Fabian Ebert, Björn von Prollius, Andreas Retter


Mit einer Einführung in Ajax und Ruby on Rails.

Projektarbeit im Sommersemester 2007 bei Prof. Walter Kriha
im Studiengang Medieninformatik an der Hochschule der Medien Stuttgart.

# 1 Web 2.0 - Merkmale, Komponenten und soziale Aspekte

# Inhaltsverzeichnis

## 1.1 Einleitung

„*Web 2.0 is the network as platform, spanning all connected devices; Web 2.0 applications are those that make the most of the intrinsic advantages of that platform: delivering software as a continually-updated service that gets better the more people use it, consuming and remixing data from multiple sources, including individual users, while providing their own data and services in a form that allows remixing by others, creating network effects through an „architecture of participation," and going beyond the page metaphor of Web 1.0 to deliver rich user experiences.*" (Tim O'Reilly[1])

„*Web 2.0 is an attitude not a technology.*" (Ian Davis[2])

Was ist *Web 2.0*? Diese Frage beschäftigt seit fast zwei Jahren die Internet-Community, die Blogosphäre, die Kritiker und die Experten. Tim O'Reilly, dessen Verlag der Begriff zugeschrieben wird, hat im September 2005 einen Artikel gleichen Namens veröffentlicht[3], um ein für alle Mal sein Verständnis von *Web 2.0* festzuhalten. Von vielen wird diese Veröffentlichung als eine Art Legaldefinition des Begriffs verstanden - und doch wird er von vielen anderen so interpretiert, wie es ihnen gerade vorteilhaft erscheint. Dazu kommen noch diejenigen, die den Begriff vollständig oder teilweise ablehnen.

Die gewollt scharfe Beschreibung einer unscharfen Thematik hat notwendigerweise für Kontroversen gesorgt. Die mit dem Begriff angedeutete Versionierung will eine Ansammlung von Neuentwicklungen auf den Punkt bringen, dabei gibt es keinen Relaunch eines neuen Internet, sondern eine stetige Entwicklung, die 2005 in das Bewusstsein der Medien und damit der Öffentlichkeit gelangte. Als medienwirksamer Begriff dieses Bewusstseins wird heute *Web 2.0* verwendet.

Wie unscharf das *Web 2.0* zu fassen ist, zeigt sich auch in der berühmt gewordenen Visualisierung des Begriffs, der so genannten *Web 2.0* Mindcloud[4].

In dieser Arbeit sollen einige Aspekte des *Web 2.0* erklärt werden. Dieser erste Teil widmet sich dem Begriff selbst, sowie den Komponenten, die zum Verständnis von *Web 2.0* geführt haben. Im zweiten und dritten Teil werden einzelne Techniken detailliert erläutert: das Ruby on Rails Framework als serverseitige Entwicklungsbasis für Web-Applikationen, und AJAX als vorwiegend clientseitige Technologie für interaktive Web-Anwendungen.

[1] http://radar.oreilly.com/archives/2005/10/web_20_compact_definition.html
[2] http://internetalchemy.org/2005/07/talis-web-20-and-all-that
[3] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
[4] s. Abb. 1.1 von Markus Angermeier, http://kosmar.de/archives/2005/11/11/the-huge-cloud-lens-bubble-map-web20/


Abbildung 1.1: Web 2.0 Mindcloud.

## 1.2 Begriffsgeschichte

Geprägt wurde der Begriff durch die „Web 2.0 Conference“[5], die gemeinsam vom internationalen Computerbuch-Verlag O’Reilly und dem Konferenzveranstalter CMP Technology (früher: MediaLive International) erstmals im Oktober 2004 und seitdem jährlich in San Francisco ausgerichtet wird. Dale Dougherty, Vizepräsident von O’Reilly, hat den Begriff *Web 2.0* als Schlagwort für die Entwicklerkonferenz ersonnen.

Die in den Jahren 2000/2001 geplatzte Dotcom-Blase[6], der Zusammenbruch des Neuen Marktes, empfanden die Veranstalter als Wendepunkt für eine neue Entwicklung im Web. Gegen 2004 hatte sich das Internet erholt, und neue Konzepte und Webapplikationen verbreiteten sich. In der Öffentlichkeit fand der Begriff sofort großen Anklang und wurde auch von anderen Firmen als Marketinginstrument verwendet. Die große Zahl der Blog-Beiträge, Artikel und Kommentare im World Wide Web zeigt, dass das *Web 2.0* - wenn auch kontrovers - wahrgenommen wurde.

[5] http://www.web2con.com/

[6] Tim O’Reilly sieht darin eher eine gesunde Marktbereinigung, http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

## 1.3 Komponenten und soziale Aspekte

Bevor wir uns der engeren Bedeutung des Begriffs *Web 2.0* zuwenden, sollen zunächst einige Begriffe und Techniken erläutert werden, die in diesem Zusammenhang eine Rolle spielen. Zusammengenommen sind dies die Komponenten, die den Internetbenutzern das Gefühl eines *Web 2.0* vermitteln. „...there's something qualitatively different about today's web", sagt Tim O'Reilly[7] dazu.

### 1.3.1 Weblogs (Blogs)

Weblogs, heutzutage fast nur noch mit der Kurzform Blog bezeichnet, sind aus online geführten Tagebüchern hervorgegangen. Es handelt sich dabei um Webseiten, auf denen regelmäßig oder zumindest in zeitlich nicht allzu langen Abständen neue Artikel erscheinen. Dabei veröffentlicht der Autor (Blogger) meistens persönliche Ansichten oder Kommentare zu aktuellen Ereignissen. Blogs können ein bestimmtes Thema verfolgen oder auch nur tagebuchartig persönliche Erlebnisse schildern. Inzwischen werden Blogs auch von Firmen und Medien genutzt, um Nachrichten zu verbreiten.

Die meisten Blogs bieten eine Kommentarfunktion für die Leser. Daraus können regelrechte Dialoge oder Diskussionen entstehen, was normalerweise auch gewünscht ist. Außerdem können Blogs nachvollziehbar Bezug auf Artikel in anderen Blogs nehmen. Das so entstehende soziale Geflecht aus Artikeln, Kommentaren und Vernetzungen wird als Blogosphäre bezeichnet.

Erste Blogs entstanden Mitte der 1990er Jahre, der Begriff Weblog wurde 1997 geprägt. Wesentlich zur Verbreitung von Blogs beigetragen haben (neben der Verbreitung von Breitband-Internetzugängen) *Weblog Publishing Systeme*. Diese webbasierten Dienste dienen zum Erstellen und Verwalten von Blogs und existieren als bereitgestellter Online-Service eines Dienstleisters (Blog Hosting) oder in Form von Software, die auf einem eigenen Webspace installiert werden kann. Bekannte Blog-Hosting-Services sind z.B. „Xanga" oder „Blogger"[8]. Sie bieten jedem Nutzer ohne weitere Vorkenntnisse die Möglichkeit, ein Blog zu erstellen, zu betreiben, und in einem gewissen Rahmen auch optisch anzupassen. Verbreitete Weblog-Software ist beispielsweise „WordPress" oder „Movable Type". Solche, in einer Skriptsprache wie PHP oder Perl geschriebenen und mit einer SQL-Datenbank arbeitenden Systeme können vielfach kostenfrei (teilweise Open Source) bezogen und vom Endanwender auf einem eigenen Webspace installiert werden. Benutzer haben dadurch nicht nur die Möglichkeit, ihren Blog auf einer eigenen Domain zu betreiben, sondern haben auch mehr Eingriffs- und Gestaltungsmöglichkeiten durch Verwenden von Templates für das Blog-Layout und von Modulen für erwei-

[7] http://radar.oreilly.com/archives/2005/08/not_20.html

[8] http://www.xanga.com/, http://www.blogger.com/; Blogger wird inzwischen von Google betrieben.


Abbildung 1.2: O'Reilly Radar, das Business-Blog des O'Reilly-Verlages.

terte Funktionen. Nichtsdestotrotz sind diese auf Blogs spezialisierten Content-Management-Systeme, die über den Webbrowser bedient werden, in der Bedienung sehr einfach gehalten und dafür auch für technisch weniger versierte Benutzer leicht zu verwenden.

Blogs sind ursprünglich und auch heute noch hauptsächlich ein Textmedium. Allerdings haben sich andere multimediale Formen gefunden, die teilweise eigene Bezeichnungen tragen. Beispielsweise werden in *Photoblogs* regelmäßig Fotografien gepostet und gegebenenfalls vom Urheber und den Blog-Besuchern kommentiert. Ähnliches gilt für Videoclips in *Vlogs* (Video-Weblogs). Die Qualität reicht hier von Schnappschüssen oder Filmen mit der Handy-Kamera bis hin zu professionellen oder künstlerischen Aufnahmen. *Moblogs* (Mobile Weblogs) zeichnen sich durch die Veröffentlichung der Beiträge per Mobiltelefon oder PDA aus.

Audioblogs, blogmäßige Veröffentlichungen von Tonbeiträgen im mp3-Format (selten auch in einem anderen Tonformat), sind vor allem unter einem anderen Namen berühmt geworden: *Podcasts* (iPod Broadcasts). Der bekannte mobile MP3-Player von Apple, iPod, trug zur Verbreitung dieser häufig sendungsartig konzipierten Beiträge bei. Podcasts sind aber nicht auf das Abspielen auf iPods beschränkt, sondern können auf jedem anderem Gerät abgespielt werden, das das Dateiformat unterstützt. Auch Video-Podcasts sind seit der Verbreitung videofähiger mobiler Abspielgeräte zunehmend beliebt, existierten aber tatsächlich auch schon davor. Technisch gesehen besteht der Unterschied von Audio- und Videoblogs zu Podcasts darin, dass letztere per Webfeed[9] abonnierbar sind und so mit passender Software automa-

[9] vgl. Abschnitt 1.3.5 Webfeeds

tisch auf dem Endgerät aktualisiert werden können.

Für die dauerhafte Erreichbarkeit von Blogartikeln werden so genannte *Permalinks* (permanent links) verwendet. Es handelt sich dabei um eine eindeutige URL, unter der ein spezifischer Artikel auch dann noch erreichbar ist, wenn er von der Startseite eines Blogs in das Blog-Archiv verschoben wurde. Mit Permalinks spiegelt sich der Urgedanke des Internets, online verfügbarer Hypertext, besonders in den Blogs wieder. Im Gegensatz zu herkömmlichen Webseiten werden bei Blogs die Artikel normalerweise auch für unbestimmte Zeit archiviert und bleiben damit immer über den Permalink erreichbar. Permalinks werden meistens in leicht lesbarer Form gewählt (z.B. nach Datum sortiert: http://blog.example.com/2006/06/18/eintrag1/). Permalinks verbreiteten sich ab dem Jahr 2000 stark und werden von aktuellen Weblog Publishing Systemen automatisch angelegt. Sie werden auch in Webfeeds verwendet, um auf die entsprechenden Artikel zu verweisen.

Ein wichtiges Instrument für blogübergreifende Diskussionen sind *Trackbacks*. Ein Trackback wird von einem Blogartikel verwendet, der auf einen anderen Blogartikel verweist oder inhaltlich Bezug nimmt. Dabei integriert der verweisende Artikel eine Trackback-URL des verwiesenen Artikels, wodurch in letzterem automatisch der Verweis (meistens mit einer Zusammenfassung des verweisenden Artikels) vermerkt wird. Dies setzt voraus, dass beide Weblog Publishing Systeme das Trackback-Protokoll[10] unterstützen. Die Trackback-Funktionalität beruht letztendlich auf dem Austausch von XML-Nachrichten. Einen vergleichbaren, aber einfacheren Mechanismus bietet Pingback[11]. Ein Pingback verzichtet auf Zusatzinformationen wie die Zusammenfassung des verweisenden Artikels und kann daher automatisiert ohne weitere Benutzereingaben erfolgen.

Zusätzlich zu der Kommentar-Funktion, die es jedem Leser eines Blogs erlaubt, direkt Kommentare zu einem Blogartikel zu hinterlassen, wird so die Möglichkeit geschaffen, blogübergreifende und nachvollziehbare Diskussionen zu führen. Trackbacks haben sich zu einem prägenden Merkmal für die soziale Vernetzung der Blogs zu einer Blogosphäre entwickelt.

Ein weiteres Vernetzungsinstrument zwischen Blogs ist die *Blogroll*. Diese Liste von Links zu anderen Blogs wird meistens auf der Startseite eines Blogs untergebracht. Hierbei kann es sich um Blogs handeln, die der Autor selbst liest oder für lesenswert hält, die in thematischem Zusammenhang mit seinem Blog stehen, oder die ihrerseits einen Verweis auf seinen Blog enthalten. Einige Weblog Publishing Systeme können die Webfeeds von anderen Blogs auf Aktualisierungen überprüfen und Blogs, die vor kurzer Zeit aktualisiert wurden, in der Blogroll hervorgehoben darstellen.

Für Blogs gibt es inzwischen eigene Suchmaschinen. Auch Google hat einen Suchdienst für

[10] Trackback wurde 2002 von Six Apart Ltd., dem Hersteller des Weblog Publishing Systems „Movable Type“ eingeführt und heute in einer Arbeitsgruppe mit dem Ziel weiterentwickelt, eine Standartisierung bei der IETF (Internet Engineering Task Force) zu erreichen. http://www.movabletype.org/trackback/

[11] http://www.hixie.ch/specs/pingback/pingback

Blogs eingerichtet[12]. Die umfassendste Suchseite ist aber „Technorati“[13] mit knapp 45 Millionen beobachteter Blogs. Die Wichtigkeit eines Blogs oder Artikels wird ähnlich wie bei Googles PageRank-Algorithmus durch die Anzahl von externen Links auf die gesuchte Seite maßgeblich beeinflusst. Da Aktualität bei Blogs eine ganz wichtige Rolle spielt, wird auch der Suchmaschinenindex sehr zeitnah aktualisiert. Eine Anmeldung mit einer Art Pingback-Mechanismus ist bei Technorati möglich, um die Suchmaschine automatisch über neue Artikel auf dem Laufenden zu halten.

Laut Technorati werden täglich 1,2 Millionen neue Artikel in Blogs gepostet, und es entstehen jeden Tag 75.000 neue Blogs[14].

### Graswurzel-Journalismus und Gegenöffentlichkeit

Durch die schnellen und einfachen Veröffentlichungsmechanismen haben sich Blogs zum geeigneten Instrument für Graswurzel-Journalismus entwickelt. Die Bürger nehmen aktiv an der Recherche, Analyse und der Verbreitung von Nachrichten und Informationen teil. Sie gestalten selbst bzw. werden selbst zu Medien. Zwar kann hier leicht der Vorwurf der mangelnden Professionalität fallen, andererseits ist der normale Mensch als Informationsquelle zunächst einmal unabhängig von Medienkonzernen. Was sich dadurch ergibt, ist eine Gegenöffentlichkeit, in der Informationen, die von den großen oder staatlichen Medien bewusst oder unbewusst verschwiegen werden, unkontrolliert und unzensiert verbreitet werden können.

Der Einfluss, den die Gegenöffentlichkeit der Blogosphäre hat, ist nicht zu unterschätzen. Populäre Themen wurden in der Vergangenheit bereits von den etablierten Medien aufgegriffen, weil sie im Web diskutiert wurden. Durch massenweise Verlinkung können einzelne Artikel einen enormen Stellenwert im Web bekommen und so in den Suchmaschinen leicht gefunden werden. Der Meinungsbildungsprozess läuft an den traditionellen Medien vorbei und wird aus der virtuellen in die reale Welt transportiert. Dies ist natürlich auch kritisch zu sehen. Quellen im Internet sind nicht so leicht auf ihre Authentizität überprüfbar, und auch hier droht die Gefahr einer bewussten oder unbewussten Falschmeldung. Die Masse zeigt sich jedoch kritisch gegenüber unglaubwürdigen Nachrichten, und Reputation hat einen hohen Stellenwert unter ernsthaften Bloggern, so dass nicht leichtfertig mit dieser Freiheit umgegangen wird.

Als prominente Beispiele für den Graswurzel-Journalismus sind Katastrophen der jüngeren Vergangenheit zu nennen. Aufnahmen und Berichte vom Geschehen rund um das Seebeben im Indischen Ozean am 26. Dezember 2004 oder den Terroranschlägen am 7. Juli 2005 in London wurden frühzeitig im World Wibe Web verbreitet.

[12] http://blogsearch.google.com/
[13] http://www.technorati.com/
[14] http://www.technorati.com/about/

### 1.3.2 Wikis

Wikis sind kollaborative Webseiten, auf denen jeder Benutzer Artikel veröffentlichen und bearbeiten kann. Das Prinzip der Wikis - einfaches Erstellen und Editieren von Webseiten für jeden Benutzer - hat in ähnlichem Maß wie bei Blogs zu einer großen Popularität von Wikis geführt. Dabei werden Wiki-Seiten mit einer sehr einfachen Auszeichnungssprache im Webbrowser bearbeitet und von der auf dem Server laufenden Wiki-Software in HTML umgewandelt. Bei der Wiki-Software handelt es sich letztlich um ein leicht zu bedienendes Content Management System. Das Seitenlayout wird durch Templates bestimmt. Die Vernetzung verschiedener Artikel durch Hyperlinks ist auch hier eine zentrale Eigenschaft und trägt zur Bildung eines Gesamtkontextes bei. Einzelne Wiki-Artikel werden mit ihrer Veröffentlichung sofort im Wiki (auf der Webseite) sichtbar. Bei den meisten Wikis wird aus Prinzip keine Inhaltskontrolle durch Administratoren vorgenommen, da letztendlich jeder Leser auch die Inhalte überprüfen kann und dazu aufgefordert ist, Fehler zu korrigieren. Außerdem können durch versionierte Archivierung der Seiten die Inhalte einer vorhergehenden Version einfach wieder hergestellt werden.

Wikis enthalten häufig frei verfügbaren und urheberrechtlich nicht geschützten Content, der von den Benutzern unter eine freie Lizenz gestellt wird. Normalerweise beschäftigen sie sich mit einem bestimmten Thema oder verfolgen eine bestimmte Ausrichtung.

Das erste Wiki („WikiWikiWeb“[15]) wurde vom amerikanischen Programmierer Ward Cunningham erdacht und programmiert. Es sollte als Hypertext-Plattform zum einfachen Ideenaustausch zwischen Programmierern dienen und daher eine schnelle Bearbeitungsfunktion für die einzelnen Webseiten bieten. Von Cunningham stammt auch die Bezeichnung - *wiki* ist das hawaiianische Wort für schnell. Das WikiWikiWeb ging 1995 online.

Das bekannteste Wiki-Projekt ist „Wikipedia“, eine Online-Enzyklopädie, die in verschiedenen Sprachen[16] existiert. Durch gemeinschaftliche Arbeit sind so 4,3 Millionen Artikel veröffentlicht worden, davon knapp 1,2 Millionen in Englisch und über 415.000 in Deutsch[17]. Wikipedia zählt als Beweis dafür, dass die Wiki-Prinzipien funtionieren: kollaboratives Arbeiten einer großen Zahl von Benutzern sorgt für eine hohe Qualität der Artikel, da jeder Leser gleichzeitig auch Artikel korrigieren und ergänzen kann. Das Wikipedia-Projekt wurde 2001 begonnen und wird seit 2003 von der Wikimedia Foundation Inc., einer Non-Profit-Organisation mit Sitz in Florida, betrieben. Wikimedia betreibt auch noch weitere Wiki-Projekte wie „Wikimedia Commons“ (multimediale Sammlung), „Wiktionary“ (Wörterbuch) und „Wikiquote“ (Zitatesammlung). Zum Einsatz kommt dabei seit 2002 die eigens entwickelte Open-Source-Software MediaWiki, die in PHP geschrieben ist und mit einem MySQL-Datenbank-Backend arbeitet.

[15] http://c2.com/cgi/wiki

[16] Wikipedia gibt es in über 200 Sprachen, davon in 14 Sprachen mit mehr als 50.000 Artikeln. http://en.wikipedia.org/wiki/Wikipedia.

[17] http://www.wikipedia.org/

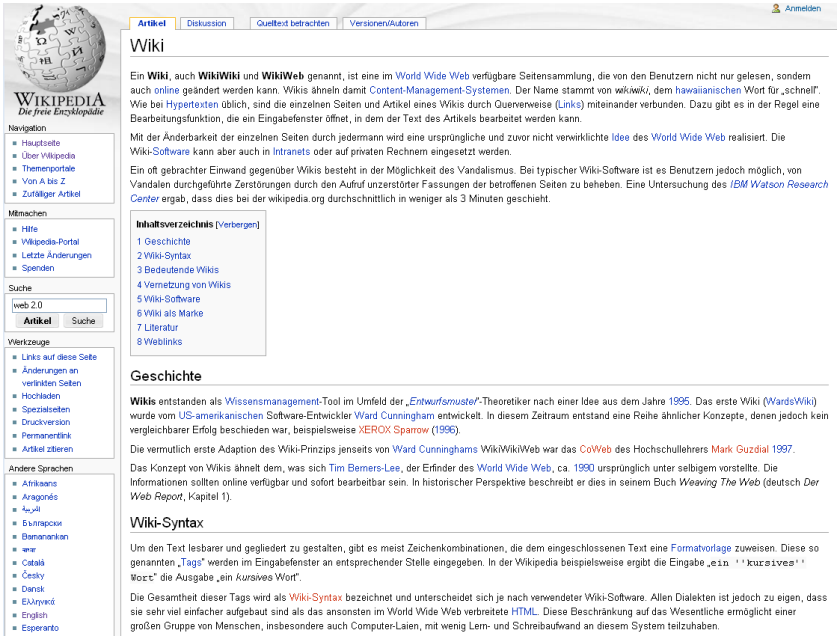


Abbildung 1.3: Ein Artikel in der Wikipedia, einer Online-Enzyklopädie im Wiki-Format.

Eine andere bekannte Wiki-Software ist das weit verbreitete, da leicht zu installierende „UseModWiki“ (1999 in Perl und ohne relationale Datenbank entwickelt), mit dem auch die „Wikipedia“ vor der Entwicklung von „MediaWiki“ betrieben wurde. Wegen seiner guten Versions- und Zugriffskontrolle ist „TWiki“ (2000, Perl, keine Datenbank) für den Einsatz in Firmen sehr beliebt.

### 1.3.3 Soziale Software

Soziale Software wurde durch die Verbreitung von PCs und insbesondere schnellen Internetzugängen in jüngerer Zeit ein Massenphänomen. Schon früh wurde das Internet zur Knüpfung sozialer Kontakte oder der Suche nach Verwandten oder gar vermissten Personen benutzt.

Vorläufer der heutigen webbasierten sozialen Software sind die textbasierten, in Echtzeit ablaufenden Dienste. Internet Relay Chat (IRC, ab 1988) ist auf eine Kommunikation mehrerer Teilnehmer ausgerichtet, während Instant Messaging (IM, ab 1996) primär den direkten Nachrichtenversand von einem Teilnehmer zu einem anderen ermöglicht. Die bekanntesten IM-Programme sind die im Besitz von AOL befindlichen ICQ (20 Million aktive Benutzer) und AOL Instant Messenger (53 Millionen) einerseits sowie MSN Messenger (29 Millionen) und Yahoo! Messenger (21 Millionen) andererseits. Über Kontaktlisten bildet sich hier jeder Benutzer sein eigenes soziales Netz. Durch Weiterentwicklung der ursprünglichen Prinzipien sind heute auch der Austausch von Dateien und multimediale Kommunikation (Internettelefonie, Videokonferenzen) möglich - ein beliebter Vertreter dieser Spielart ist die Voice-over-IP-


Abbildung 1.4: Beim Business-Netzwerk OpenBC können nur angemeldete Benutzer fremde Profile einsehen.

Software Skype.

Der erste Webseite zur Bildung sozialer Netzwerke, „classmates.com“, ging bereits im Jahr 1995 online. „classmates.com“ ermöglicht das Wiederfinden von ehemaligen Freunden aus der Schulzeit. Mit über 40 Millionen Benutzern ist es der umfangreichste Dienst dieser Art. „classmates.com“ betreibt inzwischen auch „StayFriends“[18], einen populären deutschen Dienst mit dem gleichen Ziel und ca. 3,6 Millionen Benutzern. Die Finanzierung dieser Dienste wird über Werbung, Kooperationsverträge und nicht zuletzt durch Benutzungsgebühren für erweiterte Funktionalitäten erreicht.

Sehr beliebt sind soziale Netzwerke, die auf Freundeskreis-Prinzipien oder dem *Kleine-Welt-Phänomen* beruhen. Der soziologische Begriff des Kleine-Welt-Phänomens wurde 1967 vom amerikanischen Sozialpsychologen Stanley Milgram geprägt und besagt, dass durch persönliche Beziehungen in sozialen Netzwerken jeder soziale Akteur mit jedem anderen über eine im Verhältnis zur Gesamtmenge der Akteure überraschend kurzen Kette von Bekanntschaften verbunden ist[19].

Das aus Deutschland stammende und vor allem in Europa und Asien verbreitete Business-

[18] http://www.stayfriends.de/

[19] Milgram überprüfte das Phänomen experimentell, in dem er die Teilnehmer Postsendungen an ihnen unbekannte Zielpersonen versenden ließ. Dabei durften die Teilnehmer die Sendung ausschließlich an ihnen bekannte Personen weitergeben. Im Ergebnis waren 80% der so geknüpften Personenketten nicht länger als vier, und fast keine Kette länger als sechs Personen. Das Ergebnis ist jedoch bis heute umstritten, da die Veröffentlichung durch Milgram nicht einwandfrei war. Für einen Einstieg in das Thema s. http://de.wikipedia.org/wiki/Kleine-Welt-Ph%C3%A4nomen

Netzwerk „openBC“[20] (1 Million Benutzer) basiert auf diesem Phänomen und und belegt es gleichzeitig. Durch den Aufbau eines eigenen Kontaktkreises bekannter Personen gewinnt der Benutzer hier eine hohe Anzahl sekundärer Kontakte (Schneeballprinzip), und zu jeder unbekannten Kontaktperson kann ein Bekanntschaftspfad dargestellt werden, der selten über mehr als vier Mittelspersonen läuft. Das System dient zum Knüpfen von beruflichen Kontakten und bietet Angebots- und Nachfragefunktionen sowie themenbezogene Diskussionsgruppen. Auch hier ist die Basisbenutzung kostenfrei und ein komfortables Mehrwertangebot als Abonnentendienst kostenpflichtig.

Bekannte Freundeskreisdienste sind „Friendster“[21] (29 Millionen Benutzer) oder „Facebook“[22] (für Studenten, 9 Millionen Benutzer). „MySpace“[23] ist derzeit der beliebteste soziale Netzwerkdienst mit über 84 Millionen registrierten Profilen. „MySpace“ rangiert auf Platz 5 der weltweit meistbesuchten Webseiten[24] und bietet einen kombinierten Dienst, bestehend aus einem persönlichem Profil, Blog, Foto- und Filmuploads, Benutzergruppen und einem internen Mail-System. Die Seiten können vom User optisch angepasst und mit Seiten von befreundeten Benutzern verknüpft werden.

Ebenfalls an Bedeutung zugenommen haben Online-Singlebörsen. Acht Prozent der deutschen Internetnutzer geben an, ihren Partner über das Internet kennengelernt zu haben[25]. Das Internet gehört zu den wichtigsten Instrumenten bei der Partnersuche. Unzählige Dienste wie „Neu.de“ oder iLove[26] haben aus dieser Nachfrage ein Geschäftsmodell entwickelt und vermitteln Bekannstschaften zwischen den zahlenden Mitgliedern.

Die Kritik an sozialen Webdiensten geht vor allem in zwei Richtungen. Aus der Innensicht gibt es bei vielen Diensten kaum Möglichkeiten, um zu überprüfen, ob die in den Benutzerprofilen angegeben Daten echt sind. Somit lassen sich auch die veröffentlichten Inhalte häufig nicht realen Personen zuorden, was bei strafrechtlich relevanten Veröffentlichungen zu Problemen bei der Strafverfolgung führt. Insbesondere offene Massendienste wie „MySpace“ sind durch die große Anzahl an sogenannten Fakes (gefälschte Identitäten) und sogar durch Kontaktversuche pädophiler Personen mit den meist jugendlichen Benutzern[27] aufgefallen. Es wurden beispielsweise auch zahlreiche Profile im Namen bekannter Persönlichkeiten wie Schauspielern oder Musikern angelegt. Kommerzielle Dienste genießen hier gegenüber offenen Varianten den Vorteil, dass die Anzahl an Fakes sehr gering ist. Ähnliches gilt für Netzwerke, die auf eine echte persönliche Kontaktaufnahme spezialisiert sind, wie z.B. openbc.com.

[20] http://www.openbc.com/
[21] http://www.friendster.com/
[22] http://www.facebook.com/
[23] http://www.myspace.com/
[24] http://www.alexa.com/site/ds/top_sites?ts_mode=global&lang=none
[25] http://www.tns-emnid.com/pdf/presse-presseinformationen/2003/2003_02_14_TNS_Emnid_Partnersuche_Internet.pdf
[26] http://www.ilove.de/
[27] http://www.tagesschau.de/aktuell/meldungen/0,1185,OID5653198,00.html

Aus der Außensicht richtet sich die Kritik an den mangelnden Selbstdatenschutz der Benutzer sozialer Webdienste mit Ihren persönlichen Daten. Benutzerprofile können oft eingesehen werden, ohne dass sich der Betrachter vorher selbst angemeldet haben muss. In den USA sollen bereits Benutzer, die so öffentlich zweifelhafte Informationen über sich selbst preisgegeben haben, ihren Arbeitsplatz verloren haben. Der amerikanische Geheimdienst NSA soll Forschungsprojekte finanzieren, die sich mit dem automatisierten Sammeln von Daten aus sozialen Webdiensten und die Verknüpfung mit anderen Datenquellen beschäftigen, um so verbesserte Personenprofile und Personennetzwerke erstellen zu können[28].

### 1.3.4 Vernetzung sozialer Software (FOAF, XFN)

Um soziale Netzwerke, die sich aus der Vernetzung von Webseiten ergeben, auch maschinenlesbar abzubilden, wurden Techniken entwickelt, die vor allem bei Blogs zum Einsatz kommen.

#### FOAF

FOAF ist ein Akronym für „Friend Of A Friend“[29] und beschreibt ein maschinenlesbares Format, mit dem die Eigenschaften von Personen (Name, E-Mailadresse, Interessen etc.) sowie die Beziehungen zu anderen Personen beschrieben werden können. FOAF basiert auf XML und RDF. Es dient zur Abbildung von sozialen Netzwerken und ermöglicht darüber hinaus die Suche nach Personen mit bestimmten Eigenschaften. FOAF ist eine Anwendung des semantischen Web[30].

#### XFN

XFN (XHTML Friends Network)[31] ist ein HTML-Mikroformat. Es benutzt ein HTML-Attribut um in Hyperlinks zu befreundeten Webseiten die Beziehung (Freund, Kollege, Partner...) zwischen der verlinkenden und der verlinkten Person maschinenlesbar darzustellen. Die Informationen, die mit XFN vermittelt werden können, sind im Vergleich zu FOAF extrem begrenzt. Dafür ist XFN für weniger technisch versierte Benutzer leicht zu verwenden.

[28] http://www.newscientisttech.com/article.ns?id=mg19025556.200&feedId=online-news_rss20
[29] http://www.foaf-project.org/
[30] s. hierzu auch Abschnitt 1.3.7 Semantisches Web
[31] http://gmpg.org/xfn/

### 1.3.5 Webfeeds (RSS, Atom)

Webfeeds sind meist XML-basierte Dokumente, die von Blogs und Nachrichtenwebdiensten zur Verbreitung neuer Artikel verwendet werden. In einer Feed-Datei können beispielsweise Veröffentlichungsdatum, Überschrift, Zusammenfassung und die Internetadresse von Beiträgen untergebracht werden. Erscheint ein neuer Beitrag auf der Webseite, so wird ein neuer Eintrag in den Webfeed vorgenommen. Es entsteht eine Liste, bei der der neueste Eintrag den ältesten verdrängt - üblicherweise werden die letzten 10 Einträge in der Liste gehalten.

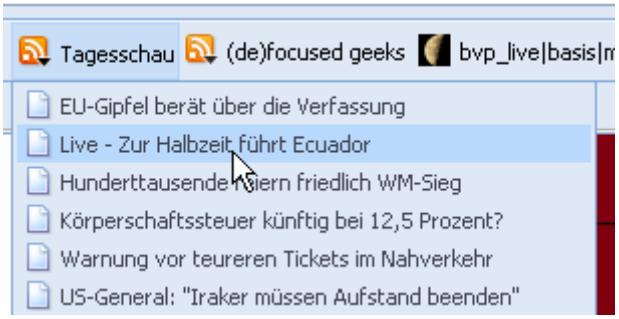


Abbildung 1.5: Bei Mozilla Firefox heißen Feeds „Dynamische Lesezeichen“.

Webfeeds können von sogenannten *Feedreadern* gelesen werden. Feedreader gibt es als einzelne Programme, sie sind aber auch in vielen Webbrowsern und einigen Mailprogrammen integriert. Der Benutzer sieht im Feedreader übersichtlich die zur Verfügung stehenden Artikel des vom ihm abonnierten Feeds. Durch regelmäßiges automatisches Neuladen des Feeds durch den Feedreader wird der Benutzer so stets auf einem aktuellen Stand gehalten, ohne dass er selbst etwas dafür tun muss. Er kann gezielt einzelne Artikel aus dem Feedreader heraus aufrufen, ohne vorher die zugehörige Webseite besucht haben zu müssen.

Feedreader können wiederum auch als webbasierte Dienste arbeiten und so aus mehreren fremden Webfeeds neue Webseiten oder sogar neue Webfeeds dynamisch zusammenstellen. Dies ist möglich, weil die Informationen durch das Datei-Format des Feeds in maschinenlesbarer Form vorliegen und automatisiert weiterverarbeitet werden können. Webfeeds werden hauptsächlich für Blog-Beiträge und Nachrichtenartikel verwendet, finden aber auch vermehrt Anwendung bei multimedialen Diensten wie Audio- und Video-Podcasts. Zwei verbreitete Webfeed-Formate sind *RSS* und *Atom*.

