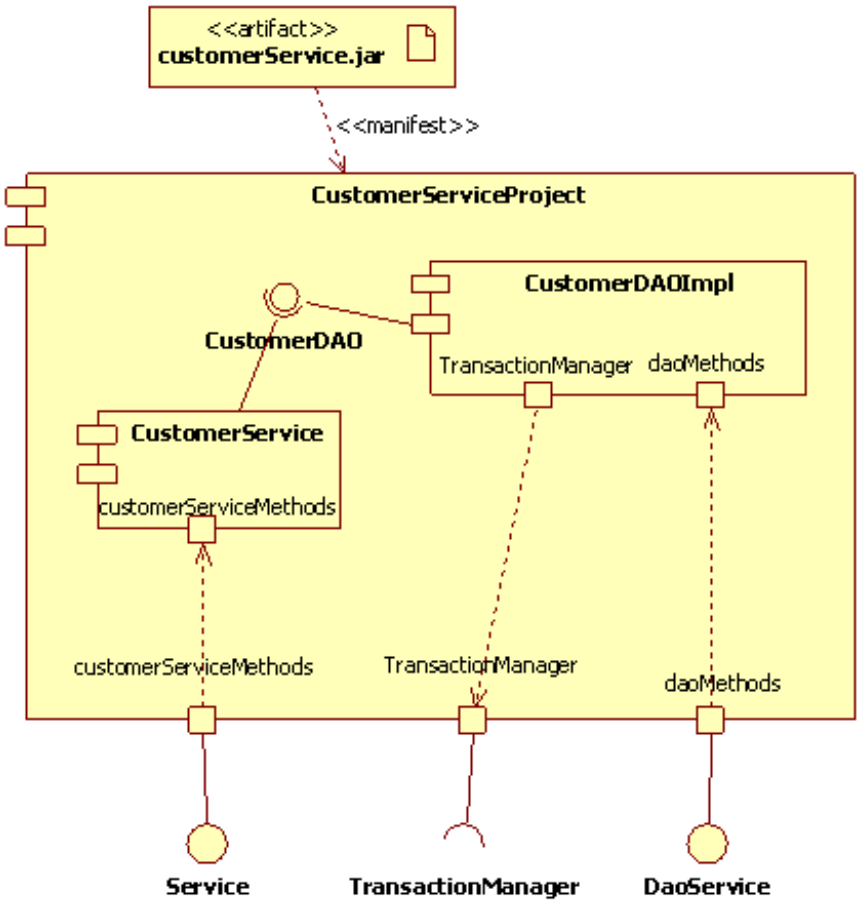

Abbildung 41: Komponentendiagramm „CustomerService“

Das Komponentendiagramm „CustomerService“ zeigt den internen Aufbau der Komponenten und die Schnittstellen nach außen. Das Service bietet zwei Schnittstellen an. Eine Schnittstelle zum Datenzugriffsservice und eine zum Businessservice. Weiters wird ein Transaktionsmanager benötigt um Daten zu lesen. Im internen Zustand der Komponente verwendet die Komponente Kundenservice das Daten-

zugriffsobjekt.

Das nächste Diagramm zeigt die implementierte Testkomponente, welche einen Transaktionsmanager definiert und auf die Schnittstellen der Komponente „CustomerService“ zugreift.

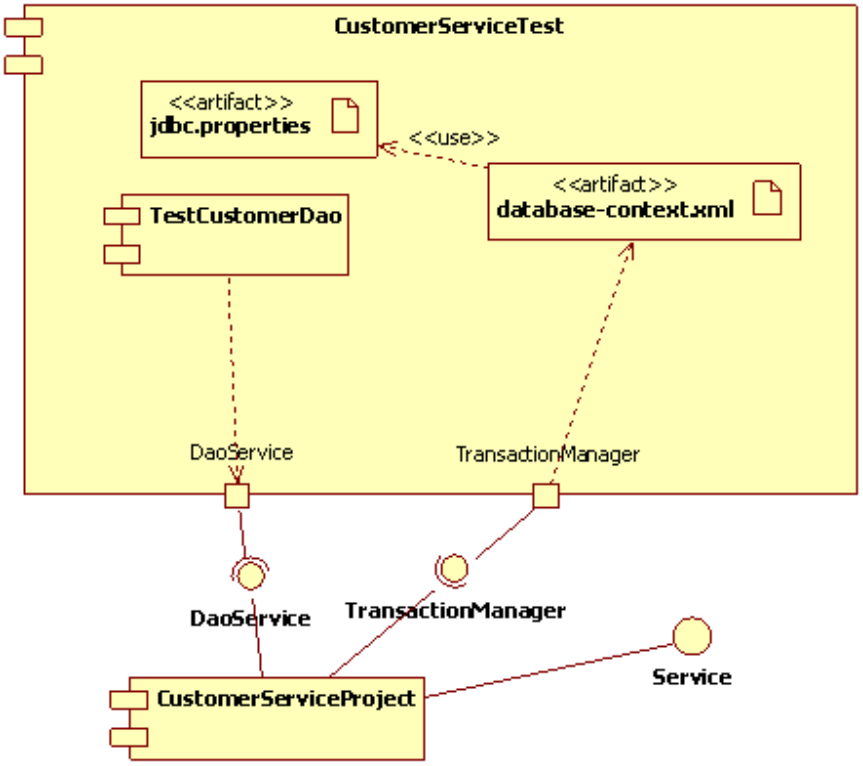

Abbildung 42: Komponentendiagramm Testkomponente „CustomerServiceTest“

Das Komponentendiagramm der Testkomponente zeigt eine in sich abgeschlossene Komponente, die einen Transaktionsmanager anbietet und ein Datenzugriffsobjekt benötigt. Die Testkomponente definiert den Transaktionsmanager mit Hilfe von zwei Konfigurationsdateien. Die Komponente, die das Datenzugriffsobjekt testen soll („TestCustomerDao“), braucht die Komponente, die sie testen soll. Das Gegenstück zur Testkomponente ist das vorhin modellierte „CustomerServiceProject“

Das nächste Diagramm zeigt die View-Komponente für den Anwendungsfall „Kunde verwalten“. Auch diese Komponente bietet einen Transaktionsmanager an, und braucht die Datenzugriffskomponente. Die View-Komponente definiert den Transaktionsmanager mit Hilfe von zwei Konfigurationsdateien. Der Controller benötigt die Datenzugriffskomponente. Weitere Teilkomponenten der Komponente „Custome-

rAdministrationView“ sind „View“, „Model“ und „Controller“. Das Zusammenspiel zwischen diesen drei Teilkomponenten wird mit Hilfe einer XML-Konfigurationsdatei beschrieben.

