Diplomarbeit

MVC in Webapplikationen

Ausgeführt zum Zweck der Erlangung des akademischen Grades eines
Dipl.-Ing. (FH) für Telekommunikation und Medien
am Fachhochschul-Diplomstudiengang Telekommunikation und Medien St. Pölten

unter der Leitung von

Dipl.-Ing. Grischa Schmiedl

ausgeführt von

Claudia Katzenbeißer

tm0210038057

St. Pölten, am 30.Juni 2006 Unterschrift:

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Diplomarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

- ich dieses Diplomarbeitsthema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der vom Begutachter beurteilten Arbeit überein.

_____ _____

Ort, Datum Unterschrift

Inhaltsverzeichnis

Kurzfassung

Die Arbeit beginnt mit einer umfassenden Erklärung des Begriffs *Design Pattern*. Neben ihrer geschichtlichen Entwicklung und ihrer Struktur wird dabei ein besonderes Augenmerk auf ihre Fähigkeiten und Vorteile gelegt, aber auch ein kritischer Blick auf ihre Grenzen und Tücken geworfen.

Der zweite Abschnitt umfasst eine detaillierte theoretische Abhandlung des Design Patterns MVC. Dabei wird konkret auf alle 3 Teile des Patterns sowie deren Beziehungen untereinander einzeln eingegangen. Für ein fundiertes Verständnis dieses Entwurfsmusters ist es außerdem notwendig, über weitere Patterns (zur Strukturierung der View, des Models und des Controllers sowie des Zugriffs auf die Datenquelle) Bescheid zu wissen. Aus diesem Grund wird auch diesem Thema ein Kapitel der Arbeit gewidmet sein.

Nach der Vermittlung des theoretischen Wissens wird die Umsetzung von MVC in der Praxis der Webprogrammierung untersucht. Es soll dabei der Frage auf den Grund gegangen werden, ob MVC nur ein theoretisches Gedankenmuster ist, oder darüber hinaus auch tatsächlich in der Entwicklung von Webapplikationen Anwendung findet. Die These *„MVC ist ein Entwurfsmuster, das in der professionellen Webprogrammierung Anwendung findet.“* wird dabei im Mittelpunkt der Untersuchungen stehen.

Sie wird anhand der Vorstellung exemplarisch ausgewählter Webframeworks, die auf unterschiedlichen (etablierten) Programmiersprachen aufbauen und MVC integrieren, als zutreffend bewiesen werden. Neben einer allgemeinen Beschreibung der Arbeitsweise der Frameworks wird dabei natürlich ein besonderer Schwerpunkt auf deren Implementierung der MVC-Architektur gelegt. Die verbale Dokumentation wird dabei zusätzlich durch illustrative Codebeispiele verdeutlicht werden.

Die zweite These dieser Abhandlung *„Die Verwendung von Frameworks in Kombination mit MVC bringt gegenüber der Programmierung ohne diese Hilfsmittel eine Effizienzsteigerung mit sich.“* wird ebenfalls über ein einfaches und dennoch für die Webprogrammierung typisches Codebeispiel untersucht werden. Sie muss aufgrund der damit gewonnenen Erkenntnisse letztendlich relativiert werden.

Abstract

This Diploma Thesis starts with a comprehensive explanation of the term *Design Pattern*. Besides their historical development and their structure there will be turned some special attention to their abilities and benefits as well as to their weaknesses and their perils.

The second section contains a detailed theoretical description about the design pattern MVC. Each of the 3 parts of this pattern as well as their relationships among each other will be treated. For an established understanding of MVC it is also crucial to have some knowledge about the other patterns that MVC brings about (for further structuring the View, the Model and the Controller as well as the access to the database). For that reason one chapter of this work is devoted to this topic too.

After this part full of theoretical knowledge the implementation of MVC in practice will be examined. We will get to the bottom of the question, if MVC is just some theoretical mind pattern or if it is really used in development of web applications beyond.
The thesis *"MVC is a design pattern that is used in professional programming."* will be in the centre of our studies.

She will be proved correct on the basis of introducing some exemplarily chosen frameworks that rely on different (established) programming languages and integrate MVC. Beside some general description of these frameworks' functioning, the main focus will be put on their implementation of the MVC-architecture. The verbal documentation will be pointed up additionally by some illustrative code examples.

The second thesis of this disquisition *"The use of frameworks in combination with MVC brings about an increase of efficiency in comparison with programming without those devices."* will also be analysed by developing an easy, but however typical code example of web application programming. Because of the gained experiences it must be put into perspective finally.

1 Einleitung

Die Anforderungen an Webapplikationen haben sich im Laufe der letzten Jahre drastisch verändert.
Zu Beginn wurde das Internet hauptsächlich als Präsentationsplattform angesehen und eine statische HTML (Hypertext Markup Language) -Seite mit einem guten Design als Online-Visitenkarte empfand man schon als das Non-Plus-Ultra.
Doch nach der Entdeckung des Marktpotentials dieses Mediums stiegen die Erwartungen drastisch. Nicht nur ein besseres Design, sondern vor allem ein Mehr an Interaktivität und Dynamik wurde gefordert. CMS (Content Management Systeme), E-Shops, Buchungs- und Bestellungssystemen begegnet man heutzutage schon fast auf jeder Webseite in irgendeiner Form.
(Fast) jedes noch so kleine Unternehmen hat in seinem Webauftritt zumindest eines dieser oben genannten Features eingebunden.

Noch eine weitere Entwicklung ist zu beobachten: Benutzerschnittstellen von komplexen Unternehmensanwendungen werden immer öfter als Webapplikationen implementiert. Das universelle Zugriffsverfahren sowie das Wegfallen der Installation zusätzlicher Client-Software sind besonders in unserer immer globalisierteren Arbeitswelt zwei entscheidende Vorteile, die diese Entscheidung rechtfertigen. Trotz dieser Verlagerung in das Web muss die Applikation natürlich nach wie vor den Ansprüchen nach Sicherheit, effizienter Fehlerbehandlung, Skalierbarkeit und Integration anderer Anwendungen gerecht werden.

Parallel zu der Wandlung der Nutzung des Mediums Internet vollzog sich natürlich auch eine Weiterentwicklung der Methoden und Mittel zur Erstellung der entsprechenden Applikationen.
Zu Beginn fand man mit HTML als reine Darstellungssprache sein Auskommen. Clientseitige Skriptsprachen wie zum Beispiel JavaScript ermöglichten in Folge die Einbindung erster Dynamik. Erweitert wurde diese Fähigkeit durch serverseitige Skriptsprachen.
PHP (PHP Hypertext Processor) als ein Vertreter dieser letzten Gruppe ist ein Musterbeispiel für die Weiterentwicklung der im Web gängigen Programmiermethoden. Am Anfang als rein prozedurale Skriptsprache für Enthusiasten allgemein belächelt und kaum für große kommerzielle Webprojekte einsetzbar, etablierte sie sich spätestens mit der Einführung von PHP 5 zu einem ernstzunehmenden Werkzeug der professionellen Webprogrammierung. Während die Zend-Engine positiv zur Schnelligkeit und Stabilität der Sprache beiträgt, bietet das neue Objektmodell von PHP 5 eine solide Unterstützung für objektorientierte Programmierung.

Mit dem Einzug der OOP (objektorientierten Programmierung) in die Welt der Webprogrammierung wird es auch zunehmend interessanter, sich mit Design Pattern zu beschäftigen.

Da ihre Lösungsansätze fast ausschließlich auf der OOP aufsetzen, waren sie bis vor kurzem Sprachen wie bspw. PHP vorenthalten.

Die Frage, die sich vielen nun aufdrängt, ist, ob man Design Pattern, die ja ursprünglich aus der Welt der klassischen objektorientierten Programmierung kommen nun so einfach auf die Entwicklung von Webapplikationen ummünzen kann und ob den Eigenheiten des Mediums Internet damit genügend Rechnung getragen wird.

Diese Abhandlung wird anhand eines populären Vertreters von Design Pattern namens MVC versuchen, aus einem möglichst objektiven Blickwinkel eine Antwort darauf zu finden.

Außerdem werden auch einige Frameworks im Zuge dieser Arbeit unter die Lupe genommen und deren Einsatz in Verknüpfung mit MVC untersucht. Die Frage, die in diesem Zusammenhang gestellt wird, ist, ob durch den kombinierten Einsatz von Frameworks und Design Pattern eine Effizienz- und Qualitätssteigerung des Entwicklungsprozesses erreicht werden kann und sie in Zukunft die Arbeitsweise in der Webprogrammierung nachhaltig beeinflussen.

Die theoretische Abhandlung des MVC-Patterns sowie die formale Beschreibung der vorgestellten Frameworks erfolgt anhand einer Inhaltsanalyse, wobei Bücher, Fachzeitschriften sowie einschlägige Internetseiten und Foren als Quellen herangezogen werden.

Die so erarbeiteten Erkenntnisse werden zusätzlich durch selbst programmierte Beispiele verdeutlicht und unterstrichen.

Die in dieser Arbeit verwendeten geschlechterbezogenen Bezeichnungen gelten aus Gründen der Lesbarkeit für beide Geschlechter. Bsp.: „In letzter Zeit gewinnt es auch immer mehr Beliebtheit unter Cold-Fusion und PHP Entwicklern.“ Damit ist sowohl ein Entwickler als auch eine Entwicklerin gemeint.

Es werden dadurch keine Gruppen diskriminiert oder gering geschätzt.

Claudia Katzenbeisser

2 Design Pattern

2.1 Ihre Entstehungsgeschichte

Schon bei der Definition dieses Begriffes scheiden sich die Geister. Mein persönlicher Favorit ist die Formulierung des Urvaters der Design Pattern (dt. Entwurfsmuster), Christopher Alexander:

„Jedes Pattern beschreibt ein Problem, das in unserer Umgebung immer wieder auftritt, und beschreibt dann den Kern der Lösung so, dass Sie diese Lösung beliebig oft anwenden können, ohne sie jemals genau auf dieselbe Weise durchzuführen.“
(Fowler 2003, S:24)

Die Geburtsstunde der Design Pattern war in den 1970er Jahren. Der Architekt Christopher Alexander fertigte zu dieser Zeit eine Sammlung von allgemeinen Entwurfsmustern an. Die Verbreitung seines Konzeptes fand aber vielmehr in der Softwareentwicklung als in seiner eigenen Branche statt. In den späten 80er Jahren wurde seine Idee wieder aufgegriffen und für die Erstellung von grafischen Benutzerschnittstellen in Smalltalk genutzt. Im Gegensatz zu anderen alten Handwerken, die vor hunderten oder sogar tausenden Jahren aufkamen und sich immer wieder weiterentwickelten, konnte man in der Programmierung nicht aus einem derart reichen Erfahrungsschatz schöpfen. Die Programmierer betraten also mehr oder weniger Neuland und mussten sich den Herausforderungen in dieser neuen Disziplin stellen.

Mit dem Aufkommen von Java und C++ bekamen Design Pattern ein größeres Publikum. Hier profitierte man davon, dass Entwurfsmuster den Übergang von einer objektorientierten Sprache zur anderen (meist) unbeschadet überstehen. Das bedeutet, dass man nicht immer warten muss, bis ein Design Pattern für die jeweilige Sprache angepasst ist, sondern von den Erkenntnissen in anderen Sprachen profitieren kann.

Zum entscheidenden Durchbruch von Design Patterns kam es 1995 mit der Veröffentlichung des Werkes „*Design Patterns - Elements of Reusable Object-Oriented Software*“ durch Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. Sie sind unter dem Spitznamen *Gang of Four*, kurz GOF jedem, der sich mit diesem Thema beschäftigt, ein Begriff und auch ihr Buch, in dem 23 Entwurfsmuster im Rahmen von Smalltalk und C++ beschrieben sind, ist ein Standardwerk im Bereich Software Engineering über Entwurfsmuster. (vgl. http://de.wikipedia.org/wiki/Design_pattern und http://www.zend.com/php/design/patterns1.php, 29.03.2006)

2.2 Wozu brauche ich sie?

Ganz gleichgültig, ob wir nun vor einem Problem in der Programmierung stehen oder vor einem ganz alltäglichen – die Vorgehensweise ist vom Prinzip her immer dieselbe.

Einzelkämpfer versuchen, sich ohne fremde Hilfe der Herausforderung zu stellen und werden zumindest beim ersten Mal mit ziemlich hoher Wahrscheinlichkeit in dieselben Fallen tappen wie schon viele vor ihnen. Mit der Zeit werden sie zwar Methoden herausfinden, um das Problem effizienter, schneller, günstiger, etc. zu lösen, aber all das kostet eine Menge Zeit und Geld – also genau das, was uns in der heutigen schnelllebigen, von Konkurrenz bestimmten Arbeitswelt fehlt.

Schlauere Menschen bauen auf die Erfahrung anderer. Sie holen sich Rat von Personen, die schon einmal vor einem ähnlichen Problem standen, befragen Experten und überlegen sich basierend darauf einen Lösungsansatz. Sie profitieren von den Fehlern anderer insofern, als dass sie über diese Bescheid wissen und sie deshalb (hoffentlich) umgehen. Das spart Kosten, Zeit und Nerven und bedeutet auch einen gewaltigen Vorsprung gegenüber den Einzelkämpfern.

„Die Entwicklungszeit verkürzt sich nach erfolgreicher Übernahme der Pattern auf 1/3 oder ¼. […] Software-Entwicklung ist teuer, jedoch steht nach umfangreichen Analysen von unzähligen Projekten (oft auch Open Source) fest, dass weit über die Hälfte der Kosten nur dadurch entstehen, dass das Rad dauernd neu erfunden wird.“ (http://www.little-idiot.de/his/design-pattern.htm, 29.03.2006)

Dieses Zitat macht deutlich, wie groß das Potential, das hinter Design Pattern steckt, bei richtiger Nutzung sein kann.

2.3 Was können sie?

Design Pattern sind in der Praxis erprobte und bewährte Lösungsanleitungen für typische, immer wiederkehrende Entwurfsprobleme.

Eine wesentliche Eigenschaft von Pattern ist die Praxiserprobtheit. Entwurfsmuster werden vom Autor also nicht alleine im stillen Kämmerchen ausgedacht, aufgeschrieben und veröffentlicht. Sie entstehen vielmehr durch die Beobachtung und das Studieren von funktionierenden Systemen. Autoren von Design Pattern versuchen, dem Kern des Problems auf die Schliche zu kommen und ihre Erkenntnisse dann festzuhalten. Genau aus diesem Grund spricht man auch von der „Entdeckung“ und nicht der „Erfindung“ von Entwurfsmustern. Es sind keine neuen Ideen, sondern einfach nur Vorgänge in der Praxis, die festgehalten werden. (vgl. Fowler 2003, 24 f)

2.4 Was können sie NICHT?

So nützlich Werkzeuge auch sein mögen: Sie alle bergen ihre Tücken und Gefahren und je weniger man mit ihrer Handhabung vertraut ist, desto wahrscheinlicher ist es, dass man sich damit verletzt.

Design Pattern bilden dabei keine Ausnahme. Sie werden oft als Kochrezepte angesehen, die ohne eigenständiges Denken angewendet werden können. So einfach ist das aber auch wieder nicht.

Erstens muss man selbst entscheiden, welches Pattern für die Lösung des Problems, vor dem man steht, am passendsten ist. Doch selbst wenn dieser Schritt getan ist, wird man

gezwungen sein, das Entwurfsmuster noch individuell an die eigenen Bedürfnisse zu adaptieren. Kein Pattern-Autor kann sämtliche Varianten voraussehen, die ein Software-Projekt annehmen kann. Patterns sind also als ein sinnvoller Ausgangspunkt und nicht als das Ziel anzusehen. (vgl. Fowler 2003, 24ff)

2.5 Ihre Vorteile

1) Beschleunigung des Developmentprozesses

   Dadurch, dass getestete und bewährte Lösungsansätze verwendet werden, können Fehler von vorne herein vermieden und Ausnahmesituationen berücksichtigt werden.

   Kostenintensive Neuprogrammierungen in späteren Projektphasen durch anfängliche Designfehler werden so unterbunden. Das Resultat ist ein schnellerer Projektablauf, ein zufriedener Kunde und ein größerer Gewinn.

2) Qualitativ bessere Software

   All die Erfahrungen bzgl. Effizienz, Wartbarkeit, Anpassbarkeit und Erweiterbarkeit, die Vorgänger mühsam in wochenlanger, kostenintensiver Entwicklung gewinnen mussten, stehen nun auf einmal gesammelt zur Verfügung.

   Das Endprodukt entspricht also auch qualitativ gesehen höchsten Ansprüchen.

3) Gemeinsames Vokabular und Konzeptverständnis

   Angenommen, man hat nun nach jahrelanger Erfahrung einen beachtlichen Wissensschatz angesammelt und sein „persönliches Design-Pattern“ entwickelt. Wie erklärt man einem neuen Teammitglied, das noch keinen Einblick in das Projekt hat, mit wenigen Sätzen die mühsam gewonnenen Erkenntnisse?

   Design-Pattern geben uns Namen zur kurzen und treffenden Benennung komplexer Techniken. Dies ermöglicht eine reibungslose Kommunikation zwischen den Teammitgliedern und vermeidet Missverständnisse.

4) Sprachübergreifende Verwendbarkeit

   Viele Entwurfsmuster können auf alle objektorientierten Programmiersprachen praktisch ident angewandt werden. Das bedeutet, dass man nicht warten muss, bis ein Pattern in einer bestimmten Sprache beschrieben wurde, bevor man es verwenden kann, sondern dass von den Entdeckungen der Programmierer anderer Sprachen profitiert werden kann.

5) Pattern bringen weitere Pattern hervor

   Ein Pattern schafft oft Bedingungen, unter denen die Verwendung eines anderen Patterns sinnvoll ist. Umgekehrt können mehrere zusammenarbeitende Pattern zu einem großen übergeordneten kombiniert werden.

   (vgl. http://www.zend.com/php/design/patterns1.php, 29.03.2006)

2.6 Ihre Nachteile

1) Problematik der richtigen Anwendung

   Welches Design Pattern für das aktuelle Problem das passendste ist, ist nicht immer gleich offensichtlich. Eine kluge Entscheidung setzt eine gewisse Erfahrung und Vorkenntnis über Entwurfsmuster voraus.

2) Design Pattern sind nicht automatisch perfekt

   Da jeder berechtigt ist, Design Pattern zu veröffentlichen, ist es nicht zu vermeiden, dass manchmal auch Unausgereiftes publiziert wird. Man sollte sich also auch hier die Erfahrungen anderer mit diesem Entwurfsmuster vor dessen Verwendung zunutze machen.

3) Verständnis für OOP ist notwendig

   Beinahe alle Design Pattern bauen ihre Lösung auf objektorientierten Prinzipien auf. Es ist also nicht möglich, deren Vorteile mit prozeduraler Programmierung zu nutzen.

2.7 Ihre Struktur

Für die Struktur von Design Pattern gibt es keine fixen Regeln, sehr wohl aber Diskussionen. Martin Fowler beispielsweise, ein renomierter Referent und Autor meherer bedeutender Sachbücher zum Thema Softwarearchitektur, bevorzugt einen weniger formalen Ansatz währen die GOF ziemlich strenge Kategorien definiert hat.

Wie auch immer man dazu stehen möchte, über 3 essentielle Komponenten der Entwurfsmuster wird man wohl nicht hinwegsehen können:

2.7.1 Das Problem

Jeder erfahrene Programmierer weiß, dass die wahre Herausforderung oft nicht in der Behebung, sondern im Finden eines Fehlers liegt. Dasselbe gilt bei der Verwendung von Design Pattern. Man muss zuerst ein Verständnis für das Problem, mit dem man es zu tun hat, entwickeln, um dann eine vernünftige Auswahl des Werkzeuges, das man zu dessen Lösung verwenden will, zu treffen.

Die Beschreibung eines Design Patterns sollte also immer mit einer Zusammenfassung und einer Beschreibung des Problems, das es behandelt, beginnen.

Die GOF meint dazu: *„In general, it's easier to see what someone is doing than to know why, and the 'why' for a pattern is the problem it solves. […] A pattern author must determine and characterize the problem that the pattern solves, even if you have to do it after you've discovered the solution.“* (http://www.zend.com/php/design/patterns1.php, 29.03.2006)

2.7.2 Die Lösung

Die Lösung eines Patterns sollte auf mehrere Arten dargestellt werden. Neben einer theoretischen Erklärung, die mit einem UML-Diagramm (Unified Modeling Language), unterstri-

chen wird, müssen auch anschauliche Codebeispiele in einer oder mehreren objektorientierten Sprachen beigefügt sein. Diese Exempel sollten einfach gehalten sein, damit man sich auf die Arbeitsweise des Patterns konzentrieren kann.

Die GOF stellte außerdem noch die „Regel der drei“ auf. Das bedeutet, dass ein Pattern drei in der Praxis verwendete Bespiele dokumentieren muss, um als gültig anerkannt zu werden.

Zu diesem Zeitpunkt ist noch einmal darauf hinzuweisen, dass Design Patterns nicht mit Kochrezepten zu verwechseln sind. Es wird in den seltensten Fällen möglich sein, diese beispielhaften Codefragmente per „copy and paste“ ohne Änderungen in sein eigenes Projekt einzufügen. Der Sinn dieser praktischen Beispiele liegt auch vielmehr in der Veranschaulichung der Lösung und des Aufbaus als in der Bereitstellung von fixfertigen Softwarepaketen.

2.7.3 Der Kontext

Das Problem, das in einem Design Pattern beschrieben wird, ist in einen Kontext eingebettet. Bei der Entwicklung eines Entwurfmusters schafft man eine neue Sammlung von Bedingungen. Diese Konsequenzen, ob positiv oder negativ werden im Detail beleuchtet sowie andere Patterns, die miteingebunden werden könnten.

(vgl. http://www.zend.com/php/design/patterns1.php, 29.03.2006)

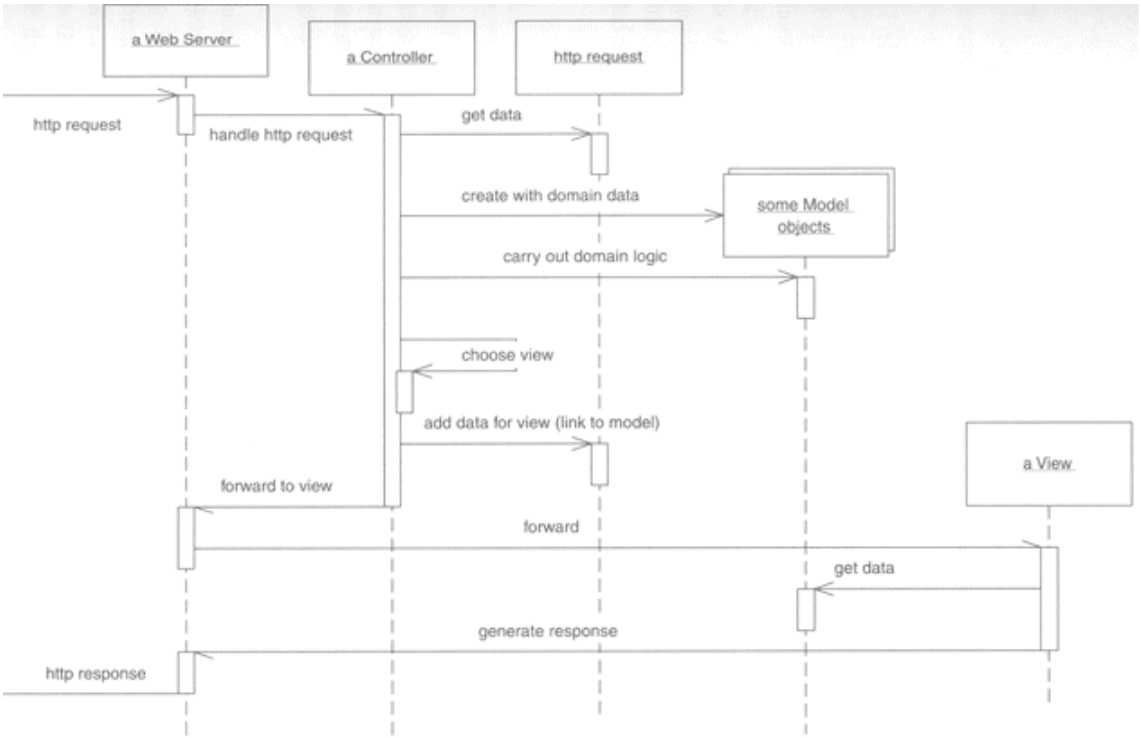
3 Model-View-Controller (MVC)

Ein Pattern, das in letzter Zeit zunehmend auch in der Webentwicklung Verbreitung gefunden hat, ist das Model-View-Controller Pattern.

Das erste Mal beschrieben im Jahr 1979 von Trygve Reenskaug für Benutzeroberflächen in Smalltalk inspirierte es in Folge etliche andere GUI (Graphical User Interface)-Frameworks (wie zum Beispiel Microsoft Foundation Classes, Swing und Qt) und beeinflusst das Denken über den Entwurf von Benutzerschnittstellen noch heute nachhaltig. In letzter Zeit gewinnt es auch immer mehr Beliebtheit unter Cold-Fusion und PHP Entwicklern.

3.1 Der Aufbau

MVC ist ein Design Pattern, das die Interaktion mit der Benutzerschnittstelle in drei verschiedene Rollen zerlegt. Grob gesagt analysiert der Controller in dieser Aufteilung die Eingaben des Benutzers, das Model übernimmt die Verarbeitung der Daten und die View die Ausgabe des Systems (siehe **Fehler! Verweisquelle konnte nicht gefunden wer-**



den.1).

Abbildung 1: Überblick über die Zusammenarbeit von Model, View und Controller in einem Webserver (Fowler 2003, S:73, Abb.4.1)

3.1.1 MVC im Three-Tier Model

Das Three-Tier Model (dt. drei Schichten Modell) ist ein Pattern der Softwarearchitektur, das zwischen folgenden drei Schichten unterscheidet:

1) Presentation Layer (Präsentationsschicht): In dieser Schicht wird das User-Interface umgesetzt. Sie ist für die Repräsentation von Daten und der Entgegennahme von Benutzereingaben verantwortlich.
2) Business Layer (Logikschicht): Die Logikschicht umfasst sämtliche Verarbeitungsmechanismen. Die Programmintelligenz ist an dieser Stelle vereint.
3) Data Layer (Datenschicht): Hier werden Informationen in einen Datenspeicher (Datenbank, Filesystem) gespeichert bzw. von dort geladen.

Folgende Grafik veranschaulicht die Position der 3 Rollen des MVC innerhalb des soeben beschriebenen Three-Tier Models:

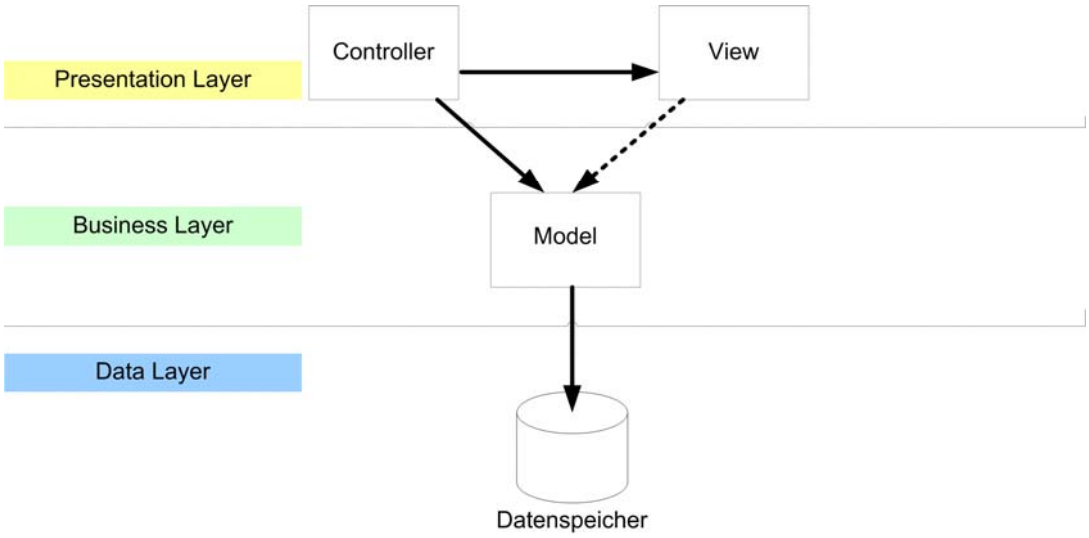


Abbildung 2: Die Position der 3 Rollen des MVC innerhalb des Three-Tier Modells

Während sowohl der Controller als auch die View dem Presentation Layer zuzuordnen sind, befinden sich die Aufgaben des Models gänzlich im Bereich des Business Layers. Für den Bereich des Data Layer ist in MVC hingegen keine spezielle Rolle vorgesehen. Da diese Schicht aber ebenfalls essentielle Aufgaben übernimmt und das Model eng mit der Datenquelle zusammenarbeiten muss, wird zwecks einer ganzheitlichen Übersicht im folgenden auch auf diesen Layer detaillierter eingegangen werden.

3.1.2 Das Model

Das Model ist ein nicht-visuelles Objekt, das sämtliche Informationen der Domäne inklusive der Business-Regeln repräsentiert. Es ist der Kern des Programms, der alle zentralen Operationen ausführt. Die Daten, die das Model der View zur Verfügung stellt, sind in der Regel darstellungsneutral, das heißt noch nicht mit einer Formatierung versehen. Der Vorteil da-

von ist, dass die Daten eines Models von verschiedensten Views weiterverwendet werden können.

Entscheidend für die erfolgreiche Realisierung dieses Patterns ist, dass das Model vom UI (User Interface) unabhängig bleiben muss. Es darf sich also weder auf Teile der View noch des Controllers beziehen und keine direkten Instanzvariablen, die auf diese beiden anderen Rollen verweisen, enthalten. Die Dienste und Daten des Models werden den anderen Schichten des MVC also nur passiv angeboten. Die Unabhängigkeit des Models kann sich sogar so weit auswirken, dass ein einziges Objekt als Model in mehreren MVC-Umsetzungen dienen kann.

Je nachdem, ob das Model nach Änderungen an seinen Daten die Aufgabe übernimmt, die anderen zu benachrichtigen oder nicht, unterscheidet man zwischen einem aktiven und einem passiven Model:

3.1.2.1 Passives Model vs. aktives Model

Bei der Verwendung eines passiven Models übernimmt der Controller die Aufgabe, die View darüber zu informieren, wann sie aktualisiert werden muss. Durch die strikte Anfrage – Antwort Kommunikation in http (Hypertext Transfer Protocol) wird die View automatisch bei jedem Zyklus neu berechnet, ganz egal ob nun Änderungen am Model erfolgt sind oder nicht. Aus diesem Grund wird das passive Model standardmäßig in Webapplikationen verwendet.

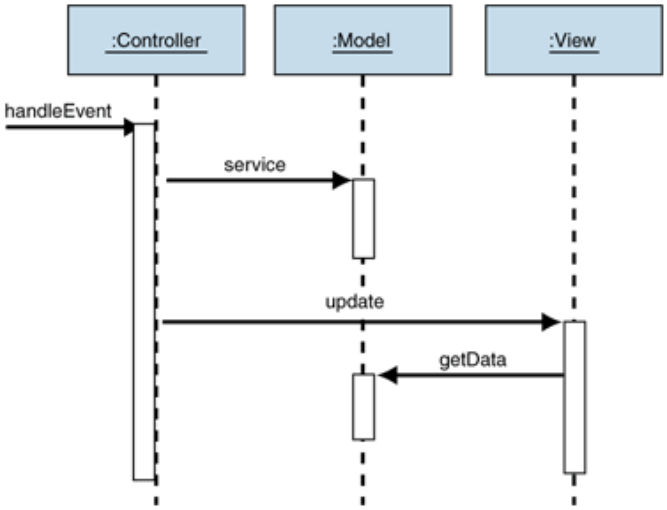


Abbildung 3: Verhalten eines passiven Models (vgl. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/ImpFrontControllerInASP.asp, 28.04.2006)

Ein aktives Model beinhaltet unter Verwendung des Observer-Patterns einen Änderungsmeldungsmechanismus. Das bedeutet, dass die View und der Controller informiert werden, sobald Modifikationen am Model vorgenommen wurden. Da das Model trotzdem keine spezifischen Kenntnisse über die beiden anderen Komponenten besitzt, werden die Regeln der Unabhängigkeit nicht gebrochen. Dieser Mechanismus ermöglicht sofortiges Updaten der Anzeige und ist ein charakteristisches Kennzeichen für die GUI-Programmierung.

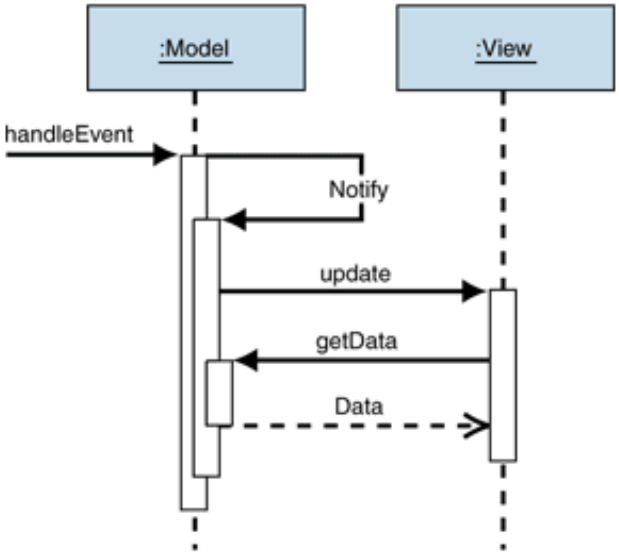


Abbildung 4: Verhalten eines aktiven Models (vgl. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/ImpFrontControllerInASP.asp, 28.04.2006)

3.1.3 Die View

Die View übernimmt einzig und allein die Aufgabe der Anzeige der Model-Daten in der Benutzerschnittstelle. Für Webapplikationen wurde dafür bis jetzt überwiegend die Auszeichnungssprache HTML verwendet. Dieses Format ist zwar nach wie vor das dominante, hat aber in letzter Zeit ernstzunehmende Konkurrenz bekommen wie bspw. Flash oder alternative Auszeichnungssprachen wie XHTML (Extensible Hypertext Markup Language), XML/XSL (Extensible Markup Language/ Extensible Stylesheet Language) oder WML (Wireless Markup Language).

Der zunehmende Aufwand der Verwaltung all dieser unterschiedlichen Interfaces wird durch MVC insofern begrenzt, als dass sämtliche Datenverarbeitungsprozesse aus diesem Teil des Patterns ausgegrenzt sind.

Views können (bzw. müssen) mittels Abfragen, die vom Model zur Verfügung gestellt werden, lesend auf dieses zugreifen. Änderungen am Status des Models sind der View nicht erlaubt.

3.1.4 Der Controller

Der Controller hat die Aufgabe, Eingaben des Anwenders entgegenzunehmen (im Web in der Regel HTTP-Anfragen). Daraus entnimmt er Informationen und ruft dementsprechend Methoden des zugehörigen Model-Objekts auf, die den Zustand des Models verändern. Bei einem passiven Model werden die Ergebnisse dieser Operationen vom Controller analysiert und er informiert dann die View, die die Antwort darstellen soll. Bei der aktiven Variante erfolgt die Benachrichtigung der View über den Änderungsmeldungsmechanismus des Models.

Der Controller ist KEIN Vermittler zwischen dem Model und der View. Es herrscht das weit verbreitete Missverständnis, dass er zwischen Model und View sitzt und Daten unter diesen beiden Schichten weiterreicht. Der Controller gibt dem Model lediglich Informationen,

die es für Berechnungen braucht und informiert die View im Fall eines passiven Models, wenn sich dieses geändert hat. Die View hat dieselben lesenden Rechte am Model wie der Controller und holt sich die Daten zur Darstellung selbst. Das Pattern *Presentation Abstraction Control*, das wegen desselben Grundprinzips der Trennung von UI und Domänenlogik gerne mit MVC verwechselt wird, verwendet den Controller als Vermittler zwischen der Logik und der Anzeige.