#### RSS

Eine Versionsvielfalt sorgt für etwas Verwirrung und Diskussion im RSS-Umfeld. Die verschieden Versionen des RSS-Dateiformats wurden von unterschidlichen Gruppen entwickelt und sind nicht zueinander kompatibel. Eine zentrale Spezifikation durch das W3C fehlt.

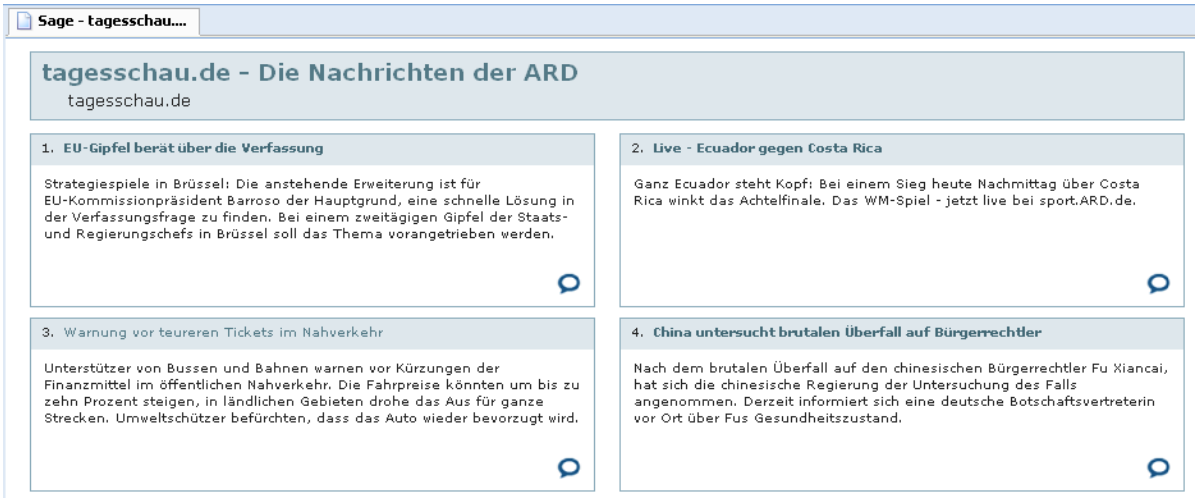


Abbildung 1.6: Eine detailliertere Feed-Ansicht bietet der Feedreader Sage, der als Firefox-Plugin erhältlich ist.

Die erste verbreitete Version war RSS 0.91 (Rich Site Summary). Die Entwicklung aus dem Jahr 1999 stammte von Netscape und basierte auf einer einfachen XML-DTD des Programmierers Dave Winer. Die 0.9x-Versionlinie wurde 2000 vom amerikanischen Softwarehersteller UserLand Software, Winers Firma, weiterentwickelt[32]. Gleichzeitig entwickelte eine unabhängige Entwicklergruppe RSS 1.0 (RDF Site Summary), ein auf dem RDF-Standard des W3C basiertes Format[33].

Im Jahr 2002 veröffentlichte UserLand dann RSS 2.0 (Really Simple Syndication)[34], eine nicht vollständig zu den vorigen 0.9x-Versionen kompatible Weiterentwicklung. Obwohl RSS 2.0 wegen der Kompatibilitätsprobleme nicht unumstritten ist, setzt es sich derzeit durch massiven Gebrauch in Blogging-Software und durch das Angebot bekannter Nachrichtensender und News-Dienste als Standard im Web durch.

Ein Weiterentwicklung von Version 2.0, RSS 3, wurde 2005 von einem israelischen Studenten angestoßen[35], fand aber keine Unterstützung.

### Atom

Das RSS-Format hat einige Mängel und Beschränkungen. Der RSS-2-Standard steht unter urheberrechtlichem Schutz der amerikanischen Harvard University (wo Dave Winer einen

[32] http://rss.userland.com/
[33] http://web.resource.org/rss/1.0/spec
[34] http://blogs.law.harvard.edu/tech/rss
[35] http://www.rss3.org/

Forschungsauftrag hat) und darf nicht ohne Genehmigung verändert werden. Eine Weiterentwicklung ist laut Spezifikation nicht geplant.

Insbesondere bewegt sich RSS nicht vollständig im Rahmen der XML-Spezifikation des W3C und enthält keine Möglichkeit zur Typisierung der transportierten Dateninhalte.

Der bekannte amerikanische Software-Entwickler Sam Ruby initiierte daher 2003 die Diskussion um die Eigenschaften eines neuen Formats für Webfeeds[36]. Eine erste Version, Atom 0.3, wurde Ende 2003 insbesondere durch Einsatz von Google in einigen seiner Services verbreitet. Um eine geordnete Weiterentwicklung zu ermöglichen wurde 2004 die „Atompub working group“[37] innerhalb der IETF (Internet Engineering Task Force) ins Leben gerufen. Atom 1.0 wurde 2005 von der IETF als proposed standard angenommen und im RFC 4287 als Atom Syndication Format veröffentlicht.

Atom genießt damit gegenüber RSS den besseren Ruf als offenes und von einem offiziellen Standartisierungsgremium akzeptiertes Format, dass aktiv weiterentwickelt wird[38]. Die Verbreitung von Atom setzt hingegen nur langsam ein. Viele Seiten bieten parallel RSS- und Atom-Feeds an.

### 1.3.6 Tagging (Gemeinsames Indexieren)

Beim gemeinschaftlichen Indexieren werden mittels sozialer Software einzelne Objekte von Benutzern mit beschreibenden Schlüsselworten (den sogenannten *Tags*) versehen. Als Prinzip dahinter steht die Annahme, dass viele Benutzer ähnliche Objekte mit ähnlichen Tags markieren. Im Ergebnis werden so bei der Suche nach einem Schlüsselwort die passenden Objekte gefunden. Die Software unterstützt dabei den Benutzer durch Vorschläge, damit Tags in möglichst gleicher Schreibweise verwendet werden und vergleichbare Inhalte auch die gleichen Tags erhalten. Die gemeinschaftlich erstellte, unkontrollierte Sammlung von Stichworten wird (in Anlehnung an Taxonomie) Folksonomy genannt. Folksonomies sind unhierarchisch, für den Gebrauch durch Internetbenutzer aber daher sehr einfach zu verwenden. Zwei Webdienste, die Tagging verwenden, sind in letzter Zeit besonders bekannt geworden: „del.icio.us“ und „Flickr“.

„del.icio.us“[39] ist eine Webseite zum öffentlichen Ablegen von Weblinks, ein sozialer Lesezeichenmanager. Benutzer können hier Links auf andere Webseiten unter ihrem Benutzernamen speichern und mit Stichworten versehen. Je mehr Benutzer den gleichen Link mit dem gleichen Tag versehen, desto relevanter wird der Link bei der Suche nach dem Tag. Die Benutzer

[36] http://www.intertwingly.net/blog/1472.html
[37] http://www.ietf.org/html.charters/atompub-charter.html
[38] Ein umfangreicher Vergleich zwischen beiden Formaten findet sich unter http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared.
[39] http://del.icio.us/

können immer sehen, wie viele andere Benutzer den gleichen Link gespeichert haben, und welche Stichworte dafür verwendet wurden.

„Flickr“[40], inzwischen ein Dienst von Yahoo, bietet den gleichen Service für Fotos. „Flickr“ dient als (öffentlicher oder privater) Service zur Ablage und zum Austausch digitaler Fotos und wird zunehmend auch von Bloggern genutzt, um Bilder für ihre Blogs abzulegen. Fotos können gruppiert werden, und auch hier kann der Benutzer die Fotos mit Tags versehen, woraus sich eine Suchmöglichkeit nach Stichworten für alle Benutzer ergibt.

Häufig verwendete Tags werden gerne in einer sogenannten *tag cloud* dargestellt, einer visualisierten Wortliste, in der die Begriffe proportional zur Häufigkeit ihrer Verwendung durch größere oder fettere Schrift hervorgehoben werden. Die Begriffe sind verlinkt und lösen beim Anklicken direkt eine Suche aus.

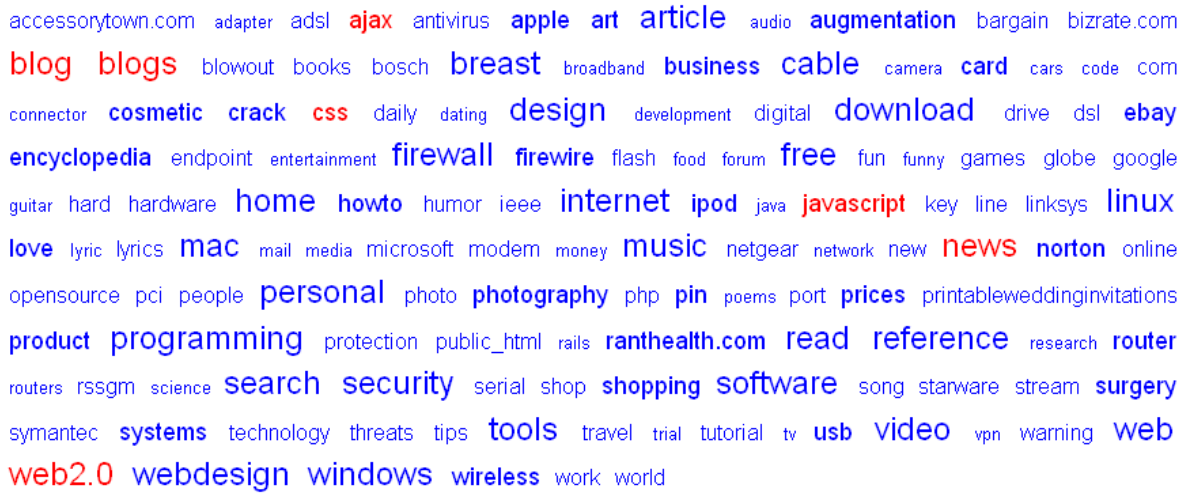


Abbildung 1.7: Eine tag cloud zeigt die am häufigsten verwendeten Tags von del.icio.us.

### 1.3.7 Semantisches Web

Das semantische Web stellt eine Erweiterung des bestehenden World Wide Web dar, die vom World Wide Web Consortium entwickelt und vorangetrieben wird[41]. Webseiten werden mit Metadaten über die inhaltliche Bedeutung (Semantik) versehen, um die maschinelle Lesbarkeit der Seiten zu erhöhen. Hierzu werden Taxonomien (hierarchische Klassifizierungen) und Ontologien (relationale Wissensrepräsentationen) eingesetzt. Als Auszeichnungssprache für Metadaten wird RDF (Ressource Description Framework) auf Basis von XML eingesetzt.

Webseiten können von Computern bisher nur über die Worte, die sie beinhalten, gelesen und

[40] http://www.flickr.com/
[41] http://www.w3.org/2001/sw/

verstanden werden. Beispielsweise Suchmaschinen können die Häufigkeit bestimmter Worte in einem Text feststellen, sie können jedoch den Inhalt nicht semantisch erfassen. Mit den Techniken des semantischen Web könnte ein Programm durch die Metadaten einen Eindruck vom Inhalt der Seite bekommen, ohne sich auf das Vorkommen bestimmter Stichworte im Text verlassen zu müssen.

Da das Auszeichnen von Webseiten mit Metadaten aufwändig ist und die Techniken nicht ohne Vorkenntnisse beherrschbar sind, setzen sich die Ideen des semantischen Web nur langsam durch. Die Bedeutung des semantischen Web geht weit über den hier dargestellten Rahmen hinaus - als Schlagwort dafür ist in neuerdings die Bezeichnung *Web 3.0* zu hören.

### 1.3.8 Browser als Rich Client

Der Webbrowser ist von seiner Konzeption her ein Programm, dass in HTML geschriebene Hypertext-Dokumente von bestimmten Orten (üblicherweise aus dem Internet) herunterladen und anzeigen kann. Schon 1993 konnte einer der ersten weit verbreiteten Browser, NCSA Mosaic, auch Grafiken anzeigen. Über die Jahre kamen viele zusätzliche Funktionen für multimedialen Content wie Video- und Audiodateien hinzu, die teilweise über Plugins realisiert wurden, teilweise auch direkt in die Browser-Funktionen integriert wurden.

Browser entwickeln sich heute mehr und mehr zu anpassungsfähigen und erweiterbaren Programmen. Der inzwischen recht weit verbreitete Mozilla Firefox[42] bietet eine Schnittstelle für Erweiterungen (Extensions), mit der Benutzer selbst zusätzliche Funktionen programmieren und an den Browser anschließen können. Hierbei werden zum einen Funktionalitäten des Browsers selbst ergänzt (z.B. Anzeigeoptionen, Lesezeichen-Verwaltung, erweiterter Feedreader), zum anderen aber auch völlig neue Funktionen hinzugefügt (z.B. Werbefilter, Mausgesten). Mit der Greasemonkey-Extension[43] kann sogar das Verhalten von Webseiten durch vom Benutzer eingebundene Skripte gesteuert werden: ein beliebtes Greasemonkey-Skript fügte Googles Webmail-Applikation einen Löschen-Button hinzu, der dort fehlte.

Der Firefox-Ableger „Flock“[44] hat sich auf die Unterstützung sozialer Webseiten spezialisiert. Unkomplizierter Upload von Bildern bei „Flickr“, einfaches Posten von Blog-Artikeln und ein erweiterter Feedreader machen den Browser zu einer Schaltzentrale für den modernen *Web 2.0*-Benutzer.

Durch Technologien wie AJAX (Asynchronous JavaScript and XML), das in einem eigenen Kapitel ausführlich erklärt wird, wird der Browser zu einer Laufzeitumgebung für Teile von Webapplikationen. Nicht mehr der Webserver allein ist für den Ablauf von webbasierten An-

[42] http://www.mozilla.com/
[43] http://greasemonkey.mozdev.org/
[44] http://www.flock.com/

wendungen verantwortlich, sondern auch der Browser übernimmt Logik-Funktionen. Durch grafische Gestaltung und das Verarbeiten von Maus-Events sind manche Webanwendungen im Browser kaum mehr von klassischen Desktop-Anwendungen zu unterscheiden.


Abbildung 1.8: Passend zur Fußball-Weltmeisterschaft 2006 gibt es ein Firefox-Plugin, dass Spiele, Ergebnisse und Neuigkeiten übersichtlich anzeigt.

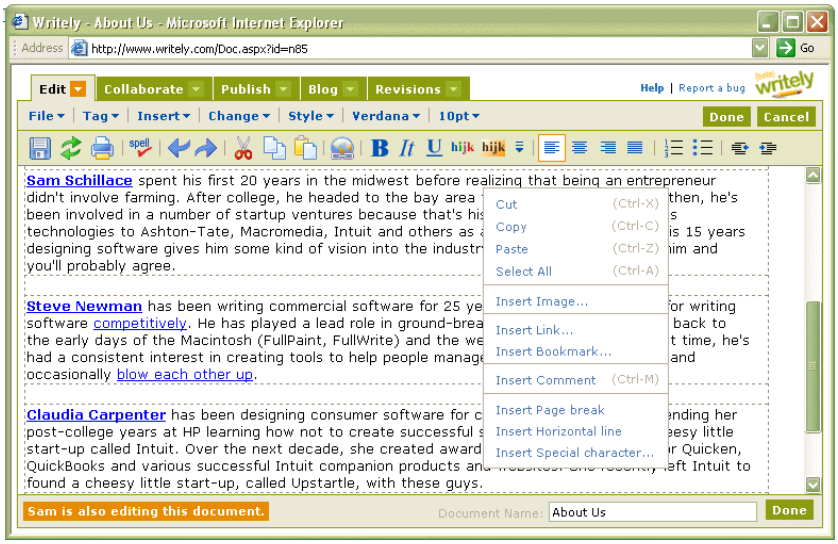


Abbildung 1.9: Die Textverarbeitungssoftware „Writely“ (http://www.writely.com/) läuft komplett im Webbrowser ab.

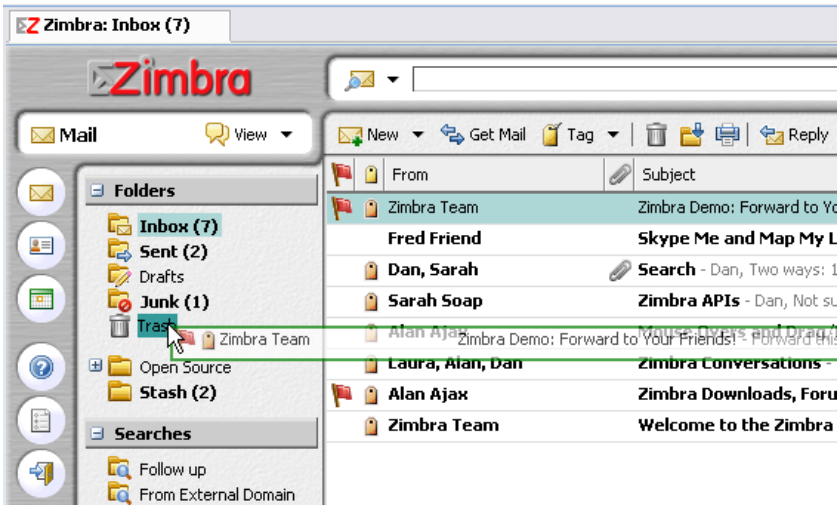


Abbildung 1.10: Der browserbasierte Email-Client von Zimbra (http://www.zimbra.com/) unterstützt „drag and drop“ und fühlt sich damit an wie ein Desktop-Mailprogramm.

## 1.4 Merkmale

„Web 2.0 is a term often applied to a perceived ongoing transition of the World Wide Web from a collection of websites to a full-fledged computing platform serving web applications to end users. Ultimately Web 2.0 services are expected to replace desktop computing applications for many purposes.“[45]

*Web 2.0* ist ein griffiges Label, eine Kategorisierung („Flickr ist *Web 2.0*“), ein Begriff, der seine Berechtigung dadurch gefunden hat, dass er verwendet und vor allem verstanden wird. Eine genaue Definition existiert ebenso wenig wie ein tatsächlich übereinstimmendes Verständnis bei den Verwendern des Begriffs. Die Kernaussage wird jedoch scheinbar von allen gleich verstanden: *Web 2.0* repräsentiert als Begriff die Erkenntnis, dass sich etwas Großes im Internet getan hat[46]. „I know it when I see it“[47], aber die Frage nach einer festen Definition führt zu vielen verschiedenen, inkonsistenten Antworten.

### 1.4.1 Sieben Punkte nach Tim O'Reilly

Im September 2005 veröffentlichte Tim O'Reilly den Artikel „What is Web 2.0“[48], in dem er die Maßstäbe für sein Verständnis des Begriffs ausformulierte und damit der zuvor insbesondere in der Blogosphäre aufgekommenen, teilweise heftigen Kritik begegnete. O'Reilly arbeitete sieben Prinzipien heraus, die für ihn als kennzeichnend und ausschlaggebend sind für die Frage, ob eine Anwendung oder ein Unternehmen „*Web 2.0* ist“:

1. Das Web als Plattform
2. Nutzung kollektiver Intelligenz
3. Daten als nächstes „Intel inside“
4. Abschaffung des Software-Lebenszyklus
5. Lightweight Programming Models
6. Software jenseits der Grenzen einzelner Geräte

[45] Dare Obasanjo, der im übrigen dem Begriff kritisch gegenüber steht. http://www.25hoursaday.com/weblog/PermaLink.aspx?guid=e8e3d501-0c1a-477d-9212-c7366020fa3d

[46] so sinngemäß Tim O'Reilly, http://radar.oreilly.com/archives/2005/08/not_20.html

[47] Dare Obasanjo, http://www.25hoursaday.com/weblog/PermaLink.aspx?guid=e8e3d501-0c1a-477d-9212-c7366020fa3d

[48] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

7. Grafische Benutzerführung (Rich User Experiences)

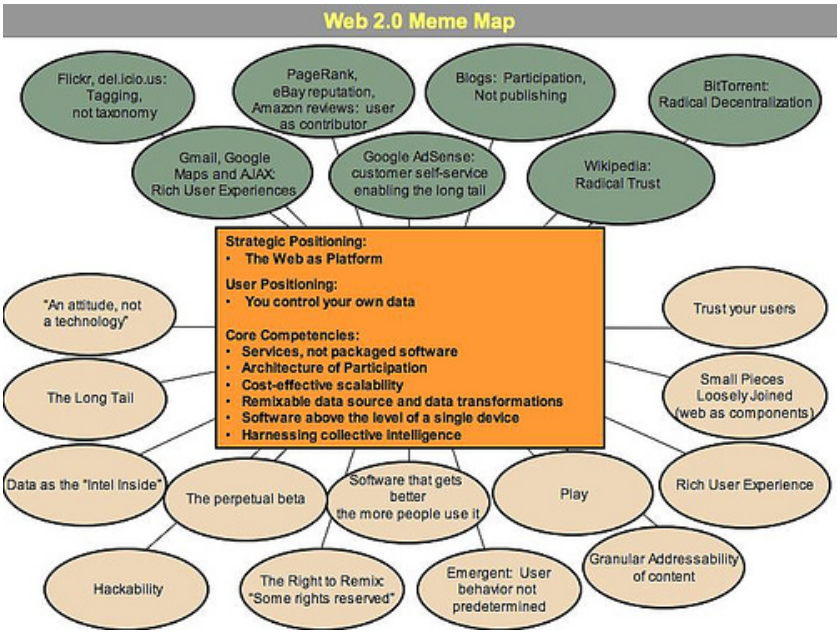


Abbildung 1.11: Auch O'Reilly visualisierte seine Vision vom Web 2.0.

### Das Web als Plattform

Anstatt auf Desktop-Applikationen zu setzen wird das Internet selbst als Plattform für Anwendungen genutzt. Die restlichen Prinzipien spielen in den Plattformgedanken herein: native Webanwendungen werden nicht punktuell veröffentlicht und lizensiert, sondern kontinuierlich gepflegt und einfach benutzt. Der Wert der Anwendung bestimmt sich maßgeblich durch die Daten, die sie verwaltet. Die Beteiligung der Benutzer an der Datenbeschaffung hilft dabei, die Bedürfnisse der breiten Masse abzudecken. Es findet eine radikale Vernetzung und Dezentralisierung der Anwendungen statt, und bestehende werden zu neuen Anwendungen kombiniert („Mash-up"). Eine „Architektur der Beteiligung" bezieht die Benutzer in die Entwicklung mit ein. Die Qualität des Dienstes steigt so mit der Anzahl der Benutzer. Das Internet ist eine offene Plattform. Versuche, durch Monopolisierung der Plattform Erfolge zu erzielen, müssen daher aus Prinzip scheitern.

### Nutzung kollektiver Intelligenz

Das World Wide Web lebt von seinen Benutzern, und wird auch von ihnen mitgestaltet. Sie stellen die kollektive Intelligenz dieses Netzes dar, in dem sie Webseiten miteinander verlinken, Kommentare hinterlassen, Wissen beisteuern, sortieren und organisieren.

Zum Beispiel Googles Erfolg beruht darauf, die (letztendlich von allen Internetbenutzern geschaffene) Link-Struktur zu analysieren, die den gigantischen Hypertext namens World Wide Web bildet. Hier haben inzwischen auch die Blogs, in denen viel verlinkt wird, einen nicht zu unterschätzenden Einfluss.

Genau so nutzen eBay und Amazon die Beiträge ihrer Nutzer, um den Service zu verbessern. Beide Dienste werden durch die zahlreichen Kommentare interessant, die über Produkte oder andere Nutzer abgegeben werden und so den Konsumenten bessere Einschätzungen bei der Kaufentscheidung ermöglichen. eBay überlässt sogar die Kategorisierung und damit die Auffindbarkeit der Produkte den Nutzern und gibt hier nur einen Rahmen in Form vordefinierter Kategorien vor. Die Unternehmen nehmen so eine Mittlerrolle ein und schaffen einen Kontext, in dem die Nutzeraktivität stattfinden kann.

Eines der erfolgreichsten benutzergetriebenen Projekte ist Wikipedia. Die Qualität der enzyklopädischen Artikel wird fast allein durch das Vertrauen in den Benutzer erreicht, qualitativ hochwertige Artikel zu schreiben oder qualitativ minderwertige Artikel zu verbessern. Das Wissen der Masse kommt hier zum Tragen.

Ebenfalls über die Masse von Benutzern funktioniert die Folksonomy, das gemeinschaftliche Indizieren. Dabei können die Tags völlig frei gewählt werden, die dem System zugrunde liegende Software macht allenfalls Vorschläge anhand bereits bestehender Tags. Hierbei wird darauf vertraut, dass eine große Anzahl von Benutzern für ähnliche Einträge ähnliche Tags verwendet.

Ein großer Teil der Infrastruktur des Internets beruht auf Open-Source-Techniken, die ihrerseits durch kollektive Intelligenz weiterentwickelt werden. Erfolg beruht im Internet normalerweise nicht auf Werbung, sondern auf Mundpropaganda und Netzwerkeffekten - ein gewisser Darwinismus ist hier zu spüren: was gut ist, setzt sich durch. Aber es baut sich auch die „Architektur der Beteiligung“ auf, in der der Konsument gleichzeitig Produzent ist: „We, the media“[49], oder für Unix-Kenner: chmod 777 web[50].

### Daten als nächstes „Intel inside“

Daten sind wichtig, nicht Applikationen. Dabei geht es nicht darum, die Daten selbst zusammenzustellen oder zu besitzen, denn Daten können lizensiert und so auch von anderen genutzt werden. Wichtig ist, die Daten zu nutzen, anzubieten und zu ergänzen. Wiederum Amazon ist hier ein Vorreiter, in dem es die offiziellen ISBN-Informationen mit den Daten der Verlage wie Titelbilder und Inhaltsangaben und letztlich auch den Benutzerkommentaren zu einem neuen Datenpool zusammengeführt hat.

[49] Dan Gillmor betitelte so sein Buch über Grasswurzel-Journalismus. vgl. auch http://wethemedia.oreilly.com/
[50] James Snell, http://www-03.ibm.com/developerworks/blogs/page/jasnell?entry=chmod_777_web

Die Daten mit Mehrwert anzureichern kann ein Schlüssel zum Erfolg sein. Über Programmierschnittstellen können die Daten einiger Dienste von Programmierern kostenfrei genutzt und damit zu neuen Diensten kombiniert werden. Ein Beispiel dafür ist „Frappr“[51], das die geographischen Daten von Google Maps verwendet. Benutzer können hier Gruppen anlegen oder beitreten, und deren Wohnorte werden auf der Karte angezeigt. Eine Anbindung an das soziale Netzwerk von „MySpace“ existiert ebenfalls.

Die Wichtigkeit der Daten könnte einerseits zu verstärkten Monopolisierungsversuchen seitens der Unternehmen führen. Auf der anderen Seite sind auch immer mehr Bestrebungen zu verzeichnen, durch entsprechende Lizenzen freie Datenquellen zu schaffen. Als Beispiel sei Wikipedia genannt, deren Artikel von den Benutzern häufig unter die freie Creative-Commons-Lizenz gestellt werden.

### Abschaffung des Software-Lebenszyklus

Das operative Geschäft wird zu einer Kernkompetenz der erfolgreichen Unternehmen im Internet. Webdienste werden teilweise permanent aktualisiert, gewartet und verbessert. Der dauerhafte Beta-Status wird schon fast zu einem Status-Symbol: Google Mail oder Flickr können als ausgewachsene Applikationen betrachtet werden und tragen mit Absicht ein „Beta“ im Logo, um die stetige Entwicklung anzudeuten. Auch besteht so die Chance, unfertige Services anzubieten und diese nach und nach zu erweitern. Der Benutzer akzeptiert einerseits die anfänglichen Unzulänglichkeiten und freut sich über jedes neue Feature. Außerdem wird er in den Entwicklungsprozess mit einbezogen, sei es durch Verbesserungsvorschläge und als kostenloser Software-Tester, vielleicht sogar als aktiver Programmierer. Das Benutzerverhalten kann schon lange vor einer klassischen Fertigstellung am laufenden Programm analysiert und in die weitere Entwicklung mit einbezogen werden. Die Maxime der Open-Source-Bewegung, „veröffentliche früh und veröffentliche häufig“, wird wörtlich genommen: Flickr veröffentlicht neue Versionen seiner Software zeitweise im Halbe-Stunden-Takt[52].

### Lightweight Programming Models

Das Web ist durch seine Einfachheit erfolgreich geworden. Anstatt auf schwergewichtige Web-Service-Frameworks zu setzen sind leichtgewichtige Programmierschnittstellen gefragt. Eine Kooperation ohne Koordination, d.h. beispielsweise ein Verteilen der Daten über leicht zu beherrschende Schnittstellen ohne Blick darauf, was mit den Daten geschieht, führt zu einer uneingeschränkten Verbreitung der Daten. Wer seine Komponenten in wiederverwend-

[51] http://www.frappr.com/
[52] http://blogs.warwick.ac.uk/chrismay/tag/flickr/

barer Form anbietet fördert den Remix-Gedanken[53], die Möglichkeit, aus Bestehendem etwas Neues zu schaffen; und lässt zu, daß sich die Nutzer damit vertraut machen und so durch weite Verbreitung de facto neue Standards entstehen.

### Software jenseits der Grenzen einzelner Geräte

Jede Webapplikation ist per Definition nicht auf ein Gerät zugeschnitten, sondern kann auf jedem Computer oder mobilen Endgerät ablaufen, das einen Browser unterstützt. Unter dem Gesichtspunkt des Web als Plattform kann die Grenze jedoch noch ausgeweitet werden: z.B. Apple verbindet mit seinem digitalen Musikservice „iTunes“ das mobile Abspielgerät (iPod) über den PC des Benutzers (auf dem die iTunes-Software läuft) mit den Servern des Internet (auf dem die Musikstücke als Daten bereitstehen). Es geht hier weniger um die reinen Webapplikationen als solche, sondern um die Nutzung des Web als Service-Plattform. Gerade im Hinblick auf mobile Geräte ist hier noch eine große Entwicklung zu erwarten.

### Grafische Benutzerführung (Rich User Experiences)

Schon früher haben verschiedene Technologien versucht, das Web durch dynamische und interaktive grafische Elemente anzureichern und letztendlich grafische Benutzeroberflächen, wie man sie aus Desktop-Applikationen gewöhnt ist, zur Verfügung zu stellen. Java Applets oder Flash stellten proprietäre Lösungen dar, die nur mit Browser-Plugins realisiert werden konnten. DHTML (dynamisches HTML, die Kombination von HTML, CSS und JavaScript) versuchte dies mit standartisierten Techniken. Über die Standards bekriegten sich jedoch Microsoft und Netscape im Browser-Krieg. Die Standards werden heute vom World Wide Web Consortium oder der Internet Engineering Task Force überwacht - diese Organisationen haben sich eine respektierte Stellung erarbeitet.

Neue Ansätze, um grafische Benutzeroberflächen darzustellen, können daher auf diese Standards bauen. AJAX ist so ein Ansatz und vereint Techniken wie XHTML, CSS, XML und JavaScript, um ein „Look-and-Feel“ von statischen, nicht webbasierten Anwendungen zu erzeugen. AJAX hat sich als Schlüsselkomponente für viele der modernen Anwendungen des *Web 2.0* erwiesen. Die Stärken des Web wie globale Erreichbarkeit und Durchsuchbarkeit lassen sich so mit den interaktiven Benutzeroberflächen verbinden.

Interessant ist gleichzeitig die Beobachtung, dass viele *Web 2.0*-Dienste ein sehr schlichtes Design adaptiert haben. Die Konzentration auf das Wesentliche ist entscheidend für eine intuitive

[53] sehr eindrucksvoll äußert sich zum Remix-Gedanken auch Prof. Edward Felten von der Princeton University in seinem Eröffnungsvortrag „Rip, Mix, Burn, Sue - Technology, Politics, and the Fight to Control Digital Media“: http://www.cs.princeton.edu/~felten/rip/

Bedienbarkeit, und ein weißer Hintergrund wird nicht mehr als leer, sondern als aufgeräumt empfunden.

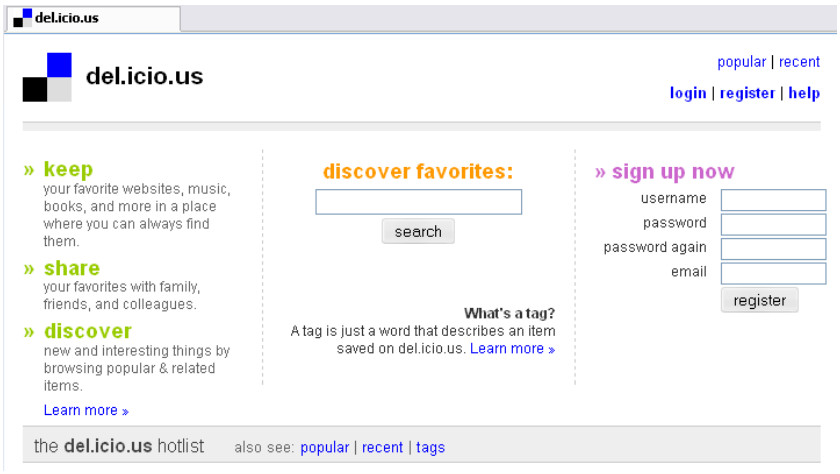


Abbildung 1.12: del.icio.us - Design reduziert auf das Wesentliche.

## 1.4.2 Würdigung

### Kritik

Von Kritikern wurde der Begriff *Web 2.0* aufgrund seiner Unschärfe als Modewort abgetan, das immer dann benutzt wird, wenn der Verwender sich selbst und möglicherweise etwas anderes als cool, neu oder unentdeckt hervorheben möchte[54]. Der Begriff habe keine eigenständige Bedeutung und könne nur anhand von Beispielen umschrieben werden, wobei verschiedenste, nicht kongruente Versionen der Beschreibung existierten. Die Anlehnung an die Versionierung von Software-Produkten sei nicht zutreffend, da es sich bei dem, was der Begriff zu beschreiben versucht, um eine stetige Entwicklung und nicht um einen fixen Meilenstein handele. Im übrigen sei sowieso fraglich, ob es sich bei der aktuellen Entwicklung im Web erst um eine zweite Stufe handelt[55].

Von O'Reilly zitierte Unternehmen wie Google, Amazon oder eBay haben die dargelegten Prinzipien schon lange vor der Kreation des Begriffs *Web 2.0* verfolgt. Aus Kapitalgebersicht hätten selbsternannte *Web 2.0*-Unternehmen zunächst nicht viel zu bieten[56]. Ein erneutes Aufkommen einer Dotcom-Blase sei zu befürchten[57].

