Universität Leipzig
Institut für Informatik
Lehrstuhl für Angewandte Telematik / e-Business
Prof. Dr. Volker Gruhn

Diplomarbeit

Klassifikation und Bewertung von Frameworks für die Entwicklung von Web-Anwendungen

Autor: Andreas Wende (Mat.-Nr. 8738889)
Studiengang: Informatik, Diplom (10. Semester)

Betreuer: Matthias Book
Erstgutachter: Prof. Dr. Volker Gruhn
Zweitgutachter: N.N.

Eingereicht am: 30.09.2005

Abstract

Die Anzahl von Frameworks zur Erstellung von Web-basierten Anwendungen im Umfeld von Java und J2EE ist inzwischen kaum noch zu überschauen. Regelmäßig kommen neue Frameworks hinzu. Dabei unterscheiden sich diese Frameworks nicht nur auf Ebene von Detaileigenschaften, technischen Qualitäten oder Anwendungsschwerpunkten – es werden auch äußerst unterschiedliche Grundkonzepte verwendet. Ziel dieser Arbeit ist es, aufbauend auf diesen unterschiedlichen Grundkonzepten, eine Klassifikation zu entwickeln, in welche sich jedes beliebige Web-Framework einordnen lässt. Danach werden die wichtigsten Frameworks aus dem Open-Source-Bereich vorgestellt und in die zuvor entwickelte Klassifikation eingeordnet. Im anschließenden praktischen Teil wird eine kleine Web-Anwendung prototypisch mit Hilfe ausgewählter Vertreter verschiedener Klassifikationskategorien implementiert, um diese zu vergleichen und zu bewerten. Ziel ist es, Auswahlrichtlinien für die verschiedenen Kategorien von Web-Frameworks abzuleiten und so die Framework-Auswahl zu unterstützen.

Vorwort

An dieser Stelle möchte ich mich bei allen Beteiligten für das Gelingen dieser Arbeit bedanken.

Besonderer Dank gebührt dabei Matthias Book vom Lehrstuhl für Angewandte Telematik / e-Business, welcher mich mit vielen hilfreichen Anmerkungen und Hinweisen unterstütz hat. Mein Dank gilt ebenfalls Sandra Wende, Susann Lehmann und Virginie Müller, welche die große Mühe nicht gescheut haben diese Arbeit von allerlei Fehlern zu befreien.

Nicht zuletzt möchte ich an dieser Stelle die Gelegenheit nutzen und meinen Eltern danken, die mir mein Studium erst ermöglicht haben.

Inhaltsverzeichnis

Abstract **3**

Vorwort **4**

Inhaltsverzeichnis **5**

Kapitel 1 Einleitung **8**

1.1 Motivation 8

1.2 Aufgabenstellung 8

1.3 Lösungsansatz 9

1.4 Einordnung in die Literatur 10

1.5 Kapitelübersicht 10

Kapitel 2 Web-Anwendungen **12**

2.1 Definition 12

2.2 Grundkonzept 13

2.3 Anforderungen 13

2.4 Vor- und Nachteile 14

2.5 Arten von Web-Anwendungen 15

2.5.1 Statische Web-Anwendungen 15

2.5.2 Dynamische Web-Anwendungen 15

2.5.3 Web-Portale 17

2.6 Technologien 18

2.6.1 Clientseitige Technologien 18

2.6.2 Serverseitige Technologien 21

2.7 Architekturen 24

2.7.1 Schichten-Architektur 25

2.7.2 Model-View-Controller-Architektur 25

2.7.3 Pipeline-Architektur 28

Kapitel 3 Frameworks für Web-Anwendungen **29**

3.1 Einleitung 29

3.2 Definition 30

3.2.1 Andere Konzepte der Softwarewiederverwendung 31

3.2.2 White-Box- und Black-Box-Frameworks 33

3.3 Anforderungen an Web-Frameworks 34

3.4 Klassifikationskriterien 34

3.4.1 Anwendungssteuerung 35

3.4.2 User-Interface-Komponenten 36
3.4.3 Darstellung 37
3.4.4 Datenübergabe 38
3.4.5 Dialogsteuerung 39
3.4.6 Validierung 43
3.4.7 Sitzungsmanagement 44
3.5 Fazit 45

Kapitel 4 Klassifikation 48

4.1 Struts 49
4.1.1 Aufbau 49
4.1.2 Controller 50
4.1.3 View 51
4.1.4 Klassifikation 52
4.2 Velocity 53
4.2.1 Funktionsweise 54
4.2.2 Klassifikation 55
4.3 Turbine 56
4.3.1 Aufbau und Funktionsweise 57
4.3.2 Klassifikation 60
4.4 Tapestry 60
4.4.1 Aufbau 61
4.4.2 Seiten 61
4.4.3 Komponenten 63
4.4.4 Klassifikation 64
4.5 JavaServer Faces 65
4.5.1 Funktionsweise 65
4.5.2 Request-Response-Zyklus 67
4.5.3 Ereignisverarbeitung 68
4.5.4 Dialogsteuerung 68
4.5.5 Klassifikation 69
4.6 ASP.NET 70
4.6.1 Funktionsweise 70
4.6.2 Klassifikation 72
4.7 wingS 73
4.7.1 Aufbau 73
4.7.2 Ereignisverarbeitung 74
4.7.3 Klassifikation 74
4.8 Echo 75
4.8.1 Funktionsweise 76
4.8.2 Ereignisverarbeitung 77
4.8.3 Klassifikation 77
4.9 Cocoon 78
4.9.1 Aufbau und Funktionsweise 79
4.9.2 Klassifikation 81
4.10 Fazit 82

Kapitel 5 Anwendungsbeispiel 84
5.1 Anforderungen 84
5.2 Analyse 85
5.3 Entwurf 86
5.4 Implementierung 87
5.4.1 Struts 87
5.4.2 JavaServer Faces / myFaces 92
5.4.3 Echo 98
5.5 Vergleich 100
5.5.1 Konfiguration, Anwendungslogik und Dialogsteuerung 100
5.5.2 Darstellung 101
Kapitel 6 Bewertung 103
6.1 Fazit 103
6.2 Diskussion 105
6.3 Ausblick 105
Literatur 107
Abbildungsverzeichnis 110
Tabellenverzeichnis 111
Abkürzungsverzeichnis 112
Index 114
Erklärung 116

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahren hat sich Java als eine führende Technologie für die Entwicklung von komplexen Web-Anwendungen etabliert. Die Java-Plattform bietet dabei Technologien wie Servlets und JavaServer Pages an, um skalierbare und robuste browserbasierte Anwenderschnittstellen für zahlreiche Anwendungen zu entwickeln. Aber immer komplexer werdende Web-Anwendungen machen es notwendig, sich im Sinne der Wartbarkeit, Wiederverwendbarkeit, Erweiterbarkeit, Konfigurierbarkeit und einfacher Entwicklung grundsätzliche Gedanken über die Struktur solcher Web-Anwendungen zu machen.

Web-basierte Anwendungen werden heute in der Regel nach dem Model-View-Controller-Entwurfsmuster [KP88] entwickelt, um Präsentations- und Anwendungslogik so weit wie möglich zu trennen. Erst dies ermöglicht eine sinnvolle Aufgabenteilung, so dass auch mehrere Entwickler parallel entwickeln können. Nur dadurch lassen sich große Web-Anwendungen entwickeln, erweitern und langfristig pflegbar halten. Für diesen Ansatz existieren eine Reihe von Frameworks, die verschiedene Teile der Web-Anwendungsentwicklung unterstützen. Unter einem Framework ist dabei ein Gerüst zu verstehen, das dem Entwickler verschiedene Bausteine (Komponenten) zur Verfügung stellt, die immer wieder verwendet werden können und nach einem bestimmten Designmuster zusammengefügt werden müssen. Durch den Einsatz solcher Frameworks kann eine gewisse Abstraktion von technischen Details erreicht werden, wodurch die Entwicklung insgesamt deutlich beschleunigt wird.

1.2 Aufgabenstellung

Die Anzahl von Frameworks zur Erstellung von Web-basierten Anwendungen im Umfeld von Java und J2EE ist inzwischen jedoch kaum noch zu überschauen. Auch kommen regelmäßig neue Frameworks hinzu. Dabei unterscheiden sich diese Frameworks nicht nur auf der Ebene von Detaileigenschaften, technischen Qualitäten oder Anwendungsschwerpunkten – es werden auch äußerst unterschiedliche Grundkonzepte verwendet.

Die meisten dieser Frameworks sind auf Open-Source-Basis frei erhältlich und basieren auf der J2EE-Plattform. Um hier den Überblick zu behalten bzw. eine gezielte Auswahl treffen zu können, ist es nötig, geeignete Kategorien zur

Klassifikation zu entwickeln. Diese können dann verwendet, um Schlüsse für die Anwendbarkeit bestimmter Frameworktypen auf bestimmte Problemstellungen zu gewinnen.

Dabei wird sich in dieser Arbeit auf die bekanntesten Frameworks aus dem Java-Umfeld konzertiert. Wo dies zu Vergleichszwecken sinnvoll erscheint, werden jedoch auch andere Frameworks in die Betrachtung einbezogen.

1.3 Lösungsansatz

In der vorliegenden Arbeit wird zunächst eine Definition für den Begriff Framework angegeben und dieser zu anderen Konzepten der Softwarewiederverwendung abgegrenzt.

Es werden die Gründe für das Entstehen solcher Frameworks erörtert und erläutert in welchem Zusammenhang sie zur allgemeinen Entwicklung von Web-Anwendungen stehen. Es wird auch geklärt, welche Arten von Web-Anwendungen es gibt, um später Aussagen darüber treffen zu können, welche Anwendungen besonders gut mit bestimmten Typen von Frameworks erstellt werden können. Weiterhin werden die technologischen Grundlagen kurz dargestellt, insbesondere wenn sie für die Entwicklung von Web-Anwendungen auf Basis von Frameworks von Bedeutung sind. Dies zielt eher auf die grundlegenden Konzepte als auf technische Details ab und wird die spätere Einordnung der Frameworks erleichtern.

Das Hauptziel der Arbeit ist, eine Klassifikation von Frameworks zu erstellen. Dazu müssen zuerst die grundlegenden Konzepte von Frameworks herausgearbeitet werden und Kriterien gefunden werden, anhand derer eine Einordnung möglich ist. Es werden dabei grundlegende konzeptionelle Unterschiede betrachtet, nicht aber die jeweiligen Feature-Listen einzelner Frameworks gegenübergestellt.

Anschließend werden ausgesuchte Frameworks anhand der definierten Kriterien eingeordnet und ihre Funktionsweise und Architektur etwas näher beleuchtet. Dabei wird sich aus Gründen des Umfangs dieser Arbeit auf die wichtigsten Frameworks aus dem Open-Source-Bereich beschränkt. Wo dies sinnvoll erscheint, werden jedoch auch weitere Frameworks einbezogen. Weiterhin bleibt zu sagen, dass es kaum möglich ist, alle Aspekte der vorgestellten Frameworks bis ins letzte Detail wieder zu geben. Dies ist aber auch nicht Ziel dieser Arbeit.

Im praktischen Teil dieser Arbeit wird eine einfache Web-Anwendung prototypisch mit jeweils einem typischen Vertreter der einzelnen Kategorien implementiert. Die Applikation wird dabei typische Funktionen großer Web-Anwendungen besitzen:

- Der Nutzer muss sich einloggen können, dabei soll es auch verschiedene Rollen geben (z.B. normaler Nutzer, Administrator usw.).
- Das Anmelden als neuer Nutzer soll realisiert werden, wenn möglich über einen mehrstufigen Dialog.
- Benutzereingaben sollen validiert werden und Fehleingaben einfach korrigiert werden können.

- Ein Navigationsmenü soll immer sichtbar sein.
- Eine Suchmaske (für Artikel, Transaktion oder ähnliches) mit entsprechender Anzeige der Ergebnisse in Tabellenform soll existieren.
- Die Anwendung soll mehrere Sprachen unterstützen.

Als konkrete Beispiele werden dabei Apache Struts (als ein eher klassischer Ansatz) und dem gegenübergestellt JavaServer Faces (als ein ereignisgesteuerter Ansatz) verwendet. Daneben wird noch das Echo-Framework verwendet, welches stark vom Charakter einer Web-Anwendung abstrahiert.

1.4 Einordnung in die Literatur

Bei der Charakterisierung des Begriffes Framework baut diese Arbeit grundsätzlich auf den Ergebnissen von [JF88, BBE95] auf:

[JF88] beschäftigt sich damit, wie die objektorientierte Programmierung genutzt werden kann, um eine Wiederverwendung von Software zu ermöglichen. Dazu wird ein Überblick über verschieden Techniken gegeben und diese genauer beschrieben. Ein zentraler Schwerpunkt liegt auf Frameworks als Technik zur Softwarewiederverwendung. Dabei konnten zwei grundsätzliche Arten von Frameworks identifiziert werden: Die grundsätzliche Unterscheidung zwischen White-Box- und Black-Box-Frameworks soll in dieser Arbeit genutzt werden, um eine bessere Klassifikation speziell von Frameworks für Web-Anwendungen zu ermöglichen.

[BBE95] beschreibt grundsätzliche Eigenschaften von Frameworks und wie diese die Wiederverwendung von Software ermöglichen. Dabei wird auf der Unterscheidung zwischen White-Box- und Black-Box-Frameworks aus [JF88] aufgebaut und an Praxisbeispielen dargestellt, wie Frameworks im Zusammenhang mit der komponentenbasierten Entwicklung stehen. Wie Frameworks die Verwendung von Komponenten unterstützen, wird später ebenfalls zur Klassifikation genutzt.

Ein ähnliches Ziel wie diese Arbeit verfolgen die Artikel [Se03, Wa04]. So werden in [Se03] zwei Web-Frameworks mit unterschiedlichen Grundkonzeptionen vorgestellt und miteinander verglichen. Dabei stellt sich die Anwendungssteuerung als ein grundlegendes Merkmal zur Unterscheidung von Web-Frameworks heraus. In [Wa04] wird ein Überblick über die wichtigsten Web-Frameworks aus dem Open-Source-Bereich gegeben. Die in diesen Arbeiten gefundenen Unterscheidungsmerkmale werden später genauer analysiert und zur Klassifikation von Web-Frameworks genutzt.

1.5 Kapitelübersicht

In Kapitel 2 wird zunächst geklärt, was in dieser Arbeit unter Web-Anwendungen zu verstehen ist. Dazu wird eine Definition angegeben und das grundlegende Konzept von Web-Anwendungen erläutert. Danach werden die sich für den praktischen Einsatz ergebenden Anforderungen beschrieben sowie auf die Vor- und Nachteile von Web-Anwendungen gegenüber anderen Anwendungen

eingegangen. Abschließend werden die wichtigsten Arten von Web-Anwendungen unterschieden sowie auf die verwendeten Technologien und typische Architekturen eingegangen.

Anschließend wird in Kapitel 3 eine Klassifikation für Web-Frameworks entwickelt. Dazu wird zuerst geklärt, warum sich Frameworks für die Entwicklung von Web-Anwendungen eignen, wie sie definiert sind und wie sie sich von anderen Konzepten der Softwarewiederverwendung unterscheiden. Danach werden typische Aufgabenfelder von Frameworks in der Entwicklung von Web-Anwendungen beschrieben. Im Hauptteil wird dann anhand von geeigneten Kriterien eine Klassifikation für Web-Frameworks aufgebaut. Insbesondere werden auch die Abhängigkeiten zwischen den einzelnen Kriterien untersucht.

Auf Basis der in Kapitel 3 entwickelten Klassifikation, wird in Kapitel 4 ein Überblick über die wichtigsten verfügbaren Frameworks gegeben. Ziel ist es, jedes dieser Frameworks in die zuvor entwickelte Klassifikation einzuordnen. Um dies zu ermöglichen, wird zu jedem Framework die grundlegende Funktionsweise und das dahinter liegende Konzept herausgearbeitet und dargestellt. Zur besseren Übersicht werden die wichtigsten Eigenschaften jedes Frameworks in einer Tabelle zu Beginn des jeweiligen Abschnitts zusammengefasst und im Anschluss ausführlich diskutiert.

In Kapitel 5 wird eine einfache Web-Anwendungen prototypisch mit jeweils einem Vertreter der in Kapitel 4 identifizierten Grundtypen von Web-Frameworks umgesetzt. Diese Umsetzung berücksichtigt dabei typische Anforderungen an große Web-Anwendungen. Ziel dieses Kapitels ist es, anhand der Implementierung mit jeweils einem typischen Vertreter Auswahlrichtlinien für die verschieden Kategorien abzuleiten.

Die Ergebnisse dieser Arbeit werden in Kapitel 6 abschließend zusammengefasst und bewertet. Die gefundene Lösung wird ausführlich diskutiert und ein Ausblick auf mögliche zukünftige Entwicklungen und weiterführende Untersuchungen gegeben.

Kapitel 2

Web-Anwendungen

In diesem Kapitel wird geklärt, was in dieser Arbeit unter Web-Anwendungen zu verstehen ist. Dazu wird eine Definition angegeben und das grundlegende Konzept von Web-Anwendungen erläutert. Danach werden die sich für den praktischen Einsatz ergebenden Anforderungen beschrieben sowie auf die Vor- und Nachteile von Web-Anwendungen gegenüber anderen Anwendungen eingegangen. Abschließend werden die wichtigsten Arten von Web-Anwendungen unterschieden sowie auf die verwendeten Technologien und typische Architekturen eingegangen.

2.1 Definition

Ursprünglich war das Web als reines Informationsmedium konzipiert. Webseiten dienten nur zur Darstellung von statischen Informationen. In letzter Zeit hat sich das Web allerdings auch zur Plattform für komplexe Softwaresysteme entwickelt. Sie stellen interaktive, datenintensive und personalisierte Dienste über diverse Endgeräte zur Verfügung. Der entscheidende Unterschied zu anderen Anwendungsarten liegt in der Nutzung der Technologien und Standards des Netzes. In dieser Arbeit verstehen wir daher unter einer Web-Anwendung folgendes:

> *„Eine **Web-Anwendung** ist ein Softwaresystem, das auf Spezifikationen des World Wide Web Consortium (W3C) beruht und Web-spezifische Ressourcen wie Inhalte und Dienste bereitstellt, die über eine Benutzerschnittstelle, den Web-Browser, verwendet werden.“ [Ka04]*

Die Funktionalität einer Web-Anwendung wird also typischerweise über das Web bereit gestellt. Allerdings können Web-Anwendungen genauso gut in internen Netzen (Intranets) eingesetzt werden. Aber auch andere Web-Clients wie mobile Endgeräte, Handys usw. sind denkbar. Die Eigenschaften, die eine Web-Anwendung demnach ausmachen, sind:

- Für den Zugriff auf die Inhalte und Dienste wird ein Web-Client (i.A. ein Browser) benötigt.
- An den Client des Nutzers werden nur Daten übertragen, jedoch keine oder nur ein geringer Teil der Anwendungslogik.

- Die Kommunikation zwischen Anwendung und Web-Client basiert auf Standards des W3C, gewöhnlich dem HTTP-Protokoll [W3C99].
- Zur Darstellung der Anwendung im Web-Client werden Markup-Sprachen, wie beispielsweise HTML [W3C01], verwendet.

2.2 Grundkonzept

Web-Anwendungen funktionieren typischerweise nach dem *Request-Response-Zyklus* (siehe Abbildung 2.1). Der Zyklus beginnt immer bei einem Web-Client, dieser schickt eine HTTP-Anfrage an den Web-Server. In der Anfrage sind häufig Benutzereingaben enthalten. Diese nimmt der Web-Server entgegen, verarbeitet sie entsprechend der Anwendungslogik und schickt eine HTTP-Antwort zurück. Die Antwort enthält normalerweise das Layout und die darzustellenden Informationen in einer Markup-Sprache wie HTML. Der Web-Client stellt dies für den Nutzer dar, welcher wiederum Links anklicken oder Formulare abschicken kann, was wieder neue Anfragen an den Web-Server auslöst.

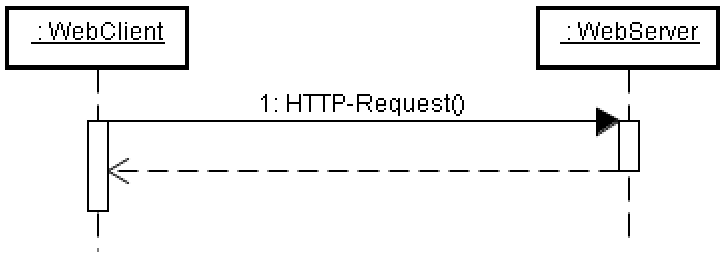

Abbildung 2.1: Request-Response-Zyklus

2.3 Anforderungen

Für den praktischen Einsatz werden verschiedene Anforderungen an Web-Anwendungen gestellt. Die wichtigsten sollen hier kurz vorgestellt werden, um später untersuchen zu können welchen Beitrag Frameworks für deren Umsetzung leisten können:

- *Sicherheit*

  Bei Web-Anwendungen werden die Anwendungsdaten, die auch personenbezogene und vertrauliche Nutzerdaten beinhalten können, in der Regel über das Internet übertragen. Der Übertragungsweg muss also vor dem Ausspähen durch Dritte geschützt werden z.B. durch Verschlüsselung. Ein anderer wichtiger Aspekt ist, dass die Identität des Nutzers bei bestimmten Anwendungen sichergestellt sein muss (*Authentifizierung*), beispielsweise bei Online-Überweisungen. Bei vielen Anwendungen ist es auch notwendig, die Rechte bestimmter Nutzergruppen zu beschränken (*Autorisierung*). So könnten z.B. alle Nutzer Lesezugriff auf bestimmte Daten haben, jedoch nur eine ausgewählte Nutzergruppe soll die Daten auch verändern dürfen. In dieser Arbeit soll unter anderem untersucht werden, wie Frameworks dies unterstützen.

- *Verfügbarkeit*

  In vielen Firmen sind Web-Anwendungen unverzichtbar für den Geschäftsbetrieb. Deren Verfügbarkeit kann daher ebenfalls eine wichtige Eigenschaft sein. Ist die Verfügbarkeit eine unverzichtbare Anforderung, so schränkt dies eventuell die verwendbaren Architekturen ein.

- *Interaktivität*

  Komplexe Anwendungen erfordern häufig eine hohes Maß an Interaktionsmöglichkeiten. Hier kann es passieren, dass die Möglichkeiten der standardisierten Steuerelemente nicht mehr ausreichen. In dieser Arbeit soll auch untersucht werden, welche Möglichkeiten Frameworks bieten, eigene komplexe Steuerelemente zu entwickeln und zu verwenden.

- *Performanz*

  Die Performanz ist ein Maß für die Effizienz, mit welcher eine Web-Anwendung die zur Verfügung gestellten Ressourcen einsetzt. Dies kann stark von der verwendeten Architektur und der Funktionsweise der Anwendung abhängen.

- *Skalierbarkeit*

  Die Skalierbarkeit kennzeichnet das Verhalten von Anwendungen bezüglich des Ressourcenbedarfs bei steigenden Anforderungen. Da sich Web-Anwendungen für den Mehrbenutzbetrieb eignen, kann es hier schnell zu einer steigenden Nutzerzahl kommen. Eine Web-Anwendung skaliert beispielsweise dann gut, wenn der Ressourcenbedarf linear mit der Nutzerzahl wächst.

- *Ressourcenzugriff*

  Web-Anwendungen sind häufig Teil von komplexen Informationssystemen (z.B. betrieblichen Informationssystemen), d.h. sie müssen meist an Ressourcen wie Datenbanken und andere Softwaresysteme angebunden werden. Hier soll untersucht werden, ob und wie Frameworks dies unterstützen.

2.4 Vor- und Nachteile

Ein entscheidender Vorteil von Web-Anwendungen gegenüber anderen Anwendungen liegt in der sofortigen Verwendbarkeit für den Nutzer. So muss keine zusätzliche Software auf dem Rechner des Nutzers installiert werden. Für die Nutzung reicht im Normalfall ein einfacher Web-Client, wie ein Browser, aus. Außerdem spielt die Plattform, auf welcher der Nutzer arbeitet, keine Rolle, da Web-Clients für fast alle Plattformen verfügbar sind.

Ein weiter wichtiger Vorteil ist die zentrale Pflege der Anwendungen, d.h. sie können zentral konfiguriert und neue Programmversionen ebenfalls zentral eingespielt werden. Zusätzlich können Informationen über Nutzerverhalten gesammelt werden, was z.B. bei Online-Shops einen Mehrwert darstellt.

Nachteile von Web-Anwendungen liegen vor allem in den, gegenüber Desktop-Anwendungen eingeschränkten, Interaktionsmöglichkeiten. So lassen sich z.B. Funktionalitäten wie Drag and Drop und Kontextmenüs nicht verwenden. Auch Funktionen, welche eine sofortige Reaktion auf Benutzereingaben erfordern, wie beispielsweise das Neuberechnen von Formeln bei Eingabe von neuen Eingangswerten oder das Verändern von Graphiken bei Mausklicks, lassen sich auf Grund der Funktionsweise von Web-Anwendungen nur sehr beschränkt umsetzen.

Negativ wirkt sich auch die Notwendigkeit aus, dass bei jeder Anfrage zusätzlich zu den eigentlichen Anwendungsdaten (Benutzereingaben und darzustellende Informationen) auch die kompletten Layoutinformationen übertragen werden müssen.

2.5 Arten von Web-Anwendungen

Web-Anwendungen können einen sehr unterschiedlichen Komplexitätsgrad aufweisen. Im Folgenden werden in Anlehnung an [RV03] die wichtigsten Arten anhand ihrer Architekturmerkmale unterschieden. Dabei wird auch kurz auf typische Einsatzgebiete sowie Vor- und Nachteile bezüglich der eben formulierten Anforderungen eingegangen.

2.5.1 Statische Web-Anwendungen

Statische Web-Anwendungen stellen die originäre Generation von Web-Anwendungen dar. Hier werden Webseiten vorgefertigt, als statische HTML-Seiten auf dem Web-Server abgelegt und durch den Nutzer abgerufen. Der Hauptvorteil liegt hier in der einfachen Architektur, welche wiederum eine hohe Stabilität und schnelle Antwortzeiten garantiert. Nachteile sind vor allem in den fehlenden Interaktionsmöglichkeiten zu sehen. Diese Architektur (Abbildung 2.2) ist sehr einfach und besteht nur aus einem einfachen Web-Server, welcher die statischen HTML-Seiten vorhält und nach Anfragen an den Client ausliefert. Statische Web-Anwendungen bieten sich vor allem für Dienste an, bei denen die Interaktionsmöglichkeiten keine Rolle spielen, z.B. zur Veröffentlichung von Dokumenten. Statische Web-Anwendungen werden in der weiteren Arbeit nicht mehr betrachtet, da hier der Einsatz von Frameworks aufgrund des einfachen Aufbaus nicht sinnvoll ist.

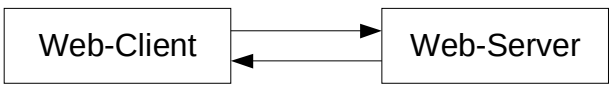

Abbildung 2.2: Architektur einer statischen Web-Anwendung

2.5.2 Dynamische Web-Anwendungen

Häufig werden an Web-Anwendungen auch Anforderungen gestellt, welche den Zugriff auf Daten aus Datenbanksystemen (DBS) und deren Anzeige erfordern. Dies kann von statischen Web-Anwendungen nicht mehr erfüllt werden. Hier müssen HTML-Seiten dynamisch zur Laufzeit erzeugt werden. Außerdem muss eine solche Anwendung auch Anwendungslogik enthalten, um z.B. eine Suche in

den Daten zu ermöglichen bzw. um die anzuzeigenden Daten eventuell in ein anderes Format zu transformieren.

Für solche Web-Anwendungen reicht ein einfacher Web-Server, der nur HTML-Dokumente ausliefern kann (wie bei statischen Web-Anwendungen), nicht mehr aus. Hier ist es notwendig, den Web-Client bzw. den Web-Server zu erweitern. Die Erweiterung des Web-Clients, beispielsweise über Java-Applets, ist dabei eher ein Spezialfall und soll deshalb nicht weiter betrachtet werden. Für die Erweiterung des Web-Servers existieren zwei Ansätze:

2.5.2.1 Erweiterung des Web-Servers

Der erste Ansatz (siehe Abbildung 2.3) ist die Integration von Modulen in den Web-Server, welche das Ausführen von Anwendungslogik ermöglichen. Die Ausgabe dieser Module wird so in die vom Web-Server gelieferte Seite integriert, dass sie für den Client als normale HTTP-Antwort des Web-Servers erscheint. Durch den Einsatz solcher Erweiterungsmodule lässt sich dann auch die Integration von Ressourcen wie Datenbanksystemen realisieren.

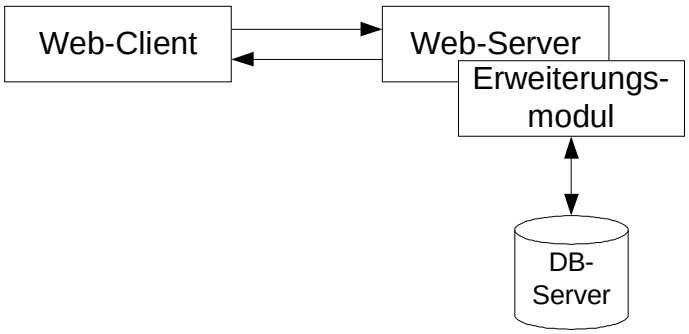

Abbildung 2.3: Architektur einer dynamischen Web-Anwendung über die Erweiterung des Web-Servers

Vorteile dieser Architektur liegen vor allem in der Gewinnung von Interaktionsmöglichkeiten gegenüber statischen Web-Anwendungen und eines trotzdem relativ einfachen Aufbaus durch die Erweiterung des Web-Servers. Nachteile können vor allem in der geringeren Performance liegen, da der Web-Server eine Seite nicht einfach vom Datenträger laden kann, sondern die Seiten eventuell bei jeder Anfrage dynamisch erzeugt werden müssen. Dieser Ansatz bietet sich deshalb besonders für kleinere bis mittelgroße Anwendungen an.

2.5.2.2 Web-Application-Server

Ein zweiter Ansatz ist der Einsatz eines Web-Application-Servers (siehe Abbildung 2.4). Hier wird der Web-Server nicht um zusätzliche Module erweitert, sondern ist selber als Teil in den Web-Application-Server integriert. Ein solcher Web-Application-Server ist eine Laufzeitumgebung für Web-Anwendungen und stellt darüber hinaus eventuell weitere Dienste für die Anwendungen bereit, z.B. die dauerhafte Speicherung von Daten in einem Datenbanksystem. Die Aufgabe des Web-Servers ist es hier nur, die von den Anwendungen erzeugten Ausgaben an den Web-Client auszuliefern.

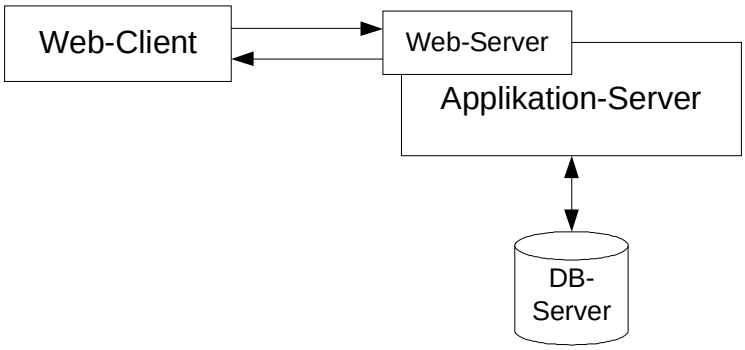

Abbildung 2.4: Architektur einer dynamischen Web-Anwendung bei Einsatz eines Web-Application-Servers

Der Vorteil beim Einsatz eines Web-Application-Servers gegenüber der Erweiterung des Web-Servers liegt vor allem in der besseren Skalierbarkeit: Da der Web-Application-Server eine eigene Laufzeitumgebung darstellt, können hier Dienste wie Lastausgleich und Caching für die Anwendungen bereitgestellt werden, ohne die eigentliche Anwendung verändern zu müssen. Auch andere vom Web-Application-Server bereitgestellte Funktionen wie Monitoring-, Logging- und Management-Funktionalitäten sowie standardisierte Schnittstellen, z.B. zu Datenbanken, können von Vorteil sein. Der Funktionsumfang einzelner Web-Application-Server kann dabei jedoch sehr unterschiedlich sein [Mar00].

Durch die aufwendige Infrastruktur die ein Web-Application-Server bereitstellt, eignet sich der Einsatz eines Web-Application-Servers vor allem für große Anwendungen, welche entweder eine hohe Last erwarten lassen oder viele zusätzliche Dienste benötigen, die ein Web-Application-Server bereit stellt. Bei kleineren Anwendungen ist der Aufwand für den Einsatz eines Web-Application-Servers dagegen eventuell zu groß.

2.5.3 Web-Portale

Die letzte vorgestellte Art von Web-Anwendungen stellen so genannte Portale dar. Die Grundidee eines Portals ist die Aggregation von so genannten Portlets auf einer Web-Seite (siehe oberen Hälfte von Abbildung 2.5). *Portlets* sind dabei an sich unabhängige Web-Anwendungen, die unter der Kontrolle eines Portlet-Containers ausgeführt werden können. Der *Portlet-Container* stellt dabei die Laufzeitumgebung für die Portlets bereit und steuert deren gesamten Lebenszyklus sowie die gesamte Interaktion mit dem Nutzer. Er stellt auch die zentrale Instanz für die Verwaltung der Nutzer sowie die Personalisierung des gesamten Portals dar. [JCP03]

Portlets können wiederum als eigenständige Anwendungen auf andere z.B. von Application-Servern bereitgestellte Anwendungen und Dienste zurückgreifen (siehe untere Hälfte von Abbildung 2.5). Durch die strikte Trennung der Portlets wird es prinzipiell auch möglich Portlets von anderen Portalen zu integrieren. Dies würde dann zu einer entfernten Ausführung des Portlets führen. Dadurch ist es auch realisierbar, dass Portale als Aggregation von anderen Portalen dienen können.

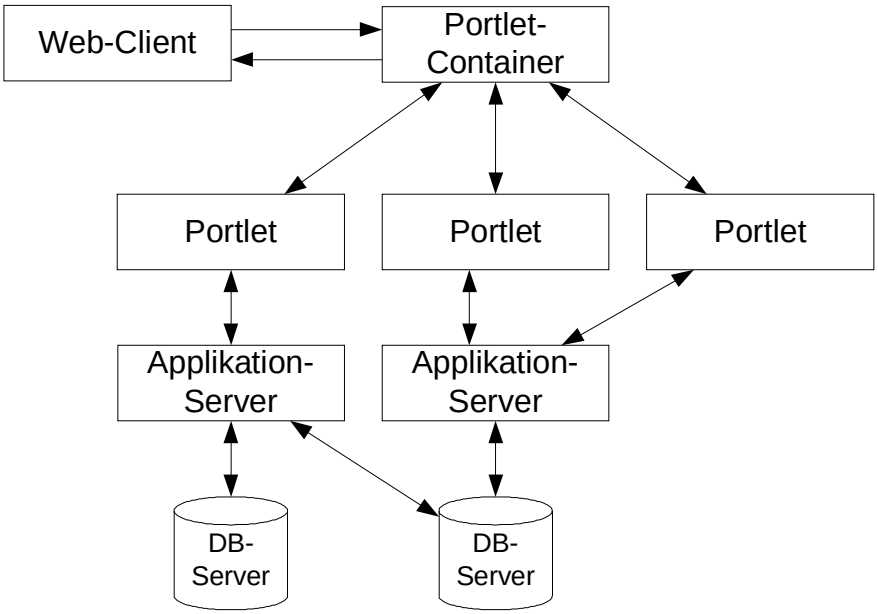

Abbildung 2.5: mögliche Architektur eines Web-Portals

Vorteile und Anwendungsmöglichkeiten von Portalen liegen vor allem in der fast beliebigen Verknüpfung und Integration von Anwendungen sowie der anwendungsübergreifenden Nutzerverwaltung. Ein möglicher Nachteil ist die Inkompatibilität vieler Portalprodukte untereinander, d.h. Portlets, welche für einen bestimmten Portlet-Container entwickelt wurden, lassen sich nicht ohne weiteres in einen Portlet-Container eines anderen Herstellers integrieren. Einen möglichen Ausweg bieten allerdings offene Standards wie z.B. in [JCP03].

2.6 Technologien

Im Folgenden sollen kurz die wesentlichen Technologien zur Realisierung von Web-Anwendungen vorgestellt werden, soweit sie für das weitere Verständnis der Arbeit wichtig sind.

2.6.1 Clientseitige Technologien

Web-Browser werden hauptsächlich zur reinen Präsentation der Web-Anwendungen verwendet. Viele gängige Browser unterstützen jedoch auch Technologien die es ermöglichen, Anwendungslogik innerhalb des Browser ausführen zu lassen. Die wichtigsten werden im Folgenden kurz dargestellt.

2.6.1.1 HTML

Die Hypertext Markup Language (HTML) ist eine reine Seitenbeschreibungssprache. Sie dient lediglich zur Darstellung und Formatierung von Inhalten und bietet selber keine Möglichkeiten, Anwendungslogik auszuführen [W3C01]. Damit ist HTML allerdings die Grundlage für fast alle Web-Anwendungen, da jeder gängige Web-Browser HTML interpretieren und darstellen kann.