3.1.5 Die Beziehungen zwischen den 3 Rollen

3.1.5.1 Beziehung Model-View

Das wichtigste Prinzip, das einem guten Software-Design zugrunde liegt, ist die Trennung von View und Model, weil:

1) Die beiden Rollen verfolgen grundsätzlich verschiedene Ziele: Bei der Entwicklung einer View stehen Anliegen wie Benutzerfreundlichkeit, Klarheit des Seitenaufbaus sowie ein ansprechendes Design im Mittelpunkt. Beim Model hingegen geht es rein darum, die relevanten Geschäftsprozesse logisch abzubilden und die Interaktion mit der Datenquelle zu optimieren. Für diese unterschiedlichen Aufgaben werden meist nicht nur andere Bibliotheken verwendet, sondern sie werden sogar von zwei verschiedenen Mitarbeitern, die sich beide auf eines der Gebiete spezialisiert haben, implementiert.
2) Model und View haben unterschiedliche Beweggründe für Änderungen und werden unterschiedlich oft abgeändert.
3) Je nach Kontext ist es notwendig, dieselben Daten des Models auf unterschiedliche Arten darzustellen. Mit einer strikten Trennung kann man nicht nur vielfältige Präsentationen, sondern sogar mehrere Schnittstellen entwickeln, die auf dasselbe Model zurückgreifen. So kann eine Applikation sowohl über einen Rich-Client, einen Webbrowser, ein Remote-API und eine Befehlszeilenschnittstelle aufgerufen werden.
4) Das Testen nicht-visueller Objekte ist meistens einfacher als das visueller Objekte. Bei der Fehlersuche in der Domänenlogik wird man also nicht von Nebensächlichkeiten, die die Darstellung betreffen, abgelenkt.

Der wesentliche Punkt, den man in der Trennung dieser beiden Rollen berücksichtigen muss, ist die Richtung der Abhängigkeiten. Die Präsentation hängt IMMER vom Model ab, aber das Model NIE von der Präsentation.

3.1.5.2 Beziehung View-Controller

Ironischerweise sind diese beiden Rollen in fast allen Smalltalk-Versionen nicht getrennt bzw. sehr eng miteinander verbunden. Jede View ist genau mit einem Controller verknüpft und umgekehrt.

Das klassische Beispiel, mit dem die Trennung der beiden Rollen begründet wird, ist die Präsentation einer editierbaren und einer nicht editierbaren Version derselben Daten. Zur Lösung dieser Aufgabe ist die Verwendung einer View und zwei Controllern sinnvoll. In der Praxis kommen die meisten GUI-Frameworks mit einem Controller pro View aus, sodass die beiden nicht getrennt werden.

Die Tatsache dieser fehlenden Trennung in der GUI-Programmierung ist Ursache für große Missverständnisse rund um das MVC-Pattern.

Die Aufgaben von Model und View sind leicht zu verstehen, aber was ist eigentlich der Controller? Bestärkt durch die unglückliche und ungenaue Namensgebung (der Terminus Controller wird nämlich in mehreren Kontexten verwendet) besteht die allgemeine Auffassung vieler, dass er – wie der Application Controller – zwischen dem Model und der View sitzt. Tatsache ist, dass der Controller in MVC mit einem Application Controller außer dem Namen nichts gemeinsam hat. Für die Rolle, von der wir sprechen, wäre der Name Input-Controller treffender.

Mit dem Einzug von MVC in die Webprogrammierung ist die Trennung zwischen View und Controller wieder aktuell geworden und das Verständnis, was die 3. Rolle in diesem Entwurfsmuster eigentlich tut, hat sich verbessert.

3.1.5.3 Beziehung Model-Controller

Der Controller ist vom Model abhängig. Änderungen am Interface des Models können also auch Modifikationen am Controller nach sich ziehen. Abhängigkeiten in die entgegengesetzte Richtung (Model vom Controller abhängig) müssen allerdings unbedingt unterbunden werden.

3.1.6 MVC2

Der Begriff MVC2 wurde von Sun geprägt, um die MVC-Architektur für webbasierte, zustandslose Anwendungen zu optimieren (http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html).
Ein wesentliches Kennzeichen dieser Abwandlung des klassischen Modells ist die explizite Trennung des Controller-Codes vom Content (also der View).

Außerdem ist das Model nun nicht mehr dafür verantwortlich, die View über Änderungen seiner Daten zu informieren.
Der Grund dafür ist, dass die Verbindung zwischen Server und Client statuslos ist. Das bedeutet, dass die Lebensdauer einer View oder eines Controllers nicht der des Models entspricht, wie das bei Desktopapplikationen der Fall ist. Das Leben eines Controllers bzw. einer View startet mit einem Request vom Webbrowser und endet mit dem Response. Das Model hingegen überdauert in der Regel viele dieser Zyklen. Daher ist es ihm nicht möglich, Objekte der View über interne Statusänderungen zu informieren.
Daraus ergeben sich mehrere Konsequenzen: Der Controller muss, nachdem er die entsprechenden Änderungen am Model hervorgerufen hat, die betroffene View über Änderungen des Models informieren. Der Code des Models wird durch das Wegfallen dieser Aufgabe generischer, da die Logik zum Auslösen von Events wegfällt und keine Eventlistener mehr registriert und freigegeben werden müssen.

3.2 Die Vorteile

Die Entwicklung von Webseiten erfolgte bis vor kurzem größtenteils mit prozeduraler Programmierung (PHP, ASP, Cold Fusion).

Als unerfahrener Entwickler gerät man durch die Möglichkeit, HTML in den restlichen Code einzubinden, automatisch in Versuchung, den für die Logik verantwortlichen Code (bspw. Datenbank-Queries, Verarbeitung der Benutzereingaben) mit der Darstellungslogik (üblicherweise HTML) bunt zu vermischen.

Obwohl die meisten routinierten Programmierer nach einer Trennung von Präsentations- und Applikationslogik streben, resultiert das eher aus langwierig gewonnener Erfahrung als auf einem bewussten Plan.

Durch die Verwendung von MVC wird man zu dieser konsequenten Aufteilung gezwungen. Die Vorteile sprechen für sich:

1) mehrere Darstellungen mit nur einem Model möglich

   Das ist wahrscheinlich einer der gewichtigsten Vorteile dieses Entwurfmusters. In unserer multimedialen Welt besteht eine immer größere Nachfrage nach mehr Zugriffsmöglichkeiten auf eine Applikation (PC, PDA, Handy…). Mit der Verwendung von MVC ist es gleichgültig, ob das Interface des Benutzers ein komplexes Flash oder ein reduziertes simples HTML ist – das Model kann mit beiden umgehen. Dadurch, dass die Daten und die Business Logik von der Anzeige unabhängig sind, wird außerdem Codevervielfältigung vermieden.

   Umgekehrt ist es dadurch, dass das Model die Daten nicht formatiert, möglich, dass jedes Interface dieselben Komponenten des Models aufrufen kann und dann nach seinen individuellen Bedürfnissen anpasst.

2) Das Model isoliert und verwaltet die Datenpersistenz.

   Komplexe Applikationen wie zum Beispiel E-Shops oder E-Commerce funktionieren gleich unabhängig von der Darstellungsart.

3) Die Änderung des UI betreffen das Model nicht

   Änderungen am UI sollten bei korrekter Einbindung des Entwurfsmusters keinen Einfluss auf die Arbeitsweise des Models haben.

   Das wird dadurch ermöglicht, dass die 3 Teile des Patterns *Black Boxes* sind. Das bedeutet, dass die interne Funktionsweise der einzelnen Teile für die anderen nicht sichtbar und auch nicht von Bedeutung ist, sondern nur das äußere Verhalten.

4) Einfacheres Testen

   Der Kern der Applikation, eingeschlossen im Model kann unabhängig von den anderen Komponenten sorgfältig getestet werden.

5) Controller als Steuerelement

   Durch den modularen Aufbau dieses Entwurfmusters ist es für den Controller ein leichtes, je nach Bedarf die benötigten Bausteine des Models zu kombinieren und die Darstellung der Ergebnisse aufzurufen.

3.3 Die Nachteile

Natürlich hat alles zwei Seiten und die Vorteile dieses Patterns ziehen auch so manchen Nachteil mit sich:

1) Es erfordert genaue Planung

   MVC zwingt den Programmierer zu einer sorgfältigen, detaillierten Planung. Man muss speziell bei den ersten Projekten mit diesem Entwurfsmuster viel Zeit und Konzeptionsarbeit investieren und seine Applikationen komplett neu überdenken.

2) Es ist komplex

   Mit der Aufteilung der Applikation in die 3 Teile ist es noch lange nicht getan. In größeren Projekten wird man nicht darum herumkommen, sich auch mit den zahlreichen Hilfspatterns, die MVC mit sich bringt, zu beschäftigen.

3) Enge Kopplung der View und des Controllers an das Model

   Das Model ist zwar völlig unabhängig von den beiden anderen Komponenten, allerdings gilt diese Beziehung nicht in die Gegenrichtung. Änderungen des Models erfordern fast immer auch parallele Modifikationen an der View und oft sogar am Controller.

4) Aufteilung in die 3 Teile manchmal uneindeutig

   Das ist mit einem einfachen Beispiel leicht zu illustrieren: Nehmen wir an, in unserem System gibt es die Regel „Negative Kontostände müssen rot angezeigt werden.“ Auf den ersten Blick scheint dies eindeutig eine Darstellungsaufgabe zu sein und somit im Verantwortungsbereich der View zu liegen. Diese Art der Gliederung wäre allerdings falsch und würde die Prinzipien des MVC verletzen. Bei genauerer Überlegung merkt man nämlich, dass diese Anforderung ein Teil unserer Geschäftslogik ist und daher das Model betrifft.

   Dadurch, dass die Benutzereingabe logischerweise immer einen Einfluss auf die Ausgabe des Systems hat, ist die strikte Trennung zwischen Controller und View noch schwieriger, wenn nicht manchmal unmöglich.

5) Mehr Dateien zu verwalten

   Natürlich sind mit der Aufteilung in die 3 Verantwortungsbereiche auch mehr Dateien verbunden.

6) Für kleine Applikationen zu aufwendig

   Bei kleinen Applikationen steht der Mehraufwand, der durch die Anwendung dieses Design-Patterns entsteht, in keinem Verhältnis zum Nutzen.

   (vgl. http://builder.com.com/5100-6386-1049862.html, 30.03.2006)

3.4 Die Implementierung

Das MVC-Pattern kann in Projekte verschiedenster Größe implementiert werden. Im Minimalfall handelt es sich um 3 Objekte, von denen jeweils ein kleiner Teil an der Umsetzung

des Patterns beteiligt ist. Ein einfaches Beispiel dafür wäre eine Checkbox-Komponente, die Daten des Model-Objektes verändert.

Diese kleinen Module können dann zu immer größeren kombiniert werden. Aus mehreren Checkboxen, Textfeldern, Radiobuttons, etc. wird ein Formular, aus mehreren Formularen ein Bestellsystem, etc. Wichtig dabei ist, die Vorgaben des MVC-Patterns nie aus den Augen zu verlieren und weiterhin konsequent durchzusetzen.

Bei großen Applikationen ist es nicht damit getan, die 3 Teile des Entwurfsmusters je in eine Klasse zu verfrachten. Jede dieser Klassen würde als ein heilloses Durcheinander enden und wohl kaum wieder verwendbar sein.

Um das zu vermeiden, sollte man andere Hilfspatterns verwenden, um den individuellen Ansprüchen von MVC auch bei großen Projekten weiterhin gerecht werden zu können.

3.4.1 Strukturierung des Models

Zur Organisation der Domänenlogik gibt es 3 bekannte Pattern: *Transaction Script*, *Domain Model* und *Table Module*:

3.4.1.1 Transaction Script

Dieses Pattern ist das einfachste der hier vorgestellten und geht davon aus, dass man Geschäftsanwendungen als eine Reihe aufeinander folgender Transaktionen auffassen kann. Prinzipiell wird für jede Aktion, die der Benutzer ausführen könnte, ein Transaction Script vorgesehen. Dieses ist im Wesentlichen eine Prozedur, die dann je nach Bedarf Eingaben prüft, Berechnungen durchführt, Operationen oder Daten von anderen Systemen abruft, Daten in Datenbanken speichert etc. Auf der Code-Ebene können gemeinsame Unterfunktionen auch in Subroutinen, die dann von mehreren Transaction Scripts gemeinsam genutzt werden, ausgelagert werden, aber dem Motto „pro Aktion eine Prozedur“ bleibt man trotzdem treu.

Die Organisation von Prozeduren in Klassen kann auf 2 Arten geschehen:

Die gebräuchlichere und in den meisten Fällen sinnvollere Methode ist, themenverwandte Transaction Scripts in einer einzelnen Klasse zusammenzufassen.

Unter Verwendung des *Command-Patterns* kann man aber auch jedes Transaction Script in eine einzelne Klasse auslagern. In diesem Fall kann man die Exemplare von Skripts zur Laufzeit als Objekte behandeln.

Eine weitere Möglichkeit ist, auf Klassen komplett zu verzichten und das Pattern rein über globale Funktionen umzusetzen.

Was eindeutig für die Verwendung dieses Patterns spricht ist die Einfachheit. Es ist ein für die meisten Entwickler schnell zu verstehendes, prozedurales Modell, das bei simplen Anwendungen sehr intuitiv erscheint.

Durch die Verwendung von *Row Data Gateway* und *Table Data Gateway* (zwei bekannte Architektur-Pattern für die Datenquelle) arbeitet es außerdem gut mit einer einfachen Datenquellen-Schicht zusammen.

Mit steigender Komplexität der Domänenlogik erhöht sich der Verbesserungsaufwand der Applikation jedoch exponentiell. Dadurch, dass sich jedes Skript auf die Bearbeitung einer einzelnen Transaktion konzentriert, gerät man leicht in Versuchung, bei ähnlichen Aufgaben Code zu duplizieren. Bis zu einem gewissen Grad kann die Auslagerung gemeinsam genutzter Codesegmente in Subroutinen weiterhelfen, doch ab einem bestimmten Komplexitätsgrad ist das Ausweichen auf ein anderes Pattern der Übersichtlichkeit des Programms zuliebe wohl unausweichlich.

3.4.1.2 Domain Model

Bei dem Domain Model wird, wie der Name schon sagt, ein Modell der Domäne erstellt. Zur Abbildung des Geschäftsbereiches wird eine komplette Schicht von Objekten eingefügt. Dabei richtet man sich hauptsächlich nach den Substantiven des Anwendungsbereichs. (Beispiel: Behandelt das System einen Geschäftsprozess, so gibt es wahrscheinlich ein Kundenobjekt, ein Produktobjekt, ein Vertragsobjekt, etc.). Die Umsetzung der Geschäftsregeln erfolgt so, dass sich jedes Objekt um die Berechnungen kümmert, die es selbst betreffen.

Der größte Vorteil dieses Patterns ist eindeutig die extreme Skalierbarkeit.
Durch die streng methodische Vorgangsweise können auch zunehmend komplexe Systemerweiterungen bewältigt werden. Anstatt immer wieder neue und noch verzweigtere Bedingungen in einem Skript hinzufügen zu müssen, kann man die zusätzliche Logik in einem neuen Objekt zusammenfassen.

Der Nachteil ist, dass die Arbeit mit diesem Modell ein gewisses Maß an Übung erfordert. Die Konzeption und der Einsatz dieses Entwurfsmusters ist mit einer größeren Arbeit verbunden als beim Transaction Script und besonders für Entwickler, die bis zu diesem Zeitpunkt ausschließlich prozedural programmiert haben, ist das Umdenken schwierig. Sieht man sich die vielen begeisterten „Objekt-Fanatiker“ an, lässt das aber vermuten, dass sich der anfängliche Mehraufwand bezahlt macht.

3.4.1.3 Table Module

Auf den ersten Blick erkennt man keinen nennenswerten Unterschied zwischen dem Table Module und dem Domain Model. Beide haben Klassen wie bspw. Kunde, Vertrag, Produkt. Das Table Module widerspricht allerdings dem traditionellen objektorientierten Ansatz (der beim Domain Model verwendet wird), dass Objekte eine Identität besitzen. Das bedeutet, dass eine Instanz der Kunden-Klasse für einen einzelnen Kunden steht und ich mittels Methoden alle möglichen Operationen mit ihm ausführen kann.

Beim Table Module Pattern wird die Geschäftslogik so aufgeteilt, dass eine Klasse einer Tabelle in der Datenbank entspricht. Der Hauptunterschied besteht darin, dass es beim Domain Model für mehrere Kunden pro Kunde ein Objekt gibt, während das Table Module genau ein Objekt enthält, über das alle Kunden verwaltet werden. Um das bewältigen zu können, ist die Verwendung von Record Sets erforderlich. Möchte man mit einem bestimmten Kunden Operationen ausführen, muss ein Schlüssel zur Identifikation als Argument mit übergeben werden. Im Regelfall ist das der Primärschlüssel der entsprechenden Tabelle in der Datenbank.

Dieses Entwurfsmuster ist in vielen Aspekten als ein Kompromiss zwischen den obigen beiden anzusehen. Durch die stärkere Strukturierung der Geschäftslogik mittels Tabellen anstatt von Prozeduren wird das Auffinden und Entfernen von dupliziertem Code gegenüber dem Transaction Script erleichtert. Auch bei der Nutzung der Stärken einer relationalen Datenbank hat diese Lösung gegenüber dem Domain Model die Nase vorne. Der Preis dafür ist der Verzicht auf fortgeschrittene Techniken wie Vererbung und andere OOP-Pattern.

Ein entscheidender Faktor für die Attraktivität des Table Modules ist die Unterstützung von Record-Sets in der Umgebung, mit der man arbeitet.

3.4.1.4 Welches Pattern soll man verwenden?

Auf diese Frage gibt es keine pauschale Antwort. Um zu einer vernünftigen Entscheidung zu kommen, muss man Faktoren wie die Komplexität des zugrunde liegenden Models, die Kenntnisse und Vorlieben des Entwicklungsteams, die Projektdurchlaufzeit, die Umgebung, mit der man arbeitet etc. berücksichtigen.

Abbildung 5 versucht, die soeben vorgestellten Pattern nach den Kriterien der Komplexität und des Aufwands einzureihen. Man erkennt, dass beim Domain Model der Einstiegsaufwand zwar deutlich höher als bei den anderen beiden liegt, der Mehraufwand bei steigender Komplexität der Domainlogik aber bei weitem nicht so hoch wie bei den anderen Entwurfsmustern ist. Das Hinzufügen zusätzlicher Funktionen bei Transaction Script und Table Module ist mit exponentiell steigendem Aufwand verbunden.

Der Grund für die fehlende Achsenbeschriftung des Diagramms ist, dass man beide verwendeten Kenngrößen nur schwer messbar machen kann. In der Praxis bedeutet das, dass man nur durch Erfahrung und eine Analyse der Systemanforderungen ein ungefähres Gefühl dafür bekommen kann, wo man mit seiner Anwendung auf der x-Achse liegt.

In manchen Fällen kann es auch eine überlegenswerte Alternative sein, die Patterns zu kombinieren. Angenommen, man erstellt eine Bildschirmmaske zur Bearbeitung von Aufträgen. Es gibt einige Verhaltensweisen, die genau einmal benötigt werden. Weist man die dafür zuständigen Funktionen der Auftrags-Klasse zu, so läuft man in Gefahr, dass diese zu groß und unübersichtlich wird (was leider ein häufigeres Problem in der Anwendungslogik darstellt). Dieses Problem kann umgangen werden, indem aufgabenspezifische Funktionen außerhalb der Kernklasse in ein Transaction Script ausgelagert werden.
Bei dieser Vorgangsweise kann jedoch wieder das Problem von Codeduplikation und Inkonsistenzen auftauchen, was die Komplexität dann erst recht wieder erhöht.
Dieses Thema ist also eine Streitfrage, über die auch unter erfahrenen Programmierern Uneinigkeit herrscht.

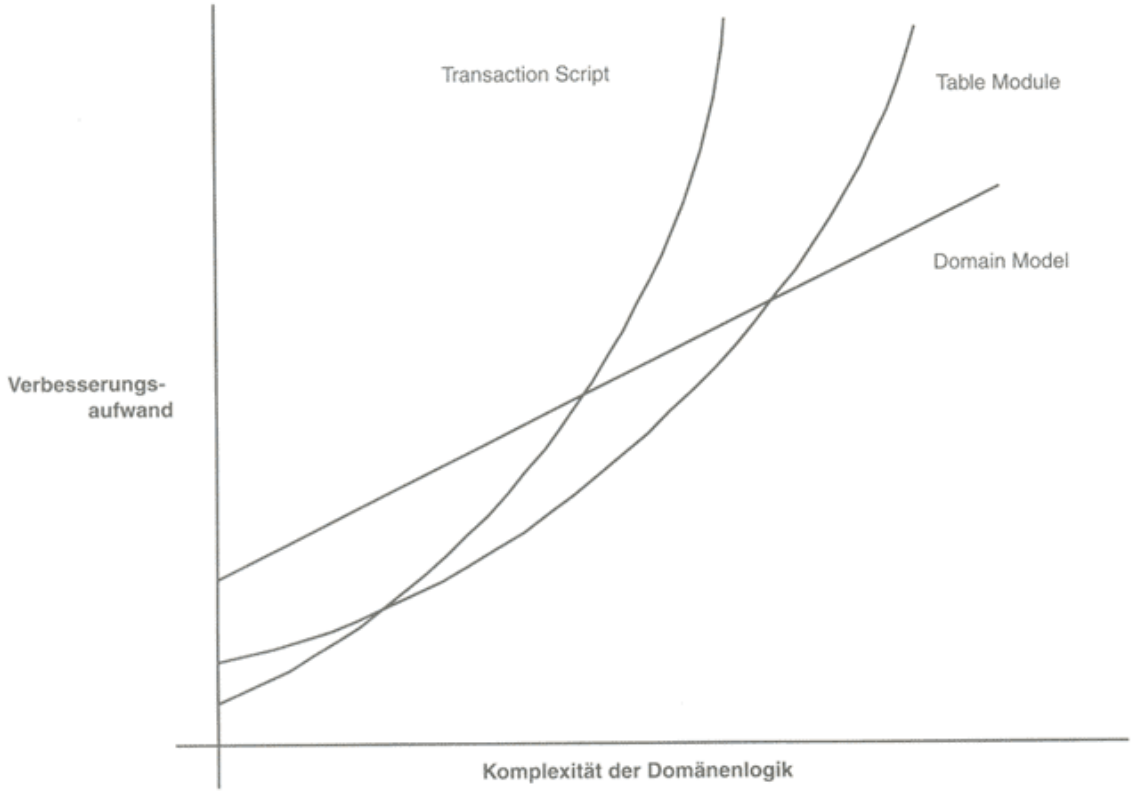


Abbildung 5: Verhältnis zwischen Komplexität und Aufwand bei verschiedenen Patterns zur Strukturierung der Domänenlogik (Fowler, 2003, S:46, Abb. 2.4)

Die folgende Tabelle soll noch einmal einen kurzen Überblick über die Stärken und Schwächen der vorgestellten Patterns zur Organisation der Domänenlogik geben:

	Transaction Script	Domain Model	Table Module
Vorteile	✓ leicht verständliches, intuitives, prozedurales Modell ✓ durch die Verwendung von Row Data und Table Data Gateway gute Zusammenarbeit mit einfachen Datenquellenschichten ✓ Transaktionen leicht abgrenzbar	✓ hochkomplexe Domänenlogik mit vertretbarem Aufwand durch objektorientierten Ansatz lösbar	✓ stärker strukturiert als Transaction Script ✓ gute Integration in Architekturen, die mit Record Sets arbeiten ✓ nutzt Stärken der relationalen DB

Nachteile	- Codeduplizierung - fehlende Struktur - exponentiell steigender Arbeitsaufwand bei steigender Komplexität der Domänenlogik	- Entwurf und Einsatz erfordert Anstrengung, Erfahrung und Planung	- Verzicht auf manche objektorientierte Ansätze

Tabelle 1: Vergleich unterschiedlicher Patterns zum Organisieren der Domänenlogik

3.4.2 Strukturierung der View

In diesem Bereich sind 3 Ansätze von praktischer Bedeutung: *Transform View*, *Template View* und *Two Step View*. Die *Two Step View* stellt dabei keine Alternative, sondern vielmehr eine Erweiterung oder Ergänzung zu den anderen beiden Patterns dar.

3.4.2.1 Template View

Die Idee dieses Patterns ist, dynamische Informationen in eine statische HTML-Seite durch spezielle Markierungen einzufügen. Vor dem Anzeigen der Seite werden diese dann durch Ergebnisse von Berechnungen (z.B. Datenbankabfragen, Session-Daten, etc.) ersetzt.

Der Name dieses Entwurfsmusters rührt daher, dass die statischen HTML-Informationen eine Art Schablone (engl. template) für die spätere Antwort darstellen.

3.4.2.1.1 Markierungen

Es gibt 3 wesentliche Methoden, wie diese in die HTML-Seite eingefügten Markierungen aussehen können:

1) Verwendung von HTML-ähnlichen Tags
   Diese Art der Auszeichnung ist besonders geeignet bei der Arbeit mit WYSIWYG (What you see is what you get)-Editoren, da diese Ausdrücke in spitzen Klammern ignorieren.
2) Verwendung spezieller Markierungen im Body-Text
3) Verwendung von Server Pages (ASP, JSP, PHP)
   Hier kann beliebiger Skriptcode – in diesem Zusammenhang auch als Skriptlet bezeichnet – in die HTML Seite eingebettet werden.
   Server Pages umfassen also eine weit größere Funktionalität als es die grundlegende Form des View-Patterns vorsieht.
   Doch genau das ist wiederum das Problem bei dieser Methode: Man gerät als Programmierer viel zu leicht in Versuchung, mehr als nur die Anzeigelogik in die Seite zu integrieren und somit die Grundregeln der Schichtentrennung zu brechen. Die Folge ist unübersichtlicher, schlecht strukturierter und nicht wiederverwendbarer Code und in weiterer Konsequenz Codeduplizierung – also all das, was das MVC-Pattern verhindern möchte.

3.4.2.1.2 Hilfsobjekte

Eine vernünftige Alternative zu Skriptlets sind Hilfsobjekte. Pro Seite gibt es also ein Objekt, das die gesamte Programmlogik enthält. In der HTML-Seite sind dann nur noch Aufrufe auf dieses Objekt zu finden. So kann sich der Designer wieder rein der optischen Gestaltung der Seite widmen, während sich der Programmierer im Hilfsobjekt um den Code kümmert.

In einfachen Systemen ist dieses Prinzip ideal: Die Markierungen werden in Aufrufe des Hilfsobjekts übersetzt und deren Rückgabewerte von der Engine in die Seite eingefügt. Somit ist sämtliche Logik aus der View verbannt und der im Hilfsobjekt eingeschlossene Code übersichtlich, modular und wieder verwendbar.

3.4.2.1.3 Logische Kontrollstrukturen

Es gibt natürlich auch kompliziertere Fälle: Angenommen, eine Textpassage soll unter einer gewissen Bedingung hervorgehoben werden. Die Verwendung eines *if*-ähnlichen Befehls scheint unvermeidbar.
Eine Lösungsmöglichkeit besteht darin, die Auszeichnung schon im Hilfsobjekt zu generieren. Allerdings entzieht man dem Designer damit die Entscheidungskraft über die Art der Hervorhebung. Will man das nicht, kommt man um die Verwendung einer Bedingung im Template nicht herum. Hierbei ist es entscheidend, dass die Bedingung auf einer einzigen booleschen Eigenschaft des Hilfsobjektes basiert und kein komplexer Ausdruck in die Seite eingefügt wird.
Bei Iterationen über Arrays (bspw. Anzeigen einer Tabelle mit einer Liste von Kundenaufträgen) ist die Verwendung einer Schleifenlogik ebenfalls praktisch unvermeidbar.

Doch macht man damit nicht aus dem Templatecode wieder eine eigene Programmiersprache? Was ist dann der große Unterschied zu Skriptlets?
Die Faustregel bei der Verwendung logischer Ausdrücke in Templates ist, dass sie immer ausschließlich der Anzeigelogik dienen dürfen. Dadurch unterscheiden sie sich auch von Skriptlets, die viel weitreichendere Aufgaben erfüllen.

3.4.2.1.4 Conclusio

Zusammenfassend liegen die Stärken dieses Patterns nicht nur in seiner hohen Leistungsstärke und Flexibilität, sondern besonders darin, dass man das Template praktisch ohne Programmierkenntnisse mit einem WYSIWYG-Editor erstellen kann. Der Designer kann also nach wie vor problemlos den Seitenaufbau per HTML festlegen, während sich der Programmierer auf den dahinterliegenden Code, der die notwendigen Daten zusammensammelt, konzentriert.

Diesen Vorteilen liegen zwei gravierende Schwachpunkte gegenüber:
Erstens erfordert es sehr viel Disziplin, die Logik in der View rein auf die Anzeige zu beschränken und den Rest in das Hilfsobjekt auszulagern. Nur zu leicht rutscht ein Skriptteil hinüber in die View und erschwert damit die Wartung für Nicht-Programmierer.
Template-Views sind im Hinblick auf den Einsatz am Webserver entworfen worden und können deshalb schwer bis gar nicht anders getestet werden.

3.4.2.2 Transform View

Während bei der Template View die Ausgabe im Mittelpunkt steht, wandelt die Transform View die Domänendaten elementweise in HTML-Code um.
Eine einfache Schleife steuert den Transformationsablauf. Sie untersucht jedes Eingabeelement separat, wählt die passende Transformation und wendet diese auf das aktuelle Element an.

Ein Großteil der Transform Views wird heute mit XSLT (Extensible Stylesheet Language Transformation) geschrieben. Für die XSLT-Transformation müssen die Daten im XML-Format vorliegen. Die Arbeit mit einem DOM (= Document Object Model) ist dabei in der Regel einfacher und schneller als mit einem String.
Der XML (Extensible Markup Language)-Code wird gemeinsam mit dem dazugehörigen XSLT-Stylesheet, das die Regeln der Transformation enthält, an die XSLT-Engine übergeben. Aus der Anwendung des Stylesheets auf den XML-Input resultiert der HTML-Output.

3.4.2.2.1 Conclusio

In der Praxis ist XSLT heutzutage die dominierende Sprache zum Entwickeln von Transform Views. Die Vor- und Nachteile dieses Patterns sind im Moment also sehr an die Stärken und Schwächen von XSLT gekoppelt.

Da XSLT auf die Transformation von XML-Dokumenten spezialisiert ist, arbeitet diese Kombination besonders effektiv zusammen.
Außerdem kann derselbe XSLT-Code zur Umwandlung von XML-Dokumenten unterschiedlichster Quellen (J2EE, .NET) verwendet werden.
Die Nachteile der Template View sind gleichzeitig die Stärken der Transform View. Die Transform View erschwert das Einfügen von Logik in die Anzeige und vereinfacht den Testprozess dadurch, dass sie nicht vom Webserver abhängig ist.

Die Nachteile, die man dafür in Kauf nehmen muss, sind unausgereifte Tools zur Unterstützung von XSLT sowie die schwerere Erlernbarkeit dieser Sprache. XSLT hat als ein Vertreter der funktionalen Programmiersprachen nämlich eine etwas andere Struktur als man es aus der objektorientierten Programmierung gewohnt sein mag.

3.4.2.3 Two Step View

Dieses Pattern steht wie vorher schon erwähnt nicht in Konkurrenz mit den obigen beiden, sondern stellt vielmehr eine Ergänzung zu ihnen dar.
Wie der Name dieses Entwurfmusters schon vermuten lässt wird die Erstellung des HTML-Codes aus den Domänendaten in zwei Stufen zerlegt. Im ersten Schritt werden die Geschäftsdaten in eine logische Präsentation ohne Formatierung umgewandelt. Erst im zweiten Schritt erfolgt die Generierung des HTML-Codes.

Die Aufgabe der ersten Stufe ist also, für die aktuelle Bildschirmseite relevante Daten aus dem Domänenmodell zu extrahieren und dann in einer präsentationsorientierten Struktur aufzubauen. Das Ergebnis ist eine unformatierte Anordnung verschiedenster Daten und Steuerungselemente wie zum Beispiel Header, Footer, Tabellen oder Formulare. Deren Darstellung in HTML ist zu diesem Zeitpunkt noch völlig offen.

Diese erste Stufe ist abhängig vom darzustellenden Bildschirm und muss daher für jede Seite der Webanwendung extra ausprogrammiert werden.

In der zweiten Stufe erfolgt die Umwandlung in den HTML-Code. Das geschieht von einem einzigen Objekt aus, das sämtliche Elemente der präsentationsorientierten Struktur aller Bildschirme kennt und weiß, welche HTML-Formatierungen darauf anzuwenden sind.

Der Gedanke hinter diesem Pattern war, dass sämtliche Seiten einer Web-Anwendung ein konsistentes Aussehen und einen einheitlichen Aufbau haben (sollten). Um globale Änderungen der Site zu erleichtern, ist es sinnvoll, die Entscheidung über den zu verwendenden HTML-Code auf eine einzige Seite zu konzentrieren. So muss nur ein Objekt modifiziert werden, um jede Bildschirmmaske der Webseite umzugestalten.
Einstufige Template oder Transform Views erschweren diese Aufgabe, weil Präsentations- und Transformationsentscheidungen über mehrere Seiten hinweg dupliziert werden und somit alle betroffenen Dateien einzeln geändert werden müssen.

Der Einsatz dieses Entwurfmusters ist natürlich nur sinnvoll, wenn sich die logische Präsentation der Daten über die Site hinweg nicht ändert, d.h. wenn das grundlegende Layout (Positionierung des Headers, Footers, Navigation, etc.) gleich bleibt. Bei sehr designbetonten Webseiten, bei denen keine logische Bildschirmstruktur entwickelt werden kann, ist die Verwendung dieses Patterns hingegen nicht möglich.

Für die Implementierung der Two Step View gibt es mehrere Möglichkeiten:

1) Entwickeln eines zweistufigen XSLT-Programmes

   Anstatt jeder Seite ein einzelnes XSLT-Stylesheet zur Umwandlung der Domänenlogik in HTML-Code zuzuweisen, wie es bei der einstufigen Transform View passiert, gibt es hier 2 Stylesheets. Das erste transformiert den domänenorientierten XML-Code in eine präsentationsorientierte Form und das zweite stellt den XML-Code in HTML dar.

2) Verwendung von Klassen

   Ein Satz von Klassen ist für die präsentationsorientierte Struktur verantwortlich. Die Domäneninformation wird zusammengestellt und Objekte dieser Klassen in einer logischen Bildschirmstruktur abgebildet. Die Umwandlung dieser Informationen in HTML kann entweder durch Methoden einer HTML-Darstellungsklasse oder durch Methoden der präsentationsorientierten Klassen selbst erfolgen.

Diese beiden Ansätze verfolgen die Organisation der Transform View. Es gibt aber auch für die Template View zweistufige Umsetzungsmöglichkeiten:

3) Logische Tags

   Im 1. Schritt sind lediglich logische Tags und kein HTML-Code in der Seitendefinition zu finden. Nahe liegend ist hier die Verwendung des XML-Formats, was das Arbeiten mit WYSIWYG-Editoren ausschließt. Das Template System wandelt dann die logischen Tags in HTML-Code um. Ein praktisches Beispiel für diese Variante sind die Webform-Controls von .NET.

3.4.2.3.1 Conclusio

Da die Entscheidung über den zu verwendenden HTML-Code bei diesem Pattern auf genau eine Stelle konzentriert ist, fallen globale Änderungen am Aussehen der Site im Vergleich zu den einstufigen Modellen sehr leicht.
Besonders wertvoll kann das sein, wenn man eine Webanwendung für mehrere Frontend-Kunden zur Verfügung stellen möchte. Ein gutes Beispiel dafür ist das Reservierungssystem unterschiedlicher Reiseanbieter. Innerhalb der Grenzen des logischen Bildschirmes kann jede Organisation das Erscheinungsbild der Applikation individuell gestalten, indem einfach die 2. Stufe der Two Step View abgeändert wird.

Angenommen, diese Anwendung umfasst 5 unterschiedliche Bildschirmmasken. Bei 3 verschiedenen Darstellungen würde man bei der Verwendung eines einstufigen Modells pro Seite und pro Darstellung ein View-Modul benötigen, also insgesamt käme man auf 5 (= Anzahl der Bildschirmseiten) mal 3 (=Anzahl der Darstellungen) Module. Bei der Two Step View benötigt man ein Modul pro Bildschirmmaske für die erste Stufe der präsentationsorientierten Darstellung und in der zweiten Stufe nochmals ein Modul pro Darstellung. D.h. in unserem Fall kämen wir auf 5 (= Anzahl der Bildschirmseiten) plus 3 (= Anzahl der Darstellungen) Module.

Schon an diesem einfachen Beispiel merkt man, dass in der Verwendung der Two Step View bei mehreren Darstellungsarten ein enormes Einsparungspotential liegt. Je umfangreicher die Darstellungen und Bildschirmmasken sind, desto mehr kommt dieser Nutzen zu tragen.