[54] Paul Boutin, http://www.slate.com/id/2138951/; Russell Shaw bezieht das - fast wertneutral - auf den O'Reilly-Verlag selbst: http://blogs.zdnet.com/ip-telephony/?p=805
[55] Tim Bray, http://www.tbray.org/ongoing/When/200x/2005/08/04/Web-2.0
[56] Das berühmt gewordene „53,651"-Posting von Josh Kopelman: http://redeye.firstround.com/2006/05/53651.html
[57] John Dvorak, http://www.pcmag.com/article2/0,1895,1931858,00.asp


Abbildung 1.13: Web 2.0: Alter Wein in neuen Flaschen? (Grafik von http://www.alistapart.com/articles/web3point0)

Technologien wie AJAX würden überschätzt. Abgesehen davon, dass sie keine Neuentwicklungen darstellen, bestünden noch heftige Implementierungs- und Usability-Probleme[58]. Auch entgegen der anderslautenden landläufigen Meinung habe Wikipedia keineswegs durch kollaboratives Zusammentragen von Wissen einen brauchbaren Standard erreicht[59].

### Unterstützung

Die weltweit renommierte Wochenzeitung The Economist gibt sich gewohnt zurückhaltend, betitelt *Web 2.0* aber immerhin als „das Enzym, dass sich durchgesetzt hat“[60]. Manche Enthusiasten stimmen dem Begriff *Web 2.0* nicht nur in seiner Gesamtheit zu, sondern glauben eine Entwicklung des Internet zu einem globalen Supercomputer, der „Voraussehenden Maschine“ zu sehen[61]. Eine völlig neue Generation von Software rechtfertige die Versionierung, und das demokratische Prinzip habe die Ausrichtung des World Wide Web grundlegend verändert[62]. Für Wikipedia-Gründer Jimmy Wales ist weniger der technische Aspekt als mehr die soziale Innovation maßgeblich; sie stelle die Grundlage für eine neue partizipative Kultur dar[63].

[58] Jeffrey Zeldman, http://www.alistapart.com/articles/web3point0
[59] Nicholas Carr, http://roughtype.com/archives/2005/10/the_amorality_o.php
[60] http://www.economist.com/printedition/displayStory.cfm?story_id=6911109
[61] Kevin Kelly in seinem Meilenstein-Artikel „We Are The Web“, http://www.wired.com/wired/archive/13.08/tech.html
[62] Paul Graham, http://www.paulgraham.com/web20.html
[63] http://www.heise.de/newsticker/meldung/74389

### Schlussfolgerung

Die Kritiker richten sich im Wesentlichen gegen den unpräzisen Begriff und den Hype um *Web 2.0*. Die zunehmende Verwendung innovativer Technologien und das verstärkte Aufkommen von sozialen Strukturen im World Wide Web sind nicht von der Hand zu weisen - und werden selbst von kritischen Beobachtern nicht bestritten. Die Entscheidung, ob man mit dem Schlagwort *Web 2.0* einverstanden ist, bleibt jedem selbst überlassen. Was man darunter versteht, kann man seit Tim O'Reillys Artikel nicht mehr völlig frei entscheiden.

O'Reilly mag ein geschickter Marketing-Stratege sein, vor allem ist er aber auch als Fachmann und „Trend Spotter"[64] bekannt und gehört zu den Internet-Pionieren[65]. Wir wollen ihm gerne glauben, dass er eine große Entwicklung für das World Wide Web voraussieht. Wie diese Entwicklung verläuft hängt aber von unzähligen Faktoren ab, allen voran von dem selbstbestimmten, eigenverantwortlichen und emanzipierten Benutzer.

Ob *Web 2.0* dem Internet wirklich einen nachhaltigen wirtschaftlichen Aufschwung bringt, wird sich ebenfalls herausstellen. Momentan zeigen sich die Investoren in Spendierlaune, und die Unternehmen zieren sich davor, sich unter Preis zu verkaufen: Facebook hat unlängst ein Angebot von 750 Mio. US-Dollar ausgeschlagen, weil die Eigentümer hoffen, 2 Milliarden Dollar erzielen zu können.[66]. Aber manche sehen schon das Platzen der nächsten Blase voraus[67]...

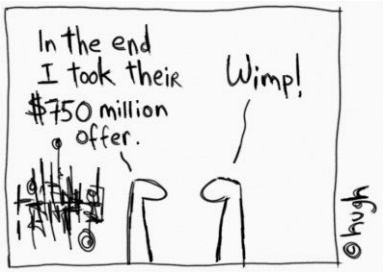


Abbildung 1.14: Wirtschaftsfaktor oder dotcom bubble 2.0? (Grafik von Hugh MacLeod, http://www.gapingvoid.com/Moveable_Type/archives/002574.html)

## 1.4.3 Rechtliche Anmerkung

Web 2.0 ist nicht mehr nur ein generischer Begriff. Der Verlag CMP, der zusammen mit O'Reilly die „Web 2.0 Conference" ausrichtet, hat die Eintragung einer Marke „Web 2.0"

[64] http://www.wired.com/wired/archive/13.10/oreilly_pr.html
[65] http://www.oreilly.com/oreilly/tim_bio.html
[66] „The price, by the way, is high but not ridiculous yet" kommentiert Michael Arrington, http://www.techcrunch.com/2006/03/28/facebook-is-doing-the-skype-dance/
[67] http://bubble20.blogspot.com/

für Konferenzen und ähnliche Veranstaltungen beantragt[68], um sich vor Nachahmern bezüglich Namen und Konzept ihrer Veranstaltungen zu schützen. Dieses Vorgehen löste in der Blogosphäre eine rege Disksussion aus. Kritiker wie Dave Winer bemerken[69], dass der kommerzielle Erfolg von O'Reillys Web 2.0-Begriff auf Ideen und Techniken beruht, die andere mit nicht-kommerzieller Absicht entwickelt haben.

[68] http://radar.oreilly.com/archives/2006/05/controversy_about_our_web_20_s.html
[69] http://www.scripting.com/2006/05/30.html#oreillyAndWeb20TrademarkIssues

# 2 AJAX

# Inhaltsverzeichnis

# 1 Einleitung

AJAX ist ein Begriff, der heute fast automatisch in Verbindung mit dem Begriff "Web 2.0" fällt. Doch was ist AJAX überhaupt? Welche Vorteile hat es? Warum nicht bei den bisherigen Techniken bleiben?

Auf diese Fragen möchte ich in diesem Kapitel eingehen, AJAX als Technologie vorstellen und praktisch in Beispielen umsetzen.

## 1.1 Warum AJAX?

AJAX (kurz für „Asynchronous JavaScript and XML“) ist mehr als ein Name. Es ist auch mehr als eine Sammlung bekannter Techniken.

AJAX ist eine Technologie zur Entwicklung von interaktiven Web-Anwendungen. Die Grundidee dabei ist, dass beim Interagieren mit der Anwendung nicht ständig neue Seiten dargestellt werden, sondern Inhalte asynchron vom Server nachgeladen werden können. Dies hat zur Folge, dass Webseiten ansprechbarer reagieren und sich dadurch die Benutzerfreundlichkeit deutlich verbessert. Mit AJAX eröffnen sich neue Perspektiven, Web-Anwendungen zu entwickeln, die bisher nicht vorstellbar waren.

Als Hauptinformationsquelle verwende ich „Ajax in Action“[1] von Dave Crane und Eric Pascarello. Das Buch hat sich als gut strukturiert erwiesen, ist einsteigerfreundlich, geht aber trotzdem in die Tiefe. Wer sich professionell mit AJAX beschäftigen will, dem sei nach diesem Paper die Lektüre dieses Buches empfohlen.

Bevor wir uns konkreten Technologien widmen, soll im Folgenden dargestellt werden, wie sich das heutige Internet dem Benutzer darstellt und welche Schwachpunkte zu finden sind. Diese Schwachpunkte müssen wir vor Augen haben, wenn wir uns der Frage nach dem Zweck und Nutzen von AJAX widmen.

### 1.1.1 Benutzeroberflächen

Stellen wir uns die Frage nach einer idealen Benutzeroberfläche, kommen wir zu dem Schluss, dass eine Benutzeroberfläche dann ideal ist, wenn sie nicht bemerkt wird. Sie sollte nur mit dem Benutzer in Interaktion treten, wenn es nötig ist und ansonsten nicht im Weg stehen.

Wir stehen heute vor folgendem Dilemma: Unsere Desktop-Anwendungen wurden über die Jahre hinweg stetig weiterentwickelt und optimiert. Um einige Details beispielhaft herauszugreifen: Schaltflächen verändern bei MouseOver ihr Aussehen, SmartTags bieten uns kontextsensitive Funktionen an, Animationen wie z.B. beim Ein- und Ausblenden eines Anwendungsfenster erleichtern uns, es erneut zu finden und die gesamte Oberfläche lässt sich komfortabel mit Maus und Tastatur bedienen.

Wie sieht es im Vergleich dazu mit Web-Anwendungen aus? Eine heutige Web-Anwendungen zeichnet sich durch folgende Merkmale aus: Auf einer einzelnen Seite wird uns eine Vielzahl von

1 Literaturquelle [AJAX]

Information präsentiert. Diese Information kann personalisiert sein. Und dennoch: Die einzige Möglichkeit, mit dieser Information zu interagieren, ist es, Hyperlinks anzuklicken oder Eingaben über HTML-Formulare zu machen. Als Ergebnis hierzu wird eine komplett neue Seite geladen.

Sicher hat es auch hier Weiterentwicklungen gegeben. Mit Hilfe von CSS und JavaScript können einige Funktionalitäten implementiert werden, die den Benutzerkomfort erhöhen.

Und trotzdem haben wir immer noch das Gefühl, dass uns die Benutzeroberfläche einer typischen Web-Anwendung im Weg steht. So stört z.B. ein komplett neuer Seitenaufbau deutlich den Arbeitsfluss. Selbst nach über 10 Jahren, in denen das Internet dem Endbenutzer zugänglich geworden ist, gibt es deutliche Mängel. Wir kennen eine ideale Benutzeroberfläche einer Desktop-Anwendung, und genau deshalb ist der Umgang mit typischen Web-Anwendungen so frustrierend.

#### 1.1.1.1 1:0 für AJAX

Mit AJAX können Oberflächen gestaltet werden, die sehr flüssig auf Benutzerinteraktionen reagieren. Dies steigert den Komfort deutlich und lässt den Anwender vergessen, dass er mit einer Web-Anwendung arbeitet.

### 1.1.2 Netzwerk-Latenz

Setzen wir uns mit der Frage der Netzwerk-Latenz auseinander, ist uns bewusst, dass das Internet nicht aus einer zentralen Anwendung besteht, sondern vielmehr aus einer verteilten Anwendung. Information kann von jeder beliebigen Stelle aus geliefert bzw. abgerufen werden. Dazwischen existieren Netzwerkverbindungen, die Verzögerungen verursachen.

Werden Informationen von einem entfernten System angefragt (Request), muss dieses die Anfrage auswerten, die benötigten Daten lesen bzw. berechnen, encodieren und an den Client zurückschicken (Response). Dieser muss die Daten verwerten und kann dann erst mit der Darstellung beginnen. Es lässt sich also feststellen, dass der ganze Prozess sehr zeitintensiv ist.

Die Datenübertragung kann durch schnellere Internetzugänge ausgeglichen werden, eine hohe Belastung der Server-Seite jedoch nicht. Es sprechen also Gründe dafür, Teile der Anwendungslogik auf den immer leistungsfähigeren Client zu übertragen.

#### 1.1.2.1 2:0 für AJAX

Bei AJAX wird ein Teil der Anwendungslogik auf den Client übertragen.

### 1.1.3 Asynchrones Interagieren

Wir sollten also davon ausgehen, dass jede Interaktion mit dem Server eine zeitaufwändige Interaktion ist, die möglichst asynchron ablaufen sollte, was uns zum nächsten Punkt führt.

Es folgt eine Definition von (a)synchronem Interagieren: „Unter asynchroner Kommunikation versteht man in der Informatik und Netzwerktechnik einen Modus der Kommunikation, bei dem das Senden und Empfangen von Daten zeitlich versetzt und ohne Blockieren des Prozesses durch bspw. Warten

auf die Antwort des Empfängers (wie bei synchroner Kommunikation) stattfindet.“ (Wikipedia: Asynchrone Kommunikation).

Ein einfaches Beispiel hierfür: Wir nehmen an, wir möchten uns morgens einen Kaffee kochen und die Zeitung aus dem Briefkasten holen. Asynchrone Kommunikation bedeutet in diesem Fall: Kaffeemaschine einschalten, die Zeitung aus dem Briefkasten holen und lesen, bis der Kaffee fertig ist. Im Gegenteil dazu würde synchrone Kommunikation bedeuten: Kaffeemaschine einschalten, einige Minuten warten, bis der Kaffee fertig gekocht ist, erst danach die Zeitung holen und lesen.

Wir halten fest: Synchrone Kommunikation entspricht nicht unserem alltäglichen Umgang; sie würde einen immensen Effektivitäts- und Zeitverlust bedeuten.

Daraus ergibt sich folgendes Problem: Im Internet existiert in heutigen Anwendungen ausschließlich synchrone Kommunikation. Der Prozessfluss einer traditionellen Web-Anwendung läuft über HTTP und folgt damit automatisch dem Request-Response-Paradigma. Das heißt: Der Client fordert Daten beim Server an (Request) und dieser antwortet (Response). Verzögert sich die Antwort, so verzögert sich auch der komplette Prozessablauf.

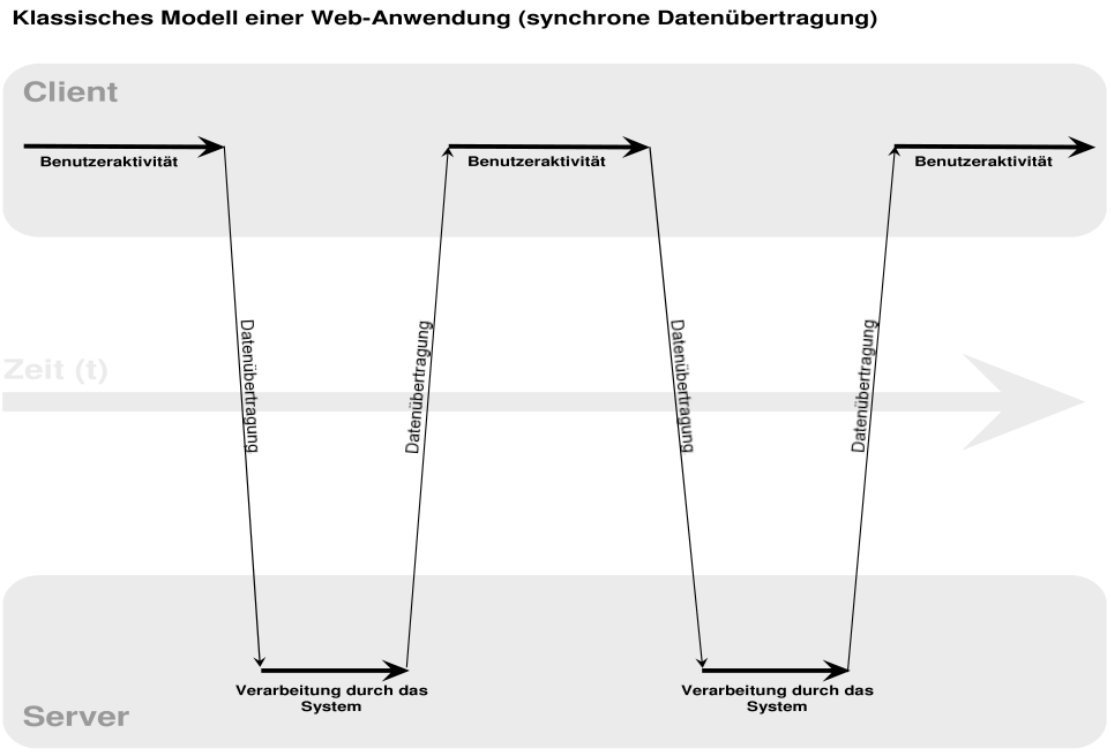


*Abbildung 1: Synchrone Kommunikation einer klassischen Web-Anwendung (Quelle: Wikipedia, "AJAX (Programmierung)")*

Zwar ist Parallelität in HTTP 1.1 integriert - pro TCP-Verbindung können mehrere Requests und Responses gesendet werden – trotzdem können für die einzelnen Requests keine Callback-Handler

festgelegt werden.

Wir kommen also zu dem Schluss, dass traditionelle Web-Anwendungen mit ihrer synchronen Kommunikation im Gegensatz zu unserer gewohnten asynchronen Kommunikation stehen.

#### 1.1.3.1 3:0 für AJAX

Mit AJAX wird asynchrone Kommunikation möglich; für jeden Request kann ein Callback-Handler definiert werden.

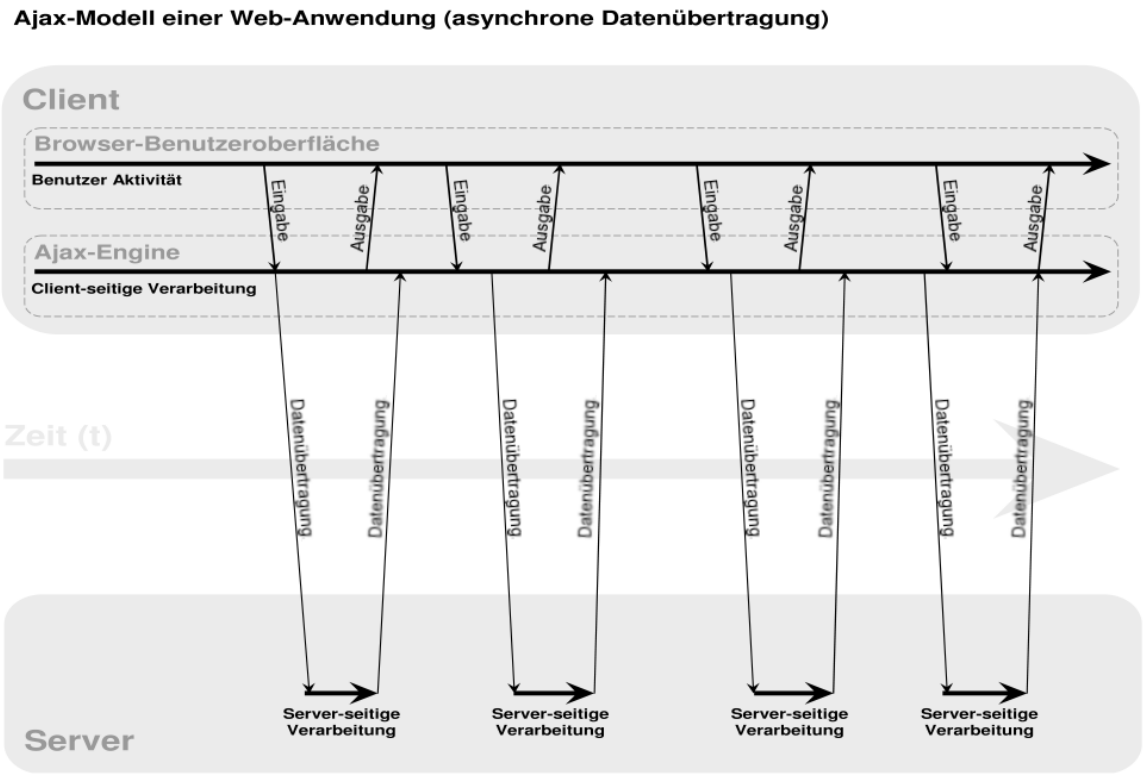


*Abbildung 2: Asynchrone Kommunikation einer AJAX-Anwendung (Quelle: Wikipedia, "AJAX (Programmierung)")*

### 1.1.4 Souveräne und transiente Anwendungen

Alan Cooper und Robert Reiman unterscheiden in ihrem Buch „About Face 2.0“[2] grundsätzlich zwischen Webseiten und Web-Anwendungen. Webseiten stellen informationsbasierte Dienste dar,

2 Literaturquelle [FACE]

wobei die Inhalte statisch abgelegt oder aus einer Datenbank angefragt werden können. Web-Anwendungen hingegen sind durch hohe Benutzerinteraktion geprägt.

Die Autoren des Buchs unterscheiden zwischen zwei verschiedenen Kategorien von Web-Anwendungen, bedingt durch ihr Benutzungsarten: souveräne und transiente Anwendungen.

Zu souveränen Anwendungen gehört beispielsweise ein Textverabeitungsprogramm. Während seiner Benutzung muss es ständig auf Eingaben reagieren und Hintergrundaufgaben wie z.B. Rechtschreibüberprüfung übernehmen. Verzögerungen fallen bei diesen Anwendungen stark auf. So ist es z.B. bei einer Grafikanwendung wie Photoshop unverzeihlich, wenn der Pinsel mit einer Verzögerung von zwei Sekunden zeichnet.

Im Gegensatz dazu sind heutige Web-Anwendungen häufig transient. Transiente Anwendungen müssen nicht in derselben Form mit dem Benutzer interagieren wie souveräne Anwendungen. Verzögerungen sind verzeihbar. Verzögert sich z.B. die Antwort auf eine Google-Suchanfrage um wenige Sekunden, kann dies noch akzeptiert werden.

Die traditionellen Technologien bieten folglich nur die Möglichkeit, transiente Anwendungen web-basiert anzubieten. Die fehlende Möglichkeit zur umfassenden Interaktion mit dem Benutzer machte es bisher nur unter erschwerten Bedingungen möglich, eine souveräne Anwendung abzubilden. Von Ausnahmen wie Outlook Web Access abgesehen ist z.B. eine heutige Webmail-Anwendung nicht vergleichbar mit einem Desktop-Mail-Client.

#### 1.1.4.1 4:0 für AJAX

Mit AJAX stehen uns Möglichkeiten zur Verfügung, souveräne Anwendungen abzubilden.

# 2 Die Technologie von AJAX

Wie bereits mehrfach erwähnt, eröffnen sich mit AJAX Perspektiven, wie wir sie vorher nicht hätten abschätzen können. Um diese Perspektiven zu erkennen, ist es allerdings notwendig, dass wir uns von einigen traditionellen Vorstellungen, wie Internet benutzbar ist, trennen.

Um allerdings verstehen zu können, warum dies weitreichende Konsequenzen hat, möchte ich im Folgenden darstellen, aus welchen Techniken AJAX besteht und wie diese zusammenspielen.

Wie bereits erwähnt, steht AJAX kurz für „Asynchronous JavaScript and XML“. Was genau versteht man darunter? Wie ist AJAX entstanden?

## 2.1 Die Entstehung von AJAX

Der Begriff „AJAX“ ist überraschenderweise noch relativ neu. Unter diesem Begriff fassst man seit 2005 Technologien zusammen, die schon länger exisitieren und lässt ihnen so eine neue Bedeutung zukommen.

Wer den Begriff AJAX erfunden hat, kann heute nicht mehr nachvollzogen werden. Geprägt wurde er durch Jesse James Garret, einem Mitarbeiter von Adaptive Path, der in seinem Aufsatz „Ajax: A New

Approach to Web Applications“[3] über die Technologie berichtete. So kann sich Garret also die Marke AJAX zuschreiben, die Technologie zur asynchronen Kommunikation über den XMLHttpRequest existierte aber bereits vorher.

### 2.1.1 Was bisher geschah

Einen ersten Ansatz, interaktive Anwendungen zu realisieren, machte Microsoft bereits 1997 mit dem Remote Scripting. Diese Technologie baiserte auf einem IFRAME (ab Internet Explorer 3 verfügbar) bzw. einem LAYER (ab Netscape 4), in dem mittels dem src-Attribut Webseiten in eine Parent-Website eingebunden werden können. Wurde in der eingebetteten Webseite JavaScript-Code eingebaut, der die Parent-Site beeinflusst, konnten hiermit AJAX-ähnliche Effekte, d.h. das Nachladen von Teilen einer Seite erzielt werden.

Im Jahr 1998 führte Microsoft das sogenannte MSRS (Microsoft Remote Scripting) ein. Hier wurde als zentrales Element ein Java-Applet verwendet, das Daten vom Server „pullt“, also anfordert. Mit diesem Java-Applet konnte über JavaScript-Methoden kommuniziert werden und so die Daten auf der Website verfügbar gemacht werden. Diese Technik wurde von Microsoft z.B. in Outlook Web Access eingesetzt, einem webbasierten PIM[4]-Client zur Verwaltung von E-Mail, Kalender, Kontakten und Aufgaben, der auf einem Microsoft Exchange Server 5.0 aufsetzte.

In einer Microsoft-Newsgroup begann eine Diskussion zur die aktuell eingesetzten Technik. Der eigentliche Umbruch zu der heute verwendeten Technologie wurde durch einen Vorschlag angestoßen, das Java-Applet durch das XMLHttpRequest-Objekt zu ersetzen.

### 2.1.2 Zentraler Bestandteil: Der XMLHttpRequest

Was genau versteht man unter einem XMLHttpRequest? Ein XMLHttpRequest ist ein Browserobjekt mit einer spezifischen API, die von JavaScript, JScript[5], VBScript und anderen Webbrowser-Skriptsprachen benutzt werden kann, um XML- oder Textdaten von oder zu einem Web-Server über HTTP zu übertragen. Dazu wird ein unabhängiger Verbindungskanal zwischen der Client- und der Server-Seite aufgebaut (asynchrone Kommunikation).

Genau dieser XMLHttpRequest dient in den heutigen Technologien, die unter dem Begriff AJAX zusammengefasst werden, zum Datenaustausch zwischen dem Client und dem Server.

## 2.2 ajaX - XML

Aus dem Begriff XMLHttpRequest lässt sich ableiten, dass XML als Übertragungsformat der Daten verwendet wird. Es können jedoch auch reine Textdaten über den XMLHttpRequest übertragen werden.

Beschränken wir uns (wie in unseren Beispielen) auf Textdaten, müssen wir uns nicht mit XML beschäftigen. Die XMLHttpRequest-API stellt jedoch auch Funktionalitäten bereit, mit XML

3 http://adaptivepath.com/publications/essays/archives/000385.php

4 PIM: Personal Information Management. Unter diesem Begriff fasst man eine Anwendung zur Verwaltung von E-Mail, Kontakten, Kalender und Aufgaben. Bekanntestes Beispiel: Microsoft Outlook

5 JScript: Microsoft-eigene Sprachvariante von JavaScript

umzugehen.

## 2.3 aJax - JavaScript

Wie bereits erwähnt, kann auf den XMLHttpRequest mittels verschiedener Skriptsprachen zugegriffen werden. Bei AJAX wird JavaScript verwendet wird, um mit dem XMLHttpRequest zu interagieren und die Darstellung der Website zu beeinflussen.

Zur Änderung der Darstellung muss das Document Object Model (DOM) einer Webseite bearbeitet werden.

Dazu folgender Hintergrund: Jedes HTML- und XML-Dokument ist baumartig aufgebaut. Das DOM ist im Grunde eine API für den Zugriff auf diesen Baum.

Versieht man z.B. Element im Dokumentenbaum mit einer ID, können sie direkt angesprochen und geändert werden (z.B. Bearbeiten eines Attributs). Außerdem kann über eine komplette Menge von gleichen Elementknotentypen (Bilder, Formularfelder etc.) iteriert werden. Später dazu mehr.

DOM wurde in der Version 1.0 bereits 1998 vom World Wide Web Consortium (W3C) definiert[6].

## 2.4 Ajax - Asynchronous Communication

Bleibt noch der letzte Bestandteil des Namens AJAX: Asynchronous. Unter asynchronous versteht man, dass der Datentransfer mittels XMLHttpRequest jederzeit (asynchron) stattfinden kann, und zwar in einer dedizierten Verbindung. So ist es möglich, mehrere XMLHttpRequests parallel abzuschicken, auf die unabhängig voneinander vom Server reagiert werden kann.

Auf die Auswirkungen dieser Art der Kommunikation zwischen Client und Server möchte ich später eingehen.

## 2.5 Browser-Unterstützung

Die Technologie hört sich soweit interessant an, aber wie sieht es mit der Browser-Unterstützung aus?

Ein grundlegender Bestandteil, das JavaScript, ist heute bekanntlich in jeden Browser integriert. Im Jahr 1995 von Netscape eingeführt und lizenziert, hat es heute den Versionsstand 1.6 erreicht, der z.B. in Firefox 1.5 implementiert ist. Andere Browserversionen wie der sehr verbreitete Internet Explorer 6 unterstützen nur die Vorgängerversion 1.5.

Die Neuerungen in JavaScript 1.6 halten sich in Grenzen[7]; für unsere Beispiele und die meisten Web-Anwendungen genügt JavaScript 1.5, das schon mit DOM umgehen kann.

Bleibt also die Frage, wie der XMLHttpRequest in heutige Browser implementiert wird. Die erstaunliche Antwort: Er wird schon seit geraumer Zeit unterstützt! Microsoft machte den Anfang, indem es den XMLHttpRequest als ActiveX-Objekt[8] in Internet Explorer 5 integrierte. Mozilla zog in der Version 1.0 nach. Heutige Browser, auch Firefox, Safari, Konqueror und Opera haben eine

6 http://www.w3.org/DOM/
7 http://developer.mozilla.org/en/docs/New_in_JavaScript_1.6
8 ActiveX: Softwarekomponenten-Modell zur Implementierung aktiver Inhalte

Unterstützung für das XMLHttpRequest-Objekt integriert.

Warum wurde also nicht früher auf die AJAX-Technologie gesetzt bzw. einen Verbund von Technologien, die heute unter dem Namen AJAX zusammengefasst werden? Warum gibt es bisher erst wenige Anwendungen?

Das Problem ist, dass es für die XMLHttpRequest-Objekt-API vom W3C erst seit April 2006 einen Entwurf einer standardisierten Spezifikation[9] gibt. Bisherige Lösungen basieren meist auf propietären Lösungen und sind systemabhängig bzw. beinhalten systemabhängige Code-Abschnitte. Dies betrifft hauptsächlich die Erzeugung des XMLHttpRequest-Objekts. Es ist empfehlenswert, die Web-Anwendung in jedem geläufigen Browser zu testen; es sollte jedoch insgesamt zu wenigen Problemen kommen.

## 2.6 Prinzipien von AJAX

Eine Web-Anwendung, die auf AJAX basiert, widerspricht in vielem unseren heutigen Vorstellungen einer klassischen Web-Anwendung. Diese möchte ich im Folgenden beleuchten.

### 2.6.1 Der Rich Client: Im Browser läuft eine Anwendung

Bei einer klassischen Web-Anwendung fungiert der Browser mehr oder weniger als Terminal. Er stellt nur den Inhalt dar und weiß nichts von der Anwendungslogik, die dahinter steckt. Das schränkt seine Möglichkeiten ein, nimmt ihm aber auch Verantwortung ab.

Die Anwendung wird komplett auf dem Server gehostet und die benutzerspezifischen Daten in einer Session gehalten. Jede Interaktion mit dem Server erfolgt durch einen Request, auf den hin der Server eine komplett neue Seite zurück liefert (Response). Auf dieser Seite sind erneut Struktur und Inhalt vermischt.

Zwar gibt es heute bereits Technologien wie .NET oder JSP, die die Erstellung einer Web-Anwendung für Entwickler deutlich einfacher machen, für die Benutzer der Anwendung verändert sich dadurch nichts. Sie bekommen das Request-Response-Paradigma deutlich zu spüren.

9 http://www.w3.org/TR/XMLHttpRequest/

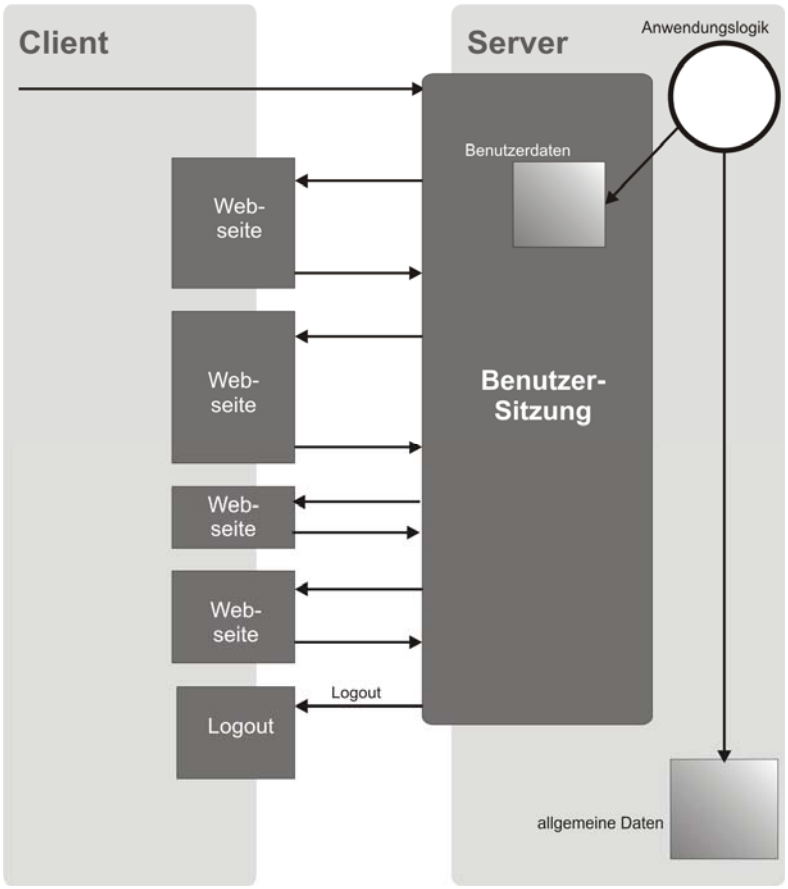


*Abbildung 3: Der Lebenszyklus einer klassischen Web-Anwendung (Englische Originalquelle: "Ajax in Action")*

Im Gegensatz dazu wird bei AJAX ein Teil der Anwendungslogik auf den Client übertragen. Der Client wird damit zum Rich Client.