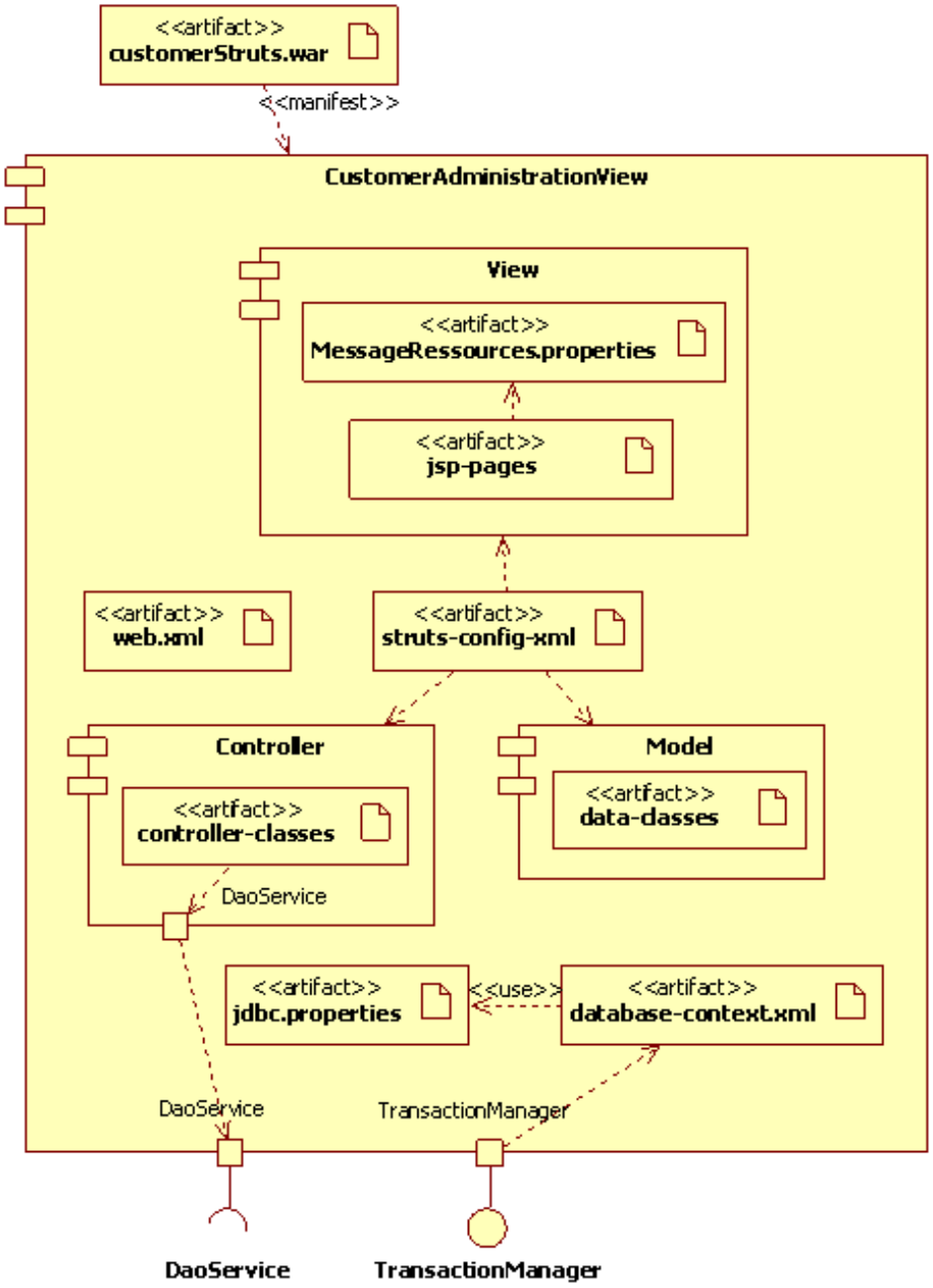

Abbildung 43: Komponentendiagramm „Kunde verwalten“ - View

Das nächste Diagramm soll die beiden Diagramme aus Abbildung 41 und Abbildung 43 vereinen, und das Zusammenspiel zwischen der View-Komponente und „Custo-

merServiceProject“ - Komponente verdeutlichen.

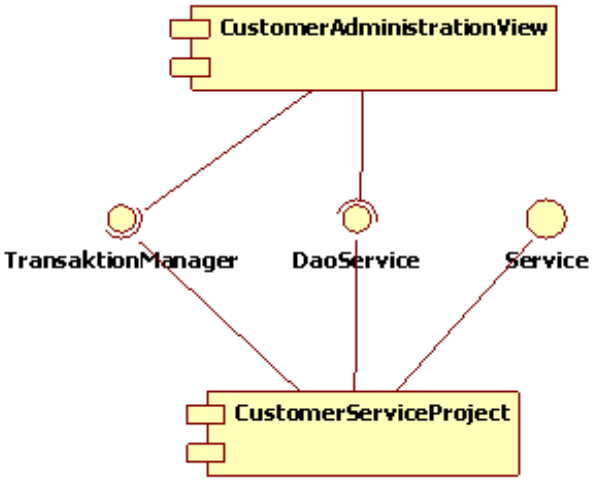

Abbildung 44: Komponentendiagramm View/Service

Dieses Komponentendiagramm zeigt das Zusammenspiel der View- und Service-Komponente. Die View-Komponente definiert einen Transaktionsmanager und braucht die Datenzugriffskomponente. Die Servicekomponente braucht einen Transaktionsmanager und stellt eine passende Datenzugriffskomponente zur Verfügung.

Anhand dieses Beispiels sieht man wie übersichtlich und aussagekräftig Komponentendiagramme in Verbindung mit Java Enterprise Anwendungen modelliert werden können. Komponentendiagramme geben je nach Abstraktionsebene eine guten Überblick oder eine detaillierte Darstellung der einzelnen Artefakte und Komponenten.

4.6 Verteilungsdiagramm

Ein Verteilungsdiagramm beschreibt die Zuordnung von Artefakten auf Hardwareeinheiten. Mit Hilfe dieses Diagramms soll die Frage „*Wie werden die Artefakte des Systems zur Laufzeit wohin verteilt?*“ beantwortet werden [Rupp05].