Ähnliches gilt bei der Bedienung unterschiedlicher Ausgabegeräte (bspw. für Anwendungen, die auf einem PDA (Personal Digital Assistant), einem Handy und in einem Browser ausgegeben werden). Auch hier muss – unter der Voraussetzung, dass überall dieselben Informationen angezeigt werden – nur noch die 2. Stufe den Bedürfnissen der Darstellung angepasst werden.

Die Nutzung der Two Step View mit all ihren Vorteilen ist allerdings an die Bedingung geknüpft, dass die präsentationsorientierte Struktur der unterschiedlichen Darstellungen dieselbe ist. Nur wenn es gelingt, ein Mindestmaß an strukturellen Gemeinsamkeiten zu finden, ist der Einsatz dieses Patterns auch möglich. Eine gewisse Einengung der gestalterischen Möglichkeiten ist dabei unvermeidbar. Für viele Websites und auch die unterschiedlichen UIs von Ausgabegeräten sind diese Einschränkungen oft schon zu weit reichend und die Verwendung dieses Patterns damit nicht mehr interessant.

Das Erlernen dieses Patterns mit seinen unterschiedlichen Schichten ist außerdem sicherlich mit einem höheren Aufwand verbunden als das bei einer intuitiven Template View der Fall sein mag.

3.4.2.4 Welches Pattern soll man verwenden?

Wie bei den Design Pattern für Models gilt auch hier, dass man keine generelle Antwort auf diese Frage geben kann.
Jedes einzelne Entwurfsmuster hat seine Stärken und Schwächen und es hängt sehr vom Team, den Tools, mit denen man arbeitet, und der Projektgröße ab, welches Pattern das günstigste ist.

Die folgende Tabelle versucht, noch einmal eine Übersicht über die Vor- und Nachteile der auf den vorigen Seiten vorgestellten View-Patterns zu geben.

	Template View	Transform View	Two Step View
Vorteile	✓ ermöglicht Arbeitstrennung: Designer erstellt Template Programmierer das Hilfsobjekt ✓ intuitive visuelle Erstellung des Seitenaufbaus ohne Programmierkenntnisse möglich (bspw. mit WYSIWYG-Editor) ✓ hohe Leistungsstärke und Flexibilität	✓ Effektive Zusammenarbeit XSLT ↔ XML ✓ Portierbarkeit von XSLT auf unterschiedlichste Web-Plattformen ✓ Transformation leicht auf die Darstellung beschränkbar ✓ Testen auch ohne Webserver leicht möglich	✓ Globale Änderungen der Seite an einer zentralen Stelle ✓ Einsparung von View-Modulen bei mehreren Bildschirmmasken und Darstellungen (1 Modul pro Bildschirmseite plus 1 Modul pro Darstellung)
Nachteile	- Logik anderer Schichten kann leicht in die View eingefügt werden - Abhängigkeit vom Webserver beim Testen - bei globalen Änderungen: mehrere Dateien müssen abgeändert werden - gleicher Code in mehreren Dateien - große Anzahl von Templates bei mehreren Bildschirmmasken und Darstellungen (1 Modul pro Bildschirmseite mal 1 Modul pro Darstellung)	- Tools für XSLT unausgereift - schwerere Erlernbarkeit von XSLT als funktionale Programmiersprache - bei globalen Änderungen: mehrere Dateien müssen abgeändert werden - gleicher Code in mehreren Dateien - große Anzahl von Templates bei mehreren Bildschirmmasken und Darstellungen (1 Modul pro Bildschirmseite mal 1 Modul pro Darstellung)	- alle Seiten bzw. UIs müssen dieselbe präsentationsorientierte Struktur aufweisen → Einschränkung der Gestaltungsmöglichkeiten - selbst für kleine Änderungen der Darstellung Programmierkenntnisse notwendig - schwerere Erlernbarkeit

Tabelle 2: Vergleich unterschiedlicher View-Patterns

3.4.3 Strukturierung des (Input)-Controllers

Ein Input-Controller hat zwei wesentliche Aufgaben: Erstens die Verarbeitung von HTTP-Requests und zweitens die Entscheidung, was damit zu geschehen hat.
Im Detail schaut das so aus:

1. Schritt: Decodieren der URL und extrahieren aller für die Aktion benötigten Daten.

2. Schritt: Die Model-Objekte, die für die Verarbeitung verantwortlich sind, erstellen.
   Die HTML-Daten werden entweder durch Setzen der Properties der Objekte oder als Argumente beim Methodenaufruf übergeben, sodass das Model nicht direkt mit dem HTTP-Request in Berührung kommt und somit unabhängig bleiben kann.

3. Schritt: Zuletzt wird die View, die das Ergebnis anzuzeigen hat, bestimmt und aufgerufen.

In dieser Arbeit werden 2 gebräuchliche Pattern, die diese Rolle übernehmen, beschrieben: der *Page Controller* und der *Front Controller.*

3.4.3.1 Page Controller

Dieses Pattern verfolgt den Ansatz, jeder logischen Seite einer Website einen Input-Controller zuzuordnen. Um ganz korrekt zu sein ist ein Controller mit Aktionen verbunden – beispielsweise dem Anklicken eines Links oder einer Schaltfläche.

Es gibt zwei Wege zur Implementierung dieses einfach zu verstehenden Modells:

Die Verwendung von Server Pages, wo der Page Controller mit einer Template View kombiniert wird, ist nur bei äußerst einfachen Anzeigeseiten empfehlenswert. Sobald komplexere Logik zur Gewinnung der Daten oder der Entscheidung über die zu verwendende View ins Spiel kommt, ist Skriptlet-Code auf der Seite meist umständlich und eine saubere Strukturierung des Programmmoduls nicht mehr möglich.
Dem kann man Abhilfe schaffen durch den Einsatz eines Hilfsobjekts. Dieses umfasst die gesamte Steuerungslogik und wird von der Server-Page aufgerufen. Nach der Bearbeitung wird die Kontrolle wieder an die aufrufende oder eine andere Server-Page zurückgegeben, die dann die Anzeige der Ergebnisse übernimmt.

Eine andere Möglichkeit ist, dass sämtliche Aufgaben des Handlers und Controllers in ein separates Skript ausgelagert werden. Diesem wird vom Webserver die Kontrolle übergeben, es führt die Verarbeitung des HTTP-Requests durch und bestimmt die View zur Anzeige.

Dabei ist es nicht notwendig, einen Page Controller in einer einzelnen Klasse zu realisieren. Gibt es mehrere Controller, die ähnliche Aufgaben erfüllen, ist es zur Vermeidung von Codeduplikation sogar ratsam, die Abschnitte, die sonst kopiert werden müssten, in ein gemeinsames Hilfsobjekt auszulagern.

Die beiden soeben vorgestellten Umsetzungsmethoden schließen einander nicht aus, was bedeutet, dass man sie problemlos in einem Projekt nebeneinander einsetzen kann. Aktionen mit einfacher Controller-Logik werden durch Server-Pages verarbeitet, komplexere durch ein Skript. Beschränkt man sich auf die durchgängige Verwendung einer Methode,

bringt das den Vorteil der Konsistenz, dafür aber auch durch unübersichtliche Skriptlets überladene Server-Pages bzw. einfache Durchlaufskripts mit sich.

3.4.3.2 Front Controller

Anstatt für jede ausführbare Aktion einen eigenen Controller zu erstellen, basiert dieser Ansatz auf der Idee, dass es nur ein Objekt gibt, das sämtliche Anfragen der Website verarbeitet.

Dieses Pattern besteht im Wesentlichen aus zwei Teilen: dem Web-Handler und der Befehlshierarchie.

Zuerst zum Web-Handler, einem Objekt, das sämtliche HTTP-Requests vom Webserver entgegennimmt, um dann aufgrund der Informationen im URL zu entscheiden, welche Art von Aktion ausgeführt werden muss. Die Ausführung der Aktion übernimmt der entsprechende Befehl, die notwendigen Daten werden ihm vom Handler mittels Parameter übergeben.

Außerdem kann ein Web-Handler entscheiden, ob ein Befehl entweder dynamisch oder statisch ausgeführt wird. Im Falle einer statischen Ausführung wird die URL geparst und Bedingungslogik eingesetzt. Hier ist die Logik explizit und Fehler werden außerdem während des Kompilierungsvorganges erkannt. Bei der dynamischen Version wird eine Befehlsklasse mit einer Standardkomponente des URL dynamisch erstellt. Das bedeutet, dass neue Befehle hinzugefügt werden können, ohne den Web-Handler dabei abzuändern. Stattdessen muss man aber den Namen der Befehlsklasse in den URL einfügen oder alternativ dazu eine Eigenschaften-Datei erstellen. Diese bindet die Namen von Befehlsklassen an URLs. Nachteil dieser Methode ist, dass man eine weitere Datei zu verwalten und warten hat, allerdings ist die Zuordnung zwischen URLs und Befehlsklassen auf genau einen Punkt konzentriert, was das Umbenennen von Klassennamen enorm erleichtert.

Um wieder auf den Web-Handler zurückzukommen: Auf den Punkt gebracht ist er nur ein ziemlich einfaches Programm, das den auszuführenden Befehl bestimmt. Er spielt in der weiteren Verarbeitung keine Rolle mehr und alle anderen Aufgaben eines Controllers (bspw. Bestimmung der View zur Anzeige der Antwort) müssen von den Befehlen übernommen werden. Die Umsetzung beider Komponenten des Patterns erfolgt meist durch eine Klasse. Die Verwendung von Server-Pages bei einem Web-Handler ist insofern sinnlos, als dass in diesem Schritt ja keinerlei Ausgabe für den Benutzer produziert wird.

3.4.3.3 Welches Pattern soll man verwenden?

Die Antwort auf diese Frage hängt im Wesentlichen von der Komplexität der Controller-Logik ab. Ist diese eher einfach, ist man mit der Verwendung eines Page Controllers eindeutig besser beraten. Das Konzept dieses Patterns ist einfach zu verstehen und intuitiv zu implementieren. Außerdem ist der Verwaltungsaufwand im Gegensatz zu dem der Alternative relativ gering.

Natürlich bringt die Verwendung eines Front Controllers nicht nur eine höhere Komplexität mit sich, sondern auch Vorteile, die den Mehraufwand oft rechtfertigen können.
Bei der Verwendung dynamischer Befehle kann man diese hinzufügen, ohne den Code abzuändern.

Es werden bei jeder Anforderung neue Befehlsobjekte erstellt. Um Thread-Sicherheit und andere Probleme der Mulit-threaded-Programmierung braucht man sich hier also nicht mehr zu kümmern. Vorsicht geboten ist nur bei der gemeinsamen und gleichzeitigen Nutzung von Modell-Objekten – das muss auf alle Fälle unterbunden werden.

Die Vermeidung von Codeduplizierung durch Auslagerung wird auch oft als einer der Vorteile des Front Controllers gegenüber seinem Gegenstück genannt. Das ist allerdings nicht ganz korrekt, muss man ja bedenken, dass Ähnliches auch beim Page Controller durch den Einsatz eines Hilfsobjektes erreicht werden kann.

	Page Controller	Front Controller
Vorteile	✓ intuitives, einfach zu verstehendes Modell ✓ Codeduplizierung durch Verwendung von Hilfsobjekten reduzierbar	✓ nur ein Front-Controller pro Applikation ✓ Verhalten zur Laufzeit mit Decorators erweiterbar ✓ Codeduplizierung durch Auslagerung vermieden (nur bedingter Vorteil, kann auch mit Hilfsobjekten beim Page Controller erreicht werden)
Nachteile	- beschränkte Komplexität der Controllerlogik umsetzbar	- höhere Komplexität - höherer Verwaltungsaufwand

Tabelle 3: Vergleich unterschiedlicher Controller-Pattern

3.4.4 Kommunikation mit der Datenquelle

Wie schon zuvor kurz angesprochen, befindet sich keine der 3 Rollen des MVC-Patterns in der Datenschicht des Three-Tier Modells. Da die Speicherung und das Laden von Daten aus einer Datenquelle aber ein essentieller Teil so gut wie jeder Applikation ist, wird in diesem Kapitel dennoch näher auf die Möglichkeiten der Kommunikation zwischen dem MVC-Model und der Datenquelle eingegangen.

Die heutzutage am meisten verwendete Methode zur Speicherung von Applikationsdaten ist die Arbeit mit relationalen Datenbanken. Einer der ausschlaggebenden Faktoren für ihren Erfolg ist unter anderem sicher auch SQL (Structured Query Language).

Trotz der weiten Verbreitung dieser Sprache ist der Einsatz nicht immer einfach und unproblematisch. Manche Anwendungsentwickler haben ein unzureichendes Verständnis von SQL, um wirklich effiziente Abfragen und Befehle zu formulieren zu können. Außerdem ist die Einbettung von SQL in eine Programmiersprache häufig umständlich. Besser wäre es also, einen Zugriff auf die Daten über Mechanismen bereitzustellen, die zur Entwicklungssprache der Anwendung passen. Datenbankadministratoren sollte außerdem

Zugriff auf den SQL-Code gewährleistet werden, damit diese ihn optimieren und bei Änderungen der zugrunde liegenden Tabellen an einer zentralen Stelle anpassen können.

All diese Gründe rechtfertigen eine Trennung des SQL-Codes von der eigentlichen Domänenlogik durch eine Auslagerung in separate Klassen. Wie das genau geschehen soll, darauf geben mehrere Entwurfsmuster mögliche Antworten. Die Entscheidung für eines dieser Architektur-Patterns sollte wohl überlegt sein, da sie weit reichende Konsequenzen für den weiteren Entwurf der Domänenlogik mit sich zieht und ein späteres Refactoring nur schwer möglich ist. Umgekehrt hat natürlich auch eine bereits bestehende Struktur der Domänenlogik bedeutenden Einfluss auf die Entscheidung.

Eine klassische Art und Weise, die Klassen mit den SQL-Befehlen zu strukturieren, ist, dass man sich an der Tabellenstruktur der Datenbank orientiert, d.h. pro Tabelle in der DB gibt es eine zuständige Klasse, die in diesem Zusammenhang auch als *Gateway* bezeichnet wird. So ist der SQL-Code bei allfälligen Änderungen leicht auffindbar und der Rest der Anwendung bleibt frei von jeglichen Datenbankabfragen.
Es gibt zwei Methoden ein Gateway zu erstellen, die im folgenden erläutert werden sollen.

3.4.4.1 Table Data Gateway

Ein Table Data Gateway ist ein Objekt, das den gesamten SQL-Code für den Zugriff auf eine einzelne Tabelle oder Sicht enthält. Über eine einfache Schnittstelle (normalerweise mehrere Find-Methoden zum Holen von Daten aus der DB sowie Update-, Insert- und Delete-Methoden) wird die Kommunikation mit der Datenquelle hergestellt. Über verschiedenste Eingabeparameter, die in den abzusetzenden SQL-Befehl abgebildet werden, können beliebige Abfragen getätigt werden.
In der Regel wird für jede Tabelle der DB ein Table Data Gateway verwendet. Nur in sehr simplen Fällen ist es möglich, ein einziges Table Data Gateway für sämtliche Tabellen einer DB einzusetzen.
Der komplexeste Teil der Arbeit mit diesem Entwurfsmuster ist die Frage nach der Datenrückgabe. Bei vielen Sprachen ist nur ein einziger Rückgabewert möglich und selbst einfache SQL-Abfragen geben mehrere Datenreihen zurück.
Ein möglicher Lösungsansatz besteht in der Verwendung einer Zuordnungstabelle, in die die Daten aus der Ergebnismenge hineinkopiert werden. Da das aus mehreren Gründen kein guter Programmierstil ist, wäre ein Data Transfer Object eine bessere Alternative.
Konzeptionell zwar etwas unsauber, aber mit dem geringsten Aufwand verbunden ist einfach die Rückgabe des Record Sets der SQL-Abfrage. Besonders bei Umgebungen, die Record Sets in größerem Umfang einsetzen (bspw. .NET) ist das eine sehr effiziente Lösung. Das Table Data Gateway ist deshalb auch die ideale Wahl in der Kombination mit einem Table Module zur Strukturierung des Domainmodels.

3.4.4.2 Row Data Gateway

Bei einem Row Data Gateway präsentiert ein Objekt genau einen einzigen Datensatz der Datenquelle. Jeder Spalte in der Tabelle entspricht ein Feld dieses Objekts. Während alle Details des Zugriffs auf die Datenquelle hinter dieser Schnittstelle verborgen bleiben, kann mit regulären Mechanismen der Programmiersprache darauf zugegriffen werden.

Möchte man polymorphe Methoden verwenden, bspw. um unterschiedliche Abfrage-Methoden für verschiedene Datenquellen zu definieren, dann kann anstatt statischer Find-Methoden ein separates Abfrage-Objekt erstellt werden, in dem die Abfrage-operationen gekapselt sind. Jede Tabelle der relationalen DB wird dann also eine Abfrage-Klasse (Finder) und eine Gateway-Klasse zugeordnet.

3.4.4.3 Active Record

Verwendet man ein Domain Model zur Strukturierung der Domänendaten, dann kommen weitere Patterns für die Kommunikation mit der Datenquelle in Frage.
In manchen Fällen hat das Domain Model eine Struktur, die gut mit der Datenbank korrespondiert, und wo eine Domänenklasse einer Tabelle entspricht. In diesem Fall ist die Geschäftslogik wenig komplex und es kann sinnvoll sein, jedem Objekt der Domäne selbst die Verantwortung für das Laden und Speichern der eigenen Daten aus bzw. in der Datenbank zu übertragen.
Der Name des Patterns, das genau diese Struktur vorschlägt ist Active Record. Active Record ist *„Ein Objekt, das eine Zeile in einer Datenbanktabelle oder in einer Sicht umhüllt, die Datenbankzugriffe kapselt und Domänenlogik zu diesen Daten hinzufügt.“* (Fowler 2003. S:185). Der entscheidende Unterschied zu dem auf den ersten Blick recht ähnlich scheinenden Row Data Gateway ist die Domänenlogik. Während in einem Row Data Gateway ausschließlich Code für die Datenbankzugriffslogik enthalten sein sollte, ist beim Active Record sehr wohl auch Domänenlogik beteiligt. Deshalb kann bspw. im Falle von redundantem Code in Transaction Scripts in Verbindung mit einem Row Data Gateway ein Refactoring in ein Active Record durch schrittweises Hinzufügen von Domänenlogik in die Klasse sinnvoll sein.

3.4.4.4 Data Mapper

Wird die Domänenlogik immer komplexer, dann stößt das Active Record Pattern recht schnell an seine Grenzen, da die eins-zu-eins Zuordnung der Domänenklassen zu den Tabellen immer mehr verschwindet. Auch mit Gateways ist das Domain Model noch immer sehr an die Datenbankstruktur gekoppelt und Transformationen zwischen den Gateway-Feldern und den Feldern der Domänenobjekte verkomplizieren die Applikation noch mehr. In diesem Fall besteht die beste Lösung darin, das Domain Model komplett von der Datenbank durch eine Indirektions-Schicht zu isolieren. Diese Schicht übernimmt die Verantwortung für die Abbildung der Domänenobjekte und Datenbanktabellen. Dieses so genannte Data-Mapper Pattern ist das komplizierteste aller hier vorgestellte Architekturen, bietet aber den Vorteil der kompletten Isolation der beiden Schichten voneinander und auch vom Mapper selbst. Genauer gesagt bedeutet das, dass die Schichten voneinander nichts wissen und daher Änderungen in der einen Schicht die andere nicht beeinflussen. Mechanismen wie Collections und Vererbung, die in relationalen Datenbanken nicht vorhanden sind, allerdings bei der Organisation einer komplexen Geschäftslogik weiterhelfen, können nun verwendet werden und es muss bei der Strukturierung der Domänenlogik keine Rücksicht mehr auf die Datenbankstruktur genommen werden.

3.4.4.5 Welches Pattern soll man verwenden?

Die Entscheidung für ein Architektur-Pattern für die Datenquelle hängt sehr von der Struktur und Komplexität der Domänenlogik und von der Entwicklungsumgebung ab.

Bei Transaction Scripts bieten sich besonders ein Table oder Row Data Gateway an. Die Entscheidung zwischen den beiden reduziert sich auf die Frage, wie die Verarbeitung mehrerer Datenzeilen erfolgt. Einer der Vorteile von Table Data Gateways ist die Möglichkeit der Verwendung von Stored Procedures.
Eine Alternative zur Verwendung von Row Data Gateway bei Transaction Scripts ist der Einsatz eines Active Record. Das ist besonders dann überlegenswert, wenn die ersten negativen Auswirkungen von Codeduplizierung und Schwierigkeiten bei Aktualisierungen auftauchen. Eine Verlagerung des duplizierten Codes macht aus dem Row Data Gateway dann nach und nach ein Active Record.

Verwendet man zur Implementierung des Models ein Table Module, dann drängt sich in Kombination dazu ein Table Data Gateway geradezu auf. Dieses Architektur-Pattern arbeitet nämlich mit Record Sets, mit dem ein Table Module sehr effizient umgehen kann.

Ist die Domänenlogik mittels eines Domain Models strukturiert, beschränkt sich die Auswahl auf Active Record und Data Mapper. Natürlich ist auch die Verwendung eines Row bzw. Table Data Gateways prinzipiell möglich, wird allerdings nicht empfohlen.

Active Record besticht durch seine intuitive Handhabung und Einfachheit. Der Preis dafür ist, dass das Funktionieren dieses Patterns nur dann garantiert ist, wenn die Active Record Objekte den Datenbanktabellen entsprechen. Eine Voraussetzung, die ab einer bestimmten Komplexität der Geschäftslogik nicht mehr zu erfüllen ist. Durch die enge Kopplung zwischen Objektentwurf und Datenbankentwurf fällt es außerdem schwer, Code bei Weiterentwicklungen auszulagern.

Das Data Mapper Pattern merzt die Schwächen des Active Record Ansatz aus, indem es die Datenbank- von der Domänenlogikschicht komplett isoliert und sie somit unabhängig voneinander macht. Diesen Vorteil bezahlt man allerdings mit einer zusätzlichen Schicht und einer nicht gerade einfachen Implementierung. Ein Data Mapper ist daher in Kombination mit einem Domain Model dann sinnvoll, wenn die Domänenlogik sehr komplex ist und das Datenbankschema und das Objektmodell unabhängig voneinander entwickelt werden sollen.

In gewissen Konstellationen sind auch Kombinationen der vorgestellten Patterns möglich. Bspw. kann es manchmal durchaus Sinn machen, die Kommunikation eines Data Mappers mit der Datenbank über ein Table Data Gateway abzuwickeln. Auch der Einsatz von Row Data Gateways in Kombination mit Data Mappern ist denkbar.

	Table Data Gateway	Row Data Gateway	Active Record	Data Mapper
Vorteile	✓ Kapselung der Zugriffe auf die Datenquelle ✓ Einfach ✓ Effizientes Arbeiten in Umgebungen mit Record Sets ✓ Verwendung von Stored Procedures möglich	✓ Einfache Implementierung ✓ Intuitiver Ansatz	✓ Einfache Erstellung ✓ Leichte Verständlichkeit (intuitiver Ansatz)	✓ Komplette Unabhängigkeit der Domänenlogik von der Datenbankstruktur ✓ Komplexe Modelle möglich
Nachteile	- Kopplung zwischen Objektentwurf und Datenbankentwurf - Datenrückgabe	- Kopplung zwischen Objektentwurf und Datenbankentwurf	- Active-Record Objekte müssen den Datenbanktabellen entsprechen - Kopplung zwischen Objektentwurf und Datenbankentwurf	- kompliziert - zusätzliche Schicht
Empfohlenes Business-Object Pattern	- Table Module - Transaction Script	- Transaction Script	- Domain Model - Transaction Script	- Domain Model

Tabelle 4: Vergleich unterschiedlicher Pattern zur Kommunikation mit der Datenquelle

4 Frameworks mit MVC

Im folgenden Teil der Arbeit soll gezeigt werden, dass das soeben vorgestellte MVC-Pattern nicht nur ein komplexes, theoretisches Gedankengerüst ist, sondern auch praktisch in der Webprogrammierung Fuß gefasst hat und Anwendung findet.
Dies geschieht durch die exemplarische Vorstellung einiger Frameworks, die auf unterschiedlichsten (größtenteils) etablierten Programmiersprachen aufbauen. Durch kleine, einfache Codebeispiele wird deren Implementierung des MVC-Patterns neben einer theoretischen Erklärung zusätzlich veranschaulicht werden.

Außerdem wird hier auch die zweite These dieser Arbeit behandelt werden, die besagt, dass die Verwendung von Frameworks in Kombination mit MVC eine Effizienzsteigerung gegenüber der Programmierung ohne jegliche Hilfsmittel mit sich bringt. Auch hier wird ein einfach gehaltenes, in seiner Aufgabenstellung für Webapplikationen typisches Codebeispiel als Veranschaulichung herangezogen.

4.1 Was ist ein Framework?

Wörtlich übersetzt bedeutet dieser Begriff aus der Softwaretechnik (Programm-)Gerüst, Rahmen(bedingung), Skelett. Ein Framework gibt in der Regel eine Anwendungsarchitektur vor, innerhalb der Softwareprojekte organisiert und entwickelt werden können. Codebibliotheken, Skriptsprachen oder andere unterstützende Programme, die die Erstellung verschiedener Teile eines Softwareprojektes erleichtern, können in einem Framework inkludiert sein.

Der Grundgedanke dahinter ist, Programmierern und Designern die Softwareentwicklung durch die Abnahme häufig wiederkehrender Routinearbeiten zu erleichtern. Dadurch bleibt mehr Zeit übrig, um sich den Kernanforderungen der Software zu widmen.

Wie überall gibt es auch hier Nachteile. Ein verbreiteter Vorwurf an Frameworks ist, dass sie aufgeblähten Code produzieren. Das breite Angebot an konkurrierenden bzw. einander ergänzenden Frameworks erfordert außerdem auch einen nicht zu unterschätzenden Lernaufwand.

4.2 Ruby on Rails

Einen wahren Begeisterungssturm löste das Framework Ruby on Rails, kurz RoR oder einfach nur Rails genannt, unter der Community der Webentwickler aus.
Es begeisterte so sehr, dass in Folge in kürzester Zeit sogar Frameworks für Python, PHP, Java und Perl entstanden, die die Philosophie von Rails wieder aufnahmen und imitierten.

Um Missverständnisse zu vermeiden, sei vorweggenommen, dass viele Funktionalitäten von Ruby on Rails weit über die eigentliche Implementierung von MVC hinausgehen und mit dem Pattern nichts zu tun haben.

4.2.1 Entwicklung

RoR entstand bei der Umsetzung des Projektmanagement-Tools Basecamp (http://www.basecamphq.com/) von David Heinemeier Hansson, Mitarbeiter bei 37signals, einer Web-Agentur in Chicago. Erstmals veröffentlicht im Juli 2004, ist man mittlerweile bei Version 1.1.4 angelangt. 37signals hat bereits zahlreiche weitere Projekte mit Ruby on Rails umgesetzt, was immer wieder neue Features dieses Frameworks mit sich gebracht hat. Das Kernteam umfasst heute 11 Entwickler, die außerdem von einer stetig wachsenden Community bei der Erweiterung des Funktionsumfangs durch Features sowie der Verbesserung der Qualität durch das Auffinden von Fehlern und das Herausgeben von Patches unterstützt werden.

4.2.2 Ruby

Wie der Name schon verrät, arbeitet RoR mit der Programmiersprache Ruby. Im nächsten Abschnitt wird diese genauer unter die Lupe genommen.

4.2.2.1 Geschichte

Hinter dem Begriff *Ruby* verbirgt sich eine objektorientierte, interpretierte Programmiersprache, die unter der GPL (General Public License) oder der Ruby License als Open Source Produkt zur Verfügung steht.
Basierend auf Ada, Perl, Smalltalk, Python, LISP, Dylan und CLU wurde sie von dem Japaner Yukihiro Matsumoto in der ersten Hälfte der 90er Jahre entwickelt. Den ersten Release gab es 1995. Die aktuelle stabile Version 1.8.4 steht auf der offiziellen Seite (http://www.ruby-lang.org) zum Download zur Verfügung.
Der Name Ruby (engl. Wort für Rubin) ist als Anspielung auf die Sprache Perl (ursprünglich Pearl, engl. Wort für Perle) zu verstehen.
Aufgrund mangelhafter Dokumentationen in westlichen Sprachen verbreitete sich Ruby zu Beginn nur in Japan und dem ostasiatischen Raum. Dort hat es allerdings bereits einen ähnlichen Stellenwert wie seine Väter Perl und Python eingenommen und wird als stabil und praxiserprobt anerkannt.
Im westlichen Raum erntete Ruby erst um die Jahrtausendwende erste Aufmerksamkeit. Auch hier überzeugte diese Technik schnell und heute gibt es dazu umfangreiche Dokumentationen und Artikel in zahlreichen Sprachen.

4.2.2.2 Syntax

Die Ziele, denen sich Ruby verschrieben hat, sind Einfachheit, Lernbarkeit und Struktur. Das merkt man besonders, wenn man sich die Syntax der Sprache ansieht.

In Punkto Datentypen gibt es kaum Unterschiede zu anderen Programmiersprachen. Variablen werden ohne weitere Zusätze (wie zum Beispiel $ in PHP oder var in JavaScript) definiert. Die Zuweisung von Werten erfolgt wie gewohnt über das Gleichheitszeichen.
Eine besondere Bedeutung hat in Ruby die Groß- und Kleinschreibung. Variablennamen müssen klein geschrieben werden, während die Großschreibung des ersten Buchstaben eines Namens eine Konstante kennzeichnet.
Codebeispiel:

```
variablenname  = “variablenwert“
Konstantenname = “Konstantenwert“
```

Das Ende von Anweisungen kann wahlweise durch Semikolons oder einfach nur Zeilenumbrüche markiert werden. Parameter von Methoden müssen nicht in Klammer gesetzt werden und die Angabe des Rückgabewertes ist ebenfalls optional. Stattdessen wird das Ergebnis der letzten Anweisung automatisch zurückgegeben. Geschweifte Klammern, wie man sie bei C- und Java-basierten Sprachen zur Abgrenzung von Funktionen gewohnt ist, können optional durch Schlüsselwörter wie do und end ersetzt werden. Statt dem üblichen Inkrement ++ bzw. dem Dekrement - - gibt es hier die Methoden upto() bzw. downto(). Mit ihnen sind simple while-Schleifen mit Einserschritten einfach zu realisieren.

Codebeispiel:

```
3.times do
puts i
i+=2
end
```

In der for-Schleife, die 3 mal durchlaufen wird, wird der Wert der Variablen i jedes mal um 2 erhöht.

4.2.3 Aufbau

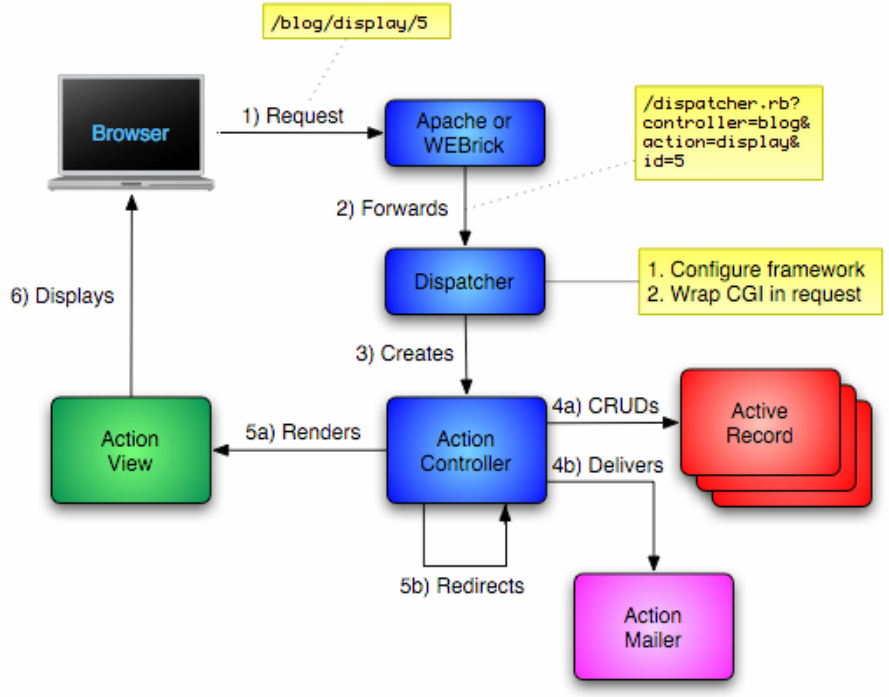


Abbildung 6: Request-Zyklus bei RoR (http://www.rubyonrails.org/images/request_cycle.png, 01.05.2006)

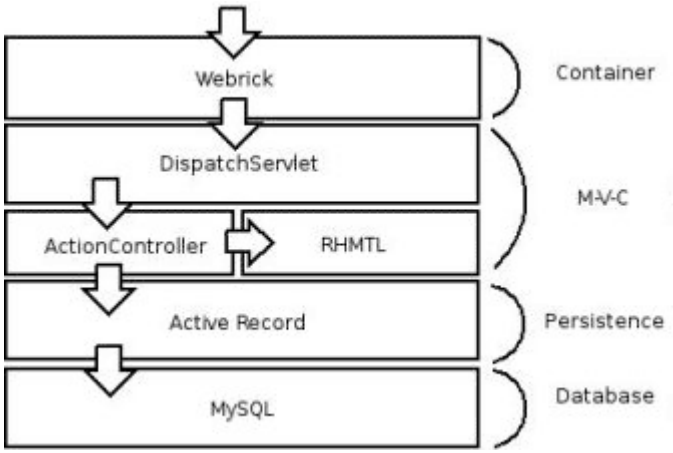


Abbildung 7: RoR Stack (http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE, 10.05.2006)

4.2.3.1 MVC

4.2.3.1.1 Model

In RoR besteht das Model aus Klassen, die die Tabellen eines RDBMS repräsentieren. Als Pattern für die Kommunikation mit der Datenquelle wird Active Record verwendet. Normalerweise genügt es, die eigenen Models als Subklassen der `ActiveRecord::Base` Klasse zu definieren und – sofern man sich bei der Namensgebung an die Konventionen gehalten hat – weiß das Framework genau, welche Tabelle zu verwenden ist und welche Spalten diese Tabelle umfasst.
Innerhalb der Models werden außerdem spezielle Relationen zwischen Tabellen mit objekt-relationalen Mapping-Befehlen festgelegt.
Gibt es beispielsweise in der Klasse namens Category die Definition `has_many :recipes`, wird damit eine 1:n-Beziehung zwischen der Tabelle Categories und Recipes festgelegt. Hat man eine Instanz von Category namens `mycategory`, so wird über den Aufruf `mycategory.recipes` ein Array mit allen Recipe-Objekten zurückgegeben, deren `category_id` mit `mycategory.id` übereinstimmt.
Ebenfalls im Model spezifiziert werden Datenvalidierungs-Handler (z.B. `validates_uniqueness_of :checksum`) und andere änderungsbezogene Handler (z.B. `after_destroy :remove_file`, `before_update :update_related_details`).

4.2.3.1.2 View

Für die Präsentationsschicht ist die Klasse ActionView, die selbst wieder ein Bestandteil der ActionPack Bibliothek ist, zuständig.
ActionView Templates können auf 3 verschiedene Methoden implementiert werden:

Die erste ist die Verwendung von Embedded Ruby (eRb). Auf diese Art umgesetzte Views sind durch die Dateiendung *.rhtml* gekennzeichnet. Hierbei werden ähnlich wie in ASP, JSP oder PHP kleine Ruby-Codefragmente in das HTML-Dokument eingebunden und von einem Interpreter zur Laufzeit ausgeführt und durch HTML ersetzt. Außerdem ist die Möglichkeit vorgesehen, mit Hilfsobjekten zu arbeiten, um die View einfach zu halten. Eine Menge Hilfsobjekte werden gleich mit Rails mitgeliefert, es gibt aber auch bereits einige von Communitymitgliedern oder die Möglichkeit, selbst welche zu erstellen.

Ist das Template vom Dateityp *.rxml*, so deutet das darauf hin, dass die Builder::XmlMarkup – Bibliothek verwendet wurde. Builder stellt eine programmatische Alternative zu Embedded Ruby dar und ist besonders bei der Erstellung von XML-Dokumenten sinnvoll.

Templates, die mit dem JavaScriptGenerator arbeiten, haben die Endung *.rjs*.
Anders als bei konventionellen Templates, die zur Darstellung von Resultaten einer be-

stimmten Aktion dienen, generieren diese Anweisungen, wie eine bereits gerenderte Seite verändert werden soll. Diese Art der Templates wird in Verbindung mit AJAX (Asynchronous JavaScript and XML) verwendet.