Die Anwendungslogik besteht in unserem Fall aus JavaScript-Code, der einen verhältnismäßig großen Teil der übertragenen Daten ausmacht. Das Dokument wird jedoch nicht bei der nächsten Benutzerinteraktion verworfen, sondern bleibt über eine Session hin bestehen. Es sind Funktionen integriert, die auf Benutzereingaben reagieren. Die Anwendungslogik kann damit umgehen und z.B. einen asynchronen Datenbank-Request an den Server schicken.

Das Dokument kann während seiner Lebenszeit durchaus sein Aussehen verändern, die Anwendungslogik dahinter bleibt allerdings dieselbe. So können auch Zustände wie der Inhalt eines Warenkorbs gespeichert werden.

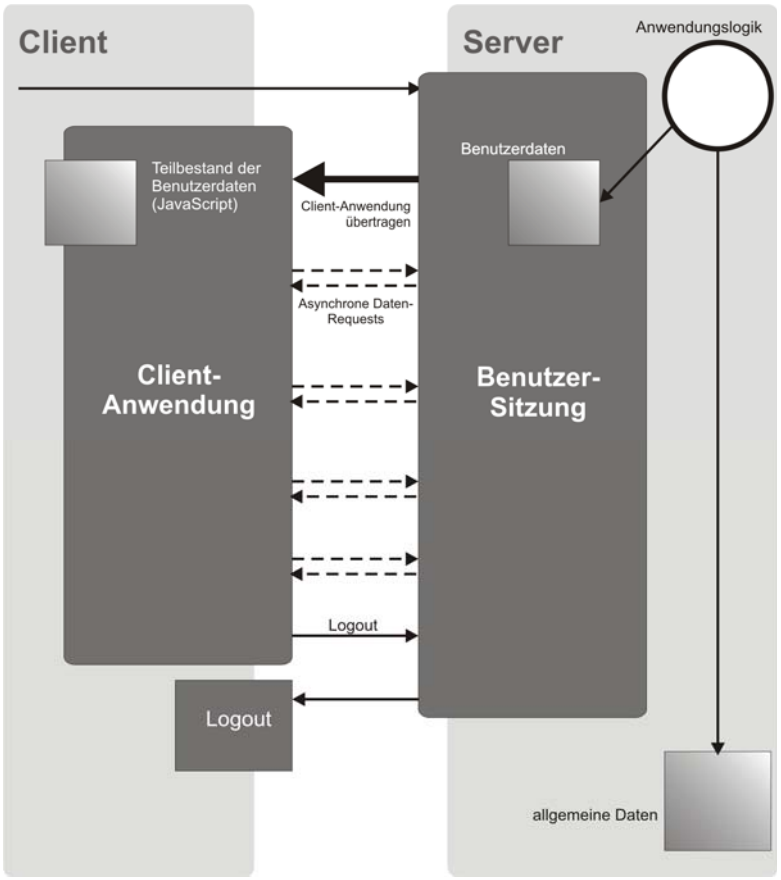


*Abbildung 4: Lebenszyklus einer AJAX-Anwendung (Englische Originalquelle: "Ajax in Action")*

### 2.6.2 Der Server übertragt Daten, keine Inhalte

Wie bereits beschrieben, überträgt der Server bei einer klassischen Web-Anwendung nur Inhalte. Diese Inhalte liegen unstrukturiert und ohne zugehörige Anwendungslogik vor.

Bei einer Web-Anwendung, die auf AJAX basiert, wird ein Teil der Anwendungslogik mit übertragen. Dies wirkt sich fundamental auf die Aufgabe des Servers aus: Der Client übernimmt einen Teil der bisherigen Aufgaben des Servers und fragt über den XMLHttpRequest nur noch Daten in strukturierter Form nach, die er selbst auswerten kann. Der Client wird zum Rich-Client, der Server „thinner“.

Dass auch Anwendungslogik und nicht nur Inhalte übertragen werden, wirkt sich auf die Ladezeiten aus – wie es scheint negativ. Doch der Schein trügt: Zwar werden beim Erstaufruf eines Dokuments mehr Daten übertragen als bei einer klassischen Web-Anwendung, dieser Mehraufwand macht sich aber schnell bezahlt. Bei weiteren Interaktionen innerhalb eines Dokuments werden nur Daten nachgeladen; die Anwendungslogik bleibt bestehen und verwendet die ankommenden Daten, um Teile des Dokuments zu ändern. Wie also in unten stehender Abbildung dargestellt steigt der Datentransfer nach einem zu Anfang höheren Datenaufkommen nur langsam an.

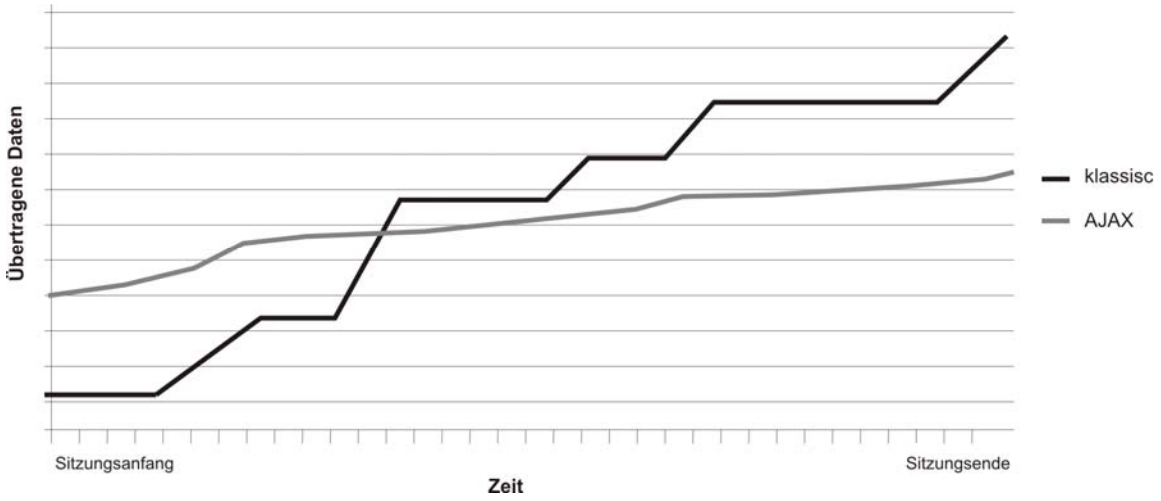


*Abbildung 5: Vergleich bezüglich des Datenverkehrs*

### 2.6.3 Der Benutzer kann flüssig mit der Anwendung interagieren

In einer klassischen Web-Anwendung stehen dem Benutzer nur zwei Möglichkeiten zur Verfügung, mit der Anwendung zu interagieren: Hyperlinks und HTML-Formulare. Dank CSS können Hyperlinks benutzerfreundlich gestaltet werden, z.B. bezüglich des Feedback-Verhaltens beim Überfahren mit der Maus. HTML-Formulare können mit Hilfe von JavaScript auf Fehleingaben überprüft werden.

Aber: Sobald ein Hyperlink geklickt wird oder ein Formular abgeschickt wird, wird der Anwender bis zur Antwort des Servers und der Aktualisierung der Seite zu einer Pause gezwungen.

Da AJAX asynchrone Kommunikation ermöglicht, eröffnen sich hier neue Möglichkeiten. Soll nur der Inhalt einer Seite geändert werden, lässt sich vieles mit JavaScript allein realisieren. Meist geht es jedoch darum, dass auch die Anwendungslogik von dieser Änderung benachrichtigt wird.

Ein Beispiel: Wir möchten einen Artikel aus einem Warenkorb entfernen. Er soll jedoch nicht nur ausgeblendet werden, sondern auch tatsächlich entfernt werden. AJAX übernimmt hier nicht nur die Darstellungsänderung, dank dem XMLHttpRequest wird auch der Server von der Änderung benachrichtigt – und das alles ohne dass es sich beim Benutzer durch einen Neuaufbau der Seite bemerkbar macht.

Man könnte argumentieren, die Seitenaktualisierung könne durch eine schnelle Internetanbindung kompensiert werden. Doch es geht nicht um die Wartezeit, in die der Benutzer zum Nichtstun gezwungen ist. Es geht darum, dass ihn eine flüssig reagierende Anwendung vergessen lässt, dass er vor dem Computer sitzt. Wie bereits erwähnt ist das ein Ziel einer sinnvollen Oberflächengestaltung. Was also bei einer Desktop-Anwendung bereits heute Gang und Gäbe ist, wird auch für Internetapplikationen möglich.

### 2.6.4 Mit AJAX wird richtig programmiert, was Disziplin erfordert

Wie wir im späteren Kapitel sehen werden, bedeutet der Einstieg in AJAX der Einstieg in richtiges Programmieren.

Sind in klassische Web-Anwendungen meist kleine, überschaubare JavaScript-Teile integriert, muss in einer AJAX-Anwendung JavaScript-Code geschrieben werden, der zum einen viel umfangreicher als bisher ist und der im Prinzip stundenlang laufen kann, performant, fehlerfrei und speichereffizient.

Bei AJAX wird also nach allen Regeln der Kunst programmiert und so ist es erforderlich, mit einer gewissen Planung und Struktur an die Erstellung des Codes zu gehen. AJAX ist nicht nur ein weiterer Entwicklungsschritt einer klassischen Web-Anwendung; es bedeutet eine grundsätzlich neuartige Herangehensweise an die Entwicklung.

## 2.7 Heutige Beispiele

Obwohl sich AJAX bisher nur zögerlich durchgesetzt hat, gibt es bereits heute einige Web-Anwendungen, die die neue Technologie einsetzen. An Hand folgender Beispiele soll die Mächtigkeit von AJAX dargestellt werden.

### 2.7.1 Google Maps

Wenn es um Innovation und Begeisterung für neue Technologie geht, muss man Google ein Kompliment aussprechen. Die wohl bekannteste Web-Anwendung, die mit AJAX implementiert wurde, stammt von Google und heißt „Google Maps“[10].

10 http://maps.google.de bzw. http://maps.google.com

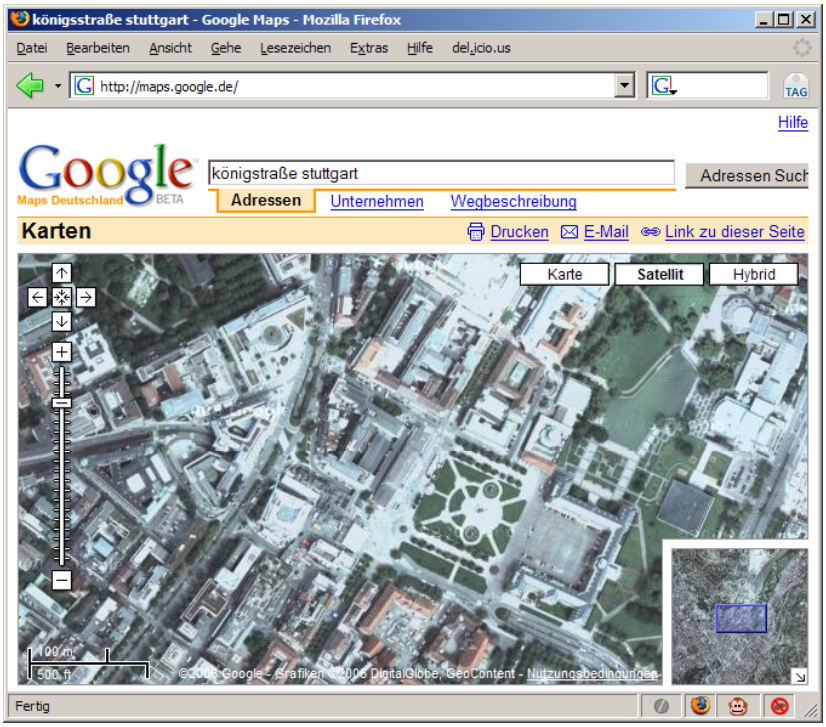


*Abbildung 6: Google Maps Deutschland: Königstraße Stuttgart*

Es handelt sich dabei um einen webbasierten Kartendienst, in den auch Satellitenaufnahmen integriert sind (inzwischen weltweit teilweise in faszinierender Qualität). Die Suche nach Orten erfolgt hierbei nach klassischer Art über ein HTML-Eingabefeld, auf das hin eine neue Seite aufgebaut wird.

AJAX kommt hier bei der Anzeige der Karte zum Einsatz. Die Karte kann mit gedrückter Maustaste verschoben werden, neue Kartenausschnitte werden asynchron nachgeladen. All dies geschieht ohne den Aufbau einer komplett neuen Seite. So kann sehr interaktiv mit der Karte interagiert werden; der Arbeitsfluss wird nicht gestört.

### 2.7.2 Google Suggest

Doch Google macht auch durch andere Anwendungen von sich zu sprechen. So ist z.B. „Google Suggest“[11] im Beta-Betrieb. Bereits während der Eingabe eines Suchbegriffs wird eine Datenbankabfrage gestartet und Suchwörter anderer Benutzer ähnlich einer Auto-Vervollständigen-Funktion als Drop-Down angezeigt. Neben möglichen Suchwörtern wird auch die Anzahl der möglichen Ergebnisse angezeigt.

11 http://www.google.com/webhp?complete=1&hl=en

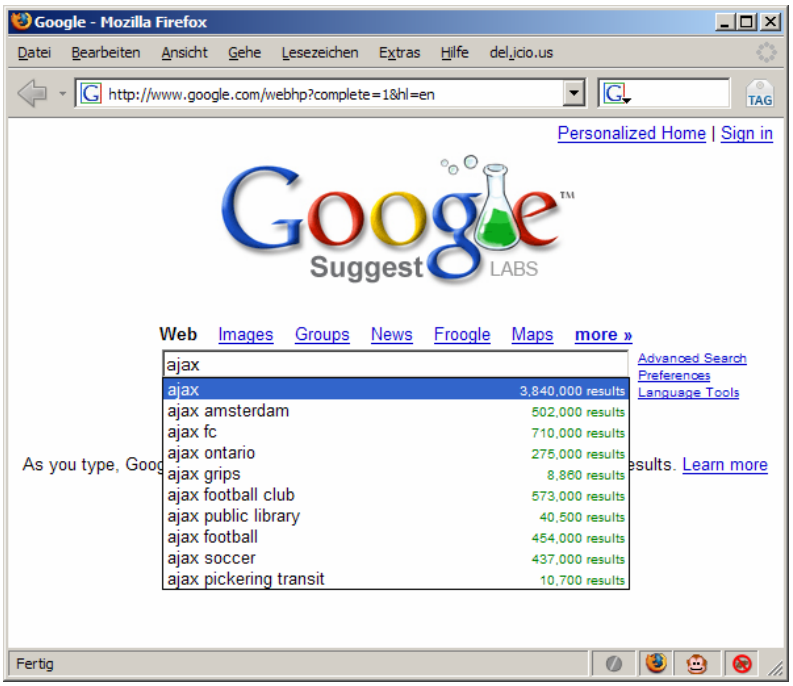


*Abbildung 7: Google Suggest*

Wahrscheinlich ist Google Suggest bis heute noch nicht in Google implementiert (und wird es vielleicht auch nie werden), weil bei jedem eingetippten Zeichen ein Datenbank-Request gestartet wird. Bei Millionen von Nutzer würde das die Performance der Suchmaschine deutlich verschlechtern.

Als Verbesserung wäre zu überlegen, ein Time-Out nach dem zuletzt getippten Zeichen zu setzen, was die Zahl der Anfragen senken würde.

### 2.7.3 Del.icio.us

Doch nicht nur Google setzt AJAX ein. Ein weiteres Beispiel ist der Bookmark-Verwalter del.icio.us[12], in dem Bookmarks hinzugefügt, getaggt[13] und mit anderen geteilt werden können. AJAX macht sich hier an mehreren Stellen hilfreich bemerkbar, zum Beispiel beim Hinzufügen eines neuen Bookmarks. Beim Eintippen eines Stichworts werden ähnlich zu Google Suggest bereits verwendete Tags vorgeschlagen. Wird der Bookmark später bearbeitet, geschieht das innerhalb des Dokuments: Es erscheint direkt unter dem Bookmark ein HTML-Formular; beim Speichern werden die Änderungen an den Server übertragen.

Leider wurde die AJAX-Philosophie nicht konsequent verfolgt, so muss z.B. zur Aktualisierung der Tag-Übersichtsliste auf der rechten Seite ein kompletter Site-Refresh durchgeführt werden. Bleibt zu hoffen, dass dies in einer zukünftigen Version implementiert werden wird.

12 http://www.del.icio.us
13 Tagging: Beschreiben eines Datums mit Hilfe von Stichwörten (Tags), die als Meta-Daten abgelegt werden

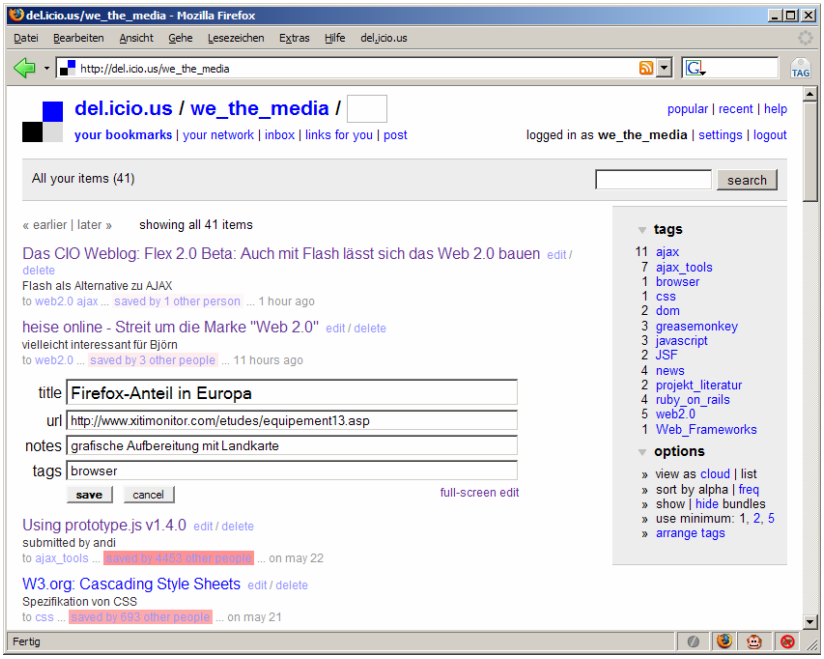


*Abbildung 8: Bookmark-Manager Del.icio.us*

Dies soll als Auswahl an Beispielen genügen.

# 3 AJAX konkret

## 3.1 Jetzt geht's los!

Ich hoffe, mit meinem Einstieg in AJAX Vorgeschmack erzeugt zu haben. Wir haben die Verbesserungen in der Benutzerfreundlichkeit betrachtet und sind auch schon zum Teil auf technische Konsequenzen eingegangen. Nun möchten wir in die Technik einsteigen, Funktionsweisen kennen lernen und diese an praktischen Beispielen umsetzen.

## 3.2 Schlüsselelemente von AJAX

Das Schöne an AJAX ist: AJAX ist eine Zusammenfassung bereits zum größten Teil bekannter Technologien. Ist man mit Web-Entwicklung vertraut, ist die Einstiegshürde gering.

Wichtig ist es bei AJAX vor allen Dingen, strukturiert vorzugehen. Dazu gehört z.B. eine strikte Einhaltung des MVC-[14] und anderer Patterns, worauf ich später eingehen möchte.

AJAX besteht aus vier grundlegenden Technologien, mit denen wir uns als Web-Entwickler

14 MVC-Pattern: Pattern des Model-View-Controller. Hierbei ist eine strikte Trennung zwischen Datenmodell, Darstellung und Anwendungslogik vorgesehen. Mehr dazu in Kapitel III

auseinandersetzen müssen:

1. **JavaScript (JS)**[15]: Eine Skriptsprache, die normalerweise in Anwendungen (in unserem Fall Web-Anwendungen) eingebettet ist. AJAX verwendet als Sprache zur Implementierung der Anwendungslogik durchweg JS, das vom JS-Interpreter des verwendeten Browsers ausgeführt wird.
2. **Cascading Style Sheets (CSS)**[16]: Mit Hilfe von CSS können sehr einfach visuelle Stile definiert werden. Diese können auf Element-Typen (wie z.B. Paragraphen) oder spezielle Elemente (über Element-Klassen oder -IDs deklariert) angewendet werden und sind vererbbar. Über CSS kann das Design einer AJAX-Anwendung interaktiv beeinflusst werden.
3. **Document Object Model (DOM)**[17]: Das Document Object Model beschreibt den Baum eines Dokuments, in unserem Falle HTML-Dokuments. Der Dokumentbaum kann beliebig bearbeitet werden, so kann z.B. über Elemente beliebigen Typs iteriert werden und Attribute geändert, einzelne Elemente über IDs angesprochen oder gar neue Elementknoten in den Baum eingefügt und auch wieder gelöscht werden. Das DOM wird per JavaScript angesprochen, das für die Bearbeitung entsprechende Funktionalitäten implementiert hat.
4. **XMLHttpRequest**: Über den XMLHttpRequest können vom Server Daten angefordert werden, die von der Anwendungslogik verwertet werden. Daten werden meist im XML-Format übertragen, können auch auch als Plain-Text zurückgegeben werden. Deshalb ist der Begriff XMLHttpRequest etwas unglücklich gewählt.

Zum XMLHttpRequest gibt es Alternativen wie z.B. das XMLDocument. Da diese jedoch wenig gebräuchlich sind, möchte ich mich in meiner Aufzählung auf den XMLHttpRequest beschränken.

### 3.2.1 DHTML

Die Technologien JavaScript, CSS und DOM fasst man unter dem Namen Dynamic HTML (DHTML) zusammen. Bei AJAX kommt der XMLHttpRequest dazu, mit dessen Hilfe asynchrone Kommunikation stattfinden kann.

## 3.3 Einstieg in die Technologien

An dieser Stelle möchte ich eine kurze Einführung in die verwendeten Technologien geben. Diese Einführung soll einen kurzen Überblick verschaffen und kann eine weiterführende Einarbeitung, falls noch nicht geschehen, nicht ersetzen. Am Ende des Papers finden sich Verweise auf Referenzen und Tutorials.

### 3.3.1 JavaScript

Beginnen möchte ich mit in JavaScript, die zentrale Technologie in AJAX.

JavaScript ist – wie der der Name impliziert - eine objektbasierte, jedoch nicht objektorientierte

15 Spezifikation: http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference
16 Spezifikation: http://www.w3.org/Style/CSS/
17 Spezifikation http://www.w3.org/DOM/

Skriptsprache, die nur schwach typisiert und universal einsetzbar ist. Sie ist nicht objektorientiert, da Vererbung in JavaScript über Prototyping realisiert ist. Dazu später mehr.

In unserem Fall dient JavaScript dazu, die Anwendungslogik abzubilden und den Zugriff auf das DOM zu ermöglichen.

**Schwach typisiert** bedeutet, dass Variablen implizit einen Typ zugewiesen bekommen, dieser sich jederzeit ohne Voraussetzung eines entsprechenden Casts wieder geändert werden kann. Hierzu ein Beispiel:

```
var x = 3.14;
x = 'pi';
```

Es handelt sich hierbei um gültigen Code. Die Variable, die als eine Fließkommazahl deklariert wird (wichtig ist das Schlüsselwort var), wird im nächsten Schritt implizit in eine String-Variable gecastet. Diese implizite Typumwandlung mag auf den ersten Eindruck einsteigerfreundlich und komfortabel sein, kann allerdings auch schwerwiegende Konsequenzen haben, wenn z.B. ungewollt Variablen überschrieben werden, weil versehentlich ein identischer Variablenname verwendet wird. In einer streng typisierten Sprache wie Java würde der Fehler durch eine Cast Exception auffallen.

**Interpretiert** bedeutet, dass der Code erst zur Laufzeit in Maschinensprache compiliert wird. Der Quellcode wird in unverändert vom Server zum Client übertragen und erst dort interpretiert. Dies hat zum einen den Nachteil, dass sich dies negativ auf die Laufzeit der Anwendung auswirkt, zum anderen aber auch dass der Code einer AJAX-Anwendung in ungeschützter Form vorliegt und vollständig einsehbar ist. Es kann also keine Closed-Source-Anwendung geschrieben werden.

**Universal einsetzbar** meint, dass die Kerntechnologie von JavaScript Unterstützung für Zahlen, Strings, Datum und Zeit, aber auch reguläre Ausdrücke, Zufallsgenerator etc. mitbringt. Außerdem sind Funktionalitäten zum Umgang mit dem XMLHttpRequest-Objekt und zum Zugriff auf das Document Object Model gegeben. Wir sind also bestens gerüstet, eine AJAX-Anwendung zu schreiben.

### 3.3.2 CSS

Zur Kontrolle der Anzeigeeigenschaften von Elementen in unserer Web-Anwendung verwenden wir durchgängig CSS. Mit Hilfe von CSS können Elementen Darstellungseigenschaften zentral zugewiesen, vererbt und auch überschrieben werden.

Diese strikte Trennung von Model und View (MVC-Pattern) ist vom W3C schon immer so gedacht; allerdings machte hierbei lange die fehlende bzw. mangelhafte Browser-Unterstützung einen Strich durch die Rechnung. So entstanden Fehlkonstrukte wie z.B. Blindtabellen zur Positionierung von Elementen, die eine Vermischung von Model und View mit sich bringen, sich leider aber bis heute in den Köpfen vieler Web-Entwickler festgesetzt haben und auch noch zusätzlich von WYSIWYG[18]-Entwicklungswerkzeugen propagiert werden.

Das ist jedoch Geschichte: In heutigen Browsern ist eine gute CSS-Unterstützung integriert; Probleme macht normalerweise nur der technisch veraltete Internet Explorer (das bekannteste Beispiel hierfür ist der Box-Model-Bug[19]). Es bleibt zu hoffen, dass sich dieser Zustand mit der kommenden Version 7,

18 WYSIWYG: What You See Is What You Get: Anwendung zum grafischen Erstellen einer Webseite, bei professionellen Entwicklern jedoch wegen ihres schlechten generierten HTML-Codes verpönt

19 http://de.selfhtml.org/css/formate/box_modell.htm#box_model_bug

die für Ende 2006 angekündigt ist, ändern wird.

Dennoch lässt sich mit kleinen Tricks (z.B. einer Browser-Weiche[20]) schon heute eine Web-Anwendung realisieren, in der der View vollständig vom Model getrennt ist. Auf HTML-Formatierungselemente wie

```
<p align="right">Text</p>
```

und Konstrukte wie die erwähnte Blindtabelle kann also vollständig verzichtet werden.

Es folgt eine Kurzeinführung in die Möglichkeiten von CSS.

#### 3.3.2.1 CSS-Selektoren

Mit diesem einfachen Codestück können z.B. alle Paragraphen blau formatiert werden:

```
p { color: blue; }
```

Mit dem Selektor p wird diese Eigenschaften auf alle <p>-Tags im HTML-Dokument angewendet. Soll diese Formatierung z.B. nur auf Paragraphen innerhalb einer Tabelle angewendet werden, wird dazu folgender Code verwendet:

```
table p { color: blue; }
```

#### 3.3.2.2 Klassen

Außerdem können mit Hilfe einer Klasse spezifische Eigenschaften vergeben werden. So wird z.B. mit Hilfe von

```
.fett { font-weight:900; }
```

der Paragraph

```
<p class="fett">Ein fetter Paragraph</p>
```

fett dargestellt.

#### 3.3.2.3 Invidualformate

Wird einem Element eine ID zugewiesen, z.B.

```
<p>Ein Paragraph mit einem <span id="besonders">besonderen Text</span></p>
```

so kann dieser Text mit Hilfe von

```
#besonders { border: 1px solid #CC0000; }
```

mit einem roten Rahmen versehen werden.

20 http://de.selfhtml.org/css/formate/box_modell.htm#workarounds

#### 3.3.2.4 Pseudo-Klassen

In CSS besteht außerdem die Möglichkeit, so genannte Pseudo-Klassen zu definierten. Darunter versteht man Elementklassen, die es eigentlich gar nicht gibt. Das bekannteste Beispiel hierfür ist die gesonderte Formatierung von Links bei Mouse-Over oder wenn diese bereits besucht wurden. Hierbei übernimmt der Browser die Verantwortung dafür, ob er diese Klassen umsetzen möchte.

Pseudo-Klassen beginnen mit einem Doppelpunkt und haben eine feste Namenskonvention. Hier ein Beispiel, bei dem alle besuchten Links schwarz gefärbt und bei Mouse-Over unterstrichen werden:

```
a {
  text-decoration:none;
  color:blue;
}
a:hover {
  text-decoration:underline;
}
a:visited {
  color:black;
}
```

#### 3.3.2.5 Farbnotationen

Farben können in CSS nach drei Mustern definiert werden.

Sechzehn Grundfarben können mit ihrem Namen ausgewiesen werden, z.B.

```
color:black;
```

Eine genauere Definition von Farben kann über die hexadezimale Schreibweise erreicht werden:

```
color:#FCAF00;  // Orange; Schema: #RRGGBB
color:#000000;  // schwarz
color:#FFFFFF;  // weiß
```

Diese beginnt mit einer Raute (#), gefolgt von jeweils zwei Zeichen pro Rot, Grün und Blau. Jeder der drei Farbtöne besitzt also 256 Schattierungen. Es wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Besteht die jeweiligen Farbanteile aus den gleichen Zeichen, also z.B.

```
color:#CC1133;
```

kann dies abgekürzt werden:

```
color:#C13;
```

Diese Farben werden in Bildverarbeitungsprogrammen häufig als „websichere“ Farben bezeichnet.

Die häufigste Schreibweise ist jedoch die herkömmliche Hexadezimalnotation.

#### 3.3.2.6 CSS-Style-Eigenschaften

In der verbreitetsten CSS-Version 2.1, die von allen gängigen Browsern unterstützt wird, gibt es rund 100 CSS-Eigenschaften. Hier gilt es, in einer Referenz[21] nachzuschlagen.

Ich möchte an dieser Stelle nur erwähnen, dass sich viele Eigenschaften entweder einzeln oder zusammengefasst notieren lassen. So definiert z.B. folgender Code

```
h1 {
  font-family: Arial;
  font-size: 12pt;
  font-weight: 900;
}
```

die gleichen Eigenschaften wie

```
h1 {
  font: 900 12pt Arial;
}
```

Das soll als Einführung in CSS genügen.

### 3.3.3 DOM

In einer AJAX-Anwendung werden wir sehr viel mit dem DOM zu tun haben. Die Struktur des vorliegenden HTML-Dokuments ist baumartig aufgebaut; mit Hilfe von JavaScript-Funktionalitäten kann darauf zugegriffen und die Struktur verändert werden.

Als ersten Einstieg ein Screenshot aus dem DOM-Inspector (in Mozilla Firefox integriert), der die Baumstruktur anschaulich abbildet:

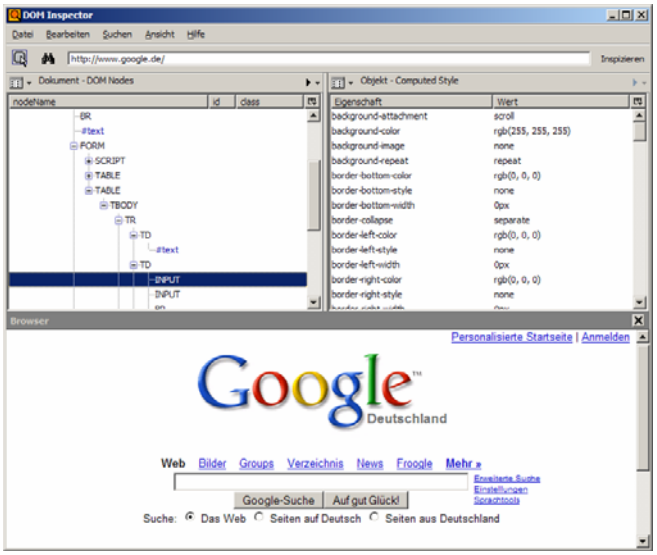


*Abbildung 9: Google im DOM-Inspector*

21 z.B. http://frixon.de/css21/css21_prop-visual.html

Im Folgenden möchte ich behandeln, wie mit JavaScript in das DOM eingegriffen werden kann.

### 3.3.3.1 Einbindung der JavaScript-Funktionalitäten

Wir haben uns für das MVC-Pattern entschieden und lagern deshalb alle JavaScript- (und Style-)Funktionalitäten in ein separate Dateien aus. Im Folgenden ein Beispiel-HTML-Dokument, in das wir die JavaScript- und CSS-Datei einbinden:

```
<html>
<head>
  <link rel='stylesheet' type='text/css' href='beispiel.css' />
  <script type='text/javascript' src='beispiel.js'></script>
</head>
<body>
  <p id='hallo'>hello</p>
  <div id='leer'></div>
</body>
```

Im CSS färben wir mit folgendem Code den 'hallo'-Paragraph rot:

```
p#hallo { color:red; }
```

### 3.3.3.2 Verändern von Style-Eigenschaften

Wir möchten eine JavaScript-Funktion schreiben, die uns diesen Paragraph blau färbt. Dazu schreiben wir folgenden Code:

```
function seitenTransformation() {
  document.getElementById('hallo').style.color = 'blue';
}
```

Wir holen uns also mit Hilfe der Funktion

```
document.getElementById('hallo')
```

eine Referenz auf das entsprechende Paragraphen-Element und setzen seine Stil-Information `color` auf `blue`. Die Funktion `getElementById(...)` greift natürlich nur dann, wenn wir einen Element eben diese ID vergeben haben. Ansonsten wird `null` zurückgegeben.

Anmerkung: Über Elemente gleicher Klasse kann nach Aufruf der Funktion `document.getElementsByClass(...)` iteriert werden.

So weit, so gut. Doch unsere Funktion wird noch nirgends benutzt. Hierzu soll ein Link in die Seite integriert werden, der bei Mausklick die Funktion aufruft. Hierzu verändern wir das Dokument wie folgt:

```
<html>
<head>
  [... ]
</head>
<body>
  <p><a href='javascript:seitenTransformation()'>Blau faerben</a></p>
  <p id='hallo'> [...]
</body>
```

Es können jedoch nicht nur einzelne Style-Informationen geändert werden, sondern auch die Elementklasse, um mehrere Änderungen zusammenzufassen. So könnten wir mit Hilfe von

```
document.getElementById('hallo').className = 'fett';
```

und

```
p.fett { font-weight:900; }
```

den Absatz als fetten Absatz deklarieren.