4.6.1 Verteilungsdiagramm in Java Enterprise Anwendungen

Die Modellierung von Verteilungsdiagrammen eignet sich nur, wenn die Anwendung auf mehreren Hardwarekomponenten (Knoten) läuft [Rupp05]. Bei Enterprise Anwendungen ist die Entwicklung von Verteilungsdiagrammen daher sehr sinnvoll, da Enterprise Anwendungen zumindest theoretisch auf mehrere Knoten verteilt werden.

Verteilungsdiagramme enthalten neben den „normalen“ Knoten auch Ausführungsumgebungsknoten. Diese Ausführungsumgebungen können z.B. ein Betriebssystem oder ein Webbrowser sein [Hitz05]. Zusätzlich werden für die Modellierung eines Verteilungsdiagramms unter anderem Artefakte und Deployment Spezifications angeboten [Hitz05]. Eine allgemeine Darstellung einer Java Enterprise Webanwendung ist in folgender Abbildung zu sehen:

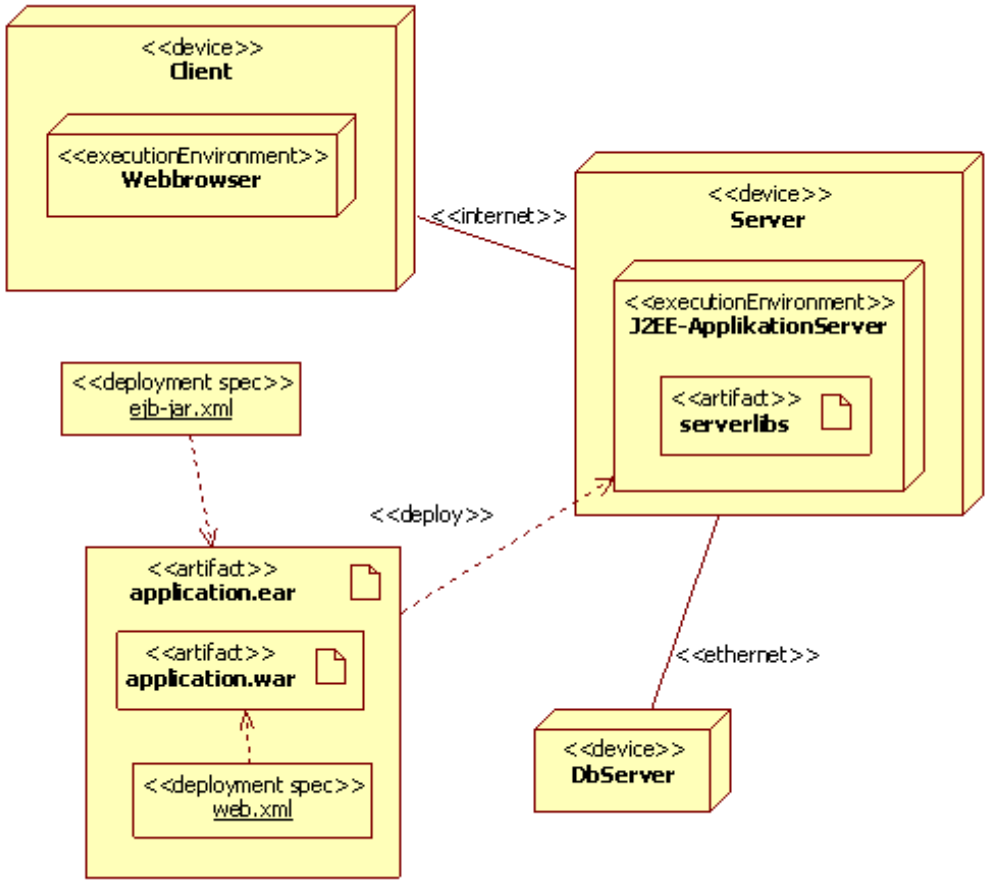

Abbildung 45: Verteilungsdiagramm Java EE Webanwendung

Eine Java EE Webanwendung besteht üblicherweise aus drei verschiedenen physischen Konten. Der Benutzer greift mit seinem Client, auf dem ein Webbrowser läuft, auf den Anwendungsserver zu. In dieser Serverumgebung ist die Anwendung installiert. Der Anwendungsserver greift auf einen externen Datenbankserver zu.

Dieses Diagramm gibt eine gute Übersicht über die verschiedenen Artefakte und deren Verteilung auf physischen Knoten. Es soll nicht unerwähnt bleiben, dass bei kleinen Projekten der DB-Server und der Anwendungsserver auf demselben physi-

schen Knoten liegen.

4.6.2 Verteilungsdiagramm im begleiteten Beispiel

Im begleiteten Beispiel trifft der oben beschriebene Fall zu. Der Anwendungsserver ist auf demselben Knoten installiert, wie die Datenbank.