4.2.3.1.3 Controller

Rails Dispatch Servlet ist ein Musterbeispiel für die Umsetzung des Front Controller Patterns. Es nimmt sämtliche HTTP-Requests vom Server entgegen, parst die URL und entnimmt ihr genügend Informationen, um entscheiden zu können, welche Art von Aktion als nächstes angestoßen werden soll. Im Falle von Rails sind Aktionen Klassen, die von der ActionController Klasse erben, die wie ActionView Bestandteil der ActionPack Bibliothek ist.
Aktionen bilden eine Art Brücke zwischen dem Front Controller und dem Model. Rein technisch gesehen sind Aktionen Teil des Controllers im MVC-Pattern, da sie auf vom Benutzer angestoßene Requests reagieren. In der Praxis ist es allerdings besonders bei kleineren Applikationen oft so, dass Businesslogik in diese Klassen eingefügt wird, wodurch sie wieder als Teil des Models angesehen werden können. Best Practice empfiehlt, dass man die Domänenlogik vom Controller fernhält und diese in eigenen domänen-spezifischen Klassen platziert.

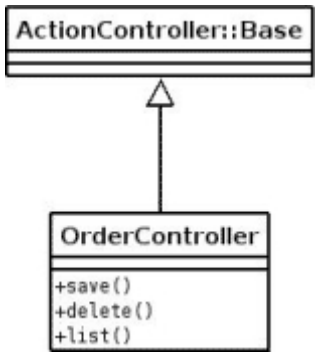


Abbildung 8: Rails Action-Controller Hierarchie (http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE, 10.05.2006)

Ein wesentliches Merkmal, durch das sich Rails von anderen Frameworks unterscheidet, ist die Art, wie dieses Mapping von HTTP-Requests auf bestimmte Aktionen implementiert ist. Während man in anderen Frameworks mittels XML-Konfigurationsdateien händisch festlegen muss, welche Aktion für welchen Request verantwortlich ist, verfolgt Rails den Ansatz der Namenskonventionen:
Ein Aufruf in der Form `recipe/list` beispielsweise ruft die Methode `list` des Controllers namens `recipe_controller` auf, das Ergebnis wird mit dem Template `/app/views/recipe/list.rhtml` (im Falle von eRb, sonst andere Dateiendung) dargestellt. Über den eingebauten Routing-Mechanismus hat man jedoch die Möglichkeit, von den Konventionen abzuweichen und den Rails-Request nach den eigenen Bedürfnissen anzupassen.
Im Controller kann außerdem der Scaffolding Befehl (siehe 4.2.4.3) durchgeführt werden, mit welchem rasch der Großteil der Logik und die Views zur Durchführung der Standardoperationen CRUD (Create Read Update Delete) erstellt werden.

4.2.3.2 Andere Module

Action Pack ist eine Bibliothek, die sich um die Request-Behandlung und die Response-Ausgabe kümmert. Der Response einer Clientanfrage wird dabei in einen Controller-Teil (Action Controller) und einen View-Teil (Action View) getrennt.

Action Mailer ist ein Modul, das es erlaubt, E-Mails von Rails-Anwendungen einfach zu versenden und zu empfangen.

Action Web Service erlaubt das Erstellen von Web Service APIs (Application Programming Interfaces) mit Rails, ohne sich zu sehr mit den Protokolldetails beschäftigen zu müssen. Dieses Modul unterstützt sowohl das SOAP (Simple Object Access Protocol) als auch das XML-RPC (Extensible Markup Language-Remote Procedure Call) Protokoll sowie eine dynamische WSDL (Web Services Description Language) Generierung für APIs.

Active Support ist eine Sammlung verschiedenster Klassen und Erweiterungen, die für Rails als nützlich empfunden wurden.

AJAX on Rails ist ein Zusammenspiel der AJAX-Technologie mit dem Rails Framework. Rails stellt mehrere Hilfsobjekte zur Verfügung, die das Implementieren von AJAX-Applikationen vereinfachen.

4.2.4 Erfolgsfaktoren

Der explosionsartige Erfolg von Ruby wird bereits heute als Phänomen bezeichnet. Doch was steckt dahinter? Was unterscheidet RoR von den zahlreichen anderen Webapplikations-Frameworks am Markt?

4.2.4.1 Philosophie

„*Convention over Configuration*“ sowie “*Don’t repeat yourself*” sind die zwei wesentlichen Leitsätze, die bei der Entwicklung dieses Frameworks beherzigt wurden und noch immer werden.

Mit dem ersten Motto will sich RoR als direktes Gegenstück zu all den schwergewichtigen Frameworks positionieren. Der Gedanke hinter diesem Framework ist, die Konfiguration auf ein absolutes Minimum zu reduzieren. So existiert beispielsweise nur eine Konfigurationsdatei für die Festlegung der Datenbankparameter. Alle anderen Zuordnungen passieren, solange man sich an die Konventionen hält, voll automatisch.
Einige Beispiele zur Verdeutlichung:

- ✓ Zuordnung von Modellen zu Tabellen in der Datenbank:
  In anderen Frameworks oft mit komplizierten XML-Dateien umgesetzt, muss man sich hier bei Einhaltung der folgenden Regel nicht mehr darum kümmern:
  Die Namen von Klassen sind immer Singular, die der dazugehörigen Tabelle Plural. → Klassenname: `weblog`, Tabellenname: `weblogs`
  Das einzige, wo Vorsicht geboten ist, ist bei irregulären Formen der Mehrzahl wie bspw. sheep, mouse, man, etc. Diese werden nur sporadisch unterstützt.
  Der Vorteil dieser Technik besteht darin, dass bei der Erweiterung der Applikation keine zusätzliche, manuelle Änderung der XML-Zuordnungsdatei notwendig ist.

- ✓ Aufruf des verantwortlichen Controllers
  Auch die Zuordnung des richtigen Controllers erfolgt automatisch durch die Benennung. → Klassenname: `weblog`, Controllername: `weblog_controller`
- ✓ Aufrufen des passenden Templates
  Automatisiert durch dieselbe Namensgebung wie die entsprechende Methode im Controller wird das passende Template aufgerufen.
  → Methode des Controllers: `show()`, Position und Name der View: `/app/views/weblog/show.html`

Will oder kann man diese Konventionen nicht einhalten, so ist es natürlich möglich, diese durch spezielle eigene Konfiguration zu erweitern.

David Heinemeier Hansson meinte bei der OSCON 2005: "*Flexibility is overrated. That's our charge from Ruby on Rails. [...] You are changing flexibility for velocity in development. […] Rails trades flexibility and gets a whole lot in return. In other words: Constraints are liberating.* " (Secrets behind Ruby on Rails, David Heinemeier Hansson, OSCON 2005)

Das zweite Motto "*Don't repeat yourself*" bezieht sich auf die Tatsache, dass man bei vielen Webprojekten immer wieder das Gleiche tut. RoR will verhindern, dass man Code, den man in Prinzip in jeder Webanwendung – wenn auch leicht abgewandelt – immer wieder braucht (wie zum Beispiel bestimmte Sicherheitsabfragen, etc.), ständig neu programmieren muss.

4.2.4.2 Marketing / Support

Ganz gleichgültig, wie man zu RoR stehen mag – man muss zugeben, dass das Team rund um David Heinemeier Hansson beachtliche Arbeit in Sachen Marketing leistet. Interessiert man sich für Webentwicklung, so kommt man an Ruby on Rails praktisch nicht mehr vorbei. Will man sich dann wirklich mehr in das Thema einarbeiten, findet man eine (für manche sogar zu große) Fülle an Tutorials, Manuals, Dokumentationen, Foren, Mailinglisten, Blogs, etc. Doch auch außerhalb des Mediums Internet tut sich einiges: Auftritte der Kernteammitglieder bei einschlägigen Konferenzen, Veranstaltung von Workshops, Herausgabe von Büchern, Interviews und vieles mehr sorgen für eine hohe Präsenz dieses Frameworks.

4.2.4.3 Scaffolding

Die Scaffold-Methode erzeugt ein einfaches Interface, in dem die Daten aus einer entsprechenden Tabelle in der DB dargestellt werden.

```
class RecipeController < ApplicationController
   scaffold :recipe
end
```

Dieser Codeabschnitt erzeugt eine Auflistung aller Datensätze aus der Tabelle `recipes`, sofern es auch ein dazu passendes recipe-Model gibt, sowie Templates zum erstellen, lesen, anschauen und löschen von Einträgen (=CRUD).
Beim Erstellen oder Editieren eines Datensatzes wird die lästige Arbeit des Formularerstellens automatisch erledigt und es werden auch Sonderformen wie Kalenderdaten intelligent formatiert und angezeigt.

Besonders dann, wenn es hauptsächlich darum geht, ein einfaches Interface für die Daten in einer Datenbank bereitzustellen, kann man von dieser Funktion profitieren. Meistens nur für das Prototyping verwendet, kann es in späteren Entwicklungsphasen durch eigenen Code ergänzt werden.

4.2.5 Barrieren

4.2.5.1 Programmiersprache Ruby

Über die Zukunft von Ruby gibt es widersprüchliche Ansichten. Die einen meinen, dass Ruby on Rails zum trojanischen Pferd werden wird, das Skripting-Sprachen den Zugang zur Welt der Business-Software-Entwicklung ermöglicht. Die anderen wiederum sind überzeugt, dass Ruby durch konkurrierende Mainstream-Sprachen in den Untergrund verdrängt wird. Sollte letzteres zutreffen, so kann sich die Verwendung von Ruby für RoR sogar als Nachteil erweisen. Zahlreiche Frameworks in verschiedensten anderen Sprachen (PHP, Python, Java, Perl) haben bewiesen, dass Ruby nicht der einzige mögliche Weg ist, um das Rails-Design zu verwirklichen.
In dem Markt, in dem sich RoR behaupten möchte, gibt es bereits sehr etablierte Programmiersprachen. Man fragt sich nun natürlich (zu Recht), ob RoR gegenüber den anderen auf Rails basierenden Frameworks einen ausreichenden Mehrwert bietet, der den Aufwand des Erlernens der Ruby-Sprache wettmacht.

4.2.5.2 (Zu) viel „Automagic“

RoR bietet jede Menge clevere Features und eine Menge an Dingen passiert einfach automatisch. Vielen Entwicklern bleibt aber verschlossen, was dabei wirklich im Hintergrund abläuft, was der in diesem Zusammenhang oft verwendete Begriff „Automagic“ beweist. Genau das ist auch der Grund, warum manche Programmierer davor zurückschrecken, dieses Framework zu verwenden – denn dadurch, dass sie die Vorgänge nicht mehr vollständig verstehen, verlieren sie auch ein gewisses Maß an Kontrolle darüber.

4.2.5.3 Editoren / IDEs

Auf der offiziellen RoR-Webseite wird TextMate als Editor empfohlen – den gibt es allerdings nur für Mac OS X. Für alle anderen Betriebssysteme wird man auf Radrails verwiesen, ein IDE (Integrated Development Environment), das auf der Eclipse RCP (Rich Client Platform) aufbaut und das momentan in der Version 0.6.2 unter der CPL (Common Public License) frei verfügbar ist. Für manche andere Editoren gibt es Plugins. Die Tatsache, dass die Auswahl an Editoren und IDEs im Vergleich mit konkurrierenden Frameworks aber eher gering und noch in der Anfangsphase ihrer Entwicklung steht, lässt sich nicht verbergen.

4.2.5.4 Hosting

Da Ruby als Programmiersprache und Ruby on Rails als Framework doch noch relativ jung sind und sich besonders im deutschsprachigen Raum (noch) nicht durchgesetzt haben, gibt es ein dementsprechend geringes Angebot an ISPs (Internet Service Provider), die Ruby on Rails auch unterstützen. Zum Zeitpunkt dieser Arbeit beläuft sich diese Zahl für den gesamten deutschsprachigen Raum (Deutschland Österreich, Schweiz) auf 5 Anbieter.

Im Gegensatz dazu wird PHP von ca. 99% der ISPs unterstützt.
Man kann nun dagegenhalten, dass Java von den Providern auch kaum gehostet wird. Was man dabei allerdings berücksichtigen muss, ist, dass Java ein Publikum anspricht, das in den meisten Fällen eigene Server für das Hosting hat
Ruby on Rails spricht allerdings die „Mittelschicht“ an: Webseiten, die eine begrenzte Skalierbarkeit haben, aber trotzdem eine ansehnliche Zahl an Visits. Die Anbieter dieser Seiten nutzen meist externes Hosting – gibt es hier kein zufrieden stellendes Angebot, ist die Verwendung von RoR schon einmal um einiges unattraktiver.

4.2.5.5 Fanatismus der Community

Rund um RoR hat sich bereits eine ansehnliche Community gebildet, die auch bereitwillig Support bei Fragen anbietet und neue Anhänger mit offenen Armen aufnimmt.
Doch wehe dem, der es wagt, Skepsis oder gar Kritik an den Tag zu legen. Tut man das, wird man meist recht unfreundlich als Unwissender oder „noch nicht reif für dieses Framework“ abgestempelt. Die Anhänger von RoR sind derart von dessen Vollkommenheit überzeugt, dass jegliche sachliche Diskussion über Kritikpunkte von vorne herein unterbunden wird.

4.2.5.6 Monopolstellung von RoR

Will man mit Ruby unter Verwendung des MVC-Patterns eine Webapplikation schreiben, so läuft es im Endeffekt auf eine einzige Möglichkeit hinaus – die Verwendung von RoR.
Im Gegensatz dazu werden bspw. für Java ein Dutzend unterschiedlicher Webframeworks angeboten. Erntet es gerade dafür von manchen Kritik, muss man auch die positiven Effekte betrachten: Durch das große Angebot entsteht eine gewisse Wettbewerbssituation, die jedes Produkt dazu zwingt, sich immer wieder zu verbessern, um nicht durch die natürliche Selektion im Untergrund zu verschwinden. Jedes neu aufkommende Framework baut wiederum auf die Stärken seiner Vorgänger auf, während es zumindest ein paar derer Fehler ausmerzt. Durch diese dynamische Entwicklungssituation verbessert sich die Qualität der Produkte schrittweise und die Anpassung an neu aufkommende Technologien bleibt gewährleistet.
Für Ruby gibt es zwar noch andere Frameworks (z.B. http://code.whytheluckystiff.net/camping/, http://nitro.rubyforge.org) – die Tatsache, dass sie die meisten, die mit RoR arbeiten, aber nicht einmal kennen, zeigt, dass diese nur geringe Relevanz haben und sich nicht mit RoR messen können.
Bei der Monopolstellung von RoR besteht eine größere Gefahr, dass diese stetige Weiterentwicklung irgendwann aus Mangel an Konkurrenz oder neuen Ideen einfriert.

4.2.5.7 Unklarheit über Skalierbarkeit und Unternehmenseinsatz

Schaut man sich nach Referenzprojekten für RoR um, so wird man bald feststellen, das ein Großteil dieser von den Entwicklern des Frameworks selbst stammt. Es gibt zwar auch zahlreiche von anderen Agenturen oder Privatpersonen. Die verfügen allerdings nicht über die notwendige Größe und Komplexität, um seriöse Aussagen über Skalierbarkeit und den produktiven Einsatz im Unternehmen zu machen.
Hier steht man vor dem typischen Henne-und-Ei Problem: Kein Unternehmen möchte als erstes den Sprung ins kalte Wasser wagen und mit einem noch relativ neuen, in der Unter-

nehmenspraxis kaum verbreiteten Framework herumexperimentieren. Zuerst müssen Erfolgsgeschichten mit RoR geschrieben werden – auch außerhalb der Entwicklungsfirma 37signals. Genau das ist bis dato aber noch nicht passiert.

4.2.5.8 Misstrauen gegenüber Open Source Produkten

Unter Benutzern und Entscheidungsträgern ohne technischen Hintergrund herrscht prinzipiell ein großes Misstrauen gegenüber Open Source Software, frei nach dem Motto „Was nichts kostet, ist nichts wert."
Die RoR-Community tut allerdings ihr Bestes, um dem entgegenzuwirken: Durch organisierte und professionelle Netzwerke wird Support gewährleistet und auch die Webseite und die Dokumentation sind von hoher Qualität.

4.2.6 Hard Facts

Entwickler:	erstmals herausgegeben von David Heinemeier Hansson (Dänemark) weiterentwickelt durch ein 11-köpfiges Kernteam und hunderte Mitglieder der Open Source Community
Erstmals erschienen:	Juli 2004
Letzte Version:	1.1.2, herausgegeben am 10.04.2006
Offizielle Webseite:	http://www.rubyonrails.org/
Lizenz:	MIT (Massachusetts Institute of Technology) -Lizenz
Motto:	”Don’t repeat yourself“, “Convention over Configuration.”
Referenzprojekte (kleine Auswahl):	✓ Instiki wiki software ✓ Typo weblog software ✓ Basecamp (http://www.basecamphq.com/) ✓ Campfire (http://www.campfirenow.com/) ✓ Backpack (http://www.backpackit.com/) ✓ Ta-Da List (http://www.tadalist.com/) ✓ BubbleShare (http://www.bubbleshare.com/) ✓ 43 things (http://www.43things.com/) ✓ Odeo (http://www.odeo.com/) ✓ Strongspace (http://www.strongspace.com/) ✓ Typo (http://www.typosphere.org/)
Programmiersprache:	Ruby (momentan empfohlene Version 1.8.4)

Installation:	✓ All-in-one Pakete: Instant Rails (für Windows) Locomotive (für Mac OS X) ✓ Manuelle Installation der einzelnen Bestandteile: Webserver, Datenbank, Ruby, RubyGems, Rails
Dokumentation/Support/ Community:	✓ Dokumentation der API (http://api.rubyonrails.org/), des Ruby Kerns (http://corelib.rubyonrails.org/) und der Ruby Standard Lib (http://stdlib.rubyonrails.org/) ✓ Bücher („Agile Web Development with Rails“, „Rails Recipes“) ✓ Tutorials ✓ Manuals (http://manuals.rubyonrails.com/) ✓ Freie Einsicht in den Source Code ✓ Verwaltung von Bugs und Patches mittels dem Projektmanagement-Tools Trac ✓ Mailingliste ✓ Foren (http://www.rubyonrailsforum.com/, http://www.ruby-forum.com/forum/3) ✓ Rails Wiki (http://wiki.rubyonrails.org/rails) ✓ Ruby on Rails Podcast (http://podcast.rubyonrails.com/) ✓ IRC (Internet Relay Chat) ✓ div. RSS-Feeds ✓ Weblogs ✓ Konferenzen ✓ Workshops
Plugins:	Zahlreiche, separat dokumentierte Plugins zu den Bereichen: ✓ Assets (images, css, js) ✓ Authentifikation und Sicherheit ✓ Controller Erweiterung ✓ Internationalisierung ✓ Model Erweiterung ✓ Rails Engines ✓ Sonstige Erweiterungen des Frameworks ✓ Statistiken und Logs ✓ Suchen und Abfragen ✓ Testen ✓ View Erweiterung

Editoren / IDEs:	✓ TextMate (für Mac OS X) ✓ RadRails (betriebssystem-unabhängig, aktuelle Version 0.6.2) ✓ Ruby Editor Plugins bei manchen Editoren verfügbar (z.B. BBEdit, jEdit, VIM,...)
Implementierte Architekturmuster:	✓ MVC ✓ Active Record für den Zugriff auf die Datenquelle ✓ Front Controller zur Steuerung von HTTP-Reuqests ✓ Template View für die Anzeige der Daten
OS (Operating System)	Betriebssystem-unabhängig, ein Unix-basiertes OS wird allerdings empfohlen
Webserverunterstützung:	✓ WEBrick (im Ruby-Paket enthalten) ✓ Apache ✓ Lighttpd mit FastCGI ✓ Jeder andere Webserver mit CGI- oder Fast CGI Unterstützung
Datenbankunterstützung:	✓ DB2 ✓ Firebird ✓ MySQL ✓ Openbase ✓ Oracle ✓ PostgreSQL ✓ SQLite ✓ SQL Server ✓ Sybase
Unterstützung Ausgabeformate/Templates:	✓ HTML ✓ XML ✓ JavaScript ✓ Binärdaten ✓ RHTML ✓ RXML ✓ RJS
Hosting:	✓ TextDrive (offizieller Host von Ruby on Rails) ✓ Im deutschsprachigen Raum: Net-Publics GbR Fritsch Hosting GPcom Media

	MIVITEC GmbH (MIRSKY.MICRO-SYSTEMS) WebJanssen ISP ltd & Co KG (vgl. http://www.webhostlist.de/host/data/compare_webhosting.php, 16.04.2006)

Tabelle 5: Fakten zu Ruby on Rails

4.2.7 Ausblick in die Zukunft

Es gibt mehrere Indikatoren, die dafür sprechen, dass der Hype rund um dieses Framework wohl nicht so schnell abflachen wird.

- ✓ Nach der weltweit ersten offiziellen RoR-Konferenz „Canada on Rails“, die von 13. - 14. April 2006 in Vancouver/Kanada stattfand, sind weitere geplant.

  Von 22. - 25. Juni findet in Chicago/USA die erste offizielle, internationale Konferenz zu diesem Framework statt. In der RoR-Community durchwegs bekannte Größen wie Martin Fowler, Paul Graham, David Heinemeier Hansson und Dave Thomas konnten bereits als Redner gewonnen werden.
  Nachdem die 400 Karten für dieses Ereignis innerhalb weniger als einer Woche ausverkauft waren und es noch eine lange Warteliste von Interessenten gab, bot man weitere 150 Karten an. Dieses Sortiment war innerhalb von knapp 24 Stunden vergeben.
  Auf der offiziellen Webseite zu dieser Konferenz (http://railsconf.org/) kann man die weitere Entwicklung verfolgen.

  Unter http://italyonrails.com/ wird bereits die erste Konferenz zu Ruby on Rails in Europa angekündigt. Sie wird von derselben Organisation (OSEVENTS) veranstaltet wie bereits Canada on Rails und findet in Rom/Italien im Herbst dieses Jahres statt. Genauere Informationen sind zum Zeitpunkt dieser Arbeit noch nicht verfügbar.

- ✓ Auf der OSCON (O’Reilly Open Source Convention) im August 2005 veröffentliche David Heinemeier Hansson erstmals Zahlen zu RoR.
  Ziemlich genau ein Jahr nach der Herausgabe der ersten Version konnte das Framework 100.000 Downloads verzeichnen und laut Schätzungen verdienten sich bereits zu diesem Zeitpunkt 250 Programmierer aus 36 Ländern ihr Geld durch die professionelle Arbeit damit.
  Nur 6 Monate nach der Konferenz fand der 300.000ste Download von RoR statt und die Zahl der professionellen Programmierer stieg auf mehr als das doppelte der damals genannten Zahl, und zwar 550 (aus 50 Ländern) an.
  Februar dieses Jahres betrug die Anzahl der am IRC-Channel teilnehmenden Entwickler 400, die Zahl der Meldungen über Message Lists und Foren steigt weiterhin kontinuierlich.
  (vgl. http://www.itconversations.com/shows/detail658.html, „Secrets behind Ruby on Rails“ und http://weblog.rubyonrails.org/articles/category/horizon, “Secrets behind Ruby on Rails: The Numbers”, 16.04.2006)

- ✓ Die verfügbare Literatur zu diesem Thema explodiert geradezu in den letzten Monaten. Kein Wunder, wurden doch von *Agile Web Development with Rail*s (erschienen im September 2005, manche bezeichnen es als die Bibel zu RoR) innerhalb des ersten halben Jahres 25.000 Exemplare verkauft.
    - o Die Beta-Version des Buches *Rails Recipes* von Chad Fowler ist seit Februar 2006 verfügbar.
    - o O'Reilly brachte kurz davor, im Januar 2006 den Rough Cut von *Ruby on Rails: Up and Running* sowie die des *Ruby Cookbook* heraus. Die Printversionen dieser Werke werden ab Juli bzw. August im Handel erhältlich sein.
    - o Im Mai 2006 ist das Buch *Ruby for Rails – Ruby techniques for Rails developers* am Markt erschienen.
    - o Mit dem Werk *Rapid Web Development mit Ruby on Rails* publizierten die Autoren Ralf Wirdemann and Thomas Baustert im Januar 2006 das erste deutschsprachige Buch zu diesem Thema.

  Mit der Veröffentlichung von insgesamt 6 Büchern innerhalb eines knappen Jahres und einigen weiteren in Entwicklung ist Ruby auf dem Weg, eines der am besten durch Literatur dokumentierten Web Frameworks am Markt zu werden.
- ✓ Während in den USA vom Pragmatic Studio (http://studio.pragprog.com/rails/) schon länger Kurse im ganzen Land angeboten werden, findet diese Geschäftsidee nun auch in Europa Anhänger.
    - o Geoffrey Grosenbach, Verantwortlicher des Rails podcast veranstaltet am 30. März einen eintägigen Workshop zu RoR in London.
    - o Von 10. - 14. April findet in London unter der Leitung von Chad Fowler, Autor des kürzlich erschienen Buches *Rails Recipes*, ein Workshop statt.
    - o Genau zur selben Zeit veranstaltet Marcel Molina, ein Mitglied des RoR-Kernteams einen fünftägigen Workshop in Frankfurt.
- ✓ Auch große Organisationen scheinen RoR mittlerweile ernst zu nehmen. Immerhin gibt es auf der Webseite von Oracle ein Tutorial, das die Frage, wie und warum man RoR mit Oracle als Datenbank benutzen sollte, beantwortet.
  Immer mehr Editoren (z.B. BBEdit, jEdit) verfügen über Plugins für Ruby.
- ✓ Im April 2006 schaffte Ruby den lang erwarteten Sprung in die Top 20 Programmiersprachen des Tiobe Programming Community Index.
  Mit einem Plus von 0,20% seit April 2005 liegt Ruby nun mit einem Rating von 0,493% auf Platz 18.
- ✓ Die offizielle Webseite von Ruby on Rails hält momentan bei einem Google Page Rank von 8 (von 10).

Ob all diese engagierten Maßnahmen und viel versprechenden Entwicklungen im Endeffekt ausreichend sein werden, um sich gegen etablierte Standards behaupten zu können, bleibe dahingestellt und kann aus heutiger Sicht auch nicht zuverlässig vorausgesagt werden. Im Web findet man zahlreiche Blogs, die sich unter anderem diesem Thema widmen.

4.2.8 Implementierung

Um die Implementierung des MVC-Entwurfsmusters in RoR weiter zu verdeutlichen, wird eine kleine und einfache, in ihrer Aufgabenstellung aber dennoch typische Applikation mit diesem Framework umgesetzt werden.

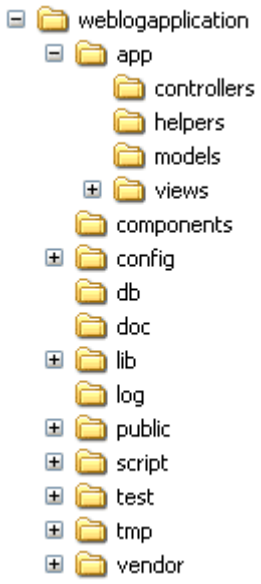
Es handelt sich dabei um eine Benutzerverwaltung, die heutzutage bereits in nahezu jedem CMS im Web zu finden ist. Benutzer des Systems können damit aufgelistet, erstellt, editiert und gelöscht werden. Die Zuordnung zu Benutzergruppen erleichtert in einem weiteren Schritt die Rechteverwaltung.

Beim Erstellen der Applikation wurde besonderer Wert auf die technische Komponente gelegt. Aus Gründen der Einfachheit und Übersichtlichkeit wurde deshalb auf umfangreiche Formatierungsarbeiten verzichtet. Das Ziel dieses Programms ist vielmehr eine Demonstration einiger typischer Features dieses Frameworks und erhebt keinen Anspruch auf Vollkommenheit und praktischen Einsatz.

Im Folgenden wird die Vorgehensweise grob skizziert. Der Quelltext steht im Anhang und auf der beiliegenden CD-ROM zur Verfügung.

✓ **Erstellen einer leeren Webapplikation**

Rails stellt einige Hilfsskripte zur Verfügung, die Standardarbeiten in der Entwicklung von Webapplikationen übernehmen. Das Rails-Helperskript erledigt die Erstellung der Ordnerstruktur und der Basisdateien für eine Applikation.
Man öffnet ein Command Window, navigiert zu dem Ordner, in dem man die Applikation erstellen möchte und führt den Befehl `rails <name_der_applikation>` aus.
Das Resultat ist rechts zu sehen:
Die meiste Entwicklungsarbeit passiert in den Unterordnern von *app*: Die Benennung dieser Ordner ist recht intuitiv und muss, wenn man das MVC-Modell kennt nicht weiter erläutert werden. Im *helpers*-Ordner werden diverse Hilfsklassen für die 3 Komponenten dieses Design Patterns ausgelagert. Das dient dazu, den Code in den Hauptteilen überschaubar und ordentlich zu halten.



✓ **Erstellen der Datenbank**

Nach dem Erstellen der Datenbank und der Tabellen muss man Rails die Zugriffsdaten auf diese bekannt geben. Das passiert in einem der wenigen Konfigurationsdateien `weblogapplication/config/database.yml`. Änderungen in dieser Datei werden erst nach einem Neustart des Servers wahrgenommen.

✓ **Generieren des Models und des Controllers**

Der Befehl `ruby script\generate model User` generiert unter anderem die Datei *user.rb*, die ein Skelett der User Klasse enthält. Dieses Model wird allein durch die Einhaltung der einfachen Namenskonvention „Tabelle → Plural, Model →Singular“, der Users-Tabelle zugeordnet und stellt Methoden für den Zugriff auf Tabelleninhalte sowie ein Attribut für jede Spalte in der Tabelle zur Verfügung.
Ähnlich wie beim Model erfolgt die Erstellung des Controllers über den Befehl `ruby`

`script\generate controller User`.
Durch die Zeile `scaffold :User` wird eine einfache Ansicht für sämtliche CRUD-Operationen für die Tabelle Users erstellt.

✓ **Manueller Feinschliff**

Natürlich kann ein Framework nicht die gesamte Arbeit der Webentwicklung abnehmen. Mittels Scaffolding wird allerdings eine gute Basis für die Einbindung applikationsspezifischer Details geschaffen. Das vorgegebene Gerüst kann nun Schritt für Schritt durch ausprogrammierte Methoden im Controller ersetzt und erweitert werden. Wichtig ist, dass es zu jeder Methode im Controller, die einen Output erzeugt, auch ein gleichnamiges Template gibt.
In diesen *.rhtml*-Dateien kann die Darstellung der Daten beliebig festgelegt werden. Durch das Einbinden sogenannter Partials kann mehrmals verwendeter Darstellungscode in eigene Dateien ausgelagert und Duplizierung somit vermieden werden. Rails Helpers nehmen das Schreiben von Standard-HTML Code ab und sorgen für mehr Übersichtlichkeit und Effizienz.
Im Model muss die Geschäftslogik eingebunden werden, in unserem konkreten Fall sind das die Nachbildung der Tabellenbeziehungen in der relationalen DB (1:n Beziehung zwischen *usergroups* und *user* über den Fremdschlüssel *usergroup_id*), die Formularvalidierung (Username, Passwort und die Zuordnung zu einer Benutzergruppe sind verpflichtend) sowie die Wahrung der DB-Integrität beim Löschen eines Datensatzes (beim Löschen einer Benutzergruppe müssen auch alle Mitglieder dieser Gruppe mitgelöscht/in eine andere Gruppe verlegt werden). Für all diese Aufgaben bietet Rails Funktionen, mit denen diese Probleme innerhalb weniger Codezeilen gelöst werden.

✓ **Fazit**

In der untenstehenden Tabelle sind die während der Entwicklung auffallenden Vor- und Nachteile dieses Frameworks übersichtlich aufgelistet. Dabei ist zu berücksichtigen, dass es sich hier um die Sicht eines Einsteigers handelt.

Vorteile	Nachteile
✓ gute Tutorials für den Start ✓ brauchbare Dokumentation ✓ schnelles erstes Ergebnis durch Scaffolding ✓ Funktionen zur Lösung von Standardaufgaben ✓ gut lesbarer Code	- Verlust der vollkommenen Kontrolle - genaues Verständnis über Vorgänge im Hintergrund fehlt - manche Funktionen ausgabespezifisch (bspw. `error_messages_for`) - viele Dateien, Ordnerstruktur anfangs verwirrend

Zusammenfassend bleibt bei diesem Framework zu sagen, dass die Einstiegshürde durch qualitativ hochwertige Tutorials, soliden Support und übersichtliche Dokumentation so gering wie möglich gehalten wird. Man hat auch als Anfänger recht rasche Erfolgserlebnisse (mittels Scaffolding). Was als störend empfunden werden kann ist die große Anzahl an Dateien und die gewöhnungsbedürftige Ordnerstruktur. Im Vergleich zu manueller Pro-

grammierung merkt man einen gewissen Kontrollverlust, da viel im Hintergrund automatisch passiert. Kleine Änderungen können somit zu einem langwierigen Prozess werden oder manchmal sogar dazu führen, dass die Funktion händisch ausprogrammiert werden muss, da sie ausgabespezifische HTML-Elemente enthält. Alles in allem merkt man aber auch schon bei kleineren Applikationen wie diesen den Gewinn von Übersichtlichkeit (durch die Trennung der 3 Komponenten des MVC-Patterns) sowie die Erleichterung der Entwicklung durch die Abnahme von Standardaufgaben.

4.3 Struts

4.3.1 Entwicklung

Apache Struts ist ein Open Source Framework zur Entwicklung von J2EE (Java 2 Platform Enterprise Edition) Webapplikationen. Einst als Teil des Apache Jakarta Projekts geführt, ist es im Jahr 2005 als *Apache Toplevel Projekt* anerkannt worden. In seiner ersten Version entwickelt und veröffentlich im Mai 2000 von Craig R. McClanahan, wurde es dann der Apache Foundation übergeben und von dieser bis zur heute aktuellen Version 1.3.2 verbessert. Die neueste Entwicklung ist das Unterprojekt *Struts Shale*, das auf JSF (Java Server Faces) als Basistechnologie aufbaut. Während von Struts in seiner klassischen Form (seit der Entwicklung Struts Shale umbenannt auf Struts Action) laut der Apache Foundation keine Version 2.x mehr vorgesehen ist, soll Struts Shale nicht einfach nur ein normales Upgrade sein, sondern eine komplette Überarbeitung vieler Kernprinzipien von Struts Action kombiniert mit den neuesten Erkenntnissen aus der Webentwicklung mit sich bringen. Die Weiterentwicklungen sind so tiefgreifend, dass manche Shale als komplett eigenständiges Framework ansehen.

In dieser Arbeit wird dennoch Struts in seiner klassischen Form unter die Lupe genommen. Struts Shale befindet sich zu diesem Zeitpunkt noch in einem recht frühen Entwicklungsstadium mit unzureichender Dokumentation und unabgeschlossenen Tests. Der Unternehmenseinsatz in diesem Stadium ist in diesem Moment noch nicht denkbar, ebenso wenig sind seriöse Aussagen über die Zukunft dieser Entwicklung zu machen.