Viele HTML-Elemente haben weitere Attribute wie `onClick`, `onChange`, `onBlur` etc., die auch JavaScript-Funktionen auslösen können. Es empfiehlt sich hierzu die Lektüre einer entsprechenden Referenz[22]. Ein Verweis zu einer vollständigen CSS-Referenz findet sich im Anhang.

#### 3.3.3.3 Dem DOM-Tree neue Elemente hinzufügen und löschen

Zum Glück sind unsere Möglichkeiten nicht nur auf das Verändern von Style-Eigenschaften begrenzt. Mit Hilfe von JavaScript können Elemente dem Dokumentenbaum hinzugefügt oder auch wieder entfernt werden.

**Knoten hinzufügen**

JavaScript besitzt zum Erstellen von Knoten die beiden Methoden

```
document.createElement('elementname')
```

zum Erstellen eines neuen Elements und

```
document.createTextNode(text)
```

zum Erstellen eines Textknotens. Die neu erstellten Knoten können mit Hilfe von

```
document.appendChild(knoten)
```

an den Dokumentenbaum angehängt werden.

Im obigen Beispiel soll unserem leeren Bereich (`<div id='leer'></div>`) ein Paragraph mit einem Textinhalt hinzugefügt werden. Dazu erweitern wir unsere Funktion `seitenTransformation()` um folgenden Code:

```
function seitenTransformation() {
  [...]
  var lB = document.getElementById('leer');
  erweitereLeerenBereich(lB,'p','neuerPar','Inhalt des neues Paragraphs');
}
function erweitereLeerenBereich(knoten,element,elementID,inhalt) {
  var kindElement = document.createElement(element);
  kindeEement.setAttribute('id',elementID);
  knoten.appendChild(kindElement);
  var neuerKindKnoten = knoten.firstChild;

  var textElement = document.createTextNode(inhalt);
  neuerKindKnoten.appendChild(textElement);
}
```

22 http://de.selfhtml.org

Wir holen uns also zuerst eine Referenz auf den leeren Paragraphen und speichern sie in einer Variablen. Diese Variable übergeben wir gemeinsam mit den Parametern des Element-Typs und des Textinhalts an eine neue Funktion namens `erweitereLeerenBereich(...)`, die uns zuerst einen Paragraphenknoten erzeugt und mit einem entsprechenden ID-Attribut versieht. Da wir noch keine Referenz auf den neu erstellten Paragraphen haben, müssen wir uns diese erst holen und können daran dann den Textknoten anhängen.

Für solch einfache Fälle wie den obigen empfiehlt es sich, das innerHTML-Attribut eines Elements zu verwenden. So führt die Funktion

```
erweitereMitInnerHTML(knoten,element,inhalt) {
  knoten.innerHTML += '<'+element+'>'+inhalt+'</'+element+'>')';
}
```

zum selben Ergebnis wie `erweitereLeerenBereich(...)`. Wir setzen dazu einfach einen String zusammen, der in den Knoten eingehängt wird. Dieser String ist nach dem Parsen gültiger HTML-Code, das DOM wird nach dem Einhängen automatisch aktualisiert.

**Knoten löschen**

Natürlich können DOM-Knoten auch wieder gelöscht werden. Möchten wir z.B. im obigen Beispiel den Textknoten wieder löschen, so geschieht dies mit Hilfe von folgendem Code:

```
var knoten = document.getElementById('leer').firstChild;
var lB = document.getElementById('leer').removeChild(knoten);
```

Hierbei müssen wir uns zuerst Zugriff auf den zu löschenden Kindknoten verschaffen. Dazu holen wir uns den ersten (und einzigen) Kindknoten unseres leeren Divs und löschen eben diesen Kindknoten mit Hilfe von `removeChild()`.

Möchten wir alle Kindknoten eines DOM-Elements löschen, geschieht dies mit folgendem Code:

```
var anzahlKindKnoten = document.getElementById('knotenId').childNodes.length;
for(var i=0; i<anzahlKindKnoten; i++) {
  document.getElementById('leer').removeChild(document.getElementById('leer').
  childNodes[0]);
}
```

Wir bestimmen also zuerst die Anzahl der Kindknoten, um danach mit einer Schleife darüber zu iterieren. Bei jedem Durchlauf löschen wir den jeweils ersten Kindknoten. Der Zugriff über den Index i würde einen Fehler erzeugen, da nach jedem Schleifendurchlauf das DOM aktualisiert wird und es deshalb nach einigen Durchläufen keinen Index i=anzahlKindKnoten mehr geben würde.

Die Eigenschaft `childNodes[0]` ist identisch mit der Eigenschaft `firstChild`.

## 3.4 Asynchrone Kommunikation über XML-Technologien

### 3.4.1 Ein früher Ansatz: IFrame

Bevor heutige XML-Technologien erfunden und in Browser implementiert waren, gab es bereits eine Möglichkeit zur asynchronen Kommunikation zwischen Client und Browser: den sogenannten IFrame. Das „I“ steht für „inline“, was bedeutet, dass der Frame völlig unabhängig von einem Frameset

beliebig positioniert und gestaltet werden kann.

Der IFrame wurde erfunden, um Inhalte darzustellen. Die Verwendung zur asynchronen Kommunikation bedeutet eigentlich ein Missbrauch seines ursprünglichen Zwecks.

Da der IFrame in unserem Fall also keinen sichtbaren Inhalt darstellen sollte, war es gebräuchlich, ihm eine Breite und Höhe von 0 Pixeln zuzuweisen. Eine `display=none` Eigenschaft hatte den Nachteil, dass sie von manchen Browser schlichtweg (sinnvollerweise) wegoptimiert wurde.

Beim Laden der Seite rief man mit Hilfe des `onload`-Attributs des Body-Tags eine JavaScript-Funktion auf, die den Inhalt des IFrames änderte. In den IFrame konnte so erneut JavaScript-Code geladen werden, der bei OnLoad ausgeführt wurde. Mit Hilfe dieses Codes konnte z.B. der DOM-Tree wie bereits beschrieben bearbeitet werden.

Wie bereits erwähnt, kam die IFrame-Lösung bei einer frühen Version von Outlook Web Access zum Einsatz.

### 3.4.2 Die XML-Lösung: XMLDocument und XMLHttpRequest

Als Ablösung des IFrame-Ansatzes wurden die beiden Technologien XMLDocument und XMLHttpRequest geschaffen. Zwar sind beide noch nicht standardisiert, trotzdem jedoch in jeden modernen Browser integriert.

XMLDocument und XMLHttpRequest bieten ähnliche Funktionalitäten zur Kommunikation zwischen Client und Server. Der XMLHttpRequest bietet jedoch feinere Methoden zur Beeinflussung dieser Kommunikation und ist heute weiter verbreitet als XMLDocument. Ich möchte mich deshalb auf den XMLHttpRequest beschränken.

### 3.4.3 XMLHttpRequest

Ursprünglich begann Microsoft, den XMLHttpRequest als ActiveX[23]-Objekte in den Internet Explorer zu integrieren, die anderen Browserhersteller (Mozilla u.a.) zogen nach und implementierten diese als native Objekte.

Der Code zur Erzeugung eines solchen Objektes enthält deshalb eine Weiche zur Überprüfung der jeweiligen Browserimplementierung; der Umfang hält sich jedoch trotzdem in Grenzen. Wir implementieren den Code in einer eigenen JavaScript-Funktion `getXMLHttpRequest()`, die wir in unseren Projekten einbinden werden:

```
function getXMLHttpRequest() {
  var xRequest=null;
  if (window.XMLHttpRequest) {
        xRequest=new XMLHttpRequest();
  }
  else if (typeof ActiveXObject != "undefined"){
        xRequest=new ActiveXObject("Microsoft.XMLHTTP");
  }
  return xRequest;
}
```

23 Microsofts Komponentenmodell für den Internet Explorer, deshalb nur unter Windows verfügbar

Es wird also überprüft, ob der verwendete Browser das window.XMLHttpRequest-Objekt (Implementierung als natives Objekt) (Z.3) unterstützt und ansonsten auf ein ActiveX-Objekt (Internet Explorer-Implementierung als ActiveX-Objekt) (Z.6) geprüft. Diese Vorgehensweise hat Vorteile gegenüber einer selten vollständigen Überprüfung des Browserherstellers und der Versionsnummer.

Nach Ausführen der Funktion haben wir über eine Variable Zugriff auf das XMLHttpRequest-Objekt. Möchten wir einen Request abschicken, müssen wir nur noch die URL der Server-Seite angeben, die uns die gewünschten Daten zurückliefert. Dies geschieht mit folgendem Code:

```
function sendRequest(url,params,HttpMethod){
  if (!HttpMethod){
        HttpMethod="POST";
  }
  var req=getXMLHTTPRequest();
  if (req){
        req.open(HttpMethod,url,true);
        req.setRequestHeader("Content-Type","application/x-www-form-
  urlencoded");
        req.send(params);
  }
}
```

Ohne den Code im Beispiel zu erklären, ist auf den ersten Blick ersichtlich, dass wir uns für den richtigen Umgang mit dem XMLHttpRequest einen Überblick über die Funktionsweise von HTTP verschaffen müssen.

### 3.4.4 Einschub: Überblick über HTTP

Beim Entwickeln einer klassischen Web-Anwendung müssen uns im Normalfall nicht mit HTTP beschäftigen. Eigentlich begegnet es uns nur bei der Definition von Hyperlinks und im action-Attribut eines Formulars, das den Wert GET oder POST annehmen kann.

AJAX gibt uns die Möglichkeit, tiefer in das Protokoll einzusteigen und eröffnet uns damit interessante Möglichkeiten.

HTTP besteht im Grunde aus einem Request an einen Server und dessen Response. Beide Nachrichten sind reine Textstrings, die mit einem Header versehen sind. Die Seite Web-sniffer.net[24] bietet die Möglichkeit, den Request und die Response einer beliebigen Seitenanfrage anzuzeigen. Fragen wir beispielsweise Google an, ergibt sich folgender Request-Header:

```
GET / HTTP/1.1
Host: www.google.de
Connection: close
Accept-Encoding: gzip
Accept:
  text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=
  0.8,image/png,*/*;q=0.5
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.8.0.3)
  Gecko/20060426 Firefox/1.5.0.3 Web-Sniffer/1.0.24
Referer: http://web-sniffer.net/
```

24 www.web-sniffer.net

und folgender Response-Header:

```
HTTP Status Code: HTTP/1.1 200 OK
Cache-Control:privateCRLFContent-Type:text/html
Set-
  Cookie:PREF=ID=b09ab70b9af86f71:TM=1148848886:LM=1148848886:S=sc_Rilv_4KZ_n6
  -m; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.de
Content-Encoding:gzip
Server:GWS/2.1
Content-Length:1563
Date:Sun, 28 May 2006 20:41:26 GMT
```

Betrachten wir den Request, so erhalten wir Informationen darüber, dass Google mit GET angefragt wurde, als User-Client Mozilla Firefox verwendet wird und können browserspezifischen Daten wie Spracheinstellungen etc. ablesen. Das Referer-Feld gibt Auskunft über den Ursprung des Verweises. Dieses Feld ist vor allem bei statistischen Auswertungen interessant.

In der Response können wir ablesen, dass ein Cookie mit definierten Attributen angelegt werden soll. Der Inhalt ist mit der gzip-Methode codiert; es wird zusätzlich die Content-Länge übertragen. Die interessanteste Information steht jedoch in der ersten Zeile: der HTTP-Status-Code.

Wenn wir AJAX-Anwendungen entwickeln, kommt es vor allem auf diesen Status-Code an. Interessanterweise bekommen wir bei einer asynchronen Anfrage nämlich nicht sofort eine Response mit dem Status-Code OK, es werden auch Status-Codes wie `loading` etc. übertragen. Diese möchten wir mit JavaScript behandeln. Dies führt uns zurück zu unserem XMLHttpRequest.

### 3.4.5 Callback-Methoden des XMLHttpRequests

Setzen wir eine asynchrone Abfrage an den Server ab und erhalten die Response dazu, müssen wir natürlich auch eine Callback-Methode registrieren, die die Response abfängt und verarbeitet. Eben in dieser Methode jonglieren wir mit den HTTP-Status-Codes, die uns vom Server über den Browser übergeben werden.

In unserem IFrame-Beispiel war die Methode `window.onLoad()` eine solche Callback-Methode. Für unseren XMLHttpRequest registrieren wir eine eigene Callback-Methode. Heißt der Request `req` und unsere Callback-Methode `callBackMethode()`, geschieht dies mit folgendem Codefragment:

```
req.onreadystatechange=callBackMethode;
```

Grundsätzlich brauchen wir aber eine Funktion, die die gesamte Anwendung anstößt. Dies kann durch Klick auf einen Link (der eine Funktion wie z.B. `starteAnwendung()` aufrufen könnte) geschehen oder per `window.onLoad()`.

### 3.4.6 Methoden und Eigenschaften des XMLHttpRequests

Bevor wir unsere bisherigen Kenntnisse in einem Komplettbeispiel einsetzen, sollen die Methoden und Eigenschaften des XMLHttpRequests in tabellarischer Form zusammengefasst werden.

### 3.4.6.1 Methoden

| *Methode* | *Beschreibung* |
|---|---|
| abort() | Abbruch der Server-Anfrage |
| getAllResponseHeaders() | Gibt den vom Server gesendeten Header als String zurück |
| open("method", "URL"[, asyncFlag[, "userName"[, "password"]]]) | Verbindung zum Server öffnen. Dabei können folgende Parameter gesetzt werden:<br>• Methode: GET, POST, PUT, HEAD<br>○ Normalerweise GET verwenden, bei größeren Datenmengen POST<br>○ HEAD liefert nur den Header ohne Body<br>• URL: relativer oder absoluter Pfad<br>• Asynchrone Kommunikation: true/false<br>○ wir verwenden true<br>• Benutzer/Passwort |
| send(content) | Daten an den Server übertragen<br>• content: null bei GET oder Anfrage-String bei POST |
| setRequestHeader("label", "value") | Den Header beeinflussen (z.B. Content-Type) |
| setMimeType(„mimetype“) | Mime-Type der geforderten Response setzen, z.B. „text/xml“<br>wird von Internet Explorer nicht unterstützt |

### 3.4.6.2 Eigenschaften

| *Eigenschaft* | *Beschreibung* |
|---|---|
| onreadystatechange | Event-Handler wird jedes Mal aufgerufen, wenn sich der Verbindungsstatus (readyState) ändert |
| readyState | Aktueller Status der Verbindung<br>• 0 = Verbindung noch nicht geöffnet (open() verwenden)<br>• 1 = keine Anfrage gesendet (send() verwenden)<br>• 2 = Anfrage gesendet, Antwort-Header und -Status können abgerufen werden<br>• 3 = Lade. Bisherige Inhalte können in responseText gelesen werden<br>• 4 = Laden abgeschlossen. Bei erfolgreichem Abschluss status=200 gesetzt |
| responseText | Enthält die vom Server gesendeten Daten als Text |

| | |
|---|---|
| responseXML | Enthält die vom Server gesendeten Daten als XML (falls im XML-Format vorliegend) |
| status | HTTP-Status als Zahl (200=erfolgreich, 404=nicht gefunden) |
| statusText | HTTP-Status als Text (z.B. „Not found“) |

## 3.5 Erstes Komplettbeispiel

Im Folgenden ein kleines Komplettbeispiel, das ich dem Buch „AJAX in Action“ entliehen habe.

In diesem Beispiel implementieren wir die Methode `onreadystateChange()` und möchten per XMLHttpRequest einen Text vom Server anzufragen. Während des Ladevorgangs wird eine Statusanzeige mit dem jeweils zurückgegebenen HTTP-Status-Code ausgegeben. Im Beispiel-Code werden viele Dinge, die bisher angesprochen wurden, gleichzeitig implementiert.

```
<html>
<head>
<script type='text/javascript'>
      var req=null;
      var console=null;

      var READY_STATE_UNINITIALIZED=0;
      var READY_STATE_LOADING=1;
      var READY_STATE_LOADED=2;
      var READY_STATE_INTERACTIVE=3;
      var READY_STATE_COMPLETE=4;

      function sendRequest(url,params,HttpMethod){
        if (!HttpMethod){
              HttpMethod="GET"; // (3)
        }
        req=initXMLHTTPRequest(); // (4)
        if (req){
              req.onreadystatechange=onReadyState; // (7)
              req.open(HttpMethod,url,true); // (5)
              req.setRequestHeader
              ("Content-Type", "application/x-www-form-urlencoded");
              req.send(params); // (6)
        }
      }

      function initXMLHTTPRequest(){
        var xRequest=null;
        if (window.XMLHttpRequest){
        xRequest=new XMLHttpRequest();
        }
        else if (window.ActiveXObject){
              xRequest=new ActiveXObject("Microsoft.XMLHTTP");
        }
        return xRequest;
      }

      function onReadyState(){
        var ready=req.readyState;
```

```
            var data=null;
            if (ready==READY_STATE_COMPLETE){ // (8)
                  data=req.responseText;
            }else{ // (9)
                  data="loading...["+ready+"]";
            }
            toConsole(data);
          }

          function toConsole(data){
            if (console!=null){
                  var newline=document.createElement("div");
                  console.appendChild(newline);
                  var txt=document.createTextNode(data);
                  newline.appendChild(txt);
            }
          }
          window.onload=function(){
            console=document.getElementById('console'); // (1)
            sendRequest("data.txt"); // (2)
          }
      </script>
      </head>
      <body>
      <div id='console'></div></body>
```

Ich möchte kurz die Funktionsweise der obigen Anwendung zusammenfassen.

Wir erzeugen eine HTML-Seite mit einem Div (`id='console'`), in den wir den Text ausgeben möchten. Beim Laden der Seite wird automatisch die Funktion `window.onload` gerufen, die den Div in einer globalen Variable speichert (1) den Request anstößt (2). Dieser setzt die HTTP-Methode auf `GET` (3), erzeugt uns mit ein browserspezifisches XMLHttpRequest-Objekt (4) und schickt den Requst nach Festlegen einiger Attribute (5) an den Server (6). Wir registrieren außerdem mit Hilfe von

```
req.onreadystatechange=onreadyState;
```

die Methode `onreadyState()` als Callback-Methode für den Request `req` (7). Diese Methode gibt uns die jeweiligen Status-Codes aus, die mit globalen Konstanten verglichen (8) werden. Bei Status `OK` (`8`) wird mit der Funktion `toConsole()` ein Textknoten im Div erzeugt und der Inhalt (`loading...[x]` oder Inhalt des Textdokuments) ausgegeben (9).

### 3.5.1 Ausgabe

Sowohl Internet Explorer 6 als auch Mozilla Firefox 1.5 erzeugen folgende Ausgabe:

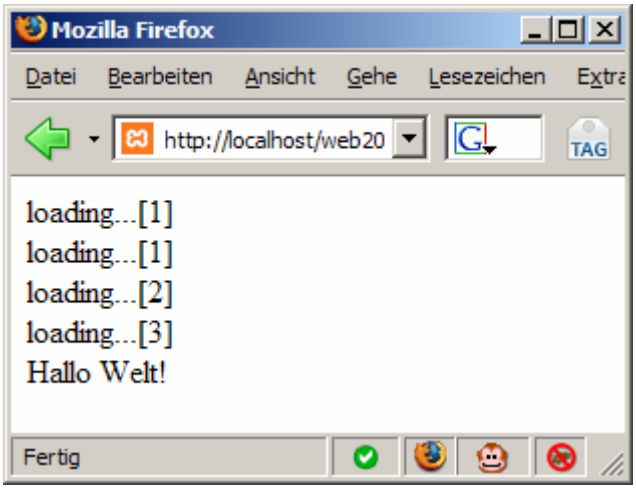


*Abbildung 10: Ausgabe des Beispiels*

Der Browser reicht also zweimal den Status-Code `loading` und einmal `loaded` weiter. Nach `OK` wird der Text ausgegeben.

### 3.5.2 Gratulation!

Gratulation zu unserer ersten AJAX-Anwendung! Mit relativ geringem Aufwand können wir bereits nach kurzer Zeit erste Schritte mit AJAX gehen.

## 3.6 Und jetzt im großen Stil?

Obiges Beispiel ist ein typisches AJAX-Widget. Unter einem AJAX-Widget versteht man eine relativ überschaubare AJAX-Anwendung, die in eine klassische Web-Anwendung integriert ist. AJAX übernimmt hier nur eine Teilfunktionalität der Anwendung. Die üblichen Einschränkungen einer klassischen Anwendung (wie z.B. Page-Reload) bleiben bestehen.

Mit AJAX ist es auch möglich, eine souveräne Anwendung (wie z.B. einen PIM-Client) webbasiert abzubilden. Die Anforderungen sind hier um einiges höher: Unser Code muss sehr viel mehr Funktionalitäten integriert haben, sehr gut strukturiert sein und vor allem unter Umständen stundenlang lauffähig, was eine durchdachte Speicherverwaltung erfordert.

Um etwas vorwegzugreifen: In JavaScript kommt ein Garbage Collector, ähnlich zu Java, zum Einsatz. Auf die Konsequenzen für den Entwickler möchte ich später eingehen.

Zwischen einer rein klassischen Implementierung und einer vollständigen AJAX-Implementierung gibt es Zwischenstufen; das Verhältnis kann beliebig schwanken.

Was auf jeden Fall festgehalten werden kann: Die Einstiegshürde in AJAX ist relativ niedrig. Die einzelnen Technologien (JavaScript, CSS, DOM und XMLHttpRequest) sind schnell erlernbar. Im Internet gibt es umfangreiche Dokumentationen und Tutorials.

Der Knackpunkt ist: Wie arbeiten diese Technologien Hand in Hand zusammen? Wie sollte man bei

größeren Projekten strukturiert vorgehen? Gibt es Vorgehensmuster (Patterns)?

Diesen Fragen möchte ich mich im Folgenden widmen. Bevor wir jedoch einsteigen, ist es an der Zeit, vorzustellen, welche Werkzeuge für die Entwicklung einer AJAX-Anwendung benötigt werden.

# 4 AJAX Tools

## 4.1 Entwicklung

### 4.1.1 Der Texteditor

Wie beim Erstellen einer klassischen Web-Anwendung reicht im Grunde ein Texteditor, um die komplette Anwendung zu erstellen. Es empfiehlt sich jedoch zumindest Unterstützung durch Code-Highlighting und -vervollständigung. Empfehlenswerte Tools sind z.B. Quanta Plus[25] unter Linux oder Phase5[26] unter Windows.

### 4.1.2 WYSIWYG

Bekannte Webeditoren, wie z.B. Adobe Dreamweaver, die WYSIWYG bieten, können zum komfortablen Erstellen der Benutzeroberfläche verwendet werden. Bei der Wahl des Programms ist auf jeden Fall darauf zu achten, dass standardkonformer Code (z.B. bezüglich CSS-Einsatz) erzeugt wird.

### 4.1.3 Die Idealkombination

Als guten Kompromiss sehe ich den Einsatz eines solchen Web-Entwicklungswerkzeuges zur Erstellung eines groben Gerüstes, das dann von Hand weiterentwickelt wird. So kann ein technisch sauberes Ergebnis in trotzdem überschaubarer Zeit erstellt werden.

## 4.2 Server

Eine AJAX-Anwendung ist nicht für sich allein lauffähig; es wird ein Server benötigt. Dieser kann entweder remote verfügbar sein oder lokal installiert werden.

Möchten wir die Server-Seite z.B. mit PHP realisieren (zur Server-Seite später mehr), empfiehlt es sich, zum Testen einen lokalen Server, meist Apache, zu installieren. Ein empfehlenswertes Komplettpakete ist z.B. XAMPP[27] für Windows. Solche Komplettpakete bringen auch gleich eine Datenbank (MySQL) mit, die wir für unsere Beispielapplikation benötigen.

Zum Bereitstellen der Anwendung auf einem Remote-Server kann ein beliebiges FTP-Programm

25 http://quanta.kdewebdev.org/
26 http://www.qhaut.de
27 Download unter www.apachefriends.org

verwendet werden.

## 4.3 Testing: Browser und Erweiterungen

Als wahrscheinlich komfortabelster Browser zum Testen einer AJAX-Anwendung ist Mozilla Firefox[28] (momentan in der Version 1.5) zu empfehlen. Dies zeigt auch die steigende Verbreitung: Das W3C[29] misst momentan eine Verbreitung von ca. 25%; andere Messungen[30] ergeben eine Verbreitung zwischen 10% und 30% (Europa).

Firefox ist zum einen empfehlenswert, da er neueste Technologien wie JavaScript 1.6 unterstützt und Seiten fehlerfrei darstellt (valides HTML/CSS natürlich vorausgesetzt). Zum anderen greift er Entwicklern von Web-Anwendungen mit von Haus aus implementierten Werkzeugen, die sich durch Addons erweitern lassen, unter die Arme.

Erwähnen möchte ich an dieser Stelle die JavaScript-Konsole (integriert), die mit wirklich lesbaren und sinnvollen Meldungen auf die Sprünge hilft, wenn der JavaScript-Code Fehler enthält.

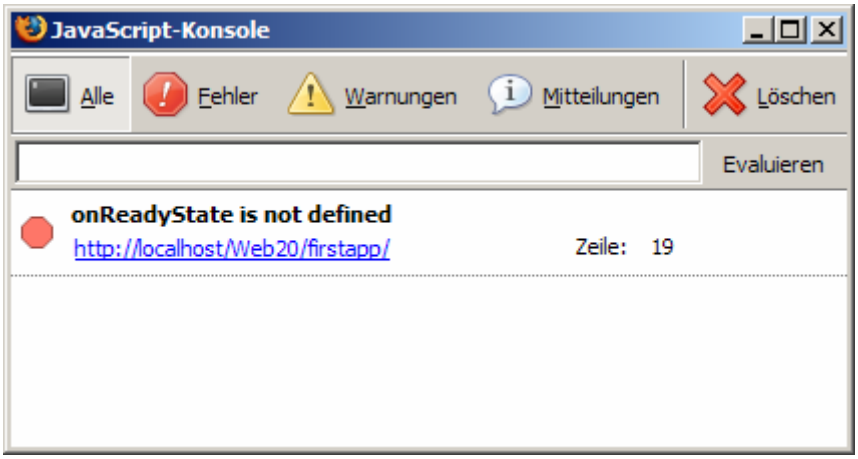


*Abbildung 11: JavaScript-Konsole*

Bei umfangreichen Projekten ist ein JavaScript-Debugger unabdingbar. Für empfehlenswert halte ich Firebug[31], der alle Funktionen eines gängigen Debuggers bietet. Auch diese Anwendung hat eine Konsole integriert.

28 Download unter www.mozilla.com
29 http://www.w3schools.com/browsers/browsers_stats.asp
30 http://www.xitimonitor.com/etudes/equipement13.asp
31 Download unter https://addons.mozilla.org/firefox/1843/

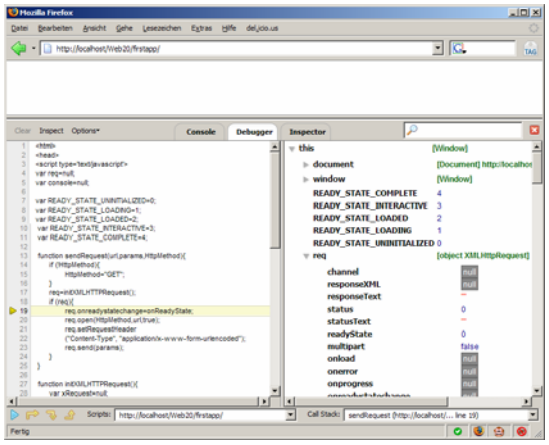

*Abbildung 12: Debugger Firebug*

Mit Hilfe des DOM-Inspectors (integriert) kann das dargestellte Dokument inspiziert werden, interessanterweise nicht nur mit den im CSS festgelegten Werten, sondern auch mit den tatsächlich berechneten („Computed Values“). So können z.B. Vererbungsfehler aufgedeckt werden.

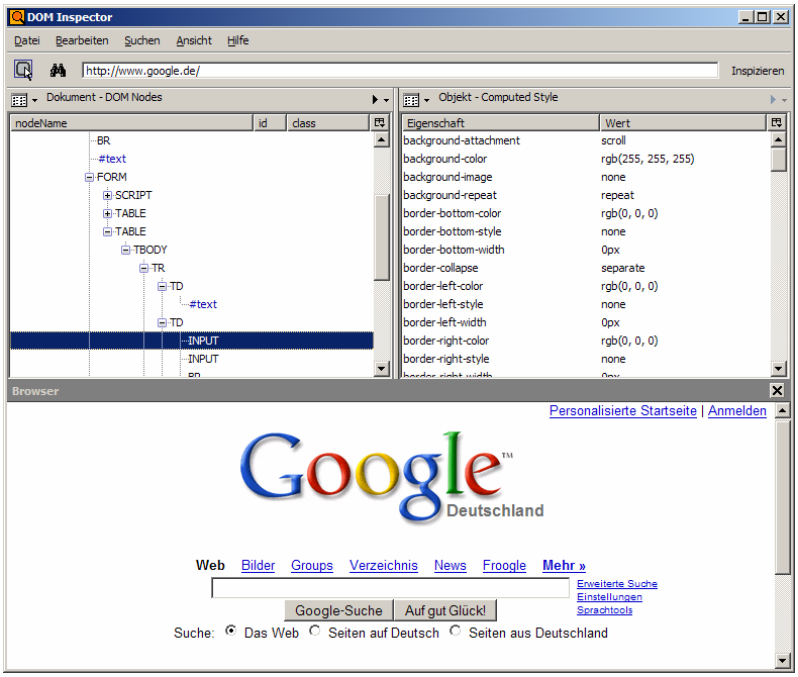


*Abbildung 13: DOM-Inspector*

Und schließlich sei als besonderes „Schmankerl“ die Web Developer Toolbar[32] empfohlen, die zahlreiche Möglichkeiten wie das Deaktivieren oder direkte Bearbeiten von CSS-Eigenschaften, Direktlink zur W3C-Validierung und jede Menge weiterer Information bietet.

32 Download unter https://addons.mozilla.org/firefox/1843/

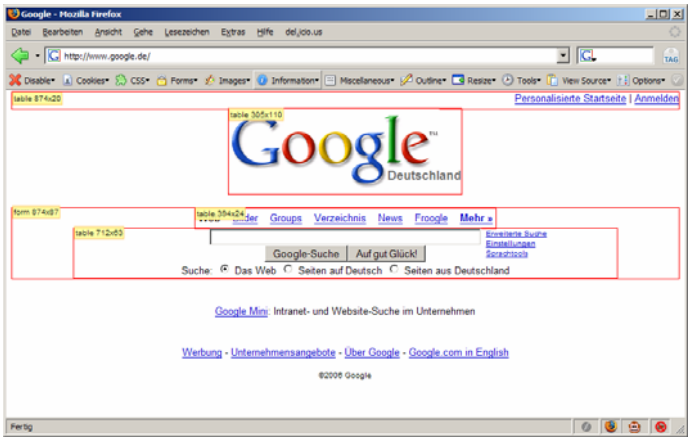


*Abbildung 14: Web Developer Toolbar: Markierung von Blindtabellen und deren Anzeigeeigenschaften*

# 5 AJAX im Einsatz - Strukturierte Vorgehensweise

Eine AJAX-Anwendung kann, wie bereits dargestellt, relativ einfach erstellt werden – zumindest so lange sich der Code-Umfang in einem überschaubaren Bereich bewegt. Doch unser eigentliches Ziel ist es nicht, AJAX-Widgets zu erstellen, sondern eine souveräne Anwendung komplett AJAX-basiert abzubilden.

Bei einem typischen Proof-Of-Concept soll schnell eine gewisse Funktionalität getestet werden. Häufig wird der Code unüberlegt geschrieben und ist hinterher nicht in eine bestehende Anwendung integrierbar, da z.B. Objektorientierung vernachlässigt wurde und Variablenkonflikte bestehen.

Ein weiteres Problem ist, dass sich während des Entstehungsprozesses einer Anwendung häufig die Anforderungen ändern. Hat man falsch geplant, kann das Einfügen einer neuen Funktionalität in einem Desaster enden.

Wichtig ist deshalb das Refactoring bestehenden Codes, um ihn z.B. in eine objektorientierte Anwendungslogik zu überführen, und das Einhalten von grundsätzlichen Verhaltensmustern, den sogenannten Patterns. Darauf möchte ich jetzt näher eingehen.

## 5.1 Refactoring

Unter Refactoring versteht man, bestehenden Code neu zu schreiben, ohne neue Funktionalität hinzuzufügen. Nicht die Erweiterung der Anwendung steht im Vordergrund, sondern die Strukturierung derselben.

Oft bezieht sich das Refactoring auf das Einführen von Patterns, die bisher vernachlässigt wurden. Dies dient zur besseren logischen Strukturierung des Codes.

Beim Refactoring wird außerdem prozeduraler Code in objektorientierten Code überführt. Dies macht ihn skalierbarer und Konflikte, die z.B. bei der Verwendung globaler Variablen auftreten können, werden vermieden. Auch wird die Les- und Wartbarkeit dadurch verbessert.

## 5.2 Objektorientierte Programmierung

Wie bereits in der Einführung erwähnt, handelt es ich bei JavaScript um eine objektorientierte Skriptsprache. Allerdings bestehen große Unterschiede zu Java.

Auch in JavaScript können Klassen mit einer festen Struktur angelegt werden. Diese werden jedoch nicht in einer eigenen Datei abgelegt. Vielmehr wird eine Variable vom Typ `Object` angelegt, der implizit neue Attribute hinzugefügt werden können. Diese sind wie alle Variablen in JavaScript lose typisiert. Das Problem von Variablenkonflikten bleibt also bestehen, da keine expliziten Casts vorgenommen werden müssen.