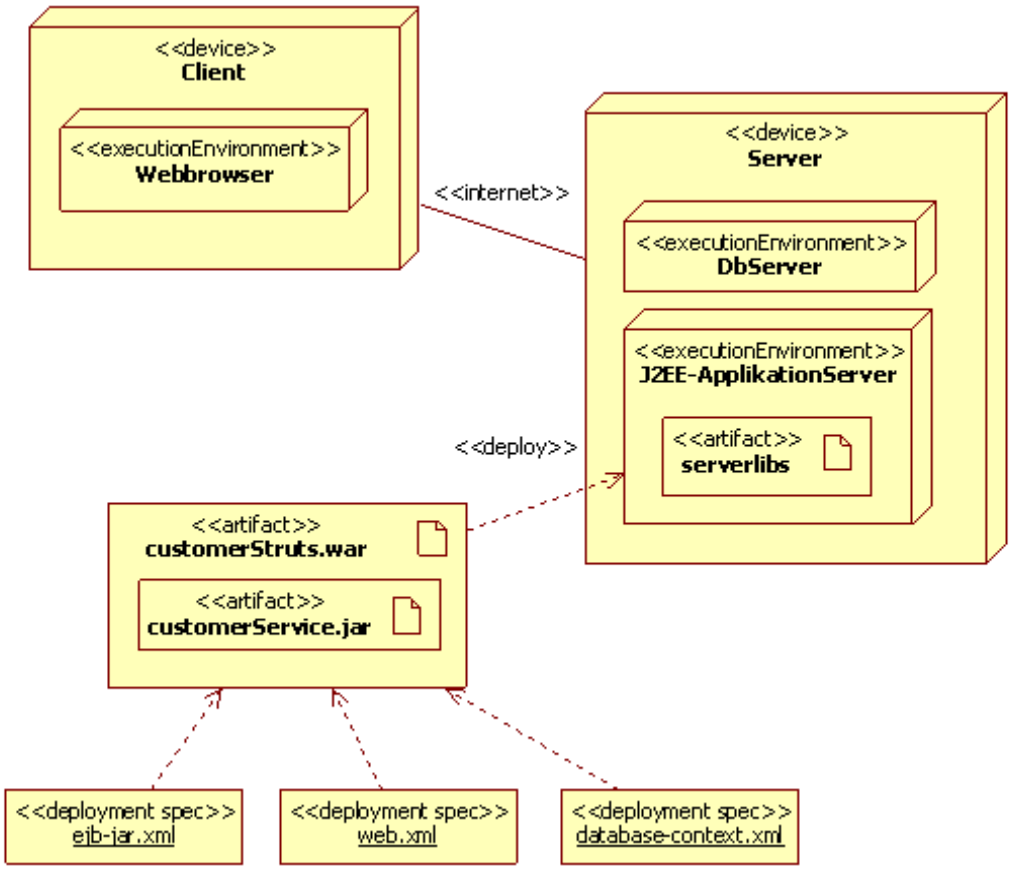

Abbildung 46: Verteilungsdiagramm „CustomerStruts“

Im begleiteten Beispiel ist der Aufbau sehr ähnlich. Der Client greift mir seinen Webbrowser auf den Anwendungsserver zu. Auf diesem physischen Server befindet sich neben dem Anwendungsserver auch die Datenbank. Das installierte Paket bedient sich an mehreren Deploymentdeskriptoren, welche vom Framework „Spring“ kommen.

Auch das spezifische Diagramm gibt einen guten Einblick über das Deployment der Anwendung.

Kapitel 5

Rational Unified Process

Das folgende Kapitel soll die in Kapitel 3 und 4 gezeigten Modelle im Rahmen eines Softwareentwicklungsprozesses, dem Rational Unified Process, beschreiben und bewerten. Ziel dieses Kapitels ist es einen Überblick über die Diagramme in UML 2.0 in Bezug auf deren Einsatz in einem Projekt, welches die Enterprise Architektur verwendet, zu bekommen.

Beim Unified Process handelt es sich um ein objektorientiertes Vorgehensmodell für die Softwareentwicklung [Zuse01], der 1999 veröffentlich wurde (vgl. [Jaco99]). Er wurde von der Firma Rational entwickelt, die ein Teil des IBM Konzerns ist.

Eigenschaften des Unified Processes [Zuse01]:

- Der Unified Process wird durch *Anwendungsfälle gesteuert*: Die Anforderungen an ein System müssen eindeutig beschrieben werden, bevor mit dem Entwurf und der Implementierung begonnen werden kann. Alle weiteren Arbeitschritte werden auf den Anwendungsfällen und deren Beschreibungen aufgebaut.
- Der Unified Process ist ein *iterativer und inkrementeller* Prozess: Die einzelnen Arbeitsschritte müssen im Entwicklungsprozess öfters durchlaufen werden. Bei jeder Iteration werden Umfang und Qualität des Softwareprodukts verbessert. Das gesamte System und deren Dokumentation wachsen dadurch inkrementell. Nach genügend Durchläufen wird das Projekt an den Kunden

ausgeliefert.

- Der Unified Process ist *architekturzentriert*: Die Beschreibung der verwendeten Architektur stellt einen wichtigen Punkt im Entwicklungsprozess dar. Die Architektur soll im Zusammenspiel mit den wichtigsten Anwendungsfällen gefunden werden. Im Rahmen dieser Arbeit ist die Architektur vorgegeben, sie sollte allerdings auch im Rahmen des Unified Processes beschrieben werden.

Im Unified Process werden fünf grundlegende Arbeitsschritte beschrieben, die für die Erstellung einer Software notwendig sind [Zuse01]. Im Rahmen des Unified Process wird jeder Arbeitsschritt in einer Iteration einmal durchlaufen. Als Ergebnis entstehen verschiedene Produkte, die für die folgenden Arbeitsschritte verwendet werden [Zuse01].

Zusätzlich wird ein Projekt im Unified Process in Phasen eingeteilt. Diese Phasen sind Beginn, Ausarbeitung, Konstruktion und Umsetzung. Jeder Arbeitschritt wird in jeder Phase entsprechend angewendet. Im Rahmen dieser Arbeit sollen UML 2.0 Diagramme im Zusammenhang mit den Arbeitsschritten gefunden werden. Die Projektphasen werden hier nur am Rande erwähnt.

Die Arbeitschritte im Unified Process sind [Zuse01]:

- *Anforderungen*: In diesem Arbeitschritt sollen Anforderungen des Projekts gefunden werden. Als Anforderung versteht man „*eine Bedingung oder eine Fähigkeit funktionaler oder nicht funktionaler Natur, welche ein Produkt erfüllen bzw. haben muss*“ [Zuse01]. Dieser erste Arbeitsschritt stellt einen Einstieg in die Modellierung eines Systems dar. In diesem Arbeitsschritt werden keine UML-Diagramme verwendet, deshalb wird nicht näher auf diesen Schritt eingegangen.
- *Analyse*: In diesem Arbeitsschritt werden die gefundenen Anforderungen gegliedert. Das Ergebnis dieser Phase ist das Analysedokument.
- *Entwurf*: In der Entwurfphase, wird basierend auf der Spezifikation, das System technologieabhängig modelliert.

- *Implementierung*: In der Implementierungsphase wird der Entwurf in eine Programmiersprache umgesetzt.
- *Test*: In diesem Arbeitsschritt wird das System systematisch gestestet. Wesentliche Bestandteile dieses Schrittes sind die Erstellung eines Testplans und die Erstellung von Testfällen.
- *Inbetriebnahme und Wartung*: In diesem Schritt handelt es sich um die Abnahme des Kunden und die Wartung des laufenden Systems. Dieser Arbeitsschritt wird von UML 2.0 nicht unterstützt. Die in UML modellierten Diagramme dienen als Referenz für diesen Arbeitsschritt.

Die folgenden Kapiteln sollen die Arbeitsschritte des Unified Processes kurz beschreiben und UML 2.0 Diagrammarten zugeordnet werden. Dabei wird Bezug auf Enterprise Anwendungen genommen.