4.3.2 Technologien

4.3.2.1 Java Server Pages (JSP)

Java Server Pages, abgekürzt mit JSP ist eine von Sun Microsystems entwickelte Technologie, die es Programmierern erlaubt, HTML, XML oder andere Dateitypen als Antwort auf einen Client-Request dynamisch zu generieren. Dabei werden Java-Code und spezielle vordefinierte JSP-Aktionen in den statischen Inhalt eingebunden.
Unter Verwendung eines speziellen JSP-Compilers wird die JSP zuerst in Java-Quellcode, der einem Java-Servlet entspricht, und anschließend in Bytecode umgewandelt. Zur Laufzeit werden die so erzeugten Java-Klassen von einem Webserver mit einer entsprechenden Servlet-Engine interpretiert.
Eine JSP kann grob in die folgenden 5 Bestandteile gegliedert werden:

- ✓ Statischer Inhalt wie HTML oder XML (auch bezeichnet als „Template Text“)
  Das sind all jene Elemente, die der Webserver ohne Veränderungen in den HTTP-Response übernimmt. Ein normales HTML-Dokument, das auf sonstige JSP-Elemente wie JSP-Aktionen oder Skriptelemente verzichtet, ist also ebenfalls eine gültige JSP. Das Einbinden dynamischer Daten in den statischen Inhalt ist nämlich zwar möglich, aber nicht zwingend notwendig. Umgekehrt kann aber auch gänzlich auf statische Elemente verzichtet werden, was das Erzeugen von Binärdaten wie bspw. Bilder über ein JSP Dokument erlaubt.
- ✓ JSP-Direktiven
  Mittels JSP-Direktiven können spezielle Seiteninformationen an den JSP-Compiler übergeben werden, bspw. welche Taglibs eingebunden werden müssen oder was im Fehlerfall passiert. Diese kontrollieren, wie der JSP-Compiler das Servlet generiert.
- ✓ JSP-Skriptelemente und Standardvariablen
  Auf Standardvariablen kann von jeder JSP aus zugegriffen werden. Sie werden auch als implizite Objekte bezeichnet.
  3 grundlegende Skriptelemente erlauben das Einfügen von Java-Code direkt in das Skriptlet.
- ✓ JSP-Aktionen
  JSP-Aktionen sind XML-Tags, die die eingebaute Funktionalität von Webservern nutzen.
- ✓ JSP Tag-Bibliotheken (Tag Libraries)
  Zusätzlich zu den vordefinierten JSP-Aktionen besteht die Möglichkeit, benutzerdefinierte Aktionen unter der Verwendung der JSP Tag Extension API einzusetzen. Dazu muss eine eigene JSP-Tag-Bibliothek in Form einer XML-Beschreibungsdatei, dem so genannten Tag Library Descriptor (TLD), zur Verfügung gestellt werden. Darin befindet sich eine Ansammlung von Tags, die wiederum mit serverseitigen Java-Klassen assoziiert werden, die die funktionale Logik eines oder mehrerer Tags implementieren.
  Die wichtigsten Customtags der verschiedensten Bibliotheken sind in der Java Server Pages Standard Tag Library (JSTL) standardisiert. Zusätzlich stehen die Jakarta Taglibs zur Verfügung, die viele zusätzliche Tag-Bibliotheken enthalten.

4.3.2.2 Java Servlets

Servlets sind Java-Klassen, deren Instanzen innerhalb eines Webservers die Anfragen eines Clients entgegennehmen und basierend darauf dynamisch eine Antwort generieren. Servlets sind somit die Antwort von Java auf andere Konzepte zur Erzeugung dynamischer Webinhalte wie bspw. CGI, PHP oder ASP.
Die Servlet API, die die Interaktionen zwischen einem Webcontainer und einem Servlet definiert, ist in der Java-Pakethierarchie `javax.servlet` enthalten.
Jedes Servlet muss folglich stets die Schnittstelle `javax.servlet.Servlet` oder eine davon abgeleitete (üblicherweise `javax.servlet.http.HttpServlet`) implementieren. Der Webcontainer ist im Wesentlichen die Komponente des Webservers, die mit dem Servlet

interagiert. Er verwaltet den Lebenszyklus eines Servlets, übernimmt das URL-Mapping auf ein bestimmtes Servlet und kümmert sich um Zugriffsrechte.

4.3.2.3 Java Beans

Java Beans sind Software-Komponenten geschrieben in der Programmiersprache Java. Sie waren ursprünglich für die „visuelle Programmierung“ gedacht, wo man nach dem Baukasten-Prinzip Programme aus Komponenten zusammenfügt, ohne selbst programmieren zu müssen.
Um eine Klasse als Java Bean bezeichnen zu können, muss diese bestimmten Konventionen bezüglich Namensgebung, Konstruktor und Verhalten gehorchen. Nur unter diesen Umständen ist gewährleistet, dass Entwicklungstools Java Beans (wieder)verwenden, ersetzen und verbinden können.
Die Konventionen sind:

- ✓ Die Klasse muss serialisierbar sein (fähig, den eigenen Status dauerhaft zu speichern und wiederherzustellen).
- ✓ Der Konstruktor hat keine Argumente
- ✓ Alle Eigenschaften einer Bean wahren das Prinzip der Datenkapselung. Der Zugriff ist also nur möglich über getter- und setter-Methoden, die bestimmten Namenskonventionen folgen müssen. Bsp.: Eigenschaft `name`, lesende Operation `getName()`, im Falle einer booleschen Variable `isName()`, schreibende Operation `setName()`.
- ✓ Die Klasse enthält alle benötigten Event-Handling Methoden.

```
// PersonBean

public class PersonBean implements java.io.Serializable {
    private String name;
    private boolean deceased;

    // Konstruktor (ohne Argumente)
    public PersonBean() {
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return (this.name);
    }

    public void setDeceased(boolean deceased) {
        this.deceased = deceased;
    }

    // andere Namenskonvention im Falle einer booleschen Variable
    public boolean isDeceased() {
        return (this.deceased);
    }
}
```

4.3.3 Aufbau

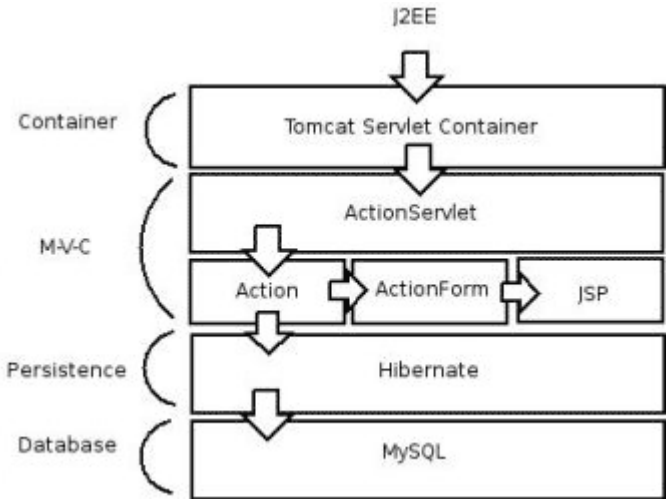


Abbildung 9: J2EE Stack (http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE, 20.06.2006)

4.3.3.1 MVC

4.3.3.1.1 Model 1 Architektur mit Servlets/JSP

Als Java Servlets auf den Markt kamen, wurden diese von vielen Programmierern mit offenen Armen empfangen. Das Schreiben von HTML-Code mittels endloser `println()`-Befehle erwies sich allerdings schnell als mühsam und problematisch.
Die Einführung der JSP-Technologie schaffte in diesem Punkt Abhilfe und erleichterte so vor allem Webdesignern ihre Arbeit. Deshalb konnte sich auch dieses Konzept innerhalb kurzer Zeit durchsetzen.
Die Model 1 Architektur basiert auf dem Gedanken, dass Datenhaltung, Geschäftslogik und die Darstellung in einer JSP vereint sind. Für sehr kleine Anwendungen hat diese Idee durchaus ihre Reize: die Ablaufsteuerung ist nicht sonderlich komplex und zusammengehörige Ansichten und Teile der Geschäftslogik können sofort in einer Seite kombiniert werden. Sobald die Applikation aber eine gewisse Größe übersteigt, stößt man bei diesem Ansatz auf zahlreiche Probleme: Die JSP wird unübersichtlich und schwer wartbar. Die Entwickler müssen sowohl HTML als auch Java-Kenntnisse aufweisen. Eine Aufteilung der Entwicklungsarbeit in Programmierung und Design ist ebenfalls nur schwer realisierbar, da beide Teams am selben Code arbeiten müssen. Die Wiederverwendbarkeit von Codefragmenten wird durch die Vermischung von Darstellung, Programmlogik und Ablaufsteuerung deutlich erschwert. Änderungen können meist nicht zentral vorgenommen werden, sondern ziehen sich über etliche Dateien hinweg.

4.3.3.1.2 Model-2-Architektur mit Struts

Die Lösung der Model-1-Architektur war auf Dauer nicht zufrieden stellend. Ein neues Konzept musste her, das die soeben genannten Probleme beseitigt. Man kam auf die Idee, JSP in Kombination mit Servlets zu verwenden: Die Servlets übernehmen dabei die Aufgabe der Programmflusssteuerung während sich die JSPs dem Schreiben des HTML-Codes widmen. Dieses Konzept ist heutzutage unter dem Namen Model 2 bekannt. Die Parallelen

des Models 2 zum MVC-Pattern waren unübersehbar. Deshalb werden heute die Begriffe Model 2, MVC oder Model2/MVC oft als Synonyme verwendet.
Die Model-2-Architektur hat ihren Einsatz im Struts-Framework gefunden. Ein wichtiges Prinzip dieses Modells ist, dass View Komponenten niemals über einen HTTP-Request direkt, sondern ausschließlich über Struts-Actions referenziert werden.

4.3.3.1.3 Model

Die Komponenten von Struts sind in der View und dem Controller anzusiedeln. Das Framework bietet keine typischen Model-Funktionalitäten. Als Model Layer werden Java Beans und EJB unterstützt.
Benannt nach den zwei wesentlichen Aufgaben des Models – der Repräsentation des aktuellen Zustands der Applikation und der Kapselung der Geschäftslogik – ist zu unterscheiden zwischen *System State Beans* und *Business-Logic Beans*.

Business-Logic Beans kapseln die Geschäftslogik ab. Innerhalb diverser Methoden, die von den Action-Klassen aufgerufen werden, kommt die Geschäftslogik zur Anwendung. Um eine saubere Trennung zwischen der Logik und den Daten zu gewährleisten, müssen die Daten an die Methoden mit übergeben werden.
Da die Java Beans keine Kenntnis davon haben (sollten), dass sie im Rahmen einer Webapplikation ausgeführt werden, können sie problemlos auch in anderen Umgebungen eingesetzt werden.

System-State Beans oder auch *ActionForm Beans* repräsentieren den Zustand einer Applikation und übernehmen die Verwaltung der vom Benutzer eingegebenen Daten.
Im Rahmen von Webapplikationen passiert die Datenübermittlung vom User über Formulare. Nach dem Befüllen verschiedenster Eingabefelder werden durch Drücken des Submit-Buttons die Inhalte an den Server übermittelt. Hat man in der Konfigurationsdatei `struts-config.xml` dem Formular eine ActionForm Bean zugeordnet, so sorgt das Struts-Framework dafür, dass die Attribute der ActionForm Bean, die mit den Namen der Felder im HTML-Formular übereinstimmen, automatisch mit den gesendeten Daten befüllt werden. Wird in dem Formular also bspw. Benutzername und Passwort abgefragt, so besitzt die verantwortlichte ActionForm Bean diese zwei Attribute, auf die über die üblichen getter- und setter-Methoden zugegriffen werden kann.
Eine Validierung der Daten nach der Übermittlung wird durch die Definition der Methode `validate()`in der ActionForm Bean erreicht. Innerhalb dieser Methode werden die Regeln, nach denen die Daten überprüft werden sollen, festgelegt. Im Falle einer fehlerhaften Benutzereingabe kann wahlweise auf eine Seite mit einer Fehlermeldung oder auf das Formular, das mit den zuvor eingegebenen Werten vorbelegt ist, weitergeleitet werden.

4.3.3.1.4 View

Die View ist für die Darstellung der Daten verantwortlich. Das Paradigma im Falle der Verwendung von JSP sollte dabei sein, so wenig Java Code wie möglich zu verwenden. Zu genau diesem Zweck stellt Struts zusätzlich zu der JSTL Taglibs zur Verfügung.
Tags dienen üblicherweise als Formatierungselemente zur Darstellung von Daten bspw. in HTML- oder JSP-Dokumenten. Darüber hinaus gibt es noch Tags, die zur logischen For-

matierung der Daten benutzt werden und die später in HTML-Tags transformiert werden. Struts stellt in den folgenden Libraries solche Formatierungselemente bereit:

- ✓ **HTML:** Unterstützt den Entwickler bei der Erzeugung HTML-basierter GUIs. Formularfelder spielen dabei eine große Rolle und stellen daher einen Großteil dieser Library dar.
- ✓ **BEAN:** Diese Taglib umfasst Tags für den Zugriff auf Java Beans und ihre Eigenschaften sowie zur Definition neuer Beans.
- ✓ **LOGIC:** Die Tags dieser Library kümmern sich um bedingte Ausgaben, Schleifen sowie Applikations-Ablaufsteuerung.
- ✓ **NESTED:** Diese Tags dienen zur Erweiterung anderer Struts-Tags und erlauben eine verschachtelte Struktur.

Neben der häufig verwendeten Technik der Struts Taglibs in Kombination mit JSP gibt es noch etliche andere View-Komponenten, die von Struts unterstützt werden wie bspw. Struts Cocoon, die Java-basierte Template-Engine Struts Velocity, Stxx (Struts for transforming XML with XSL) etc.

4.3.3.1.5 Controller

Vergleichbar mit dem Dispatch Servlet von Ruby on Rails handelt es sich bei dem *Action Servlet* in Struts um eine Musterimplementierung des Front Controller Patterns.
Die zentrale Steuerung der Abläufe in Struts passiert von hier aus. Die eintreffenden Requests werden zur Weiterverarbeitung den entsprechenden Actions zugewiesen. Diese können beim Aufruf entweder selbstständig die Business Logik ausführen oder (bei strikter Einhaltung des MVC) diese Aufgabe an Komponenten der zugrunde liegenden Applikation delegieren. Ist die Aktion ausgeführt, leitet das Action Servlet den Request weiter zu der Präsentationsschicht, also bspw. an eine Java Server Page, welche dann die zur Anzeige benötigten Daten aus dem Model extrahiert und dem Benutzer das Ergebnis in einem für ihn lesbaren Format aufbereitet.

Vom Action Servlet gibt es nur eine einzige Instanz, es ist also ein Singleton. Der Prozess der Weitergabe der HTTP-Anfragen an die zuständigen Aktionen wird auch als Request- oder Action-Mapping bezeichnet. Die Informationen für das Mapping entnimmt das Action Servlet aus einer Konfigurationsdatei, der `struts-config.xml`, die bei jedem Start des Applikationsservers eingelesen wird. Hier werden logische Namen mit physikalischen Adressen assoziiert.

Es gibt 4 Rubriken in dieser Konfigurationsdatei:

- ✓ **Data Sources** (= Verbindungen zu Datenbanken, optional)**:** Datenbanken können hier zentral definiert werden, um dann den Action-Klassen zur Verfügung zu stehen.

```
<data-sources>
    <data-source>
      <set-property property="autoCommit"
                    value="false"/>
      <set-property property="description"
                    value="MySQL DataBase"/>
      <set-property property="driverClass"
```

```
                        value="org.gjt.mm.mysql.Driver"/>
      <set-property property="maxCount"
                        value="4"/>
      <set-property property="minCount"
                        value="2"/>
      <set-property property="password"
                        value="mypassword"/>
      <set-property property="url"
                        value="jdbc:mysql://localhost:3306/dbname"/>
      <set-property property="user"
                        value="me"/>
    </data-source>
</data-sources>
```

- ✓ **Form Beans:** Das sind im wesentlichen Java Beans mit ein paar Spezialitäten, die dem Programmierer das Arbeiten mit HTML-Forms erleichtern sollen. Für jede HTML-Form sollte eine entsprechende Form Bean in der Konfigurationsdatei vorgesehen werden.
  In dem Codebeispiel unten wird eine Form Bean namens `loginForm` durch die Java-Klasse `ch.innovate.innoagent.form.LoginForm` implementiert.

```
<form-beans>
  <form-bean name="loginForm"
    type="ch.innovate.innoagent.form.LoginForm"/>
</form-beans>
```

- ✓ **Global Forwards:** Sie sind in allen Action Klassen quasi als globale Variablen verfügbar und mappen einen logischen Namen einer HTML, JSP oder Servlet Ressource zu.

```
<global-forwards>
    <forward name="logoff" path="/logoff.do"/>
    <forward name="error" path="/error.jsp"/>
</global-forwards>
```

- ✓ **Action Mappings**: Auch hier werden wiederum logische Namen den eigentlichen Java-Klassen, den Aktionen, zugeordnet.
  Bei unserem Beispiel leitet das Action Servlet einen Request `login.do` an die Klasse `ch.innovate.innoagent.action.LoginAction` weiter.
  Diese hat Zugriff auf die Form Bean `loginForm`, welche durch das Action Servlet bereits instanziert wurde und in der die Benutzereingaben aus der JSP Seite `login.jsp` in Form von Objektvariablen zur Verfügung stehen. `scope="request"` signalisiert, dass die Form Bean nach der Abarbeitung des Requests wieder verworfen wird. Die beiden lokalen `forwards` leiten die Antwort der LoginAction bei `success` auf die `main.jsp` oder bei `failure` auf den `loginerror.jsp` Bildschirm.

```
<action-mappings>
<action path="/login"
      type="ch.innovate.innoagent.action.LoginAction"
      name="loginForm"
      scope="request"
      input="/login.jsp">
      <forward name="failure" path="/loginerror.jsp"/>
      <forward name="success" path="/main.jsp"/>
</action>
</action-mappings>
```

Actions sind die Verknüpfung zwischen dem Struts-Framework und der zugrunde liegenden Business-Applikation. Jede Anfrage, die eine Änderung des Zustands der Anwendung mit sich bringt, wird durch den Aufruf der zuständigen Aktion bzw. ihrer `execute()`-Methode realisiert. Um die Unabhängigkeit des Models von den anderen beiden Komponenten des MVC zu gewährleisten, sollte die Business-Logik selbst allerdings nicht in der Aktion, sondern auf der Ebene der Business-Anwendung implementiert werden.
Actions sind außerdem im weiteren Sinne für die Navigation durch die Webseite verantwortlich. Die `execute()`-Methode gibt nach erledigter Arbeit ein ActionForward-Objekt zurück, mittels dem Struts entscheidet, welche Komponente die weitere Verarbeitung übernehmen muss. Im Regelfall ist das eine JSP-Seite, die den Response generiert, es kann aber auch auf andere Aktionen weiter verwiesen werden.

4.3.4 Erfolgsfaktoren

4.3.4.1 Investitionsschutz

Struts ist ein Framework, das mittlerweile 6 Jahre am Markt existiert. Entwickler wurden darauf geschult, eigene Taglibs wurden erstellt oder zugekauft, Tools wurden getestet und sorgfältig ausgewählt. In all das floss jede Menge Geld und Zeit. Seitens der Unternehmer ist es nur allzu verständlich, dass sie diesen Prozess nicht noch einmal durchleben möchten. Struts ist praxiserprobt, über Kinderkrankheiten ist dieses Framework schon längst hinweg, es hat sich auch in großen Projekten bewiesen.
In Punkto Investitionssicherheit spricht also alles für Struts.

4.3.5 Nachteile

4.3.5.1 Hohe Abhängigkeit vom Framework

Ein POJO (plain old Java Object) bezeichnet ein “ganz normales” Objekt, also frei von etwaigen externen Abhängigkeiten wie bspw. zwingend zu implementierenden Schnittstellen, einzuhaltenden Namenskonventionen oder notwendigen Annotationen. Der Vorteil solcher POJOs ist die Unabhängigkeit von der verwendeten Infrastruktur und somit die leichte Wiederverwendbarkeit und Wartbarkeit.
Um also zu gewährleisten, dass die Bindung zwischen Logik und Framework möglichst gering bleibt, ist die Verwendung von POJOs eines der obersten Prinzipien.
Genau dieses wird bei Struts aber gebrochen: Sämtliche Action-Klassen müssen von einer bestimmten Struts-Basisklasse erben. Die Actions sind also keine POJOs mehr, sondern verschmutzt durch frameworkgebundene, technische Details.
Entkopplung kann man lediglich durch die Einführung einer Zwischenschicht gewährleisten: Die Action-Klasse ruft eine andere Klasse auf, die nicht von einer Struts-Klasse erbt – das ist allerdings ein recht hoher Preis für die Entkopplung, den die meisten Entwickler kaum konsequent zu zahlen bereit sind.

4.3.5.2 Testbarkeit problematisch

Bedingt durch oben genannte Abhängigkeiten können mit Struts erstellte Anwendungen lediglich im Servlet-Container getestet werden oder mit bestimmten Frameworks wie StrutsTestCase oder Cactus.

4.3.5.3 Thread-Handling

Actions werden von Struts gecacht und immer wieder verwendet. Das gewährt einerseits eine Optimierung der Performance, ist allerdings auf der anderen Seite als Einschränkung zu sehen, da der Entwickler selbst die Actions zur Nutzung in einer multi-threaded Umgebung absichern muss.

4.3.5.4 Keine Verwendung der Standard-HTML-Forms

Forms werden in Struts üblicherweise mit einem speziellen Set von Custom-Tags implementiert. Die Syntax weicht dabei stark von der in HTML gebräuchlichen ab, was weder notwendig noch erstrebenswert ist. Andere Frameworks haben bereits bewiesen, dass auch bei dynamisch befüllten Formularen die Verwendung der einfachen HTML-Syntax möglich ist. Außerdem kann ein Entwickler sein Wissen von Standard-HTML nicht nutzen und wird stattdessen gezwungen, sich eine neue, Struts-spezifische Syntax der Custom-Tags anzueignen.

4.3.5.5 Keine Methoden zur Übersetzung der Request-Parameter in Java-Objekte

Alle Java-Webapplikationen müssen die Request-Parameter in Java Objekte wie bspw. Integer, Date oder BigDecimal konvertieren. Struts bietet dazu keine Unterstützung an. Es definiert weder Standardmethoden noch erleichtert es die Definition solcher Methoden in irgendeiner Weise.

4.3.5.6 ActionForms – unnötige Parallelarbeit

ActionForms sind nicht zur Implementierung der Model-Komponente gedacht, sondern vielmehr als vorausgehende Validierungsschicht zwischen dem HTTP-Request und der richtigen Model-Klasse zu verstehen. ActionForms und Model-Klassen existieren paarweise, was die Berechtigung von ActionForms stark in Frage stellt.

Dieser Nachteil ist eng verbunden mit dem vorhergehenden: Gäbe es Standardmethoden zur Übersetzung von Request-Parametern in Java Objekte, so könnten die Model Objekte mehr oder weniger direkt aus dem Request heraus erzeugt werden und ActionForms wären obsolet.

Obwohl Struts verlangt, dass Model-Objekte ausgehend von den mit ihnen verbundenen ActionForms erstellt werden, bietet es dem Entwickler außerdem keine Unterstützung zur Bewältigung dieser Aufgabe.

4.3.5.7 Fehlerhafte Implementierung des Command Patterns

Die `execute()`-Methode der Aktionen hat 4 Argumente, die immer übergeben werden müssen, ganz gleichgültig, ob diese für die Aktion notwendig sind oder nicht. Das widerspricht den Prinzipien, die von der GOF gelehrt werden. Während die zentralen Methoden frei von Implementierungsdetails gehalten werden, werden die Daten dem Konstruktor übergeben. Das hält die Klasse rein und implementierungsspezifische Daten werden übergeben, wo Vererbung und Polymorphismus nicht greifen.

4.3.5.8 Problematische Portierbarkeit

Die oben genannten Designprobleme von Struts werfen große Probleme bei Portierbarkeit auf. Die Integration von Struts-Anwendungen bspw. in Portallösungen ist mit einem hohen Änderungsaufwand verbunden.

4.3.6 Hard Facts

Entwickler:	erstmals herausgegeben von Craig R. McClanahan, dann der Apache Foundation übergeben
Erstmals erschienen:	Mai 2000
Letzte Version:	1.3.2
Offizielle Webseite:	http://struts.apache.org/
Lizenz:	Apache-Lizenz
Programmiersprache:	Java
Installation:	✓ Java Development Kit ✓ Servlet Container ✓ XML Parser ✓ Struts Action Library
Dokumentation/Support/ Community:	✓ Dokumentation ✓ Bücher ✓ Tutorials ✓ Manuals ✓ Freie Einsicht in den Source Code ✓ Mailinglisten ✓ Foren ✓ Struts Wiki (http://wiki.apache.org/struts/) ✓ IRC (Internet Relay Chat) ✓ div. RSS-Feeds ✓ Weblogs

	✓ Konferenzen ✓ Workshops
Plugins:	✓ Struts Taglibs ✓ Struts Tiles ✓ Struts Scripting ✓ Validation Plugin ✓ Struts Faces
Editoren / IDEs:	✓ Adalon ✓ IBM WebSphere Studio ✓ MDE for Struts ✓ Eclipse ✓ NetBeans ✓ JBuilder ✓ JCreator ✓ IntelliJ DEA ✓ Oracle JDeveloper ✓ JetBrains IDEA ✓ …
Implementierte Architektur-muster:	✓ MVC ✓ Front Controller zur Weiterleitung des HTTP-Reuqests ✓ Command Pattern
OS (Operating System)	betriebssystemunabhängig
Webserverunterstützung:	✓ Apache Tomcat ✓ Jrun ✓ Borland ✓ Bluestone ✓ iPlanet ✓ iPortal ✓ Novell ExteNd ✓ Orion ✓ Trifork ✓ WebLogic ✓ Websphere ✓ …

Unterstützung Ausgabeformate/Templates:	✓ XML/XSLT ✓ HTML ✓ JSP ✓ JSF ✓ Struts Taglib ✓ Struts Cocoon ✓ Struts Velocity ✓ Stxx
Hosting:	✓ Im deutschsprachigen Raum: 25 Anbieter (vgl. http://www.webhostlist.de/host/data/compare_webhosting.php, 16.04.2006)

Tabelle 6: Fakten zu Struts

4.3.7 Ausblick in die Zukunft

Obwohl Struts den Status eines gut dokumentierten, etablierten und ausgereiften Frameworks genießt und sich nach wie vor beachtlicher Beliebtheit erfreut, wird es durch die Konkurrenz vor neue Herausforderungen gestellt.
Andere, schlankere Frameworks wie bspw. Spring oder Tapestry ziehen das Interesse auf sich und Java Server Faces werden – zumindest wenn es nach der Meinung so mancher Entwickler geht – Struts sowieso mit der Zeit vom Markt verdrängen.

Struts hat auf diese Gefahr hin aber prompt mit der Entwicklung von Struts Shale geantwortet: Eine komplette Überarbeitung des klassischen Struts-Frameworks auf Basis von Java Server Faces.

Wer weiter mit dem klassischen Struts (seit Struts Shale umbenannt auf Struts Action) arbeitet, muss auch nicht unbedingt auf JSF verzichten: Das Struts Faces-Paket enthält eine Bibliothek, die die Nutzung der JSF User Interface Technologie anstelle von Struts Custom-Taglibs unterstützt.

4.4 Zoop

PHP Frameworks erfreuen sich seit geraumer Zeit steigender Aufmerksamkeit in der PHP-Community. Mittlerweile erscheinen beinahe täglich neue auf dem Markt. Mit über 40 konkurrierenden Frameworks fällt die Entscheidung, welches für die persönlichen Bedürfnisse am besten geeignet ist, schwer, besonders weil auch jedes von ihnen andere Funktionalitäten bietet.
In zahlreichen Blogs und Artikeln wird versucht, die Frage nach dem „perfekten" Framework zu beantworten, nur um am Ende der Diskussion dann festzustellen, dass es dieses einfach nicht gibt und – neben projektspezifischen Anforderungen – bis zu einem gewissen Grad auch Sache des persönlichen Geschmacks ist.

Diese Arbeit wird sich nicht in diese Diskussion einmischen. Da jedoch nicht alle PHP-Frameworks, die das MVC-Pattern implementieren, behandelt werden können, muss dennoch eine Auswahl getroffen werden.
Die Wahl fiel dabei auf das Framework Zoop. Die Gründe für diese Entscheidung sind folgende:

- ✓ Etliche der zur Auswahl stehenden Frameworks sind nach der Architektur von Ruby on Rails oder Struts modelliert. Da diese beiden Ansätze bereits behandelt wurden und nicht noch einmal wiederholt werden sollen, fielen diese aus der Wahl heraus.
  Zoop hat keines der beiden Vorbilder und bietet somit wieder einen neuen, interessanten Implementierungsansatz.
- ✓ Manche Frameworks befinden sich noch in einem relativ frühen Entwicklungsstadium und haben noch nicht einmal die Version 1.0 erreicht. Sie sind noch kaum praxiserprobt, fehlerbehaftet und ihre Zukunft ungewiss.
  Am Zoop-Framework wird mittlerweile seit 5 Jahren entwickelt und es hat bereits Version 1.1 erreicht. Man kann also davon ausgehen, dass es bereits stabiler und ausgereifter als so manch anderes Konkurrenzprodukt ist.
- ✓ Letztendlich ausschlaggebend waren außerdem noch die angebotenen Features. Mit der Implementierung von GuiControls, AJAX-Support, automatischer Form-Validierung und PDF-Erstellung stach es aus der breiten Masse hervor. Trotz der vielen Funktionen wird dennoch PHP 4 unterstützt.

4.4.1 Entwicklung

Zoops Geschichte begann mit der Herausgabe der ersten Version im Jahre 2001, damals noch unter dem Namen PeHpPy. Mittlerweile wurde es von einem relativ kleinen Kernteam, bestehend aus 4 Mitgliedern (hauptsächlich Mitarbeiter von Supernerd LLC) weiterentwickelt. Die letzte Version 1.1 wurde im Dezember 2005 veröffentlicht. Die Herausgabe von 1.2, die für den April 2006 angekündigt war, ist bis dato allerdings noch nicht erfolgt.

4.4.2 Technologien

4.4.2.1 PHP

PHP ist eine serverseitig interpretierte Skriptsprache mit einer an Java bzw. Perl angelehnten Syntax. Diese Open Source Software wird hauptsächlich zur Erstellung von dynamischen Webseiten oder ganzen Webanwendungen verwendet. PHPs Vorteile sind unter anderem die leichte Erlernbarkeit, die breite Datenbankunterstützung, die weitgehende Betriebssystemunabhängigkeit und die zahlreichen Funktionsbibliotheken. Für ausreichenden Support ist in zahlreichen Foren, Mailinglisten, Chats, Newsgroups etc. durch die ansehnliche Community gesorgt.

4.4.2.2 Smarty

Smarty ist eine quelloffene, in PHP geschriebene Template Engine. Durch das Platzieren spezieller Smarty Tags (meist) innerhalb von HTML ermöglicht sie eine Trennung von Code und Ausgabe und wird hauptsächlich zum Generieren dynamischer HTML-Seiten verwendet. Die Ausgabe kann allerdings auch jedes andere textbasierte Dateiformat (bspw. XML) sein. Erlaubte Tags sind Variablen in der Form von `{$var}` sowie eine Auswahl von logischen Operatoren (`{if}{else}`), Schleifen (`{foreach}, {section}`) und Modifikatoren (`{nl2br}, {lower}`).
Die Verarbeitung der Template-Dateien passiert in zwei Schritten. Im ersten wird aus der Kombination von Auszeichnungssprache-Elementen und Smarty Tags ein PHP File generiert. Diese temporäre Datei wird nur dann aktualisiert wenn sich in der zugrunde liegenden Template-Datei etwas ändert. Das hat den Vorteil, dass das Template nicht bei jedem Seitenaufruf neu geparst werden muss.
Im zweiten Schritt kann Smarty optional die endgültige Ausgabe auch zwischenspeichern. Der Webserver liefert in diesem Fall bei jedem Aufruf die statische gecachte Ausgabedatei. Das bewirkt besonders bei Datenbankzugriffen in PHP eine Entlastung des Servers und der Datenbank und beeinflusst die Performance positiv. Bei einer Aktualisierung der zugrunde liegenden Datenbank ist der Programmierer dann aber selbst dafür verantwortlich, den Cache zu löschen.

4.4.3 Aufbau

4.4.3.1 Zoop Skeleton Applikation

Dieses Skelett ist die minimalste Version einer mit Zoop geschriebenen Applikation. In ihm ist nichts Programmspezifisches enthalten, sondern lediglich die nackte Ordnerstruktur sowie einige unverzichtbare Dateien. Da bei jeder neuen Zoop-Version auch Änderungen am Skelett vorgenommen werden, empfiehlt es sich, immer das der Version des Frameworks entsprechende Skelett zu verwenden.

4.4.3.2 Konfiguration

Zwei wesentliche Dateien, die zugleich den Ausgangspunkt jeder Zoop-Applikation darstellen sind config.php und include.php

config.php
Das ist die Haupt-Konfigurationsdatei der Applikation, in der die wesentlichen Parameter zur Bestimmung des Verhaltens von Zoop gegenüber der Applikation gesetzt werden. In den meisten Fällen kommt man zu Beginn mit Beibehaltung der Standard-Einstellungen aus. Der einzige Wert, der unbedingt angepasst werden muss ist `zoop_dir`, also der Pfad zu Zoop am System.

includes.php
Der Inhalt dieser Datei trägt ebenfalls entscheidend zum Erfolg bzw. Misserfolg der Zoop-Applikation bei und gliedert sich in die folgenden 5 Teile:

1) Configuration
   Hier werden grundlegende Dateien wie bspw. config.php oder zoop.php eingebunden. Nach dem Erstellen einer neuen Instanz der Zoop-Klasse werden die in der Applikation benötigten Komponenten von Zoop inkludiert. Dadurch, dass man konsequent nur die Teile einbindet, die auch tatsächlich verwendet werden vermeidet man, dass die Applikation durch die Verwendung des Frameworks unnötig aufgebläht wird – ein Vorwurf, den sich andere Frameworks oft gefallen lassen müssen.
2) Zones
   Hier müssen alle in der Applikation verwendeten Zonen inkludiert werden. Beim Erstellen einer neuen Zone muss man also selbst darauf achten, dass diese auch hier eingebunden wird. Zone_admin wird also beispielsweise mit dem Befehl `$zoop->addZone('admin');` inkludiert.
3) Objects
   In diesem Teil werden Objekte inkludiert. Als Objekte werden in Zoop Bibliotheken bezeichnet. Sie sind im Applikationshauptverzeichnis unter dem Ordner *objects* zu finden und werden über den Befehl `$zoop->addObject('objectname');` eingebunden.
4) Misc
   Dieser Teil steht zur freien Verfügung für Standard-PHP includes und ist für organisatorische Zwecke gedacht.
5) Pear
   Wie der Name schon sagt ist dieser Abschnitt zum Einbinden von PEAR (PHP Extension and Application Repository)-Bibliotheken vorgesehen. Zoop verwendet PEAR auch in manchen Komponenten. Diese kümmern sich allerdings automatisch darum, dass die benötigten Libraries inkludiert sind.

4.4.3.3 Model

Wie bei Struts beschränken sich die Komponenten von Zoop auf die View und den Controller. Das Framework bietet also keine typischen Model-Funktionalitäten. Diese müssen selbst in Form von Klassen ausprogrammiert werden.

4.4.3.4 View

Zoop sieht für die View die Template Engine Smarty vor. Die Verwendung dieser Engine ist zwar keine zwingende Bedingung, wird aber dringend empfohlen, um die Trennung der View von den anderen beiden Teilen des MVC-Modells zu gewährleisten. Zoops GUI-Komponente stellt einige zusätzliche Funktionen zur Verfügung, die im Smarty Kernsystem nicht unterstützt werden. Der größte Unterschied ist aber der, dass Smarty ein Objekt verwendet, das standardmäßig `$Smarty` genannt wird und Zoop eines mit dem Namen `$Gui`. Der Einfachheit halber einigt man sich meistens auf `$Gui`.
Template Files werden im Applikationsverzeichnis unter */templates/default* abgespeichert. Diesbezügliche Konfigurationseinstellungen kann man in *config/gui.php* vornehmen. Beachtliche Geschwindigkeitsvorteile können durch einmalige Kompilierung und Caching der Templates erreicht werden. Deshalb ist diese Funktion auch standardmäßig aktiviert.

4.4.3.5 Controller

Zoop arbeitet mit so genannten Zonen.
Am besten werden diese anhand eines Vergleiches mit dem klassischen Aufbau einer PHP-Applikation erklärt. Dort gibt es viele unterschiedliche PHP-Dateien, die irgendwo in einem Verzeichnisbaum liegen. Will man so eine Datei ausführen, tut man das über die URI http://serveradresse/pfad_zur_datei/name_der_datei.php.
Die Entwickler von Zoop sehen in diesem Aufbau folgende Nachteile:

- ✓ Es ist unorganisiert und unübersichtlich.
- ✓ Es erfolgt keine Unterscheidung zwischen Bibliotheksdateien und Files, die Content generieren.
- ✓ Ein neues PHP-File pro Seite.
- ✓ Keine Umsetzung des MVC-Architekturmusters.

Aus diesen Gründen haben sie sich für die Verwendung von Zonen entschieden. Eine Zone wird von ihnen als *„Gruppe von Seiten mit einer ähnlichen Funktion oder einem ähnlichen Thema“* beschrieben.
Als Beispiel wird der Administrationsteil einer Webapplikation genannt. Im klassischen PHP-Beispiel gäbe es dafür wohl einen */admin*-Ordner, in dem alle für diesen Bereich zuständigen Files gespeichert werden würden. Im Zoop-Framework wird der Aufbau allerdings so organisiert, dass der Code einer Seite innerhalb einer Methode namens *page_XXX* in der *zone_admin* Klasse liegt. Diese wiederum ist in der gleichnamigen Datei *zone_admin.php* gespeichert. Anstatt vieler einzelner Files im Admin-Ordner gibt es also nun nur noch eine einzige Datei *zone_admin.php*, in der die Klasse *zone_admin* gespeichert ist. Jede Methode dieser Klasse behandelt eine einzelne Seite – daraus resultiert auch ihr Name *page functions*.