2.6.1.2 Applets

Bei Applets handelt es sich um - innerhalb eines Browsers lauffähige - Java-Programme. Sie sind auf dem Web-Server abgelegt und werden bei Aufruf in eine HTML-Seite eingebettet zum Browser transferiert. Dort werden sie innerhalb des Browsers in einer Java Virtual Machine (JVM) ausgeführt. Standardmäßig besitzen sie dort nur geringe Rechte, so können sie z.B. nicht auf das lokale Dateisystem zugreifen. Das Hauptproblem beim Einsatz von Applets liegt darin, dass in verschiedenen Browsern recht verschiedene JVM verwendet werden. So kann es sein, dass bestimmte Applets erst nach der Installation einer aktuellen JVM funktionieren. Da sich diese Arbeit hauptsächlich mit serverseitigen Technologien beschäftigt, werden sie im Folgenden jedoch nicht weiter betrachtet.

2.6.1.3 JavaScript / DOM

JavaScript ist eine von Netscape entwickelte Skriptsprache, welche eingebettet in HTML vom Browser interpretiert und ausgeführt wird. Zielsetzung von JavaScrript ist es, den statischen HTML-Seiten innerhalb des Browser die Möglichkeit zu geben, auf Benutzereingaben zu reagieren. Dadurch kann die Wartezeit für den Nutzer deutlich verringert werden, da der Browser dafür keine Anfrage an den Web-Server schicken muss.

Da die verschiedenen Browser den aktuellen JavaScript-Standard [ECM05] nur teilweise oder nicht dem Standard entsprechend umsetzen, hat dies zur Folge, dass man bei der Entwicklung von Web-Anwendungen nicht grundsätzlich davon ausgehen kann, dass der entwickelte JavaScript-Code in allen Browsern korrekt ausgeführt wird. Aus diesem Grund ist beim Einsatz von JavaScript immer abzuwägen, ob die Anwendung immer noch funktioniert, wenn bestimmte JavaScript-Funktionen in einem Browser nicht unterstützt werden, oder man bereit ist, solche Browser komplett von der Nutzung der Anwendung auszuschließen.

Im engen Zusammenhang mit JavaScript steht auch das Document Object Model (DOM) [W3C05]. Dabei handelt es sich um eine vom W3C-Konsortium verabschiedete Norm, welche eine Schnittstelle beschreibt, wie Programmiersprachen auf beliebige Elemente eines Auszeichnungssprachen-Dokuments zugreifen können. Das DOM ist also weder selber eine eigene Programmiersprache, noch ist es grundsätzlich auf HTML beschränkt. Der häufigste Anwendungsfall ist jedoch das dynamische Ändern von HTML-Seiten innerhalb des Browsers. So kann JavaScript mittels DOM auf beliebige Elemente der HTML-Seite zugreifen und deren Eigenschaften auslesen oder verändern.

2.6.1.4 AJAX

Bei *Asynchronous JavaScript Technology and XML* (AJAX) handelt es sich nicht um eine Technologie im eigentlichen Sinne, sondern vielmehr einen Ansatz, wie verschiedene clientseitige Web-Technologien verwendet werden können, um mehr Interaktivität in Web-Anwendungen zu ermöglichen.

Normalerweise funktioniert eine Web-Anwendung nach dem Request-Response-Zyklus (siehe Abschnitt 2.2), d.h. die meisten Nutzeraktionen lösen eine Anfrage an den Server aus, welcher daraufhin eine neue Antwortseite generiert und an Browser zurückschickt. Mit Hilfe von JavaScript ist es jedoch auch möglich

asynchron Anfragen an den Server zu senden und die entsprechenden Antworten entgegen zu nehmen, ohne das die komplette HTML-Seite neu geladen werden muss [Ga05, Mu05].

In Abbildung 2.6 wird der typische Ablauf einer asynchronen JavaScript-Anfrage auf Basis von AJAX dargestellt [Mu05]:

1. Durch ein Benutzerereignis wird eine JavaScript-Funktion aufgerufen.
2. Ein `XMLHttpRequest`-Objekte wird erzeugt und sendet eine asynchrone HTTP-Anfrage an den Web-Server.
3. Der Web-Server verarbeitet die Anfrage und sendet eine HTTP-Antwort in Form eines XML-Dokumentes zurück.
4. Das XMLHttpRequest-Objekt nimmt die Antwort entgegen und ruft eine entsprechende JavaScript-Funktion auf.
5. Die JavaScript-Funktion verarbeitet die Antwort und kann damit beispielsweise die aktuelle HTML-Seite mit Hilfe des DOM verändern.

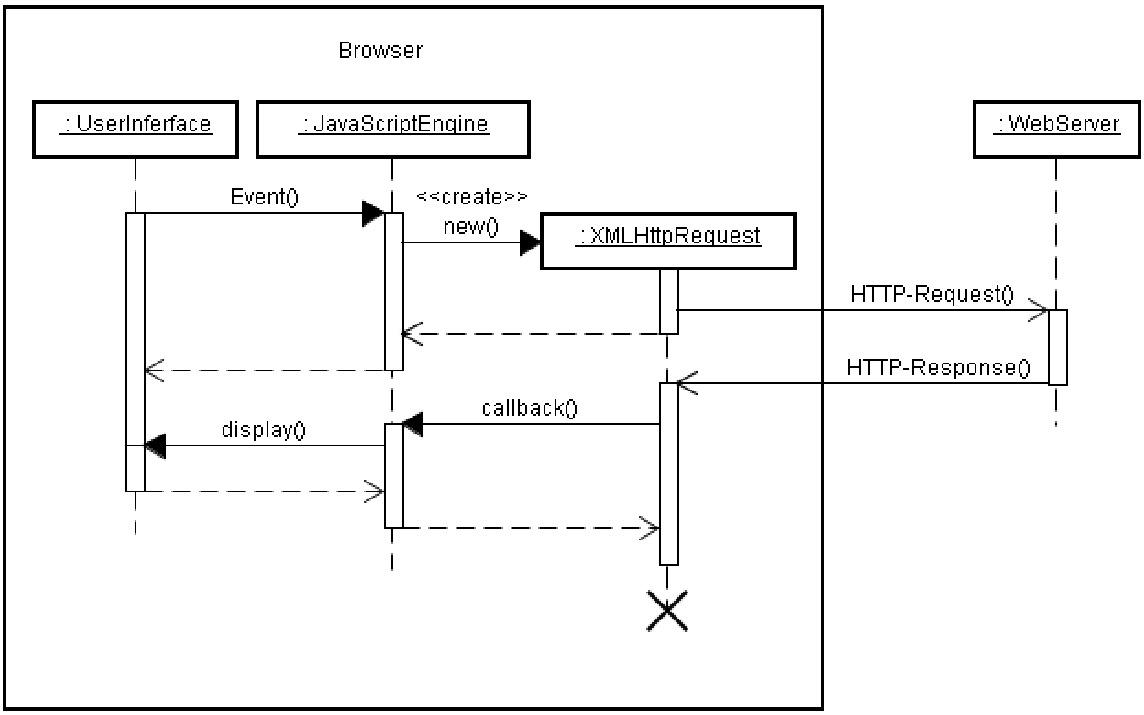

Abbildung 2.6: Asynchrone JavaScript-Anfragen (AJAX) [Mu05]

Der entscheidende Unterschied von AJAX zum normalen Request-Response-Zyklus ist also, dass der Nutzer nicht mehr direkt mit dem Server interagiert, sondern mit der aktuellen HTML-Seite innerhalb des Browsers, welche ihrerseits bei Bedarf Anfragen an den Web-Server sendet. Der AJAX-Ansatz baut vollständig auf JavaScript und DOM auf (siehe dazu Abschnitt 2.6.1.3) und benötigt daher keine Installation von zusätzlicher Software auf dem Client.

AJAX lässt sich für verschiedene Zwecke einsetzen [Mu05]:

- serverseitige Validierung von Benutzereingaben während der Eingabe

- automatische Vervollständigung von Nutzereingaben
- dynamisches Einblenden von Informationen (z.B. von Börsenkursen, Tickermeldungen, Wetter, Verarbeitungsstatus von serverseitigen Prozessen) ohne Neuladen der Seite

Mit AJAX lassen sich jedoch auch komplette Web-Anwendungen entwickeln, welche in ihren Interaktionsfähigkeiten Desktop-Anwendungen sehr nahe kommen. So baut eines der später vorgestellten Web-Frameworks vollständig auf diesem Ansatz auf [Ne05].

2.6.2 Serverseitige Technologien

Durch serverseitige Technologien werden HTML-Seiten dynamisch zur Anfragezeit zusammengebaut und an den Web-Client geschickt. Dies ermöglicht, auf Aktionen des Nutzers zu reagieren und so beispielsweise aktuelle Inhalte aus Datenbanksystemen darzustellen. Da durch den Einsatz von serverseitigen Technologien keine Anwendungslogik zum Browser geschickt werden muss, sind diese, im Gegensatz zu clientseitigen Technologien, unabhängig vom verwendeten Browser. Die wichtigsten Technologien werden im Folgenden kurz dargestellt.

2.6.2.1 J2EE

Java 2 Enterprise Edition (J2EE) [Sun05c] spezifiziert Technologien für die Entwicklung von mehrschichtigen unternehmensweiten Anwendungen [SSJ02]. Diese Spezifikation, welche auf dem Konzept des Web-Application-Servers aufbaut (siehe Abschnitt 2.5.2.2), enthält Standards zu Implementierung, Konfiguration, Verteilung und Einsatz solcher Anwendungen. Dabei wird durchgängig Java als Programmiersprache verwendet.

J2EE definiert ein verteiltes Multi-Tier-Modell für J2EE-Anwendungen. So kann eine J2EE-Anwendung generell als eine 3-Tier-Anwendung (Abbildung 2.7) aufgefasst werden, wobei der Schwerpunkt auf dem Middle-Tier und den in ihr enthaltenen Containern liegt [ABB04]:

- Im *Client-Tier* wird die graphische Benutzeroberfläche zur Verfügung gestellt. Sie kann innerhalb eines Browsers aus Applets oder HTML bestehen.
- Im *Middle-Tier* liegt der Web-Container. Er enthält Servlets, JSP und HTML-Dokumente, welche er zur Darstellung an die Client-Schicht liefern kann. Im EJB-Container wird die eigentliche Anwendungslogik implementiert. Im Gegensatz zum Web-Container werden hier verschiedene Dienste, wie z.B. Transaktionen und Persistenz, zur Verfügung gestellt.
- Über den EJB-Container kann dann auf den letzten Tier, den Enterprise-Information-System-Tier oder kurz *EIS-Tier*, zugegriffen werden. Er stellt Ressourcen wie Datenbanken oder andere unternehmensinterne Anwendungen dar.

Meist kann bei relativ einfachen Web-Anwendungen auf den EJB-Container verzichtet werden, insbesondere dann, wenn die vom EJB-Container bereitgestellten Dienste nicht benötigt werden. Auf die Ressourcen der EIS-Tier wird dann direkt vom Web-Container aus zugegriffen.

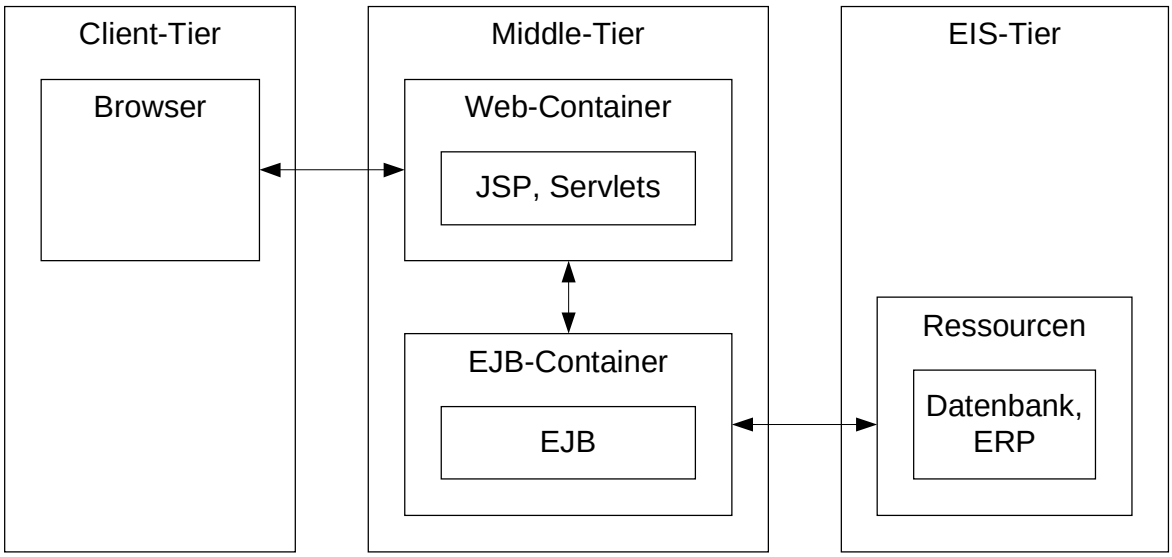

Abbildung 2.7: 3-Tier-Modell für J2EE-Anwendungen [SSJ02]

Um sicher zu stellen, dass die einzelnen Teile der J2EE-Spezifikationen auch realisierbar sind, existieren Referenzimplementierungen z.B. der Tomcat-Server [Ap05a] für die im Folgenden dargestellten Servlet- und JSP-Technologien:

Servlets [Sun05a] stellen eine Technologie im Rahmen der J2EE-Spezifikation dar. Sie lassen sich als eine Programmierschnittstelle auffassen, welche es erlaubt, Anwendungslogik in den Request-Response-Zyklus (siehe 2.2) des Web-Servers zu integrieren. Genauer handelt es sich dabei um Instanzen von speziellen Java-Klassen, die bei Bedarf von einer Laufzeitumgebung, dem so genannten Servlet-Container, erzeugt werden. Die Möglichkeit der Nutzung von objektorientierten Konzepten sowie die Plattform- und Herstellerunabhängigkeit der Java Servlet API sprechen für den Einsatz von Servlets. Durch die Verwendung von Java können Servlets die umfangreichen Funktionalitäten der Java-Klassenbibliotheken verwenden. Auch bieten Servlets im Vergleich zu anderen Technologien eine bessere Performance. So bleiben Servlets nach dem ersten Request im Speicher erhalten, während z.B. CGI-Prozesse immer wieder neu gestartet werden müssen. Des weiteren werden die Java-Klassen beim ersten Laden innerhalb des Servlet-Containers kompiliert und müssen nicht wie Skript-Sprachen (z.B. PHP) bei jedem Aufruf erneut interpretiert werden.

Eine weitere Technologie im Rahmen der J2EE-Spezifikation stellen JavaServer Pages (JSP) [Sun05b] dar. Im Gegensatz zu Servlets bei denen die Anwendungslogik im Vordergrund steht, konzentrieren JavaServer Pages sich auf das Layout einer Web-Anwendung. In der ursprünglichen Spezifikation besteht eine JavaServer Page aus HTML-Code, welcher mit Java-Code innerhalb spezieller Tags, so genannter Scriptlets, angereichert ist. In der aktuellen Spezifikation sind jedoch auch Möglichkeiten hinzugekommen, eigene Tags zu erstellen und in so genannten Tag-Libraries zu verwalten. Dadurch können komplexe Funktionalitäten gesondert programmiert und leichter wieder verwendet werden. JavaServer Pages werden von speziellen Compilern in Servlets übersetzt und können damit genauso wie Servlets in Servlet-Containern ausgeführt werden. Durch dieses Konzept kann die Komplexität der Servlet-Entwicklung zur Generierung dynamischer Web-Seiten vor dem Entwickler verborgen werden. Das