### 5.2.1 Objektorientiertes Refactoring am Beispiel

Als Beispiel für objektorientiertes Refactoring soll unser obiges Komplettbeispiel (wieder dem Buch „AJAX in Action“ entnommen) überarbeitet werden. Es folgt der neue Code, der im Folgenden erklärt wird. Anhand des Beispiels werden wir interessante Eigenheiten von JavaScript kennnenlernen und Patterns, die danach vorgestellt werden, wiedererkennen.

```
var net=new Object(); // (1)

net.READY_STATE_UNINITIALIZED=0; // (2)
net.READY_STATE_LOADING=1;
net.READY_STATE_LOADED=2;
net.READY_STATE_INTERACTIVE=3;
net.READY_STATE_COMPLETE=4;

net.ContentLoader=function(url,onload,onerror){ // (3)
  this.url=url; // (5)
  this.req=null;
  this.onload=onload;
  this.onerror=(onerror) ? onerror : this.defaultError; // (4)
  this.loadXMLDoc(url);
}

net.ContentLoader.prototype={
        loadXMLDoc:function(url){
               if (window.XMLHttpRequest){ // (6)
                      this.req=new XMLHttpRequest();
               } else if (window.ActiveXObject){
                      this.req=new ActiveXObject("Microsoft.XMLHTTP");
               }
               if (this.req){
                      try{
                             var loader=this;
                             this.req.onreadystatechange=function(){ // (7)
                                    loader.onReadyState.call(loader);
                             }
                             this.req.open('GET',url,true); // (8)
                             this.req.send(null); // (9)
                      }catch (err){
                             this.onerror.call(this);
                      }
               }
        },
```

```
        onReadyState:function(){
          var req=this.req; // (10)
          var ready=req.readyState; // (11)
          if (ready==net.READY_STATE_COMPLETE){ // (12)
                 var httpStatus=req.status;
                 if (httpStatus==200 || httpStatus==0){ // (13)
                        this.onload.call(this);
                 }else{
                        this.onerror.call(this);
                 }
          }
        },

        defaultError:function(){
               alert("error fetching data!"
               +"\n\nreadyState:"+this.req.readyState
               +"\nstatus: "+this.req.status
               +"\nheaders: "+this.req.getAllResponseHeaders());
        }
  }
```

Die zentrale Änderung an unserer Anwendungslogik findet sich gleich in der ersten Zeile: Wir legen ein Objekt `net` an, dem wir später unser Request-Objekt anhängen (1). Dem `net`-Objekt weisen wir die Ready-Status-Codes, die zuvor als globale Variablen definiert waren, als Objektattribute zu (2).

Als nächstes folgt der Konstruktor für unser Objekt `ContentLoader` (3), mit dem wir den Inhalt der Textdatei vom Server laden möchten. [Anmerkung: Es ist also möglich, dem `net`-Objekt weitere Unterobjekte hinzuzufügen.] Als Parameter können dem Konstruktor die URL der Textdatei, der Name der Callback-Methode und der Name der Fehlerbehandlungsmethode übergegeben werden. Aus der Anwendungslogik des Konstruktors ergibt sich, dass bei der Parameter `onerror` nicht verpflichtend ist. Bei fehlendem Parameter `onerror` wird der Default-Error-Handler verwendet (4). Implizit spreche ich hier das Thema variable Anzahl der Parameter von JavaScript-Methoden an, das ich weiter unten behandle.

Im Konstruktor werden die Callback-Methode und der Error-Handling-Methode als Objektattribute gespeichert (5). Beim Error-Handler wird auf die Existenz des entsprechenden Parameters geprüft (die verwendete Notierung ist eine Abkürzung für `if-then-else`) und gegebenenfalls der Default-Error-Handler registriert.

Mit dem Prototype-Konstrukt definieren wir uns eine Art Klasse, die die drei Methoden `loadXMLDoc()`, `onReadyState()` und `defaultError()` enthält.

In der Methode **`loadXMLDoc()`** erzeugen wir uns zuerst ein browserspezifisches XMLHttpRequest-Objekt (6). Diesem Objekt weisen wir die Callback-Methode `onReadyState()` zu (7), die auf die HTTP-Status-Codes reagiert. Interessanterweise wird hier die Funktion `onReadyState.call(loader)` verwendet. Wir können also jede beliebige Methode mit

```
pfad.zum.methodenname.call(parameter)
```

aufrufen. Als Parameter übergeben wir in unserem Fall eine Referenz auf unser `loader`-Objekt, um später Zugriff auf den Request zu haben.

Danach wird der Request abgeschickt (8) und `'null'` als Content gesendet (9), da es sich um eine GET-Anfrage handelt.

Bei jedem Aufruf der Callback-Methode, also bei jeder Rückgabe eines HTTP-Status-Codes, holt sich die Methode **`onReadyState()`** eine Referenz auf den Request (10) und dessen Attribut `readyState` (11). Der Ready-Status wird auf `loaded` geprüft (12). Ist zusätzlich der HTTP-Code `OK` (also 200) (13), wird die Callback-Methode aufgerufen, die wir als Parameter `onload` im Konstruktor des Objekts übergeben haben.

Diese Methode (nicht im Code implementiert) übernimmt die Ausgabe des Textstrings. Auch dieser Aufruf erfolgt über `call`; zusätzlich übergeben wir eine Referenz auf unser `loader`-Objekt. Diese Referenz wird später benötigt, um Zugriff auf den Request zu haben.

Bevor ich den obigen Objekt-Code in ein komplettes Beispiel einbinden möchte, sollen die beiden Eigenheiten von JavaScript behandelt werden, die einem bei näherer Betrachtung auffallen.

### 5.2.2 Methodenaufruf mit variablen Parametern

In unserem Beispiel wird einmal ein Konstruktor mit drei Parametern verwendet und ich werde ihn später mit nur zwei Parametern aufrufen. Außerdem sind parameterlose Methoden implementiert, die ich im Beispiel mit Parameter(n) aufrufe. Warum funktioniert das?

In JavaScript gilt die Regel, dass Methoden und Konstruktoren mit variablen Parametern aufgerufen werden können. Rufen wir eine Methode also mit weniger Parametern als vorgesehen auf, werden die Parameter von links nach rechts aufgefüllt. Das bereitet uns in keinem Fall Probleme, da wir lose Typisierung haben. Trotzdem können sich semantische Fehler einschleichen, da eventuell erwartete Parameter den Wert `null` haben, also nicht belegt sind.

Rufen wir eine Methode mit mehr Paramtern auf als vorgesehen, ist es abhängig von der Methodenimplementierung, ob sie verwendet werden. Übergeben wir wie in unserem Beispiel nur einen Parameter, haben wir in der Methode Zugriff darauf über `this`. Unschöne Effekte können dabei allemal auftreten.

Es ist wie bei der losen Typisierung: Was auf den ersten Blick einfach und benutzerfreundlich aussieht, kann bei umfangreichem Code schnell zu semantischen Problemen führen. Das Geheimnisprinzip wird verletzt; Implementierer und Verwender einer Methode müssen sich abstimmen. Man sollte sich also der Gefahren bewusst sein und wenn möglich auf variable Parameter verzichten bzw. sie nur eingeschränkt verwenden.

### 5.2.3 Vererbung per Prototyping

Vererbung funktioniert in JavaScript anders als von Java gewohnt, nämlich per Prototyping.

Jedes vom Function-Prototyp abgeleitete Objekt (siehe obiges Beispiel) hat neben einem Konstruktor die Eigenschaft `prototype`. Über `prototype` können gemeinsame Eigenschaften aller Objekte, die mit diesem Konstruktor erzeugt wurden, festgelegt werden.

In obigem Beispiel definieren wir darüber die drei Objekt-Methoden `loadXMLDoc()`, `onReadyState()` und `defaultError()`.

Über Prototyping ist außerdem (mehrstufige) Vererbung möglich. Ein anschauliches Beispiel, das sich

in abgewandelter Form in Wikipedia[33] findet:

Wir definieren einen Objekttyp Fahrzeug:

```
function Fahrzeug(Fabrikat) {
   this.Fabrikat = Fabrikat;
   this.Eigenschaft1 = "Beispielwert";
}
```

Davon leiten wir den Typ PKW ab:

```
function PKW (Fabrikat) {
   this.constructor(Fabrikat);
   this.Eigenschaft2 = "Beispielwert";
}
PKW.prototype = new Fahrzeug();
```

Erzeugen wir nun neue PKW über

```
var Astra = new PKW(„Opel Astra“);
var Golf = new PKW(„Volkswagen Golf“);
```

so wird mit obigem Konstruktor die abgeleitete Objekteigenschaft `Fabrikat` gesetzt. Diese Eigenschaft ist wie gewohnt über die Punkt-Notation erreichbar.

Um das Beispiel abzurunden, definieren wir weitere Objekteigenschaften und Methoden, die sich auf alle Objekte dieses Typs auswirken:

```
PKW.prototype.Radanzahl = 4;
PKW.prototype.zeigeRadanzahl = function () {
   window.alert(this.Fabrikat + " hat " + this.Radanzahl + " Räder.");
};
```

Zum Testen verwenden wir folgenden Code:

```
Astra.zeigeRadanzahl(); // Ausgabe: „Opel Astra hat 4 Räder.“
Golf.zeigeRadanzahl(); // Ausgabe: „Volkswagen Golf hat 4 Räder.“
```

### 5.2.4 Einbindung in komplettes Dokument

Die Einbindung unseres obigen Beispiels in ein komplettes Dokument erfolgt mit nachfolgendem Code. Darin sind eine Callback-Methode zur Ausgabe der Textdatei und die `window.onload()`-Funktion zum Anstoßen der Anwendung integriert. Der Text wird als Elementknoten `<p>` in den div mit der ID `console` eingehängt, was unserem vorigen Beispiel entspricht.

```
<html>
<head>
<script type='text/javascript'>
var console=null;

// ------
// Objektorientierter Code aus obigem Beispiel
// ------

function toConsole(){
```

33 http://de.wikipedia.org/wiki/Javascript

```
  if (console!=null){
        var data = this.req.responseText;

        var newline=document.createElement("p");
        console.appendChild(newline);
        var txt=document.createTextNode(data);
        newline.appendChild(txt);
  }
}

window.onload=function(){
  console=document.getElementById('console');
  var loader=new net.ContentLoader('data.txt',toConsole);

}

</script>
</head>
<body>
  <div id='console'></div>
</body>
```

Wir fügen also unseren objektorientierten Code in den Skriptbereich unserer Webseite ein. Die strikte Trennung nach dem Model-View-Controller-Pattern, die eine Auslagerung der JavaScript-Datei fordern würde, soll der Einfachheit halber vernachlässigt werden.

Die Methode **`window.onload()`** stößt den Ladevorgang an, indem es einen neuen Content-Loader erzeugt und eine Referenz in der Variablen `loader` speichert. Wie versprochen rufen wir den Konstruktor tatsächlich nur mit zwei Parametern auf: Die zu lesende Textdatei und die Callback-Methode `toConsole()` zur Ausgabe des Inhalts. Es wird also der Default-Error-Handler verwendet.

Unsere Callback-Methode **`toConsole()`** wird mit dem Parameter `loader` aufgerufen, auf den wir uns Zugriff über `this` verschaffen. Der Inhalt der Textdatei steht im Requestattribut `responseText`. Daraus generieren wir uns einen Paragraphen-Knoten, in den wir den Text als Kindknoten einhängen. Den Paragraphen-Knoten hängen wir in unseren Div `console` ein, auf den wir über die globale Variable `console` eine Referenz erhalten.

Trotz des umfangreichen Codes erhalten wir folgende bescheidene Ausgabe:


*Abbildung 15: Ausgabe unseres objektorientierten Komplettbeispiels*

## 5.3 Patterns

Wie bereits erwähnt, finden sich in unserem objektorientierten Beispiel implizit Patterns. Im Folgenden möchte ich einige Patterns vorstellen. Diese sind für eine strukturierte AJAX-Anwendung unabdingbar. Im Folgenden eine Auswahl an Patterns, die ich dem Buch „Entwurfsmuster“[34] entnommen habe.

### 5.3.1 Facade-Pattern

Das Facade-Pattern dient dazu, eine abstrakte Schnittstelle zu definieren, die eine komplexere Implementierung eines Subsystems verbirgt. Dadurch lässt sich die Komplexität eines Programms verringern und Abhängigkeiten reduzieren.

In unserem Beispiel wenden wir das Pattern in der Methode `loadXMLDoc()` an. Im Grunde möchten der Anwendung nur mitteilen: „Lade mir den Inhalt der Text `data.txt` über einen XMLHttpRequest. Die Callback-Methode lautet `toConsole()`“. Was sich jedoch hinter dieser Schnittstelle verbirgt, bleibt uns verborgen. Aber wir möchten uns dafür auch gar nicht interessieren.

### 5.3.2 Adapter-Pattern

Über das Adapter-Pattern passen wir eine Schnittstelle, die nicht kompatibel zu einer erwarteten Schnittstelle ist, über einen Adapter an.

Dieses Pattern wird in obigem Beispiel nicht angewendet. Doch spielen wir eine kleine Erweiterung des Beispiels gedanklich durch: Beim Erzeugen eines XMLHttpRequest-Objekts möchten wir keine browserspezifische Implementierung beachten. Doch die Erzeugung des Objekts muss natürlich browserspezifisch durchgeführt werden.

Sinnvoll wäre es also, die Erzeugung des Request-Objekts in einer Methode `initXMLHttpRequest()` zu kapseln, die wir in `loadXMLDoc()` aufrufen. Die beiden Schnittstellen wären so adaptiert.

### 5.3.3 Observer-Pattern

Beim Observer-Pattern geht es darum, eine 1:n-Beziehung zwischen Objekten zu definieren. Wird ein Objekt geändert, sollen alle abhängigen Objekte automatisch benachrichtigt und aktualisiert werden.

Bei einer AJAX-Anwendung kommt dieses Pattern z.B. zum Einsatz, wenn mehrere Callback-Methoden implementiert und verwendet werden sollen.

Einfaches Beispiel: Beim Start unserer Anwendung sollen zwei Knoten des DOM in Variablen referenziert werden, um Kindknoten einhängen zu können.

Nach dem Facade-Pattern könnte dies z.B. so aussehen:

34 Literaturquelle [PATTERNS]

```
window.onload=function(){
  getKnoten1();
  getKnoten2();
}
function getKnoten1(){
  knoten1 = document.getElementById('kn1');
}
function getKnoten2(){
  knoten2 = document.getElementById('kn2');
}
```

Besser ist es jedoch, einen Array von Callback-Methoden anzulegen und die Callback-Methoden einzufügen (über eine selbst implementierte Methode `addOnLoadListener()`)...

```
window.onloadListeners=new Array();
window.addOnLoadListener(listener){
  window.onloadListeners[window.onloadListeners.length]=listener;
}
```

... um dann in der `window.onload`-Methode über den Array zu iterieren:

```
window.onload=function(){
  for(var i=0;i<window.onloadListeners.length;i++){
        var func=window.onLoadListeners[i];
        func.call();
  }
}
```

Wollten wir diese OnLoad-Listener nicht mehr beim Start initialisieren, sondern erst später z.B. durch einen expliziten Methodenaufruf, wäre dies bei der Implementierung als Array kein Problem. Es folgt also eine automatische „Benachrichtigung“ und „Aktualisierung“.

### 5.3.4 Model-View-Controller

Das Model-View-Controller-Pattern, das eine strikte Trennung zwischen der Datenstruktur (Model), der Präsentation (View) und der Anwendungslogik (Controller) vorschreibt, ist eigentlich ein Spezialfall des Observer-Patterns. MVC wird in Kapitel III ausführlich behandelt.

#### 5.3.4.1 Praktische Anwendung des MCV-Patterns

**Kein View in der Anwendungslogik**

Auch von der anderen Seite muss eine deutliche Trennung zwischen View und Anwendungslogik erfolgen.

Ein Verletzung in umgekehrter Richtung könnte z.B. durch den falschen bzw. fehlenden Einsatz von CSS auftreten. Stellen wir uns vor, wir möchten einen Textknoten erstellen und in den DOM einhängen. Der dafür vorgesehene Knoten sei in einer globalen Variable `knotenFehlermeldung` referenziert.

Da es sich bei diesem Text um eine Fehlermeldung handelt, möchten wir diese rot hervorheben. Eine einfache Methodik hierfür wäre z.B.

```
ausgabeFehlermeldung=function() {
```

```
  if (knotenFehlermeldung != null){
              var meldung = document.createElement('p');
              meldung.setAttribute('style','color:red;');
              knotenFehlermeldung.appendChild(meldung);
              var text = document.createTextNode('Es ist ein Fehler
        aufgetreten.');
              meldung.appendChild(text);
  }
}
```

Wir weisen also unserem Fehlerparagraphen die CSS-Eigenschaft `color` zu und setzen diese auf rot. Die direkte Definition im HTML verletzt die Trennung zwischen Model und View. In obigem Beispiel liegt gleichzeitig eine Verletzung der Trennung von Controller/View, da wir eine Style-Information in der Anwendungslogik integriert haben.

Es gilt also, stets die Verantwortlichkeiten zu beachten und strikt zu trennen. Daraus ergibt sich folgende Lösung:

```
ausgabeFehlermeldung=function() {
  if (knotenFehlermeldung != null){
              var meldung = document.createElement('p');
              meldung.setAttribute('class','fehlermeldung');
              knotenFehlermeldung.appendChild(meldung);
              var text = document.createTextNode('Es ist ein Fehler
        aufgetreten.');
              meldung.appendChild(text);
  }
}
```

Die zugehörige CSS-Definition lautet:

```
.fehlermeldung {
  color:red;
}
```

Wir haben somit eine Trennung der Verantwortlichkeiten erreicht.

**Keine Anwendungslogik im View**

Ein Ziel des MVC-Pattern ist es, Anwendungslogik nicht mit in den View einzubauen.

Ein Beispiel hierfür: In unserer Anwendung möchten wir einen Newstext vom Server laden. Wir haben uns eine Methode namens `holeDatei(dateiname, zielknoten)` geschrieben, die nach Übergabe der Datei und des Zielknotens den Textknoten in das DOM einhängt.

Eine Implementierung wie

```
<a href="javascript:holeDatei('news.txt','div_news');">News laden</a>
```

würde eine Verletzung des MVC-Patterns bedeuten, da Anwendungslogik im View integriert ist.

Korrekterweise müssten wir eine Methode `holeNews()` schreiben, die die dahinterliegende Logik kapselt. Unser Link

```
<a href="javascript:holeNews();">News laden</a>
```

ruft also folgende Methode auf:

```
holeNews=function() {
```

```
  holeDatei('news.txt','div_news');
}
```

### 5.3.5 Die Behaviour-Library

Dem aufmerksamen Leser ist vielleicht schon aufgefallen, dass auch obiges Beispiel keine absolute Trennung von Model und View gewährleistet. Zwar haben wir die Funktion `holeDatei('news.txt','div_news')` in einer Methode `holeNews()` gekapselt; trotzdem rufen wir diese Funktion im View auf. Model und View sind vermengt.

#### 5.3.5.1 Entwurf des W3C

Auch das W3C ist sich dieses Problems bewusst und arbeitet seit 1999 daran, das Verhalten von DOM-Elementen aus dem HTML-Dokument auszulagern. Geplant war eine Verlagerung in CSS nach etwa folgendem Muster:

```
a#holeNews {
  font-weight: 900;
  onClick: „holeNews()“
}
```

Leider ist es bis heute nicht gelungen, einen Standard zu verabschieden; das Dokument „Behavioral Extensions to CSS“[35] hat noch heute den Status „Arbeitsentwurf“.

#### 5.3.5.2 Behaviour Library (Ben Nolan)

Eine sinnvolle und durchdachte Lösung kommt von dem Entwickler Ben Nolan, der die Behaviour-Library[36] geschrieben hat.

Er preist sie vor allem für AJAX-Implementierungen an. Der Grund dafür ist einfach: Bei einer AJAX-Anwendungen definieren wir viele Elemente mit dem Attribut `onclick`, `onChange` etc., die JavaScript-Aktionen auslösen. Diese Attribute sollen ausgelagert und an zentraler Stelle definiert werden.

Zum Verwenden der Library muss die `behaviour.js` von Nolans Website geladen werden und in das HTML-Dokument eingebunden werden.

Betrachten wir also folgenden Code:

```
<a id="alert" href="" onClick=“alert('Meldung');“>Meldung zeigen</a>
```

Mit der Behaviour-Library verwenden wir diesen HTML-Code:

```
<a id="alert" href="">Alert zeigen</a>
```

und implementieren den folgenden JavaScript-Code in einer eigenen Datei, die wir zusätzlich zur behavious.js einbinden:

```
var myrules = {
```

35 http://www.w3.org/TR/1999/WD-becss-19990804
36 http://www.bennolan.com/behaviour/

```
  'a#alert' : function(element){
        element.onclick = function(){
              alert("Meldung");
        }
  },
  [ weitere Handler... ]
};

Behaviour.register(myrules);
```

Wir erhalten dieselbe Funktionalität.

Zur Erklärung: Wir definieren uns also ein assoziatives Array, dessen Index-Werte die CSS-Attribute bilden. Hier können beliebig Elementtypen, Klassen oder IDs verwendet werden. Nach obigem Schema können auch Handler für `onChange` oder weitere Ereignisse definiert werden.

Nach Definition aller Handler werden diese mit Hilfe von `Behaviour.register(alleEventHandler);` registriert.

Der obige Code wird sinnvollerweise in eine Datei wie z.B. `registerHandlers.js` ausgelagert. Entscheidend ist, die Behaviour-Library vor der Handler-Datei einzubinden, da sonst die entsprechende Behaviour-Klasse nicht zur Verfügung steht.

Wichtig ist außerdem, bereits zu Beginn alle Handler zu definieren, auch wenn die entsprechenden DOM-Elemente noch nicht im DOM-Tree vorhanden sind. Um bisherige Regeln auf neue Elemente anzuwenden, muss im Verlauf der Anwendungslogik die Methode `Behaviour.apply();` aufgerufen werden.

Hintergrundinformation für Interessierte: Behaviour verwendet intern die Methode `document.getElementsBySelector()`, die DOM-Elemente mit der entsprechenden CSS-Deklaration als Array zurückgibt. Diese wurde 2003 von Simon Willison[37] entwickelt.

Wir werden die Behaviour-Library in unserem Beispiel nicht verwenden. Sie empfiehlt sich allerdings zum Einsatz in Projekten ab mittlerer Größe.

### 5.3.6 Zusammenfassung

Dies soll als Auswahl von Patterns und deren praktische Anwendung genügen.

Wichtig ist mir noch der Hinweis, dass Patterns zwar grundsätzlich richtig und wichtig sind, dass es in der Praxis aber immer zu einer gewissen Vermischung bzw. zu Grauzonen kommt. Eine Implementierung kann nicht immer klar einem Pattern zugewiesen werden; genausowenig kann ein Pattern immer über eine Implementierung gestülpt werden.

Trotzdem müssen wir einige grundsätzliche Patterns im Hinterkopf behalten, um strukturiert vorgehen zu können.

# 6 Die Server-Seite

Auf der Client-Seite können wir nun unseren Code gut strukturieren. Allerdings haben wir einen

37 http://simon.incutio.com/archive/2003/03/25/getElementsBySelector

relativ wichtigen Aspekt der Anwendung bisher vernachlässigt: die Server-Seite.

In bisherigen Beispielen haben wir als Proof-Of-Concept eine einfache Textdatei vom Server geladen. Doch wir brauchen auch erweiterte Funktionalitäten wie z.B. die Möglichkeit zu Datenbankanfragen. Wie können wir die Server-Seite implementieren?

## 6.1 Grundsätzliches zu Server-Seite

Der Server hat in einer AJAX-Anwendung zwei grundsätzliche Aufgaben:

1. Er muss die Anwendung zum Browser transportieren. Bisher gingen wir davon aus, dass es sich dabei um eine statische Anwendung handelt, also eine Serie von HTML-, CSS- und JavaScript-Dateien. Dies muss jedoch nicht immer so sein. Hier können auch dynamische Werkzeuge wie z.B. PHP zum Einsatz kommen.
2. Er muss mit dem Client kommunizieren. Fragt der Client Daten an, muss der Server den Request handeln können und die Daten zurück liefern bzw. die übermittelten Daten schreiben. Zur Kommunikation wird HTTP verwendet, das dem Request-Response-Paradigma folgt. Folglich muss der Client die Kommunikation initiieren.

## 6.2 Klassische Server-Implementierungen...

In einer klassischen Web-Anwendung ist die Server-Seite schwergewichtig. Während der Client nur ein Terminal repräsentiert, muss der Server die Anwendungslogik bereitstellen und mit dem Datenmodell einen zugehörigen View bauen. Außerdem übernimmt er vollständig das Session-Management und verwaltet zu jeder Benutzersitzung einen Zustand.

Die Server-Seite einer klassischen Anwendung ist in einer spezifischen Sprache geschrieben. Die bekanntesten Beispiele hierfür sind PHP, Java, ASP, ASP.NET und Ruby.

Eine klassische Server-Architektur besitzt zwei Schichten: Eine Business-Schicht und eine Präsentationsschicht. Die Business-Schicht verwaltet die Anwendungslogik, macht Datenbankzugriffe und Ähnliches. Die Präsentationsschicht erzeugt auf der Business-Schicht aufbauend eine Präsentation der Anwendung. Diese wird dem Client, also dem Browser, übergeben.

## 6.3 ...und was sich mit AJAX ändert

Bei einer AJAX-Anwendung ist neu, dass ein Teil der Anwendungslogik zum Client übertragen wird.

AJAX führt zur Einführung einer dritten Schicht im Client, die dem Server einige Verantwortlichkeiten abnimmt. Die Vorteile dieser neuen Schicht haben wir bereits kennen gelernt.

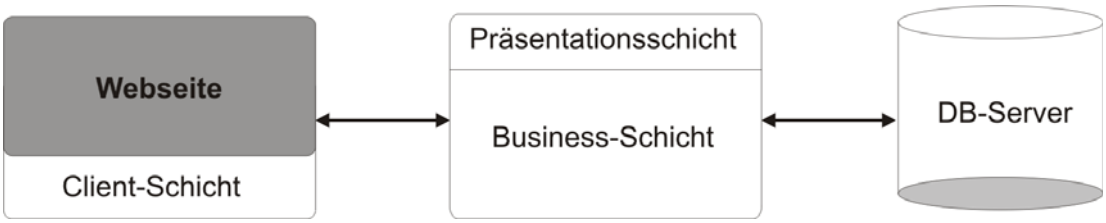


*Abbildung 16: Schichten einer AJAX-Anwendung*

## 6.4 Web-Frameworks

Es gibt jedoch nicht nur naive Implementierungen auf der Server-Seite. Häufig kommen Web-Frameworks zum Einsatz. Bereits für klassische Web-Anwendungen gibt es eine Vielzahl von davon; teilweise können sie auch für AJAX-Anwendungen verwendet werden.

Die Anzahl von Frameworks ist unüberblickbar. Einen Einblick in Frameworks gibt das Kapitel Ruby on Rails. Deshalb soll an dieser Stelle auf weitere Ausführungen verzichtet werden.

## 6.5 Datenübertragung

Wie bereits erwähnt, können Daten in einer AJAX-Anwendung in XML-Form oder als Plain-Text übertragen werden. Die entsprechenden Antworten des Servers können mit Hilfe der XMLHttpRequest-Eigenschaften `responseText` und `responseXML` ausgelesen werden.

XML-Nachrichten können server-seitig z.B. von einem Java-Web-Service generiert werden. So liegen z.B. SOAP-Nachrichten im XML-Format vor.

In unseren Beispielen werde ich sowohl auf Übertragungen im Plain-Text- als auch im XML-Format eingehen.

## 6.6 Zusammenfassung/Ausblick

Dies soll als Überblick genügen.

In unseren Beispielen werden wir die Server-Seite klassisch implementieren. Ich habe mich für PHP entschieden. Dies hat mehrere Vorteile: Es kann günstig gehostet werden und lässt sich lokal mit wenig Konfigurationsaufwand testen. Verwenden wir uns ein Komplettpaket wie XAMPP, das bereits erwähnt wurde, steht uns direkt nach der Installation ein lauffähiger Server zur Verfügung, der PHP interpretiert und eine MySQL-Datenbank mitbringt. Dieses werden wir in unseren Beispielen verwenden.

Und der wohl größte Vorteil von PHP: Es ist für Einsteiger leicht verständlich und hat eine niedrige Einstiegshürde. In diesem Paper möchten wir uns bei der AJAX-Entwicklung auf die Client-Seite konzentrieren und deshalb die Server-Seite zügig implementieren und dann als Blackbox hinnehmen.

Auf die genaue Implementierung der Server-Seite in PHP werde ich im Beispiel eingehen.

# 7 Security in AJAX

Beim Entwickeln einer AJAX-Anwendung steht auch die Frage der Sicherheit im Raum. Welche Lücken bestehen und wie können diese abgesichert werden?

Grundsätzlich haben wir folgende Security-Fragen zu klären:

- Wie sicher läuft unser Code im Client?
- Wie können wir Benutzerdaten sichern?
- Wie können wir die Kommunikation zwischen Client und Server absichern?

Viele Sicherheitsfragen, die wir im Zusammenhang mit einer AJAX-Anwendung diskutieren, betreffen auch klassische Web-Anwendungen. Ich möchte hier nur auf einige grundsätzliche Dinge eingehen.

## 7.1 JavaScript und Browser-Sicherheit

Wenn wir eine Anwendung in unserem Client, also dem Browser, laufen lassen, bringen wir dem Autor und Herausgeber der Anwendung Vertrauen entgegen. Auf unserer Maschine läuft ausführbarer Code, der automatisch über das Netzwerk übertragen wurde und mit seinem Ursprung kommunizieren kann. Dies birgt natürlich ein gewisses Sicherheitsrisiko.

Aus diesem Grund läuft der JavaScript-Code im Browser in einer Sandbox. Er hat somit nur eingeschränkten Zugriff auf unser System und kann z.B. nicht im lokalen Dateisystem lesen oder schreiben.

Außerdem kann die Anwendung keine Verbindung zu anderen Netzwerken herstellen. Auch können JavaScripts, die von zwei unterschiedlichen Servern geladen wurden, nicht miteinander kommunizieren. Diese beiden Fälle fasst man unter dem Stichwort „Server Of Origin“ zusammen.

Zum Nachteil des Entwicklers geht jedoch die Einschränkung des Ursprungsservers so weit, dass selbst Subdomains, die auf die gleiche IP-Adresse zeigen, als unterschiedliche Domains gehandhabt werden.

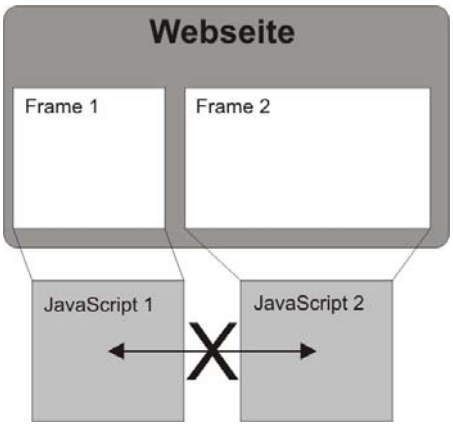


*Abbildung 17: Server Of Origin*

Trotz möglicher Nachteile auf Entwicklerseite arbeitet das Sicherheitskonzept zum Vorteil der Sicherheit und ist als ausgereift zu bezeichnen. Implementierungsfehler in Browsern können jedoch nicht ausgeschlossen werden.

## 7.2 Sicherheit von Benutzerdaten

Bei einer Web-Anwendung müssen wir uns immer auch Gedanken über die Sicherheit von Benutzerdaten machen. Dies liegt daran, dass keine direkte Verbindung zwischen Client und Server besteht, sondern die Kommunikation über Zwischenstationen verläuft.

Somit besteht die Möglichkeit einer Man-In-The-Middle-Attacke. Sämtlicher Datenverkehr läuft unverschlüsselt über HTTP. Gelingt es einem Angreifer, an die HTTP-Pakete heranzukommen (z.B. über Manipulation des Routings), stehen ihm sämtliche Daten der einzelnen Requests und Responses zur Verfügung.

Eine Möglichkeit, unsere Kommunikationsdaten abzusichern, ist es, eine Ende-Ende-Verschlüsselung (zum Beispiel über ein Private/Public-Key-Verfahren) einzusetzen. In JavaScript stehen uns Bibliotheken zur Verfügung, die verschiedene Verschlüsselungsalgorithmen implementieren. Weitere Informationen hierzu finden sich am Ende des Papers.

## 7.3 Gesicherte Kommunikation zwischen Server und Client

Eine andere Möglichkeit, die Kommunikation zwischen Client und Server abzusichern, ist es, eine gesicherte HTTP-Verbindung aufzubauen. Beim sogenannten Hyper Text Transfer Protocol over Secure Socket Layer kommt dabei ein Private/Public-Key-Mechanismus zum Einsatz.

Dies schließt natürlich eine Man-In-The-Middle-Attacke nicht aus. Die Pakete sind allerdings in verschlüsselter Form nutzlos.

Um HTTPS verwenden zu können, muss dies vom Server und vom Client unterstützt werden. Auf Client-Seite macht dies keine Probleme; alle Browser bringen dafür eine Unterstützung mit. Auf der Server-Seite muss Verschlüsselung vom Hoster unterstützt werden. Die monatlichen Kosten dafür halten sich jedoch in Grenzen.

Ich möchte noch kurz auf die Nachteile einer gesicherten HTTP-Verbindung eingehen:

1. HTTPS erzeugt Kommunikations-Overhead. Während wir in einer klassischen Anwendung nur Schlüsselelemente über eine gesicherte Verbindung übertragen, Bilder und Ähnliches über eine ungesicherte Verbindung, ist dies bei einer AJAX-Anwendung nicht möglich. Gesicherte Server (https://) und ungesicherte Server (http://) werden als unterschiedliche Domains erkannt. Interkommunikation zwischen JavaScripts beider Domains würde das Sicherheitsmodell von JavaScript verletzen.
2. HTTPS sichert nur die Übertragung der Daten. Wie sicher die Daten z.B. auf dem Server liegen, liegt nicht im Verantwortungsbereich von HTTPS.

Trotzdem ist HTTPS ein guter Ansatz, um die Kommunikation unserer AJAX-Anwendung abzusichern.

## 7.4 Sicherheit des Quellcodes

Beim Entwickeln unserer Anwendung muss uns bewusst sein, dass es sich bei JavaScript um eine interpretierte, nicht kompilierte Skriptsprache handelt. Das heißt konkret, dass uns keine Möglichkeit gegeben ist, unseren Quellcode vor Einblicken zu schützen. Jede AJAX-Anwendung ist praktisch open-source; Codediebstahl kann nicht verhindert werden.