Aufgrund dieser Architektur ist auch die URI, um auf eine bestimmte Seite zu gelangen anders aufgebaut. Angenommen es gibt eine Seite zum Editieren der Daten eines Benutzers. Die Seite befindet sich im Admin-Bereich, z.B. *application_directory/admin/edituser.php*. Um diese Seite in einem Webbrowser aufzurufen würden man die URI http://hostname/application_directory/admin/edituser.php angeben. In Zoop sieht das anders aus. Der Code zum Editieren eines Benutzers würde in der Datei *zo-*

ne_admin.php in der Funktion *page_EditUser* zu finden sein und mit http://hostname/application_directory/index.php/admin/editUser ausgeführt werden.

Die Angabe von *index.php* nach dem Applikationspfad ist verpflichtend. *Index.php* ist ein Front Controller. Sollte danach in der URI nichts mehr angegeben sein oder die Angabe keiner gültigen Zone entsprechen, wird der Request zu *zone_default.php* weitergeleitet und darin die *page_default*-Funktion ausgeführt. In unserem Beispiel wird jedoch eine Zone angeführt (und zwar die namens Admin). Das bewirkt, dass ein Objekt der Klasse *zone_admin* instanziert wird und *index.php* den Request zu dieser Zone weiterleitet. Nach der Angabe des Zonennamens kann man an die Zone auch noch Parameter durch die URI in der Form http://hostname/application_directory/index.php/admin/parameter/editUser mitübergeben. Dahinter werden optional der Name der auszuführenden Methode innerhalb der Zone sowie an diese Methode zu übergebende Parameter angeführt. In unserem Beispiel wäre der Name der Methode *page_editUser* mit keinen übergebenen Parametern.

4.4.3.6 Kompenten

Das Zoop Framework setzt sich aus mehreren Komponenten zusammen, wobei jede einen fest definierten Funktionsumfang umfasst. Möchte man die Komponenten verwenden, muss man diese mittels des Befehls `$zoop->addComponent('komponentenname');` im Konfigurationsteil der *include.php* Datei einbinden.

Hier eine kurze Auflistung der Komponenten mit ihren Funktionen:

App:

- ✓ grundlegende Funktionen
- ✓ Fehlerbehandlung und –protokollierung
- ✓ XML-RPC Client/Server Integration
- ✓ Dateiverwaltung

Zone:

- ✓ URL-basierter Front-Controller
- ✓ Unterstützung der MVC-Architektur
- ✓ einfaches Erstellen neuer Seiten
- ✓ Organisierung ähnlicher Seiten in Zonen
- ✓ einfachere Sicherheitsabfragen durch Zonen
- ✓ trennt GET- und POST Requests durch die Verwendung von page/post-Funktionen

Gui:

- ✓ Integration von Smarty
- ✓ Bereitstellung von GUI-Controls
- ✓ automatische Validierung der Benutzereingaben durch JavaScript und PHP

DB:

- ✓ Verbindung zu jeder von PEARDB unterstützten Datenbank
- ✓ SQL-Funktionen, die über die Standardverbindung Daten aus der Datenbank extrahieren und formatiert zurückgeben
- ✓ ComplexUpdate für Updates mehrerer Reihen

Forms:

- ✓ Automatisiert die Auflistung (inkl. Nummerierung und Sortierung) und das Suchen von Datensätzen aus einer einzelnen Tabelle der Datenbank
- ✓ Automatisiert das Betrachten, Erstellen und Editieren von Datensätzen einer Tabelle
- ✓ Benutzt Gui-Controls, damit die Benutzereingaben automatisch validiert werden

Sequence:

- ✓ Nutzt XML zur Beschreibung des Navigationsablaufs
- ✓ Vermeidet komplizierte Logik beim Umgang mit der Benutzernavigation

Validate:

- ✓ Serverseitige Validierung
- ✓ Clientseitige JavaScript Validierung
- ✓ Unterstützung vieler verbreiteter Formate

Mail:

- ✓ Generierung von Mails mithilfe von Templates
- ✓ Unterstützung von HTML und Text E-Mails
- ✓ Unterstützung von SMTP und *sendmail* zum Senden von Nachrichten

PDF:

- ✓ Verwendet die Library http://www.ros.co.nz/pdf zum Generieren von PDF-Dateien
- ✓ Ergänzt Library mit HTML-ähnlichen Strukturen wie PdfTable, PdfTableRow, PdfTableCell

Session:

- ✓ Verwendet eine Datenbank anstatt von Standarddateien zum Speichern von Sessions

Spell:

- ✓ Rechtschreibprüfung, inklusive Verwaltung der vom Benutzer hinzugefügten Wörter

Userfiles:

- ✓ Speichert Files in einer Datenbank

4.4.4 Vorteile

4.4.4.1 GuiControls

GuiControls werden von den Zoop-Entwicklern als Hauptvorteil ihres Frameworks angeführt und gerne als Pendant zu .NETs Webcontrols genannt. Sie werden als wiederverwendbare Widgets beschrieben. Für die Validierung der eingegebenen Daten kann man den Datentyp sowie andere einfache Regeln (muss ausgewählt sein, Länge zwischen x und y Zeichen, etc.) festlegen.

4.4.4.2 Unterstützung von PHP4

Trotz der Bereitstellung etlicher attraktiver Features werden sowohl PHP 4 als auch PHP 5 unterstützt.

4.4.5 Nachteile

4.4.5.1 Zu wenige Informationsquellen

Das ist leider nicht nur ein Problem von Zoop, sondern auch von konkurrierenden PHP-Frameworks. Man macht das Framework dem Entwickler zwar mit vielen Features schmackhaft, will sich dieser dann aber einarbeiten, mangelt es an Tutorials und einer brauchbaren Dokumentation. Die Entwickler konzentrieren sich fast ausschließlich auf die technische Komponente des Frameworks und schenken dabei der Übermittlung ihres Wissens an andere viel zu wenig Aufmerksamkeit. Die Tutorials in Zoop bspw. sind unvollständig und offensichtlich fehlerhaft. Die Features, mit denen man angelockt wird, werden unzureichend bis gar nicht erklärt.
Die Community ist außerdem recht klein und bietet daher auch keine zufrieden stellende Unterstützung.

4.4.5.2 Wesentliche Funktionen fehlen

Zoop unterstützt die automatische Auflistung von Datensätzen aus einer Tabelle. Die Betonung liegt hier auf dem Wort „einer“. Sobald man nämlich Daten mehrerer Tabellen anzeigen möchte, fängt wieder die Handarbeit an – und die ist dann manchmal sogar langwieriger und komplizierter, als wenn man gleich alles selbst schreibt.
Automatisierte Funktionen greifen nur bei sehr allgemeinen Anliegen, sobald man irgendwelche von der Norm abweichenden Forderungen hat, muss man abermals selbst Hand anlegen. Die Vorteile des Frameworks werden somit durch den Mehraufwand bei der Integration eigenen Codes relativiert.

4.4.5.3 Undurchsichtigkeit

Dieser Vorwurf richtet sich weder speziell an Zoop noch an PHP-Frameworks, sondern an Frameworks im Allgemeinen. Durch die Automatisierung vieler Aufgaben geht auch ein gewisses Maß an Kontrolle verloren. Dadurch dass man nicht genau weiß, was der Code des Frameworks wie erledigt, steht man Fehlermeldungen oft hilflos gegenüber. Man weiß nicht, ob der Fehler im eigenen Code liegt, Funktionen des Frameworks falsch implementiert wurden oder – im schlimmsten Fall – der Fehler im Code des Frameworks selbst liegt. Kommt dazu noch das Problem von mangelndem Support und lückenhafter Dokumentation, ist die Frustration perfekt.

4.4.5.4 Zu großes Angebot

Es gibt zurzeit eine Unmenge an Angeboten für PHP-Frameworks. Wenn die Auswahl auch ihre Vorteile hat (natürliche Selektion, Verbesserung durch Konkurrenzsituation, etc.), so ist es hier schon fast zu viel des Guten. Anstatt immer wieder neue Frameworks mit leicht abgewandelten Konzepten (auch wenn sie sich manchmal im Endeffekt im Wesentlichen nur durch die Namensgebung unterscheiden) zu entwerfen, wäre es sinnvoller, sich zu größeren Communities zusammenzuschließen und das vorhandene Angebot qualitativ zu verbessern. Neben den technischen Features gäbe es da besonders bezüglich des Supports eine Menge Potential.
Zurzeit fällt der Einstieg in die Verwendung eines PHP-Frameworks doppelt schwer:

Als erstes muss man sich einen Überblick über die verfügbaren Frameworks schaffen und entscheiden, welches für die eigenen Bedürfnisse am bestem geeignet ist – das ist eine äußerst zeitintensive Aufgabe, wobei die endgültige Entscheidung im Endeffekt auch vom persönlichen Geschmack mit geprägt wird.

Zweitens muss man die Besonderheiten des Frameworks verstehen und damit umgehen lernen. Mangelnder Support erschweren das Überwinden dieser Hürde zusätzlich.

4.4.6 Hard Facts

Entwickler:	Kernteam bestehend aus 4 Mitgliedern, überwiegend Mitarbeiter von Supernerd LLC
Erstmals erschienen:	2001, damals noch unter dem Namen PeHpPy
Letzte Version:	1.1, herausgegeben am 13. Dezember 2005 (1.2 für April 2006 angekündigt, bis dato allerdings noch nicht freigegeben)
Offizielle Webseite:	http://zoopframework.com/
Lizenz:	ZPL (Zoop Public License, kompatibel mit der GPL)
Programmiersprache:	PHP (ab Version 4.3.10 möglich, 5.1 empfohlen)
Installation:	✓ PHP (ab 4.3.10, 5.1 bevorzugt) ✓ PEAR ✓ Zoop ✓ Zoop Skeleton Application
Dokumentation/Support/ Community:	✓ Tutorials ✓ Manuals ✓ Programmiertipps ✓ Videos ✓ Dokumentation ✓ der API ✓ Weblogs ✓ Foren ✓ Mailingliste
Plugins:	✓ App ✓ Zone ✓ Gui ✓ Db ✓ Forms

	✓ Sequence ✓ Validate ✓ Mail ✓ Pdf ✓ Session ✓ Storage ✓ Userfiles ✓ Spell
Editoren / IDEs:	✓ Zend Studio ✓ Dreamweaver ✓ Proton ✓ Weaverslave ✓ phpEdit ✓ PHPEclipse ✓ Scribe! ✓ Symantec Development Studio ✓ Dev-PHP ✓ Bluefish ✓ Vim ✓ CoolEdit ✓ …
Implementierte Architektur-muster:	✓ MVC ✓ Front Controller zur Weiterleitung des HTTP-Reuqests
OS (Operating System)	betriebssystemunabhängig
Webserverunterstützung:	✓ Apache ✓ Caudium ✓ Microsoft IIS ✓ OmniHTTPd ✓ Sambar ✓ Sun, iPlanet und Netscape Server ✓ Xitami ✓ Alle Webserver, die CGI unterstützen (vgl. http://www.phpbar.de/w/Webserver, 24.05.2006)

Datenbankunterstützung:	✓ MySQL ✓ Oracle ✓ IBM DB2 ✓ Microsoft SQL Server ✓ PostgreSQL ✓ Firebird ✓ SQL Lite ✓ MSSQL ✓ Sybase ✓ … (vgl. http://www.phpbar.de/w/Datenbank, 24.05.2006)
Unterstützung Ausgabeformate/Templates:	✓ HTML ✓ XML ✓ andere textbasierte Ausgabeformate ✓ Smarty
Hosting:	so gut wie jeder Provider bietet PHP, viele auch bereits PHP5

Tabelle 7: Fakten zu Zoop

4.4.7 Ausblick in die Zukunft

PHP-Frameworks profitieren davon, dass sie mit PHP als Programmiersprache eine breite Community ansprechen können.
Das große Angebot an Frameworks trägt einerseits zwar zur Erschwerung des Einstiegs bei Interessenten bei, fördert aber andererseits auch immer wieder neue Ideen zutage und sorgt für eine natürliche Selektion, was die Qualität der Produkte dann schrittweise verbessert.

Der entscheidende Punkt des Fortbestandes eines einzelnen Frameworks innerhalb des großen Angebotes ist neben den technischen Möglichkeiten auch die Dokumentation und der Support.
Am Beispiel von Zoop sieht man genau, wo die Probleme liegen: Interessenten werden mit beeindruckenden Features angelockt, beim Einstieg in die Materie dann aber durch fehlerhafte Tutorials, lückenhafte Dokumentation und mangelnden Support frustriert und im schlimmsten Fall wieder vertrieben.
Hier zeigt sich, dass technische Features nicht alles sind und nur dann vom Benutzer angenommen werden können, wenn er diese auch versteht.

4.5 ASP.NET

4.5.1 Entwicklung

ASP.NET ist der Name einer Reihe von Webentwicklungs-Technologien im Rahmen von Microsofts .NET-Plattform. Obwohl es der unmittelbare Nachfolger von Microsofts ASP-Technologie ist und auch von dieser den Namen weiterführt, unterscheidet sie sich in vielen wesentlichen Details von ihr.
Im Jahr 2002 erschien die finale Version 1.0 von ASP.NET am Markt. Gemeinsam mit Windows 2003 Server wurde rund ein Jahr später die Version 1.1 veröffentlicht. Zu diesem Zeitpunkt aktuell ist Version 2.0, die Ende 2005 gemeinsam mit Visual Studio 2005 und SQL Server 2005 herausgegeben wurde.
Diese Version brachte neben einer Vielzahl neuer Controls auch Werkzeuge zur einfacheren Verwaltung und Konfiguration von ASP.NET-Anwendungen sowie eine weitere Verbesserung der Performance mit sich.

4.5.2 Technologie

4.5.2.1 CLI – CLR – CIL

Eine der wichtigsten Prinzipien des .NET-Frameworks (und somit auch von ASP.NET) ist die Common Language Infrastructure, kurz CLI. CLI ist eine Spezifikation, die sprach- und plattformneutrale Anwendungsentwicklung und -ausführung ermöglicht. Microsofts Implementierung der CLI wird Common Language Runtime, abgekürzt CLR, genannt. Sie stellt eine Laufzeitumgebung für verschiedene von .NET unterstütze Hochsprachen zur Verfügung und kann so als Pendant zur Java Virtual Machine angesehen werden.

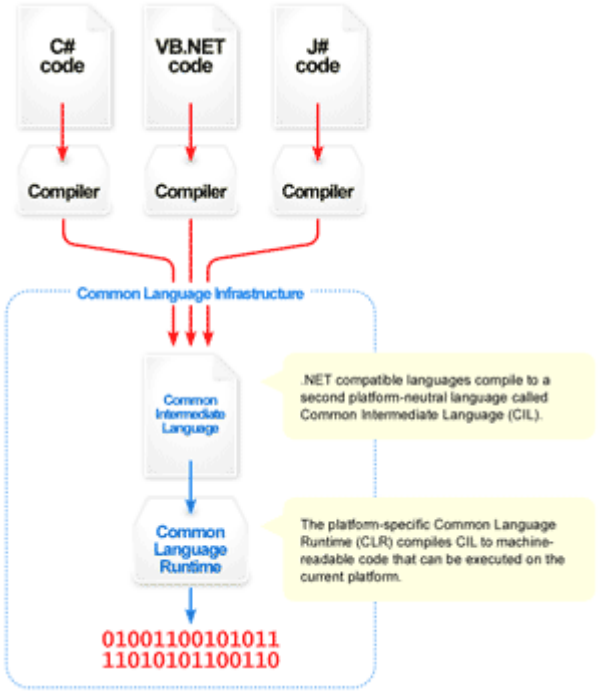


Abbildung 10: Übersicht über die CLI (http://en.wikipedia.org/wiki/Image:Overview_of_the_Common_Language_Infrastructure.png, 20.06.2006)

Die Kompilierung einer .NET-Applikation erfolgt in folgenden Schritten:
Zuerst wird der geschriebene Code zur Übersetzungszeit oder zur Laufzeit beim ersten

Aufruf in einen standardisierten Zwischencode, die CIL (Common Intermediate Language) übersetzt. Die CIL ist ein plattformunabhängiger Binärcode und kann mit dem Bytecode von Java verglichen werden. In älteren Publikationen ist die CIL auch noch unter ihrem früheren Namen MSIL (Microsoft Intermediate Language) zu finden. Sie wurde im Rahmen der Standardisierung durch die ECMA umbenannt. Damit auch in unterschiedlichen Sprachen geschriebene Programmteile auf gemeinsame Ressourcen zugreifen können, wurde für die CIL ein sprachübergreifendes System von objektbasierten Datentypen definiert.
Der in die CIL übersetzte Code wird in einer Assembly in einem speziellen Ordner des Dateisystems abgelegt, sodass beim nächsten Aufruf – sofern sich die Seite nicht geändert hat – eine erneute Übersetzung nicht notwendig ist.
Bei der Ausführung des Programms übersetzt der JIT(Just in Time)-Compiler, der in der Laufzeitumgebung CLR enthalten ist, den Zwischencode in Maschinencode, der nur auf der aktuellen Plattform ausgeführt werden kann.

4.5.2.2 Webforms

ASP.NET umfasst ein Programmiermodell für Webseiten, das als Webforms-Programmiermodell bezeichnet wird. Eine Webform ist eine .aspx-Seite, die neben konventionellem HTML-Code auch spezielle Tags mit dem Attribut `runat="Server"` umfasst. Diese Tags nennt man auch Server-Steuerelemente. Sie werden nicht direkt an den Client übertragen, sondern zuvor auf dem Server in ein oder mehrere Tags (und ggf. clientseitigen Scriptcode) umgewandelt, wobei erst zur Laufzeit über das Ausgabeformat entschieden wird. Der Client erhält nur die fertig umgewandelten Tags. Aus dem Steuerelement `<asp:Textbox>` wird bei der Umwandlung in einen HTML-Tag bspw. `<input type="text">`.

Die Server-Steuerelemente werden im Rahmen der serverseitigen Programmierung durch .NET-Objekte repräsentiert mit Attributen, Methoden und Ereignissen. Ein Webform-Objekt ist eine Hierarchie von Server-Steuerelement-Objekten.

Beispiel:

Deklaration in der ASPX-Seite (Server-Steuerelement-Tags)	Verwendung im Programmcode (Server-Steuerelement-Objekt)
`<asp:TextBox id="Vorname" runat="server"> bitte Vornamen hier eintragen </asp:TextBox>`	`Vorname.Text = "Claudia"`

Im Unterschied zum sequentiellen Erzeugen von HTML-Code, wie es im klassischen ASP oder vergleichbaren Techniken wie PHP üblich ist, erfolgt hier der programmatische Zugriff auf die Inhalte der Webseite objektorientiert. Microsoft behauptet, dass dadurch ASP.NET gegenüber ASP mit 40 – 70% weniger Programmcode auskommt.

Obwohl sich die Programmiermodelle von Windows-Anwendungen (Windows Forms) und Webseiten (Webforms) in letzter Zeit sehr ähnlich geworden sind, sind sie noch überhaupt nicht miteinander kompatibel. Der Traum vieler Entwickler, in der Windows-Welt gestaltete UIs 1:1 in das Web übernehmen zu können oder umgekehrt, bleibt also weiterhin unerfüllt.

4.5.2.3 Ereignisse

Der Programmcode in ASP.NET wird in serverseitigen Ereignisbehandlungsroutinen, die auf die Server-Steuerelement-Ereignisse reagieren, organisiert. Dabei muss man unterscheiden zwischen Ereignissen, die im Rahmen verschiedener Verarbeitungsschritte einer ASPX-Datei auf dem Server ausgelöst werden (bspw. `Page_load()`) und Ereignisse, die Aktionen und Zustandsänderungen am Client repräsentieren (`Button_Click()`), also typische Page-Controller Methoden.

Zweitere sind bei genauerer Betrachtung Pseudo-Ereignisse, da sie nicht sofort auf Änderungen reagieren, sondern zeitverzögert bei einer erneuten HTTP-Anfrage (=Roundtrip) ausgelöst werden. Ein wichtiges Prinzip des Webforms-Programmiermodells ist, dass alle Roundtrips HTTP-POSTs sind – man spricht hier von einem Postback. Erreicht werden kann das durch die Konfiguration der Server-Steuerelemente (Attribut `AutoPostBack`) oder durch clientseitigen Scriptcode bei Links oder Änderungen in Eingabefeldern.

Um die serverseitigen Ereignisse abarbeiten zu können, muss bei diesem Modell bei einem Postback immer die Seite selbst wieder aufgerufen werden. Die Seitenübergänge werden serverseitig durch die Übergabe der Programmkontrolle an eine andere ASPX-Seite oder durch Umleitung des Clients abgewickelt.

Diese ereignisorientierte Programmierung ermöglicht außerdem die saubere Trennung zwischen dem für das Layout verantwortlichen Code und dem Programmcode. Dieser Ansatz trägt den Namen *Code-Behind-Modell*.

Die Code-Behind-Datei trägt standardmäßig denselben Namen wie die zugehörige ASPX-Datei gemeinsam mit einer zusätzlichen Dateierweiterung, die die Programmiersprache bestimmt. Logisch verbunden werden die beiden Files durch das Attribut `CodeBehind` in der @Page-Direktive der ASPX_Datei.

Aus technischer Sicht basiert das Modell auf Vererbung. Die Code-Behind-Datei implementiert eine Klasse, die von `System.Web.UI.Page` erbt und die Klasse der ASPX-Datei erbt wiederum von der Code-Behind-Klasse.

Das Code-Behind-Modell muss nicht zwingend verwendet werden, sondern stellt lediglich eine Alternative zum bisherigen Single-File-Modell dar.
(vgl. http://www.aspnetdev.de/default2.aspx?start=http://www.aspnetdev.de/dotnet/aspnet/webforms.asp, 26.05.2006)

4.5.2.4 Steuerelemente

Man unterscheidet 2 Typen serverseitiger Steuerelemente: HTML-Server-Steuerelemente und Web-Server-Steuerelemente.

Web-Server-Steuerelemente erkennt man einfach am Namespace *asp*, der gleich am Beginn vor dem Tagnamen steht. Sie sind die mächtigste Form der Steuerelemente.

HTML-Steuerelemente sind konzeptionell eine Mischung aus Web-Server-Steuerelementen und Standard HTML-Elementen. Ein HTML-Server-Steuerelement unterscheidet sich rein optisch von einem normalen HTML-Tag lediglich durch den Zusatz `runat="server"`. Programmatisch gesehen besitzt es dennoch eine serverseitige Ereignisbehandlung und kann wie Web-Server-Steuerelemente objektorientiert verändert werden. Bei der Übersetzung besteht eine 1:1-Abbildung zwischen den HTML-Server-Steuerelementen und den HTML-Tags.
Funktional gesehen haben HTML-Server-Steuerelemente folgende Nachteile gegenüber Web-Server-Steuerelementen:

- ✓ Dadurch, dass ein HTML-Server-Steuerelement auch immer nur als HTML-Tag ausgegeben werden kann, ist das Ausgabeformat von vorn herein festgelegt. Auch die browserspezifische Ausgabe fällt weg.
- ✓ Es ist keine Ereignisweiterleitung an übergeordnete Server-Steuerelemente möglich.
- ✓ Keine Möglichkeit zum direkten Postback gegeben, da das `AutoPostBack`-Attribut fehlt.

Dennoch ist die Verwendung von HTML-Server-Steuerelementen in manchen Fällen durchaus gerechtfertigt:

- ✓ Sie erleichtern die Migration von klassischem HTML- oder ASP-Code durch die 1:1-Abbildung zwischen HTML-Tags und HTML-Server-Steuerelementen
- ✓ Die Hinterlegung clientseitiger Scripts wird vereinfacht, da keine Umwandlung der Quellcode-Tags seitens des Servers erfolgt
- ✓ Einige HTML-Tags können nur durch HTML-Server-Steuerelemente abgebildet werden, da es dafür (noch) keine Web-Server-Steuerelemente gibt

Manche Steuerelemente erfüllen ganz besondere Aufgaben: Validierungs-Server-Steuerelemente haben die Aufgabe, den Inhalt anderer Server-Steuerelemente nach bestimmten Kriterien zu überprüfen. Das können einfache Regeln wie „nicht leer“ oder „mindestens x Zeichen lang“ sein oder auch komplexere, die mithilfe von regulären Ausdrücken oder eigenen Validierungsfunktionen umgesetzt werden. Ist die Regel nicht erfüllt, so geben die Validierungs-Steuerelemente eine vorher festgelegte Fehlermeldung aus.

Es gibt außerdem noch Steuerelemente, die Datenmengen aus Listen oder Tabellen ausgeben können. Das mühsame zeilenweise Auslesen und die zeilenweise Ausgabe der Datenquelle entfallen hier.

4.5.3 Aufbau

4.5.3.1 MVC

4.5.3.1.1 Model

Wie auch bei den beiden vorher untersuchten Frameworks Struts und Zoop beschränken sich die Komponenten von ASP.NET auf die View und den Controller. Das Framework

bietet also keine typischen Model-Funktionalitäten. Diese müssen selbst in Form von Klassen ausprogrammiert werden.

Es gibt eine Hand voll Frameworks für .NET, die dem Entwickler diese Aufgabe erleichtern wollen: Maverick.NET, MaverickLite, Websharp, NStruts, MonoRail und IngeniousMVC. Meist mangelt es hier aber an Dokumentationen und Support und – schenkt man Blogs und Foren der Entwickler-Community Glauben – werden sie auch aus diesem Grund kaum verwendet.

4.5.3.1.2 View

Das Code-behind Modell, das in Visual Studio standradmäßig aktiviert ist, erleichtert bzw. automatisiert die Trennung zwischen der Präsentation (View) und dem Modell-Controller Teil. Der gesamte logische Code der ASP.NET Seite wird dabei in einer separaten Datei gespeichert. In einer zweiten Datei befindet sich lediglich noch der darstellungsspezifische Code. Mittels der Zeile `<%@ Page language="c#" Codebehind="Solution.aspx.cs" Inherits="Solution" %>` in der Präsentations-Datei wird der ASP.NET-Umgebung mitgeteilt, wo die Code-behind Klasse, auf deren Methoden in der View verwiesen wird, liegt und wie sie heißt.

4.5.3.1.3 Controller

Im Gegensatz zu den zuvor untersuchten Frameworks liefert ASP.NET ein typisches Beispiel für die Implementierung eines Page Controllers.

4.5.4 Vorteile

4.5.4.1 Programmiersprachenunabhängigkeit

Zurzeit gibt es ca. 40 Sprachen, die über einen Compiler für das .NET-Framework verfügen und deshalb innerhalb dieses verwendet werden können.
Microsoft unterstützt allerdings nur einen Bruchteil dieser Sprachen aktiv. Der Rest setzt sich aus von Drittanbietern entwickelten Sprachen zusammen. Diese mussten aufgrund der speziellen Anforderungen der CLR teilweise bedeutenden Änderungen unterzogen werden. Die meisten dieser Sprachen verfügen über frei erhältliche Compiler und manche sogar über eine eigene IDE.

4.5.4.2 Solider Support

Vermutlich auch dadurch, dass das Produkt einen kommerziellen Hintergrund hat, bekommt man ausführliche Dokumentationen und Support in mehreren Sprachen bereitgestellt.

4.5.4.3 Investitionssicherheit

Im Vergleich zu manchen Open Source Produkten ist die Investitionssicherheit dieses Frameworks – auch aufgrund seines kommerziellen Hintergrunds – höher als bei anderen.

4.5.5 Nachteile

4.5.5.1 Schwerere Erlernbarkeit

ASP.NET hat eine steilere Lernkurve als so manch andere konkurrierende Technologien.

4.5.5.2 Plattformunabhängigkeit nur theoretisch

Theoretisch ist die Plattformunabhängigkeit unter .NET vollständig gegeben, da der Hersteller Microsoft allerdings die .NET-Plattform nur für die eigene Produktlinie Windows anbietet, ist sie in der Praxis nur sehr bedingt vorhanden. Die wesentlichen Funktionen stehen aber in Form des Open Source Projekts Mono, das als freie .NET-Alternative gilt, bereits zur Verfügung.

4.5.6 Hard Facts

Entwickler:	Microsoft
Erstmals erschienen:	16.01.2002
Letzte Version:	Version 2.0, am 28.10.2005
Offizielle Webseite:	http://www.asp.net/
Lizenz:	Proprietäre Software
Programmiersprache:	Programmiersprachenunabhängig (alle unterstützten .NET-Sprachen)
Installation:	✓ Webserver ✓ .NET Framework SDK (Software Development Kit) – ist im Falle von Visual Studio schon inbegriffen
Dokumentation/Support/ Community:	✓ Bücher ✓ Tutorials ✓ Manuals ✓ Mailinglisten ✓ Foren ✓ Wikis ✓ IRC (Internet Relay Chat) ✓ div. RSS-Feeds ✓ Weblogs ✓ Konferenzen ✓ Workshops ✓ Magazine

Editoren / IDEs:	Visual Studio .NET ASP.NET Web Matrix Macromedia Dreamweaver Visual Web Developer 2005 Express Edition SharpDevelop …
OS (Operating System)	Windows 2000 und darüber
Webserverunterstützung:	Microsoft IIS (5.0/6.0) XSP-Server (Teil des Mono-Projekts, welches .NET unterstützt) Apache (mit dem Modol mod_mono, mit Einschränkungen)
Datenbankunterstützung:	Alle Datenbanken, die ODBC unterstützen
Unterstützung Ausgabeformate/Templates:	✓ Jegliches textbasierte Ausgabeformat
Hosting:	✓ Im deutschsprachigen Raum: 10 Anbieter (vgl. http://www.webhostlist.de/host/data/compare_webhosting.php, 16.04.2006)

Tabelle 8: Fakten zu ASP.NET

Zusammenfassung und Ausblick

In dieser Arbeit wurde das Design Pattern MVC in Zusammenhang mit der Entwicklung von Webapplikationen untersucht. Neben der theoretischen Abhandlung wurde auch der praktische Einsatz dieses Entwurfsmusters in mehreren relevanten Frameworks erforscht und dokumentiert.

Die Tatsache, dass die Ideen des MVC-Patterns in der Architektur jedes dieser Frameworks zumindest teilweise erfolgreich umgesetzt werden, zeigt, dass MVC sehr wohl auch in der Entwicklung von Webapplikationen praktische Relevanz besitzt.
Die im theoretischen Teil dieser Arbeit genannten Vor- und Nachteile dieses Design Patterns wurden außerdem durch ein für die Webprogrammierung typisches Beispiel untermauert (siehe Anhang). Vergleicht man den Code, der MVC berücksichtigt, mit dem, bei dem alle 3 dieser Rollen vermischt werden, so bewahrheiten sich die Vorteile dieses Entwursmusters schon selbst bei diesem kleinen und einfachen Programm.

Der Einsatz von Frameworks unter Verwendung der MVC-Architektur als Hilfsmittel für die Programmierung ist differenziert zu sehen: Frameworks können viele Standardaufgaben abnehmen und so den Entwicklungsprozess beschleunigen. Allerdings muss man dafür in der Anfangsphase einen nicht zu verachtenden Mehraufwand an Zeit investieren:
Erstens in die Auswahl des geeigneten Frameworks. Dieser Prozess kann in manchen Sprachen (wie bspw. Ruby) aufgrund des geringen Angebots einfach sein, erfordert aber meist eine genauere Beschäftigung mit mehreren konkurrierenden Produkten.
Zweitens muss man die Einarbeitungszeit berücksichtigen. Deren Dauer ist sehr von der Qualität der Dokumentation und des Supports abhängig.
Außerdem ist man gewungen, einen gewissen Grad an Abhängigkeit von dem gewählten Framework zu akzeptieren. Diese äußert sich einerseits aus der Sicht des Programmierers darin, dass die Applikation mehr oder weniger frameworkspezifische Details enthält, die die Portierbarkeit erschweren und andererseits, dass man sich auf die Funktionen des Frameworks verlässt und ein gewisses Maß an Kontrolle aufgibt. Aus der Sicht des Managements ist der Einsatz dieses Hilfsmittels mit einem gewissen Investitionsrisiko verbunden, da besonders bei Open Source Produkten eine Weiterentwicklung und Support nicht garantiert sind.

Für die Zukunft bleibt zu sagen, dass die Idee von MVC mit hoher Wahrscheinlichkeit nicht so schnell im Untergrund verschwindet. Ob sich die in dieser Arbeit beschriebenen Frameworks auch weiterhin erfolgreich gegenüber der Konkurrenz durchsetzen, kann an dieser Stelle nicht mit Sicherheit vorhergesagt werden. Die Verwendung von Frameworks im Allgemeinen hat sich allerdings bereits ausreichend durchgesetzt und wird in nächster Zukunft auch weiterhin dominieren.

Eine interessante Frage für weitere Untersuchungen in diesem Gebiet wäre, ob es mithilfe von MVC und Frameworks gelingt, eine Brücke zwischen der Webprogrammie-

rung und der klassischen GUI-Programmierung zu schlagen. Das Model von MVC sollte ja laut der Spezifikation bereits so programmiert werden, dass es unabhängig von der Einsatzumgebung verwendet werden kann. Fortschritte in dieser Richtung könnten die Welt der Programmierung in Zukunft entscheidend beeinflussen.