folgende Beispiel zeigt den Einsatz einiger wichtiger Elemente einer JavaServer Page [ABB04]:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions"
prefix="fn" %>
<html>
<head><title>Hello</title></head>
<body bgcolor="white">
   <img src="duke.waving.gif">
   <h2>My name is Duke. What is yours?</h2>
   <form method="get">
      <input type="text" name="username" size="25">
      <p></p>
      <input type="submit" value="Submit">
      <input type="reset" value="Reset">
   </form>
   <jsp:useBean id="userNameBean" class="hello.UserNameBean"
scope="request"/>
   <jsp:setProperty name="userNameBean" property="name"
value="${param.username}" />
   <c:if test="${fn:length(userNameBean.name) > 0}" >
      <%@include file="response.jsp" %>
   </c:if>
</body>
</html>
```

Mit `<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>` wird beispielsweise eine Tag-Library geladen und unter dem Prefix „c“ zur Verfügung gestellt. Mittels `<jsp:useBean/>` und `<jsp:setProperty/>` wird auf JavaBeans zugegriffen. `<c:if/>` ist ein spezieller Tag aus einer zuvor geladen Tag-Library und implementiert eine bedingte Anweisung.

Zusammen mit Servlets bilden JavaServer Pages die technische Basis für viele der in dieser Arbeit vorgestellten Web-Frameworks. Weitere Informationen zu deren Einsatzmöglichkeiten findet man z.B. in [SSJ02].

2.6.2.2 .NET

Microsofts .NET-Strategie steht in direkter Konkurrenz zu Suns J2EE. Während J2EE eine reine Spezifikation ist, umfasst .NET zusätzlich Produkte, die auf dieser Spezifikation beruhen. Im Gegensatz zu J2EE ist .NET nicht plattformunabhängig, dafür aber sprachunabhängig. Da alle .NET-Anwendungen in der Common Language Runtime ausgeführt werden, kann jede von .NET unterstützte Programmiersprache (C#, J#, Managed C++ und VB.NET) verwendet werden. Ein Nachteil von .NET ist jedoch die Beschränkung auf Windows-Betriebsysteme. Allerdings ist auch eine Open-Source-Implementierung von .NET unter dem Namen „Mono“ in Arbeit, welche auch auf anderen Plattformen lauffähig ist. Zur Zeit beschränkt sich der Markt jedoch noch allein auf Microsoft Produkte. Weitere Informationen zu .NET findet man unter [Ms05].

2.6.2.3 Datenbanken

Nahezu alle Web-Anwendungen sind auf die dauerhafte Speicherung ihrer Daten angewiesen. Hierzu bieten sich Datenbanken an. Meist werden heutzutage relationale Datenbanksysteme verwendet. Dabei werden die Daten in Tabellen gespeichert, wobei den einzelnen Spalten unterschiedliche Datentypen zugeordnet werden können. Zur Abfrage von Daten steht die standardisierte Structured Query Language (SQL) zur Verfügung. Damit sind umfangreiche Abfragen auch über mehre Tabellen möglich. Außerdem ist, sofern man wirklich nur den standardisieren Sprachumfang von SQL verwendet, ein Austausch des Datenbanksystems relativ einfach möglich. Ein Nachteil ergibt sich allerdings bei ihrem Einsatz mit objektorientierten Programmen. Hier kommt es zu einem Strukturbruch, da Objektstrukturen mit Vererbung und Aggregation auf Tabellen abgebildet werden müssen, was zwar möglich, jedoch semantisch nicht eindeutig ist.

Daneben existieren objektorientierte Datenbanksysteme. Hier können sowohl die Methoden und Attribute als auch die Beziehungen zwischen den Objekten gespeichert werden. Rein objektorientiere Datenbanksysteme konnten sich allerdings bis jetzt nicht durchsetzen. So geht die aktuelle Entwicklung in Richtung objektrelationaler Datenbanksysteme, d.h. der Erweiterung des relationalen Datenmodells und der Abfragesprache SQL um Objektorientierung (siehe z.B. [RV03]).

Um den Strukturbruch zwischen den häufig eingesetzten relationalen Datenbanksystemen und der Anwendung zu verringern, werden meist so genannte Persistenzdienste oder Persistenzschichten eingesetzt. Hier wird ein Mapping zwischen Anwendungsobjekten und Tabellen durchgeführt, so dass Objekte immer einer Zeile in einer Tabelle innerhalb der Datenbank entsprechen. Solche Dienste steuern dann automatisch den Abgleich der Daten in den Objekten mit den Daten in der Datenbank. Bei der Entwicklung der Anwendung kommt man dann nicht mehr mit der Datenbank in Berührung, sondern benutzt nur noch die Persistenzschicht. EJB-Container bieten z.B. einen solchen Persistenzdienst an.

2.6.2.4 Andere Technologien

Neben den eben vorgestellten Technologieplattformen existieren noch eine Reihe weiterer Technologien, wie z.B. CGI (Common Gateway Interface), PHP und ASP (Active Server Pages), auf die hier nicht weiter eingegangen werden soll, da sie eher für kleinere bis mittlere Web-Anwendungen gedacht sind, in denen der Einsatz von Frameworks meist nicht lohnt.

2.7 Architekturen

Durch den Einsatz eines Frameworks wird in der Regel auch die Architektur der Anwendung festgelegt. Im Folgenden werden die wichtigsten Architektur-Ansätze für Web-Anwendungen vorgestellt.

2.7.1 Schichten-Architektur

Bei der Strukturierung von Software-Anwendungen hat sich die Schichtenbildung (Abbildung 2.8) als effektives Mittel etabliert. Dabei stellt jede Schicht eine Menge von Diensten bereit. Auf diese kann jeweils nur von der direkt darüber liegenden Schicht zugegriffen werden. Die Komplexität wird dabei von jeder Schicht gekapselt und ist von außen nicht sichtbar. [St02]

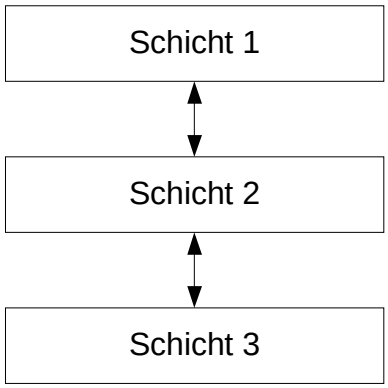

Abbildung 2.8: Schichten-Architektur

Eine solche Schichtenbildung bietet eine Reihe von Vorteilen:

- Die Sichten sind sowohl in der Entwicklung als auch im Betrieb unabhängig voneinander, da sie Daten nur über definierte Schnittstellen austauschen.
- Die Implementierung einer Schicht kann ausgetauscht werden, solange sie die selben Schnittstellen und Dienste anbietet.
- Die Schichtenbildung ist trotz ihrer Leistungsfähigkeit ein sehr leicht verständliches Konzept.

Dem stehen natürlich auch Nachteile gegenüber:

- Die Schichtenbildung wirkt sich negativ auf die Performance des Systems aus, da z.B. bei einer Anfrage mehrere Schichten durchlaufen werden müssen. Je mehr Schichten es gibt, desto stärker tritt dieser Aspekt in den Vordergrund. Der Grad der Strukturierung mittels Schichten muss demnach immer mit dem Aspekt der Performanz abgewogen werden.
- Die getrennte Implementierung der einzelnen Schichten erfordert einen Mehraufwand, da jede Funktionalität als Dienst bereit gestellt werden muss.

Eine Einteilung einer Anwendung in Schichten dient meist zur groben Strukturierung einer Anwendung in eine geringe Anzahl von Schichten. Diese Schichten können dann ihrerseits wieder verschiedene Architekturen besitzen. Ein Beispiel für eine solche Schichtenbildung stellt das in Abschnitt 2.6.2.1 vorgestellte Multi-Tier-Modell der J2EE-Plattform dar.

2.7.2 Model-View-Controller-Architektur

Die Model-View-Controller-Architektur (MVC-Architektur, siehe auch Abbildung 2.9) ist ein, in Web-Anwendungen häufig vorkommendes, Entwurfsmuster und dient der Entkoppelung von Datenhaltung (Model), Darstellung (View) und

Ablaufsteuerung (Controller) eines Systems. Sie stammt aus der Entwicklung von Smalltalk80-Anwendungen und findet seitdem Verwendung bei der Entwicklung von graphischen Benutzeroberflächen. [St02, Ga01]

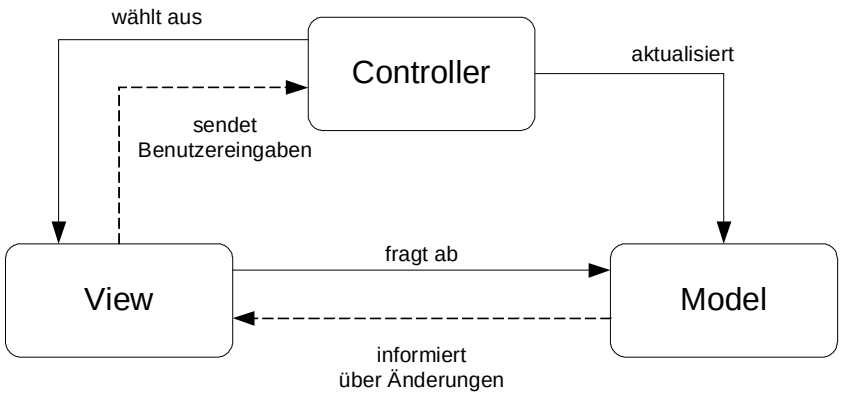

Abbildung 2.9: Model-View-Controller-Architektur [SSJ02]

Die drei Bestandteile haben folgende Funktion [SSJ02]:

- Das *Model* definiert die Funktionalität der Anwendung und kapselt deren Zustand. Das Model benachrichtigt den View über Änderungen und definiert eine Schnittstelle, mit welcher der View den aktuellen Status abfragen kann und der Controller auf die Funktionen des Models zugreifen kann.
- Der *View* dient zur Darstellung der Daten des Models. Er fragt die Daten vom Model ab und definiert, wie diese darzustellen sind. Er aktualisiert die Darstellung der Daten, wenn sich das Model ändert. Weiterhin nimmt er Eingaben des Benutzer entgegen und leitet sie zum Controller weiter.
- Der *Controller* steuert das Verhalten der Anwendung. Hierzu werden Benutzeraktionen auf Modeländerungen abgebildet und der entsprechende View zur Darstellung des Models gewählt.

Die Aufteilung der Verantwortlichkeiten auf Model, View und Controller dient, ähnlich wie bei einer Schichten-Architektur, dem Ziel der Strukturierung der Anwendung.

Durch seine erfolgreiche Verwendung bei der Erstellung von graphischen Benutzeroberflächen wurde auch versucht, das MVC-Muster bei der Erstellung von Web-Anwendungen einzusetzen. Der Großteil der im J2EE-Umfeld verfügbaren Frameworks basiert auf diesem Muster, wobei jedoch auch unterschiedliche Schwerpunkte gesetzt werden.

Aufgrund des verteilten Charakters von Web-Anwendungen und der Funktionsweise des zugrunde liegenden HTTP-Protokolls [W3C99] (siehe auch Abschnitt 2.2) treten jedoch bei der Umsetzung einige Besonderheiten auf. Die Model-View-Controller-Architektur würde sich dann, wie in Abbildung 2.10 konzeptionell dargestellt, verändern:

- Der *View* kann nicht vom Model nach Bedarf aktualisiert werden. Vielmehr kann er erst bei der nächsten Anfrage des Web-Clients das Model abfragen.

- Der *Controller* nimmt in Web-Anwendungen die Benutzereingaben direkt an, da der View nicht direkt auf dese reagieren kann.

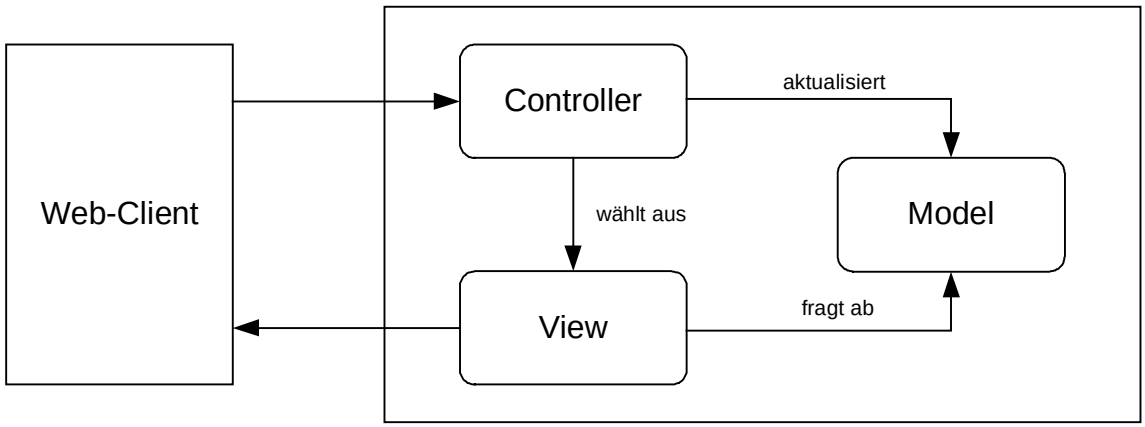

Abbildung 2.10: Model-View-Controller-Architektur für Web-Anwendungen

Häufig werden im Zusammenhang mit dem MVC-Konzept auch die Begriffe „Model 1“ und „Model 2“ verwendet. Diese stammen aus frühen Versionen der JSP-Spezifikation und beschreiben grundlegende Nutzungsmuster für die JSP-Technologie. „Model 1“ beschreibt dabei die alleinige Nutzung von JavaServer Pages als View und Controller gleichzeitig (siehe Abbildung 2.11). Dies bringt jedoch z.B. den Nachteil mit sich, dass die Trennung zwischen Layout und Anwendungslogik verloren geht, da der Code innerhalb von Scriptlets in die JSP integriert werden muss. Dies wird bei größeren Anwendungen schnell unübersichtlich und eignet sich daher maximal für kleinere Anwendungen.

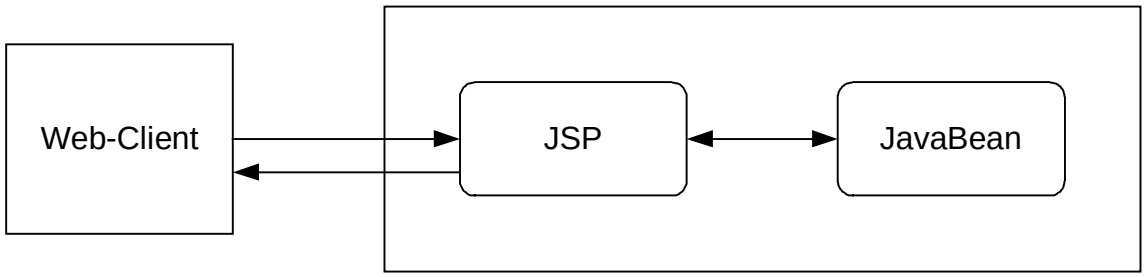

Abbildung 2.11: „Model 1“-Architektur [Ses99]

Dagegen ist „Model 2“ eine strikte Umsetzung der MVC-Architektur in J2EE und schlägt den Einsatz von JSP als View, Servlets als Controller und JavaBeans als Model vor (siehe Abbildung 2.12). „Model 2“ und MVC werden heute (fälschlicherweise) fast synonym verwendet. [SSJ02]

Mit der Verwendung von Servlets an Stelle von JSPs als Controller gehen weitere Vorteile einher. Durch die Fähigkeit von Servlets, mit ihrem Ausführungskontext Daten auszutauschen, ist es möglich, Objekte Request-übergreifend verfügbar zu machen. Der Lebenszyklus von Ressourcenobjekten ist damit nicht mehr zwingend mit der Anfragebearbeitung gekoppelt. Damit ist eine effizientere Nutzung von Ressourcen innerhalb der Applikationslogik, wie z.B. EJBs und Datenbankverbindungen, möglich.

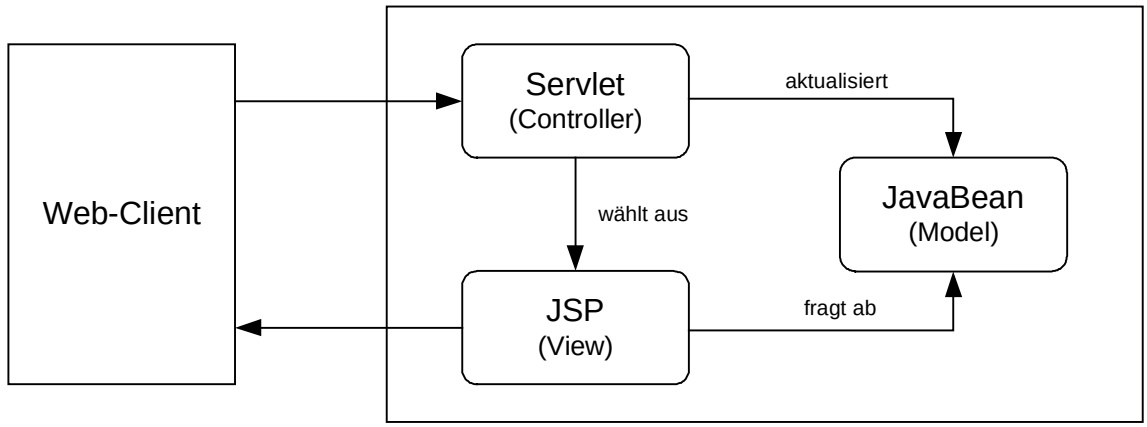

Abbildung 2.12: „Model 2"-Architektur [Ses99]

2.7.3 Pipeline-Architektur

Die Pipeline-Architektur ist besonders für Anwendungen geeignet, bei denen einzelne Arbeitschritte autonom und in einer bestimmten Reihenfolge abgearbeitet werden. Eine Pipeline besteht dabei aus einer Menge von Pipes und Filtern. Die Pipes stellen dabei Datenströme dar und verbinden die Ein- und Ausgänge von Filtern. Innerhalb von Filtern können die Datenströme transformiert werden. Die einzelnen Filter arbeiten dabei völlig unabhängig voneinander. Hierdurch sind sie sehr leicht wieder zu verwenden und die Pipelinestruktur ist leicht veränderbar.

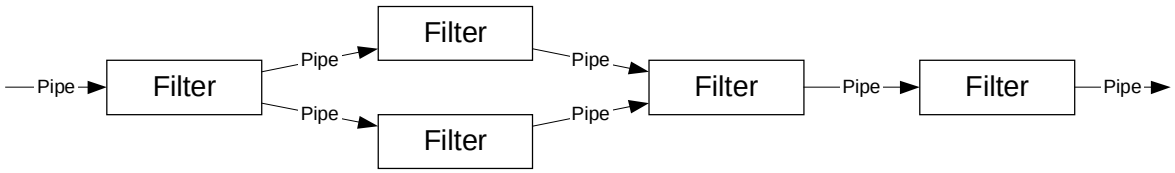

Abbildung 2.13: Pipeline-Architektur

Kapitel 3

Frameworks für Web-Anwendungen

In diesem Kapitel wird eine Klassifikation für Web-Frameworks entwickelt. Dazu muss zuvor geklärt werden, warum sich Frameworks für die Entwicklung von Web-Anwendungen eignen, wie sie definiert sind und wie sie sich von anderen Konzepten der Softwarewiederverwendung unterscheiden. Danach werden typische Aufgabenfelder von Frameworks in der Entwicklung von Web-Anwendungen beschrieben. Im Hauptabschnitt wird dann anhand von geeigneten Kriterien eine Klassifikation für Web-Frameworks aufgebaut. Insbesondere werden auch die Abhängigkeiten zwischen den einzelnen Kriterien untersucht.

3.1 Einleitung

Bei der Implementierung von Web-Anwendungen stößt man immer wieder auf die selben Aufgaben. Dies können Dinge sein wie z.B. die Lokalisierung von Texten, die Einbindung von Datenbanksystemen, die Verfolgung von Benutzersitzungen, die Darstellung von dynamischen Inhalten oder die Transaktionsverwaltung. Alle diese Probleme haben gemeinsam, dass sie in Web-Anwendungen gelöst werden müssen. Es liegt nahe, einmal entwickelte Lösungen wieder zu verwenden. Will man nicht nur einzelne Aspekte, sondern den gesamten Entwurf einer Web-Anwendung wieder verwenden, bieten sich hierfür Frameworks an.

Bei der Entwicklung von Web-Anwendungen ist die Erstellung der Präsentationsschicht mit dem größten Aufwand verbunden. Hier muss die Anwendung in den Request-Response-Zyklus des Web-Servers eingebunden werden, Benutzereingaben validiert und weitergegeben sowie der Zustand der Nutzersitzung zwischen einzelnen Anfragen gespeichert werden. Da an diese Schicht bei fast allen Web-Anwendungen dieselben Anforderungen gestellt werden, bietet sich hier die Wiederverwendung durch Frameworks an.

Es gibt auch noch andere Einsatzgebiete für Frameworks in Web-Anwendungen, beispielsweise zur Realisierung eines Persistenzdienstes (siehe Abschnitt 2.6.2.3) oder zum Zugriff von Ressourcen auf verteilten Rechnern. Diese werden hier aber nicht weiter betrachtet, da sie nicht spezifisch für Web-Anwendungen sind, sondern ebenso auch in anderen Anwendungen Einsatz finden.

Auch fördern Frameworks einen „guten“ Softwareentwurf, indem sie die Verwendung von Entwurfsmustern fördern oder gar vorschreiben. Damit ist eine erleichterte Wartung des Codes gegeben. Häufig werden Web-Frameworks aber

auch so populär, dass man von einem Quasi-Standard sprechen kann. Dies ermöglicht auch projektfremden Entwicklern, sich schnell in ein Projekt einzuarbeiten, wenn sie bereits mit dem verwendeten Framework vertraut sind.

Ein Vorteil von Frameworks liegt auch in ihrer Anpassungsfähigkeit. Dadurch können sie nicht nur als Lösung für ein Problem, sondern unmittelbar für eine ganze Klasse von Problemen, dienen. Als Ergebnis können damit Anwendungen nicht nur schneller entwickelt werden, sondern sie haben auch ähnliche Strukturen. Sie werden somit konsistenter und einfacher wartbar. Allerdings verliert man dadurch auf der anderen Seite einige kreative Freiheit, da viele Entwurfsentscheidungen bereits festgelegt sind.

Frameworks entstehen meist aus Erfahrungen, die aus mehreren Entwicklungen innerhalb eines ähnlichen Problemkontextes stammen. Sie enthalten somit einen großen Erfahrungsschatz, den der Anwender eines Frameworks unmittelbar nutzen kann.

Eine weitere Reihe von Produkten, die im Zusammenhang mit der Programmierung von Web-Anwendungen stehen, sind so genannte Templating Engines. Dabei handelt es sich um Bibliotheken zur dynamischen Generierung von Ausgabeseiten. Hier können statische Vorlagen, wie z.B. HTML-Seiten, durch Anweisungen erweitert werden, welche zur Laufzeit dynamische Inhalte in diese Seite einbinden. Auch wenn Templating Engines nicht als vollwertige Frameworks angesehen werden können (siehe nächsten Abschnitt), so verwenden jedoch viele Frameworks eigene oder existierende Templating Engines zur dynamischen Generierung der Oberfläche.

Daneben existieren noch eine Reihe von Portal-Frameworks, die meist auf Frameworks für Web-Anwendungen aufbauen und als Grundlage für die Erstellung von Portalen dienen (siehe Abschnitt 2.5.3). Da hier das zentrale Augenmerk auf der Integration statt auf der Entwicklung von Web-Anwendungen liegt, werden sie ebenfalls nicht weiter in die folgenden Betrachtungen einbezogen.

In den letzten Jahren hat sich J2EE als führende Plattform für die Entwicklung von komplexen Web-Anwendungen etabliert. Aus diesem Grund basieren die meisten Frameworks auf dieser Plattform. Diese Arbeit beschränkt sich deshalb und aus Gründen des Umfanges auf Frameworks für die J2EE-Plattform. Wo dies zu Vergleichszwecken sinnvoll erscheint, werden jedoch auch andere Frameworks in die Betrachtung einbezogen.

Wie man die J2EE-Technologien anwendet, ist beispielsweise in den Blueprints [Sun05c] von Sun beschrieben. Die dort vorgestellten Entwurfsmuster geben Lösungsansätze für die Praxis an, stellen selber jedoch keine Software dar, die man sofort verwenden könnte. Die Architektur vieler Web-Frameworks basiert jedoch meist auf diesen Entwurfsmustern.

3.2 Definition

Eine der wichtigsten Arten von Softwarewiederverwendung ist die Wiederverwendung des Entwurfes. Eine abstrakte Klasse ist der Entwurf für eine

einzelne Klasse. Eine Menge von Klassen kann dazu verwendet werden, einen gesamten Entwurf zu beschreiben. So beschreibt die Unterteilung einer Anwendung in Klassen, die jeweiligen Zuständigkeiten, die Zusammenarbeit der Klassen und Objekte sowie der Kontrollfluss den Entwurf dieser Anwendung. Diese Idee der Entwurfswiederverwendung beschreibt das Grundkonzept von Frameworks. In der Literatur finden sich zwar teilweise unterschiedliche, in der Grundaussage jedoch übereinstimmende Definitionen. In [Ga01] findet man z.B.:

> *„Ein **Framework** besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wieder verwendbaren Entwurf für eine bestimmte Klasse von Software darstellen."*

Ein Framework bestimmt somit den Entwurf einer Anwendung oder Teilen davon. Oberstes Ziel ist dabei die Wiederverwendung dieses Entwurfes, also die Erstellung eines Entwurfes für eine ganze Klasse von Problemen.

Ein weiterer wichtiger Aspekt bei den hier betrachteten Frameworks ist, dass (anders als bei anderen Konzepten der Softwarewiederverwendung) das Framework die eigentliche Anwendungslogik aufruft und nicht umgekehrt. Daher soll in dieser Arbeit der Begriff Framework wie folgt eingeschränkt werden:

> *„Ein **Framework** besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wieder verwendbaren Entwurf für eine bestimmte Klasse von Software darstellen wobei das Framework der Teil der Anwendung ist welcher die eigentliche Anwendungslogik aufruft."*

3.2.1 Andere Konzepte der Softwarewiederverwendung

Frameworks sind eine von vielen Konzepten zur Softwarewiederverwendung. Andere Konzepte sollen hier kurz gegenüber Frameworks abgegrenzt werden.

3.2.1.1 Entwurfsmuster

Balzert definiert Entwurfsmuster wie folgt [Ba00]:

> *„Ein **Entwurfsmuster** (design pattern) gibt eine bewährte generische Lösung für ein häufig wiederkehrendes Entwurfsproblem an, das in bestimmten Situationen auftritt."*

Gamma und andere haben aufbauend auf diesen Gedanken in [Ga01] den Grundstein zur Lösung von Entwurfsproblemen mit Entwurfsmustern gelegt. Viele der von ihnen identifizierten und dort beschriebenen Entwurfsmuster werden heute in Frameworks verwendet. Gamma gibt dabei drei Unterscheidungsmerkmale zwischen Frameworks und Entwurfsmustern an [Ga01]:

- *Entwurfsmuster sind abstrakter als Frameworks*: Während letztere als Code dargestellt werden können, ist dies bei Entwurfsmustern nur beispielhaft möglich. Frameworks können in konkreten Programmiersprachen aufgeschrieben und somit ausgeführt und direkt wieder verwendet werden. Dies ist bei Entwurfsmustern nicht der Fall, sie müssen bei jeder Verwendung erneut implementiert werden.

- *Entwurfsmuster sind weniger spezialisiert als Frameworks*: Frameworks haben immer einen bestimmten Anwendungsbereich. Entwurfsmuster können hingegen in nahezu allen Anwendungsbereichen verwendet werden.
- *Entwurfsmuster sind kleiner als Frameworks*: Während Frameworks meist ein oder mehrere Entwurfsmuster enthalten, enthalten Entwurfsmuster niemals ein Framework.

3.2.1.2 Komponenten

Ähnlich der in der Industrie üblichen Vorgehensweise (Endprodukte aus vorgefertigten Komponenten zusammenzusetzen) wird in der komponentenbasierten Software-Entwicklung versucht, durch Verwendung von vorgefertigten Softwarebausteinen die Herstellung von Anwendungen einfacher, schneller und preisweiter zu gestalten. Diese Softwarebausteine werden als Komponenten bezeichnet:

> *„Eine **Komponente** (component) ist ein abgeschlossener, binärer Software-Baustein, der eine anwendungsorientierte, semantisch zusammengehörende Funktionalität besitzt, die nach außen über Schnittstellen zur Verfügung gestellt wird.“ [Ba00]*

Komponenten sind von ihrem Umfang her zwischen Anwendungen und Klassen einzuordnen. Sie sind als Softwarebausteine, kleiner als Anwendungen, kapseln jedoch eine höhere Komplexität und Funktionalität als eine einzelne Klasse. Ein Spezialfall von Komponenten sind PlugIns, die als Erweiterung in eine bestehende Anwendung eingeklinkt werden können. Sie implementieren bekannte Erweiterungsschnittstellen einer Anwendung und stellen dadurch zusätzliche Funktionen zur Verfügung.

In Beziehung zu Frameworks stehen Komponenten darin, dass Frameworks meist auf Komponenten zugreifen, um sie in den Anwendungskontext einzubeten. Das Framework greift zur Erledigung einzelner Teilaufgaben auf Komponenten zurück und steuert ihre Zusammenarbeit [BBE95].

3.2.1.3 Klassenbibliotheken

Klassenbibliotheken bieten die Möglichkeit, Wiederverwendung im Kleinen zu realisieren. Sie lassen sich wie folgt definieren:

> *„Eine **Klassenbibliothek** ist eine organisierte Software-Sammlung, aus der der Entwickler nach Bedarf Einheiten verwendet.“ [Ba00]*

Eine Klassenbibliothek ist meist eine Menge von wieder verwendbaren Klassen, die entworfen wurden, um nützliche und allgemeine Funktionalität zur Verfügung zu stellen. Bei Klassenbibliotheken steht im Gegensatz zu Frameworks nicht die Entwurfs- sondern die Codewiederverwendung im Vordergrund. Anders als bei Frameworks wird die Funktionalität der Klassenbibliotheken stets von der Anwendung aufgerufen, nicht umgekehrt.

3.2.2 White-Box- und Black-Box-Frameworks

Ein wichtiges Merkmal von Frameworks ist die Umkehrung der Steuerung (*Inversion of Control*) zwischen der Anwendung und der Software die sie verwendet. Wenn in einer Anwendung Funktionalitäten einer Klassenbibliothek, einer Komponente usw. aufgerufen werden, schreibt der Entwickler den Hauptteil der Anwendung selbst und ruft den Code auf, der wieder verwendet werden soll. Bei Frameworks ist dies meist anders: Hier stellt das Framework den Hauptteil der Anwendung dar und steuert den Ablauf der Gesamtanwendung. Diese Umkehrung der Steuerung gibt dem Framework die Möglichkeit, als flexibles und erweiterbares Gerüst für Anwendungen zu dienen. Je nach Art und Weise, wie Frameworks vom Entwickler angepasst und erweitet werden müssen, kann man zwischen White-Box- und Black-Box-Frameworks unterscheiden [JF88, BBE95]:

White-Box-Frameworks bestehen u.a. aus einer Reihe von abstrakten Klassen. Der Entwickler muss dann die abstrakten Methoden dieser Klassen in konkreten Unterklassen überschreiben. Das Grundprinzip der Wiederverwendung von White-Box-Frameworks ist also die Spezialisierung durch Konkretisierung, d.h. Anpassungen erfolgen durch Ersetzen allgemeiner Teile des Frameworks durch anwendungsspezifische. Dazu ist allerdings ein tiefgehendes Verständnis der Architektur des Frameworks notwendig. Deswegen werden sie White-Box-Frameworks genannt. Das Hauptproblem bei solchen Frameworks ist, dass sehr viele Unterklassen erzeugt werden müssen. Obwohl diese meist sehr einfach sind, macht allein ihre große Anzahl das Verständnis der Anwendung nicht einfach.

Einen anderen Ansatz verfolgen *Black-Box-Frameworks*. Diese bieten bereits fertige Klassen an, von denen der Entwickler Instanzen bilden und diese konfigurieren kann. Sie können verwendet werden, ohne dass man versteht, wie die Objekte intern zusammenspielen. Es ist nur ein grundlegendes Verständnis für das Gesamtframework notwendig. Im Gegensatz zu White-Box-Frameworks ist der Aufwand zur Nutzung dadurch deutlich geringer, allerdings schränkt dies auch die Anpassbarkeit stark ein.

Beispiele für Black-Box-Frameworks sind häufig Frameworks zur Erstellung von Benutzeroberflächen (User Interfaces). Hier wird eine Menge von Dialog-Elementen zur Verfügung gestellt. Diese können dann z.B. über einen grafischen Editor platziert und konfiguriert werden. Um die Darstellung und die Initialisierung muss man sich nicht mehr kümmern.

In diesem Sinne stehen Black-Box-Frameworks in engem Zusammenhang mit Komponenten. So stellt das Framework die Umgebung zur Verwendung der Komponenten dar und definiert Schnittstellen, welche die Zusammenarbeit mit den Komponenten ermöglicht. Die komponentenbasierte Entwicklung kann in diesem Sinne als Weiterentwicklung des Black-Box-Ansatzes von Frameworks verstanden werden. [BBE95]

Der Übergang von White-Box- zu Black-Box-Frameworks ist meist fließend, und viele Frameworks sind eine Mischung aus beiden Ansätzen. Bei der Verwendung von White-Box-Frameworks dominiert jedoch die Programmierung, wohingegen bei Black-Box-Frameworks eher die Konfiguration im Vordergrund steht. Mit steigendem Reifegrad tendieren Frameworks meist zum Black-Box-Ansatz.

3.3 Anforderungen an Web-Frameworks

Im folgenden Abschnitt werden die wichtigsten Aufgabenfelder, welche bei der Entwicklung von Web-Anwendungen auftreten und von Web-Frameworks zu lösen sind, aufgezählt:

- *Einbindung in den Anfrage/Antwort-Zyklus des Web-Servers:* Eine der ersten Aufgaben bei der Entwicklung von Web-Anwendungen ist die Einbindung in den Anfrage/Antwort-Zyklus des Web-Servers: Die Anfrage des Web-Clients muss entgegengenommen, die entsprechenden Aktionen ausgelöst sowie das Ergebnis zurück an den Web-Client geschickt werden.
- *Formulare und Validierung:* In den „Grundsätzen der Dialoggestaltung“ [ISO95] wird gefordert: „Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand durch den Benutzer erreicht werden kann.“ D.h. Daten eines Formulars müssen auf Plausibilität, fehlende oder unvollständige Eingaben geprüft, bevor sie abgesendet werden. Fehlermeldungen müssen für den Benutzer klar erkenntlich sein.
- *Trennung von Code und Layout:* Einen sehr wichtigen Beitrag zur Wartbarkeit von Web-Anwendungen trägt die Trennung von Code und Layout bei. Durch diese nicht-funktionale Anforderung können Web-Designer das Layout einer Web-Anwendung ohne größere Kenntnisse der Architektur überarbeiten und anpassen. Auf der anderen Seite ist der Programmcode zur Steuerung der Anwendung zentral in Klassen zusammengefasst und nicht weit verteilt im Layout versteckt. So lassen sich Fehler schneller finden und Änderungen z.B. in der Dialogfolge schneller umsetzen.
- *Verfolgung und Speicherung von Benutzersitzungen:* Ein zentraler Punkt von Web-Anwendungen ist die Verfolgung von Benutzersitzungen. Da Web-Anwendungen Mehrbenutzeranwendungen sind und normalerweise auf dem zustandslosen HTTP-Protokoll basieren, tritt das Problem auf, jeder Anfrage an den Web-Server einen Nutzer zuzuordnen. Dies ist wichtig, um Zustände zwischen Anfragen zu speichern (beispielsweise den Inhalt des Warenkorbes in einer Shop-Anwendung).
- *Mehrsprachigkeit:* Da Web-Anwendungen häufig einem großen Nutzerkreis zugänglich gemacht werden sollen, ist auch die Mehrsprachigkeit der Anwendung eine häufige Anforderung: Dialoge, Texte, Meldungen und Beschriftungen sollen sich möglichst ohne großen Aufwand in mehreren Sprachen darstellen lassen.

3.4 Klassifikationskriterien

In diesem Abschnitt werden verschiedene Unterscheidungsmerkmale dahingehend untersucht, welchen Beitrag sie zur Klassifikation von Frameworks leisten können. Diese Kriterien leiten sich einerseits aus den formulierten Anforderungen und andererseits aus möglichen Architekturen und dem verwendeten Programmiermodell ab.

Da Frameworks ein Mittel zur Entwurfswiederverwendung sind, geben sie implizit auch einen Teil der Architektur einer Anwendung vor. Die in Abschnitt 2.7

vorgestellten Architekturansätze stellen somit auch ein Kriterium zur Klassifikation von Frameworks dar, welches hier nicht noch einmal dargestellt werden soll.

3.4.1 Anwendungssteuerung

Unter der Anwendungssteuerung [Se03, Wa04] versteht man, wie die Anwendung auf Eingaben reagiert und diese weiter verarbeitet. Hier haben sich zwei grundsätzliche Ansätze entwickelt: der aktionsgesteuerte und der ereignisgesteuerte Ansatz. Dieses Unterscheidungsmerkmal ist das wichtigste, da es einerseits den Entwurf grundlegend beeinflusst und andererseits fast alle anderen Merkmale von ihm abhängen.

Aktionsgesteuerte Frameworks orientieren sich sehr stark am Request-Response-Zyklus von Web-Anwendungen. Ihre Architektur lehnt sich deshalb meist nah an das „Model 2“ an (siehe Abschnitt 2.7.2). Aktionsgesteuerte Frameworks setzen also eine MVC-Architektur voraus. Der Controller dient hier als zentrale Instanz zur Annahme von Anfragen und führt zentrale Aufgaben, wie die Validierung von Parametern, durch. Danach ruft er je nach Anfrage verschiedene Aktionen auf. Diese enthalten die eigentliche fachliche Logik und greifen auf das Model zu. Sie werden im Normalfall vom Programmierer durch Ableiten von abstrakten Klassen erstellt. Hier zeigt sich auch eindeutig der White-Box-Charakter (siehe Abschnitt 3.2.2) von aktionsgesteuerten Frameworks mit den entsprechenden Vor- und Nachteilen. Nachdem die Aktion abgearbeitet wurde, wählt der Controller den entsprechenden View aus. Dieser erstellt die Antwort und schickt sie an den Web-Client zurück. Damit ist der Verarbeitungszyklus für die Anfrage beendet. Abbildung 3.1 stellt diesen Ablauf noch einmal schematisch dar.

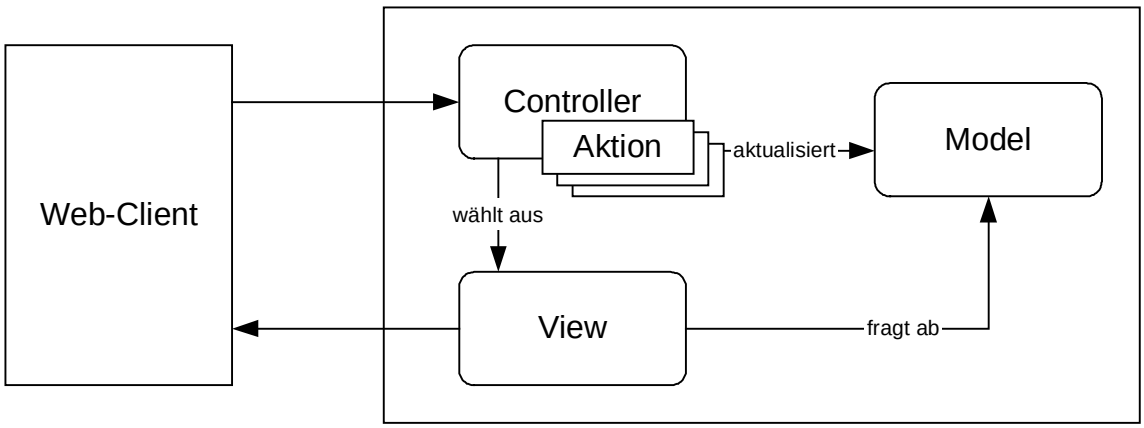

Abbildung 3.1: Architektur von aktionsgesteuerten Frameworks

Ereignisgesteuerte Frameworks verfolgen hingegen einen ganz anderen Ansatz. Hier wird versucht, den für Web-Anwendungen typischen Request-Response-Zyklus vor dem Anwendungsentwickler zu verbergen. In Anlehnung an die Entwicklung von Benutzeroberflächen von Desktop-Anwendungen werden die Benutzerinteraktionen auf Ereignisse von Komponenten abgebildet. Jede dargestellte Seite wird dabei als eine Menge von Komponenten aufgefasst, welche mit serverseitigen Objekten verknüpft sind. Demzufolge ist beispielsweise ein Klick auf einen Button ein Ereignis für die Button-Komponente und kann auf Serverseite entsprechend behandelt werden. Die Stärken dieses Ansatzes sind, dass sich die Vorteile der objektorientierten Programmierung – Kapselung von Objekten mit

Zustand und Verhalten – auf die Programmierung der Benutzeroberfläche von Web-Anwendungen überträgt. Auch bietet die Verwendung von Komponenten mehr Möglichkeiten zur Wiederverwendung. Meist werden sogar alle benötigten Komponenten, wie Links, Buttons, Eingabefelder usw., vom Framework bereit gestellt. Sie müssen vom Entwickler nur noch entsprechend verwendet werden. Hier zeigt sich wieder eine Parallele zur Entwicklung von graphischen Benutzeroberflächen. Und ähnlich wie dort handelt es sich auch bei ereignisgesteuerten Frameworks um einen Black-Box-Ansatz (siehe dazu Abschnitt 3.2.2). Auch lässt sich feststellen, dass der ereignisorientierte Ansatz dem ursprünglich in Smalltalk formulierten MVC-Ansatz viel näher kommt, als dies beim „Model 2“ der Fall ist. In Abbildung 3.2 wird dies noch einmal schematisch dargestellt.

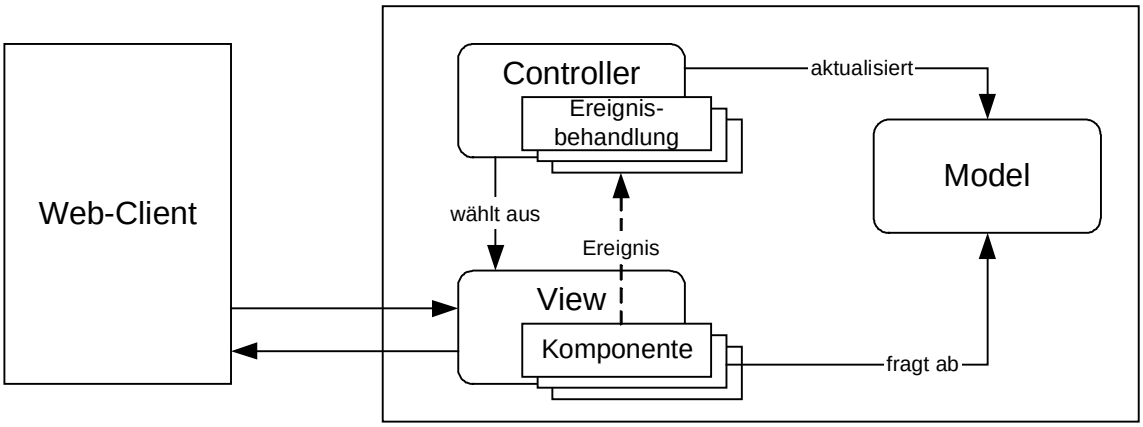

Abbildung 3.2: Architektur von ereignisgesteuerten Frameworks

3.4.2 User-Interface-Komponenten

Bei der Erstellung von graphischen Benutzeroberflächen (Web-Oberflächen einmal ausgenommen), z.B. mit dem Swing-Framework, sind wieder verwendbare Komponenten eine Selbstverständlichkeit. Bis auf seltene Fälle käme man kaum auf die Idee, einen JTable oder einen JTree neu zu programmieren. Dies ist im Bereich von Web-Anwendungen ganz anders. Dies mag einerseits an der hohen Individualität von Web-Anwendungen liegen. Ein weiterer Grund ist sicher darin zu suchen, dass die meisten Web-Technologien dies nur sehr schlecht unterstützen.

Zwar gibt es Ansätze, wie z.B. JSP Tag-Libraries – diese reichen aber nicht weit genug, da komplexe Komponenten nicht nur aus reinen Präsentationen bestehen, sondern auch sehr spezifisches Verhalten implementieren. Will man eine Seite aus verschiedenen Komponenten mit jeweils eigenständigem Verhalten zusammensetzen, so muss es ein Verfahren geben, das Benutzeraktionen für bestimmte Komponenten an diese weiterleitet. Damit gelangt man wieder zur eben beschriebenen ereignisorientierten Anwendungssteuerung.

Dies ist allerdings nicht verwunderlich, da ereignisgesteuerte Frameworks typischerweise einen Black-Box-Ansatz verfolgen, welcher wiederum die Grundlage zur komponentenbasierten Entwicklung ist [BBE95]. In diesem Sinne sind die Begriffe „ereignisgesteuert“ und „komponentenbasiert“ für Web-Frameworks als synonym zu betrachten. Im Folgenden wird daher der Begriff

ereignisgesteuert auch immer die Verwendung von User-Interface-Komponenten (UI-Komponenten) einschließen.

3.4.3 Darstellung

Eine große Bedeutung bei Web-Anwendungen hat die Darstellung. Um sie gegenüber dem Endnutzer bekannt zu machen, wird häufig viel Geld in das Marketing investiert. Deshalb sind der Wiedererkennungswert und die Individualität einer Web-Anwendung im Gegensatz zu anderen Anwendungen von großer Bedeutung. Auf der anderen Seite haben solche Web-Anwendungen eine große Lebensdauer, deshalb spielen auch nichtfunktionale Anforderungen, wie die Wartbarkeit, eine große Rolle. Ein Ansatz dafür ist beispielsweise die Trennung von Code und Layout. Sie ermöglicht einerseits eine Aufgabenteilung zwischen Programmierung und Web-Designer und andererseits eine bessere Strukturierung der Anwendung, da der Code an zentraler Stelle liegt und nicht verteilt im Layout versteckt ist. Um diese Anforderungen zu erfüllen, werden von verschiedenen Frameworks sehr verschiedene Ansätze verfolgt. Im Folgenden sollen die wichtigsten Grundtypen vorgestellt werden:

Der am häufigsten verwendete Ansatz ist die Verwendung von so genannten *Templates*. Sie sind am flexibelsten und ermöglichen die Ausschöpfung des vollen Potentials von HTML. Aber auch beliebige andere Seitenbeschreibungssprachen, wie beispielsweise *Wireless Markup Language* (WML), sind einsetzbar. Dazu werden in die darzustellende Seite neben der eigentlichen Seitenbeschreibung, welche im weiteren Sinne als Vorlage (Template) für die Seite dient, auch spezielle Code-Fragmente oder Anweisungen eingebettet. Diese werden vor dem Senden an den Web-Client ausgewertet und können so z.B. Inhalte dynamisch einfügen. Der Web-Client erhält dann die rein statische Seitenbeschreibung mit dynamisch eingefügten Inhalten übermittelt. Viele Technologien, wie z.B. PHP, ASP und JSP arbeiten nach diesem Ansatz (siehe Abschnitt 2.6.2).

Ein anderer Ansatz besteht aus der vollständigen *Generierung* der Seitenbeschreibung. Dieser bietet sich besonders bei ereignisgesteuerten Frameworks an, da sich hier jede Seite aus einer Reihe von Komponenten zusammensetzt. Für jede dieser Komponenten ist beschrieben, wie sie in einer konkreten Seitenbeschreibungssprache darzustellen ist. Bei strikter Anwendung dieses Ansatzes ist es so z.B. möglich, Anwendungen zu schreiben, ohne jemals eine Zeile HTML zu verwenden. Auch kann ohne großen Aufwand die Seitenbeschreibungssprache einer ganzen Anwendung geändert werden. Dazu muss nur die Generierung der einzelnen Komponenten zentral geändert werden. Dies ist besonders bei Anwendungen mit verschiedenen Endgeräten (Multi-Channel-Anwendungen) ein interessanter Aspekt. Allerdings erweist sich dieser Ansatz als problematisch, wenn es um die individuelle Gestaltung der Oberfläche geht, da hier nur ein indirekter Zugriff (über die Komponenten) auf die Seitenbeschreibung möglich ist.

Ein weiterer Ansatz ist die Generierung der Seitenbeschreibung durch *Transformation*. Hier liegt die Ausgangsrepräsentation als XML vor, und wird beispielsweise durch ein extern anpassbares XSLT-Stylesheet in das Zielformat umgewandelt. Dies erlaubt große Flexibilität durch leichte Anpassbarkeit an verschiedene Ausgabeformate. Allerdings ist das Erstellen der Transformationsregeln (XSLT-Stylesheets) aufwendig.

3.4.4 Datenübergabe

Ein weiterer wichtiger Aspekt ist die Datenübergabe an den View. Bei der Generierung der Seitenbeschreibung aus den einzelnen Komponenten, wie es bei ereignisgesteuerten Frameworks vorkommt, wird die Datenübergabe automatisch über die *Bindung* der Komponenten an das Model realisiert. Das Model agiert dabei als Datenquelle für die Komponenten, welche ihrerseits für die Darstellung der Modeldaten zuständig sind. Dabei werden die Daten nicht nur vom Model zur Komponente übertragen, sondern bei Eingabekomponenten auch wieder zurück in das Model.

Bei der Erzeugung der Seitenbeschreibung über einen Template-Ansatz sieht dies anders aus. Hier gibt es zwei verschiedene Ansätze, wie der View mit Daten aus dem Model versorgt wird:

Beim *Push*-Ansatz stellt der Controller die vom View benötigten Daten in einem Kontext zur Verfügung. Der Kontext ist vergleichbar mit einer Hash-Tabelle und enthält Name-Value-Paare. Um im View auf einen Wert aus dem Kontext zuzugreifen, muss der Entwickler nur dessen Namen kennen. Der Push-Ansatz ist sehr einfach, weißt jedoch eine geringe Flexibilität auf, da eine Änderung in der Anzeige oft eine Code-Änderung im Controller verursacht. Soll beispielsweise ein zusätzliches Feld angezeigt werden, muss auch der Controller-Entwickler seinen Code anpassen, um die entsprechenden Daten zur Verfügung zu stellen. Abbildung 3.3 verdeutlich dies noch einmal graphisch.

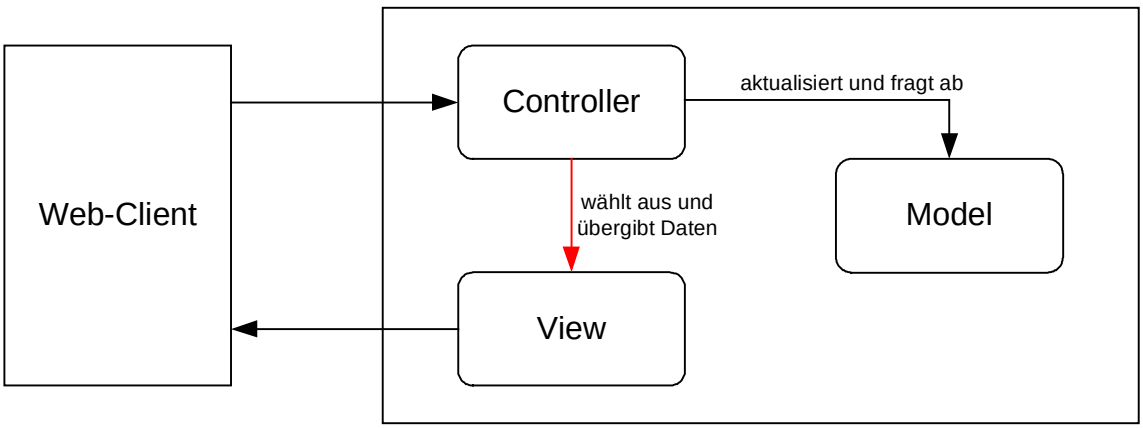

Abbildung 3.3: Push-Ansatz zur Datenübergabe an den View

Ein anderes Konzept verfolgt an dieser Stelle der *Pull*-Ansatz. Hier wird dem View eine allgemeine Schnittstelle zum Zugriff auf beliebige Modeldaten zur Verfügung gestellt. So kann der View frei gestaltet werden, ohne dass eine Anpassung des Controllers nötig wird. Diese erhöhte Flexibilität für die Präsentation muss aber mit einer Zunahme der Komplexität bezahlt werden, da der View-Entwickler alle Zugriffe auf das Model selbst vornehmen muss. Abbildung 3.4 verdeutlich dies noch einmal graphisch.

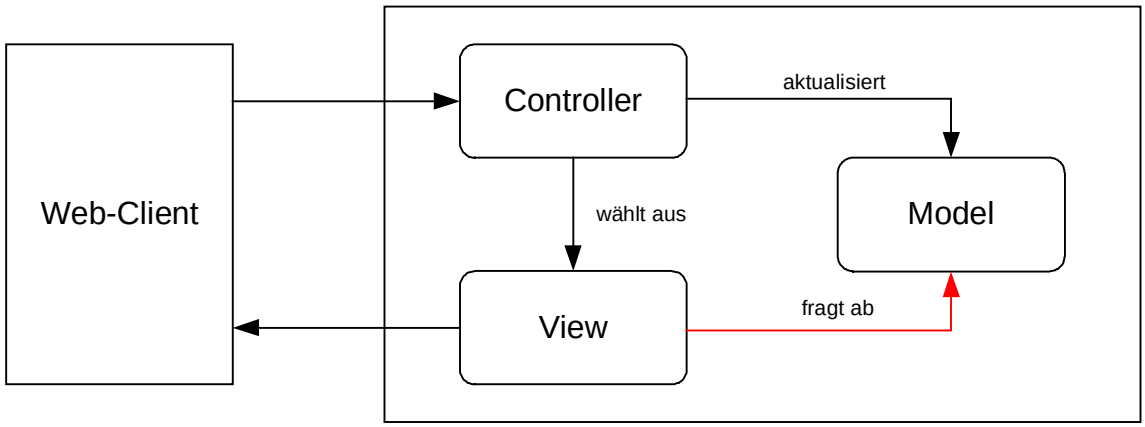

Abbildung 3.4: Pull-Ansatz zur Datenübergabe an den View

3.4.5 Dialogsteuerung

Unter der Dialogsteuerung versteht man, wie Dialogabläufe in der Anwendung definiert werden. Häufig passiert dies implizit im Code. D.h. jede Aktion oder Ereignisbehandlung entscheidet selbstständig, welcher Dialog als nächstes aufgerufen wird. Es gibt keine zentrale Instanz, welche den Dialogablauf kontrolliert oder steuert. Dieser Ansatz ist in vielen Frameworks zu finden, da er relativ einfach zu verwenden ist und keine komplizierte Konfiguration benötigt. Allerdings lassen sich so komplizierte Dialogabläufe kaum auf einen Blick erfassen. Dies macht die Wartung und Anpassung von Dialogen aufwendig und fehlerträchtig.

3.4.5.1 Modellierung als Endlicher Automat

Eine andere Möglichkeit ist es, die Dialogsteuerung explizit an einer zentralen Stelle zu spezifizieren. Hierfür existieren verschiedene Modelle. Die einfachste Möglichkeit ist dabei die Modellierung von Dialogen mittels endlicher Automaten:

> *Ein **endlicher Automat** ist ein 5-Tupel (Z,Σ,d,z_0,E). Hierbei bezeichnet Z die Menge der Zustände und Σ ist das Eingabealphabet, $Z \cap \Sigma = \emptyset$. Z und Σ müssen (wie der Name schon sagt) endliche Menge sein. $z_0 \in Z$ ist der Startzustand, $E \subseteq Z$ ist die Menge der Endzustände und $d : Z \times \Sigma \to Z$ heißt die Überführungsfunktion.*

Zur Veranschaulichung von endlichen Automaten dienen Zustandsgraphen, dabei handelt es sich um gerichtete Graphen, wobei die Zustände die Knoten sind. Der Knoten, der dem Startzustand entspricht, wird durch einen hineingehenden Pfeil besonders markiert und alle Endzustände werden durch doppelte Kreise gekennzeichnet. Die Kanten im Graphen sind dabei folgendermaßen definiert: Von z_1 nach z_2 geht eine mit $a \in \Sigma$ beschriftete Kante, falls $d(z_1,a) = z_2$.

Um einen solchen endlichen Automaten zur Modellierung von Dialogen zu verwenden, interpretiert man die Zustände als Dialogmasken und das Eingabealphabet als Benutzereignisse bzw. Aktionsergebnisse. Abbildung 3.5 stellt den Zustandgraphen eines endlichen Automaten für einen Dialog zur Kundensuche beispielhaft dar:

Initialzustand des Dialoges ist die Kundensuche. Abhängig von der Anzahl der Treffer die zur Suchanfrage gefunden werden, verzweigt der Dialog zurück zur Kundensuche (0 Treffer), zur Dialogmaske Kundendetails (1 Treffer) oder zur Dialogmaske Kundenliste (bei mehr als einem Treffer). In der Dialogmaske Kundenliste sind zwei Benutzerereignisse möglich: „Neue Suche“ oder „Kunde ausgewählt“. Die Dialogmaske Kundendetails erlaubt lediglich eine „Neue Suche“.

In diesem Beispiel zeigen sich aber auch die Probleme dieses Ansatzes. Zwar lassen sich die Übergänge zwischen den einzelnen Dialogmasken (also Zustandsübergänge des endlichen Automaten) sehr gut modellieren, jedoch kann man nicht zwischen Übergängen unterscheiden, welche der Nutzer direkt auslöst (wie z.B. „Neue Suche“ und „Kunde ausgewählt“) und solchen Übergängen, welche das Ergebnis von anwendungsinternen Aktionen sind. So löst der Nutzer zwar in der Dialogmaske „Kundensuche“ die eigentliche Suche aus, welche Dialogmaske jedoch als nächstes aufgerufen wird, entscheidet das Ergebnis der Such-Routine.

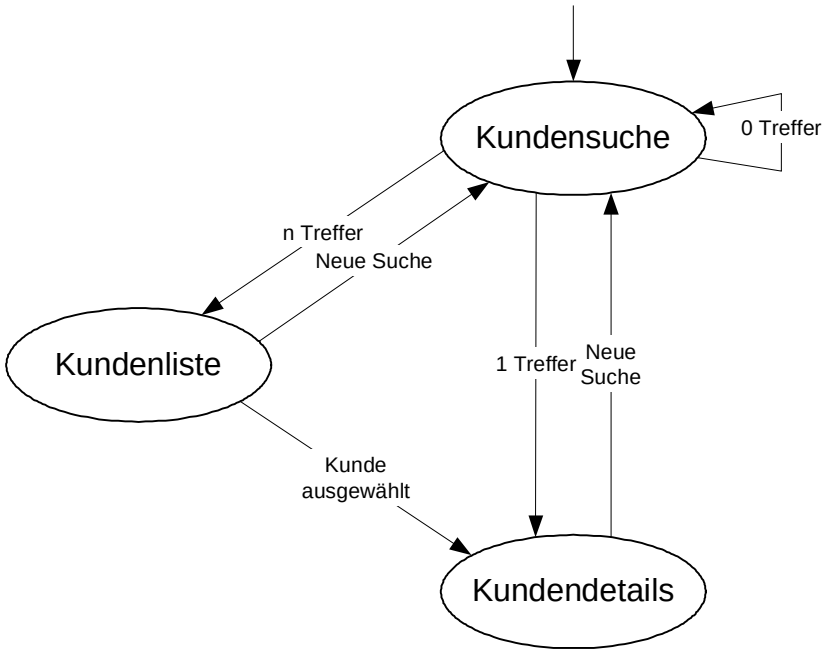

Abbildung 3.5: Kundensuche als endlicher Automat modelliert

Eine Weiterentwicklung der Dialogmodellierung mit endlichen Automaten wird beispielsweise in [KK02] vorgestellt. Im Unterschied zum vorherigen Ansatz stellen hier die Zustände des endlichen Automaten nicht nur Dialogmasken dar, sondern zusätzlich auch Aktionen (vergleiche dazu Abbildung 3.6). Unter Aktionen sind dabei abgeschlossene Programmroutinen zu verstehen, welche von der Anwendung als Reaktion auf Nutzereingaben ausgeführt werden. Je nach Ergebnis einer solchen Aktion kann dann wieder eine Dialogmaske aufgerufen oder weitere Aktionen ausgeführt werden. Zwischen zwei Dialogmasken kann es also beliebig viele Aktionen geben.

Ein weiterer Ansatz zur expliziten Dialogsteuerung wird in [BG03] vorgestellt. In der dort beschrieben Dialog Flow Notation (DFN) wird auch zwischen Dialogmasken und Aktionen unterschieden, allerdings wird ein ähnliches Konstrukt wie Harel’s Statecharts [Ha87] verwendet. Dieser Ansatz ermöglicht es,

Sequenzen aus mehreren Dialogschritten in wieder verwendbare Dialogmodule zu kapseln, die fast beliebig verschachtelt werden können.

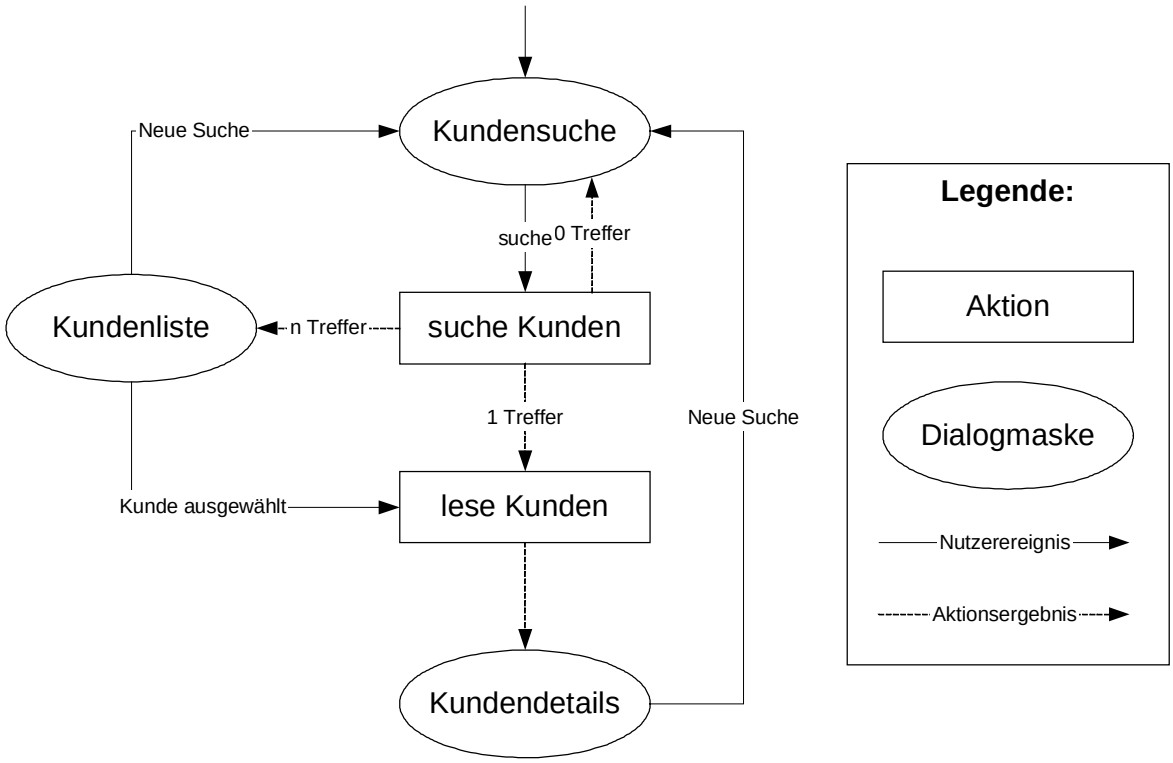

Abbildung 3.6: Kundensuche als erweiterter endlicher Automat modelliert [KK02]

Normalerweise werden Web-Anwendungen als endliche Automaten modelliert. Dies reicht jedoch bei Anwendungen mit komplexen Interaktionen meist nicht mehr aus, so dass viele Erweiterungen entwickelt wurden (wie beispielsweise die zuletzt vorgestellten Ansätze). Grundsätzlich kann der Zustand eines solchen Automaten jedoch immer abhängig von den Nutzereingaben (auch in der Vergangenheit) und der Position im Dialogfluss sein. Durch die Kombination dieser beiden Aspekte entstehet eine hohe Anzahl an Zuständen und Zustandsübergängen, welche behandelt werden müssen. Dies kann die Dialogmodellierung mittels Automaten sehr aufwendig gestallten.

3.4.5.2 Dialogsteuerung als Programm

Einen völlig anderen Ansatz hingegen verfolgen „Struts Flow“ [Ap05c] bzw. „Cocoon Control Flow“ [Ap05b]. Hier wird der Dialogfluss als ein einziges durchgängiges Programm beschrieben. Immer wenn das Programm Eingaben von Nutzerseite benötigt, wird eine Web-Seite mit entsprechenden Eingabeformularen an den Nutzer geschickt und die Ausführung des Programms unterbrochen bis eine entsprechende Antwort eintrifft. Folgendes Beispiel modelliert den Dialog für einen einfachen Addierer:

```
function calculator()
{
   var a, b;

   sendPage(“getA.html”);
   a = request.getParameter(“a”);
```

```
   sendPage(“getB.html”);
   b = request.getParameter(“b”);

   sendPage(“result.html”, a + b);
}
```

Zur Umsetzung dieses Ansatzes wird das so genannte *Continuations*-Konzept genutzt. Continuations werden dabei gewöhnlich als eine Funktion definiert, welche den Rest der Berechnung eines Programms repräsentiert [Bel04]. In objektorientierten Sprachen kann man sich ein Continuation als ein Objekt vorstellen, welches für einen bestimmten Punkt in einem Programm das Abbild des Stacks (inklusive aller lokalen Variablen) und des Programmzählers enthält. Mit diesem Objekt kann man dann die Ausführung des Programms an dieser Stelle wiederherstellen.

Bei Web-Anwendungen kann dies genutzt werden, um die oben beschriebene Unterbrechung des Programmflusses zu bewerkstelligen. Dazu wird bei Aufruf der `sendPage`-Funktion ein Continuation-Objekt erzeugt, welches zusätzlich mit einer eindeutigen Id versehen wird. Daraufhin kann die in der Funktion angegeben Seite mit der Id des Continuation-Objekts an den Web-Client zurück geschickt werden. Der Programmfluss wird an dieser Stelle solange unterbrochen (der Thread jedoch nicht angehalten), bis der Nutzer eine Antwort mit der entsprechenden Id schickt. Erhält der Server eine Anfrage mit einer Continuation-Id, versucht er das zugehörige Contination-Objekt zu finden und damit das Programm in den Zustand zu versetzen, in dem bei Aufruf der `sendPage`-Funktion unterbrochen wurde.

Da das Konzept der Dialogsteuerung von Web-Anwendungen mittels Continuations noch relativ neu ist (erste verwendbare Umsetzungen in [Ap05b, Ap05c]) lässt sich noch nicht viel über Vor- und Nachteile für den praktischen Einsatz sagen. Es lässt sich jedoch vermuten, dass dieses Konzept besonders für lineare Workflows, welche sich einfach als ein imperatives Programm beschreiben lassen, geeignet ist. Beispiele dafür sind zum Beispiel mehrseitige Formulare. Weniger geeignet dürften dagegen Dialoge sein, welche stark verzweigen, wie z.B. Navigationsmenüs, da dies in Programmen zu langen und unübersichtlichen If-Else-Blöcken führt. Gegen diesen Ansatz spricht auch, dass er in gewisser Weise eine Rückkehr zur imperativen Programmierung bedeutet, welche in der Anwendungsentwicklung eigentlich durch die objektorientierte/ereignisgesteuerte Programmierung abgelöst wurde.

3.4.5.3 Zusammenfassung

Abbildung 3.7 gibt noch einmal einen Überblick über die Arten der Dialogsteuerung. Dabei gibt es auf der einen Seite die *implizite Dialogsteuerung*, welche direkt im Code erfolgt. Dies ist am flexibelsten und einfachsten zu Implementieren. Auf der anderen Seite steht die *explizite Dialogsteuerung*. Hier übernimmt eine zentrale Instanz die Steuerung und führt ein Dialogmodell aus. Dafür gibt es wieder zwei mögliche Ansätze: Einmal ein Modell, welches auf einem endlichen Automaten basiert. Dies lehnt sich stark an die Funktionsweise von Web-Anwendungen an und eignet sich besonders für Dialoge mit vielen Verzweigungen. Eine weitere Möglichkeit ist die Modellierung des Dialogs als

Programm. Hier wird stark vom Charakter einer Web-Anwendung abstrahiert. Daher eignet sich dieser Ansatz nur für lineare Dialogfolgen mit wenigen Verzweigungen, welche einfach als „imperatives Programm“ beschrieben werden können.

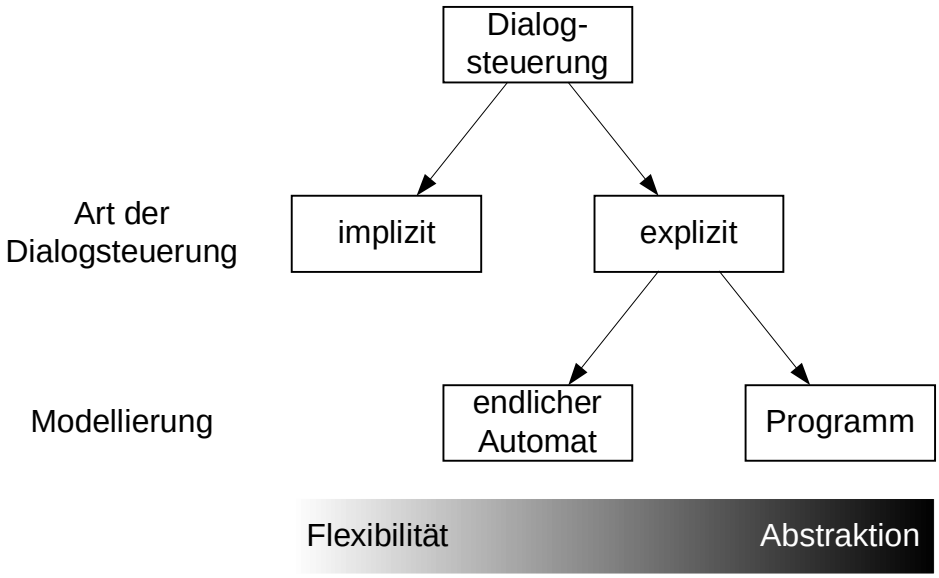

Abbildung 3.7: Arten der Dialogsteuerung

3.4.6 Validierung

Eine Anfrage an eine Web-Anwendung enthält meist Eingabedaten, welche entweder vom Benutzer in Formularen ausgefüllt oder in der URL kodiert sind. Viele Frameworks bieten die Möglichkeit, die Eingabedaten automatisch in ein Modelobjekt zu übertragen, so dass man direkt mit dem Modelobjekt arbeiten kann. Hier ist es oft sehr sinnvoll die Eingabedaten vorher zu validieren, d.h. zu prüfen, ob sie bestimmten Formaten entsprechen, bzw. dass überhaupt Daten für bestimmte Felder übergeben wurden. Insbesondere sollte es dem Nutzer möglich sein, fehlerhafte Eingaben mit keinem oder minimalen Aufwand zu korrigieren (siehe Grundsätze der Dialoggestaltung [ISO95]).

Da Web-Anwendungen einen verteilten Charakter besitzen, bieten sich hier prinzipiell zwei Möglichkeiten an:

- Die *clientseitige Validierung* verwendet Technologien wie JavaScript (siehe 2.6.1.3), welche innerhalb des Browsers begrenzte Programmlogik ausführen können. Diese bietet den Vorteil, dass der Nutzer nicht das komplette Formular ausfüllen und abschicken muss, sondern gleich bei der Eingabe eine Rückmeldung über die Richtigkeit der Eingaben erhält. Problematisch ist bei diesem Ansatz allerdings, dass man sich nicht auf die korrekte Ausführung von JavaScript auf Client-Seite verlassen kann (siehe dazu Abschnitt 2.6.1.3).
- Die zweite Möglichkeit ist die *serverseitige Validierung*. Sie kann zum Einsatz kommen, wenn ein Formular als Anfrage an den Server geschickt wurde. Konnten die Daten nicht korrekt validiert werden, wird die weitere Verarbeitung auf Seite des Servers abgebrochen und das Formular mit einer entsprechenden Fehlermeldung zurück geschickt.

Da beide mögliche Verfahren getrennt voneinander arbeiten können, bietet sich hier eine Kombination beider Verfahren an, die sich aus einer Vor-Validierung auf Client-Seite und einer Nach-Validierung auf Server-Seite zusammensetzt. Dies kombiniert die Bequemlichkeit der clientseitigen mit der Sicherheit der serverseitigen Validierung, verursacht aber auch einen deutlich höheren Implementierungsaufwand und erzeugt Redundanz.

Da die Validierung bei fast allen Web-Anwendungen ein zentraler Aspekt ist, bietet sich die Integration in Frameworks an. Insbesondere bei der serverseitigen Validierung existieren dabei zwei Umsetzungen:

- Bei der *prozeduralen Validierung* werden die Regeln an bestimmen Stellen im Code programmiert. Dies ist ein sehr einfacher und flexibeler Ansatz, welcher allerdings bei einer großen Anzahl zu validierenden Feldern viel Redundanz erzeugt und schwer zu warten ist.
- Bei der *deklarativen Validierung* werden hingegen vorgefertigte Validierungsroutinen verwendet, welche sich beispielsweise zentral über XML-Dokumente konfigurieren lassen. Dies bietet zum einen den Vorteil der zentralen Konfiguration und zum anderen die Möglichkeit, eigene Validierungsroutinen zu entwickeln, falls die vorhandenen den Anforderungen nicht entsprechen.

3.4.7 Sitzungsmanagement

Ein wichtiger Aspekt bei Web-Anwendungen ist das Sitzungsmanagement. Dies beinhaltet zwei zentrale Aspekte:

- Das Aufrechterhalten der logischen Verbindung einer Benutzersitzung über mehrere Anfragen hinweg
- Das Verwalten von sitzungsbezogenen Anwendungsdaten

Der erste Punkt kann dabei vom Servlet-Container auf Basis von Cookies oder URL-Rewriting geleistet werden (siehe dazu *Java Servlet Specification* in [Sun05a]).

Über eine Schnittstelle der Servlet-API ist es dann möglich, sitzungsbezogene Daten abzulegen. Diese Vorgehensweise ist bei vielen aktionsgesteuerten Frameworks anzutreffen, da diese sich näher an den Gegebenheiten der Systemumgebung orientieren. Sitzungsbezogene Daten, wie beispielsweise der Inhalt eines Warenkorbes in einer Shop-Anwendung, muss der Entwickler also bei jeder Anfrage dort auslesen und Änderungen dort wieder ablegen.

Bei ereignisgesteuerten Frameworks sieht dies meist anders aus, da hier versucht wird, den eigentlichen Charakter von Web-Anwendungen vor dem Entwickler zu verbergen und ihm die Illusion zustandsbehafteter Programmierung zu vermitteln. Sie bieten meist ein implizites Management von Objektzuständen, so dass Attribut-Werte auch über Anfrage-Grenzen hinweg erhalten bleiben.

3.5 Fazit

Die meisten der eben vorgestellten Kriterien hängen voneinander ab, wie beispielsweise die Datenübergabe von der Darstellung und die Darstellung wiederum von der Anwendungssteuerung. Abbildung 3.8 verdeutlich diese Abhängigkeiten in einem Schema.

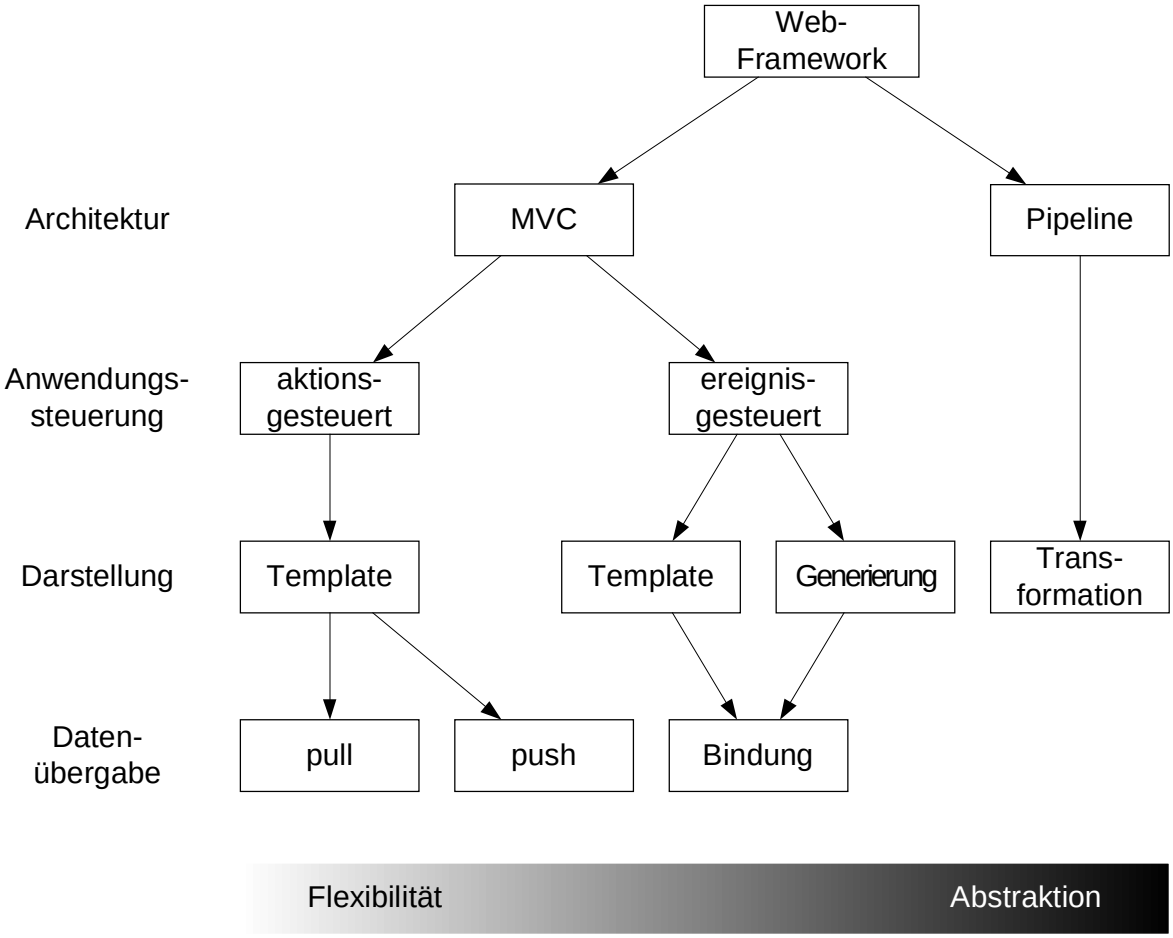

Abbildung 3.8: Abhängigkeiten zwischen den Klassifikationsmerkmalen

So lassen sich Web-Frameworks anhand ihrer Architektur nach dem Pipeline- und dem MVC-Ansatz unterscheiden. MVC-Frameworks wiederum teilen sich auf in aktionsgesteuerte und ereignisgesteuerte Frameworks.

Bei letzteren kann man wieder unterscheiden zwischen Frameworks, bei denen die Darstellung komplett aus den UI-Komponenten generiert wird und Frameworks bei, denen Templates als Vorlage zur Darstellung der UI-Komponenten dienen. Da bei aktionsgesteuerten Frameworks keine UI-Komponenten, wie bei ereignisgesteuerten Frameworks existieren, kann die Darstellung hier nur über Templates erfolgen.

Ebenso kann hier die Datenübergabe nicht über die Bindung der UI-Komponenten an das Model realisiert werden, so dass sich hier zwei verschiedene Ansätze entwickelt haben. Einmal der Push-Ansatz, bei dem der View die Daten vom Controller übermittelt bekommt, oder der Pull-Ansatz, bei dem der View aktiv die Daten beim Model nachfragt.

In gewisser Weise einen Spezialfall stellen Frameworks dar, welche auf einer Pipeline-Architektur basieren. Da sie sich grundlegend vom MVC-Ansatz unterscheiden, ist hier eine Einteilung in aktionsgesteuerte oder ereignisgesteuerte Anwendungssteuerung nicht ohne weiteres möglich, bzw. auch nicht sinnvoll. Auch kommt hier als Darstellungs-Methode nur die Generation der Seitenbeschreibung durch Transformation innerhalb der Pipeline in Frage.

Eine Einordnung in dieses Schema lässt auch Rückschlüsse über die Einsatzmöglichkeiten des Frameworks zu. So sind aktionsgesteuerte Frameworks relativ einfach aufgebaut und besitzen eine hohe Anpassungsfähigkeit und Flexibilität. Ereignisgesteuerte Frameworks hingegen besitzen ein viel abstrakteres Konzept, welches beispielsweise die Wiederverwendung von Komponenten vereinfacht. Dies kommt auch dadurch zum Ausdruck, dass ereignisgesteuerte Frameworks eher einen Black-Box-Ansatz verfolgen als aktionsgesteuerte Frameworks mit einem klassischen White-Box-Ansatz. Einen sehr abstrakten Ansatz verfolgen auch die Pipeline-basierten Frameworks. Auch hier steht wieder der Black-Box-Ansatz, im Sinne der Hintereinanderschaltung von Filtern (oder Komponenten) im Vordergrund.

Auch bietet die Erzeugung der Darstellung über Templates eine weitaus größere Flexibilität als die vollständige Generierung über UI-Komponenten, da hier noch direkt auf die Seitenbeschreibung zugegriffen werden kann. So kann das Layout der erzeugten Seiten relativ frei gestaltet werden. Ebenso ist bei der Datenübergabe das Pull-Konzept flexibler, da hier vom View aus - quasi beliebig - auf das Model zugegriffen werden kann. Im Gegensatz dazu ist beim Push-Konzept immer der Controller beteiligt, um die benötigen Daten bereit zu stellen.

Es lassen sich jedoch nicht alle Kriterien sinnvoll in das Schema in Abbildung 3.8 integrieren. So hängt die Verwendung von UI-Komponenten zur Darstellung vollständig von der Anwendungssteuerung ab. Denn nur bei einer ereignisgesteuerten Anwendungssteuerung ist die Verwendung von UI-Komponenten möglich. Zur weiteren Klassifikation wird es somit auch nicht als eigenständiges Kriterium betrachtet.

Daneben existieren Kriterien, welche vollkommen unabhängig von dem in Abbildung 3.8 vorgestellten Schema sind. Dazu zählen die Dialogsteuerung, die Validierung und das Sitzungsmanagement. Dabei kann die Dialogsteuerung einen wichtigen Aspekt bei der Auswahl eines Frameworks darstellen, insbesondere wenn eine dialogintensive Anwendung erstellt werden soll. Bei Validierung und Sitzungsmanagement handelt es sich hingegen um technische Aspekte, daher sollen sie im Gegensatz zur Dialogsteuerung nicht weiter zur Klassifikation verwendet werden. Tabelle 3.1 gibt noch einmal einen abschließenden Überblick über alle ausgearbeiteten Kriterien und deren mögliche Ausprägungen.

Kriterium	**Ausprägung**	
Architektur	MVC	
	Pipeline	
Anwendungssteuerung	Aktionsgesteuert	
	Ereignisgesteuert	
User-Interface-Komponenten	Ob Komponenten zur Erzeugung der Benutzeroberfläche verwendet werden, hängt vollständig von der Anwendungssteuerung ab.	
Darstellung	Template	
	Generierung	
	Transformation	
Datenübergabe	Pull	
	Push	
	Bindung	
Dialogsteuerung	Implizit	
	Endlicher Automat	
	Programm	
Validierung	Deklarativ	Clientseitig
	Prozedural	Serverseitig
Sitzungsmanagement	Hier lassen sich keine speziellen Ausprägungen identifizieren, da diese Anforderung sehr unterschiedlich gelöst werden kann.	

Tabelle 3.1: Mögliche Ausprägungen von Klassifikationskriterien

Kapitel 4

Klassifikation

Auf Basis der in Kapitel 3 entwickelten Klassifikation wird nun ein Überblick über die am weitesten verbreiteten verfügbaren Frameworks gegeben. Zu Vergleichzwecken werden jedoch auch weniger bekannte Frameworks mit interessanten Konzepten, wie beispielsweise wingS und Echo, vorgestellt. Ziel ist es, jedes dieser Frameworks in das zuvor entwickelte Klassifikationsschema einzuordnen. Um dies zu ermöglichen, wird zu jedem Framework die grundlegende Funktionsweise und das dahinter liegende Konzept herausgearbeitet und dargestellt.

Zur besseren Übersicht werden die wichtigsten Eigenschaften jedes Frameworks in einer Tabelle zu Beginn des jeweiligen Abschnitts zusammengefasst und im Anschluss ausführlich diskutiert. In dieser Tabelle enthalten sind:

- Der Name des Frameworks
- Die Firma oder Organisation, welche das Framework entwickelt
- Die Quelle, von wo das Framework und entsprechende Dokumentationen bezogen werden können
- Die Lizenz, unter welcher Framework verwendet werden darf
- Die Version des Frameworks, auf welche sich die Darstellung der Funktionsweise und Konzepte bezieht
- Die Einordnung anhand der untersuchten Klassifikationskriterien Architektur, Anwendungssteuerung, Darstellung, Datenübergabe und Dialogsteuerung

4.1 Struts

Name	Struts
Entwickler	Apache Software Foundation
Quelle	http://struts.apache.org/
Lizenz	Apache Software License 2.0
Betrachtete Version	1.2.7
Architektur	MVC
Anwendungssteuerung	Aktionsgesteuert
Darstellung	Template (JSP, Velocity)
Datenübergabe	Pull
Dialogsteuerung	Implizit

Tabelle 4.1: Eigenschaften Struts-Framework

Struts ist das Standard-Framework für Entwicklung von Web-Anwendungen auf Basis von J2EE und wird in der Praxis am häufigsten eingesetzt [Wa04]. Struts ist ein typisches MVC-Framework, welches die „Model 2“-Architektur umsetzt. Dabei werden viele Standard-Technologien wie JSP, Servlets und XML verwendet. Dies macht das Framework sehr flexibel und leicht erlernbar. Aufgrund seiner großen Popularität ist es inzwischen gut dokumentiert, zum einen durch die Online-Dokumentation [Ap05c] und zum anderen durch ein breites Angebot an Sekundärliteratur.

Struts ist im Vergleich zu vielen anderen Frameworks ein sehr schlankes Framework, welches sich stark an den Request-Respons-Zyklus von Web-Anwendungen anlehnt. Mit Struts kann man nicht mehr, aber auch nicht weniger als eine Model-2-Web-Anwendung implementieren. Dadurch ist es sehr flexibel und kann für viele Zwecke angepasst und erweitert werden, wie beispielsweise in [KK02] zu sehen ist.

4.1.1 Aufbau

Das Zusammenwirken von Model-, View- und Controllerkomponenten ist in Abbildung 4.1 schematisch dargestellt. Die Abbildung verdeutlicht auch, dass der Schwerpunkt von Struts eindeutig auf den View- und Controllerkomponenten liegt. Über das Model wird keine Aussage gemacht. Hier ist es dem Entwickler überlassen, wie er die Anbindung beispielsweise an Datenbanken oder EJBs realisiert.

Als View werden in Struts standardmäßig JavaServer Pages (JSP) verwendet. Dafür werden auch spezielle Tag-Libraries bereitgestellt. Daneben lassen sich aber auch andere Technologien als View integrieren, beispielsweise die Template-Engine Velocity [Ap05d]. Im Folgenden wird sich jedoch nur auf die Verwendung von JSP bezogen, da dies von Struts am besten unterstützt wird.

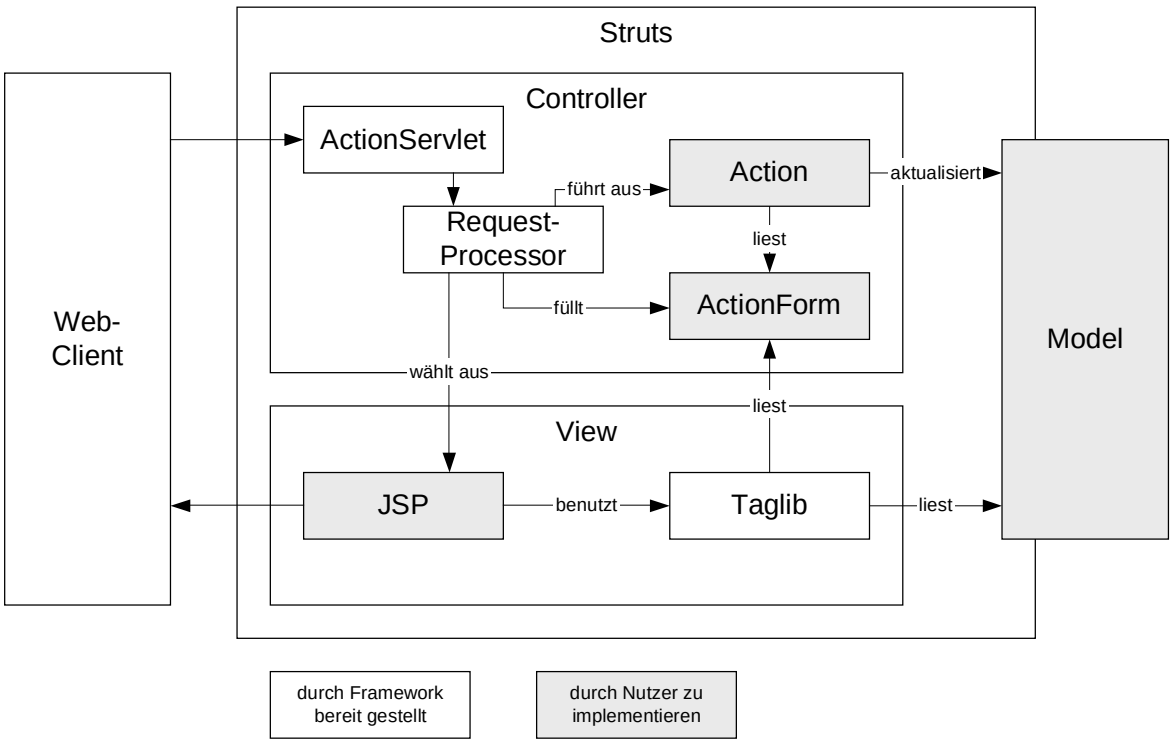

Abbildung 4.1: Schematischer Aufbau des Struts-Frameworks

4.1.2 Controller

Der Controller – im Sinne von MVC – ist (wie in Abbildung 4.1 zu sehen) das Herzstück des Struts-Frameworks. Er besteht im Kern aus vier Klassen:

- Das *Action-Servlet* bindet die Anwendung in den Request-Response-Zyklus des Servlet-Containers ein.
- Der *Request-Prozessor* übernimmt die meisten Aufgaben innerhalb des Controllers. Er durchläuft eine Prozesskette, um einen Request abzuarbeiten. Darauf wird im Folgenden noch näher eingegangen.
- Die *Aktionen* implementieren die eigentliche Anwendungslogik (im Gegensatz zu den ersten beiden Klassen, welche eher zum technischen Teil des Frameworks gehören). Hier kann auf das Model und die Formulardaten in den ActionForms zugegriffen sowie die anzuzeigende Seite festgelegt werden.
- *ActionForms* enthalten die Daten von HTML-Formularen, mit welchen der Nutzer über mehrere Seiten interagieren kann. Da sie ähnlich wie normale JavaBeans zur Datenhaltung dienen, könnte man sie eigentlich auch dem Model zuordnen. In [Ap05c] werden sie jedoch ausdrücklich als Teil des Controllers betrachtet, da ihre Aufgabe nur in der Bereitstellung und Validierung der Formulardaten dient.

Der grobe Ablauf der Request-Verarbeitung im Controller ist in Abbildung 4.2 dargestellt. Dort sieht man, wie die vier zentralen Klassen des Controllers zusammenarbeiten. Hier sollen noch einmal die wichtigsten Schritte während des Durchlaufens der Prozessklette im Request-Prozessor beschrieben werden [Ap05c]:

- Zuerst wird ein Mapping für den übergebenen Pfad gesucht.

- Es wird eine ActionForm für das aktuelle Mapping bereitgestellt. Diese wird entweder im Request- oder Session-Kontext gespeichert.
- Mit `processPopulate()` werden die Request-Parameter in die entsprechenden Felder der ActionForm übertragen.
- Die Daten in der ActionForm werden validiert.
- Es wird eine Instanz der zu diesem Mapping gehörenden Aktion bereitgestellt.
- Der Prozessor führt die `execute()`-Methode der Aktion aus.
- Entsprechend der Rückgabe der Aktion wird der entsprechende View ausgewählt.

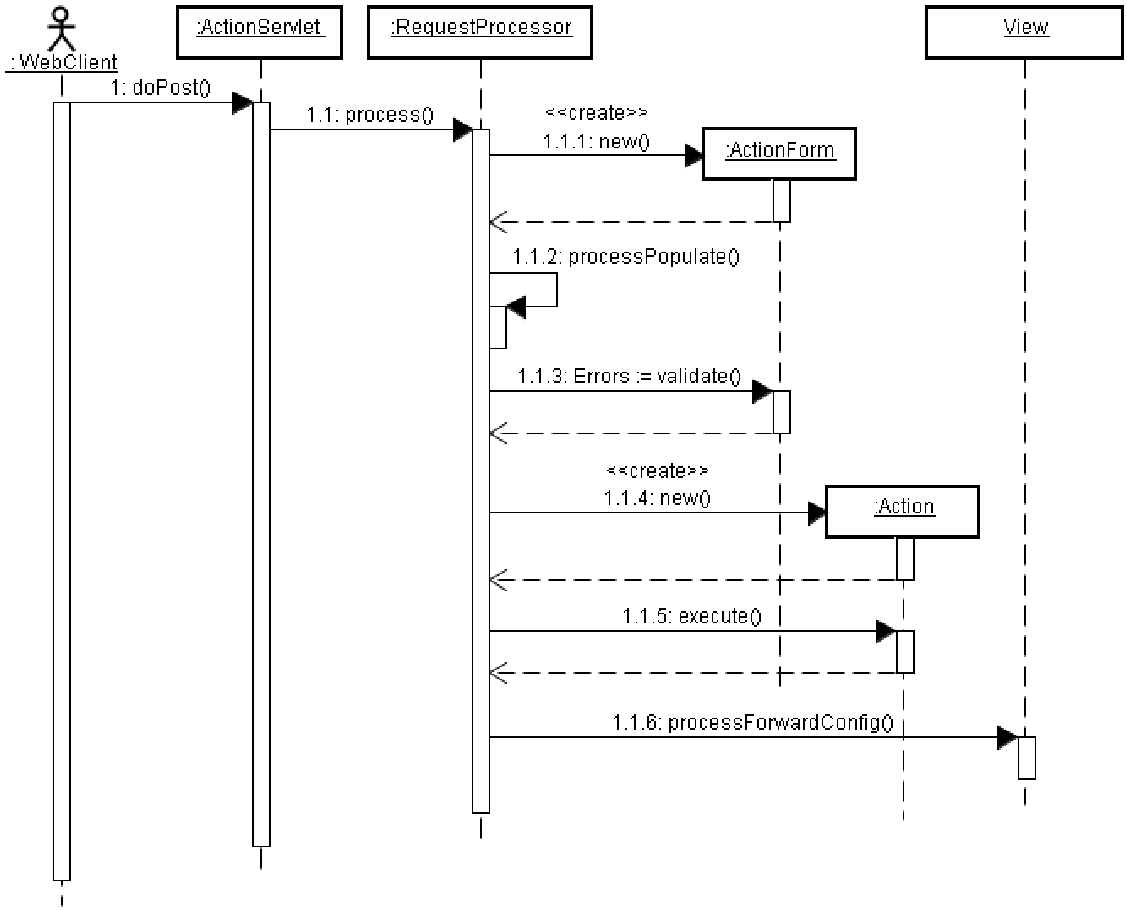

Abbildung 4.2: Ablauf der Request-Verarbeitung im Struts-Controller [Ap05c]

4.1.3 View

Der View besteht in Struts standardmäßig aus zwei Teilen: Zum einen den JSP-Seiten (siehe dazu Abschnitt 2.6.2.1) und zum anderen speziellen Struts-eigenen Tag-Libraries. Durch den Einsatz dieser Tags besteht die Möglichkeit, den Einsatz von Scriptlets in den JSP-Seiten auf ein Minimum zu reduzieren bzw. ganz zu vermeiden. Durch die sehr große Anzahl solcher Tags, welche die verschiedensten Bereiche abdecken, wird das Erstellen von internationalisierten, interaktiven und formularbasierten JSP-Seiten vereinfacht. Daneben besteht noch die Möglichkeit, eigene Tags zu implementieren. Durch dieses Vorgehen wird auch die Trennung zwischen deklarativem (JSP-Seiten) und prozeduralem (Tag-

Libraries) Teil der Darstellung erhöht, was die Wartbarkeit und Wiederverwendbarkeit beider Teile erhöht.

Eine spezielle Tag-Library von Struts stellt das so genannte „Tiles“ dar. Damit wird es möglich, JSP-Seiten aus einzelnen Teilen zusammenzubauen. Jeder Teil („Tile“) kann so oft wie nötig in einer Anwendung verwendet werden. Dies reduziert den zu erstellenden Markup-Code und vereinfacht es, das Layout einer Web-Anwendung ohne großen Aufwand zu verändern.

Dazu werden die Seiten in wiederkehrende Teilelemente und Rahmenseiten zerlegt, welche jeweils als eigene JSP implementiert werden. Eine solche Rahmenseite (hier als `layout.jsp` bezeichnet) legt das globale Layout fest und könnte wie folgt aussehen:

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
   <body>
      <tiles:insert attribute=”header” />
      <tiles:insert attribute=”body” />
      <tiles:inster attribute=”footer” />
   </body>
</html>
```