# 8 AJAX-Performance

Kurz anschneiden möchte ich auch das Thema Performance. Wenn wir dem Server Verantwortlichkeiten in der Anwendungslogik abnehmen, bürden wir uns unter anderem die Verantwortung auf, dass unsere Anwendung performant auf dem Client läuft.

Was versteht man eigentlich unter Performance? Ich möchte zwei Performance-Aspekte betrachten:

1. Wie schnell läuft unsere Anwendung?
2. Wie viele Ressourcen belegt sie?

## 8.1 Geschwindigkeit

Geschwindigkeit bezieht sich in unserem Fall auf die Ausführungsgeschwindigkeit von JavaScript. JavaScript ist nicht gerade als schnelle Sprache bekannt. Bei einer klassischen Implementierung hat dies nur bedingt Konsequenzen, da sich der JavaScript-Code-Anteil in Grenzen hält. Bei einer AJAX-Anwendung können sich Performance-Untersuchungen aber durchaus auszahlen.

Bei der Ausführungsgeschwindigkeit gilt natürlich die Regel: Umso schneller, umso besser. Wie messen wir Geschwindigkeit und wie können wir an der Geschwindigkeitsschraube drehen?

### 8.1.1 Testen und Optimieren der Ausführungsgeschwindigkeit

Ein Ansatz, um die Ausführungsgeschwindigkeit zu messen, ist es, eine Art Stop-Uhr einzubauen. Zu Beginn und zu Ende einer Anfrage nehmen wir die Systemzeit und berechnen daraus die Differenz, die wir danach ausgeben.

Eine komfortablere Möglichkeit bieten grafische Werkzeuge wie z.B. der Venkman Profiler[38], der in Mozilla-Browser eingebettet werden kann. Dieser kann auch als JavaScript-Debugger verwendet werden. Leider ist die momentan aktuelle Version noch nicht unter Mozilla Firefox 1.5 lauffähig.

Im Folgenden möchte ich einige grundsätzliche Optimierungsmöglichkeiten vorstellen, die ich dem Buch „Ajax in Action“ entnommen habe.

### 8.1.2 Schleifen optimieren

Eine kleine, aber wirkungsvolle Möglichkeit, Schleifen zu optimieren, ist die Vermeidung von

38 Download unter http://www.hacksrus.com/~ginda/venkman/

Funktionen im Kopf der Schleife. Nehmen wir an, wir haben eine Funktion `maxAnzahl()`, die uns die Anzahl der Schleifendurchläufe berechnet. Diese Funktion sei aus einem beliebigen Grund rechenintensiv, gebe aber stets das gleiche Ergebnis zurück.

Verwenden wir also folgenden Code, um einen Counter hochzuzählen:

```
var counter=0;
for (var i=0; i<maxAnzahl(); i++){
  counter += i;
}
```

ist uns meist nicht bewusst, dass bei jeder Iteration der Schleife die Funktion `maxAnzahl()` neu ausgeführt werden muss. Dies kann zu einem Einbruch der Performance führen.

Unser optimierter Code würde also so aussehen:

```
var counter=0;
var anzahl=maxAnzahl();
for (var i = 0; i<anzahl; i++ {
  counter += 1;
}
```

### 8.1.3 Umgang mit dem DOM

Um den View unserer Anwendung zu verändern, bearbeiten wir den DOM-Tree, hängen neue Knoten ein, bearbeiten diese und löschen sie wieder. Ein Ziel der Performance-Optimierung ist es also, diese Vorgänge möglichst effizient zu gestalten.

Beim Hinzufügen, Bearbeiten und Löschen von Knoten müssen wir uns bewusst sein, dass eine Änderung jeweils ein teures, da rechenintensives Re-Rendering nach sich zieht. Es gilt also, die Häufigkeit von Re-Renderings zu optimieren.

Dies können wir dadurch, indem wir alle benötigten Knoten erzeugen und zuerst aneinander hängen (falls wir Kindknoten von neuen Knoten erzeugt haben). Erst im letzten Schritt sollten wir den kompletten Teilbaum in unseren Gesamt-DOM einhängen. Dies optimiert die Anzahl der Re-Renderings, was auch vom Benutzer wahrgenommen wird („Flackern“ des Browserinhalts).

### 8.1.4 Die Punkt-Notation

Bei JavaScript handelt es sich um eine objektorientierte Sprache. Objekte können beliebig ineinander verschachtelt werden. Dabei kann schnell zu langen Punkt-Notationen kommen.

Ein Beispiel: Wir möchten uns die Farbe und Größe einer Schiffskapitänsmütze in Variablen speichern. Dies lässt sich z.B. mit Hilfe von folgendem Code realisieren:

```
var farbe = schiff.kapitaen.muetze.farbe;
var groesse = schiff.kapitaen.muetze.groesse;
```

Doch Punkt-Notationen sind teuer. Es empfiehlt sich daher folgende Optimierung:

```
var muetze = schiff.kapitaen.muetze;
var farbe = muetze.farbe;
var groesse = muetze.groesse;
```

## 8.2 Ressourcen

Bei einer AJAX-Anwendung müssen wir verantwortungsbewusster mit Ressourcen umgehen als in einer klassischen Web-Anwendung.

Bei dieser stellt, wie bereits geschrieben, der Client nur ein Terminal dar; die Anwendungslogik verbleibt auf dem Server. Performance-Probleme treten meist nur auf der Server-Seite auf und können von uns als Entwicklern gehandhabt und gelöst werden.

Bei AJAX ändert sich dies. Der Client wird richer, also müssen ihm auch mehr Ressourcen zur Verfügung stehen. Wir als Entwickler haben die Client-Seite nur bedingt in der Hand. Wir können zwar die Ausführungsgeschwindigkeit und den Ressourcen-Hunger unserer Anwendung optimieren, haben jedoch keinerlei Einfluss auf die Umgebung, in der die Anwendung laufen wird.

Trotzdem gilt es, einige Optimierungsmöglichkeiten auszunutzen. Auch diese habe ich wieder „Ajax in Action“ entnommen.

### 8.2.1 Memory Leaks

Beim Entwickeln unserer Anwendung müssen wir darauf achten, eine speichereffiziente Anwendung zu schreiben. Es darf nicht mehr Speicher belegt werden, als nötig.

Dies tritt immer dann auf, wenn reservierter Speicher nach Benutzung nicht mehr freigegeben wird. Was in einer klassischen Web-Anwendung kein Thema ist, kann in einer AJAX-Anwendung, die unter Umständen stundenlang laufen soll, zum Problem werden.

In JavaScript wird Speicher genauso wie in Java gehandhabt. Im Hintergrund läuft ein Garbage-Collector, der alle Speicherbereiche, auf die keine Referenzen mehr bestehen, wieder freigibt. Dieser Automatismus entlässt uns jedoch nicht unserer Verantwortung, uns mit dem Speichermanagement zu befassen und Referenzen auf nicht mehr benötigte Speicherbereiche zu entfernen.

Dies können wir dadurch erreichen, indem wir Variablen auf `null` setzen, egal ob dies primitive Variablen oder Objekte sind. Es empfiehlt sich, Destruktor-Methoden zu entwickeln, die z.B. auch kaskadierendes Löschen von Referenzen implementieren können.

### 8.2.2 Zyklische Referenzen

Eine AJAX-Awendung ist üblicherweise in einen Model, einen View und einen Controller unterteilt. Das Problem von zyklischen Referenzen bezieht sich in einer AJAX-Anwendung auf Beziehungen zwischen Model und View.

Wie gerade behandelt, wird in JavaScript per Garbage-Collection nicht mehr benötigter Speicher freigegeben. Nicht mehr benötigt wird definiert über nicht mehr referenziert.

Es ist in unserer Anwendung deshalb unbedingt zu vermeiden, dass sich zwei Objekte gegenseitig referenzieren. Dies kann z.B. zwischen DOM-Knoten und Objekten auftreten. Wird z.B. ein Knoten aus dem DOM-Tree entfernt, ist darauf zu achten, dass auch das zugehörige JavaScript-Objekt gelöscht wird.

### 8.2.3 Der Umgang mit dem DOM

Wie bereits erwähnt, sind DOM-Operationen teuer (bezogen auf Rechenaufwand und Speicherbelastung).

Der DOM-Tree liegt vollständig im Speicher; dies können wir nicht ändern. Jedoch können wir versuchen, ihn möglichst klein zu halten. Ein konsequenter Einsatz von CSS kann Wunder wirken, da viele HTML-Formatierungselemente wegfallen, die eigene Knoten im DOM-Tree darstellen.

Außerdem empfiehlt es sich, DOM-Knoten zu recyclen. Das Löschen und Hinzufügen einen neuen, leicht veränderten Knotens ist vielfach teurer als das Bearbeiten eines bereits vorhandenen Knotens.

# 9 Ajax in der Praxis: Beispiel-Anwendungen

Nachdem wir nun mit den Technologien relativ sicher umgehen können, uns gewisser Design Patterns bewusst sind, die server-seitige Programmierung kennen gelernt haben und uns Security- und Performance-Fragen bewusst sind, möchten wir das Ganze praktisch anwenden.

Zu Beginn soll eine Art Google Suggest implementiert werden. Es handelt sich dabei um ein typisches AJAX-Widget, also eine Mini-Anwendung, die in eine klassische Web-Anwendung integriert werden kann.

Da sich im ersten Beispiel die Übertragung auf das Plain-Text-Format beschränkt, werde ich danach die Übertragung im XML-Format vorstellen und im Beispiel eines Adressbuchs implementieren.

## 9.1 Beispiel 1: „Suggest“

Im Folgenden möchten wir „Google Suggest“ implementieren.

Es handelt sich dabei um eine Vervollständigen-Funktion eines Suchfeldes. Bei jeder Eingabe werden Wortvorschläge angezeigt. Diese können per Mausklick in das Textfeld übernommen werden, woraufhin eine Suchabfrage gestartet werden kann.

Das Beispiel ist dem Buch „Ajax in Action“ in leicht abgewandelter Form entnommen. Die Anwendung ist in eine klassische Implementierung eingebettet; es handelt sich also um ein typisches AJAX-Widget.

Das Ergebnis unserer Anwendung wird so aussehen:

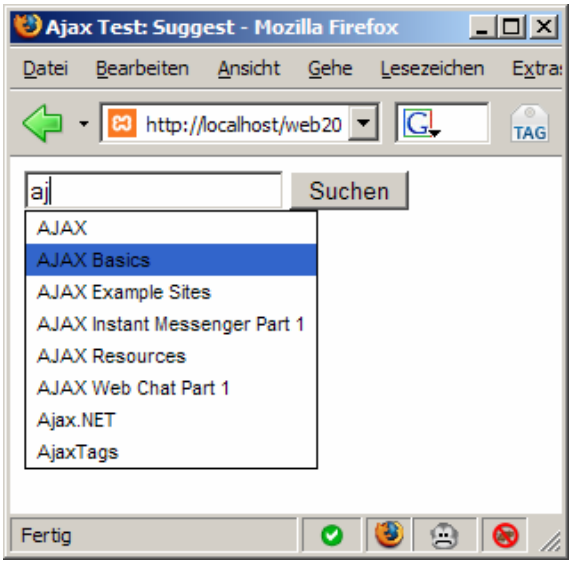


*Abbildung 18: Auto-Vervollständigen nach dem Beispiel von Google Suggest*

In obiger Abbildung werden uns Wortvorschläge für „aj“ angezeigt. Bei Klick auf einen Eintrag wird das Wort in das Suchfeld übernommen. Die Suchanfrage soll mit dem Suchen-Button an Google weitergereicht werden.

### 9.1.1 Vorbereitungen

Um das Beispiel nachvollziehen zu können, muss ein Apache-Server mit einer MySQL-Datenbank zur Verfügung stehen. Da die Wortvorschläge der Datenbank entnommen werden, müssen ein entsprechender Wortschatz zur Verfügung stehen. Es muss dazu eine Tabelle `suggest` mit entsprechenden Beispieleinträgen angelegt werden, die wir mit PHP ansprechen werden.

Die Struktur der Tabelle lautet wie folgt:

```
CREATE TABLE `suggest` (
  `suggest_id` int(11) NOT NULL auto_increment,
  `title` varchar(255) default NULL,
  PRIMARY KEY  (`suggest_id`)
)
```

Als Primary Key verwenden wir eine Integer-Zahl, unsere Wortvorschläge sind in `title` gespeichert.

Zu Beginn eine Übersicht über unsere Projektstruktur:

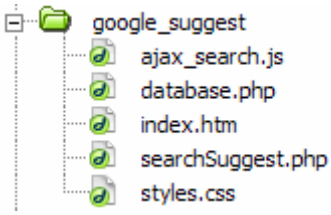


*Abbildung 19: Projekt-Struktur des Beispiels*

Der Einstiegspunkt in unsere Anwendung ist die Datei `index.htm`; hier ist das Suchformular implementiert. CSS-Stile sind in der Datei `styles.css` definiert, unser AJAX-Code befindet sich in `ajax_search.js`. Diese beiden Dateien binden wir in die Hauptseite `index.htm` ein.

Mithilfe von asynchronen Requests werden wir `searchSuggest.php` mit entsprechenden Parametern GET-Parametern ansprechen, das uns die Suchergebnisse zurückliefern wird. Der Aufbau der Datenbankverbindung erfolgt in `database.php`.

### 9.1.2 Einstiegspunkt (index.htm)

Der Einstiegsdokument `index.htm` ist wie folgt aufgebaut:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <title>Ajax Test: Suggest</title>
  <script language="JavaScript" type="text/javascript"
  src="ajax_search.js"></script>
  <link href="styles.css" rel="stylesheet" type="text/css">
</head>

<body>
<form id="formSearch" action="http://www.google.de/search" method="get">
  <input type="text" id="q" name="q" alt="Search Criteria"
        onkeyup="searchSuggest();" autocomplete="off" />
  <input type="submit" name="Suchen" value="Suchen"  />
  <div id="search_suggest"></div>
</form>
</body>
</html>
```

Im Head der HTML-Datei binden wir die CSS-Datei und den AJAX-Code ein.

Im Body definieren wir ein Formular mit einem Eingabefeld für das Suchwort und einem Abschicken-Button. Da wir mit dem Suchwort eine Google-Suche starten möchten, setzen wir die `action` auf die deutsche Google-Seite und `method=get`.

Eine einfache Google-Suchabfrage erfolgt über:

```
http://www.google.de/search?q=suchbegriff
```

Unser Eingabefeld für den Suchbegriff benennen wir deshalb mit `q`, da dieser als Name für den GET-Parameter verwendet wird. Über die Eigenschaft `onkeyup` weisen wir die JavaScript-Methode `searchSuggest()` zu. Es werden also nach jeder Eingabe eines Zeichens die Suchvorschläge aktualisiert. Zusätzlich deaktivieren wir mit `autocomplete=off` die browsereigene Vervollständigen-Funktion, die sich mit unseren Wortvorschlägen überlagern würde.

Danach folgt die Definition des Suchen-Buttons.

In den Div mit der `id=search_suggest` werden wir unsere Wortvorschläge schreiben.

### 9.1.3 CSS-Stile (styles.css)

Kurz ansprechen möchte ich die verwendeten CSS-Stile, die größtenteils selbsterklärend sind:

```
body {
  font: 11px arial;
  }

.suggest_link {
        background-color: #FFFFFF;
        padding: 2px 6px 2px 6px;
  }

.suggest_link_over {
        background-color: #3366CC;
        padding: 2px 6px 2px 6px;
  }

#search_suggest {
        position: absolute;
        background-color: #FFFFFF;
        text-align: left;
        border: 1px solid #000000;
        cursor:pointer;
  }
```

Erwähnen möchte ich die beiden CSS-Klassen `.suggest_link` und `.suggest_link_over`, die wir für die einzelnen Wortvorschläge definieren. Diese unterscheiden sich nur in der Hintergrundfarbe. Wir werden das Zuweisen der beiden Klassen bei Mouse-Over und Mouse-Out in eigenen JavaScript-Methoden implementieren.

Interessant ist die Eigenschaft `cursor:pointer` in `#search_suggest`. Damit legen wir fest, dass beim Auswählen der Suchvorschläge ein Hand-Cursor angezeigt wird.

### 9.1.4 AJAX-Code (search_suggest.js)

Wie bereits erwähnt, wird bei jedem `onkeyup` im Suchformular die Methode `searchSuggest()` aufgerufen. Diese befindet sich in unserem Anwendungscode in `search_suggest.js`.

Hier der zugehörige Code:

```
// Methode zur browserspezifischen Erzeugung eines XmlHttpRequest-Objekts
function getXmlHttpRequestObject() {
  if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
  } else if(window.ActiveXObject) {
        return new ActiveXObject("Microsoft.XMLHTTP");
  } else {
        alert("Bitte Browser updaten");
  }
}

// globales XMLHttpRequest-Objekt erzeugen
var searchReq = getXmlHttpRequestObject(); // (1)

// Wir bei onkeyup im Suchfeld aufgerufen. Startet den AJAX-Request
```

```
function searchSuggest() {
  if (searchReq.readyState == 4 || searchReq.readyState == 0) { // (2)
        var str = escape(document.getElementById('q').value); // (3)
        searchReq.open("GET", 'searchSuggest.php?search=' + str, true); // (4)
        searchReq.onreadystatechange = handleSearchSuggest; // (5)
        searchReq.send(null); // (6)
  }
}
```

Wir erzeugen uns ein browserspezifisches XMLHttpRequest-Objekt (1) und legen es in einer globalen Variable `searchReq` ab. Das ist in diesem Fall völlig ausreichend, da wir nur einen Request gleichzeitig zu behandeln haben werden.

In der Methode `searchSuggest()`, die bei jedem Eintippen eines Zeichens gerufen wird, prüfen wir zuerst, ob eine offene Verbindung besteht. Nur wenn keine Verbindung offen ist (`readyState=0`) oder das Laden erfolgreich abgeschlossen wurde (`readyState=4`), starten wir einen Request (2).

Wir holen uns zunächst den Suchbegriff aus dem Eingabefeld q und maskieren[39] eventuell enthaltene Sonderzeichen mit Hilfe von escape() (3).

Danach erzeugen wir einen neuen asynchronen Request und rufen `searchSuggest.php` mit GET-Parameter `search=str` auf (4). `SearchSuggest.php` wird uns Wortvorschläge zurückliefern.

Wir registrieren die Handler-Methode `handleSearchSuggest()` (5) und schicken als Daten `null` (da GET-Anfrage) (6).

### 9.1.5 Die Server-Seite (database.php und searchSuggest.php)

Kommen wir zur Server-Seite, die wir wie versprochen in PHP implementieren.

Wir benötigen ein Skript, um die Datenbankverbindung aufzubauen. Dieses implementieren wir in `database.php`:

```
<?php
  $db_verbindung = mysql_connect("server","benutzername","passwort");
  $db_db = mysql_select_db("datenbankname");
?>
```

Wir verbinden uns zum MySQL-Datenbank-Server mit den Parametern `server`, `benutzername` und `passwort` und selektieren wir die Datenbank. Alle Parameter müssen natürlich angepasst werden.

Die Verbindung muss bei jeder Anfrage aufgebaut werden. Mit dem Befehl

```
require("database.php");
```

binden wir deshalb die Datei in `searchSuggest.php` ein.

Der komplette Code von `searchSuggest.php` lautet:

```
<?php
require("database.php");
// Wurde GET-Parameter gesetzt?
```

39 Escaping: Maskieren von bestimmten Steuer- und Funktionszeichen durch Voranstellen eines Backslashes \. Dadurch werden diese nicht gesondert behandelt. Beispiele: ' + „ = ? &

```
if (isset($_GET['search']) && $_GET['search'] != '') { // (1)
  // Slashes hinzufügen, damit keine SQL-Fehler auftrefen
  $search = addslashes($_GET['search']); // (2)
  // Suchvorschläge abfragen
  $suggest_query = mysql_query("SELECT distinct(title) as suggest FROM suggest
  WHERE title like('" . $search . "%') ORDER BY title"); // (3)
  while($suggest = mysql_fetch_array($suggest_query)) {
        // Suchvorschläge zurückgeben, getrennt durch Zeilenumbrüche
        echo $suggest['suggest'] . "\n"; // (4)
  }
}
?>
```

Wir überprüfen, ob der GET-Parameter `search` gesetzt ist und nicht leer ist (1). Den Wert des Parameters speichern wir in der Variable `$search`, fügen aber zuvor Slashes hinzu (Maskieren zum Vermeiden von SQL-Fehlern, s. Fußnote 39) (2).

Danach starten wir die Datenbankabfrage (3). Unsere Wortvorschläge sind wie bereits erwähnt in `title` abgelegt; mit `distinct` erreichen wir, dass alle doppelten Wortvorschläge ausgeschlossen werden. Als Where-Clause verwenden wir `LIKE suchbegriff%`. Damit werden uns alle Wörter zurückgegeben, die mit dem Suchbegriff beginnen.

Die Wortvorschläge werden im Alias `suggest` abgelegt, auf den wir danach zugreifen.

In der While-Schleife geben wir alle Wörter getrennt durch Zeilenumbrüche zurück (4). Ein direkter

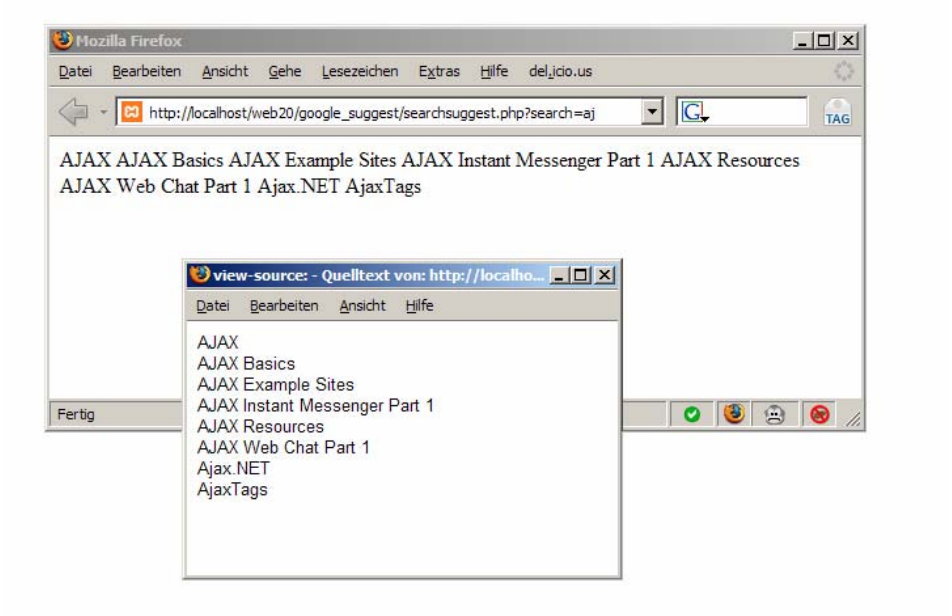


*Abbildung 20: Direktaufruf mit GET-Parameter search=aj*

Aufruf der `searchSuggest.php` bestätigt dies:

Wir erhalten also reine Textdaten zurück, die nicht in XML-Form vorliegen.

### 9.1.6 Verarbeitung der Rückgabewerte mit JavaScript (ajax_search.js)

Zur Verarbeitung dieser Daten erweitern wir unseren AJAX-Code um folgende Funktion:

```
// Handler für Rückgabe der Suchvorschläge
function handleSearchSuggest() {
  if (searchReq.readyState == 4) {
        var ss = document.getElementById('search_suggest') // (1)
        ss.innerHTML = ''; // (2)
        var str = searchReq.responseText.split("\n"); // (3)
        for(i=0; i < str.length - 1; i++) {
              // Element-String zusammenbauen
              var suggest = '<div onmouseover="javascript:suggestOver(this);"
  ';
              suggest += 'onmouseout="javascript:suggestOut(this);" ';
              suggest += 'onclick="javascript:setSearch(this.innerHTML);" ';
              suggest += 'class="suggest_link">' + str[i] + '</div>';
              ss.innerHTML += suggest;
        }
  }
}
```

In vorherigen Beispielen haben wir festgestellt, dass der Handler mehrmals aufgerufen wird. Die Ausgabe soll aber erst erfolgen, wenn der Ladevorgang erfolgreich abgeschlossen wurde, was wir in Zeile 3 prüfen.

Wir holen uns eine Referenz auf den Div, in den wir die Ergebnisse schreiben werden (1), und entfernen eventuell schon vorhandene Vorschläge vorangehender Suchvorgänge (2).

Da unsere Rückgabedaten in Textform vorliegen, haben wir mit `searchReq.responseText` Zugriff darauf. Den String teilen wir nach Zeilenumbrüchen auf und speichern ihn in einen Array (3), den wir daraufhin durchlaufen.

In jedem Schleifendurchlauf erweitern wir das `innerHTML` unseres Divs `search_suggest` um folgenden Code:

```
<div onmouseover="javascript:suggestOver(this);'
  onmouseout="javascript:suggestOut(this);“
  onclick="javascript:setSearch(this.innerHTML);"
  class="suggest_link">Suchvorschlag</div>
```

Jeder Suchvorschlag liegt also in einem eigenen Div mit der CSS-Klasse `suggest_link`.

Bei Mouse-Over soll dieser farblich hervorgehoben werden, bei Mouse-Out soll die Hervorherbung entfernt werden. Dazu weisen wir die beiden Methoden `suggestOver()` und `suggestOut()` zu und übergeben jeweils unseren Element-Knoten, den wir danach direkt ansprechen können.

Die beiden Methoden implementieren wir wie folgt:

```
// Mouse-Over-Funktion Suchvorschlag
function suggestOver(div_value) {
  div_value.className = 'suggest_link_over';
}
// Mouse-Out-Funktion Suchvorschlag
function suggestOut(div_value) {
  div_value.className = 'suggest_link';
}
```

Das Hervorheben erfolgt also durch Ändern der CSS-Klasse.

Zum Schluss benötigen wir eine Methode, die uns bei `onClick` den Vorschlag in das Eingabefeld übernimmt. Die Methode `suggest_link()` implementieren wir wie folgt:

```
function setSearch(value) {
  document.getElementById('q').value = value;
  document.getElementById('search_suggest').innerHTML = '';
}
```

Als Parameter wird der Suchbegriff selbst übergeben. Wir setzen den Wert des Eingabefeldes auf den Wortvorschlag und entfernen alle Wortvorschläge.

### 9.1.7 Suchanfrage bei Google

Nach Betätigen des Suchen-Buttons wird eine Google-Suchanfrage gestartet. Dazu wird die URL `http://www.google.de/search?q=suchbegriff` per `GET` angefragt. Zusätzlich wird als Parameter der Name unseres Buttons übertragen, der aber von Google ignoriert wird.

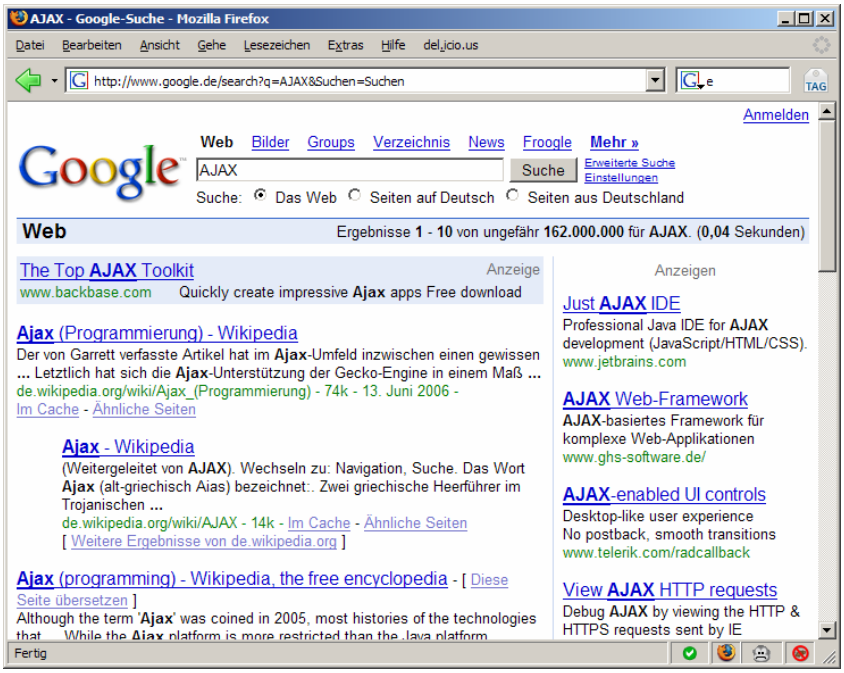


*Abbildung 21: Ergebnis unserer Google-Suche "ajax"*

### 9.1.8 Übertragungsformat der Daten

In obigem Beispiel übertragen wir die Rückgabedaten in reiner Textform, was bei solch einfach strukturierten Daten kein Problem ist.

Müssen wir jedoch mit komplexer strukturierte Daten umgehen, empfiehlt sich die Übertragung per

XML. Im Folgenden möchte ich darstellen, wie dies auf Server- und Client-Seite implementiert werden kann.

## 9.2 Kommunikation über XML

### 9.2.1 Einleitung

In vorigen Kapiteln haben wir festgestellt, dass wir mit dem XMLHttpRequest über Daten in XML-Form kommunizieren können. Diese können wir mit JavaScript lesen, ausgeben oder auch ändern.

Doch wie geht das? Und wie verarbeiten wir XML-Daten auf der Server-Seite?

### 9.2.2 Java Web-Services (Sun Microsystems)

Eine Möglichkeit besteht darin, einen Java Web-Service zu implementieren, der auf einem Java Application Server läuft. Java Web Services kommunizieren über SOAP-Nachrichten, die XML-Form besitzen. Links zur offiziellen Seite und zu einem Tutorial finden sich am Ende des Papers.

### 9.2.3 ASP.NET (Microsoft)

Auch mit Active Server Pages können XML-Daten erzeugt und übertragen werden. Auch hierzu finden sich am Ende des Papers Ressourcen.

### 9.2.4 PHP

Ich möchte mich auch hier wieder auf eine Implementierung in PHP konzentrieren. Wir werden dafür Funktionalitäten benutzen, die die aktuelle PHP-Version 5 voraussetzen. Ansonsten gibt es keine weiteren Voraussetzungen – wir können direkt durchstarten.

## 9.3 Beispiel 2: Adressbuch

Im Folgenden soll eine kleine Beispielanwendung implementiert werden, bei der Server und Client Daten in XML-Form austauschen. Es handelt sich um eine Art Adressbuch, das Einträge aus einer Datenbank als Visitenkarte anzeigt.

Unser Ergebnis soll so aussehen:

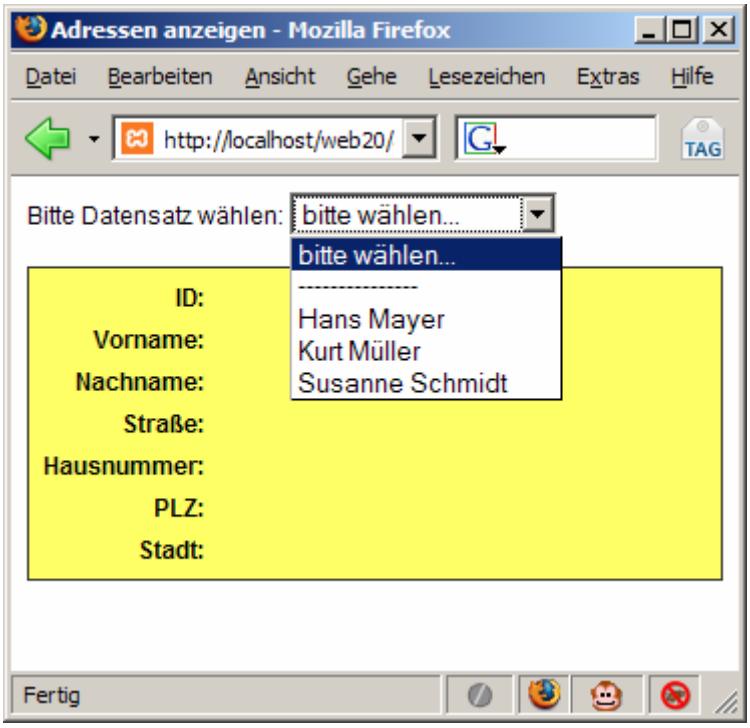


*Abbildung 22: Adresse auswählen*

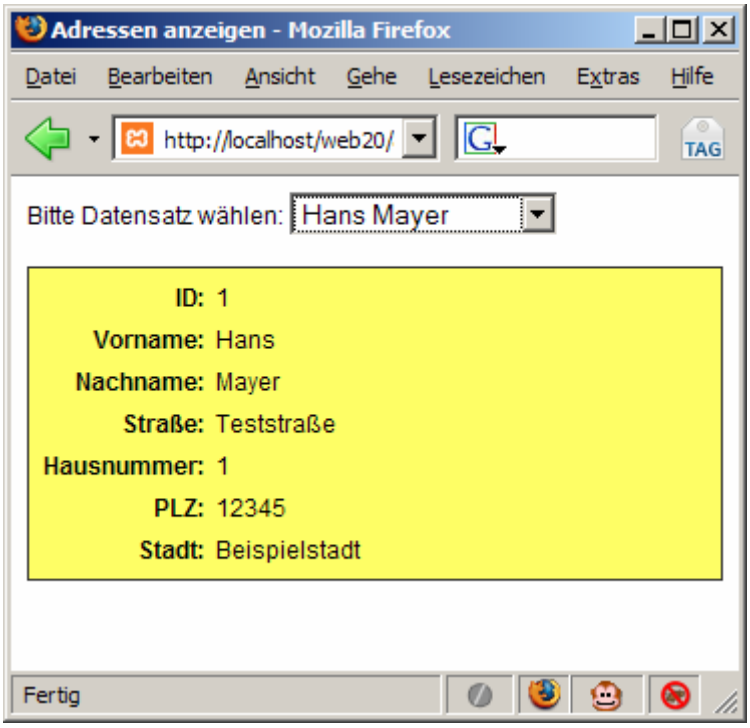


*Abbildung 23: Adresse anzeigen*

### 9.3.1 Vorbereitungen

Auch bei diesem Beispiel benötigen wir eine MySQL-Datenbank, in der die Adressdaten abgelegt sind. Die Tabellenstruktur der Adressdaten lautet wie folgt:

```
CREATE TABLE `adressen` (
  `id` int(11) NOT NULL auto_increment,
  `vorname` varchar(255) NOT NULL,
  `nachname` varchar(255) NOT NULL,
  `strasse` varchar(255) NOT NULL,
  `hausnummer` varchar(255) NOT NULL,
  `plz` varchar(255) NOT NULL,
  `stadt` varchar(255) NOT NULL,
  PRIMARY KEY  (`id`)
)
```

Als Primary Key verwenden wir eine Integer-Zahl; die anderen Spalten enthalten Adressdaten. Zum Testen müssen natürlich einige Beispieleinträge erzeugt werden.

Wir werden folgende Projektstruktur verwenden:

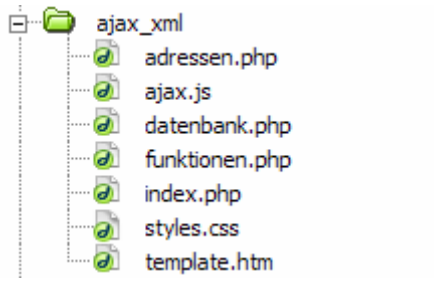


*Abbildung 24: Projektstruktur*

Unsere HTML-Seite lagern wir dieses Mal in ein Template (`template.htm`) aus, da wir die Einträge des Auswahlmenüs mit PHP dynamisch hinzufügen werden. Dies geschieht in `index.php`, die den Einstiegspunkt unserer Anwendung darstellt. Zum Einlesen des Templates implementieren wir in `funktionen.php` eine eigene Funktion.

CSS-Stile liegen in `styles.css`; unser AJAX-Code in `ajax.js`. Die Adressdaten werden von `adressen.php` angefordert, das dazu mit `datenbank.php` eine Datenbank-Verbindung aufbaut.

### 9.3.2 HTML-Template (template.htm)

Unser HTML-Template sieht wie folgt aus:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Adressen anzeigen</title>
```

```
<link href="styles.css" rel="stylesheet" type="text/css">
<script language="javascript" src="ajax.js"></script>
</head>

<body>
<div id="auswahl">
  <form name="form1" method="post" action="">
    Bitte Datensatz w&auml;hlen:
    <select id="selectAdresse" name="selectAdresse"
  onChange="javascript:adresseAnzeigen()">
        <option value="">bitte wählen...</option>
        <option value="">---------------</option>
        <!--adressen-->
    </select>
  </form>
</div>
<div id="anzeige">
    <table border="0">
      <tr>
        <th>ID:</th>
        <td><span id="id"></span></td>
      </tr>
      <tr>
        <th>Vorname:</th>
        <td><span id="vorname"></span></td>
      </tr>
      <tr>
        <th>Nachname:</th>
        <td><span id="nachname"></span></td>
      </tr>
      <tr>
        <th>Stra&szlig;e:</th>
        <td><span id="strasse"></span></td>
      </tr>
      <tr>
        <th>Hausnummer:</th>
        <td><span id="hausnummer"></span></td>
      </tr>
      <tr>
        <th>PLZ:</th>
        <td><span id="plz"></span></td>
      </tr>
      <tr>
        <th>Stadt:</th>
        <td><span id="stadt"></span></td>
      </tr>
  </table>
</div>
</body>
</html>
```

Im oberen Teil definieren wir ein Select-Feld, das bei `onChange` die Funktion `adresseAnzeigen()` aufruft. Wir brauchen also keinen gesonderten Button „Anzeigen“ o.ä..