Anhang A: Usermanagement mit RoR

Voraussetzungen:

- ✓ Instant Rails 1.3, zu finden auf http://rubyforge.org/frs/?group_id=904: enthält Ruby 1.8.4, Ruby on Rails 1.1.2, MySQL 4.1.9, Apache 1.3.33, MySQL/Ruby 2.7, SCGI 1.7+, SCGI Rails Runner 0.4.3, phpMyAdmin 2.6.1)
  (vgl. http://rubyforge.org/frs/shownotes.php?release_id=4907, 16.06.2006)
- ✓ SQLFront 3.2 auf http://www.sqlfront.com/download.html (30-tägige Trial Version)
- ✓ Editor RadRails auf http://www.radrails.org/ zum Betrachten des Codes

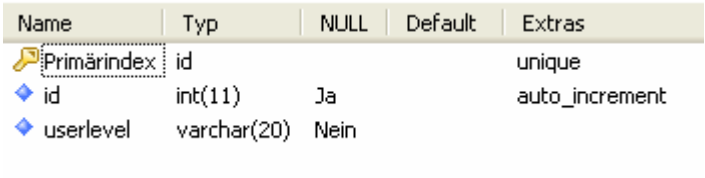
Vorgehensweise:

- ✓ Programme installieren
- ✓ DB namens *usermanagement* und Tabellen mithilfe SQLFront erstellen (*users.sql* und *usergroups.sql* auf der CD-ROM im Ordner *RoR* importieren)
- ✓ Ordner *usermanagement* (im RoR-Ordner) in *InstantRails\rails_apps* kopieren
- ✓ InstantRails starten, *Rails Applications ->Manage Rails Applications*, *usermanagement* anhaken und *Start with WEBrick*
- ✓ Aufruf der Applikation über http://127.0.0.1:3000/usergroup/list/

DB Tabelle *users*:

Name	Typ	NULL	Default	Extras
Primärindex	id			unique
id	id			unique
id	smallint(3) unsigned	Ja		auto_increment
username	varchar(150)	Nein		
password	varchar(32)	Nein		
usergroup_id	varchar(10)	Nein		
salutation	varchar(5)	Nein		
firstname	varchar(30)	Nein		
lastname	varchar(50)	Nein		
title	varchar(30)	Nein		
company	varchar(250)	Nein		
business	varchar(50)	Nein		
address	varchar(100)	Nein		
zip	smallint(6) unsigned	Nein	0	
city	varchar(50)	Nein		
country	varchar(50)	Nein		
state	varchar(50)	Nein		
phone	varchar(40)	Nein		
mobilephone	varchar(40)	Nein		
fax	varchar(40)	Nein		
email	varchar(100)	Nein		

DB Tabelle *usergroups*:



Name	Typ	NULL	Default	Extras
Primärindex	id			unique
id	int(11)	Ja		auto_increment
userlevel	varchar(20)	Nein		

Code des User-Modells (app/models/user.rb):

```
class User < ActiveRecord::Base

  # logische Verlinkung zwischen usergroups und users                #
  # Fremdschlüssel wird durch Namenskonventionen automatisch erkannt #
  belongs_to :usergroup
  validates_associated :usergroup

  # username, password und die Zuordnung zu einer Benutzergruppe (über
usergroup_id) sind obligatorisch #
  # Bestimmung der Fehlermeldung über den Parameter message
#
  validates_presence_of :username, :message=>"Bitte geben Sie einen Benut-
zernamen an"
  validates_presence_of :password, :message=>"Bitte geben Sie ein Passwort
an"
  validates_presence_of :usergroup_id, :message =>"Bitte ordnen Sie den
User einer Benutzergruppe zu"

  # Salutation muss entweder Mann oder Frau sein #
  validates_inclusion_of :salutation, :in=>%w( Herr Frau ),
:message=>"Bitte geben Sie Ihr Geschlecht an!"

  # Firstname muss zwischen 2 und 20 Zeichen lang sein #
  validates_length_of :firstname, :within => 2..30, :message=>"Der Vorname
muss zwischen 3 und 30 zeichen lang sein"

end
```

Code des User-Controllers (app/controllers/users_controller.rb):

```
class UsersController < ApplicationController
  # Defaultfunktion
  def index
    list
    render :action => 'list'
  end

  # Auflistung aller Benutzer
  def list
    @user_pages, @users = paginate :users, :per_page => 10
  end

  # Funktionen zur Sortierung nach Tabellenspalten
  def list_by_usergroup
    @user_pages, @users = paginate :users,
    :per_page => 10, :order_by => 'usergroup_id'
    render_action 'list'
  end

  def list_by_lastname
    @user_pages, @users = paginate :users,
```

```
    :per_page => 10, :order_by => 'lastname'
    render_action 'list'
  end

  def list_by_firstname
    @user_pages, @users = paginate :users,
    :per_page => 10, :order_by => 'firstname'
    render_action 'list'
  end

  def list_by_title
    @user_pages, @users = paginate :users,
    :per_page => 10, :order_by => 'title'
    render_action 'list'
  end

  def list_by_username
    @user_pages, @users = paginate :users,
    :per_page => 10, :order_by => 'username'
    render_action 'list'
  end

  # Anzeigen einer Liste von Details über einen bestimmte User
  def show
    @user = User.find(params[:id])
  end

  # Maske zum Anlegen eines neuen Benutzers
  def new
    @usergroups = Usergroup.find_all
    @user = User.new
  end

  # Maske zum Editieren eines Benutzers
  def edit
    @usergroups = Usergroup.find_all
    @user = User.find(@params[:id])
  end

  # Erstellen eines neuen Benutzers
  def create
    @user = User.new(params[:user])
    if @user.save
      flash[:notice] = 'User was successfully created.'
      redirect_to :action => 'list'
    else
      @usergroups = Usergroup.find_all
      render :action => 'new'
    end
  end

  # Updaten eines bestehenden Benutzers
  def update
    @user = User.find(params[:id])
    if @user.update_attributes(params[:user])
      flash[:notice] = 'User was successfully updated.'
      redirect_to :action => 'list', :id => @user
    else
      @usergroups = Usergroup.find_all
      render :action => 'edit'
    end
  end

  # löschen eines bestehenden Benutzers
  def destroy
```

```
    User.find(params[:id]).destroy
    redirect_to :action => 'list'
  end

end
```

Code-Beispiel der View zum Anzeigen einer Liste der Benutzerdatensätze (app/views/users/list.rhtml):

```
<% @heading = "Benutzerverwaltung" %>
<%= start_form_tag :action => 'new' %>
<table class="listusers">
<tr>
<th><%= link_to "Benutzergruppe", {:action => "list_by_usergroup"}, "alt"
=> "Sortieren nach Benutzergruppe" %></th>
<th><%= link_to "Nachname",{:action => "list_by_lastname"}, "alt" => "Sor-
tieren nach Nachname" %></th>
<th><%= link_to "Vorname",{:action => "list_by_firstname"}, "alt" => "Sor-
tieren nach Vorname" %></th>
<th><%= link_to "Titel", {:action => "list_by_title"}, "alt" => "Sortieren
nach Titel" %></th>
<th><%= link_to "Benutzername", {:action => "list_by_username"}, "alt" =>
"Sortieren nach benutzername" %></th>
<th> </th>
<th> </th>
</tr>
<!-- Einbinden des Partials _list_stripes.rhtml -->
<%= render_collection_of_partials "list_stripes", @users %>
</table>
<hr />
<%= submit_tag "Neuen Benutzer anlegen" %>
<%= end_form_tag %>
<%= "Page: " + pagination_links(@user_pages, :params => { :action =>
@params["action"] || "index" }) + "<hr />"
   if @user_pages.page_count>1 %>
```

Code des in list.rhtml aufgerufenen Partials (app/views/users/_list_stripes.rhtml):

```
<!-- if-else Konstrukt, mit dem die Klasse der Tabellenreihe anhand eines
Zählers alternierend -->
<!-- auf dk_gray und lt_gray gesetzt wird
-->
<tr class="<%= list_stripes_counter.modulo(2).nonzero? ? "dk_gray" :
"lt_gray" %>">
<td>
    <!-- h escaped den von Benutzer eingegebenen HTML-Code -->
    <%=h list_stripes.usergroup["userlevel"] %>
</td>
<td>
    <%=h list_stripes["lastname"] %>
</td>
<td>
    <%=h list_stripes["firstname"] %>
</td>
<td>
    <%=h list_stripes["title"] %>
</td>
<td>
    <%=h list_stripes["username"] %>
</td>
```

```
<td>
    <%= link_to_image("edit", { :controller => 'users', :action => "edit",
:id => list_stripes.id }) %>
</td>
<td>
    <%= link_to_image("delete", { :controller => 'users', :action =>
"destroy", :id => list_stripes.id }, :confirm => "Wollen Sie diesen Benut-
zer wirklich löschen?")%>
</td>
</tr>
```

Anhang B: Usermanagement ohne Framework

Voraussetzungen:

- ✓ XAMPP 1.5.3a
  (enthält Apache 2.2.2, MySQL 5.0.21, PHP 5.1.4, phpMyAdmin 2.8.1, FileZilla FTP Server 0.9.16c)

Vorgehensweise:

- ✓ Programme installieren
- ✓ DB namens *usermanagement* und Tabellen mithilfe phpMyAdmin erstellen (*users.sql* und *usergroups.sql* auf der CD-ROM im Ordner *PHP* importieren)
- ✓ Ordner *usermanagement* (im PHP-Ordner) in *xampp/htdocs* kopieren
- ✓ Xampp starten
- ✓ Aufruf der Applikation über http://localhost/usermanagement/usermanagement.php

DB Tabelle *users*:

Field	Type	Collation	Attributes	Null	Default	Extra
id	smallint(3)		UNSIGNED	No		auto_increment
username	varchar(150)	latin1_swedish_ci		No		
password	varchar(32)	latin1_swedish_ci		No		
usergroup_id	varchar(10)	latin1_swedish_ci		No		
salutation	varchar(5)	latin1_swedish_ci		No		
firstname	varchar(30)	latin1_swedish_ci		No		
lastname	varchar(50)	latin1_swedish_ci		No		
title	varchar(30)	latin1_swedish_ci		No		
company	varchar(250)	latin1 Swedish, case-insensitive		No		
business	varchar(50)	latin1_swedish_ci		No		
address	varchar(100)	latin1_swedish_ci		No		
zip	smallint(6)		UNSIGNED	No	0	
city	varchar(50)	latin1_swedish_ci		No		
country	varchar(50)	latin1_swedish_ci		No		
state	varchar(50)	latin1_swedish_ci		No		
phone	varchar(40)	latin1_swedish_ci		No		
mobilephone	varchar(40)	latin1_swedish_ci		No		
fax	varchar(40)	latin1_swedish_ci		No		
email	varchar(100)	latin1_swedish_ci		No		

DB Tabelle *usergroups*:

Field	Type	Collation	Attributes	Null	Default	Extra
id	int(10)			No		auto_increment
userlevel	varchar(20)	latin1_general_ci		No		

```
<html>
<head>
    <title>Benutzerverwaltung</title>
    <link href="style.css" media="screen" rel="Stylesheet" type="text/css"
/>
</head>
<body>
<?php

// *** Datenbankverbindung aufbauen
include("database.php");
$link_identifier = ls_db_connect();

//---------------- Benutzer löschen ------------------//
if ($_GET['action'] == "delete" && isset($_GET['id'])) {
   $sql = "DELETE FROM users WHERE id = $_GET[id]";
   if (ls_query($sql,$link_identifier)){
      $success_msg = "Benutzer wurde erfolgreich gelöscht";
   }
   else {
      $error_msg = "Benutzer konnte nicht gelöscht werden.";
   }
}

//---------------- Benutzerdatenn speichern ----------//
if (isset($_POST['save'])) {
   //*** Datenvalidierung
   if (!$_POST['username']) {
      $errors[] = "Bitte geben Sie einen Benutzernamen an.";
   }
   if (!$_POST['password']) {
      $errors[] = "Bitte geben Sie ein Passwort an.";
   }
   if(!isset($_POST['usergroup_id'])) {
      $errors[] = "Bitte ordnen Sie den User einer Benutzergruppe zu.";
   }
   if (!in_array($_POST['salutation'],array("Herr", "Frau"))){
      $errors[] = "Bitte geben Sie ihr Geschlecht an!";
   }
   if (strlen($_POST['firstname']) < 2 || strlen($_POST['firstname']) >
30){
      $errors[] = "Der Vorname muss zwischen 2 und 30 Zeichen lang sein!";
   }

   //*** keine Fehler -> Datenbankaktionen
   if (empty($errors)){
      $set_str = "SET username = '$_POST[username]', password =
'$_POST[password]', usergroup_id = '$_POST[usergroup_id]',
                  salutation = '$_POST[salutation]', firstname =
'$_POST[firstname]', lastname = '$_POST[lastname]', title =
'$_POST[title]',
                  company = '$_POST[company]', business =
'$_POST[business]', address = '$_POST[address]', zip = '$_POST[zip]', city
= '$_POST[city]',
                  country = '$_POST[country]', state = '$_POST[state]', pho-
ne = '$_POST[phone]', mobilephone = '$_POST[mobilephone]',
                  fax = '$_POST[fax]', email = '$_POST[email]'";
      if ($_POST['save'] == "new"){
         $sql = "INSERT INTO users {$set_str}";
      }
      else if ($_POST['save'] == "edit"){
         $sql = "UPDATE users {$set_str} WHERE id = {$_POST[id]}";
      }
      if (ls_query($sql, $link_identifier)){
```

```
            $success_msg = "Benutzerdaten wurden gespeichert.";
        }
        else {
            $error_msg = "Benutzerdaten konnten nicht gespeichert werden.";
        }
    }
    //*** Festlegen der Fehlermeldung
    else {
        $error_msg = "Daten konnten nicht gespeichert werden: Folgende Feh-
ler sind aufgetaucht.";
        $myuser = $_POST;
    }
}


//--- Ausgabe eventueller Erfolgs- bzw. Fehlermeldungen -----//
if (isset($success_msg)){ ?>
    <div style="background-color:#CCFFCC; border:2px solid #006600;
width:50% ; padding:5px">
        <?=$success_msg?>
    </div>
<? }
else if (isset($error_msg)){ ?>
    <div style="background-color:#CC9999; border:2px solid #990000;
width:50%">
        <? echo "{$error_msg} <ul>";
         foreach ($errors as $error){
            echo "<li>{$error}</li>";
         }
        ?>
    </ul></div>
<? }

//--------- Benutzerformular ------------------------------//
//*** extrahieren der benötigten Daten aus der DB
if (($_GET['action'] == "edit" || $_GET['action'] == "show") && is-
set($_GET['id'])){
    $sql = "SELECT id, username, password, usergroup_id, salutation, first-
name, lastname, title, company, business, address, zip, city, country,
            state, phone, mobilephone, fax, email FROM users WHERE id =
$_GET[id]";
    $res = ls_query($sql, $link_identifier);
    $myuser = mysql_fetch_assoc($res);
}

if($_GET['action'] == "edit" || $myuser['save'] == "edit") { ?>
    <h1>Benutzer editieren</h1>
    <form action="usermanagement.php" method="post">
        <input type="hidden" name="save" value="edit"> <?php
}

if($_GET['action'] == "new" || $myuser['save'] == "new"){ ?>
    <h1>Neuen Benutzer anlegen</h1>

    <form action="usermanagement.php" method="post">
        <input type="hidden" name="save" value="new">
<?php }

if ($_GET['action'] == "new" || $_GET['action'] == "edit" || is-
set($myuser['save'])){
    $sql = "SELECT id, userlevel FROM usergroups";
    $res = ls_query($sql, $link_identifier);
    $salutation_arr = array("chose" => "Bitte w&auml;hlen", "Frau" =>
"Frau", "Herr" => "Herr");
?>
    <input type="hidden" name="id" value="<?=$myuser['id']?>">
```

```
   <table>
      <tr>
         <td><label for="username">Username:</label></td>
         <td><input id="username" name="username" size="30" type="text"
value="<?=$myuser['username']?>" /></td>

         <td><label for="password">Passwort:</label></td>
         <td><input id="password" name="password" size="30" ty-
pe="password" value="<?=$myuser['password']?>" /></td>
      </tr>
      <tr>
         <td><label for="usergroup_id">Benutzergruppe:</label></td>
         <td><select name="usergroup_id">
         <?php while ($usergroup = mysql_fetch_assoc($res)){?>
            <option value="<?=$usergroup['id']?>"
            <?php if ($usergroup['id'] == $myuser['usergroup_id']){?>
               selected
            <?php } ?>
            ><?=$usergroup['userlevel']?></option>
         <?php }?>
            </select></td>
         <td>Anrede:</td>
         <td><select id="salutation" name="salutation">
               <?php foreach ($salutation_arr as $salutation_key => $sa-
lutation_value){?>
                     <option value="<?=$salutation_key?>"
                     <?php if ($myuser['salutation'] == $salutati-
on_key){?>
                        selected
                     <?php } ?>
                     ><?=$salutation_value?></option>
               <?php }?>
            </select>
         </td>
      </tr>
      <tr>
         <td><label for="firstname">Vorname:</label></td>
         <td><input id="firstname" name="firstname" size="30" type="text"
value="<?=$myuser['firstname']?>" /></td>
         <td><label for="lastname">Nachname:</label></td>
         <td><input id="lastname" name="lastname" size="30" type="text"
value="<?=$myuser['lastname']?>" /></td>
      </tr>
      <tr>
         <td><label for="title">Titel:</label></td>
         <td><input id="title" name="title" size="30" type="text" va-
lue="<?=$myuser['title']?>" /></td>
         <td><label for="company">Firma:</label></td>
         <td><input id="company" name="company" size="30" type="text" va-
lue="<?=$myuser['company']?>" />
      </tr>
      <tr>
         <td><label for="business">Gesch&auml;ft:</label></td>
         <td><input id="business" name="business" size="30" type="text"
value="<?=$myuser['business']?>" /></td>
         <td><label for="address">Adresse:</label></td>
         <td><input id="address" name="address" size="30" type="text" va-
lue="<?=$myuser['address']?>" /></td>
      </tr>
      <tr>
         <td><label for="zip">PLZ:</label></td>
         <td><input id="zip" name="zip" size="30" type="text" va-
lue="<?=$myuser['zip']?>" /></td>
         <td><label for="city">Stadt:</label></td>
         <td><input id="city" name="city" size="30" type="text" va-
lue="<?=$myuser['city']?>" /></td>
```

```
      </tr>
      <tr>
         <td><label for="country">Land:</label></td>
         <td><input id="country" name="country" size="30" type="text" va-
lue="<?=$myuser['country']?>" /></td>
         <td><label for="state">Staat:</label></td>
         <td><input id="state" name="state" size="30" type="text" va-
lue="<?=$myuser['state']?>" /></td>
      </tr>
      <tr>
         <td><label for="phone">Telefon:</label></td>
         <td><input id="phone" name="phone" size="30" type="text" va-
lue="<?=$myuser['phone']?>" /></td>
         <td><label for="mobilephone">Mobil:</label></td>
         <td><input id="mobilephone" name="mobilephone" size="30" ty-
pe="text" value="<?=$myuser['mobilephone']?>" /></td>
      </tr>
      <tr>
         <td><label for="fax">Fax:</label></td>
         <td><input id="fax" name="fax" size="30" type="text" va-
lue="<?=$myuser['fax']?>" /></td>

         <td><label for="email">E-Mail:</label></td><td>
         <input id="email" name="email" size="30" type="text" va-
lue="<?=$myuser['email']?>" /></td></tr>
         <!--[eoform:user]-->


     </table>

     </hr><br><br>
     <input name="commit" type="submit" value="Aenderungen speichern" />
   </form>

   <a href="usermanagement.php">Zur&uuml;ck zur
&Uuml;bersicht</a> | 
   <a href="usermanagement.php?action=show&id=<?=$myuser['id']?>">Details
anzeigen</a>
<? return;
}

//----------- Detailansicht eines bestimmten Benutzers ---------//

if ($_GET['action'] == "show"){?>
   <table>
      <tr>
         <td><label for="username">Username:</label></td>
         <td style="padding-right:10px;"><?=$myuser['username']?></td>
         <td><label for="password">Passwort:</label></td>
         <td><?=$myuser['password']?></td>
      </tr>
      <tr>
         <td><label for="usergroup_id">Benutzergruppe:</label></td>
         <td style="padding-right:10px;"><?=$myuser['usergroup_id']?></td>
         <td>Anrede:</td>
         <td><?=$myuser['salutation']?></td>
      </tr>
      <tr>
         <td><label for="firstname">Vorname:</label></td>
         <td style="padding-right:10px;"><?=$myuser['firstname']?></td>
         <td><label for="lastname">Nachname:</label></td>
         <td><?=$myuser['lastname']?></td>
      </tr>
      <tr>
         <td><label for="title">Titel:</label></td>
         <td style="padding-right:10px;"><?=$myuser['title']?></td>
```

```
            <td><label for="company">Firma:</label></td>
            <td><?=$myuser['company']?></td>
         </tr>
         <tr>
            <td><label for="business">Gesch&auml;ft:</label></td>
            <td style="padding-right:10px;"><?=$myuser['business']?></td>
            <td><label for="address">Adresse:</label></td>
            <td><?=$myuser['address']?></td>
         </tr>
         <tr>
            <td><label for="zip">PLZ:</label></td>
            <td style="padding-right:10px;"><?=$myuser['zip']?></td>
            <td><label for="city">Stadt:</label></td>
            <td><?=$myuser['city']?></td>
         </tr>
         <tr>
            <td><label for="country">Land:</label></td>
            <td style="padding-right:10px;"><?=$myuser['country']?></td>
            <td><label for="state">Staat:</label></td>
            <td><?=$myuser['state']?></td>
         </tr>
         <tr>
            <td><label for="phone">Telefon:</label></td>
            <td style="padding-right:10px;"><?=$myuser['phone']?></td>
            <td><label for="mobilephone">Mobil:</label></td>
            <td><?=$myuser['mobilephone']?></td>
         </tr>
         <tr>
            <td><label for="fax">Fax:</label></td>
            <td style="padding-right:10px;"><?=$myuser['fax']?></td>
            <td><label for="email">E-Mail:</label></td>
            <td><?=$myuser['email']?></td>
         </tr>
            <!--[eoform:user]-->
       </table>
       </hr><br><br>
   <a href="usermanagement.php">Zur&uuml;ck zur
&Uuml;bersicht</a> | 
   <a
href="usermanagement.php?action=edit&id=<?=$myuser['id']?>">Benutzerdaten
editieren</a>
<? return;
}

//------- Auflistung aller Benutzer -------------------------//

// Sortierung nach Tabellenspalten
switch ($_GET['action']){
   case "list_by_usergroup":
      $orderclause = "ORDER BY usergroup_id";
      break;
   case "list_by_lastname":
      $orderclause = "ORDER BY lastname ASC";
      break;
   case "list_by_firstname":
      $orderclause = "ORDER BY firstname ASC";
      break;
   case "list_by_title":
      $orderclause = "ORDER BY title ASC";
      break;
   case "list_by_username":
      $orderclause = "ORDER BY username ASC";
      break;
}

// Zusammenstellen der Benutzergruppen
```

```
$sql="SELECT id, userlevel
     FROM usergroups
     ORDER BY userlevel ASC";
$res = ls_query($sql, $link_identifier);

while($usergroup = mysql_fetch_object($res)){
   $usergroup_arr[$usergroup->id] = $usergroup->userlevel;
}

// Benutzerdaten extrahieren
$sql = "SELECT id, usergroup_id, username, lastname, firstname, title
      FROM users {$orderclause}";
$user_res = ls_query($sql, $link_identifier);

// Datenausgabe
?>
<h1>Benutzerverwaltung</h1>

<table class="listusers">
<tr>
<th><a
href="usermanagement.php?action=list_by_usergroup">Benutzergruppe</a></th>
<th><a href="usermanagement.php?action=list_by_lastname">Nachname</a></th>
<th><a href="usermanagement.php?action=list_by_firstname">Vorname</a></th>
<th><a href="usermanagement.php?action=list_by_title">Titel</a></th>
<th><a
href="usermanagement.php?action=list_by_username">Benutzername</a></th>
<th> </th>
<th> </th>
</tr>

<?php
   $colour_counter = 0;
   while ($user = mysql_fetch_assoc($user_res)){
      $my_usergroupid = $user['usergroup_id'];?>
      <tr class="<?php if ($colour_counter % 2 == 0) { echo "dk_gray"; }
else {echo "lt_gray"; }?>">
      <td><?=$usergroup_arr[$my_usergroupid]?></td>
      <td><?=$user['lastname']?></td>
      <td><?=$user['firstname']?></td>
      <td><?=$user['title']?></td>
      <td><?=$user['username']?></td>
      <td><a
href="usermanagement.php?action=edit&id=<?=$user['id']?>"><img
src="user_edit.png" alt="Benutzer editieren"></a></td>
      <td><a href="usermanagement.php?action=delete&id=<?=$user['id']?>"
onClick="return confirm('Wollen Sie diesen Benutzer wirklich
l&ouml;schen?')"><img src="user_delete.png" alt="Benutzer
l&ouml;schen"></a></td>
      </tr>
   <? $colour_counter ++;
   } ?>
</table>
<hr />
   <a href="usermanagement.php?action=new">Neuen Benutzer anlegen</a>
</body>

</html>
```

Glossar

Applet:
Zusammengesetzt aus den Begriffen **Appl**ication und snipp**et** (= Anwendungs-Schnipsel) wird unter diesem Ausdruck meist ein Java-Applet verstanden. Das ist ein kleines, in der Programmiersprache Java geschriebenes Programm, das auf der Client-Seite (als Browser-Plugin) ausgeführt wird.

AJAX (Asynchronous JavaScript and XML):
Ein Konzept der Datenübertragung zwischen Server und Client, welches interaktive, desktopähnliche Webanwendungen ermöglicht. Durch die Zusammenarbeit von HTML, CSS, JavaScript, DOM, XML, XSLT, dem XMLHttpRequest-Objekt oder On-Demand JavaScript muss nicht mehr bei jeder HTTP-Anfrage die komplette Seite neu geladen werden. Stattdessen werden lediglich die Teile, die sich verändert haben, sukzessiv von Server geholt.

Apache HTTP Server:
Der Apache HTTP Server ist ein von der Apache Software Foundation entwickelter, frei erhältlicher HTTP Webserver. Er ist nach wie vor der am weitesten verbreitete Webserver im Internet.

Apache Software Foundation:
Eine ehrenamtlich arbeitende Non-Profit Organisation, die sich der Förderung von Apache Softwareprojekten verschrieben hat.

API (Application Programming Interface):
Ein API (auf Deutsch Programmierschnittstelle) ist eine Schnittstelle, die von einem Betriebssystem, einer Bibliothek oder einer Applikation zur Verfügung gestellt wird, um Funktionsaufrufe von oder Datenaustausch mit anderen Programmen zu ermöglichen.

Applikation:
Synonym zum deutschen Ausdruck *Anwendung(sprogramm)* bezeichnet dieser aus dem lateinischen stammender Begriff in der EDV ein Computerprogramm zur Lösung bestimmter Aufgaben.

Application Controller:
„Ein zentraler Punkt für die Steuerung der Bildschirmnavigation und des Flusses einer Anwendung.“ (Fowler 2003, S:419)

Argument:
Eine Variable, die einem Unterprogramm beim Aufruf übergeben wird. Dieses passt sein Verhalten während der Laufzeit abhängig vom Wert des Arguments an.

Array:
In der Programmierung bezeichnet der aus dem englischen kommende Begriff Array (= *Anordnung, Reihung*) eine Datenstruktur. Eine Reihe von Datenelementen – üblicherweise vom selben Datentyp – werden darin gespeichert und können entweder im Falle eines indizierten Arrays über einen Index oder bei einem assoziativen Array über den jeweiligen Schlüssel angesprochen werden.

ASP (Active Server Pages):

Hinter diesem Begriff verbirgt sich eine von Microsoft entwickelte, serverseitige Technologie, die unter Einsatz einer Skriptsprache wie VBScript oder JScript Webseiten dynamisch generiert.

Auszeichnungssprache:
Ursprünglich von den Setzern aus der Druckindustrie verwendet, werden mit einer Auszeichnungssprache (engl. Markup Language, Abk. ML) die Eigenschaften, Zugehörigkeiten und Verfahren von bestimmten Textabschnitten mittels Tag-Markierungen bestimmt.

Benutzerschnittstelle:
siehe UI

Betriebssystem:
siehe OS

Blog:
Ein Blog (Kurzform für Weblog) bezeichnet eine (meist auf ein bestimmtes Thema spezialisierte) Webseite, die periodisch neue Einträge erhält, welche dann in umgekehrter chronologischer Reihenfolge (also mit dem neuesten Eintrag an der Spitze) aufgelistet werden.

Browser:
Angelehnt an das englische *browse* (blättern, durchsuchen, schmökern) bezeichnet dieser Begriff ein Computerprogramm, das dem Benutzer ein Betrachten und Interagieren mit Inhalten aus dem WWW ermöglicht.

Bytecode:
Der Bytecode ist eine Art maschinenunabhängiger Zwischencode. Er wird in manchen Programmiersprachen (bspw. Java) bei der Kompilierung anstatt des Maschinencodes erzeugt. Der Maschinencode wird dann erst zur Laufzeit für den jeweiligen Prozessor von der virtuellen Maschine durch Ausführung dieses Zwischenergebnisses generiert.

Cache:
In der EDV bezeichnet Cache einen in unterschiedlichsten Geräten zum Einsatz kommenden Puffer-Speicher, der Kopien von Inhalten anderer Speicher enthält und dadurch den Zugriff darauf beschleunigt.

CGI (Common Gateway Interface):
CGI ist ein Webstandard, der den Datenaustausch zwischen einem Webserver und den darauf bereitstehenden Applikationen ermöglicht. Unterstützt die Webserver-Software diese Schnittstelle und steht der dementsprechenden Laufzeitinterpreter zur Verfügung, kann ein CGI-Programm in allen möglichen Programmiersprachen geschrieben sein. Diese schon länger bestehende Technik, dynamische, interaktive Webseiten zu erzeugen hat den Nachteil einer geringen Geschwindigkeit.

Client:
Der Begriff Client bezeichnet ein Computerprogramm, das die Ressource einer zentralen Station (eines Servers) über ein Netzwerk nutzt. Ein typisches Beispiel für einen Client ist ein Webbrowser, der zu einem Webserver Kontakt aufnimmt, um von diesem eine Webseite abzurufen.

COM (Component Object Model):
COM ist eine im Jahr 1993 von Microsoft herausgegebene, proprietäre Plattform, die die leichte Wiederverwendung von geschriebenem Programmcode und programmiersprache-nunabhängige Entwicklung ermöglichen soll.

Command-Pattern:
Das Command-Pattern ist ein von der GoF entwickeltes Entwurfsmuster. In einem Command-Objekt werden Aktionen und dazugehörige Parameter gekapselt. Das bringt nicht nur den Vorteil eines besseren Programmdesigns und einer generischen Struktur mit sich, sondern ist auch sinnvoll, wenn das Erstellen und das Ausführen des Befehls zu verschiedenen Zeiten oder in einem anderen Kontext stattfindet oder wenn ein Rückgängig-Mechanismus implementiert werden soll.

Community:
Eine Community ist eine Gruppe von Menschen, die gemeinsames Wissen entwickeln, Erfahrungen teilen und dabei eine eigene Identität aufbauen. Im Falle einer Online-Community passiert der Austausch über eigens dafür eingerichtete Plattformen, wobei Chat, Instant-Messenger und Foren die verbreitetsten Kommunikationswege sind. (vgl. http://de.wikipedia.org/wiki/Community, 02.06.2006)

Compiler:
Ein Compiler (auch Kompilierer oder Übersetzer genannt), ist ein Computerprogramm, das ein in der Quellsprache geschriebenes Programm in ein semantisch äquivalentes Programm der Zielsprache übersetzt. Üblicherweise handelt es sich dabei um die Umwandlung eines von einem Entwickler in einer Programmiersprache geschriebenen Quelltextes in Assemblersprache, Bytecode oder Maschinensprache.

CMS (Content Management System):
Bezeichnung für ein Anwendungsprogramm, das die gemeinschaftliche Erstellung und Bearbeitung von Text- und Multimedia-Inhalten ermöglicht und organisiert.

CPL (Common Public License):
Die CPL ist eine von IBM herausgegebene Open Source Software Lizenz, die das kostenlose Recht zur Nutzung der Software gewährt.

CRUD (Create Read Update Delete):
In der Informatik bezeichnet dieser Begriff die 4 Basisoperationen einer Datenbank oder einer anderen Persistenz-Schicht. Von manchen Autoren wird alternativ das Akronym RUDI (Read Update Delete Insert) verwendet.

CSS (Cascading Stylesheet):
Eine Stylesheet-Sprache, die die Präsentation eines in einer Markup-Sprache geschriebenen Dokumentes bestimmt. Die am meisten verbreitete Anwendung ist das Gestalten von Webseiten, die in HTML oder XHTML geschrieben sind. CSS kann aber auch auf jede Art von XML-Dokument angewandt werden.

Dateiformat/Dateityp:
Das Dateiformat bzw. der Dateityp legt die Form der Abspeicherung von Computerdaten fest. Da für das Betriebssystem sämtliche Daten nur Aneinanderreihungen von Bytes sind, müssen Konventionen über ihre Interpretation getroffen werden. Die Summe all dieser

Konventionen für eine bestimmte Art von Dateien bezeichnet man auch als Dateiformat/Dateityp.

Datenkapselung:
Datenkapselung (engl. *encapsulation*) ist ein wichtiges Konzept der OOP. Ein Zugriff auf Methoden und Attribute von Klassen durch andere Klassen erfolgt dabei ausschließlich über definierte Schnittstellen. So wird einerseits verhindert, dass der interne Zustand einer Klasse unerwartet geändert oder gelesen wird und somit Invarianzen des Programms entstehen und andererseits das Innenleben einer Klasse mit all den Implementierungsdetails vor dem Verwender verborgen.

Datenstruktur:
Eine Datenstruktur ist eine bestimmte Art der Datenspeicherung, die eine effektive Verwaltung und einen geeigneten Zugriff der enthaltenen Daten garantiert. Mittels vordefinierten Operationen wird eine Manipulation der Daten ermöglicht.

Datentyp:
Im Kontext mit Programmiersprachen bezeichnet dieser Begriff die Zusammenfassung konkreter Wertebereiche und darauf definierten Operationen zu einer Einheit.
(vgl. http://de.wikipedia.org/wiki/Datentyp, 02.06.2006)

DB (Datenbank):
Der Datenspeicher eines Datenbanksystems.

DBMS (Datenbankmanagementsystem):
Die Verwaltungssoftware eines Datenbanksystems, die die Aufgabe hat, Daten in der Datenbank zu speichern, zu organisieren und zu modifizieren oder über die Abfragesprache gestellte Abfragen zu beantworten.

DBS (Datenbanksystem):
Ein Datenbanksystem ist ein System zur elektronischen Datenverwaltung, dessen wesentliche Aufgabe die sichere Speicherung großer Datenmengen ist. Benutzer oder Anwendungssoftware können diese durch Abfragen beeinflussen.
Das System besteht aus einer Verwaltungssoftware, auch genannt Datenbankmanagementsystem und aus dem eigentlichen Datenspeicher, der Datenbank.

Dekrement:
Vom lat. *decrementare* (vermindern) abgeleitet bezeichnet dieser Begriff in Programmiersprachen eine Operation, bei der der Wert einer Variablen auf den nächst niedrigeren geändert wird.

Dispatching:
In der Programmierung bezeichnet Dispatching den Prozess der Zuordnung einer Nachricht zu einem bestimmten Codeabschnitt (einer Methode) zur Laufzeit.

DOM (Document Object Model):
Das DOM ist eine vom W3C definierte API, die durch einen Satz von Klassen zusammen mit deren Methoden und Attributen Programmen erlaubt, dynamisch den Inhalt, die Struktur oder das Layout eines HTML- oder XML-Dokuments zu verändern.

EJB (Enterprise Java Beans) :

Enterprise Java Beans sind in Java geschriebene, serverseitige Komponenten, die Businesslogik kapseln und die Entwicklung komplexer, verteilter Softwaresysteme vereinfachen. Sie stellen wichtige Konzepte (bspw. Transaktions-, Namens- und Sicherheitsdienste), die für die Geschäftslogik benötigt werden, bereit.

eRb (Embedded Ruby):
eRuby ist eine Implementierung, die Ruby-Programmcode in ein HTML oder ein anderes Textdokument einbettet. Zur Laufzeit muss – ähnlich wie bei ASP, JSP oder PHP – ein Interpreter die Codefragmente übersetzen und durch HTML oder anderen Content ersetzen. Eine eRuby Datei ist durch die Endung *.rthml* erkennbar.

Eventlistener:
siehe Observer Pattern

FastCGI:
Vergleichbar mit dem CGI ist FastCGI ein Standard zur Generierung dynamischer Webseiten durch die Einbindung externer Software in einen Webserver. Im Gegensatz zu CGI bietet FastCGI eine bessere Performance und Skalierbarkeit.

Fat-Client:
Synonym für Rich-Client oder Smart-Client; Bezeichnung, die in einer Client-Server Architektur einen Client beschreibt, auf dem die Verarbeitung der Daten vor Ort erfolgt und der meist auch eine grafische Benutzeroberfläche zur Verfügung stellt.

GOF (Gang of Four):
Sammelbegriff für Erich Gamma, Richard Helm und John Vlissides, die Autoren des Buchs *„Design Patterns – Elements of Reusable Object-Oriented Software“*, einem Standardwerk im Bereich Software Engineering über Entwurfsmuster.

GPL (General Public License):
Eine von der Free Software Foundation herausgegebene Lizenz für die Lizenzierung freier Software, die vier Freiheiten gewährt: 1. Die uneingeschränkte Nutzung des Programms zu jedem Zweck, 2. Die kostenlose Verteilung von Kopien des Programms (inklusive Verfügungstellung des Quellcodes), 3. Das Studieren der Arbeitsweise des Programms sowie die Anpassung an die eigenen Bedürfnisse, 4.Der Vertrieb veränderter Versionen des Programms unter den Bedingungen der Regel 2.

GUI (Graphical User Interface):
Auf Deutsch als „grafische Benutzeroberfläche“ bezeichnet, beschreibt dieser Begriff eine Softwarekomponente, die dem Benutzer unter Verwendung eines Zeigergerätes (typischerweise einer Maus) die Interaktion mit dem Computer über grafische metapherhafte Elemente (z.B. Arbeitsplatz, Papierkorb) ermöglicht.

Hosting:
Der Begriff Hosting (übersetzt bewirten) bezeichnet ein Service, das Organisationen und Privatpersonen erlaubt, Inhalte im Internet bereitzustellen.

HTML (Hypertext Markup Language):
HTML ist eine Auszeichnungssprache, die typischerweise Webseiten zur Darstellung im Browser als Hypertext beschreibt. Die Sprache wird zur Strukturierung – Textabschnitte könne als Überschriften, Paragraphen, Aufzählungspunkte und ähnliches gekennzeichnet

werden – aber auch bis zu einem gewissen Grad zur Bestimmung des Aussehens und der Semantik von Informationen verwendet.
HTML wurde vom W3C bis Version 4.01 weiterentwickelt und dann von XHTML abgelöst.

HTTP (Hypertext Transfer Protocol):
Dieses zustandslose Internetprotokoll wird dazu verwendet, um HTML-Seiten und andere Daten im Web zu übertragen.

IDE (Integrated Development Environment):
Mit diesem Begriff (auf Deutsch Integrierte Entwicklungsumgebung) wird ein Anwendungsprogramm beschrieben, das Programmierer bei der Entwicklung von Software unterstützen soll. Die wichtigsten Komponenten einer IDE sind ein Sourcecode-Editor, ein Compiler oder/und Interpreter, ein Linker, ein Debugger sowie Quelltextformatierungsfunktionen.