In diesem Beispiel wird die Seite in drei Teile zerlegt. Eine Seite, welche diese Rahmenvorlage benutzt, würde dann wie folgt aussehen:

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<tiles:insert template="layout.jsp">
   <tiles:put name="header" value=”header.jsp” />
   <tiles:put name="body" value=”samplePage.jsp” />
   <tiles:put name="footer" value=”footer.jsp” />
</tiles:insert>
```

Hier wird die obige Vorlage mit `<tiles:insert template=“layout.jsp“>` in die Seite eingefügt und jeweils mit `<tiles:put />` festgelegt, welche JSP die Platzhalter in der Vorlage ersetzen soll. So kann beispielsweise für die gesamte Anwendung derselbe `header` und `footer` verwendet werden und mit dem jeweils seitenspezifischen `body` kombiniert werden.

4.1.4 Klassifikation

Bei der *Architektur* verfolgt Struts einen klassischen MVC-Ansatz, wie in Abbildung 4.1 gut zu sehen ist. Dabei bilden die zu implementierenden Aktionen und `ActionForms` den Controller der Web-Anwendung. Der View wird durch die JSP-Seiten und Tag-Libraries gebildet.

Die *Anwendungssteuerung* basiert auf Aktionen. Diese werden anhand der eingehenden Anfrage ausgewählt und führen dann die entsprechende Anwendungslogik aus.

Bei der *Darstellung* setzt Struts standardmäßig auf JSP, einen Template-basierten Ansatz. Durch die offene und leicht erweiterbare Architektur von Struts lassen sich

jedoch auch andere Template-basierte Technologien, wie beispielsweise Velocity [Ap05d], als View verwenden.

Wie in Abbildung 4.1 zu sehen, wird in Struts (bei Verwendung von JSP) über die Tag-Libraries ein Pull-Mechanismus für *Datenübergabe* an den View realisiert, d.h. der View holt sich aktiv (mit Hilfe der Tag-Libraries) die Daten aus dem Model.

Von Haus aus unterstützt Struts nur eine implizite *Dialogsteuerung*, da der Aufruf der nächsten Dialogseite allein durch den Rückgabewert der ausgeführten Action bestimmt wird. Bei der Popularität von Struts ist es daher nicht verwunderlich, dass es eine Reihe von Ansätzen gibt, welche Struts um eine explizite Dialogsteuerung erweitern. Einerseits besitzt Struts selber das Unterprojekt „Struts Flow“ [Ap05c] (Abschnitt 3.4.5.2), zum anderen existieren verschiedene Ansätze, welche auf endlichen Automaten aufbauen [KK02].

4.2 Velocity

Name	Velocity
Entwickler	Apache Software Foundation
Quelle	http://jakarta.apache.org/velocity/
Lizenz	Apache Software License 2.0
Betrachtete Version	1.4
Architektur	-
Anwendungssteuerung	-
Darstellung	Template
Datenübergabe	Push-Ansatz
Dialogsteuerung	-

Tabelle 4.2: Eigenschaften Velocity-Template-Engine

Im eigentlichen Sinne ist Velocity [Ap05d] kein Web-Framework, welches beispielsweise einen MVC-Ansatz umsetzt. Vielmehr handelt es sich um eine *Template Engine*. Da es aber in einigen Web-Frameworks Anwendung findet (beispielsweise in Turbine [Ap05e]) wird es hier kurz vorgestellt.

Velocity kann grundsätzlich dazu dienen, alle Arten von Code dynamisch zu erzeugen, seien es HTML, WML, SQL, Java-Quellcode oder andere. Im Kontext von Web-Frameworks wird es häufig als Alternative zu JSP eingesetzt. Dies liegt vor allem daran, dass JSP in ihrer ursprünglichen Form, als mit Skriptlets angereicherte HTML-Seiten, von vielen Entwicklern aufgrund der komplizierten und unübersichtlichen Syntax nicht akzeptiert wurden. Velocity bot hier eine einfachere Syntax, welche auch für Web-Designer schnell zu erlernen war. Obwohl diese Probleme durch die Einführung der Tag-Libraries zum größten Teil behoben wurden, blieb Velocity weiterhin beliebt. [Wa04]

4.2.1 Funktionsweise

Das wesentliche Merkmal von Velocity ist die strikte Trennung von Anwendungslogik und Darstellung. Dies ist bei JSP nicht grundsätzlich gegeben, da hier mittels Skriptlets auch Anwendungslogik direkt in eine Seite integriert werden kann. Velocity stellt stattdessen eine einfache Skriptsprache zur Verfügung, welche dazu dient, innerhalb der Seitenvorlage dynamisch auf Daten zuzugreifen.

Die Verwendung von Velocity funktioniert dabei grundsätzlich immer nach den folgenden Schritten [Ap05d]:

(1) Initialisieren von Velocity

(2) Erzeugen des Kontextobjektes

(3) Datenobjekte zum Kontextobjekt hinzufügen

(4) Template auswählen

(5) Verbinden von Kontextobjekt und Template zu einem Ausgabestrom

In Java sieht dies wie folgt aus [Ap05d]:

```
Velocity.init(); // (1)
VelocityContext context = new VelocityContext(); // (2)

context.put("name", new String("Velocity")); // (3)

Template template = null;
try {
   template = Velocity.getTemplate("sample.vm"); // (4)
}
catch( Exception e ) {}

StringWriter sw = new StringWriter();
template.merge( context, sw ); // (5)
```

Meist wird Velocity jedoch innerhalb eines Servlet-Containers verwendet. Dafür wird ein spezielles Servlet (`VelocityServlet`) bereitgestellt, welches bei seiner Initialisierung automatisch Velocity initialisiert (1), ein Kontextobjekt erzeugt (2) sowie Template und Datenobjekte verbindet und in den Response-Ausgabestrom schreibt (5). Die Schritte (3) und (4) werden durch Implementierung der abstrakten Methode `handleRequest` umgesetzt, wie folgendes Beispiel zeigt:

```
public class SampleServlet extends VelocityServlet
{
   public Template handleRequest(HttpServletRequest request,
HttpServletResponse response, Context context)
   {
      context.put("name", new String("Velocity")); // (3)
      List list = new ArrayList();
      List.add("table1");
      List.add("table2");
      List.add("table3");
      context.put("table", list); // (3)
```

```
        Template template = null;
        try
        {
           template = getTemplate("sample.vm"); // (4)
        }
        catch( Exception e ) {}

        return template;
    }
}
```

In Velocity sind Templates Dokumente im zu erzeugenden Zielformat (beispielsweise HTML), welche mit Velocity-eigenen Skriptbefehlen angereichert sind. Mit diesen Befehlen ist es möglich, auf Datenobjekte innerhalb des Kontextobjektes zuzugreifen. Diese sind wie in Hashtabellen über ihren Indexwert abrufbar. Dabei kann es sich auch um komplexe Objektstrukturen handeln. Das folgende Beispiel zeigt ein Template für eine HTML-Seite - passend zum Kontextobjekt aus dem letzen Beispiel:

```
<html>
   <head><title>$name sample</title></head>
   <body>
      <p>$table.size() table entries found!</p>
      <table>
         <tr><th>table header</th></tr>
         #foreach($entry in $table)
            <tr><td>$entry</td></tr>
         #end
      </table>
   </body>
</html>
```

Wie im Beispiel zu sehen, bietet Velocity beispielsweise Befehle, um über komplexe Objektstrukturen zu iterieren (`#foreach`) und Methoden von Datenobjekten aufzurufen (`$table.size()`). Der volle Umfang dieser Skriptsprache kann in [Ap05d] nachgelesen werden. Darüber hinaus ist es möglich, das gleiche Kontextobjekt mit verschiedenen Templates, welche beispielsweise unterschiedliche Zielformate generieren, zu verwenden. Dies kann unter anderem dazu dienen, verschiedene Endgeräte (z.B. mobile Endgeräte) mit unterschiedlichen Ausgabeformaten zu versorgen.

4.2.2 Klassifikation

Da es sich bei Velocity nur um eine Template-Engine und kein vollwertiges Framework handelt ist hier nur eine eingeschränkte Klassifikation möglich. Betrachtet man Velocity im Rahmen einer MVC-Architektur, konzentriert es sich allein auf den View. Daher gibt Velocity weder die *Architektur* noch die *Anwendungssteuerung* einer Web-Anwendung vor. Ebenso fehlt eine Unterstützung zur *Dialogsteuerung*.

Bei der *Darstellung* implementiert Velocity einen klassischen Template-Ansatz mit Hilfe einer einfachen Skriptsprache innerhalb der Templates. Die *Datenübergabe*

mittels eines Kontextobjektes setzt dabei einen Klassenischen Push-Ansatz um (siehe 3.4.4).

4.3 Turbine

Name	Turbine
Entwickler	Apache Software Foundation
Quelle	http://jakarta.apache.org/turbine/
Lizenz	Apache Software License 2.0
Betrachtete Version	2.3
Architektur	MVC
Anwendungssteuerung	Aktionsgesteuert
Darstellung	Template (Velocity, aber auch JSP und andere möglich)
Datenübergabe	Push- und Pull-Ansatz werden unterstützt
Dialogsteuerung	Implizit

Tabelle 4.3: Eigenschaften Turbine-Framework

Turbine [Ap05e] aus dem Jakarta-Projekt von Apache ist ein serviceorientiertes Framework, welches eine MVC-Architektur umsetzt. Einzelne Dienste in Turbine kapseln dabei abgeschlossene Funktionalitäten, welche wieder verwendbar sind und lose zu einer Web-Anwendung verbunden werden können. Turbine stellt dabei die Grundlage zur Nutzung dieser Dienste bereit.

Die Entwicklung von Turbine geht auf das Jahr 1999 zurück, als es noch keine J2EE-Plattform gab. Dies führte dazu, dass für Turbine viele Dienste entwickelt wurden, welche den kompletten Verarbeitungszyklus einer HTTP-Anfrage vom Servlet bis hin zur Datenbank abdecken. So ist Turbine eines der umfangreichsten Frameworks mit einer großen Anzahl von Subprojekten geworden. Einige dieser Subprojekte sind inzwischen aus Turbine herausgelöst worden und können als eigenständige Produkte verwendet werden. [Wa04]

Die Entwicklung von Web-Anwendungen mit Turbine nach dem MVC-Ansatz nutzt nur einen kleinen Teil des gesamten Frameworks. Tabelle 4.11 gibt einen Überblick über die wichtigsten (aber bei weitem nicht allen) von Turbine bereit gestellten Dienste. Viele dieser Dienste sind nicht per se integrale Bestandteile von Turbine, vielmehr handelt es sich dabei häufig um Schnittstellen zu externen Produkten.

Dienstname	**Beschreibung**
Cache Service	Ein Caching-Dienst, der zum temporären Zwischenspeichern global verfügbarer Objekte dient. Inzwischen unter dem Namen Java Caching System (JCS) auch als eigenständiges Projekt bei Apache vertreten.

Dienstname	Beschreibung
Component Service	Komponenten-Dienst zum Laden externer Module. Daraus ist unter anderem das Service-Framework Fulcrum hervorgegangen.
Intake Service	Dient der Validierung von Formulardaten (deklarativ über Konfigurationsdatei).
JSP Service	Integriert JavaServer Pages als View-Komponente.
Localization Service	Ermöglicht den Zugriff auf lokalisierte Ressourcen.
Logging Service	Unterstützung von Logging-Diensten wie z.B. Log4J.
Naming Service	Stellt JNDI Naming Context bereit, z.B. für den Aufruf von EJBs.
Pull Service	Stellt Anwendungsdaten in allen Templates bereit. Ermöglicht so die Datenübergabe nach dem Pull-Ansatz.
Scheduler Service	Ermöglicht das Ausführen von Aktionen zu bestimmten (auch wiederkehrenden) Zeitpunkten.
Security Service	Verwaltet Benutzer in Gruppen mit Rollen und Rechten.
Session Service	Verwaltung des Nutzerzustandes.
Velocity Service	Integriert die Velocity Template Engine als View-Komponente.
XML-RPC Service	Ermöglicht vereinfachte Remote Procedure Calls auf anderen Servern.
XSLT Service	Dienst, um XML-Daten mittels XSLT-Stylesheets zu transformieren.

Tabelle 4.4: Auswahl von Turbine-Diensten [Ap05e]

4.3.1 Aufbau und Funktionsweise

Ähnlich wie andere aktionsgesteuerte Frameworks orientiert Turbine sich sehr stark am Request-Response-Zyklus einer Web-Anwendung. In Abbildung 4.3 ist dies schematisch dargestellt. Dabei leitet im ersten Schritt das `TurbineServlet` die Anfrage an die `Page`-Klasse weiter. Diese sind den im Folgenden ausgeführten Modulen Layout, Navigation und Screen zugeordnet. Zuerst wird jedoch geprüft, ob für diese Anfrage eine Aktion definiert wurde. Wenn ja, wird diese ausgeführt. Anschließend wird der zu dieser Anfrage gehörende bzw. von der zuvor ausgeführten Aktion gesetzte Screen aufgerufen. Entsprechend der Rückgabe des Screen wird dann das entsprechende Layout mit den enthaltenen Navigationselementen ausgeführt. Als letzter Schritt kann dann die entsprechende Antwortseite erzeugt und an den Web-Client zurück geschickt werden.

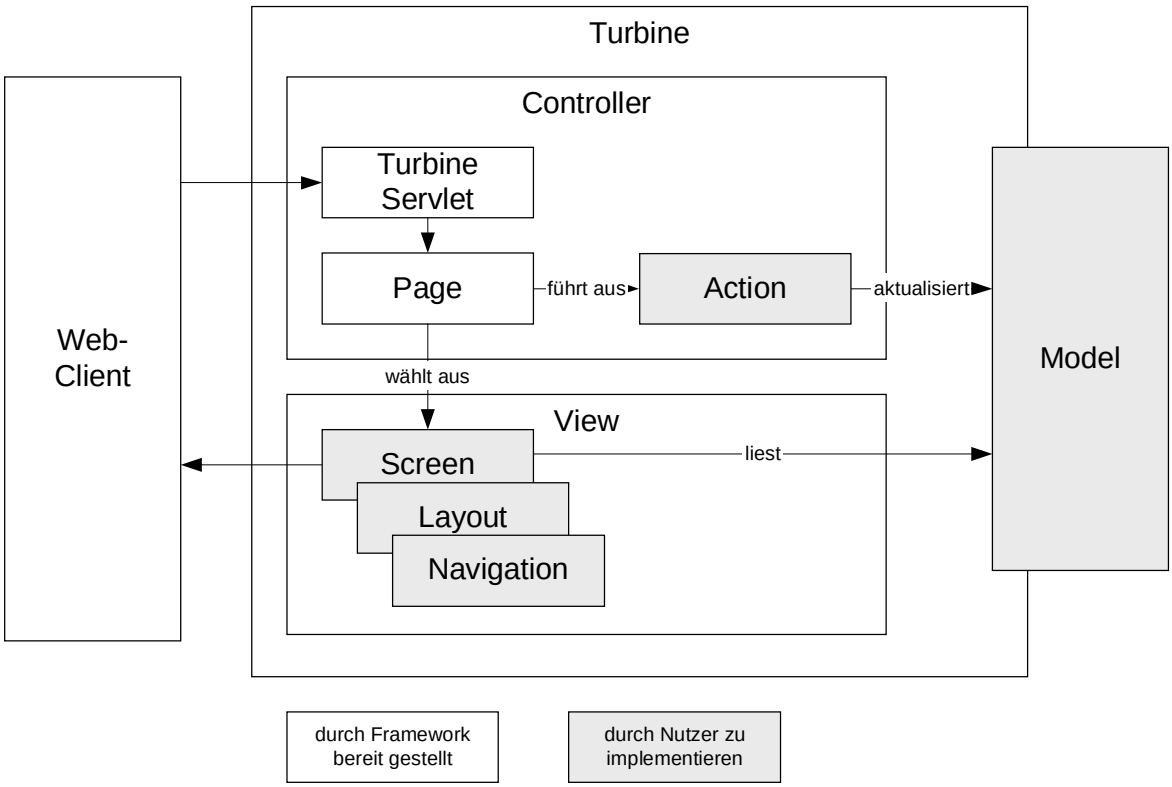

Abbildung 4.3: Schematischer Ablauf des Request-Response-Zyklus in Turbine

Eine solche Seite ist in Turbine strikt hierarchisch aufgebaut. Sie enthält ein Layout, welches den Seitenaufbau (siehe Abbildung 4.4) festlegt. Das Layout enthält wiederum Navigation und Screen. Bei der Navigation handelt es sich dabei typischerweise um Kopf- und Fußzeilen, eventuell auch Navigationsmenüs oder ähnliches. Der eigentliche Inhalt einer Seite wird vom Screen zur Verfügung gestellt und ist verantwortlich für den Großteil der Benutzerinteraktion und Datenanzeige.

Für jedes dieser Seitenelemente wird von Turbine ein eigenes Template zur Verfügung gestellt. Dabei wird das Navigation- und Layout-Template von mehreren Seiten verwendet, um so beispielsweise ein einheitliches Layout für die gesamte Anwendung zu definieren. Jedem dieser Templates ist dabei eine Java-Klasse zugeordnet, welche für dessen Verarbeitung zuständig ist und es mit Daten versorgt. Diese werden, wie oben erläutert, im Rahmen des Request-Response-Zyklus von Turbine ausgeführt (siehe Abbildung 4.3).

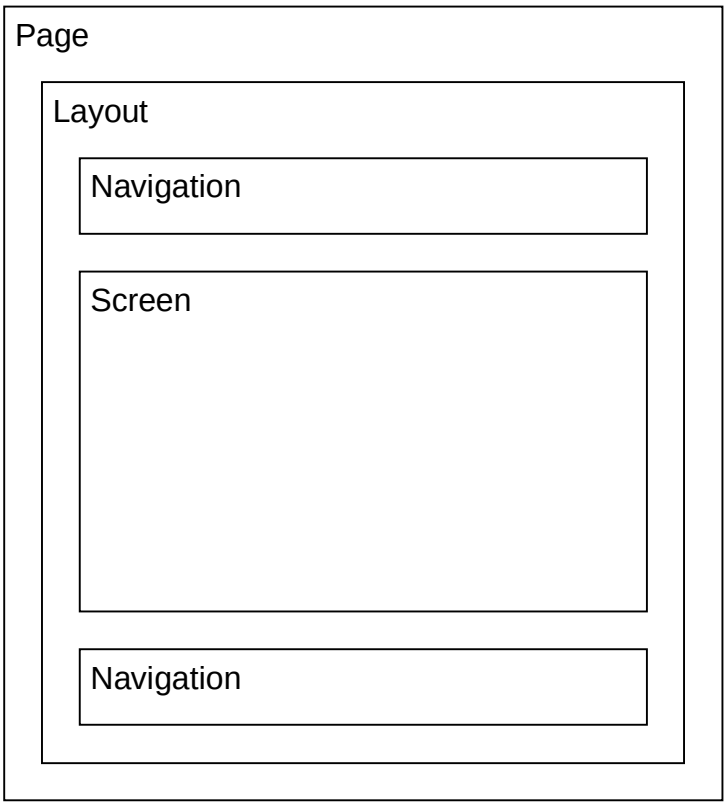

Abbildung 4.4: typische Seitenstruktur in Turbine [Ap05e]

Will man beispielsweise eigene Daten im Screen-Template bereitstellen, muss man eine entsprechende Screen-Klasse implementieren, welche dann statt der Defaultklasse verwendet wird. Dies könnte z.B. bei der Verwendung von Velocity als View-Technologie wie folgt aussehen:

```
package homepage.modules.screens;

import org.apache.turbine.modules.screens.VelocityScreen;
import org.apache.turbine.util.RunData;
import org.apache.velocity.context.Context;

public class content extends VelocityScreen {
   public void doBuildTemplate( RunData data, Context context )
   {
      try
      {
         context.put("name", "Turbine");
      }
      catch( Exception e )
      {
         return;
      }
   }
}
```

4.3.2 Klassifikation

Wie in Abbildung 4.3 zu sehen, verwendet Turbine bei der *Architektur* einen klassischen MVC-Ansatz. Der Controller wird hauptsächlich durch die `Page`- und `Action`-Klassen gebildet. Der View baut sich aus den `Screen`-, `Layout`- und `Navigation`-Klassen mit den entsprechenden Templates zusammen.

Ähnlich wie Struts (Abschnitt 4.1) verwendet auch Turbine eine aktionsgesteuerte *Anwendungssteuerung*, welche sich stark an den Request-Response-Zyklus der Web-Anwendung anlehnt.

Zur *Darstellung* stellt Turbine verschiedene Dienste zur Integration von Template-basierten Technologien bereit, wobei bevorzugt Velocity (siehe Abschnitt 4.2) eingesetzt wird. Daneben ist es jedoch auch möglich, andere Technologien wie z.B. JSP zu verwenden.

Die *Datenübergabe* erfolgt dabei, wie bei Velocity üblich, über ein vorher zu befüllendes Kontextobjekt (Push-Ansatz). Daneben besteht die Möglichkeit, über den Pull-Service spezielle Tool-Objekte bereit zu stellen, welche als Schnittstelle für den Zugriff auf das Model dienen (Pull-Ansatz).

Die *Dialogsteuerung* wird in Turbine implizit über die Rückgabe der Aktionen definiert.

4.4 Tapestry

Name	Tapestry
Entwickler	Apache Software Foundation
Quelle	http://jakarta.apache.org/tapestry/
Lizenz	Apache Software License 2.0
Betrachtete Version	4.0
Architektur	MVC
Anwendungssteuerung	Ereignisgesteuert
Darstellung	Eigene Template-Engine
Datenübergabe	Bindung
Dialogsteuerung	Implizit

Tabelle 4.5: Eigenschaften Tapestry-Framework

Tapestry [Ap05f] ist ein komponentenbasiertes Framework, das mit GUI-Komponenten und Event-Handling arbeitet. Dies abstrahiert stark vom Request-Response-Zyklus einer Web-Anwendung und ermöglicht so die Entwicklung von Benutzeroberflächen - ähnlich wie bei Destop-Anwendungen mit wieder verwendbaren Komponenten.

Durch die Verwendung von eigenen oder fremden Komponenten kann ein hoher Grad an Wiederverwendung erreicht werden, was zu einer deutlichen

Produktionssteigerung führen kann. Allerdings benötigt man eine gewisse Einarbeitungszeit, da der Ansatz von Tapestry nicht dem eines typischen aktionsgesteuerten Frameworks entspricht.

4.4.1 Aufbau

Das Kernstück einer Tapestry-Anwendung ist die so genannte Application Engine. Sie dient als zentraler Controller der Anwendung und nimmt die HTTP-Anfragen entgegen, um sie an die entsprechenden Seiten weiterzuleiten und für deren Darstellung zu sorgen. Ähnlich wie bei den Komponenten von Swing werden in Tapestry sämtliche in der Seite enthaltenen Komponenten über die `render()`-Methode (siehe Abbildung 4.5) aufgefordert, sich darzustellen.

Grundsätzlich handelt es sich bei Tapestry-Seiten um spezielle Komponenten (siehe Abbildung 4.5), welche auch als Container für weitere Komponenten dienen. Sie werden zur Laufzeit durch Objekte dargestellt, die bei der Initialisierung mit Session-Daten gefüllt werden. Zudem werden die von der Seite verwendeten Komponenten initialisiert. Dabei können die Parameter einer Komponente an Datenquellen gebunden sein, was in Tapestry als Binding bezeichnet wird. [ST05]

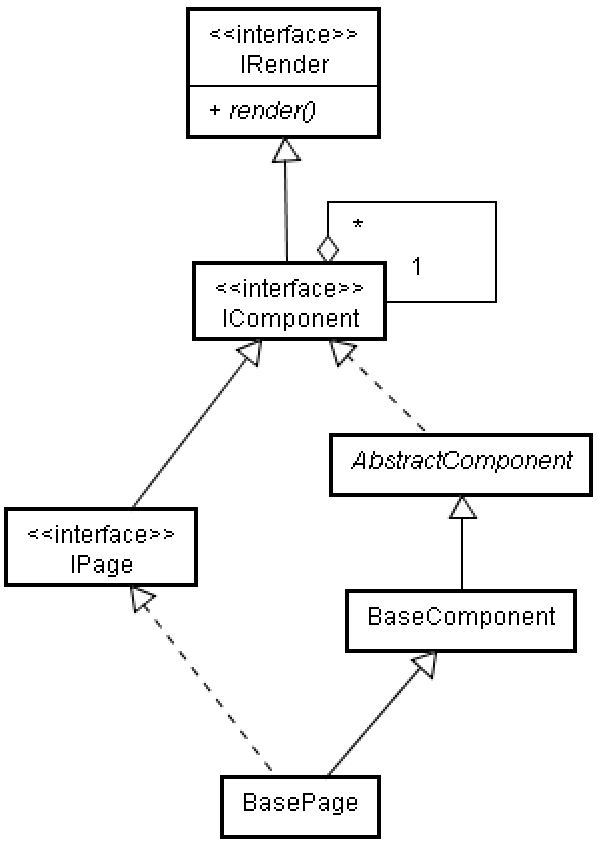

Abbildung 4.5: Aufbau des Komponentenbaumes in Tapestry [Ap05f]

4.4.2 Seiten

Jede Komponente in Tapestry, also auch eine Seite (siehe Abbildung 4.5), besteht im Allgemeinen aus drei Teilen:

- Die *Seitenspezifikation* definiert, welche Parameter und Komponenten innerhalb der Seite verwendet werden. Außerdem wird auf die Java-Klasse verwiesen, die diese Seite implementiert. Wird keine Klasse angegeben, wird die Basis-Seitenklasse des Frameworks (`BasePage`) verwendet.
- Das *HTML-Template* definiert das Layout der Seite. Es besteht ausschließlich aus statischen HTML-Elementen (ohne spezielle Tags oder Scriptbefehle wie, beispielsweise in JSP oder Velocity). Die mit einem `jwcid`-Attribut ergänzten Elemente, welche auf eine Komponente verweisen, werden jedoch beim Erzeugen der Seite durch die Darstellung dieser Komponente ersetzt.
- Die *Page-Klasse* der Seite enthält die Zugriffs-Methoden für die Parameter der Seite. Darüber hinaus kann sie noch Methoden enthalten, welche die Benutzerinteraktionen auf der Seite behandeln. So kann z.B. zu einer Link-Komponente auf eine Listener-Methode verwiesen werden, welche beim Anklicken des Links aufgerufen wird.

Im Folgenden wird das Zusammenwirken dieser drei Teile exemplarisch an einer Login-Seite dargestellt. Dazu wird in der Seitenspezifikation (`Login.page`) festgelegt, welche Java-Klasse (`sample.tapestry.page.Login`) die Seite implementiert. Außerdem wird noch eine Komponente vom Type `Form` mit der ID `form` definiert und die Methode `doLogin()` der Page-Klasse als Event-Listener daran gebunden:

```
...
<page-specification class="sample.tapestry.page.Login">
   <component id="form" type="Form">
      <binding name="listener" value="listener:doLogin"/>
   </component>
</page-specification>
```

Das zur Seite gehörende HTML-Template (`Login.html`) sieht dann wie folgt aus:

```
...
<form jwcid="form">
   <table>
      <tr>
         <th>User id:</th>
         <td>
            <input jwcid="@TextField" value="userId"size="8"/>
         </td>
      </tr>
      <tr>
         <th>Password:</th>
         <td>
            <input jwcid="@TextField"
               value="password"
               size="8"
               hidden="true"/>
         </td>
      </tr>
      <tr>
         <td colpsan="2">
            <input type="submit" value="Login"/>
```

```
         </td>
      </tr>
   </table>
</form>
...
```

In diesem HTML-Template werden verschiedene Komponenten eingebunden. Die erste Komponente wurde schon zuvor in der Seitenspezifikation aufgeführt und kann deshalb im `form`-Tag über ihre ID referenziert werden. Sie ist zuständig für die gesamte Formularverarbeitung. Zusätzlich werden noch zwei Komponenten vom Type `TextField` eingebunden. Diese stellen Eingabefelder für die `UserId` und das `Password` dar und werden mit dem `value`-Attribute an die entsprechenden Attribute der Page-Klasse gebunden.

Die Page-Klasse stellt dann die entsprechenden Zugriffsmethoden auf die Attribute dar und implementiert die Methode `doLogin()`, welche von der `form`-Komponente aufgerufen wird wenn das Formular abgeschickt wird:

```
Package sample.tapestry.page;

import org.apache.tapestry.IRequestCycle;
import org.apache.tapestry.html.BasePage;

public abstract class Login extends BasePage
{
   public abstract String getUserId();
   public abstract String getPassword();

   public void doLogin()
   {
      // do something to login user
   }
}
```

4.4.3 Komponenten

Die mitgelieferten Standardkomponenten reichen durchaus für die meisten Anwendungsfälle aus, können aber auch an ihre Grenze stoßen, so dass es nötig oder sinnvoll ist, eigene Komponenten zu entwickeln. Die Entwicklung einer Komponente gestaltet sich dabei größtenteils analog zu normalen Tapestry-Seiten, welche ein Spezialfall einer Komponente sind (siehe Abbildung 4.5).

So hat jede Komponente eine Komponentenspezifikation, in der definiert ist, welche Klasse die Komponente implementiert, welche Parameter zulässig sind und wie der Body[1] der Komponente zu verarbeiten ist. Darüber hinaus sind auch alle Definitionen wie in der Seitenspezifikation möglich.

Eine Komponente kann ebenfalls ein HTML-Template besitzen, welches ihre Darstellung definiert. In der zugehörigen Komponentenklasse, welche von `BaseComponent` ableitet, kann jedoch auch die `renderComponente()`-

[1] Damit ist in Tapestry der Kontext zwischen dem öffnenden und dem schließenden HTML-Tag gemeint, welcher auf die Komponente verweist.

Methode überschrieben werden, um eine andere Darstellung zu erzeugen. In der `Foreach`-Komponente wird dies beispielsweise verwendet, um den Komponentenbody mehrfach auszugeben. Daneben dient die Komponentenklasse noch dazu, den Zustand der Komponente in Form von Attributen zu speichern und mittels Listener-Methoden auf Ereignisse zu reagieren.

4.4.4 Klassifikation

Ähnlich wie aktionsgesteuerte Frameworks baut Tapestry bei der *Architektur* auf dem MVC-Ansatz auf. Auch hier gibt es einen zentralen Controller, die Application Engine, welche den Anwendungsablauf und die Zusammenarbeit der Komponenten steuert. Über das Model werden, wie bei vielen anderen Frameworks, keine konkreten Vorgaben gemacht und die Anbindung beispielsweise an Datenbanksysteme oder ähnliches dem Entwickler überlassen.

Den View bilden in Tapestry die UI-Komponenten, wobei es hier jedoch, im Gegensatz zu aktionsgesteuerten Frameworks, eine Besonderheit gibt: Jede Tapestry-Komponente (also auch jede Seite) implementiert das MVC-Muster, als eigenständig wieder verwendbarer Softwarebaustein.

So bildet die Komponentenklasse das Model und den Controller der Komponente, indem sie einerseits den Zustand der Komponente speichert und andererseits auf Benutzereingaben über Listener-Methoden reagiert und den View zur Darstellung auswählt.

Das entsprechende Template (sofern es sich um eine visuelle Komponente handelt) bildet dann den View, indem es die Darstellung der Komponente definiert, auf Daten der Komponentenklasse zugreift und Benutzereingaben an die entsprechenden Listener-Methoden weiterleitet. Ein ähnliches Konzept verfolgt auch Swing zur Implementierung von User- Interface-Komponenten [Sun98]. Hier zeigt sich wieder das Anliegen von ereignisgesteuerten Frameworks, der Programmierung von Desktop-Anwendungen möglichst nah zu kommen.

Tapestry ist im Gegensatz zu den bisher vorgestellten Frameworks ein komponentenbasiertes Framework und besitzt folglich auch eine ereignisgesteuerte *Anwendungssteuerung*. Dies wird z.B. durch die Verwendung der Listener-Methoden besonders deutlich.

Zur *Darstellung* verwendet Tapestry Templates, welche aus statischen HTML-Elementen bestehen und nur mit Hilfe von speziellen Tag-Attributen auf Komponenten verweisen. Diese Komponenten können wiederum eigene Templates zur Darstellung verwenden.

Die *Datenübergabe* zum View erfolgt in Tapestry über Bindung der UI-Komponenten an das Model. Dies wird durch die Angabe der Datenquelle (mit Hilfe des `value`-Attributes) beim Einbinden der Komponente in das HTML-Template definiert.

Die *Dialogsteuerung* kann in Tapestry nur implizit über die Rückgabe der Listener-Methoden definiert werden.

4.5 JavaServer Faces

Name	JavaServer Faces
Entwickler	Sun Microsystems, Inc.
Quelle	http://java.sun.com/j2ee/javaserverfaces/
Betrachtete Version	1.1
Lizenz	Sun Microsystems, Inc. Binary Code License Agreement
Architektur	MVC
Anwendungssteuerung	Ereignisgesteuert
Darstellung	Template (JSP)
Datenübergabe	Bindung
Dialogsteuerung	Endlicher Automat

Tabelle 4.6: Eigenschaften JavaServer-Faces-Spezifikation

JavaServer Faces (JSF) [Sun05d, ABB04] ist eine Spezifikation von Sun und baut auf der J2EE-Spezifikation [Sun05c] auf. Neben der reinen Spezifikation stellt Sun allerdings auch eine Referenzimplementierung bereit. Daneben existiert noch eine Reihe von weiteren Implementierungen. Die bekannteste ist das MyFaces-Projekt [Ap05g] der Apache Foundation. In diesem Abschnitt soll sich jedoch nur auf die grundlegenden Eigenschaften von JavaServer Faces bezogen werden, wie sie in der Spezifikation beschrieben sind.

Ziel der JavaServer-Faces-Spezifikation ist es, die Erfahrungen aus der Entwicklung von Web-Anwendungen, beispielsweise mit Struts (siehe Abschnitt 4.1), mit dem Konzept von komponentenbasierter, ereignisgesteuerter Benutzerschnittstellen wie Swing zu kombinieren [Sun05d, BD04, Ha04]. In der Spezifikation wird dies wie folgt beschrieben [Sun05d]:

- einfache Konstruktion von Eingabeformularen mit Hilfe von wieder verwendbaren Komponenten
- einfache Übertragung von Anwendungsdaten von und zur Oberfläche
- einfaches Management von Oberflächenzuständen über mehrere Server-Anfragen hinweg
- Bereitstellung eines einfachen Modells zur Übertragung von Oberflächenereignissen in den Anwendungscode
- einfache Konstruktion von Oberflächenkomponenten und deren Wiederverwendung

4.5.1 Funktionsweise

Ähnlich wie Struts bauen auch JavaServer Faces auf JSP auf. Hierfür werden auch spezielle Tag-Libraries zur Verfügung gestellt. Diese dienen zur Integration von Komponenten in die Seite. Eine JSP ist bei den JavaServer Faces also weniger für die eigentliche Darstellung, sondern vielmehr dafür zuständig, wie die

Komponenten auf der Seite angeordnet werden. Für die Darstellung ist dann jede Komponente mit Hilfe eines Renderers selber zuständig.

Die folgende JSP stellt einen einfachen Login-Dialog mit Hilfe von JavaServer Faces beispielhaft dar:

```
<%@ page contentType="text/html" language="java" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<html>
<head><title></tille></head>
<body>

<f:view>
   <h:form>
      <h:panelGrid columns=”2”>
         <h:outputLabel value=”Login:”/>
         <h:inputText id=”login” value=”#{loginBean.login}”/>
         <h:outputLabel value=”Password:”/>
         <h:inputSecret id=”password”
            value=”#{loginBean.password}”/>
      </h:panelGrid>
      <h:commandButton id=”submit” value=”Login”
         action=”#{loginBean.doLogin}”/>
   </h:form>
</f:view>