Einen Überblick über die Einsetzbarkeit der Diagramme soll folgende Abbildung geben:

	Analyse	Entwurf	Implementierung	Test
Anwendungsfalldiagramm	1			
Aktivitätsdiagramm	1	2	2	
Zustandsdiagramm	1	2	2	
Sequenzdiagramm	2	1		1
Kommunikationsdiagramm	2	2		
Zeitdiagramm		3		
Interaktionsübersichtsdiagramm	3	3		
Klassendiagramm	1	1	1	
Paketdiagramm	2	1		
Objektdiagramm	1			1
Kompositionsstrukturdiagramm	2	1		
Komponentendiagramm	2	1		
Verteilungsdiagramm		1		

Abbildung 47: Übersicht UML 2.0 Diagramme im Unified Process

In dieser Übersicht wird zwischen „häufig eingesetzt“ (1), „benutzbar“ (2) und „wenige Einsatzmöglichkeiten“ (3) unterschieden. Ein frei gelassenes Feld deutet darauf hin, dass die entsprechende Diagrammart für diesen Arbeitsschritt nicht verwendet werden kann, zumindest im Zusammenhang mit einer Enterprise Anwendung.

5.1 Analyse

Der Arbeitsschritt Analyse wird in zwei Teilarbeitsschritte gegliedert [Zuse01]:

- *Anforderungsanalyse*: In der Anforderungsanalyse sollen funktionale und nicht funktionale Anforderungen gefunden und beschrieben werden.
- *Analysemodell*: Das Analysemodell soll die Anforderungsanalyse auf Machbarkeit und Vollständigkeit überprüfen.

Anforderungsanalyse

Die Anforderungsanalyse soll aus Sicht der Benutzer beschrieben werden und keine technischen Details enthalten [Zuse01]. Die Produkte der Anforderungsanalyse sind die Systembeschreibung, ein Begriffsverzeichnis, die Aktorenliste, das Anwendungsfalldiagramm, die Anwendungsfallbeschreibungen, ein Prototyp und das Domänenmodell [Zuse01]. Bei den drei erstgenannten handelt es sich um textuelle Beschreibungen. Das Anwendungsfalldiagramm ist ein eigenes Modell in UML 2.0.

Die Beschreibung von Anwendungsfällen erfolgt meist in Textform, kann allerdings, zumindest teilweise, auch mit Hilfe von UML 2.0 Diagrammen gemacht werden. Mit Hilfe des Aktivitätsdiagramms lässt sich das Zusammenspiel zwischen Benutzer und System modellieren. Mittels Zustandsdiagrammen können Anwendungsfälle, die die Präsentationsschicht betreffen, modelliert werden. Weitere Möglichkeiten Teile von Anwendungsfällen zu beschreiben bieten das Sequenz- und Kommunikationsdiagramm, wenn die zeitliche Abfolge im Anwendungsfall von Bedeutung ist.

Der Prototyp soll Teile der Anwenderschnittstellen implementieren [Zuse01]. Für diesen Teilarbeitsschritt gibt es keine Unterstützung von UML 2.0.

Ein Domänenmodell enthält die wichtigsten Objekte der Anwendungsdomäne, deren Attribute und Beziehungen zueinander. Zur Darstellung dieses Modells wird ein UML Klassendiagramm verwendet [Zuse01]. Eine weitere Möglichkeit für die Darstellung eines Domänenmodells ist das Kompositionsstrukturdiagramm. Der Vorteil gegenüber dem Klassendiagramm besteht darin, dass eine bestimmte Klasse im Mittelpunkt steht (Kontextklasse). Ausgehend von dieser Kontextklasse werden die anderen Klassen modelliert. Dadurch entsteht ein übersichtliches Bild des Domänen-

modells, in dem allerdings nur Assoziationen zwischen Klassen sichtbar sind.

Um Fehler im Domänenmodell schon in einer frühen Phase des Projekts zu vermeiden, kann ein Objektmodell zur Überprüfung modelliert werden. Das Objektmodell kann auch vor dem eigentlichen Domänenmodell modelliert werden, um Domänen und deren Zusammenhänge leichter zu finden.

Analysemodell

Während für die Anforderungsanalyse Kundensprache verwendet wird, wird für das Analysemodell die Sprache des Programmierers verwendet [Zuse01]. Im Analysemodell wird die interne Sicht auf das System und Schnittstellen nach außen gegeben.

Das Ergebnis des Analysemodells soll ein Analysemodelldiagramm, eine Beschreibung der Anwender- und Systemschnittstellen, eine Beschreibung der Geschäftslogik und eine Beschreibung von Entitäten sein [Zuse01]. Die Darstellung von Anwenderschnittstellen und die Beschreibung von Entitäten werden in UML 2.0 nicht unterstützt.

Das Analysemodelldiagramm wird mit Hilfe eines Klassendiagramms modelliert. Dabei werden neben den Domänenklassen, auch jene Klassen, die für die funktionale Umsetzung des Systems von Bedeutung sind, modelliert. In diesem Arbeitsschritt werden die modellierten Klassen mit Hilfe von Paketdiagrammen gegliedert.

Die Beschreibung der Systemschnittstellen erfolgt nach [Zuse01] in Textform. Dabei wird unter anderem eine Liste von Methoden definiert, die das System als Schnittstelle zur Verfügung stellen soll. Mittels Komponentendiagrammen können in UML 2.0 die angebotenen und benötigten Schnittstellen graphisch dargestellt werden.

Für die Beschreibung der Geschäftlogik stehen in UML 2.0 einige Diagrammarten zur Verfügung. Für die Beschreibung von statischen Eigenschaften können Klassen-, Paket- und Objektdiagramme verwendet werden. Dynamische Eigenschaften können mit Hilfe von Sequenz-, Kommunikations- oder Zustandsdiagrammen modelliert werden [Zuse01].

Zusammengefasst kann gesagt werden, dass im Arbeitsschritt Analyse des Unified Process beinahe jede Art von UML 2.0 Diagrammen angewendet werden können.

Ausnahmen sind das Zeitdiagramm und das Verteilungsdiagramm. Während das Verteilungsdiagramm in einer späteren Phase angewendet werden kann, kommt das Zeitdiagramm bei der Modellierung von Enterprise Anwendungen nicht zum Einsatz.

5.2 Entwurf

Im Gegensatz zur Analyse wird beim Entwurf der Aufbau des Systems weniger abstrakt, dafür mehr technologieabhängig modelliert [Zuse01]. Aufbauend von den Produkten der Analysephase, wird in der Entwurfphase eine exakte technische Beschreibung des Systems gemacht.

Folgende Tätigkeiten werden während der Entwurfphase durchlaufen [Zuse01]: Architektur beschreiben, Geschäftlogik entwerfen, Subsysteme entwerfen, Anwenderschnittstelle gestalten und Datenbasis entwerfen. Für die Gestaltung von Anwenderschnittstellen werden in UML 2.0 keine Diagramme angeboten.