Index:
In der Informatik wird dieser Ausdruck verwendet für:
Eine Datenstruktur zur schnellen Auffindung von Datensätzen in Datenbanken (Datenbankindex). ODER
Eine Positionsangabe innerhalb strukturierter Speicherbereiche mittels eines nicht negativen Skalars.

Inkrement:
Vom lat. *incrementare* (vergrößern) abgeleitet bezeichnet dieser Begriff in Programmiersprachen eine Operation, bei der der Wert einer Variablen auf den nächst höheren geändert wird.

Instanzierung/Instanzieren:
In der OOP beschreibt dieser Begriff das Erzeugen eines Objekts einer bestimmten Klasse.

Interpreter:
Im Sinne der Softwaretechnik bezeichnet ein Interpreter ein Computerprogramm, das ein anderes Programm ausführt. Im Gegensatz zu einem Compiler oder Assembler wird dabei der Code nicht in eine andere, ausführbare Sprache (Maschinencode) übersetzt und in einer Datei gespeichert, sondern zur Laufzeit selbst eingelesen, analysiert und ausgeführt.

IRC (Internet Relay Chat):
IRC ist ein rein textbasiertes Chat-System, das hauptsächlich für Gruppengespräche in Diskussionsforen, sogenannten Channels (Gesprächskanälen) entworfen wurde, aber auch Gespräche zwischen zwei Teilnehmern erlaubt. IRC findet im Internet statt, gehört aber nicht zum WWW, weshalb zur Teilnahme auch ein spezielles Programm, der IRC-Client anstatt des Browsers verwendet wird.

ISP (Internet Service Provider):
Ein ISP (übersetzt Internetdienstanbieter) ist ein Unternehmen, das Benutzern i.d.R. gegen Entgelt Zugang und andere technische Leistungen im Zusammenhang mit Internet bietet.

Iteration:
In der Informatik spricht man von Iteration bei schrittweisem, bzw. wiederholtem Zugriff auf Datenstrukturen. In der Praxis handelt es sich dabei um Arrays, Listen, Maps, Hashes oder Sets, die mit for-, foreach- oder do-while- Schleifen durchlaufen werden.

Jakarta-Projekt:
Das Jakarta-Projekt entwickelt, wartet und unterstützt Open Source Software unter der Apache Lizenz für die Java-Plattform unter der Aufsicht der Apache Software Foundation.

J2EE (Java 2 Platform Enterprise Edition):
J2EE ist die Spezifikation einer primär in der Programmiersprache Java erstellten Standardarchitektur, auf deren Basis verteilte, mehrschichtige Anwendungen aus modularen Softwarekomponenten und vordefinierten Diensten entwickelt werden können.

Jakarta Taglibs:
Ein Projekt der Apache Software Foundation, das neben einer Referenzimplementierung der JSTL auch noch weitere Tag-Libraries umfasst.

JavaScript:
JavaScript ist eine objektorientierte Skriptsprache, deren Haupteinsatzgebiet im Webbereich liegt. Der Code ist in den Quelltext der Webseiten integriert und wird vom Browser interpretiert. Das bringt den Vorteil mit sich, dass der Webserver nicht für jede Änderung eine neue Seite an den Client schicken muss.

JIT (Just in Time)-Compiler:
Der Just in Time (auf Deutsch *gerade rechtzeitig*)-Compiler übersetzt den Bytecode erst zur Laufzeit und bei Bedarf in nativen Maschinencode. Er ist zur Beschleunigung der Programmausführung gedacht und ein Teil moderner Virtueller Maschinen.

JSF (Java Server Faces):
Java Server Faces ist ein auf Java basierendes Webframework, das die Entwicklung von UIs unter Berücksichtigung des MVC-Modells für Java EE Applikationen erleichtern soll. Es beinhaltet ein ähnliches Konzept wie die WebControls von ASP.NET, das RAD von Web-Anwendungen ermöglicht.

JSP (Java Server Pages):
JSP ist eine von Sun Microsystems entwickelte Technologie, die Programmierern die dynamische Generierung von HTML, XML und anderen Dateitypen als Antwort auf einen Client Request ermöglicht. Java-Code und vordefinierte JSP-Aktionen können in den statischen Inhalt eingebettet werden und werden später von einem speziellen JSP-Compiler in Java-Quellcode umgewandelt.

JSTL (Java Server Pages Standard Tag Library):
JSTL ist eine vom JCP entwickelte Komponente der J2EE Webentwicklungsplattform, die die JSP-Spezifikation durch Hinzufügen von Tag-Libraries für typische Aufgaben wie Schleifen, Bedingungen, Internationalisierung, etc. erweitert.

Konfiguration:
Im Zusammenhang mit Softwareprogrammen bezeichnet Konfiguration das Festlegen bestimmter Einstellungen, die das Verhalten und die Performance der Applikation beeinflussen.

Konstruktor:
In der objektorientierten Programmierung ist der Konstruktor eine spezielle Methode einer Klasse, die dazu verwendet wird, Objekte derselbigen zu erstellen. Meist haben Konstruktoren denselben Namen wie ihre Klassen selbst. Sie werden bei der Erstellung eines neuen

Objekts automatisch aufgerufen und prüfen zuerst die Erfüllung der Vorbedingungen, um sicherzugehen, dass das entstehende Objekt auch die Invarianten seiner Klasse erfüllt.

Laufzeit:
In der Informatik bezeichnet Laufzeit die Zeitspanne, während der ein Programm im Arbeitsspeicher ausgeführt wird.

Laufzeitumgebung:
Die Laufzeitumgebung ist eine Softwareschicht zwischen der Applikations- und der Betriebssystemschicht. Vom Programm benötigte Grundfunktionen werden von ihr zur Verfügung gestellt und sie ermöglicht dadurch eine Plattformunabhängigkeit der Programme.

Maschinencode:
Der Begriff Maschinencode bezeichnet eine Reihe von Instruktionen in Form von Bitfolgen, die für die CPU eines Computers direkt verständlich, für den Menschen dafür aber kaum lesbar sind. Da jedes CPU-Modell über einen eigenen Befehlssatz verfügt und nicht alle Modelle komplett kompatibel sind, ist diese Form des Codes maschinenabhängig.

MIT-Lizenz:
Die MIT-Lizenz (oft auch mit dem Namen X-Lizenz oder X11-Lizenz bezeichnet) ist eine aus dem Massachusetts Institute of Technology stammende Lizenz für die Benutzung verschiedenster Arten von Computersoftware. Die Wiederverwendung der unter dieser Lizenz stehenden Software ist unabhängig davon, ob der Code frei einsehbar ist oder nicht, erlaubt.

ML (Markup Language):
Siehe Auszeichnungssprache

Multithreading:
Multithreading bezeichnet das „zeitgleiche“ Abarbeiten mehrerer Threads. Bei softwareseitigem Multithreading ist nur ein Prozessor beteiligt und die scheinbare Gleichzeitigkeit wird in Wirklichkeit nur durch geschickte Programmierung und Timeslicing vorgetäuscht.

.NET:
.NET ist Microsofts Implementierung des CLI-Standards. Die Basis für Entwicklungen bilden eine virtuelle Laufzeitumgebung, ein Framework von Klassenbibliotheken und angeschlossene Dienste. Ähnlich wie Java-Programme liegen Programme in .NET nicht in Maschinencode, sondern in einem Zwischencode vor und benötigen eine Laufzeitumgebung.

Objektrelationales Mapping:
Die meisten Datenquellen sind heute in Form von relationalen Datenbanken organisiert, während man objektorientierte Programmiersprachen verwendet. Daraus entsteht die Notwendigkeit eines objektrelationalen Mappings, das die Objekte mit deren Attributen und Beziehungen auf die Tabellenstruktur und Spalten abbildet.

Observer Pattern:
Das Observer Pattern ist ein von der GoF beschriebenes Design Pattern aus der Softwareentwicklung und ist auch unter dem Namen *publish-subscribe* bekannt.
Ein oder mehrere Objekte (auch als Observer oder Listener bezeichnet) werden registriert oder registrieren sich selbst, um dann ein Event zu überwachen, das von dem überwachten Objekt (dem Subjekt) ausgelöst werden kann. Das Subjekt wiederum führt eine Liste der

Observer, die benachrichtig werden müssen, wenn ein Event auftritt, sowie Methoden zum Hinzufügen neuer und Entfernen alter Listener.
Ein typisches Einsatzgebiet dieses Patterns ist in das Überwachen eines externen Events (i.d.R. eine Benutzeraktion wie bspw. ein Mausklick) in der ereignisorientierten Programmierung.

OOP (Objektorientierte Programmierung):
OOP ist ein Programmier-Paradigma, bei dem zusammengehörige Daten und die damit arbeitende Programmlogik zu Einheiten – den so genannten Objekten – zusammengefasst werden. Das Programm umfasst eine Sammlung aus Objekten, durch deren Kommunikation und interne Zustandsveränderungen die Programmlogik umgesetzt wird.

Open Source:
Meist im Zusammenhang mit Computersoftware genannt, darf der englische Ausdruck Open Source (= *Quelloffenheit*) im Sinne der Open Source Definition für alle Programme verwendet werden, die jedermann Einblick in den Quelltext gewähren und die Erlaubnis der Veränderung und Weitergabe dieses Quellcodes in deren Lizenzverträgen beinhalten.

OS (Operating System):
Ein OS (auf Deutsch Betriebssystem) ist eine grundlegende Software, die die Hard- und Softwarekomponenten eines Computers verwaltet. Es übernimmt elementare Aufgaben wie Speicherverwaltung, Steuerung der Ein- und Ausgabegeräte, Ausführung von Programmen und vieles mehr.

Page Rank:
Dieser von Larry Page (daher auch der Name Page Rank) und Sergey Brin an der Stanford University entwickelte Algorithmus bewertet und gewichtet eine Menge verlinkter Dokumente wie zum Beispiel das WWW anhand ihrer Struktur. Heutzutage von Google als Grundlage zur Bewertung von Seiten verwendet, verfolgt das Verfahren zwei Grundprinzipien: Je mehr Links auf die Seite verweisen, desto mehr Gewicht hat diese Seite. Je höher das Gewicht der verweisenden Seiten ist, desto größer ist der Effekt. Dies soll einen zufällig durch das Netz surfenden Benutzer nachbilden. Die Wahrscheinlichkeit, dass er auf eine gewisse Webseite stößt korreliert mit dem Page Rank.

Parameter:
siehe Argument

Parser:
Abgeleitet vom engl. *to parse (= analysieren)* bzw. vom lat. *pars (=Teil)*.
Analysiert und überprüft das Skript vor der Abarbeitung auf die korrekte Syntax und reicht Teile an den Prozessor weiter.

Patch:
In der EDV bezeichnet der Begriff Patch (= *flicken*) ein kleines Programm für Korrekturen oder Updates einer Software. Das kann sein, um Sicherheitslücken zu schließen, die Benutzerfreundlichkeit oder Performance zu verbessern, Fehler zu beheben oder neue Funktionen einzubinden.

PDA (Personal digital Assistant):
Ein tragbarer Computer, der ursprünglich hauptsächlich zur persönlichen Kalender-,

Adress- und Aufgabenverwaltung gedacht war, dessen Fähigkeiten heutzutage aber weit vielfältiger sind. Neuere PDAs können bereits als Mobiltelefon, Webbrowser oder Media Player verwendet werden.

PDF(Portable Document Format):
PDF bezeichnet ein offengelegtes, aber dennoch kommerzielles, von der Firma Adobe Systems entwickeltes Dateiformat zur plattform- und auflösungsunabhängigen Darstellung zweidimensionaler Dokumente.

PEAR (PHP Extension and Application Repository):
PEAR ist ein von der Community angetriebenes und der PEAR Group überwachtes, 1999 gegründetes Projekt, das als Framework und Verteilungssystem für PHP Codekomponenten dient. Die Ziele dieses Projekts sind die Bereitstellung einer strukturierten Codebibliothek, die Förderung eines standardisierten Programmierstils und die Erhaltung eines Systems zur Verteilung von Code.

Persistenz:
Ein Begriff aus der Informatik, der die Fähigkeit bezeichnet, Datenstrukturen in nichtflüchtigen Speichermedien wie Dateisystemen oder Datenbanken zu speichern.

PHP (PHP Hypertext Processor):
PHP ist eine Open Source Skriptsprache mit einer an Java bzw. Perl angelehnten Syntax, die hauptsächlich zur Erstellung dynamischer Webseiten oder serverseitiger Webanwendungen verwendet wird.

Plugin:
Plugin (vom engl. to plug in = *einstöpseln, anschließen*) ist die gängige Bezeichnung für eine Applikation, die mit einem anderen Programm über eine definierte Schnittstelle interagiert, um eine bestimmte, meist genau abgegrenzte Funktion zur Verfügung zu stellen.

POJO (Plain old Java object):
Dieser im September 2000 von Martin Fowler, Rebecca Parsons und Josh MacKenzie kreierte Begriff bezeichnet „ganz normale“ Objekte, die sich von komplexen, durch externe Abhängigkeiten belasteten Objekten wie bspw. EJBs durch ihre Unabhängigkeit und dadurch leichte Wart- und Wiederverwendbarkeit unterscheiden.

Polymorphie:
Dieses aus dem griechischen entlehnte Wort (übersetzt *Vielgestaltigkeit*) bezeichnet ein Programmierkonzept, das es ermöglicht, ein Objekt mehreren unterschiedlichen Datentypen zuzuordnen.

Portierung:
„Unter Portierung versteht man bei der Softwareentwicklung den Vorgang, ein Computerprogramm, das unter einem bestimmten Betriebssystem, einer Betriebssystemversion oder Hardware abläuft, auch auf anderen Betriebssystemen, anderen Versionen oder anderer Hardware lauffähig zu machen. Teilweise wird unter Portierung auch der Wechsel von einer Programmiersprache auf eine andere verstanden.“
(http://de.wikipedia.org/wiki/Portierung, 05.06.2006)

Presentation Abstraction Control:
Presentation Abstraction Control (Darstellung Abstraktion Kontrolle) ist der Name eines

Entwurfsmusters zur Strukturierung interaktiver Software. Es teilt ein System in 2 Richtungen auf: Einmal ähnlich wie MVC in die drei Einheiten grafische Benutzerschnittstelle (Presentation), Vermittlung und Kommunikation (Control) und das Datenmodell (Abstraction) und darüber hinaus noch hierarchisch in unterschiedliche Teile (sogenannte Muster Agenten), die jeweils einen Teil der Aufgaben des Systems anbieten.

Primärschlüssel:
In einer Relationalen Datenbank dienen Schlüssel zur eindeutigen Identifikation eines Datensatzes einer Tabelle. Der Primärschlüssel wird aus allen in Frage kommenden Schlüsselkandidaten als der bevorzugte ausgewählt.

Proprietäre Software:
Nach den Regeln der Free Software Foundation ist dieser Begriff zur Bezeichnung jeder Software geeignet, die nicht den Regeln der Foundation für freie Software entspricht. In diesem Fall hält ein Individuum oder eine Firma die exklusiven Rechte an einer Software und der Zugang zum Quelltext sowie das Recht des Kopierens, Veränderns, der Weitergabe oder des Studierens der Software bleibt anderen, solange sie diese Rechte nicht erwerben, verwehrt.

Prototyping:
Im Zusammenhang mit Softwareentwicklung bezeichnet Prototyping einen bestimmten Entwicklungsprozess. Nach der Ermittlung der Anforderungen wird dabei sehr schnell eine erste lauffähige Version des Endprodukts (ein Prototyp) entwickelt und dem Benutzer präsentiert. Basierend auf dessen Feedback wird die Applikation dann verbessert und erweitert und der Entwicklungs-Evaluierungszyklus startet von vorne. Diese Vorgangsweise hat den Vorteil, dass die ersten Ergebnisse schnell verfügbar sind und der Endnutzer sehr bald Feedback geben kann. Innerhalb mehrerer Entwicklungszyklen wird das Produkt so schrittweise optimiert.

Prozedurale Programmierung:
Ein Programmierparadigma, bei dem ein Programm in kleinere Teilaufgaben zerlegt wird. Diese werden in sogenannten Prozeduren, die jederzeit zur Laufzeit von anderen Prozeduren oder sogar von sich selbst aufgerufen werden können, gelöst. Dieser Ansatz zielt darauf ab, Quelltexte wieder verwendbar zu machen, Codeduplizierung zu vermeiden und das Programm besser zu strukturieren.

Provider:
siehe ISP

Prozessor:
Abgeleitet vom lat. *procedere (=voranschreiten)*. Eine Software die Input zu Output verarbeiten kann.

Quellcode (Source Code):
Ein für Menschen lesbarer Text, der einen Ausschnitt einer in einer Programmiersprache geschriebenen Applikation darstellt.

Query:
Vom lat. *quaerere (= fragen, suchen)* abgeleitet ist das der englische Begriff für eine Anfrage an ein Datenbanksystem.

RDBMS (Relationales Datenbankmanagementsystem):
Ein Datenbankmanagementsystem, das auf dem relationalen Datenbankmodell basiert. (siehe Relationale Datenbank, DBMS).

Relation:
Verwendet im Zusammenhang mit relationalen Datenbanken bezeichnet dieser Begriff eine zweidimensionale Tabelle zur Speicherung von Datensätzen.

Relationale Datenbank:
Eine relationale Datenbank ist eine Datenbank, die auf dem relationalen Datenmodell basiert. Dabei werden die Daten mithilfe zweidimensionaler Tabellen verwaltet, die über Schlüssel miteinander logisch verknüpft sind.

Rich-Client:
siehe Fat-Client

RSS:
RSS ist eine Familie von XML-basierten Dateiformaten, die es dem Benutzer ermöglichen, über Neuigkeiten einer Webseite automatisch informiert zu werden. Die mit dieser Technologie erzeugte Datei nennt man RSS-Feed, Webfeed, RSS-Stream oder RSS-Channel. Sie enthält Zusammenfassungen des Contents gemeinsam mit Links zu den vollständigen Artikeln und andere Metadaten. Die Abkürzung RSS hat abhängig von der technischen Spezifikation unterschiedliche Bedeutung (siehe Abkürzungsverzeichnis).

RUDI (Read Update Delete Insert):
siehe CRUD

Semantik:
In der Semantik beschäftigt man sich mit dem Sinn und der Bedeutung von sprachlichen Ausdrücken basierend auf den Regeln der Syntax.

Server:
In der EDV bezeichnet der Begriff Server ein Computersystem, das anderen Computern – so genannten Clients – Dienste über ein Netzwerk zur Verfügung stellt.

Server Pages:
Server Pages sind Skripts, die in HTML eingebunden werden. Bevor die Seite an den Browser des Clients geschickt wird, werden diese am Server abgearbeitet.

Servlets:
Kombination der beiden Begriffe „Server“ und „Applet“. Servlets sind das serverseitige Pendant zu Applets, also Java-Klassen, deren Instanzen innerhalb eines Webservers Anfragen von Clients entgegennehmen und beantworten.

Singleton:
Im Bereich der Softwareentwicklung ist Singleton der Name eines von der GOF entdeckten Entwurfsmusters, das sicherstellt, dass von einer Klasse nur genau ein Objekt bzw. nur genau eine vorher festgelegte Anzahl an Objekten erzeugt werden kann.

Skriptlet:
Eine Technologie, bei der Code von Skriptsprachen wie ASP, PHP oder JSP in HTML eingebettet wird.

Skriptsprache:
Skriptsprachen sind Programmiersprachen, die ursprünglich zur Umsetzung einfacher und kleiner Aufgaben gedacht waren. Daher wurde auf manche Sprachelemente, deren Nutzen erst bei größeren Projekten zum Tragen kommt, bewusst verzichtet. Heutzutage gibt es einige weit entwickelte Skriptsprachen und sie sind in zahlreichen Gebieten der Programmierung professionell im Einsatz.

Smart-Client:
siehe Fat-Client

SOAP (Simple Object Access Protocol):
SOAP (der Nachfolger von XML-RPC) ist ein Protokoll zum Austausch von XML-basierten Nachrichten und zur Durchführung von Remote Procedure Calls über ein Computernetzwerk, meist unter der Verwendung von HTTP und TCP.

Software Engineering:
Software Engineering ist die Tätigkeit des strukturierten Entwickelns und Wartens von Software unter Berücksichtigung der Kenntnisse aus der Informatik, dem Projektmanagement, der Mathematik und vielen anderen Zweigen.

SQL (Structured Query Language):
SQL ist eine Datenbanksprache für relationale Datenbanken. Sie hat sich mit ihrer relativ einfachen Syntax mittlerweile als Quasi-Standard zum Erstellen, Manipulieren, Auslesen und Löschen von Datensätzen aus relationalen Datenbanken etabliert.

Stylesheet:
In der Informationstechnik ist Stylesheet eine Beschreibungssprache, die das Aussehen von durch Auszeichnungssprachen gekennzeichneten Elementen in einem Dokument bestimmt. Diesem Konzept liegt die Idee der Trennung von Daten und Darstellung zugrunde.

Syntax:
Im Bereich der EDV bzw. besonders im Kontext mit Programmiersprachen bezeichnet der Begriff Syntax die Summe aller Regeln bzgl. erlaubter, reservierter Wörter sowie ihrer Parameter und der korrekten Wortanordnung innerhalb eines Ausdrucks.

Tag:
Im Zusammenhang mit Auszeichnungssprachen bezeichnet der Begriff Tag durch Kleiner- und Größerzeichen abgeschlossene Sprachelemente, die sowohl zur Formatierung von visuellen Elementen als auch zur Klassifizierung und Strukturierung der Seite eingesetzt werden.

Taglib/Tag Library:
Taglibs (Kurzform für Tag-Libraries) gehören zur JSP-Spezifikation und bestehen aus einer Sammlung von Tag-Klassen (= Java-Klassen, die eine bestimmte Schnittstelle implementieren) sowie einer TLD, die jedem Tag die zuständige Klasse zuordnet und auflistet, welche Attribute das Tag bietet. In der JSP können diese in der Taglib definierten Tags durch XML-Notation in der Form `<libraryname:tagname attribut1="wert_attribut1" attribut2="wert_attribut2"/>` integriert werden. So ist es möglich, Java-Code (=Logik) weitgehend aus der JSP (=Darstellung) auszulagern.

Template-Engine:
Ein Programm, das den Input (das Template = eine Vorlage) verarbeitet, indem es nach bestimmten Platzhaltern sucht und diese durch dynamische Inhalte aus externen Quellen ersetzt. Das Konzept kann mit Seriendruckfeldern in der Textarbeitung verglichen werden und wird in der Programmierung zur Trennung von Design und Logik verwendet.

Thin-Client:
Dieser Ausdruck (wörtlich übersetzt *dummer Kunde*) bezeichnet einen Computer als Endgerät (Terminal) eines Netzwerkes, dessen funktionale Ausstattung auf die Ein- und Ausgabe beschränkt ist. Die Daten (und teilweise sogar das Betriebssystem) werden dabei möglichst vollständig vom Server bezogen.

Thread:
Threads sind eine Möglichkeit, ein Programm in mehrere (pseudo)gleichzeitig laufende Stränge aufzuteilen Der Unterschied zu einem Prozess liegt im Wesentlichen in der Art der Erstellung und der Ressourcenteilung untereinander.

Tiobe Programming Community Index:
Dieser monatlich erstellte Index versucht, Aussagen über die Popularität von Programmiersprachen zu machen. Berechnet wird er über die Anzahl der Treffer der am meisten verbreiteten Suchmaschinen (im Moment Google, MSN, Yahoo! Web search und die Google Newsgroups) zu den Themen „Programmierer“, „Kurse“ und „Drittverkäufer“.

TLD (Tag Library Descriptor):
In Zusammenhang mit Java Server Pages bezeichnet dieser Begriff eine XML-konforme Meta-Beschreibungsdatei, die eigene Markup-Elemente (Tags) samt ihrer Attribute definiert und mit entsprechenden serverseitig ausgeführten Klassen assoziiert.

UI (User Interface):
Das User Interface, auf Deutsch Benutzerschnittstelle, ist ein Überbegriff für sämtliche Maßnahmen, die dem Benutzer die Interaktion mit einer Maschine, einem Gerät, einem Computerprogramm oder einem anderen komplexen System ermöglichen. Einerseits sind das Wege zur Eingabe, die es dem User ermöglichen, das System zu kontrollieren und andererseits Mittel zur Ausgabe, über die das System mit dem Benutzer kommuniziert.

UML (Unified Modeling Language):
Die UML ist eine von der Object Management Group entwickelte und standardisierte Sprache für die Modellierung von Software und anderen Systemen.

URI (Uniform Resource Identifier):
Ein URI wird im Internet und dort vor allem im WWW eingesetzt, um Ressourcen durch eine nach einer festgelegten Syntax aufgebauten Zeichenkette anzusprechen. URLs waren ursprünglich die einzige Art von URIs, weshalb die beiden Begriffe häufig als Synonyme verwendet werden.

URL (Uniform Resource Locator):
Eine nach einem Standardformat zusammengesetzte Zeichenkette, die eine Ressource im Internet über ihren primären Zugriffsmechanismus und ihren Ort im Netzwerk identifiziert.

Validierung:
In der Softwaretechnik bezeichnet Validierung die Kontrolle eines der Applikation übergebenen Wertes nach Kriterien wie Datentyp, Wertebereich, Wertemenge und Ähnlichem.

Vererbung:
Ein Konzept der OOP, bei dem eine abgeleitete Klasse die Methoden und Eigenschaften von ihrer Basisklasse übernimmt. So kann die abgeleitete Klasse die bereits vorhandene Programmlogik wieder verwenden.

W3C (World Wide Web Consortium):
W3C ist ein internationales Gremium zur Standardisierung und Entwicklung von Techniken rund um das World Wide Web.

WAP (Wireless Application Protocol):
WAP ist der Überbegriff für eine Sammlung von Technologien und Protokollen, deren Ziel es ist, Inhalte aus dem Internet für die langsamere Übertragungsrate und die längere Antwortzeiten im Mobilfunk sowie für die kleinen Displays der Mobiltelefone verfügbar zu machen.

Webcontainer:
Ein Webcontainer bezeichnet die logische Komponente der J2EE-Architektur, die eine Laufzeitumgebung für Servlets und JSPs zur Verfügung stellt. Diese verwaltet die Sicherheit, den (gleichzeitigen) Zugriff, Transaktionen, sowie Lebenszyklen der Servlets und JSPs.

Weblog:

siehe Blog

Webserver:
Ein Webserver bezeichnet im engeren Sinne eine Software, die für die Entgegennahme von HTTP-Requests seitens der Clients und für die Bereitstellung von Webseiten an diese verantwortlich ist. Im weiteren Sinne wird der Begriff Webserver auch für den Host verwendet, der den Server-Dienst anbietet.

Webservice:
Laut W3C ist ein Webservice ein Softwaresystem, das die Interaktion zweier Enduser über ein Netzwerk unterstützt. Dies geschieht durch den Austausch XML-basierter Nachrichten über internetbasierte Protokolle.

WML (Wireless Markup Language):
WML ist eine XML-basierte Seitenbeschreibungssprache, die als Teil der WAP-Spezifikation zur Darstellung veränderlicher Inhalte auf Mobiltelefonen entwickelt wurde.

WSDL (Web Service Description Language):
WSDL ist eine plattform-, programmiersprachen- und protokollunabhängige XML-Spezifikation zur Beschreibung von Web Services.

WWW (World Wide Web):
Das World Wide Web ist ein globales Hypertext-System, auf das man mit verschiedensten mit dem Internet verbundenen Geräten zugreifen kann.

WYSIWYG (What you see is what you get):
Dieser Begriff beschreibt Softwaresysteme, in denen der Inhalt während des Editierens schon genauso (oder zumindest ähnlich) angezeigt wird, wie später bei der endgültigen Ausgabe. Typische Anwendungsbereiche sind Textverarbeitungsprogramme oder auch Content Management Systeme.

XHTML (Extensible Hypertext Markup Language):
Dieser W3C-Standard ist eine Neuformulierung von HTML4 mit XML als Sprachgrundlage. XHTML-Dokumente halten sich also an die Syntaxregeln von XML.

XML (Extensible Markup Language):
XML ist eine vom W3C definierte Metasprache, d.h. sie ist ein Standard zu Definition von beliebigen in ihrer Grundstruktur jedoch stark verwandten Auszeichnungssprachen. XML definiert die Regeln für den Aufbau maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur. Für einen genauen Anwendungsfall müssen die Strukturelemente und ihre Anordnung innerhalb des Dokumentbaums jedoch noch spezifiziert werden.

XML-RPC (Extensible Markup Language - Remote Procedure Call):
Eine Spezifikation, die es Software auf verschiedenen Systemen und unterschiedlichen Umgebungen erlaubt, miteinander über ein TCP/IP basiertes Netzwerk zu kommunizieren. Das Datenaustauschformat ist dabei XML.

XPath (XML Path Language):
XPath ist eine Abfragesprache der XSL-Familie, um Teile eines XML-Dokuments zu adressieren.

XSL (Extensible Stylesheet Language):
XSL ist der Name einer Sprachfamilie zur Erzeugung von Layouts für XML-Dokumente. Zu XSL gehören 3 Spezifikationen: Das XML-basierte eigentliche XSL (zur Unterscheidung XSL-FO genannt) für die Beschreibung eines Dokuments als Baum mit Formatierungsanweisungen und Stilangaben. XSLT wird für die Transformation eines XML-Dokuments in einen anderen Baum verwendet und mittels XPath können beliebige Teile des XML-Baums angesprochen werden.

XSL-FO (Extensible Stylesheet Language – Formatting Objects):
Eine Seitenbeschreibungssprache der XSL-Familie, mittels derer man das Layout der Ausgabe eines XML-Dokumets bestimmen kann.

XSLT (Extensible Stylesheet Language Transformation):
XSLT ist eine Programmiersprache der XSL-Familie, die aufbauend auf der logischen Baumstruktur eines XML-Dokuments die Definition von Umwandlungsregeln erlaubt.

Zend-Engine:
Die nach den Vornamen ihrer Entwickler benannte Zend-Engine wird von PHP als Parser und Compiler benutzt. Erstmals erschienen im Jahr 1999 gemeinsam mit PHP 4, trug sie durch eine Verbesserung der Performance, Erweiterbarkeit und Zuverlässigkeit zu dem Siegeszug der Sprache wesentlich bei. In der aktuellen Version PHP 5.0 kommt die Zend-Engine 2 zum Einsatz. (vgl. http://de.wikipedia.org/wiki/Zend_Engine, 29.03.2006)

Literaturverzeichnis

Fowler, Martin (2003): Patterns für Enterprise Applikation-Architekturen, mitp-Verlag/Bonn

Microsoft. (2002): Application Architecture for .NET – Designing Applications and Services, Microsoft Corporation/USA

Schlossnagle, George (2006): Professionelle PHP 5 – Programmierung**,** Addison-Wesley Verlag/München

Sweat, Jason E. (2005): php|achitect's Guide to PHP Design Patterns, php|architect nanobooks/USA

WWW:

Apache Software Foundation (2006): Offizielle Struts-Webseite
http://struts.apache.org/

Beust, Cédric (2006): Why Ruby on Rails won't become mainstream
http://beust.com/weblog/archives/000382.html

Canada on Rails (2006):
http://www.canadaonrails.com/

Heinemeier Hansson David (OSCON 2005): Secrets behind Ruby on Rails
http://www.itconversations.com/shows/detail658.html

Italy on Rails (2006):
http://italyonrails.com/

Jackson, Simon (2006): Ruby on Rails – ein echtes Framework
http://www.zdnet.de/builder/program/0,39023551,39141885,00.htm

Kotek, Brian (2006): MVC design pattern brings about better organization and code reuse
http://builder.com.com/5100-6386-1049862.html

Moore, Jeff (2006): PHP-Web Application Component Toolkit
http://www.phpwact.org/pattern/model_view_controller

MSDN (2006): Model-View-Controller
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/ImpFrontControllerInASP.asp

MSDN (2006): ASP.NET
http://msdn.microsoft.com/asp.net/

O'Hanley John (2006): Struts seems mediocre
http://www.javapractices.com/Topic193.cjp

PHP (2006): Offizielle PHP-Webseite
http://www.php.net/

PHPbar (2006): php::bar – Wissen für Einsteiger und Profis
http://www.phpbar.de/w/Hauptseite

PHPit (2006): Taking a look at ten different PHP frameworks
http://www.phpit.net/article/ten-different-php-frameworks/1/

PHPwact (2006): PHP MVC Frameworks
http://www.phpwact.org/php/mvc_frameworks

Pragmatic Studio (2006):
http://studio.pragprog.com/

RailsConf2006 (2006): The first official international Rails conference
http://railsconf.org/

Ruby (2006): Offizielle Ruby-Webseite
http://www.ruby-lang.org

RubyForge (2006):
http://rubyforge.org/

Ruby on Rails (2006): Offizielle RoR-Webseite
http://www.rubyonrails.org/

Rustad Aaron (2006): Ruby on Rails and J2EE: Is there room for both?
http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE

Schiwchtenberg, Holger (2006): ASP.NET Developer – Die deutsche Community-Site für Webprogrammierung mit Active Server Pages .NET (ASP.NET)
http://www.aspnetdev.de/

Stepken, Guido (2006): Open Source Entwicklungen und ihre Dynamiken http://www.little-idiot.de/his/design-pattern.htm

The PHP Group (2006): Offizielle PHP-Webseite
http://www.php.net/

The PHP Group (2006): Offizielle Smarty-Webseite
http://smarty.php.net/

Tiobe Software (2006): TIOBE Programming Community Index for April 2006
http://www.tiobe.com/index.htm?tiobe_index

webhostlist.de (2006):
http://www.webhostlist.de/

Wikimedia Foundation Inc (2006): Wikipedia
http://www.wikipedia.org/

Zend – the PHP Company, Matt Zandstra (2006): PHP 5 and Design Patterns - An Introduction http://www.zend.com/php/design/patterns1.php

Zoop Group (2006): Offizielle Zoop-Webseite
http://zoopframework.com/ss.4/6940/zoop.html

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
CGI	Common Gateway Interface
CIL	Common Intermediate Language
CLI	Common Language Interface
CLR	Common Language Runtime
CMS	Content Management System
COM	Component Object Model
CPL	Common Public License
CPU	Central Processing Unit
CRUD	Create Read Update Delete
CSS	Cascading Stylesheet
DB	Database, Datenbank
DBMS	Database Management System, Datenbankmanagementsystem
DBS	Database System, Datenbanksystem
DOM	Document Object Model
ECMA	European association for standardizing information and communication systems
EDV	elektronische Datenverarbeitung
EJB	Enterprise Java Beans
eRb	Embedded Ruby
GOF	Gang of Four
GPL	General Public License
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IIS	Internet Information Server
IRC	Internet Relay Chat

ISP	Internet Service Provider
J2EE	Java 2 Platform Enterprise Edition
JCP	Java Community Process
JSF	Java Server Faces
JSP	Java Server Pages
JSTL	Java Server Pages Standard Tag Library
LISP	List Processing
MIT	Massachusetts Institute of Technology
ML	Markup Language
MSDN	Microsoft Developer Network
MSIL	Microsoft Intermediate Language
MVC	Model View Controller
ODBC	Open Database Connectivity
OOP	Objektorientierte Programmierung
OS	Operating System
OSCON	O'Reilly Open Source Convention
PC	Personal Computer
PDA	Personal digital Assistant
PDF	Portable Document Format
PEAR	PHP Extension and Application Repository
PHP	PHP Hypertext Preprocessor (ursprünglich Personal Home Page Tools)
POJO	Plain Old Java Object
RAD	Rapid Application Development
RCP	Rich Client Platform
RDBMS	Relational Database Management System, Relationales Datenbankmanagementsystem
RoR	Ruby on Rails
RSS	Really Simple Syndication, Real-time Simple Syndication (RSS 2.0) Rich Site Summary (RSS 0.91, RSS 1.0) RDF Site Summary (RSS 0.9, RSS 1.0)
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
Stxx	Struts for transforming XML with XSL

TLD	Tag Library Descriptor
UI	User Interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WSDL	Web Service Description Language
WWW	World Wide Web
WYSIWYG	What you see is what you get
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XML-RPC	XML-Remote Procedure Call
XPath	XML Path Language
XSL	Extensible Stylesheet Language
XSL-FO	Extensible Stylesheet Language – Formatting Objects
XSLT	Extensible Stylesheet Language Transformation
XSP	Extensible Server Pages
ZPL	Zoop Public License