</body>
</html>
```

HTML-Elemente spielen hier eine untergeordnete Rolle und definieren nur noch die Grobstruktur der Seite. Jedes Oberflächenelement wird durch ein spezielles JSF-Tag dargestellt. Das HTML-Formular wird beispielsweise mit `<h:form>` erzeugt, die darin enthaltenen Eingabefelder mit `<h:inputText>` bzw. `<h:inputSecret>`. Üblicherweise erzeugen solche Tags (wie beispielsweise bei Struts) direkt den HTML-Code, wie er auf der Seite dargestellt wird. JavaServer Faces arbeiten an dieser entscheidenden Stelle anders: Über die Tags wird serverseitig ein Komponentenbaum aufgebaut, welcher ähnlich wie bei Swing, die Oberfläche repräsentiert. Der Unterschied zwischen Swing und JavaServer Faces ist nur, dass bei Swing der Komponentenbaum implizit im Code definiert wird anstatt deklarativ innerhalb einer JSP.

Auf dem Server gibt es für jedes Oberflächenelement ein Objekt in diesem Komponentenbaum, dieses hält serverseitig den Zustand des UI-Elementes, z.B. den Inhalt eines Textfeldes. Das Objekt ist auch für seine eigene graphische Darstellung zuständig. Dafür besitzt es einen separaten Renderer. Abbildung 4.6 zeigt einen Teil des aus der obigen JSP-Seite erzeugten Komponentenbaums als Objektdiagramm.

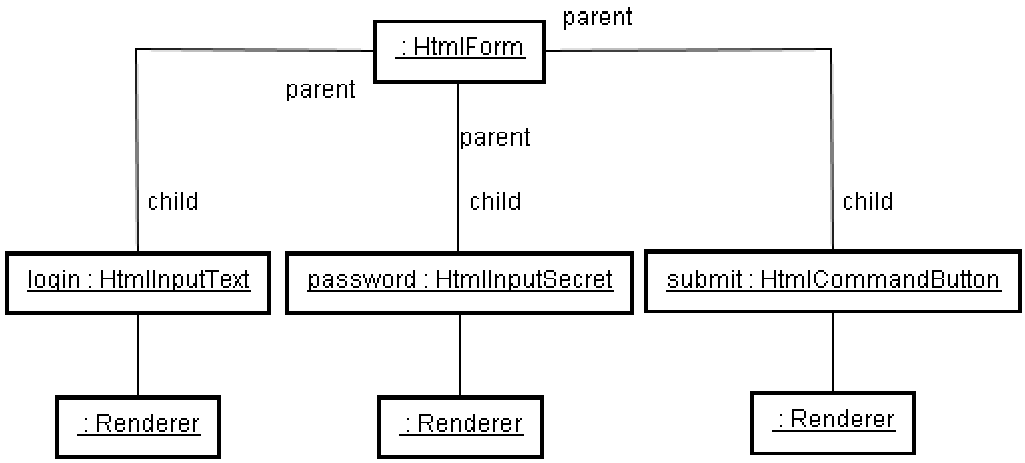

Abbildung 4.6: Beispiel eines Komponentenbaumes für JavaServer Faces als Objektdiagramm

4.5.2 Request-Response-Zyklus

Die JSF-Spezifikation teilt die Verarbeitung einer Anfrage in sechs Phasen ein, welche sequenziell abgearbeitet werden (siehe Abbildung 4.7). Tritt ein Fehler in einer dieser Phasen auf, kann entweder direkt zur letzen Phase gesprungen oder der komplette Zyklus abgebrochen werden. Zwischen den Phasen können auftretende Ereignisse abgearbeitet werden. Die einzelnen Phasen werden in Tabelle 4.7 kurz charakterisiert.

Restore View	Wurde in der letzten Anfrage bereits ein Komponentenbaum für die aktuelle Seite aufgebaut, wird dieser wiederhergestellt, andernfalls wird ein leerer Komponentenbaum aufgebaut.
Apply Request Values	Die Daten aus der Anfrage werden in die entsprechenden Komponenten des Komponentenbaumes übertragen.
Process Validations	Sind Validatoren an eine Komponente gebunden, findet eine Überprüfung des Komponentenzustandes statt.
Update Model Values	Die Daten aus den Komponenten werden in das Model übertragen.
Invoke Application	Die Anwendungslogik wird ausgeführt.
Render Response	Die Antwortseite wird erzeugt, indem sich die Komponenten über ihre Renderer darstellen.

Tabelle 4.7: Phasen des Request-Response-Zyklus bei JavaServer Faces [Sun05d]

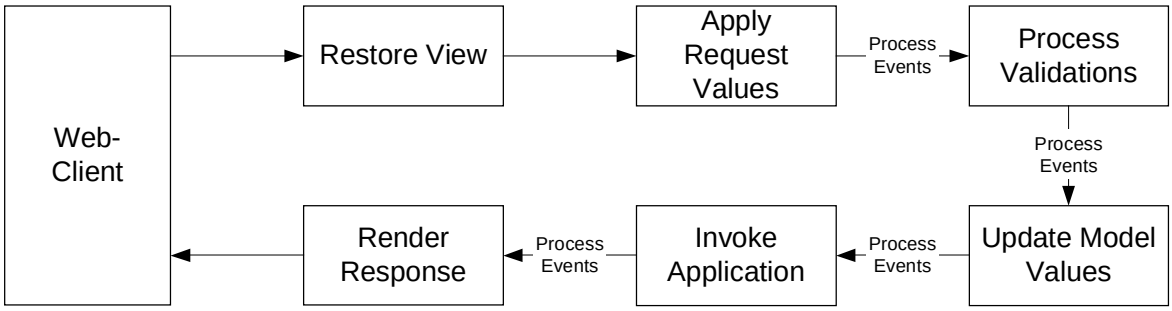

Abbildung 4.7: Request-Response-Zyklus bei JavaServer Faces [Sun05d]

4.5.3 Ereignisverarbeitung

Ähnlich wie Swing bieten JavaServer Faces eine ereignisgesteuerte Anwendungssteuerung. Dazu können Listener bei Oberflächenelementen angemeldet werden. Erzeugt nun ein Element ein Ereignis, z.B. in dem der Nutzer einen Button anklickt, kann dies von der entsprechenden Listener-Methode verarbeitet werden. Dabei gibt es zwei Arten von Ereignissen: Wenn beispielsweise ein Button gedrückt wurde, wird ein `ActionEvent` ausgelöst. Ein `ValueChangeEvent` wird hingegen ausgelöst, wenn sich z.B. der Wert in einem Texteingabefeld geändert hat.

Der folgende Ausschnitt einer JSP zeigt, wie ein Listener an einen Button gebunden wird:

```
<h:commandButton id=”welcome” value=”Welcome” action=”success”>
   <f:actionListener type=”WelcomeListener”/>
</h:commandButton>
```

Klickt man diesen Button an, wird ein entsprechendes `ActionEvent` ausgelöst. Ein entsprechender Listener müsste dann wie folgt aussehen:

```
public class WelcomeListener implements ActionListener
{
   public void processAction(Actionevent event)
   {
      UIComponent comp = event.getComponent();
      // do something to handle event
   }
}
```

4.5.4 Dialogsteuerung

Im Gegensatz zu den meisten anderen vorgestellten Frameworks bieten JavaServer Faces von Haus aus die Möglichkeit, Dialogabläufe explizit zu definieren. Als Modell dient dazu ein einfacher endlicher Automat (siehe Abschnitt 3.4.5.1). Hierbei definieren die einzelnen Seiten die Zustände des Automaten. Die Überführungsfunktion wird dabei in Form von XML angegeben:

```
<navigation-rule>
   <from-view-id>/welcome.jsp</from-view-id>
   <navigation-case>
      <from-outcome>success</from-outcome>
```

```
      <to-view-id>/application.jsp</to-view-id>
   </navigation-case>
   <navigation-case>
      <from-outcome>failure</from-outcome>
      <to-view-id>/problem.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

<from-view-id> und <to-view-id> definiert dabei den Ausgangs- und Zielzustand. <from-outcome> ist die Kantenmarkierung des Automatengraphen und wird durch das Action-Attribute einer UICommand-Komponente definiert:

```
<h:commandButton action="success" value="Welcome"/>
```

Dies kann auch durch Aufruf einer Action-Methode passieren, welche dynamisch einen Wert für <from-outcome> liefert:

```
<h:commandButton action="#{loginBean.logonAction}"
value="Welcome"/>
```

4.5.5 Klassifikation

Die *Architektur* der JavaServer Faces baut, wie bei allen bisher vorgestellten Frameworks, auf dem MVC-Muster auf. Der Controller besteht dabei zum einem aus dem technischen Controller (das FacesServlet), welcher die Anwendung in den Request-Response-Zyklus des Web-Servers einbindet sowie zum anderem aus Listener- und Action-Methoden, welche an Komponenten gebunden werden und auf Benutzereingaben reagieren. Den View bilden die JSP-Seiten, welche die Grundstruktur der Seiten bestimmen und Oberflächenkomponenten einbindet, sowie der Komponentenbaum. Das Model der Anwendung stellen dann JavaBeans dar, welche mit einzelnen Attributen an bestimmte Oberflächenkomponenten gebunden werden können.

JavaServer Faces ist, wie das im vorherigen Abschnitt vorgesellte Tapestry, ein komponentenbasiertes Framework mit einer ereignisgesteuerten *Anwendungssteuerung*.

Bei der *Darstellung* gehen die JavaServer Faces einen uneinheitlichen Weg: Einerseits dient mit JSP eine Template-Technologie zur Definition der Grundstruktur der Seite und zur Einbindung der Komponenten, anderseits wird die Darstellung der einzelnen Komponenten vollständig aus dem Code der jeweiligen Komponente generiert.

Die *Datenübergabe* zwischen Model und Controller wird, wie bei ereignisgesteuerten Frameworks üblich, durch das Binden von Komponentenattributen an Modeldaten und durch das automatische Übertragen, zwischen beiden realisiert. Dies passiert in der vierten Phase des Request-Response-Zyklus (siehe Abschnitt 4.5.2).

Im Gegensatz zu den meisten anderen vorgestellten Frameworks bieten JavaServer Faces die Möglichkeit einer expliziten *Dialogsteuerung*. Als Modell dient dazu ein einfacher endlicher Automat (siehe Abschnitt 4.5.4).

4.6 ASP.NET

Name	ASP.NET
Entwickler	Microsoft Corporation
Quelle	http://msdn.microsoft.com/asp.net/
Lizenz	Frei verfügbar
Betrachtete Version	1.1
Architektur	MVC
Anwendungssteuerung	Ereignisgesteuert
Darstellung	Template
Datenübergabe	Bindung
Dialogsteuerung	Implizit

Tabelle 4.8: Eigenschaften ASP.NET

Bei ASP.NET [Ms05] handelt es sich um die Weiterentwicklung von Microsofts Active-Server-Pages-Technologie (ASP) im Rahmen der .NET-Strategie (siehe Abschnitt 2.6.2.2). Bei ASP.NET handelt es sich also nicht um ein eigenständiges Framework, sondern vielmehr um den Teil des .NET-Frameworks, welcher sich mit der Entwicklung von Web-Anwendungen beschäftigt.

ASP.NET ist zwar weder ein Open-Source-Produkt, noch basiert es auf der J2EE-Plattform. Es soll hier aber trotzdem vorgestellt werden, da es im Bereich der .NET-Plattform eine wichtige Rolle spielt und dort häufig zur Entwicklung von Web-Oberflächen eingesetzt wird. In diesem Zusammenhang kann es auch als direkter Gegenspieler zu den JavaServer Faces gesehen werden, da beide ein sehr ähnliches Konzept, jedoch auf Basis verschiedener Plattformen, verfolgen.

4.6.1 Funktionsweise

Grundsätzlich baut ASP.NET auf einem ähnlichen Konzept wie JavaServer Faces auf. Statt JSP werden jedoch ASP-Seiten verwendet. Dabei handelt es sich ähnlich wie bei JSP um HTML-Seiten, in die mit Hilfe spezieller Tags Code-Fragmente eingefügt werden können. Dabei kann eine beliebige .NET-kompatible Sprache, wie beispielsweise C# oder Visual Basic, verwendet werden. In ASP.NET fungieren sie jedoch, ähnlich wie JSP bei JavaServer Faces, als Container für Oberflächenkomponenten (so genannte *Web Forms*), welche über spezielle Tags eingebunden werden. Weiterhin gehört zu jeder ASP.NET-Seite noch eine entsprechende Seitenklasse. Diese ist für die Verarbeitung von Ereignissen auf der Seite zuständig.

Im Folgenden wird eine Login-Seite beispielhaft dargestellt:

```
...
<html>
   <head><title>LoginPage</title></head>
   <body>
      <form method="post" runat="server">
```

```
         <asp:Label id="loginLabel" runat="server">
            Login
         </asp:Label>
         <asp:TextBox id="login" runat="server" />
         <asp:RequiredFieldValidator id="Validator1"
            runat="server"
            ErrorMessage="Login is required"
            ControlToValidate="login"/>
         <br/>
         <asp:Label id="passwordLabel" runat="server">
            Password
         </asp:Label>
         <asp:TextBox id="password" runat="server" /><br/>
         <asp:Button id="loginButton"
            runat="server" Text="Login" />
      </form>
   </body>
</html>
```

Für jedes Tag auf der Seite, welches ein Attribut `runat="server"` besitzt, existiert ein serverseitiges Objekt. Auf diese Objekte kann dann von der Seitenklasse aus zugegriffen werden. Eine zur obigen Seite gehörende Seitenklasse könnte dann wie folgt aussehen:

```
...
public class LoginPage : System.Web.UI.Page
{
   protected System.Web.UI.WebControls.TextBox login;
   protected System.Web.UI.WebControls.Button loginButton;
   protected System.Web.UI.WebControls.Label loginLabel;
   protected System.Web.UI.WebControls.Label passwordLabel;
   protected System.Web.UI.WebControls.TextBox password;

   override protected void OnInit(EventArgs e)
   {
      this.loginButton.Click +=
         new System.EventHandler(this.loginButton_Click);
      this.Load += new System.EventHandler(this.Page_Load);

      base.OnInit(e);
   }

   private void loginButton_Click(object sender, System.EventArgs
e)
   {
      string loginString = login.Text;
      string passwordString = password.Text;

      // do something to logon user
   }
}
...
```

In der Seitenklasse existiert für jeden Tag ein entsprechendes `WebControl`-Objekt. An diese Objekte können dann für bestimmte Ereignisse EventHandler

angemeldet werden. In diesem Beispiel wird für das Anklicken des `loginButton` die Methode `loginButton-Click()` als EventHandler definiert. Auf ähnliche Weise lassen sich für alle Benutzeraktionen Ereignisbehandlungsmethoden schreiben.

4.6.2 Klassifikation

Bei der *Architektur* lassen sich ASP.NET-Anwendungen auch als Umsetzung des MVC-Musters auffassen, wobei Microsoft jedoch vom „Page Controller Pattern“ spricht, da hier nicht (wie bei den meisten J2EE-basierten Frameworks) ein einzelnes Servlet als Controller dient, sondern jede Seite ihren eigenen Controller (in Form einer Seitenklasse) besitzt. Als View dienen die beschriebenen ASP-Seiten, welche ähnlich wie JSP einen Template-basierten Ansatz verfolgen. Für das Model stehen verschiedene Komponenten aus dem .NET-Framework bereit, wie beispielsweise `DataSet`-Komponenten, welche eine einfache Anbindung an Datenquellen wie z.B. Datenbanken ermöglichen.

Zur *Anwendungssteuerung* verwendet ASP.NET einen ereignisgesteuerten Ansatz. Dazu werden EventHandler-Methoden innerhalb der Seitenklassen verwendet, welche auf verschiedene Ereignisse von Oberflächekomponenten reagieren können.

Zur *Darstellung* verwendet ASP.NET eine Erweiterung der templateorientierten ASP-Technologie. Diese ermöglicht es, neben der Verwendung von Anwendungscode innerhalb der Templates, auch ASP.NET-Komponenten (so genannte *Web Forms*) über spezielle Tags einzubinden.

Die *Datenübergabe* an den View erfolgt über das Binden der Oberflächenkomponenten an Datenquellen. Dazu können in der Seitenklasse den Attributen der Oberflächenkomponenten die entsprechenden Daten zugewiesen werden.

Eine Unterstützung für eine explizite *Dialogsteuerung* bietet ASP.NET nicht.

4.7 wingS

Name	wingS
Entwickler	wingS Project Team
Quelle	http://j-wings.org/
Lizenz	GNU Lesser General Public License
Betrachtete Version	1.0.5
Architektur	MVC
Anwendungssteuerung	Ereignisgesteuert
Darstellung	Generierung
Datenübergabe	Bindung
Dialogsteuerung	Implizit

Tabelle 4.9: Eigenschaften wingS-Frameworks

Die Entwicklung von Web-Anwendungen mit wingS [Wi05] ähnelt sehr stark der Entwicklung von Benutzeroberflächen mit Swing. Wie auch in Swing lassen sich diese visuellen Komponenten beliebig erweitern und über Containerklassen zu kompletten Dialogen kombinieren.

Mit wingS wollen die Entwickler zeigen, dass es möglich ist, Web-Anwendungen auch vollständig über Java-Komponenten zu entwickeln und damit ein Gegengewicht zu JavaServer Faces mit seinem „seitenorientierten HTML-Java-Mix“ bilden [Sc05].

4.7.1 Aufbau

Oberflächen werden in wingS durch das Erstellen eines hierarchischen Objektbaumes erzeugt, wofür ein erweiterbarer Satz an visuellen Komponenten bereit steht. Neben einfachen Eingabeelementen, Labels und Containers gehören dazu auch komplexere Komponenten wie Tabellen, Bäume und ToolTips. Wie beim Vorbild Swing erben alle Komponenten von einer gemeinsamen Klasse `SComponent`.

Wie auch in Swing lassen sich diese visuellen Komponenten beliebig erweitern und über Containerklassen zu kompletten Dialogen kombinieren. Im Folgenden wird ein einfacher Login-Dialog beispielhaft dargestellt:

```
import org.wings.*;

public class Login
{
   public Login()
   {
      SFrame frame = new SFrame();
      SGridLayout layout = new SGridLayout(2);
      SForm panel = new SForm(layout);

      panel.add(new SLabel(“Login:”));
```

```
      STextField loginField = new STextField();
      panel.add(loginField);

      panel.add(new SLabel("Password:"));
      STextField passwordField = new STextField()
      panel.add(passwordField);

      SButton loginButton = new SButton("Login");
      panel.add(loginButton);

      frame.getContextPane().add(panel);
      frame.show();
   }
}
```

Für die Darstellung der Komponenten sind spezifische Render-Klassen verantwortlich, die als Sammlung austauschbare *Pluggable Look & Feels* bilden. Diese Renderer arbeiten die gesetzten visuellen und funktionalen Eigenschaften der Komponente in ihren generierten HTML-Code ein.

4.7.2 Ereignisverarbeitung

Zur Verarbeitung von Benutzereingaben verwendet wingS ein in weiten Teilen von Swing adaptiertes Event-Konzept. So bieten die Oberflächenkomponenten Methoden zum Registrieren von Listener-Objekten für bestimmte Ereignisse an. Die Implementierung der Listener kann dann die aktuellen Werte der Eingabekomponenten abfragen und weiterverarbeiten. Für das obige Beispiel könnte dies wie folgt aussehen:

```
...
loginButton.addActionListener(new ActionListener{
   public void actionPerformed(ActionEvent e)
   {
      String login = loginField.getText();
      String password = passwordField.getText();

      // check login
   }
});
...
```

4.7.3 Klassifikation

Für die *Architektur* wird in wingS das MVC-Muster, ähnlich wie bei JavaServer Faces, umgesetzt. So bildet das `wingSServlet`, welches die Anbindung an den Request-Response-Zyklus umsetzt, und die Event-Listener den Controller. Den View bildet der Komponentenbaum mit dem dazu gehörigen Render-Klassen. Die Umsetzung des Models wird hingegen vollständig dem Entwickler überlassen. Daneben implementieren die einzelnen Oberflächenkomponenten, ähnlich wie Swing-Komponenten [Sun98], auch das MVC-Muster.

Die *Anwendungssteuerung* erfolgt wie bei Swing ereignisgesteuert über Listener-Objekte, welche bei Oberflächenkomponenten für Ereignisse registriert werden können.

Für die *Darstellung* werden im Gegensatz zu JavaServer Faces oder Tapestry jedoch keine Templates zur Anordnung der Oberflächenkomponenten auf der Seite verwendet. Vielmehr werden die Darstellung und das Layout einer Seite komplett aus den Komponenten heraus generiert. Dieses Konzept ist stark an die Funktionsweise von Swing angelehnt.

Die *Datenübergabe* erfolgt auch bei wingS über das Binden von Modeldaten an Oberflächenkomponenten. Dazu können den Attributen der Oberflächenkomponenten entsprechende Daten zugewiesen werden.

Eine Unterstützung für eine explizite *Dialogsteuerung* bietet wingS nicht.

4.8 Echo

Name	Echo
Entwickler	NextApp, Inc.
Quelle	http://www.nextapp.com/products/echo2/
Betrachtete Lizenz	Mozilla Public License oder GNU Lesser General Public License
Version	2.0.rc1
Architektur	MVC
Anwendungssteuerung	Ereignisgesteuert
Darstellung	Generierung
Datenübergabe	Bindung
Dialogsteuerung	Implizit

Tabelle 4.10: Eigenschaften Echo-Framework

Das Echo-Framework [Ne05] verfolgt ein ähnliches Konzept wie wingS im vorherigen Abschnitt. Auch hier erfolgt die Entwicklung der Benutzeroberfläche allein durch Verwendung einer Swing-ähnlichen Programmierschnittstelle. Anders als bei wingS (und allen anderen hier vorgestellten Frameworks) basiert die Kommunikation zwischen Browser und Web-Server jedoch nicht auf dem Request-Response-Zyklus, sondern auf der Verwendung des AJAX-Ansatzes (siehe Abschnitt 2.6.1.4).

Echo versucht also nicht nur die Entwicklung von Web-Anwendungen ähnlich zu Desktop-Anwendungen zu gestalten (wie beispielsweise bei wingS), sondern auch deren Interaktionsmöglichkeiten nachzubilden, d.h. eine Echo-Anwendung wird nicht nur wie eine Desktop-Anwendung entwickelt, sondern soll sich auch so verhalten.

4.8.1 Funktionsweise

Für die Entwicklung der Oberfläche stellt das Echo-Framework verschiedene Komponenten, wie beispielsweise Buttons, Textfelder usw. zur Verfügung. Daraus kann nach dem Baukastenprinzip die Oberfläche zusammengebaut werden. Dies geschieht, wie bei Swing, in Form von Java-Klassen. An die Instanzen der einzelnen Komponenten können dann Event-Listener zur Ereignisbehandlung gebunden werden. Daneben dienen sie noch zur Repräsentation des Zustandes der Oberflächenkomponenten auf dem Server.

Dieses Konzept ermöglicht auch ein einfaches Sitzungsmanagement: So wird für jede Benutzersitzung eine Instanz der entsprechenden Web-Anwendung auf dem Server erzeugt, d.h. alle in Objektvariablen gespeicherten Daten sind in der aktuellen Benutzersitzung verfügbar. Statische Variablen können dagegen von allen Programminstanzen (also allen Benutzersitzungen) gemeinsam genutzt werden.

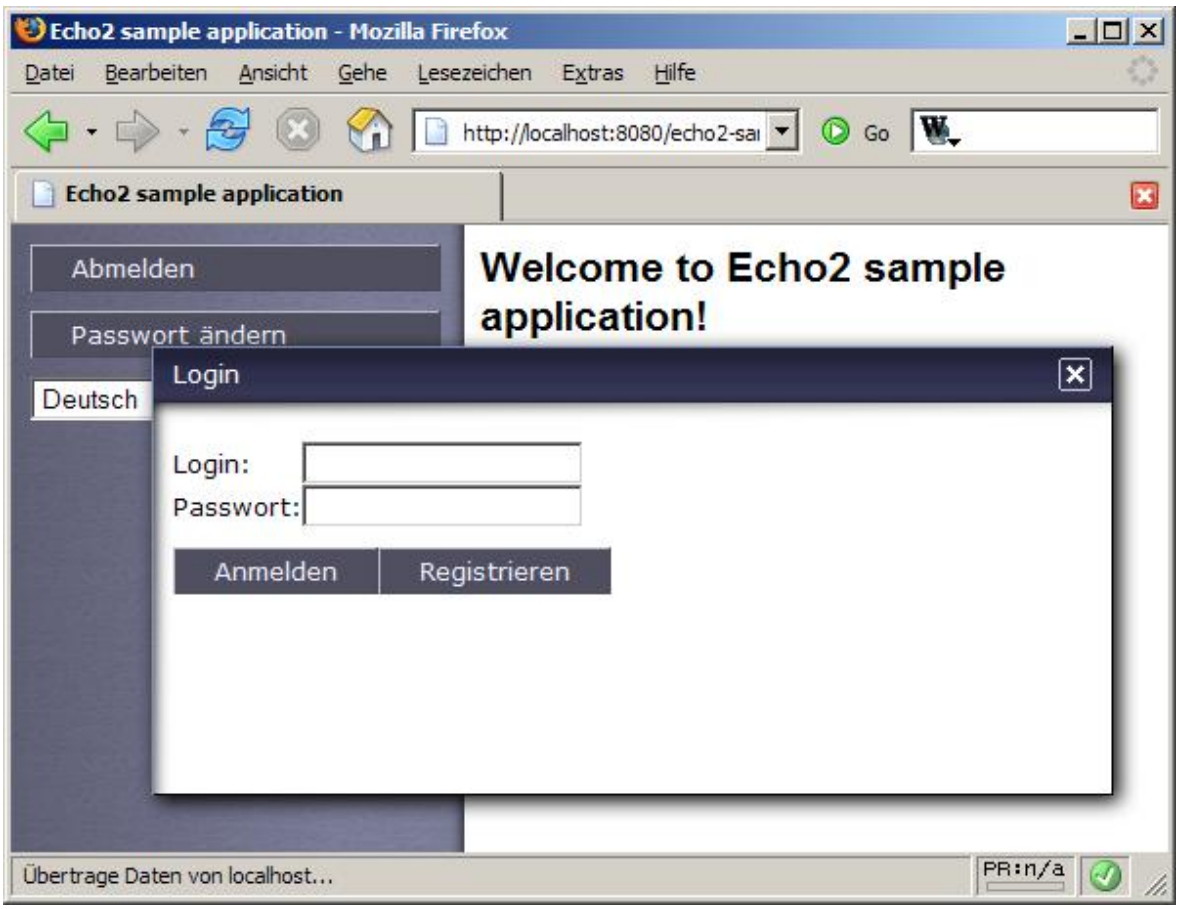

Abbildung 4.8: Darstellung von Fenstern innerhalb des Browsers mit Hilfe des Echo-Frameworks

Eine mit Echo entwickelte Web-Anwendung besteht immer aus einer einzelnen HTML-Seite. Diese wird beim ersten Aufruf der Anwendung vom Web-Server geladen und bleibt die gesamte Sitzung über aktiv.

Abbildung 4.8 zeigt z.B. die Darstellung eines Login-Dialoges. Hier wird auch die Möglichkeit gezeigt, Fenster innerhalb des Browsers zu darzustellen. Dadurch

ergeben sich ähnliche Möglichkeiten, Anwendungen mit mehreren Unterfenstern zu entwickeln, wie dies häufig bei Desktop-Anwendungen der Fall ist. Technisch wird dies vom Framework durch die umfangreiche Verwendung von JavaScript innerhalb des Browsers realisiert.

4.8.2 Ereignisverarbeitung

Ereignisse im Browser, wie beispielsweise das Anklicken eines Buttons oder Eingaben in einem Textfeld, werden durch JavaScript mit Hilfe von asynchronen Anfragen an den Web-Server weitergeleitet. Auf Serverseite wird dann der Zustand der jeweiligen Objekte aktualisiert und für das Ereignis registrierte EventListener-Methoden ausgeführt. Dadurch wird erreicht, dass das DOM im Browser stets mit den Zuständen der Komponenten auf der Serverseite synchronisiert ist.

Anschließend werden die Teile der Oberfläche, welche aktualisiert werden müssen, in Form von XML an den Browser geschickt. Im Browser werden die entsprechenden Teile der Seite mittels JavaScript durch Manipulation des DOM aktualisiert. Die eigentliche HTML-Seite wird dabei jedoch nicht neu geladen.

Durch dieses Konzept, nur die wirklich benötigten Teile einer HTML-Seite neu zu laden, ergibt sich in den meisten Fällen ein deutlicher Performancevorteil gegenüber Web-Anwendungen, welche den klassischen Request-Response-Zyklus verwenden.

4.8.3 Klassifikation

Ähnlich wie Swing [Sun98] setzt auch Echo ein MVC-Ansatz bei der *Architektur* um. Dabei implementiert jede Oberflächenkomponente das MVC-Muster als eigenständig wieder verwendbarer Softwarebaustein. Jede Komponente besteht dabei aus einer Komponenten-Klasse (als View), welche alle nötigen Informationen zur Darstellung der Komponenten besitzt. Bei diesen Klassen können auch EventListener registriert werden, welche auf Ereignisse der Komponente reagieren und als Controller dienen. Ein- und Ausgabekomponenten besitzen zusätzlich noch eine Modelklasse, welche den Zustand der Komponenten kapselt.

Die *Anwendungssteuerung* erfolgt, wie bei den meisten zuletzt vorgestellten Frameworks, ereignisgesteuert über EventListener-Methoden. Diese werden für Ereignisse bei Oberflächenkomponenten registriert und können deren Zustand verändern.

Die *Darstellung* der Anwendung wird vollständig aus den Komponenten generiert, wobei die Darstellung jeder einzelnen Komponente bei Bedarf mittels AJAX an den Browser übertragen wird. Im Browser werden dann die entsprechenden Teile der Seite mittels JavaScript durch Manipulation des DOM aktualisiert. Die eigentliche HTML-Seite wird dabei jedoch nicht neu geladen.

Die *Datenübergabe* erfolgt bei Echo über das Binden von Modeldaten an Oberflächenkomponenten. Dazu können den Attributen der Oberflächenkomponenten entsprechende Daten zugewiesen werden.

Ein Unterstützung für eine explizite *Dialogsteuerung* bietet Echo nicht.

4.9 Cocoon

Name	Cocoon
Entwickler	Apache Software Foundation
Quelle	http://cocoon.apache.org/
Lizenz	Apache Software License 2.0
Betrachtete Version	2.1
Architektur	Pipeline
Anwendungssteuerung	-
Darstellung	Transformation
Datenübergabe	-
Dialogsteuerung	Implizit oder als Pogramm (mittels „Control Flow“)

Tabelle 4.11: Eigenschaften Cocoon-Framework

Das Apache Cocoon Framework [Ap05b] basiert weitgehend auf XSLT und lässt sich wegen seines sehr eigenständigen Konzeptes nicht in die zuvor entwickelten Kategorien für MVC-Frameworks einordnen. Vielmehr orientiert sich die Architektur an einem Pipeline-Ansatz. Die Kernidee dahinter ist es, Inhalte aus statischen oder dynamischen Quellen zu lesen und mittels Transformationen in Ausgabeformate wie HTML oder PDF zu transformieren. Die einzelnen Komponenten, welche in der Pipeline angeordnet werden, arbeiten dabei völlig unabhängig und tauschen Daten nur über XML aus.

Das Konzept, beidem jede Komponente nur für eine bestimmte Aufgabe zuständig ist, wird auch als *Trennung der Zuständigkeit* (speration of concerns) bezeichnet und stellt neben der Komponenten-basierten Entwicklung ein zentrales Anliegen des Cocoon-Frameworks dar. Dies soll es ermöglichen, eine Web-Anwendung in Cocoon nach dem Baukasten-Prinzip zu entwickeln, ohne etwas programmieren zu müssen.

Cocoon kann mit vielen Datenquellen zusammenarbeiten, wie beispielsweise dem Dateisystem, relationalen oder XML-basierten Datenbanksystemen, Web-Services oder SAP-Systemen. Als Ausgabeformate stehen unter anderem HTML, WML, PDF, SVG (Scalable Vector Graphics) und RTF (Rich Text Format) zur Verfügung. Außerdem werden Funktionen zur Formulargenerierung und Verarbeitung (Cocoon Forms), zur Dialogsteuerung (Cocoon Flow) und zur Entwicklung von Portalen auf Basis von Cocoon bereitgestellt.

Ein weiterer interessanter Aspekt von Cocoon ist, dass es als abstrakte Maschine entworfen wurde, d.h. es ist nicht an eine konkrete Umgebung gebunden. Man kann Cocoon sowohl als Standalone-Applikation, als auch integriert in eine andere Java-Anwendung oder als Web-Anwendung in einem beliebigen Servlet-Container

betreiben. Die letzte Variante bildet allerdings den Schwerpunkt und wird mit Abstand am häufigsten verwendet.