Architekturbeschreibung

Die Architekturbeschreibung kann man in die Beschreibung von Systemarchitektur, in Bezug auf das System selbst und in Bezug auf verwendete Technologien aufteilen. Bei der Verwendung von Patterns wird zwischen Architektur Patterns (vgl. [Bush96]) und Design Patterns (vgl. [Gamm95]) unterschieden.

Diese Arbeit bezieht sich auf Enterprise Anwendungen. In den vorigen Kapiteln werden viele Möglichkeiten vorgestellt, wie diese Systemarchitektur mit Hilfe von UML 2.0 modelliert werden kann. Der allgemeine Aufbau dieser Architektur kann mit einer Vielzahl von Diagrammen in UML 2.0 dargestellt werden. Das sind insbesondere Komponenten-, Kommunikations-, Paket- und Sequenzdiagramme.

Für die Darstellung von Designpatterns eignet sich am besten ein Klassendiagramm in Verbindung mit einem Sequenzdiagramm, wie es auch öfters in der Literatur verwendet wird. Kompositionsstrukturdiagramme eignen sich gut um die Verwendung eines Patterns zu verdeutlichen.

Komponentendiagramme eignen sich in dieser Phase des Projektes um eine detaillierte Struktur des Projektes zu modellieren. Der Vorteil des Komponentendiagramms ist, dass jegliche Form von Artifakten, wie z.B. Konfigurationsdateien in die

Modellierung eingebunden werden können. Dadurch kann mit Komponentendiagrammen die exakte Deploymentumgebung modelliert werden. Das ist bei Enterprise Anwendungen relativ schwierig, da es sehr viele verschiedene Artifakte, wie z.B. war-, jar-, ear-Dateien gibt. Außerdem können mittels Komponentendiagrammen, gewisse Konfigurationsdateien (Deploymentdiskreptoren), die an einer bestimmten Stelle im Archiv liegen müssen, damit diese vom Container erkannt werden, modelliert werden.

Die Systemarchitektur kann am besten mit Hilfe des Klassen- und Paketdiagramms modelliert werden. Besonders zu beachten ist die Möglichkeit das System in verschiedenen Abstraktionsebenen zu modellieren. Ein Klassen- oder Paketdiagramm kann sowohl einen groben Überblick über das System geben, wie auch eine detaillierte Beschreibung des Systems.

Eine weitere Möglichkeit für die Beschreibung der Systemarchitektur bietet das Verteilungsdiagramm. Mit Hilfe dieser Diagrammart wird festgelegt auf welchen physischen Einheiten die verschiedenen Systeme und Teilsysteme installiert werden.

Geschäftslogik

Unter dem Entwurf der Geschäftslogik wird die Modellierung der funktionalen Anforderungen verstanden [Zuse01]. Aufbauend auf dem Analysemodell wird ein Klassendiagramm um Funktionen und Schnittstellen erweitert.

Klassendiagramme sind das Modellierwerkzeug schlechthin in der Entwurfphase. In dieser Abstraktionsebene des Klassendiagramms werden sowohl der Typ der Attribute, wie auch die Signatur der zu implementierenden Methoden festgelegt. In den meisten Fällen ist eine Einteilung des relativ unübersichtlichen Klassendiagramms in Paketen in diesem Schritt des Projektes zu empfehlen.

Die Modellierung der Geschäftslogik erfolgt mit Hilfe von Aktivitäten-, Zustands-, Sequenz-, Kommunikations- und Zeitdiagrammen. Mittels Sequenz- und Kommunikationsdiagrammen kann der detaillierte Ablauf von Operationsaufrufen modelliert werden. Mit dieser Form des Sequenzdiagramms kann eine gute Ausgangsbasis für die Implementierung der Kernfunktionalitäten geschaffen werden. Zeitdiagramme werden bei Enterprise Anwendungen kaum verwendet.

Wurden viele zusammenhängende Interaktionsdiagramme modelliert, kann ein Interaktionsübersichtdiagramm modelliert werden, um die Übersicht zu wahren.

Subsysteme

Für die Aufteilung des Systems in Subsystemen eignet sich am besten das Paketdiagramm. Grosse Pakete werden in kleinere zerlegt, um den Überblick über das System zu behalten. Die funktionale Gliederung mittels Paketdiagrammen ist in dieser Phase des Projektes unumgänglich. Weiters können in UML 2.0 Komponenten- und Verteilungsdiagramme eingesetzt werden, um Subsysteme zu finden.

Datenbasis

Für die Modellierung einer relationalen Datenbank wird in UML 2.0 kein eigenes Diagramm angeboten. Eine Übersicht kann mit einem Klassendiagramm gegeben werden, allerdings erlaubt ein Klassendiagramm keine Modellierung von Feldlängen und Pflichtfelder. Für die Modellierung einer relationalen Datenbank sollte ein EER-Modell (Extended-Entity-Relationship-Modell) verwendet werden [Zuse01].

5.3 Implementierung

Für den Implementierungsschritt sollte zuerst ein Integrationsplan erstellt werden. Aufbauend auf diesem sollte mit der Implementierung der Subsysteme, der Architektur und der Klassen begonnen werden [Zuse01].

In UML 2.0 gibt es wenige Diagrammarten, die bei der Implementierung hilfreich sind, da UML eine Modellierungssprache ist und viele Diagramme schon in vorigen Phasen des Projekts eingesetzt wurden. Nichtsdestotrotz sollten ein paar Diagrammarten in Hinblick auf eine modellgetriebene Entwicklung nicht unerwähnt bleiben.

So könnte das Zustandsdiagramm als gute Ausgangsbasis für die automatische Erstellung verschiedener Konfigurationsdateien dienen. Die beiden vorgestellten Java Enterprise Frameworks erlauben eine sehr gute Abbildung von deren Konfigurationsdateien auf das Zustandsdiagramm.

Mit Hilfe eines Aktivitätsdiagramms können unter anderem, seit UML 2.0 auch Algorithmen modelliert werden. Diese sind dem Nassi-Shneidermann-Struktgramm

(vgl. [Soph05]) sehr ähnlich. In diesem Fall würde ich allerdings andere, schon länger bewährte Methoden vorziehen (z.B. Pseudocode, Struktogramm, …). Meiner Meinung nach wurde in UML 2.0 diese Möglichkeit geschaffen, um auch eine Möglichkeit zu bieten, Algorithmen zu modellieren, was mit älteren Versionen von UML nicht möglich war.