4.9.1 Aufbau und Funktionsweise

Das Kernelement jeder Cocoon-Anwendung sind die so genannten Komponenten-Pipelines. Diese werden in einer zentralen XML-Konfigurationsdatei, der so genannten Sitemap, definiert. Hier werden unter anderem die verwendeten Komponenten konfiguriert und zu den entsprechenden Pipelines verbunden. Eine einfache Sitemap könnte dabei wie folgt aussehen [Wa04]:

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

   <!-- Components ================================ -->
   <map:components>
      <map:generators default="file"/>
      <map:transformers default="xslt"/>
      <map:readers default="resource"/>
      <map:serializers default="html"/>
      <map:matchers default="wildcard"/>
      <map:selectors default="browser"/>
   </map:components>

   <!-- Pipelines ================================= -->
   <map:pipelines>
      <map:pipeline>
         <map:match pattern="sample">
            <map:generate src="samples.xml"/>
            <map:transform
               src="context://samples/simple-samples2html.xsl">
            </map:transform>
            <map:serialize/>
         </map:match>
         <map:match pattern="order">
            <map:generate src="stream/OrderPage.xml"/>
            <map:transform
               src="context://samples/dynamic-page2html.xsl">
               <map:parameter
                  name="sitemapURI"
                  value="{request:sitemapURI}"/>
               <map:parameter name="file" value=".xsp"/>
            </map:transform>
            <map:serialize type="html"/>
         </map:match>
      </map:pipeline>
   </map:pipelines>
</map:sitemap>
```

Nach XML-Header und Namespace-Angabe in den ersten beiden Zeilen werden im Abschnitt `<map:components>` die einzelnen Komponententypen konfiguriert. In diesem Fall wird für jeden Typ nur definiert, welche spezielle Komponente standardmäßig verwendet werden soll. Grundsätzlich wird dabei in Cocoon zwischen folgenden Komponententypen unterschieden:

- *Generatoren* stellen Datenquellen dar, welche XML-Daten erzeugen und initialisieren die Pipeline-Verarbeitung.
- *Transformatoren* transformieren XML-Daten in XML-Daten einer anderen Struktur, in diesem Fall beispielsweise mittels XSLT-Stylesheets.
- *Serializer* schließen die Pipeline ab und transformieren die XML-Daten in das gewünschte Ausgabeformat für den Client.
- *Selektoren* ermöglichen bedingte Verzweigungen und Fallunterscheidungen innerhalb der Pipeline.
- *Matcher* dienen dazu, URLs bestimmten Pipeline-Abschnitten zuzuordnen. Dies kann z.B. über Muster passieren, wie im obigen Beispiel.
- *Actions* können innerhalb der Pipeline Anwendungslogik ausführen und den weiteren Ablauf der Pipeline verändern, indem sie Parameter setzen, welche später von Selektoren verarbeitet werden.
- *Reader* dienen als Start- und Endpunkt einer Pipeline und fassen die Funktionalität von Generatoren und Serializern zusammen. Reader sind vor allem nützlich, um Ressourcen (wie Bilder und ähnliches) auszuliefern.

Im darauf folgenden Abschnitt `<map:pipelines>` wird eine Pipeline `<map:pipeline>` definiert. Danach wird mit einem Matcher die Pipeline aufgespalten, der erste Zweig `<map:match pattern="">` wird durchlaufen, wenn kein virtueller Pfad in der URL mitgegeben wurde, der zweite `<map:match pattern="order">`, falls die URL den virtuellen Pfad „order“ enthält. Im ersten Zweig wird dann die XML-Datei „samples.xml“ mittels eines Generators eingelesen und anschließen durch einen Transformator mit einem XST-Stylesheet transformiert. Am Ende gibt ein Serializer das Ergebnis als HTML aus. Im zweiten Zweig liest ein Generator die Datei „stream/OrderPage.xml“ ein. Diese wird dann mit einem anderen XST-Stylesheet transformiert, wobei noch zwei Parameter übergeben werden. Anschließend wird das Ganze von einem Serializer in HTML umgewandelt.

Grundsätzlich lässt sich eine Cocoon-Pipeline (wie beispielhaft in Abbildung 4.9 dargestellt) in die folgenden drei Abschnitte unterteilen:

1. Zunächst wandelt ein *Generator* die Daten einer Datenquelle in einen XML-Datenstrom um. So erzeugt beispielsweise der HTML-Generator die XML-Darstellung einer HTML-Seite, der Directory-Generator die XML-Darstellung eines Verzeichnisses, und der Request-Generator wandelt eine eingehende Anfrage in eine XML-Darstellung um. Prinzipiell kann hier jegliche Datenquelle angesprochen werden, sofern es hierfür einen passenden Generator gibt. Notfalls muss eben ein neuer Generator entwickelt werden.

2. Im zweiten Schritt wandeln ein oder mehrere *Transformatoren* den eben erzeugten XML-Datenstrom in eine neue Form. Als Ergebnis erhält man wieder XML. So wandelt zum Beispiel der XSLT Transformer auf Grundlage eines XSLT-Stylesheets den eingehenden XML-Datenstrom entsprechend um. Auch hier stehen wieder unterschiedliche Komponenten zur Auswahl: So kann der SQL Transformer den eingehenden Datenstrom in eine

Datenbankabfrage umwandeln, diese ausführen und das Ergebnis wiederum als XML-Datenstrom darstellen. An den SQL Transformer könnte nun der Filter Transformer angeschlossen werden, der die ersten 10 Zeilen des Abfrageergebnisses aus dem Datenstrom herausfiltern könnte.

3. Im letzten Schritt wird die Pipeline durch einen *Serializer* abgeschlossen, dessen Aufgabe es ist, den erhaltenen XML-Datenstrom in das gewünschte Ausgabeformat umzuwandeln. So existieren unter anderem Serializer für HTML, WML, PDF oder auch Microsoft Excel.

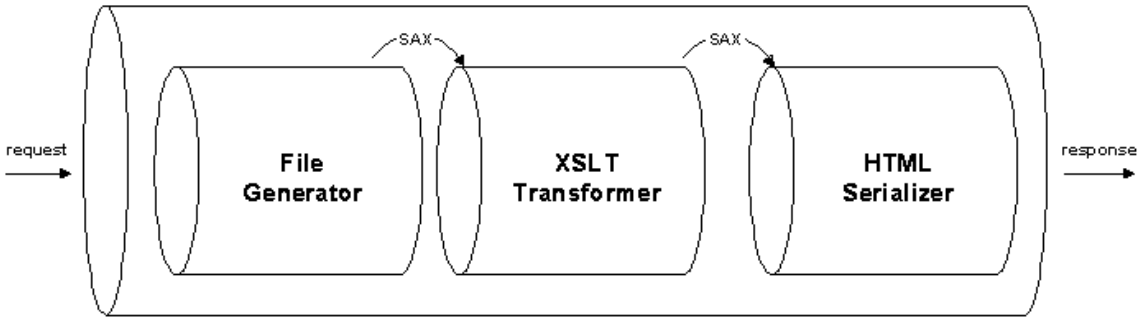

Abbildung 4.9: Schematische Darstellung einer möglichen Cocoon-Pipeline [Ap05b]

Eine Pipeline kann aber auch mehrere Verzweigungen besitzen. Hier können Selektoren den Fluss innerhalb der Pipeline beeinflussen. Durch die Verwendung eines Selektors, der auf bestimmte Parameter reagieren kann, lässt sich auch eine Ablauflogik innerhalb der Pipelines festlegen. Diese Parameter lassen sich über Aktionen verändern, welche ebenfalls in die Pipeline integriert werden können und so indirekt den weiteren Ablauf innerhalb der Pipeline verändern können.

4.9.2 Klassifikation

Das Cocoon-Framework lässt sich wegen seines sehr eigenständigen Aufbaus nicht in die für MVC-Frameworks entwickelten Kategorien einordnen. Vielmehr verfolgt es schon von der *Architektur* her ein ganz anderes Konzept. Es stellt eher eine abstrakte Maschine zur Verarbeitung von XML-Datenströmen dar. Schon aus diesem Grund hat es eine eigene Kategorie neben den MVC-Frameworks verdient.

Auch lassen sich solche Eigenschaften wie *Anwendungssteuerung* und *Datenübergabe* von MVC-Frameworks nicht auf Cocoon übertragen.

Die Erzeugung der *Darstellung* erfolgt grundsätzlich mittels Serializern im letzen Schritt einer Pipeline. Dadurch ist es beispielsweise möglich, ohne großen Aufwand, Seiten für verschiedene Ausgabekanäle (Browser, mobiles Endgerät usw.) zu serialisieren.

Die *Dialogsteuerung* funktioniert in Cocoon entweder implizit oder mittels „Control Flow“ (siehe dazu Abschnitt 3.4.5.2). Solche Programmscripte zur Dialogsteuerung werden als *FlowScript* bezeichnet und können beispielsweise über

```
<map:flow language="javascript">
   <map:script src="myApplication.js"/>
</map:flow>
```

innerhalb der Sitemap aufgerufen werden. Dabei wird zurzeit nur ein spezieller JavaScript-Dialekt als Programmiersprache unterstützt.

4.10 Fazit

In diesem Kapitel wurde ein Überblick über die wichtigsten Frameworks zur Entwicklung von Web-Anwendungen gegeben. Dazu wurden die Grundkonzepte der einzelnen Frameworks erläutert und diese in die in Kapitel 3 entwickelte Klassifikation eingeordnet. Tabelle 4.12 gibt einen Überblick über alle vorgestellten Frameworks und deren Klassifikationsmerkmalen.

Alle vorgestellten Frameworks lassen sich dabei (bis auf einen Spezialfall) in die folgenden drei Typen einteilen:

- Bei dem ersten Typ handelt es sich um klassische aktionsgesteuerte Frameworks, welche eine Template-Technologie zur Darstellung nutzen. Hierzu gehören Struts und Turbine, welche sich beide stark am Request-Response-Zyklus einer Web-Anwendung orientieren.
- Der zweite Typ umfasst komponentenbasierte ereignisgesteuerte Frameworks, welche ebenfalls eine Template-Technologie zur Darstellung bzw. Einbindung von Komponenten in Web-Seiten verwenden. Von diesem Type sind Frameworks wie Tapestry, JavaServer Faces und ASP.NET. Diese Frameworks bieten alle die Möglichkeit, die Entwicklung der Web-Anwendung durch Verwendung von wieder verwendbaren Komponenten zu beschleunigen. Durch die Verwendung von Templates werden die Gestaltungsmöglichkeiten der Oberfläche jedoch kaum eingeschränkt.
- Bei dem dritten Typ handelt es sich ebenfalls um komponentenbasierte ereignisgesteuerte Frameworks. Bei der Darstellung wird jedoch auf die Verwendung von Templates verzichtet und die Darstellung vollständig aus den Layoutinformationen der einzelnen Komponenten und deren Beziehungen zueinander erzeugt. Dies ähnelt stark dem Konzept von Swing zur Erzeugung von Benutzeroberflächen für Desktop-Anwendungen. Zu diesem Typ gehören die Frameworks wingS und Echo.

Bei dem Spezialfall handelt es sich um das Framework *Cocoon*, welches sich schon allein vom Architekturprinzip stark von den anderen Frameworks unterschiedet und somit nicht einem der drei Typen entspricht.

Framework	Architektur	Anwendungs-steuerung	Darstellung	Daten-übergabe	Dialog-steuerung
Struts	MVC	Aktionsgesteuert	Template	Pull	Implizit
Turbine	MVC	Aktionsgesteuert	Template	Pull / Push	Implizit
Tapestry	MVC	Ereignisgesteuert	Template	Bindung	Implizit
JavaServer Faces	MVC	Ereignisgesteuert	Template	Bindung	Endlicher Automat
ASP.NET	MVC	Ereignisgesteuert	Template	Bindung	Implizit
wingS	MVC	Ereignisgesteuert	Generierung	Bindung	Implizit
Echo	MVC	Ereignisgesteuert	Generierung	Bindung	Implizit
Cocoon	Pipeline	-	Transformation	-	Implizit / Programm

Tabelle 4.12: Überblick über die wichtigsten Klassifikationsmerkmale der vorgestellten Frameworks

Kapitel 5

Anwendungsbeispiel

In diesem Kapitel wird eine einfache Web-Anwendung prototypisch mit jeweils einem Vertreter der in Kapitel 4 identifizierten Typen umgesetzt. Diese Prototypen setzten typische Anforderungen an große Web-Anwendungen um, wie beispielsweise die Anmeldung und Verwaltung von Nutzern. Sie implementieren jedoch keine eigentliche Fachlichkeit. Ziel dieses Kapitels ist es, anhand der Implementierungen Rückschlüsse auf die Praxistauglichkeit und Besonderheiten des jeweiligen Ansatzes ziehen zu können. Es werden dabei, folgende Frameworks verwendet:

- *Struts* (Abschnitt 4.1) als ein Vertreter der aktionsgesteuerten Frameworks.
- *JavaServer Faces* (Abschnitt 4.5) in der MyFaces-Implementierung als ein Vertreter der komponentenbasierten ereignisgesteuerten Frameworks, welche eine Template-Technologie zur Darstellung und Konfiguration der UI-Komponenten verwenden.
- *Echo* (Abschnitt 4.8) als ein Vertreter der komponentenbasierten ereignisgesteuerten Frameworks, welche die Darstellung vollständig aus den UI-Komponenten generieren.

5.1 Anforderungen

Im Folgenden werden die Anforderungen an die Anwendungen formuliert. Dabei geht es um grundsätzliche Anforderungen an die Benutzeroberfläche, wie sie häufig an große Web-Anwendungen gestellt werden. Ziel ist es nicht, eine vollständig einsetzbare Anwendung oder eine bestimmte Fachlichkeit umzusetzen, vielmehr soll gezeigt werden, ob und wie sich bestimmte Anforderungen mit den entsprechenden Frameworks umsetzen lassen.

Aus diesem Grund sind die zu entwickelnden Anwendungen als Prototypen auszulegen. Das Hauptaugenmerk liegt dabei auf der Verwendung des Frameworks zur Gestaltung der Benutzeroberfläche. Anbindungen an Back-End-Systeme, wie beispielsweise Datenbanksysteme, sind in diesen Prototypen grundsätzlich nicht nötig und können, wenn sie doch benötigt werden, geeignet simuliert werden.

In den folgenden Absätzen werden nun konkrete Anforderungen formuliert:

(1) Ein wichtiger Aspekt beim praktischen Einsatz von Web-Anwendungen ist häufig die Authentizität des Nutzers (siehe Abschnitt 2.3). Typischerweise wird dies dadurch gelöst, dass der Nutzer sich anmelden muss, bevor er bestimmte Funktionen nutzen kann. Meist werden den Nutzern auch bestimmte Rollen zugeordnet, um so die Rechte einzelner Nutzer einschränken zu können (Autorisierung). So soll ein Nutzer mit der Rolle „Administrator“ die Möglichkeit besitzen, die anderen Nutzer zu verwalten, während hingegen ein normaler Nutzer nur Zugriff auf seine eigenen Daten haben soll.

(2) In diesem Zusammenhang ist es sinnvoll, dass sich potenzielle neue Nutzer selber über entsprechende Dialoge anmelden können. Da dabei häufig sehr viele Daten angegeben werden müssen, ist es hier sinnvoll, die Eingabeformulare auf mehrere Dialogschritte zu verteilen.

(3) Ebenso sollte bei der Eingabe eine Validierung stattfinden, um so ein Grundmaß an Qualität der Daten sicherstellen zu können. Beispielsweise sollen nur 5-stellige Zahlen als Postleitzahlen und gültige e-Mail-Adressen akzeptiert werden. Auch sollte es dem Nutzer möglich sein, fehlerhafte Eingaben mit keinem oder minimalem Aufwand zu korrigieren (siehe Grundsätze der Dialoggestaltung [ISO95]).

(4) Um sich bei einer Vielzahl von Seiten zu Recht zu finden, bieten Web-Anwendungen normalerweise ein Navigationsmenü an. Dies sollte dementsprechend auch auf jeder Seite sichtbar sein.

(5) Da Web-Anwendungen häufig zur Verwaltung von großen Datenmengen in Datenbanksystemen oder als Online-Shops dienen, werden meist Suchmasken benötigt. Die Ergebnisse werden dann in Tabellen angezeigt. Damit die Tabellen nicht zu groß und unübersichtlich werden, wird meist nur ein Teil der Ergebnismenge angezeigt. Durch „blättern“ innerhalb der Tabelle sind dann auch die restlichen Ergebnisse aufrufbar.

(6) Viele Web-Anwendungen, wie beispielsweise Online-Shops, sind meist nicht auf einen nationalen Markt beschränkt, vielmehr sollen sie über Länder- und Sprachgrenzen hinweg eingesetzt werden. Dazu ist es nötig, dass die Anwendung mehrere Sprachen unterstützt, zwischen denen der Nutzer wählen kann.

5.2 Analyse

Aus Anforderung (1) geht hervor, dass es mindestens zwei Arten von Nutzern geben soll. Diese werden im Folgenden als *Nutzer* und *Administrator* bezeichnet und entsprechend den Anforderungen, Anwendungsfällen zugeordnet (siehe Abbildung 5.1).

Nach Anforderung (1) existieren für beide Akteure die Anwendungsfälle „Login“ und „Logout“. Entsprechend Anforderung (2) kann man sich als Nutzer bei der Anwendung neu anmelden. Ist der Anmeldevorgang erfolgreich, beinhaltet dies den Login des Nutzers. Ein weiterer Anwendungsfall ist die Verwaltung der Nutzer durch den Administrator. Dieser Anwendungsfall soll dazu dienen, Anforderung (5) umzusetzen. Für die Anforderung (6) wird noch der Anwendungsfall „Sprache wechseln“ benötigt.

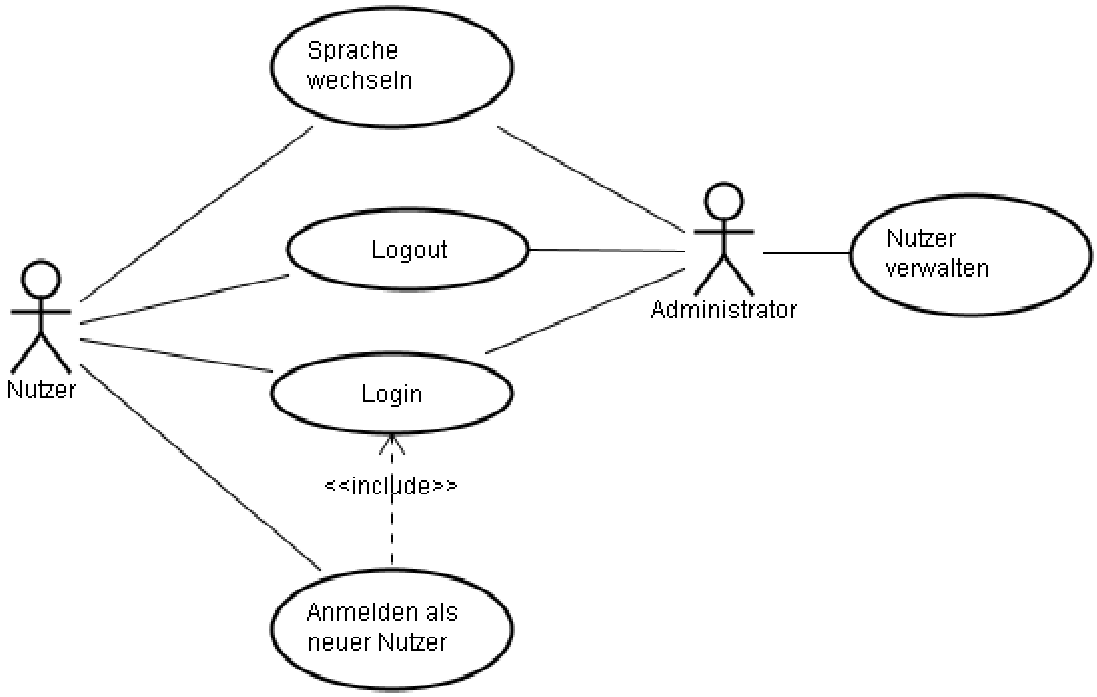

Abbildung 5.1: Anwendungsfälle

5.3 Entwurf

Wie beschrieben liegt das Hauptaugenmerk bei den zu entwickelnden Prototypen auf der Oberfläche. Daher kann hier auf den Entwurf eines komplexen Datenmodells bzw. eines Datenbankschemas verzichtet werden.

Einzig zur Realisierung des Login- und Anmeldevorgangs ist ein einfaches Modell zur Speicherung der Nutzer notwenig (siehe Abbildung 5.2). Dabei dienen die Instanzen der Klasse `User` zur Repräsentation der einzelnen Nutzer mit ihren Daten (Login, Passwort usw.). Diese werden in einer `java.util.Map`-Datenstruktur gespeichert. Als Implementierung des `Map`-Interfaces wird eine `java.util.Hashtable` verwendet, da diese synchronisiert ist. Dies ist insbesondere bei Web-Anwendungen wichtig, da hier im Normalfall mehrere Threads aktiv sind, welche parallel auf das Datenmodell zugreifen können. Als Hülle kommt die Klasse `UserDB` zum Einsatz, welche eine statische Referenz und entsprechende Zugriffsfunktionen auf die Datenstruktur bietet.

Die Speicherung der Nutzer in einer Datenstruktur, welche statisch referenziert wird, ermöglicht es, dass alle Threads innerhalb des Servlet-Containers darauf zugreifen können. Diese Daten stehen so lange zur Verfügung bis der Servlet-Container neu gestartet wird. Dies reicht für die prototypische Implementierung aus und erspart eine Anbindung (beispielsweise an ein Datenbanksystem).

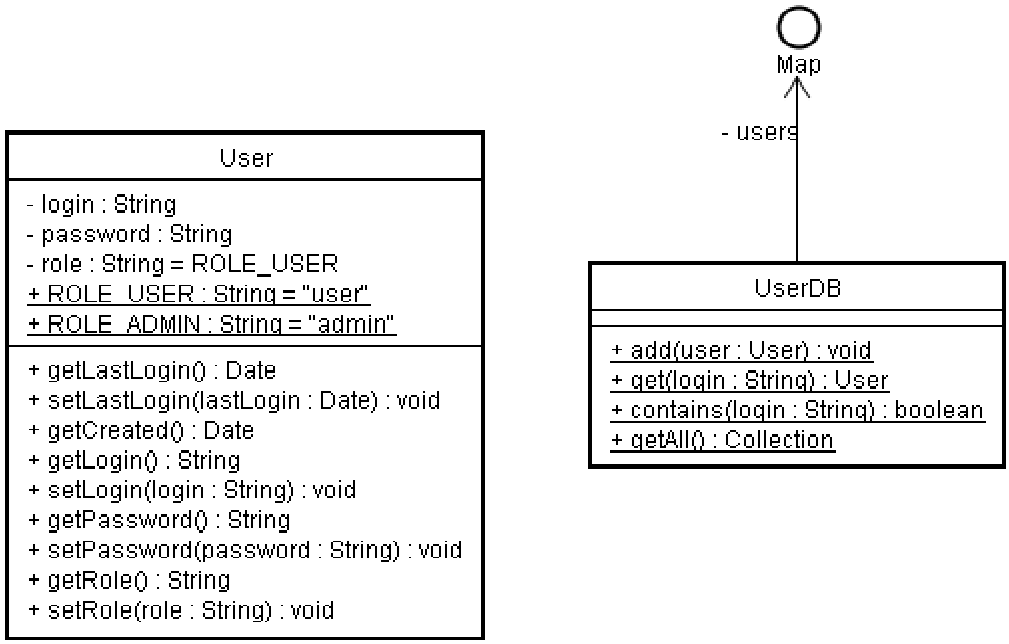

Abbildung 5.2: Datenmodell zur Speicherung von Nutzern

5.4 Implementierung

In diesem Abschnitt wird die Implementierung mit Hilfe der einzelnen Frameworks beschrieben. Dabei wird insbesondere auf Besonderheiten bei der Umsetzung der zuvor definierten Anforderungen eingegangen.

5.4.1 Struts

Für die Implementierung des Prototypen mit Struts wurde die Version 1.2.7 des Frameworks verwendet. Als Servlet-Container diente Tomcat [Ap05a] in der Version 5.0 und die JVM in Version 1.5.0_01-b08. Eine genauere Darstellung von Aufbau und Funktionsweise des Frameworks findet sich in Abschnitt 4.1.

5.4.1.1 Anwendungslogik

Zur Realisierung der einzelnen Anwendungsfälle wurde auf die vom Framework bereitgestellte Klasse `org.apache.struts.action.Action` zurückgegriffen. Als Beispiel wird hier die `LoginAction` vorgestellt:

```
package sample.struts.actions;

import org.apache.struts.action.*;
import javax.servlet.http.*;
import sample.struts.forms.*;
import sample.struts.*;

import java.util.*;

public class LoginAction extends Action
{
   public ActionForward execute(ActionMapping mapping,
```

```
      ActionForm form,
      HttpServletRequest request,
      HttpServletResponse response) throws Exception
   {
      /* find user */
      User user = UserDB.get(((LogonForm) form).getLogin());

      /* set last login date */
      user.setLastLogin(new Date());

      /* mark user in session as logged in */
      request.getSession().setAttribute(
         Constants.SESSION_USER_ATTRIBUTE,
         user);

      return mapping.getInputForward();
   }
}
```

In diesem Beispiel wird zuerst mit Hilfe der zugeordneten ActionForm das `User`-Objekt geladen, das Login-Datum gesetzt und das `User`-Objekt in der Session abgelegt. Die eigentliche Überprüfung des Passwortes wurde schon zuvor mit Hilfe eines Validierers in der zugeordneten `ActionForm` durchgeführt:

```
...

public class LogonForm extends ActionForm
{
   private String login;
   private String password;

   public ActionErrors validate(ActionMapping mapping,
      HttpServletRequest request)
   {
      ActionErrors errors = new ActionErrors();

      /* find user */
      User user = UserDB.get(login);

      if (user != null)
      {
         if (user.getPassword().equals(password))
         {
            // validation correct: no errors, exec LoginAction
            return null;
         }
         else
         {
            errors.add("loginMenu",
               new ActionMessage("error.password.wrong"));
         }
      }
      else
      {
         errors.add("loginMenu",
```

```
            new ActionMessage("error.login.notfound"));
      }
      return errors;
   }

...
}
```

Die Realisierung eines mehrseitigen Dialoges zur Anmeldung eines neuen Nutzers war durch Verwendung einer gemeinsamen `ActionForm` für mehrere Dialogseiten relativ einfach möglich. Die aktuell aufgerufene `Action` kann dadurch nicht nur auf die Formulareingaben der aktuellen Formularseiten zugreifen, sondern auch auf die der vorherigen. Dies ermöglicht es, die Nutzerdaten wirklich erst dann im Datenmodell abzulegen, wenn der komplette Anmeldevorgang erfolgreich abgeschlossen ist.

Die Validierung von Benutzereingaben ist in Struts immer an die entsprechende `ActionForm` gebunden. Sie kann einmal prozedural mit Hilfe der `validate`-Methode, wie bei der Überprüfung des Passwortes in der `LogonForm`, oder deklarativ in einer XML-Datei erfolgen:

```
...
<form-validation>
   <formset>
      <form name="registrationForm">
      ...
         <field property="zip" depends="mask">
            <msg name="mask" key="errors.invalid.zip"/>
            <var>
               <var-name>mask</var-name>
               <var-value>^\d{5}\d*$</var-value>
            </var>
         </field>
      ...
      </form>
   </formset>
</form-validation>
```

Dazu kann jedem Attribute einer ActionForm ein Validator zugeordnet werden. Im obigen Beispiel wird das Feld `zip` der `registrationForm` dem `mask`-Validator zugeordnet. Dazu kann noch eine Fehlermeldung zugeordnet werden, welche über einen Schlüssel aufgelöst werden kann. Die meisten Validatoren können dann noch entsprechend konfiguriert werden. Hier wird z.B. die Maske, mit welcher das Attribut geprüft werden soll, als regulärer Ausdruck festgelegt.

5.4.1.2 Darstellung

Zur Erzeugung des Grundlayouts (inklusive Navigationsmenü) wird die Tag-Library *Tiles* verwendet. Damit ist es relativ einfach möglich, JSP-Seiten aus einzelnen Teilen (den so genannten „Tiles“) zusammenzubauen. Das Grundlayout ist dabei wie folgt definiert:

```
...
```

```
<html:html locale="true">
   <head>
      <title><bean:message key="application.title"/></title>
   </head>
   <body>
      <table class="maintable" align="center">
         <tr>
            <td width="170" valign="top">
               <logic:notEmpty name="user" scope="session">
                  <logic:equal name="user" scope="session"
                        property="role" value="user">
                     <tiles:insert page="mainMenu.jsp"/>
                  </logic:equal>
                  <logic:equal name="user" scope="session"
                        property="role" value="admin">
                     <tiles:insert page="adminMenu.jsp"/>
                  </logic:equal>
               </logic:notEmpty>
               <logic:empty name="user" scope="session">
                  <tiles:insert page="loginMenu.jsp"/>
               </logic:empty>
            </td>
            <td width="100%">
               <table class="contenttable">
                  <tr>
                     <td valign="top">
                        <tiles:insert attribute="content"/>
                     </td>
                  </tr>
               </table>
            </td>
         </tr>
         <tr>
            <td colspan="2" height="20">
               <table class="footertable">
                  <tr>
                     <td>Copyright 2005 Andreas Wende</td>
                     <td align="right">
                        <bean:message key="footer.title"/>
                     </td>
                  </tr>
               </table>
            </td>
         </tr>
      </table>
   </body>
</html:html>
```

Es definiert eine Vorlage für das Aussehen der HTML-Seiten und enthält Platzhalter für weitere JSP-Seiten, beispielsweise für den eigentlichen Inhalt mit Hilfe des Elementes `<tiles:insert attribute="content"/>`. Mit Hilfe von speziellen `logic`-Tags kann auch auf Daten aus der Session zuzugreifen werden. Dies ermöglicht es zu prüfen, ob der Nutzer angemeldet ist und wenn ja, welche Rolle er besitzt. Dementsprechend kann dann das passende Navigationsmenü eingeblendet werden.

Zur Unterstützung von mehrsprachigen Anwendungen bietet Struts den `<bean:message>`-Tag. Dieser ermöglicht es, mit Hilfe eines Schlüssels, Texte aus einer speziellen Textdatei (so genannte *ResourceBundle*) in der aktuellen Sprache einzublenden. Dadurch wird es z.B. möglich, die Anwendung in einer neuen Sprache verfügbar zu machen, indem man die Übersetzungen in den Textdateien ergänzt. Anwendungscode und JSP-Seiten müssen dazu nicht verändert werden.

Die Realisierung einer Administrationsseite zur Suche und Verwaltung von Nutzern, gestaltet sich mit Hilfe einer entsprechenden `Action` und Tags zur Iteration über Datenstrukturen relativ unkompliziert. Tabellen können damit einfach und schnell erzeugt werden.

Als etwas aufwendiger gestaltete es sich zu verhindern, dass nicht angemeldete Nutzer oder Nutzer mit einer falschen Rolle auf bestimmte JSP-Seiten zugreifen können. Zur Realisierung solcher grundlegender Sicherheitsfunktionen bietet Struts leider keinerlei Unterstützung. Das Problem wurde durch die Implementierung eines eigenen Tags gelöst:

```
<app:checkRole role="admin" page="/welcome.do"/>
```

Dieser kann in eine beliebige JSP-Seite eingebunden werden und prüft, ob der Nutzer die (im `role`-Attribute) angegeben Rolle besitzt. Ist dies nicht der Fall, wird, statt die Seite anzuzeigen, auf die im `page`-Attribute angegebene Seite weitergeleitet.

5.4.1.3 Konfiguration

Um die Anwendungslogik mit den entsprechenden JSP-Seiten zu verbinden, dient bei Struts die `struts-config.xml`. Im folgenden Ausschnitt ist der Teil dargestellt, welcher für den mehrstufigen Registrierungsdialog notwendig ist:

```
...
<struts-config>
   <form-beans>
      <form-bean name="registrationForm"
         type="sample.struts.forms.RegistrationForm"/>
      ...
   </form-beans>
   <action-mappings>
      <action path="/signOn" forward="/pages/registration1.jsp" />
      <action path="/registration1" name="registrationForm"
         type="sample.struts.actions.Registration1Action"
         scope="session" validate="true"
         input="/pages/registration1.jsp">
         <forward name="next" path="/pages/registration2.jsp"/>
         <forward name="cancel" path="/pages/welcome.jsp"/>
      </action>
      <action path="/registration2" name="registrationForm"
         type="sample.struts.actions.Registration2Action"
         scope="session" validate="true"
         input="/pages/registration2.jsp">
         <forward name="complete" path="/pages/welcome.jsp"/>
```

```
         <forward name="back" path="/pages/registration1.jsp"/>
      </action>
      ...
   </action-mappings>
   ...
</struts-config>
```

In Struts wird generell zu jeder Seite eine `Action` benötigt, welche einem bestimmten Pfad im Request zugeordnet ist. Im einfachsten Fall (im Beispiel `path="/signOn"`) ist dies eine vom Framework bereit gestellte `ForwardAction`, welche die angegebene JSP-Seite darstellt.

In den meisten Fällen wird einem Request-Pfad jedoch eine selbst programmierte `Action`-Klasse zugeordnet. Diese kann dann die entsprechende Anwendungslogik ausführen und dynamisch festlegen, welche JSP-Seite angezeigt werden soll. Diese Seiten werden in Form von `forward`-Tags der `Action` zugeordnet. Daneben kann einer Action noch eine ActionForm (mit Hilfe des `name`-Attributes) zugeordnet werden.

5.4.2 JavaServer Faces / myFaces

Für die Implementierung des Prototypen mit Hilfe von JavaServer Faces wurde, statt der Referenzimplementierung [Sun05d] von Sun, die myFaces-Implementierung [Ap05g] von Apache in der Version 1.0.9 verwendet. Diese bietet (zusätzlich zur Referenzimplementierung) weitere Komponenten, wie Bäume, sortierbare Datenlisten, Navigationsmenüs und usw. und eine Unterstützung für die aus Struts bekannte „Tiles“-Library. Als Servlet-Container diente wieder Tomcat [Ap05a] in der Version 5.0 und die JVM in Version 1.5.0_01-b08. Eine genauere Darstellung von Aufbau und Funktionsweise des Frameworks findet sich in Abschnitt 4.5.

5.4.2.1 Anwendungslogik

Für den Zugriff auf Benutzereingaben (z.B. aus Formularen) dienen in JavaServer Faces so genannte *Managed Beans*. Diese besitzen eine ähnliche Aufgabe wie in Struts die `ActionForms`, sind aber flexibler einsetzbar, da sie nicht fest an eine bestimmte `Action` gebunden sind. Im Folgenden ist der Grundaufbau der Managed Bean für den Login-Dialog dargestellt:

```
...
public class LogonBean
{
   private String login;
   private String password;
   private User user;

   /* logout current user */
   public void doLogout(ActionEvent event)
   {
      login = null;
      password = null;
      user = null;
```

```
   }

   /* login user */
   public void doLogin(ActionEvent event)
   {
      User user = UserDB.get(login);

      if (user == null)
      {
         /* set error message: login not found */
         ...
      }
      else
      {
         if (user.getPassword() == null ||
            user.getPassword().equals(password))
         {
            this.user = user;
            user.setLastLogin(new Date());
         }
         else
         {
            /* set error message: password wrong */
            ...
         }
      }
   }

   /* getter and setter for private attributes */
   ...
}
```

Diese enthält, neben den Getter- und Setter-Methoden für die Attribute, welche an die entsprechenden `InputText`-Komponenten gebunden werden, auch die ActionListener-Methoden, welche beim Betätigen des Login- und Logout-Buttons ausgeführt werden.

5.4.2.2 Darstellung

Zur Erzeugung der Grundstruktur der Seiten wird (wie bei Struts) die Tag-Library *Tiles* verwendet. Dies ermöglicht es relativ einfach das Grundlayout zentral mit Hilfe von HTML zu definieren und dynamisch Unterseiten einzubinden.

In JSP-Seiten lassen sich dann innerhalb von `<f:view>`- oder `<f:subview>`-Tags Komponenten einbinden. Für den Login-Bereich sieht dies wie folgt aus:

```
<f:subview id="loginMenu">
   <h:form id="loginForm" rendered="#{empty logonBean.user}">
      <h:message for="loginForm" styleClass="error"/>
      <h:inputText id="login" value="#{logonBean.login}"
         style="width:100%" maxlength="60"/>
      <f:verbatim><br></f:verbatim>
      <h:inputSecret id="password" value="#{logonBean.password}"
         style="width:100%" maxlength="60" redisplay="true"/>
      <f:verbatim><br></f:verbatim>
```

```
      <h:commandButton id="doLogin"
         actionListener="#{logonBean.doLogin}" action="welcome"
         value="#{msg['menu.login']}"/>
      <f:verbatim> </f:verbatim>
      <h:commandButton id="doSignOn" value="#{msg['menu.signOn']}"
         action="signOn"/>
   </h:form>
   <h:form id="logoutForm" rendered="#{not empty logonBean.user}">
      <h:outputText value="#{msg['menu.loginAt']}:"/>
      <f:verbatim><br></f:verbatim>
      <h:outputFormat value="{0, date, medium}">
         <f:param value="#{logonBean.user.lastLogin}"/>
      </h:outputFormat>
      <f:verbatim><br></f:verbatim>
      <h:outputFormat value="{0, time, medium}">
         <f:param value="#{logonBean.user.lastLogin}"/>
      </h:outputFormat>
      <f:verbatim><br></f:verbatim>
      <f:verbatim><br></f:verbatim>
      <h:commandButton id="doLogout"
         actionListener="#{logonBean.doLogout}"
         value="#{msg['menu.logout']}"/>
   </h:form>
</f:subview>
```

Für einzelne Komponenten kann mit Hilfe des `rendered`-Attributes festgelegt werden, ob es dargestellt werden soll. Dies kann dazu verwendet werden, um (abhängig von Werten in Managed Beans) verschiedene Menüoptionen anzuzeigen: Ist kein Nutzer angemeldet (`rendered="#{empty logonBean.user}"`) wird hier z.B. ein Formular für den Login angeboten, andernfalls (`rendered="#{not empty logonBean.user}"`) werden Informationen über den aktuellen Nutzer angezeigt und eine Option zum Logout.

Zur Unterstützung von mehrsprachigen Anwendungen bietet JavaServer Faces einen ähnlichen Mechanismus wie Struts, welcher es ermöglicht, auf *ResourceBundle*-Datei mit Übersetzungen in die einzelnen Sprachen zuzugreifen. Der Zugriff erfolgt mit Hilfe einer speziellen Syntax (`#{msg['menu.login']}`) und kann bei beliebigen Komponenten verwendet werden.

Neben einfachen UI-Komponenten, wie beispielsweise Texteingabefeldern und Buttons, stellt die myFaces-Implementierung auch komplexere Komponenten bereit. So wird z.B. für das Navigationsmenü eine `panelNavigation`-Komponente verwendet:

```
<x:panelNavigation
   styleClass="navigation"
   separatorClass="navseparator"
   itemClass="navitem"
   activeItemClass="navitem_active"
   openItemClass="navitem_open">
   <x:commandNavigation id="mainPage"
      value="#{msg['menu.item.home']}" action="welcome"/>
   <x:commandNavigation id="changePasswordPage"
```

```
      value="#{msg['menu.item.changePassword']}"
      action="changePassword" rendered="#{not empty
      logonBean.user}"/>
   <x:commandNavigation id="adminPage"
      value="#{msg['menu.item.admin']}"
      rendered="#{logonBean.user.role == 'admin'}">
      <x:commandNavigation id="userPage"
         value="#{msg['menu.item.user']}" action="users" />
   </x:commandNavigation>
</x:panelNavigation>
```

Dadurch lassen sich ohne großen Aufwand komplexe Navigationsmenüs mit mehreren Ebenen erzeugen, wie Abbildung 5.3 zeigt.

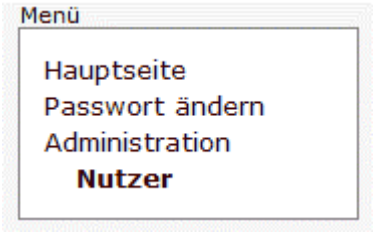

Abbildung 5.3: Darstellung einer panelNavigation-Komponente in JavaServer Faces

Ebenso existieren Komponenten, welche die Darstellung von großen Datenmengen in Tabellen vereinfachen. Abbildung 5.4 zeigt das Ergebnis der Verwendung von der `dataTable`- und `dataScroller`-Komponente für die Nutzerverwaltung durch einen Administrator:

```
<x:dataTable
   id="users"
   var="user"
   value="#{usersBean.users}"
   rows="#{usersBean.rowsPerPage}"
   styleClass="list"
   sortColumn="#{usersBean.column}"
   sortAscending="#{usersBean.asc}">
   <h:column>
      <f:facet name="header">
         <x:commandSortHeader columnName="login" arrow="true">
            <h:outputText value="#{msg['page.users.login']}"/>
         </x:commandSortHeader>
      </f:facet>
      <h:outputText value="#{user.login}" />
   </h:column>
   ...
</x:dataTable>
...
<x:dataScroller
   for="users"
   fastStep="10"
   paginator="true"
   paginatorMaxPages="10"
   paginatorActiveColumnStyle="font-weight:bold;">
```

```
   <f:facet name="first" >
      <h:graphicImage url="../resources/arrow-first.gif"
         border="1" />
   </f:facet>
   ...
</x:dataScroller>
```

Dabei kann die `dataTable`-Komponente über das `value`-Attribut an eine `Collection` einer Managed Bean gebunden werden, welche die eigentlichen Daten enthält. Mit Hilfe der `<h:column>`-Tags können dann die anzuzeigenden Daten für jede einzelne Spalte definiert werden. Im Tabellenkopf werden Links bereitgestellt, welche das Sortieren nach der jeweiligen Spalte ermöglichen. Die `dataScroller`-Komponente wird über das `for`-Attribut an die `dataTable`-Komponente gebunden und stellt Buttons zum Blättern innerhalb der Daten bereit.

Die Verwendung von Komponenten stellt im Bezug zu Struts einen großen Fortschritt dar. Zwar ließ sich auch dort relativ einfach eine Tabelle zur Nutzerverwaltung erzeugen. Zusatzfunktionen wie das Sortieren nach Spalten und Blättern innerhalb der Tabelle wären jedoch nur mit einem deutlichen Mehraufwand realisierbar gewesen. Bei den hier verwendeten Komponenten können sie ohne zusätzliche Implementierung genutzt werden.

Abbildung 5.4: Darstellung einer Tabelle zur Nutzerverwaltung mit dataTable- und dataScroller-Komponente in JavaServer Faces

5.4.2.3 Dialogsteuerung

Als einziges der in dieser Arbeit vorgestellten Frameworks bieten JavaServer Faces die Möglichkeit, den Dialogfluss als endlichen Automaten zu modellieren.

Abbildung 5.5 zeigt den endlichen Automaten für die implementierte Anwendung. Dabei stellt jeder Knoten eine JSP-Seite dar. Zur Vereinfachung der Darstellung wurde zusätzlich ein spezieller Knoten mit der Bezeichnung „*“ eingefügt. Er dient als Ursprungsknoten für alle Übergänge mit beliebigem Ursprung. Dies sind typischerweise Seitenwechsel, welche im Navigationsmenü global verfügbar sind.

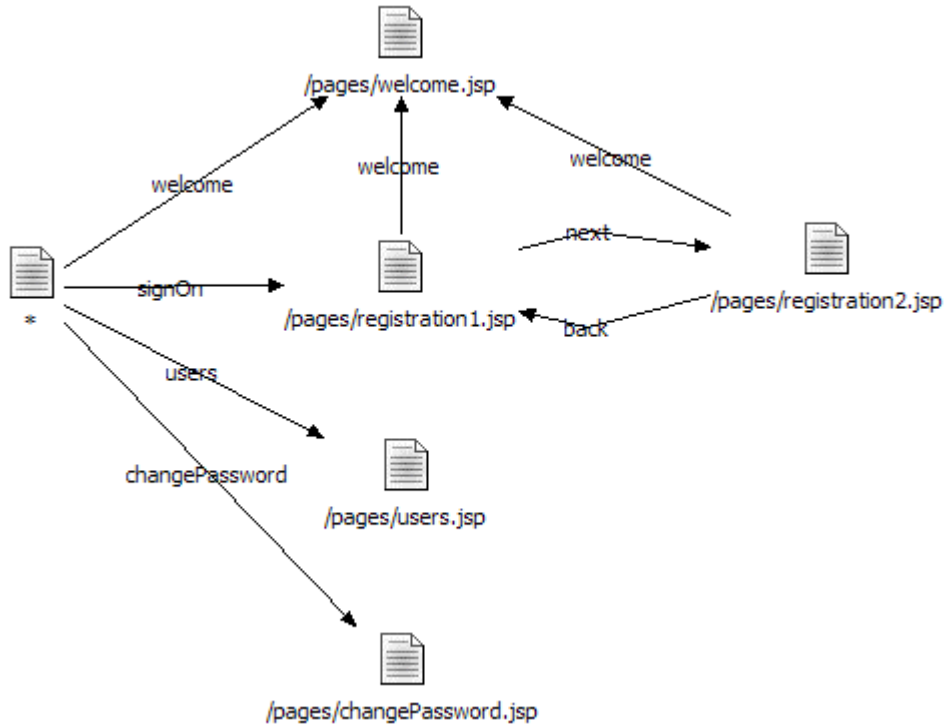

Abbildung 5.5: Dialogfluss der implementierten Anwendung als endlicher Automat

5.4.2.4 Konfiguration

Ähnlich wie bei Struts erfolgt auch bei JavaServer Faces die Konfiguration über eine zentrale XML-Datei. Hier können beispielsweise eigene UI-Komponenten registriert oder ein anderes *RenderKit* verwendet werden. Ein RenderKit definiert dabei, wie die UI-Komponenten in einer bestimmten Markup-Sprache darzustellen sind. Ein RenderKit zur Erzeugung von Standard-HTML ist in der Referenzimplementierung von Sun vorhanden. Die myFaces-Implementierung enthält zusätzlich ein RenderKit für WML.

Hauptsächlich dient die Konfigurationsdatei jedoch zur Definition der Seitenübergänge für die Dialogsteuerung (wie in Abbildung 5.5 dargestellt) und zur Konfiguration der Managed Beans:

```
<?xml version="1.0"?>
<faces-config>
   ...
   <managed-bean>
      <managed-bean-name>logonBean</managed-bean-name>
      <managed-bean-class>
         sample.myFaces.beans.LogonBean
      </managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
```

```
   ...
</faces-config>
```

Dafür wird für jede Managed Bean ein eindeutiger Name vergeben. Dieser wird unter anderem dazu benötigt, innerhalb von JSP-Seiten auf das Managed Bean zuzugreifen. Daneben muss noch die implementierende Klasse und der Scope angegeben werden. Für den Scope existieren drei mögliche Werte:

- *Request*: Eine Instanz der Managed Bean existiert nur während des aktuellen Requests.
- *Session*: Eine Instanz der Managed Bean bleibt für die gesamte Benutzersitzung erhalten. Dies wird in der Anwendung, z.B. für die Speicherung des Login-Status und mehrseitiger Dialoge, verwendet.
- *Application*: Eine Instanz der Managed Bean ist für alle Benutzersitzungen sichtbar.

5.4.3 Echo

Für die Implementierung des Prototypen mit Hilfe von Echo wurde die Version 2.0.rc1 des Frameworks verwendet. Als Servlet-Container diente wieder Tomcat [Ap05a] in der Version 5.0 und die JVM in Version 1.5.0_01-b08. Eine genauere Darstellung von Aufbau und Funktionsweise des Frameworks findet sich in Abschnitt 4.8.

5.4.3.1 Anwendungslogik

Die Anwendungslogik wird, wie bei allen ereignisgesteuerten Frameworks, mit Hilfe von EventListener-Methoden implementiert, welche für bestimmte Ereignisse an Oberflächenkomponenten registriert werden können. Das folgende Beispiel zeigt deren Verwendung zur Realisierung der Login-Funktion der Anwendung:

```
...
public class LoginWindow extends WindowPane implements
ActionListener
{
   private TextField loginField;
   private TextField passwordField;
   ...