5.4 Test

Die Haupttätigkeiten im Arbeitsschritt Test ist die Erstellung eines Testplans und die Erstellung der Testfälle. Bei der Durchführung der Tests werden Testberichte und Fehlerberichte erstellt [Zuse01].

Bei der Erstellung der Testfälle kann das Sequenzdiagramm eine Hilfe darstellen. Die Diagrammart erlaubt die Abbildung von Testszenarien. Besonders hervorzuheben sind die in UML 2.0 neu eingeführten „neg-Fragemente“ im Sequenzdiagramm, sie erlauben die Modellierung von negativen Testfällen. Die Modellierung von Testdaten kann mit einem Objektdiagramm erfolgen.

Für einen Test, wie das System implementiert wurde kann das Klassendiagramm mit dem Ergebnis verglichen werden. Dieser Vergleich kann als Qualitätsmerkmal der Implementierung verstanden werden.

Kapitel 6

Zusammenfassung und Ausblick

Als Zusammenfassung soll eine kurze Beschreibung der Einsatzmöglichkeiten von UML 2.0 Diagrammen gegeben werden. Außerdem soll ein kurzer Ausblick angeführt werden.

6.1 Einsatzmöglichkeiten

In dieser Arbeit wurde eine Übersicht über die Einsatzmöglichkeiten von UML 2.0 bei der Entwicklung von Enterprise Anwendungen gegeben. Zusammengefasst kann gesagt werden, dass die Architektur von Enterprise Anwendungen und die Modellierungssprache UML 2.0 gut zusammenspielen, wobei einige Diagrammarten gut eingesetzt werden können und andere weniger gut.

Besonders hervorzuheben ist die Modellierung der Steuerung der Userschicht mit Hilfe des Zustandsdiagramms. Das Zustandsdiagramm bietet die Möglichkeit Konfigurationsdateien von Java Enterprise Frameworks herzuleiten.

Im Rahmen dieser Arbeit wurde gezeigt welche Diagrammart von UML 2.0, in welcher Schicht der mehrschichtigen Enterprise Anwendung zu verwenden ist, und mit welchen Diagramm das Zusammenspiel der Schichten modelliert werden kann. Das Anwendungsfalldiagramm bietet dabei eine gute Übersicht über das gesamte System, ohne dabei auf die Schichten einzugehen. Aktivitäts- und Zustandsdiagramme eignen sich besonders gut, um die Präsentationsschicht einer Enterprise Anwendung zu beschreiben. Das Zusammenspiel der verschiedenen Schichten kann am besten mit Hilfe von Paket-, Komponenten- und dem Verteilungsdiagramm modelliert werden.

Einen Überblick über das Zusammenspiel der Schichten kann auch mit dem Sequenz- und Kommunikationsdiagramm gegeben werden.

Diese Arbeit zeigt auch, wie die Architektur einer Enterprise Anwendung beschrieben werden kann, und welche Aspekte mit welcher Diagrammart modelliert werden kann. Dabei sind Sequenzdiagramme im Zusammenhang mit Klassendiagrammen besonders hervorzuheben. Mit Hilfe dieser Diagrammarten können auch Designpatterns sehr gut dargestellt werden (vgl. auch [Gamm96]).

In dieser Arbeit wurde auch das Zusammenspiel zwischen UML 2.0 und dem Unified Process im Bezug auf Enterprise Anwendungen evaluiert. Es wurde gezeigt welche Diagrammart in welcher Phase des Unified Process verwendet werden kann. Hier sind das Anwendungsfall-, das Klassen- und das Sequenzdiagramm besonders hervorzuheben. Prinzipiell können alle 13 Diagrammarten von UML 2.0 eingesetzt werden, wobei das Zeitdiagramm und das Interaktionsübersichtsdiagramm in Bezug auf Enterprise Anwendungen am wenigsten verwendet wird.

Zum Abschluss der Zusammenfassung wird noch eine Bewertung der Diagrammarten in Bezug auf Enterprise Anwendungen und in Bezug auf den Unified Process gegeben. Die Bewertung spiegelt die gewonnenen Einsichten wieder. In dieser Übersicht wird zwischen „häufig eingesetzt“ (1), „benutzbar“ (2) und „wenige Einsatzmöglichkeiten“ (3) unterschieden.

	Enterprise Applikationen	Unified Process
Anwendungsfalldiagramm	2	1
Aktivitätsdiagramm	2	1 - 2
Zustandsdiagramm	1	1 - 2
Sequenzdiagramm	1	1
Kommunikationsdiagramm	2	2
Zeitdiagramm		3
Interaktionsübersichtsdiagramm	3	3
Klassendiagramm	1	1
Paketdiagramm	1	1 - 2
Objektdiagramm	1 - 2	1
Kompositionsstrukturdiagramm	1	1 - 2
Komponentendiagramm	1	1 - 2
Verteilungsdiagramm	1	1

Abbildung 48: Bewertung UML 2.0 Diagramme

6.2 Ausblick

Zusammengefasst kann gesagt werden, dass UML 2.0 im Zusammenspiel mit Enterprise Anwendungen sehr gut zu gebrauchen ist, da jeder Bereich sehr gut abgedeckt wird.

Ein kleiner Nachteil dabei ist allerdings, dass die mächtige Modellierungssprache UML 2.0 nicht vollständig zum Einsatz kommt, und sich einige Diagramme nicht für die Modellierung von Enterprise Anwendungen eignen. Das sind insbesondere das Zeitdiagramm und das Interaktionsübersichtdiagramm. Das Zeitdiagramm ist bei der Modellierung von *Embedded Systems* ein zentraler Bestandteil, da es vor allem bei zeitkritischen Anwendungen verwendet wird. Eine weitere Arbeit könnte sich mit UML 2.0 im Zusammenhang mit *Embedded Systems* beschäftigen. Enterprise Anwendungen lassen sich jedoch mit Hilfe von Zeitdiagrammen nicht modellieren, da meist viele unvorhersehbare Ereignisse, wie die System- oder Netzwerklast, beachtet werden müssen. Das Interaktionsübersichtsdiagramm ist als neues Diagramm noch nicht etabliert und auch bei UML-Experten umstritten.

Eine weitere Erweiterung und Verbesserung von der Modellierung von Enterprise Anwendungen stellen UML 2.0 Profile dar. „*Profile ermöglichen das Zuschneiden (Tailoring) der UML auf Ihre Projektumgebung durch Anpassung von UML-Standardelementen. Die Erweiterungen bzw. Einschränkungen einzelner Elemente des UML-Metamodells lassen sich in Paketen als Profile bündeln und beliebig auf Modelle anwenden, austauschen und wieder entfernen.*“ [Rupp05].

Es existieren einige UML 2.0 Profile, die sich mit Enterprise Java Beans und der Modellierung von Patterns auseinandersetzen (vgl. [OMG04] und [OMG04a]). Weitere Profile und Arbeiten beschäftigen sich mit der Modellierung des User Interfaces und der Navigation von Webanwendungen (vgl. z.B. [Henn01]). Mit der Modellierung im Web Engineering beschäftigt sich unter anderem [Kapp04].

Eine weitere Arbeit könnte sich mit der Kombination von EJB Profilen und Profile für Webanwendungen beschäftigen.

Als abschließenden Punkt soll die Generierung von Code aus den Diagrammen (*MDA - Model Driven Architecture* - vgl. [OMG07b]) bzw. die Generierung von Diagrammen aus Code (*ADM - Architecture Driven Modernization* - vgl.

[OMG07c]) angesprochen werden. Sehr viele Arbeiten beschäftigen sich mit Modellgetriebener Softwareentwicklung. Ziel dieser Ansätze ist es, aus den Modellen lauffähigen Code zu erzeugen. In Bezug auf Enterprise Anwendungen wird dieser Ansatz schon, zumindest teilweise, angewendet (vgl. [Arcs07]).

Literaturverzeichnis

[Arcs07] ArcStyler; http://www.arcstyler.com [16.10.2007]

[Ahme02] K. Ahmed, C. Umrysh; Developing Enterprise Java Applications with J2EE and UML, Addison-Wesley, Indianapolis, 2002

[Busc96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal; Pattern-oriented Software Architecture. A System of Patterns, Wiley, Chichester (England), 1996

[Demo06] M. Demolsky; State of the Art in Java Enterprise Web Application Development, Diplomarbeit, TU Wien, 2006

[Dona07] K. Donald, E. Vervaet; Spring Web Flow, http://www.springframework.org/webflow, [22.7.2007]

[Gamm95] E. Gamma, R. Helm, R. E. Johnson; Desgin Patterns. Elements of Reusable Object-Oriented Software, Addison-Wesley, Amsterdam, 1995

[Henn01] R. Hennicker, N. Koch; Modeling the User Interface of Web Applications with UML, 2001

[Hitz05] M. Hitz, G. Kappel; UML@work - Objektorientierte Modellierung mit UML 2, 3. Auflage, dpunkt, Heidelberg, 2005

[Haig05] S. Haiges (Hrsg.), A. Bien, M. May, B. Woehrlich; Struts - Java Framework für Webanwendungen, Software & Support Verlag GmbH., Frankfurt, 2005

[Jaco99] I. Jacobson, G. Booch, J. Rumbaugh; The unified software development process, Addison-Wesley, 1999

[John03] R. Johnson; Expert one-on-one, J2EE Design and Development, Wiley, Indianapolis, 2003

[John05] R. Johnson; Introduction to the String Framework; The Server Side; http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework, 2005, [18.8.2007]

[John07] R. Johnson; Spring 2.0: What's New and Why it Matters; http://www.infoq.com/articles/spring-2-intro, 2007 [30.4.07]

[Kapp04] G. Kappel, B. Pröll, S. Reich, W. Retschitzegger (Hrsg.); Web Engineering - Systematische Entwicklung von Web-Anwendungen, dpunkt, 2004

[Lang06] T. Langner, D. Reiberg; J2EE und JBoss - Verteilte Enterprise Anwendungen auf der Basis von J2EE, JBoss & Eclipse, Hanser, München, 2006

[Mari02] F. Marinescu; EJB Design Patterns - Advanced Patterns, Processes and Idioms, Wiley, New York, 2002

[Möss05] H. Mössenböck; Sprechen Sie Java?, dpunkt, Juli 2005; Musterlösungen, http://www.ssw.uni-linz.ac.at/Misc/JavaBuch/Muster [1.5.07]

[Oate07] R. Oates, T. Langer, S. Wille, T. Lueckow, G. Bachlmayr; Spring & Hibernate - Eine praxisbezogene Einführung, Hanser, München 2007

[OMG04] Metamodel and UML Profile for Java and EJB Specification, http://www.omg.org/docs/formal/04-02-02.pdf, 2004 [1.8.2007]

[OMG04a] UML Profile for Patterns Specification, http://www.omg.org/docs/formal/04-02-04.pdf, 2004 [1.8.2007]

[OMG07] The Object Management Group (OMG), http://www.omg.org [21.7.2007]

[OMG07a] The Object Management Group (OMG), CORBA Basiscs, http://www.omg.org/gettingstarted/corbafaq.htm [21.7.2007]

[OMG07b] The Object Management Group (OMG), Model Driven Architecture, http://www.omg.org/mda [16.10.2007]

[OMG07c] The Object Management Group (OMG), Architecture-Driven Modernization (ADM), http://adm.omg.org [16.10.2007]

[Reic06] M. Reichold; Evaluierung des UML Modellierungswerkzeuges StarUML, Magisterarbeit, TU Wien, Oktober 2006

[Rupp05] C. Rupp, J. Hahn, S. Queins, M. Jeckle, B. Zengler, UML 2 glasklar - Praxiswissen für die UML-Modellierung und Zertifizierung, Hanser, München, 2005

[Schä02] R. Schätzle, T. Seifert, J. Kleine-Gung; Enterprise JavaBeans - Kritische Betrachtungen zu einer modernen Software-Architektur, Wirtschaftsinformatik 44, 2002, S. 217 - 224

[Soph05] Sophist Group; Vergleich von Nassi-Shneidermann-Struktogrammen und Aktivitätsdiagrammen, http://www.uml-glasklar.com [1.5.07]

[Star05] T. Stark; J2EE - Einstieg für Anspruchsvolle, Addison-Wesley, München 2005

[Star07] http://staruml.sourceforge.net/en/ (22.7.2007)

[Sun07] Sun Microsystems, Core J2EE Patterns: Patterns index page, http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html [29.4.07]

[Wang03] D. Wang; Mehr als ein Kaffeefilter - Java Servlet-Filter in der Webapplikation, Java Magazin, 02.2003

[Wiki07] Wikipedia, Operation (Informatik) http://de.wikipedia.org/wiki/Operation_%28Informatik%29, [10.8.2007]

[Zuse01] W. Zuser, S. Biffl, T. Grechenig, M. Köhle, Software-Engineering mit UML und dem Unified Process, Pearson Studium, 2001