   /* Constructor */
   public LoginWindow()
   {
      super();
      /* building window content */
      ...

      Button loginButton =
         new Button(Messages.getString("button.login"));
      loginButton.setStyleName("Default");
      loginButton.setActionCommand("login");
      loginButton.addActionListener(this);

      Button signOnButton =
```

```
         new Button(Messages.getString("button.signOn"));
      signOnButton.setStyleName("Default");
      signOnButton.setActionCommand("signOn");
      signOnButton.addActionListener(this);
      ...
   }

   public void actionPerformed(ActionEvent e)
   {
      ...
      if (e.getActionCommand().equals("login"))
      {
         User user = UserDB.get(loginField.getText());
         if (user == null)
         {
            /* set error message: login not found */
            ...
         }
         else
         {
            if (user.getPassword() == null ||
user.getPassword().equals(passwordField.getText()))
            {
               Echo2SampleApp app = (Echo2SampleApp)
ApplicationInstance.getActive();
               app.setCurrentUser(user);
               app.setContent(new WelcomePane());
            }
            else
            {
               /* set error message: password wrong */
               ...
            }
         }
      }
      else if (e.getActionCommand().equals("signOn"))
      {
         this.getParent().add(new RegistrationWindow());
         this.getParent().remove(this);
      }
   }
}
```

In dem hier dargestellten Dialogfenster existieren neben entsprechenden Eingabefeldern zwei Buttons. Der erste dient zum Login, der zweite zum neu Anmelden. Bei beiden Buttons wird dazu die aktuelle Klasse als ActionListener registriert. Dafür muss diese das Interface `ActionListener` mit der Methode `actionPerformed()` implementieren. Zur Unterscheidung zwischen den beiden Buttons wird noch ein entsprechendes ActionCommand gesetzt.

5.4.3.2 Darstellung

Die Darstellung wird in Echo vollständig aus den Oberflächenkomponenten generiert. Die Anordnung und die Eigenschaften der verwendeten Komponenten

werden dabei vollständig im Code definiert. Folgender Auszug zeigt z.B. die Verwendung von Eingabefeldern für Login und Passwort:

```
Grid grid = new Grid(2);
grid.add(new Label(Messages.getString("app.item.login")+":"));
loginField = new TextField();
loginField.addActionListener(this);
loginField.setActionCommand("login");
grid.add(loginField);
grid.add(new Label(Messages.getString("app.item.password")+":"));
passwordField = new PasswordField();
passwordField.addActionListener(this);
passwordField.setActionCommand("login");
grid.add(passwordField);
```

Es existieren verschiedene Elemente zur Gruppierung innerhalb des Fensters, wie z.B. das verwendete `Grid`. Einzelne Elemente, wie `Label` und `TextField`, werden dann zu diesem `Grid` hinzugefügt.

Zur Umsetzung der Mehrsprachigkeit wird mit Hilfe der `Messages`-Klasse ein Schlüssel in die aktuell eingestellte Sprache übersetzt. Die Zuordnung der Schlüssel zu den Übersetzungen in den einzelnen Sprachen erfolgt in *ResourceBundle*-Dateien.

5.4.3.3 Konfiguration

Anders als bei den beiden anderen verwendeten Frameworks ist bei Echo keine Konfiguration notwendig. Dies liegt daran, dass es bei Echo grundsätzlich keine Templates gibt, welche in Konfigurationsdateien mit dem Anwendungscode verbunden werden müssen.

5.5 Vergleich

Die drei verwendeten Frameworks zeigen, wie unterschiedlich das MVC-Muster umgesetzt werden kann. Auf der einen Seite des Spektrums steht Struts als ein Framework, welches sich stark am Request-Response-Zyklus von Web-Anwendungen orientiert und eine strikte „Model 2“-Umsetzung des MVC-Musters verwendet. Auf der anderen Seite des Spektrums steht das Echo-Framework. Es verwendet eine Umsetzung des MVC-Musters, welches sich stark an Swing [Sun98] anlehnt. Dazwischen findet man JavaServer Faces, welches Teile von beiden Ansätzen verwendet.

5.5.1 Konfiguration, Anwendungslogik und Dialogsteuerung

Vergleicht man den Implementierungsaufwand, fällt besonders der unterschiedliche Konfigurationsaufwand auf: Bei Struts fällt dieser am größten aus. Hier muss prinzipiell zu jedem Pfad eine Zuordnung zu einer Aktion definiert werden, eventuell noch mit entsprechenden Weiterleitungen abhängig vom Ergebnis der Aktion. Zusätzlich müssen noch entsprechende ActionForms definiert werden. Insbesondere bei großen Web-Anwendungen mit vielen Seiten kann dies zu einem starken Anwachsen der Konfigurationsdatei führen. Bei JavaServer Faces fällt die Konfiguration etwas moderater aus. Hier liegt der

Hauptaufwand in der Definition der Managed Beans und der Navigationsregeln. Bei Echo ist hingegen grundsätzlich keine Konfiguration notwendig.

Der Aufwand zur Implementierung der Anwendungslogik ist bei allen Frameworks ungefähr gleich. Hier besteht nur ein Unterschied in welcher Form sie umgesetzt wird. Entweder als Aktionen (wie bei Struts) oder in Form von Ereignisbehandlungsroutinen (wie bei JavaServer Faces und Echo).

Ein Vorteil von JavaServer Faces gegenüber den anderen beiden Frameworks ist, dass sie eine zentrale Dialogsteuerung besitzt. Dies ermöglicht es einerseits Änderungen am Dialogfluss zentral mit geringerem Aufwand vorzunehmen und andererseits die Abläufe und die in der Anwendung umgesetzten Geschäftsprozesse einfacher zu dokumentieren.

5.5.2 Darstellung

Große Unterschiede gibt es hingegen bei der Entwicklung der Darstellung. Bei Struts muss prinzipiell viel HTML-Code geschrieben werden. Durch die Verwendung von speziellen Tag-Libraries wird dabei die Verbindung zur Anwendungslogik hergestellt. Dieser Ansatz ist daher sehr darstellungsorientiert. Insbesondere ist hier bei der Entwicklung eine gewisse Erfahrung in HTML unabdingbar. Auf der anderen Seite bietet dies aber die maximal (in HTML) mögliche Flexibilität bei der Gestaltung der Benutzeroberfläche.

Bei JavaServer Faces steht die Verwendung von HTML hingegen nicht mehr im Mittelpunkt. Hier werden hauptsächlich UI-Komponenten verwendet, welche ihre Darstellung in HTML-Code selber mit Hilfe von Render-Klassen erzeugen. Selbstgeschriebener HTML-Code dient bei JavaServer Faces grundsätzlich nur zur Definition des Grundlayouts der Seite, in welches dann die einzelnen Komponenten eingebunden werden.

Die Verwendung von UI-Komponenten allein bringt jedoch im Gegensatz zu Struts keinen großen Gewinn in Hinsicht auf den Implementierungsaufwand. Das eigentliche Potenzial liegt hier vielmehr darin, wiederkehrende HTML-Repräsentationen in eigene Komponenten zu kapseln bzw. dafür Komponenten von Drittanbietern zu verwenden (z.B. für Navigationsmenüs und komplexe Tabellen wie in 5.4.2.2 geschildert).

Jedoch geht durch die Verwendung von Komponenten auch ein Teil der Flexibilität bei der Gestaltung verloren. Dies hängt insbesondere stark von den Konfigurationsmöglichkeiten jeder einzelnen Komponente ab, was somit ein wichtiges Qualitätsmerkmal von UI-Komponenten darstellt. Häufig lassen sie sich (z.B. durch die Verwendung von Stylesheets) anpassen.

Große Potenziale im Hinblick auf die Verkürzung der Entwicklungszeit bieten graphische Entwicklungsumgebungen, wie sie beispielsweise für Swing existieren. Da die JavaServer-Faces-Spezifikation jedoch noch relativ jung ist und sich hier noch keine große Entwicklergemeinde entwickeln konnte, stehen zurzeit noch keine frei verfügbaren Entwicklungsumgebungen bereit, welche zu diesem Zweck untersucht werden konnten.

Bei der Entwicklung einer Anwendung mit Hilfe des Echo-Frameworks kommt man hingegen vollständig ohne HTML-Kenntnisse aus. Hier wird die Darstellung (ähnlich wie bei Swing) vollständig im Code definiert, somit reichen Java-Kenntnisse grundsätzlich aus. Als Nachteil erweist sich jedoch, dass es zurzeit noch keine graphische Entwicklungsumgebung gibt und die Entwicklung im Text-Editor sehr aufwendig ist. Sollte sich dies in Zukunft ändern, könnten mit Echo ähnlich schnell Anwendungen entwickelt werden wie mit Swing.

Kapitel 6

Bewertung

In dieser Arbeit wurden verschiedene Unterscheidungsmerkmale von Web-Frameworks untersucht und geklärt, inwieweit sie sich zu einer Klassifikation eignen. Dabei konnten fünf Klassifikationskriterien mit ihren entsprechenden Ausprägungen identifiziert werden. Bis auf eines dieser Kriterien hängen alle anderen voneinander ab. Aus diesen Abhängigkeiten konnte ein Schema abgeleitet werden. Durch die Einordnung in dieses Schema können auch Hinweise darauf gewonnen werden, ob es sich dabei um ein Framework handelt, welches stark vom Charakter einer Web-Anwendung abstrahiert oder im Gegensatz dazu eine größere Flexibilität und Anpassungsfähigkeit ermöglicht.

Um die Anwendbarkeit des entwickelten Ansatzes zu prüfen, wurde im Folgenden eine Auswahl von Web-Frameworks klassifiziert. Dazu wurden insbesondere die am weitesten verbreiteten verfügbaren Frameworks einbezogen, daneben zu Vergleichszwecken auch einige weniger bekannte Frameworks mit interessanten Konzepten. Dabei konnte jedes dieser Frameworks sinnvoll in das Klassifikationsschema eingeordnet und drei Grundtypen von Frameworks identifiziert werden. Frameworks gleichen Typs weisen dabei entweder eine gleiche oder sehr ähnliche Klassifikation auf.

Um aus der gefundenen Klassifikation auch Rückschlüsse auf die Praxistauglichkeit der einzelnen Ansätze ziehen zu können und Auswahlkriterien abzuleiten, wurden anschließend mit jeweils einem Vertreter jedes Typs eine kleine Web-Anwendung prototypisch implementiert. Dabei konnte festgestellt werden, dass sich die Ansätze deutlich unterscheiden. Dies betrifft insbesondere Implementierungsaufwand, Flexibilität und Abstraktion.

6.1 Fazit

Durch den Vergleich der Implementierungen in Kapitel 5 lassen sich auch Auswahlrichtlinien für den konkreten Projekteinsatz ableiten. Die verwendeten Frameworks stehen dabei stellvertretend für den gesamten Typ, da hier das grundlegende Konzept und weniger die technischen Details betrachtet werden:

Bei Projekten, bei denen eine große Flexibilität in Hinsicht auf Anpassbarkeit gefragt ist, bietet sich ein aktionsgesteuerter Ansatz wie Struts an. Solche Frameworks orientieren sich stark am Request-Response-Zyklus von Web-Anwendungen und stellen daher meist nur ein Grundgerüst bereit, wie

verschiedene Dienste und Technologien zu einer Web-Anwendung verbunden werden können. Aktionsgesteuerte Frameworks verfolgen einen White-Box-Ansatz, welcher die Anpassung durch Programmierung viel weiter in den Vordergrund stellt als dies bei ereignisgesteuerten Frameworks mit einem Black-Box-Ansatz der Fall ist. Dort steht eher die Anpassung durch Konfiguration von vorhandenen Komponenten im Vordergrund.

Auch bei Projekten, bei denen die graphische Gestaltung der Nutzeroberfläche eine große Rolle spielt, bietet sich ein aktionsgesteuertes Framework an, da hier der zur Darstellung verwendete HTML-Code über Templates direkt veränderbar ist. Dies bietet eine weit größere Flexibilität als bei ereignisgesteuerten Frameworks, welche teilweise oder vollständig UI-Komponenten zur Erzeugung der Darstellung nutzen. Hier ist der Entwickler allein auf die Anpassungsfähigkeit dieser Komponenten beschränkt und kann darüber hinaus kaum Einfluss auf die Darstellung nehmen.

Auch in kritischen Projektphasen, in denen unter Zeitdruck schnelle Erfolge erzielt werden müssen, ist es ratsam, ein klassisches aktionsgesteuertes Framework zu verwenden. Die Einarbeitung in ein ereignisgesteuertes Framework erfordert meist viel Zeit und Geduld, was sich erst in späteren Projektphasen auszahlt. Erst dann kann durch die Wiederverwendung von bestehenden Komponenten eine hohe Entwicklungsgeschwindigkeit und Produktivität erreicht werden.

Ein wichtiges Entscheidungskriterium kann auch die Verbreitung und die Verfügbarkeit von Literatur und Dokumentation sein. Dies lässt sich nicht an bestimmten Klassifikationskriterien ausmachen, soll aber trotzdem hier nicht unerwähnt bleiben. Hier hat ein - über viele Jahre gewachsenes - Framework wie Struts klare Vorteile, da es durch die große Nutzergemeinde inzwischen nahezu erschöpfend dokumentiert ist.

Für Web-Anwendungen, welche eine hohes Maß an Interaktionen benötigen, bietet sich meist ein ereignisgesteuerter Ansatz an, da sich hier die Manipulation von Oberflächenkomponenten durch Benutzereingaben meist sehr einfach über Ereignisbehandlungsroutinen umsetzen lässt. Insbesondere muss nicht für jede Darstellungsänderung eine eigene Aktion (wie bei aktionsgesteuerten Frameworks) geschrieben werden, welche zuerst das Model aktualisiert und anschließend daraus wieder die Darstellung generiert.

Auch wenn im Projektteam viel Erfahrung bei der Entwicklung von Desktop-Anwendungen vorhanden ist, bietet sich meist ein ereignisgesteuerter Ansatz an, da dieser dem Entwicklungsprinzip von Desktop-Anwendungen sehr nahe kommt, bzw. ihn nachbildet. Dies kann auch bei einer Portierung von Desktop- zu Web-Anwendungen vorteilhaft sein.

Bei umfangreichen Projekten bietet sich ebenfalls häufig ein ereignisgesteuerter Ansatz an, da hier durch die Verwendung von Oberflächenkomponenten eine Softwarewiederverwendung möglich ist. Dies kann durch die Entwicklung eigener oder von Dritten zur Verfügung gestellter Komponenten realisiert werden. Durch die Kapselung von Funktionalitäten und Steuerelementen in Komponenten wird auch eine Reduzierung der Komplexität der Anwendung erreicht, was der Wartbarkeit und Pflege der Anwendung zu Gute kommt. Bei kleineren Projekten

kann hingegen die Entwicklung von eigenen Komponenten in Relation zu einer möglichen Wiederverwendung zu aufwendig sein.

6.2 Diskussion

In dieser Arbeit konnten verschiedene Auswahlkriterien für Frameworktypen angegeben werden. Dies kann jedoch in einem konkreten Projekt nur als eine Vorauswahl dienen, da sich auch Frameworks desselben Typs in vielen technischen Details stark unterscheiden können. Diese Arbeit hat sich aus Gründen des Umfangs bewusst auf die konzeptionellen Unterschiede und weniger auf technische Details konzentriert. Technische Details können bei Projekten jedoch auch ein entscheidender Faktor sein und zum Erfolg oder Misserfolg des gesamten Projekts beitragen.

Auch die Auswahl der klassifizierten Frameworks muss aus Gründen des Umfangs begrenzt bleiben. Neben den hier vorgestellten häufig verwendeten Frameworks, gibt es noch eine Reihe weiter weniger bekannter Frameworks. Auch viele kommerzielle Frameworks konnten aus Budgetgründen nicht untersucht werden. Trotzdem sollte die getroffene Auswahl die wichtigsten Frameworktypen repräsentieren.

Da sich der Bereich der Web-Anwendungen schnell weiterentwickelt, kann diese Arbeit nur die aktuelle Situation darstellen. Einige der vorgestellten Frameworks (wie z.B. JavaServer Faces) und Konzepte (wie ereignisgesteuerten Web-Anwendungen insgesamt) sind noch relativ neu und werden daher auch in Zukunft einer raschen Weiterentwicklung unterliegen. Wie sich die Situation in 5 Jahren darstellt, kann daher in dieser Arbeit nicht abgeschätzt werden.

6.3 Ausblick

Aufbauend auf dieser Arbeit sind verschiedene weiterführende Untersuchungen denkbar. Abschließend sollen hier einige Möglichkeiten skizziert werden:

In dieser Arbeit wurde unter anderem das auf der AJAX-Technologie aufbauende Framework Echo vorgestellt. Dort wird versucht das Konzept der ereignisgesteuerten Anwendungssteuerung mit den Interaktionsmöglichkeiten von Desktop-Anwendungen zu verbinden. Ohne dabei zusätzliche Software auf dem Client zu installieren. Dieser Ansatz steckt noch in den Anfängen, lässt aber Potenziale erkennen. Hier könnte z.B. genauer untersucht werden, wie groß der Performanzgewinn ist, wenn bei Zustandsänderungen nicht mehr die komplette Seite neu erzeugt und zum Client übertragen werden muss. Es ist anzunehmen, dass die Last bei diesem Ansatz besser auf Client und Server verteilt wird, weil jetzt der Client zum großen Teil für die Erzeugung und Aktualisierung der Darstellung zuständig ist. Die Anwendungslogik jedoch weiter zentral auf dem Server ausgeführt wird. Dies sollte durch weitere Untersuchungen noch genauer geklärt werden.

Die Verwendung einer zentralen Dialogsteuerung auf Basis von endlichen Automaten bei JavaServer Faces zeigt die Möglichkeiten auf, welche ein solcher Ansatz bietet. In diesem Zusammenhang könnte beispielsweise untersucht

werden, wie sich eine explizite Dialogsteuerung in verschiedene Frameworks integrieren lässt und welche Frameworktypen sich dazu am besten eignen.

Eine zunehmend an Bedeutung gewinnende Art von Web-Anwendungen sind Web-Portale. In diesem Zusammenhang könnte untersucht werden, welche Unterstützung die vorgestellten Frameworks zur Integration in Portale bieten, bzw. ob dies überhaupt möglich ist. Auch könnte untersucht werden, ob die hier entwickelte Klassifikation sinnvoll auf Web-Portale übertragbar ist.

Literatur

[ABB04] Eric Armstrong, Jennifer Ball, Stephanie Bodoff, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock: *The J2EE™ 1.4 Tutorial*. http://java.sun.com/j2ee/1.4/docs/ (16.12.2004)

[Ap05a] Apache Software Foundation: *Jakarta Tomcat*. http://jakarta.apache.org/tomcat/.

[Ap05b] Apache Software Foundation: *Cocoon*. http://cocoon.apache.org/.

[Ap05c] Apache Software Foundation: *Struts*. http://struts.apache.org/.

[Ap05d] Apache Software Foundation: *Velocity*. http://jakara.apache.org/velocity/.

[Ap05e] Apache Software Foundation: *Turbine*. http://jakarta.apache.org/turbine/.

[Ap05f] Apache Software Foundation: *Tapestry*. http://jakarta.apache.org/tapestry/.

[Ap05g] Apache Software Foundation: *MyFaces*. http://myfaces.apache.org/.

[Ba00] Helmut Balzert: *Lehrbuch der Software-Technik I, Software-Entwicklung*. 2. Heidelberg, Berlin: Spektrum Akademischer Verlag, 2000.

[BBE95] Andi Birrer, Walter Bischofberger, Thomas Eggenschwiler: Wiederverwendung durch Frameworktechnik - vom Mythos zur Realität. In: *OBJEKTspektrum* (1995), September/Oktober.

[BD04] Gerd Beneken, Florian Deißenböck: Inside JavaServer Faces. In: *JavaSpektrum* (2004), Nr. 4.

[Bel04] Abhijit Belapurkar: Use continuations to develop complex Web applications. in: *IBM developer works*. http://www-128.ibm.com/developerworks/library/j-contin.html (21.12.2004).

[BG03] Matthias Book, Volker Gruhn: A Dialog Control Framework for Hypertext-Based Applications. *Proceedings of the 3rd International Conference on Quality Software (QSIC 2003)*, 170-177. IEEE Computer Society, 2003.

[ECM05] European Computer Manufacturers Association: *Standard ECMA-262 - ECMAScript Language Specification*. http://www.ecma-international.org/publications/standards/Ecma-262.htm, (24.07.2005).

[Ga01] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2001.

[Ga05] Jesse James Garrett: *Ajax: A New Approach to Web Applications*. http://www.adaptivepath.com/publications/essays/archives/000385.php, (18.02.2005).

[Ha87] D. Harel: Statecharts: A visual formalism for complex systems. In: *Science of Computer Programming* (1987), Nr. 8, S. 231-274.

[Ha04] Sven Haiges: JavaServer Faces Teil 2: Event Handling und Navigation. In: *Java-Magazin* (2004), Nr. 10, S. 77-81.

[ISO95] Europäisches Komitee für Normung: *EN ISO 9241- Teil 10: Grundsätze der Dialoggestaltung* (1995).

[JCP03] Java Community Process: *JSR 168: Portlet Specification, Final Release*, http://www.jcp.org/en/jsr/detail?id=168, (27.10.2003).

[JF88] Ralph E. Johnson, Brian Foote: Designing Reuseable Classes. In: *Journal of Object-Oriented Programming* (1988), Volume 1, Number 2.

[Ka04] Gerti Kappel, Birgit Pröll, Siegfried Reich, Werner Retschitzegger: *Web Engineering - Systematische Entwicklung von Web-Anwendungen*. dpunkt.verlag, 2004.

[KK02] Christian Kamm, Carsten Klein: Komplexe Web-Anwendungen mit „Struts“. In*: Objekt Spektrum* (2002), Nr. 3.

[KP88] Glenn E. Krasner, Stephen T. Pope: A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. In: *Journal of Object-Oriented Programming*, 1 (1988), Nr. 3, S. 26-49.

[Mar00] Mariucci M.: *Enterprise Application Server Development Environments*. Technical Report, Universität Stuttgart, October 2000.

[Ms05] Microsoft Corporation: *Microsoft Developer Network*. http://msdn.microsoft.com/.

[Mu05] Greg Murray: *Asynchronous JavaScript Technology and XML (AJAX). with Java 2 Platform, Enterprise Edition*. http://java.sun.com/developer/technicalArticles/J2EE/AJAX/, (09.06.2005).

[Ne05] NextApp, Inc.: *Echo*. http://www.nextapp.com/products/echo2/.

[RV03] Erhard Rahm, Gottfried Vossen (Hrsg.), *Web & Datenbanken - Konzepte, Architekturen, Anwendungen*. dpunkt.verlag, 2003.

[Sc05] Benjamin schmid: Beflügelte Welten – Eine Einführung in das komponentenbasierte Web-Framework wingS. In: *Java-Magazin* (2005), Nr. 7, S. 98-101.

[Se03] Christian Sell: Eine Typologie für Web-Frameworks – Es muss nicht immer Struts sein. In: *Java-Magazin* (2003), Nr. 7, S. 25-35.

[Ses99] Govind Seshadri: Understanding Java Server Pages Model 2 Architecture. In: *JavaWorld* (1999), Nr. 12.

[SSJ02] Inderjeet Singh, Beth Stearns, Mark Johnson: *Designing Enterprise Applications with the J2EE Platform, Second Edition*. Addison-Wesley, 2002.

[St02] Gernot Starke: *Effektive Software-Architekturen*. Carl Hanser Verlag, 2002.

[ST05] Ludger Springmann, Oliver Tigges: Das webframework Tapestry. In: JavaSpektrum (2005), Nr. 1, S. 18-25.

[Sun98] Todd Sundsted: MVC meets Swing - Explore the underpinnings of the JFC's Swing components. In: *JavaWorld* (1998), Nr. 4.

[Sun05a] Sun Microsystems, Inc.: *Java Servlet Technology*. http://java.sun.com/products/servlet/.

[Sun05b] Sun Microsystems, Inc.: *JavaServer Pages Technology*. http://java.sun.com/products/jsp/.

[Sun05c] Sun Microsystems, Inc.: *Java 2 Platform, Enterprise Edition (J2EE)*. http://java.sun.com/j2ee/.

[Sun05d] Sun Microsystems, Inc.: *JavaServer Faces*. http://java.sun.com/j2ee/javaserverfaces/.

[W3C01] World Wide Web Consortium: *HTML 4.01 Specification*. http://www.w3.org/TR/html401/, (27.10.2001).

[W3C99] World Wide Web Consortium: *Hypertext Transfer Protocol*. http://www.w3.org/Protocols/, (Juni 1999).

[W3C05] World Wide Web Consortium: Document Object Model. http://www.w3.org/DOM/.

[Wa04] Dapeng Wang: Bunte Rahmen – Überblick über Web-Frameworks. In: *Java-Magazin* (2004), Nr. 9, S. 36-44.

[Wi05] wingS Project Team: wingS. http://j-wings.org/.

Abbildungsverzeichnis

Abbildung 2.1: Request-Response-Zyklus 13
Abbildung 2.2: Architektur einer statischen Web-Anwendung 15
Abbildung 2.3: Architektur einer dynamischen Web-Anwendung über die Erweiterung des Web-Servers 16
Abbildung 2.4: Architektur einer dynamischen Web-Anwendung bei Einsatz eines Web-Application-Servers 17
Abbildung 2.5: mögliche Architektur eines Web-Portals 18
Abbildung 2.6: Asynchrone JavaScript-Anfragen (AJAX) [Mu05] 20
Abbildung 2.7: 3-Tier-Modell für J2EE-Anwendungen [SSJ02] 22
Abbildung 2.8: Schichten-Architektur 25
Abbildung 2.9: Model-View-Controller-Architektur [SSJ02] 26
Abbildung 2.10: Model-View-Controller-Architektur für Web-Anwendungen 27
Abbildung 2.11: „Model 1“-Architektur [Ses99] 27
Abbildung 2.12: „Model 2“-Architektur [Ses99] 28
Abbildung 2.13: Pipeline-Architektur 28
Abbildung 3.1: Architektur von aktionsgesteuerten Frameworks 35
Abbildung 3.2: Architektur von ereignisgesteuerten Frameworks 36
Abbildung 3.3: Push-Ansatz zur Datenübergabe an den View 38
Abbildung 3.4: Pull-Ansatz zur Datenübergabe an den View 39
Abbildung 3.5: Kundensuche als endlicher Automat modelliert 40
Abbildung 3.6: Kundensuche als erweiterter endlicher Automat modelliert [KK02] 41
Abbildung 3.7: Arten der Dialogsteuerung 43
Abbildung 3.8: Abhängigkeiten zwischen den Klassifikationsmerkmalen 45
Abbildung 4.1: Schematischer Aufbau des Struts-Frameworks 50
Abbildung 4.2: Ablauf der Request-Verarbeitung im Struts-Controller [Ap05c] 51
Abbildung 4.3: Schematischer Ablauf des Request-Response-Zyklus in Turbine 58
Abbildung 4.4: typische Seitenstruktur in Turbine [Ap05e] 59
Abbildung 4.5: Aufbau des Komponentenbaumes in Tapestry [Ap05f] 61
Abbildung 4.6: Beispiel eines Komponentenbaumes für JavaServer Faces als Objektdiagramm 67
Abbildung 4.7: Request-Response-Zyklus bei JavaServer Faces [Sun05d] 68
Abbildung 4.8: Darstellung von Fenstern innerhalb des Browsers mit Hilfe des Echo-Frameworks 76
Abbildung 4.9: Schematische Darstellung einer möglichen Cocoon-Pipeline [Ap05b] 81
Abbildung 5.1: Anwendungsfälle 86
Abbildung 5.2: Datenmodell zur Speicherung von Nutzern 87
Abbildung 5.3: Darstellung einer panelNavigation-Komponente in JavaServer Faces 95
Abbildung 5.4: Darstellung einer Tabelle zur Nutzerverwaltung mit dataTable- und dataScroller-Komponente in JavaServer Faces 96
Abbildung 5.5: Dialogfluss der implementierten Anwendung als endlicher Automat 97

Tabellenverzeichnis

Tabelle 3.1: Mögliche Ausprägungen von Klassifikationskriterien 47
Tabelle 4.1: Eigenschaften Struts-Framework 49
Tabelle 4.2: Eigenschaften Velocity-Template-Engine 53
Tabelle 4.3: Eigenschaften Turbine-Framework 56
Tabelle 4.4: Auswahl von Turbine-Diensten [Ap05e] 57
Tabelle 4.5: Eigenschaften Tapestry-Framework 60
Tabelle 4.6: Eigenschaften JavaServer-Faces-Spezifikation 65
Tabelle 4.7: Phasen des Request-Response-Zyklus bei JavaServer Faces [Sun05d] 67
Tabelle 4.8: Eigenschaften ASP.NET 70
Tabelle 4.9: Eigenschaften wingS-Frameworks 73
Tabelle 4.10: Eigenschaften Echo-Framework 75
Tabelle 4.11: Eigenschaften Cocoon-Framework 78
Tabelle 4.12: Überblick über die wichtigsten Klassifikationsmerkmale der vorgestellten Frameworks 83

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript Technology and XML
API	Application Programming Interface
ASP	Active Server Pages
CGI	Common Gateway Interface
DBS	Datenbanksystem
DOM	Document Object Model
EIS	Enterprise Information Systems
EJB	Enterprise JavaBeans
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
J2EE	Java 2 Platform, Enterprise Edition
JCP	Java Community Process
JSF	JavaServer Faces
JSP	JavaServer Pages
JVM	Java Virtual Machine
MVC	Model-View-Controller
PDF	Portable Document Format
PHP	PHP Hypertext Preprocessor
RTF	Rich Text Format
SQL	Structured Query Language
SVG	Scalable Vector Graphics
UI	User Interface
URL	Uniform Resource Locator
W3C	World Wide Web Consortium

WML	Wireless Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation
XML	Extensible Markup Language

Index

.NET 23
AJAX 19
Aktionsgesteuertes Framework 35
Anwendungssteuerung 35
  aktionsgesteuert 35
  ereignisgesteuert 35
Applet 19
Architektur 24
ASP.NET 70
Authentifizierung 13
Autorisierung 13
Bindung 38
Black-Box-Framework 33
clientseitige Validierung 43
Cocoon 78
Controller 26
Darstellung 37
  Generierung 37
  Templates 37
  Transformation 37
Datenbanksystem 24
Datenübergabe 38
  Bindung 38
  Pull 38
  Push 38
deklarativen Validierung 44
Dialogsteuerung 39
  Endlicher Automat 39
  explizit 42
  implizit 42
  Programm 41
DOM 19
Dynamische Web-Anwendungen 15
Echo 75
EJB-Container 21
Endlicher Automat 39
Entwurfsmuster 31
Ereignisgesteuertes Framework 35
explizite Dialogsteuerung 42
Framework 31
  Anforderungen 34
  Klassifikationskriterien 34
Generierung 37
HTML 18
HTTP 13
implizite Dialogsteuerung 42
Inversion of Control 33
J2EE 21
Java 2 Enterprise Edition 21
Java Virtual Machine 19
JavaScript 19
JavaServer Faces 65
JSP 22
Klassenbibliothek 32
Klassifikationsschema 45
Komponente 32
Model 26
Model-1 27
Model-2 27
Model-View-Controller 25
MyFaces 65
Pipeline-Architektur 28
Portlet 17
Portlet-Container 17
prozedurale Validierung 44
Pull 38
Push 38
Request-Response-Zyklus 13
Schichten-Architektur 25
serverseitige Validierung 43
Servlet 22
Sitzungsmanagement 44
SQL 24
Statische Web-Anwendungen 15
Struts 49
Tapestry 60
Template Engine 53
Templates 37
Transformation 37
Turbine 56
User-Interface-Komponenten 36
Validierung 43
  clientseitig 43
  deklarativ 44
  prozedural 44
  serverseitig 43
Velocity 53
View 26

Web-Anwendung 12
Anforderungen 13
Architektur 24
Arten 15
Grundkonzept 13
Technologien 18
Vor- und Nachteile 14
Web-Application-Server 16
Web-Portal 17
White-Box-Framework 33
wingS 73

Erklärung

Ich versichere, das ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, den 24. September 2005

Andreas Wende