Den Kommentar `<!--adressen-->` werden wir in `index.php` durch Optionen aus der Datenbank ersetzen. Jeder Eintrag wird als eigene Option angezeigt werden.

Im unteren Bereich definieren wir eine Tabelle mit Spans, die mit einer ID versehen sind. In diese

Spans werden wir die Adressdaten schreiben.

### 9.3.3 CSS-Stile (styles.css)

Die CSS-Stile möchte ich unkommentiert stehen lassen. Sie sollten selbsterklärend sein.

```
body {
  font-family:Arial, Helvetica, sans-serif;
  font-size:12px;
}

#auswahl {
  padding-bottom:15px;
}

#anzeige {
  background-color:#FFFF66;
  border:1px solid #333333;
  text-align:center;
  padding:3px;
}

table {
  border:0;
}

th {
  text-align:right;
  padding:2px;
}

td {
  padding:2px;
  text-align:left;
}
```

### 9.3.4 Einstiegspunkt (index.php)

Als Einstiegspunkt in unsere Anwendung verwenden wir `index.php`. In dieser Datei werden wir das Template einlesen und ergänzen. Es folgt der Code:

```
<?php
  // benötigte Dateien einbinden
  require("funktionen.php");
  require("datenbank.php");

  // Einträge auslesen
  $anfrage = "SELECT * FROM adressen";
  $ergebnis = mysql_query($anfrage);

  // String zusammensetzen
  while($zeile = mysql_fetch_array($ergebnis)) {
        $options .= "<option value='".$zeile['id']."'>".$zeile['vorname']."
  ".$zeile['nachname']."</option>\n";
  }
```

```
  // Template einlesen
  $templateInhalt = templateEinlesen('template.htm');
  $templateInhalt = str_replace("<!--adressen-->",$options,$templateInhalt);

  // Seite ausgeben
  echo $templateInhalt;
?>
```

Wir binden `funktionen.php` ein, um auf die Funktion `templateEinlesen()` Zugriff zu haben. Nach Herstellen der Datenbank-Verbindung (`datenbank.php`) lesen wir alle Adressdaten aus und erzeugen einen String nach dem Schema

```
<option value="1">Vorname Nachname</option>
<option value="2">Vorname Nachname</option>
```

Jede Option ist mit einem Wert versehen, der der ID des Datensatzes entspricht.

Mit Hilfe der Funktion `templateEinlesen()` (Implementierung s.u.) steht uns der Inhalt von `template.htm` in einer Variablen zur Verfügung. Den Platzhalter `<!--adressen-->` ersetzen wir durch den String `$options`.

Dadurch erhalten wir Auswahleinträge ähnlich folgender Abbildung:

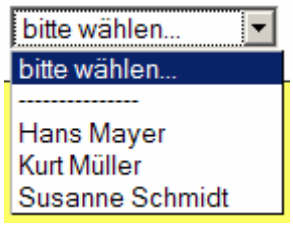


*Abbildung 25: Adresseinträge im Auswahlfeld*

Danach geben wir die Seite aus.

### 9.3.5 Einlesen des Templates (funktionen.php)

In `funktionen.php` implementieren wir die Methode `templateEinlesen()`, die uns das Template einliest und den Inhalt zurückliefert:

```
<?php
function templateEinlesen($datei) {
  $templZeiger = fopen($datei,"r");
  return fread($templZeiger,filesize($datei));
}
?>
```

Wir öffnen also die als Parameter `$datei` übergebene Datei lesend („`r`“) und lesen daraus alle Zeichen. Dies geschieht mit `fread`, das einen Dateizeiger und als zweiten Parameter die Anzahl der zu lesenden Bytes erwartet. Diese bestimmen wir mit `filesize($datei)`.

### 9.3.6 Herstellen der Datenbankverbindung (datenbank.php)

Wie bei vorherigem Beispiel stellen wir in `datenbank.php` die Datenbankverbindung her:

```
<?php
  $db_verbindung = mysql_connect("server","benutzername","passwort");
  $db_db = mysql_select_db("datenbankname");
?>
```

Auch hier müssen wieder die entsprechenden Parameter angepasst werden.

### 9.3.7 AJAX-Code (ajax.js)

AJAX kommt bei jedem onChange des Auswahlmenüs ins Spiel. Wird ein neuer Eintrag ausgewählt, sollen die zugehörigen Adressdaten angefordert werden.

Dazu senden wir einen XMLHttpRequest mit Hilfe von folgendem Code:

```
// Methode zur browserspezifischen Erzeugung eines XmlHttpRequest-Objekts
function getXmlHttpRequestObject() {
  if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
  } else if(window.ActiveXObject) {
        return new ActiveXObject("Microsoft.XMLHTTP");
  } else {
        alert("Bitte Browser updaten");
  }
}

// globales XMLHttpRequest-Objekt erzeugen
var searchReq = getXmlHttpRequestObject();

// Wir bei onkeyup im Suchfeld aufgerufen. Startet den AJAX-Request
function adresseAnzeigen() {
  if (searchReq.readyState == 4 || searchReq.readyState == 0) {
        var str = escape(document.getElementById('selectAdresse').value);
        searchReq.open("GET", 'adressen.php?id=' + str, true);
        searchReq.onreadystatechange = handleAdresse;
        searchReq.send(null);
  }
}
```

Der Code sollte inzwischen größtenteils selbsterklärend sein. Die Adressdaten erwarten wir von `adressen.php`, der wir die Datensatz-ID als GET-Parameter übergeben, also z.B.

```
adressen.php?id=1
```

Die ID des Adressdatensatzes erhalten wir durch

```
document.getElementById('selectAdresse').value;
```

### 9.3.8 Die Server-Seite (adressen.php)

Neu an der Server-Seite ist dieses Mal, dass wir die Daten in XML-Form zurückliefern möchten.

Es folgt der erste Code-Abschnitt aus `adressen.php`:

```
<?php
header("Content-Type:text/xml");

// Wurde GET-Parameter gesetzt?
if (isset($_GET['id']) && $_GET['id'] != '') {

  require("datenbank.php");

  // Suchvorschläge abfragen
  $ergebnis = mysql_query("SELECT * FROM adressen WHERE id='".$_GET['id']."'
  LIMIT 1");
```

Gleich zu Beginn setzen wir mit der PHP-Funktion `header`, mit der wir den Response-Header bearbeiten können, den Content-Type auf „text/xml“. Dies ist unabdingbar, da sonst die Daten später nicht mit der `responseXML`-Eigenschaft des XMLHttpRequests ausgelesen werden können.

Falls ein korrekter ID-Parameter übergeben wurde, stellen wir eine Datenbankverbindung her und rufen den entsprechenden Adressdatensatz ab.

Jetzt kommt XML ins Spiel. Zum Erzeugen von XML-Daten benutzen wir DOM-Funktionen von PHP 5.

Wir legen uns ein neues DOM-Dokument-Objekt an:

```
$dok = new DomDocument('1.0');
```

Auf Objektmethoden und -attribute haben wir in PHP über den Pfeil-Operator Zugriff. Dies entspricht der Punkt-Notation in Java(-Script).

Wir erzeugen einen Wurzelknoten:

```
$root = $dok->createElement('root');
$root = $dok->appendChild($root);
```

Um komfortabler auf den Datensatz zugreifen zu können, schreiben wir uns den Datensatz in einen assoziativen Array:

```
$zeile = mysql_fetch_assoc($ergebnis);
```

Mithilfe von folgendem Code hängen wir die einzelnen Zeilen in den Wurzelknoten ein.

```
foreach ($zeile as $feldname => $feldwert) {
  // Zeichensatz konvertieren
  $feldname = iconv("ISO-8859-1","UTF-8",$feldname);
  $feldwert = iconv("ISO-8859-1","UTF-8",$feldwert);

  // Elementknoten
  $name = $dok->createElement($feldname);
  $kindKnoten = $root->appendChild($name);
  // Textknoten
  $wert = $dok->createTextNode($feldwert);
  $wert = $kindKnoten->appendChild($wert);
}
```

Hier zeigt sich der Vorteil des assoziativen Arrays: Nach der Foreach-Deklaration haben wir über die beiden Variablen `$feldname` und `$feldwert` Zugriff auf die Einträge einer Assoziation.

Wir erzeugen einen Elementknoten, der nach dem Spaltennamen benannt ist, und hängen darin einen

Textknoten mit dem Wert der Spalte ein.

Wichtig ist die Zeichenkonvertierung mit der PHP-Funktion `iconv()`, damit keine Probleme mit Umlauten wie ß in „Straße“ auftreten. Adressdaten liegen in der Datenbank im ISO-Format vor, im XML-Dokument wird UTF-8-Formatierung vorausgesetzt.

Danach schließen wir das Dokument ab und speichern es in der Variablen `$xml_string`:

```
$xml_string = $dok->saveXML();
```

die wir dann zurückgeben:

```
echo $xml_string;
```

Zum Abschließen des PHP-Skripts wird noch folgender Code benötigt:

```
} ?>
```

### 9.3.9 Testen der Server-Seite

Zum Testen der Server-Seite empfiehlt es sich, `adressen.php` direkt mit einem GET-Parameter `id` aufzurufen, also z.B.:

```
adressen.php?id=1
```

In Mozilla Firefox erhalten wir folgende farblich formatierte Ausgabe:

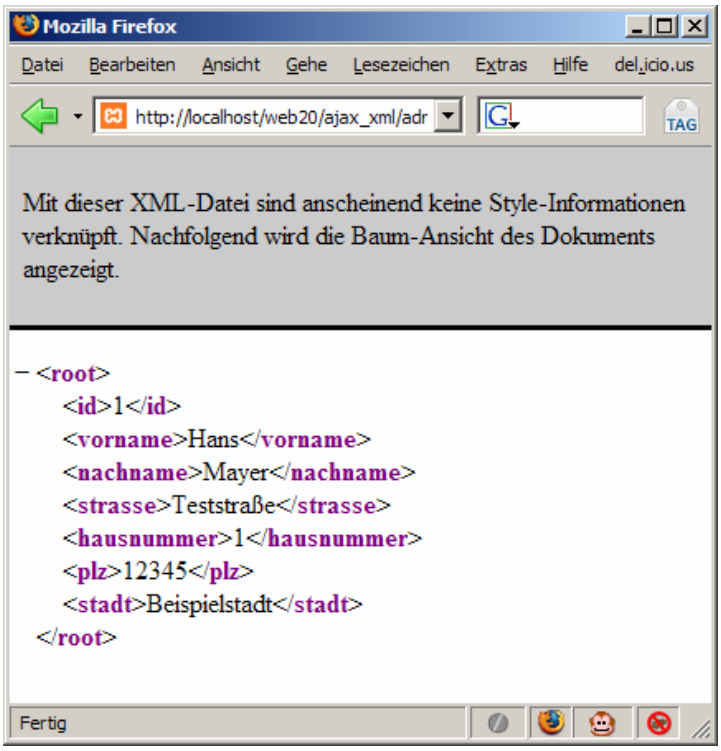


*Abbildung 26: Ausgabe der XML-Daten*

Es funktioniert also alles wie gewünscht: Jede Spalte wurde in einem Elementknoten unterhalb von root eingehängt; der jeweilige Wert ist in einem Textknoten festgehalten.

### 9.3.10 Verarbeitung der Rückgabewerte mit JavaScript (ajax.js)

Die XML-Daten möchten wir nun mit JavaScript auslesen und in die Spans schreiben.

Wir haben dazu den Request-Handler `handleAdressen()` registriert, den wir wie folgt implementieren:

```
// Handler für Rückgabe der Adressdaten
function handleAdresse() {
  if (searchReq.readyState == 4) {
        var adressDaten = searchReq.responseXML;

        // Ausgabe der einzelnen Daten
        document.getElementById('id').innerHTML =
  adressDaten.getElementsByTagName('id')[0].firstChild.nodeValue;
        document.getElementById('vorname').innerHTML =
  adressDaten.getElementsByTagName('vorname')[0].firstChild.nodeValue;
        document.getElementById('nachname').innerHTML =
  adressDaten.getElementsByTagName('nachname')[0].firstChild.nodeValue;
        document.getElementById('strasse').innerHTML =
  adressDaten.getElementsByTagName('strasse')[0].firstChild.nodeValue;
        document.getElementById('hausnummer').innerHTML =
```

```
    adressDaten.getElementsByTagName('hausnummer')[0].firstChild.nodeValue;
          document.getElementById('plz').innerHTML =
    adressDaten.getElementsByTagName('plz')[0].firstChild.nodeValue;
          document.getElementById('stadt').innerHTML =
    adressDaten.getElementsByTagName('stadt')[0].firstChild.nodeValue;
    }
}
```

Nach erfolgreichem Laden (`readySate=4`) speichern wir uns die XML-Daten in die Variable `adressDaten`.

Da wir in `adressen.php` den Content-Type auf `text/xml` gesetzt haben, haben wir über die Eigenschaft `responseXML` darauf Zugriff. Dies ist Voraussetzung für den darauffolgenden Code.

Würde die Definition des Content-Type fehlen, wäre `responseXML null`. Zwar stünden uns die XML-Daten in `responseText` zur Verfügung; wir könnten aber nicht mit Methoden wie `getElementsByTagName()` darauf zugreifen.

Mit Hilfe eben dieser Mehode erhalten wir alle Elemente mit entsprechendem Tag-Namen und wählen daraus das erste (und einzige) Element (`[0]`). Von dem eingehängten Textknoten holen wir uns mit Hilfe von

```
adressDaten.getElementsByTagName('stadt')[0].firstChild.nodeValue;
```

den Wert und schreiben ihn in `innerHTML` des entsprechenden Spans.

Dadurch erhalten wir folgende Ausgabe:

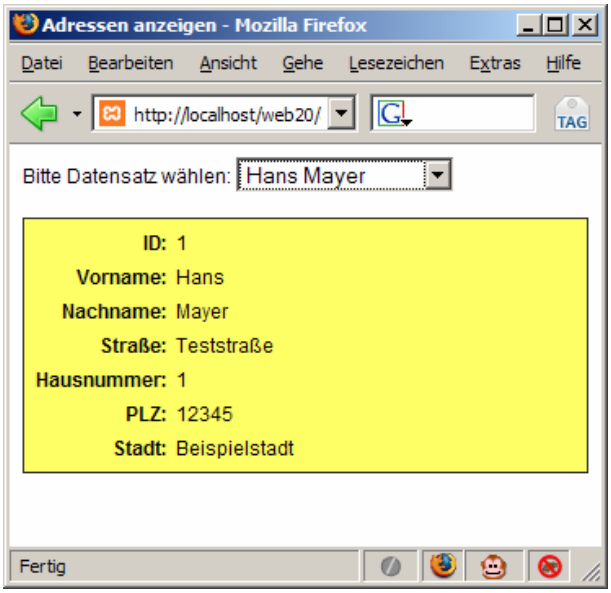


*Abbildung 27: Adresse anzeigen*

Dank der Umkodierung des Zeichensatzes treten keine Umlautfehler auf.

### 9.3.11 Zugriff auf Attribute

In obigem Beispiel benötigen wir keinen Zugriff auf Attribute. Trotzdem möchte ich dies der Vollständigkeit halber erwähnen.

Wir gehen von folgendem XML-Dokument aus:

```
<root>
  <party datum="01.01.2006">
        <gast>Hans Mayer</gast>
        <gast>Kurt Müller</gast>
        <gast>Susanne Schmidt</gast>
  </party>
</root>
```

Die XML-Daten sollen uns in der Variablen `xmlDaten` zur Verfügung stehen.

Dann erhalten wir mit Hilfe von folgendem Code Zugriff auf das Attribut `datum`:

```
var party = xmlDaten.getElementsByTagName("party")[0];
var datum = party.attributes.getNamedItem("datum").value;
```

### 9.3.12 Zusammenfassung

JavaScript bietet uns also vielfältige Funktionalitäten zum komfortablen Umgang mit XML-Daten.

Die Übertragung im XML-Format ist Voraussetzung, wenn wir komplex strukturierte Daten übertragen möchten. Eine Übertragung im Textformat, getrennt durch Zeilenumbrüche o.ä., würde eine starke Einschränkung der Strukturierung bedeuten.

Wir sind also bestens gerüstet, Anwendungen im großen Stil zu implementieren.

# 10 Bewertung und Ausblick

## 10.1 Typische AJAX-Fehler

AJAX bietet viele Vorteile und Möglichkeiten. Trotzdem gibt es typische AJAX-Fehler. Sehr gut zusammengefasst finden sich diese im Blog von Alex Bosworth[40].

**AJAX nur um des AJAX Willen verwenden**

Ein grundsätzlicher Fehler ist es, AJAX nur um des AJAX Willen zu verwenden. Man muss Vor- und Nachteile abwägen, die eine Realisierung mit AJAX mit sich bringt. So spricht z.B. weiterhin nichts dagegen, eine Google-Suche klassisch zu implementieren.

Ein weiteres Problem: AJAX ist in aller Munde; im Internet finden sich hunderte Proof-Of-Concept-Beispiele. Trotzdem trauen sich bisher nur wenige, eine komplette AJAX-Anwendung zu implementieren. Hier fehlt es bisher an Akzeptanz und Erfahrung.

**Fehlender Zurück-Button**

40 http://alexbosworth.backpackit.com/pub/67688

In einer AJAX-Anwendung verliert der durchaus nützliche Zurück-Button des Browsers seine Funktion. Allerdings ist dies eine Henne-Ei-Frage: Ist AJX Schuld, dass der Button nicht mehr die gewünschte Funktionalität bietet, oder impliziert der Browser ein seitenorientiertes Design, das nicht mehr gegeben ist?

Es gibt bereits erste Ansätze[41], den Zurück-Button auch in einer AJAX-Anwendung zu implementieren. Der Zustand der Anwendung wird dabei über Seitenanker (also z.B. `index.htm#15`) gesichert. Eine ausführliche Darstellung dieser Lösung würde jedoch den Rahmen sprengen.

**Fehlende Aktivitätsanzeige**

Wird im Browser eine neue Seite geladen, dient ein Fortschrittbalken als Aktivitätsanzeige. Dies ist bei einer AJAX-Anwendung, die per JavaScript Requests auslöst, nicht vorgesehen. Es empfiehlt sich daher, selbst eine Aktivitätsanzeige zu implementieren. Google Mail z.B. blendet oben rechts am Bildschirmrand eine kleine „Loading"-Anzeige ein.

**Kein Offline-Arbeiten möglich**

Dank AJAX ist es möglich, souveräne Anwendungen webbasiert zu implementieren. Dennoch darf nicht vergessen werden, dass heute noch manche einen langsamen Internetzugang haben.

Eine gute Lösung hierfür ist, Alternativen anzubieten. So sollte z.B. trotz einer gelungenen Webmail-Browser-Anwendung der klassische POP/STMP-Zugang nicht fehlen.

**Hohe Wartezeiten**

Während man bei einer klassischen Web-Anwendung davon ausgehen kann, dass bis zum Klicken eines Links oder Abschicken eines Formulars kein Verkehr zwischen Client und Server zu Stande kommt, ist dies bei einer AJAX-Anwendung nur bedingt vorhersehbar. Dies kann sich vor allem bei langsamen Internetverbindungen zu Verzögerungen führen.

**Sensible Daten werden ungesichert übertragen**

Wie in einer klassischen Anwendung werden auch in einer AJAX-Anwendung Daten ungesichert übertragen, nur eben asynchron. Deshalb kann auch AJAX eine klassische Sicherheitskonzeption wie z.B. Übertragung über eine gesicherte Verbindung nicht ersetzen.

**Entwicklerplattform mit Benutzerplattform gleichsetzen**

Wie bei einer klassischen Anwendung muss man von einer hohen Zahl von Benutzerplattformen ausgehen. In AJAX ist dies jedoch noch schwerwiegender: Während bei einer klassischen Anwendung häufig nur Darstellungsfehler auftreten, können bei einer AJAX-Anwendung JavaScript-Fehler, also Anwendungsfehler, auftreten. Ausführliches Testen ist unabdingbar.

**Performance auf Benutzerseite überschätzen**

Wenn wir Anwendungslogik auf den Client übertragen, ihn also zum Rich Client machen, muten wir ihm zusätzliche Rechenbelastung zu. Schlecht implementierter Code oder ein wenig leistungsfähiger Client können zu Performance-Einbußen führen.

**JavaScript voraussetzen**

41 http://www.contentwithstyle.co.uk/Articles/38/fixing-the-back-button-and-enabling-bookmarking-for-ajax-apps

Laut den W3C Browser-Statistiken[42] haben ca. 10% der Benutzer JavaScript, was für unsere AJAX-Anwendung unabdingbar ist, deaktiviert. Ein Bewusstsein über unsere Zielgruppe und deren Eigenheiten ist deshalb unabdingbar.

**Neue Konventionen für Benutzungsoberflächen einführen**

Beim Entwickeln einer AJAX-Anwendung stehen uns sämtliche Möglichkeiten zum Implementieren einer Benutzungsoberfläche zur Verfügung. Es ist deshalb darauf zu achten, dass wir bekannten Benutzungsmustern folgen. Beispiel: Schließen eines Elements durch Kreuz-Button oben rechts.

**Status-Veränderungen mit Links realisieren**

In einer AJAX-Anwendung dienen Links nicht nur zur Navigation innerhalb einer Anwendung, sondern können auch nur deren Zustand ändern. Hier muss eine Umgewöhnung der Anwender erfolgen.

**Den Benutzer durch unvorhergesehenes Reagieren der Anwendung verwirren**

Mit AJAX können asynchrone Requests gestartet werden. Ebenso kann die Anwendung asynchron auf Ereignisse und Responses vom Server reagieren. Es ist deshalb darauf zu achten, den Anwender nicht durch unvorhergesehenes Reagieren der Anwendung zu verwirren.

**Keine Links zum Bookmarken oder Weitergeben bereitstellen**

Es ist immer wieder praktisch, Links an andere weitergeben oder in den Lesezeichen speichern zu können, die mich bei späterem Aufruf wieder genau an die Stelle zurückbringen, an der ich mich gerade befinde. Dies ist bei einer AJAX-Anwendung nicht implizit gegeben. Es ist deshalb zu überlegen, Links zum direkten Aufruf des momentanen Zustands der Anwendung anzubieten. Dies ist z.B. in Google Maps vorgesehen.

**Lokale Änderungen nicht kaskadierend weitergeben**

Bei Änderungen des Anwendungszustand ist darauf zu achten, dass diese Änderung an allen relevanten Stellen der Anwendung durchgeführt wird. Häufig werden hier Kleinigkeiten übersehen.

Beispiel: Zwar können Bookmarks in Del.icio.us integriert bearbeitet werden und die Änderungen werden danach auch gleich angezeigt. Die Tag-Liste, die anzeigt, wie viele Bookmarks welchen Tag haben, wird jedoch erst bei einem kompletten Neuladen der Seite aktualisiert.

**Den Benutzer mit vielen kleinen Request ärgern**

Dem Anwender können viele kleine Requests als ständiger Netzverkehr negativ auffallen. Biete ich z.B. ein Optionen-Menü an, ist zu überlegen, ob jede Änderung sofort asynchron übertragen wird oder die Änderungen mit einem „Speichern“-Button abgeschlossen werden sollen.

Zudem können viele kleine Requests den Server unnötig belasten.

**Die Anwendung scrollen und den Fokus verschieben**

Ändert sich die Darstellung einer Anwendung, kann es natürlich auch zu einer Veränderung der Seitenlänge, also zu einer Veränderung der Scroll-Eigenschaft, kommen. Es ist dringend davon abzusehen, dass dem Benutzer dabei der Fokus verschoben wird. Auf der anderen Seite kann es passieren, dass er neue Inhalte nicht wahrnimmt, weil sie außerhalb seines Fokus liegen.

42 http://www.w3schools.com/browsers/browsers_stats.asp

Eine sinnvolle Planung ist auch hier unabdingbar.

**Ausschließen von Suchmaschinen**

Es ist zu bedenken, dass sich dynamische AJAX-Anwendungen nachteilig auf Suchmaschinen-Robots auswirken können. Der Anwendung sollte also so statisch wie möglich und so dynamisch wie nötig gehalten werden.

## 10.2 Das Gute an AJAX

Insgesamt lässt sich also sagen, dass mit AJAX viele Schwachpunkte, die sich in heutigen Web-Anwendungen ergeben, korrigiert werden können. Mit AJAX stehen uns umfangreiche Möglichkeiten zur Verfügung, souveräne Anwendung webbasiert abzubilden.

Wir haben festgestellt, dass die Abkehr von klassischen Implementierungen Neuerungen mit sich bringt, die wir bedenken müssen. So erfordert die Implementierung einer AJAX-Anwendung sehr viel Disziplin. Es gilt, Patterns einzuhalten und die Implementierung zu planen und strukturieren.

Das Zusammenspiel zwischen Client und Server erfolgt nach neuen Prinzipien. Daraus ergeben sich Konsequenzen (Stichwort Rich Client), die wir im Hinterkopf haben müssen. Auch müssen wir uns von klassischen Denkweisen wie seitenorientiertem Design gedanklich verabschieden. Dies hat Konsequenzen, die bedacht werden müssen. Gibt es eine Möglichkeit, den Zurück-Button zu ersetzen? Wie kann ich den Zustand einer Anwendung als Lesezeichen anbieten? Eine Umgewöhnung muss sowohl auf Entwickler- als auch auf Anwenderseite geschehen.

Grundsätzlich ist zu prüfen, wann eine klassische Implementerung einer AJAX-Implementierung vorzuziehen ist oder inwieweit AJAX-Anwendungen und klassische Implementierungen vermischt werden können.

Auf jeden Fall bleibt es spannend, was die Zukunft bereit hält und wie sich langfristig das Verhältnis klassische Implementierung – AJAX-Implementierung entwickeln wird.

# 11 Ressourcen

## 11.1 JavaScript

Einführung in JavaScript: http://de.selfhtml.org/javascript/sprache/index.htm

## 11.2 Verschlüsselung mit JavaScript

Verschlüsselung mit MD5-Algorithmus: http://pajhome.org.uk/crypt/md5/md5src.html

## 11.3 CSS und HTML

Einführung in CSS: http://de.selfhtml.org/css/index.htm

Vollständige CSS-Referenz: http://frixon.de/css21/css21_prop-visual.html

CSS-Validierung: http://jigsaw.w3.org/css-validator/

## 11.4 DOM

Objektreferenz: http://de.selfhtml.org/javascript/objekte/index.htm

## 11.5 Java Web Services (Sun Microsystems)

Java Web Services: http://java.sun.com/webservices/

Tutorial zu Java Web Services:
http://java.sun.com/webservices/docs/2.0/tutorial/doc/JavaWSTutorial.pdf

## 11.6 Active Server Pages / ASP.NET (Microsoft)

Offizielle Webseite: www.asp.net

OpenBook zu ASP.NET: http://www.galileocomputing.de/openbook/asp/

Tutorial zu ASP.NET: http://www.html-world.de/program/asp_ov.php

# 12 Quellen

## 12.1 Literaturquellen

| | |
|---|---|
| [AJAX] | Ajax in Action<br>Dave Crane, Eric Pascarello mit Darren James<br>©2006 by Manning Publications Co. |
| [FACE] | About Face 2.0 – The Essentials Of Interaction Design<br>Alan Cooper & Robert Reimann<br>Wiley Publishing Inc.<br>©2003 Alan Cooper |
| [PATTERNS] | Design Patterns: Elements of Reusable Object-Oriented Software<br>Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides<br>©1995 Addison-Wesley Publishing Company |
| iX 6/2006, Heise-Verlag: S. 72: „JavaScript Knigge“ | Behaviour Library |

## 12.2 Quellen im Internet

| | |
|---|---|
| http://de.selfhtml.org | Umfassende HTML-, CSS-, JavaScript- und DOM-Referenz |
| http://developer.mozilla.org | Mozilla Developer Center |
| http://www.w3.org | W3-Consortium |
| http://frixon.de/css21/css21_prop-visual.html | CSS 2.1-Referenz |
| http://ajax.get-the-code.de | Methoden und Eigenschaften des XMLHttpRequest-Objekts |
| http://www.w3schools.com | W3-Schools: Tutorials zu verschiedenen Themen |
| http://www.tonymarston.net/php-mysql/dom.html | Mit PHP XML schreiben |
| http://www.w3schools.com/xml/xml_parser.asp | Mit JavaScript XML lesen |

## 12.3 Bildquellen

| | |
|---|---|
| http://de.wikipedia.org/wiki/Ajax_(Programmierung) | Synchrone und asynchrone Kommunikation |

# 3 Ruby on Rails

*Teil 3*

# Einführung in Ruby on Rails

# Inhaltsverzeichnis

# 1. Übersicht über das Ruby on Rails Framework

## 1.1 Einleitung

Thema dieses und der folgenden Unterkapitel der Untersuchung der Web 2.0 Technologien ist das Ruby on Rails Framework. Wir werden einzelne Aspekte von Ruby on Rails auch unter dem Blickwinkel der im folgenden zitierten Aussagen betrachten, also versuchen, herauszufinden, was Ruby on Rails von anderen Web-Frameworks unterscheidet, und ob positiven Aussagen berechtigt sind:

“Ruby on Rails is an open-source web framework that’s optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.” (http://www.rubyonrails.com/)

“Ruby on Rails is a framework that makes it easier to develop, deploy, and maintain web applications.” (Dave Thomas in Agile Web Development with Rails)

“Rails is the killer app for Ruby.”
(Yukihiro Matsumoto (a.k.a. Matz), Creator of Ruby)

“Rails is the most well thought-out web development framework I’ve ever used. And that’s in a decade of doing web applications for a living. I’ve built my own frameworks, helped develop the Servlet API, and have created more than a few web servers from scratch. Nobody has done it like this before.”
(James Duncan Davidson, Creator of Tomcat and Ant)

“Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days.”
(Tim O’Reilly, Founder of O’Reilly Media)

Ruby on Rails wurde im Juli 2004 erstmals der Öffentlichkeit vorgestellt und liegt jetzt - im Frühjahr 2006 - in der Version 1.1 vor. Entwickelt wurde Ruby on Rails von David Heinemeier Hansson, einem 26 jährigen Dänen, der seit Ende 2005 in Chicago, USA, lebt und Gesellschafter von 37signals ist (http://www.37signals.com). Ruby on Rails entstand als Extrakt der Anwendung Basecamp, einem webbasierten Projektmanagement Tool, und soll - laut David Heinemeier Hanssons eigener Aussage - nach Basecamp erst die zweite Anwendung überhaupt gewesen sein, die er in Ruby schrieb.

Der Begriff Extrakt ist dabei so zu verstehen, dass er während Design und Implementierung von Ruby on Rails versucht hat, möglichst viele der Hauptaufgaben und “Best Practices” (wie z.B. Refactoring und Testing), die bei der Erstellung einer kommerziellen Web Anwendung auftreten, zu identifizieren und diese direkt in das Framework (als eine generische Codebasis) zu integrieren.[1]

Neben Ruby on Rails, das David Heinemeier Hansson inzwischen gemeinsam mit dem sog. “core team of committers” weiterentwickelt, entstehen unter seiner Regie weitere webbasierte Anwendungen wie z.B. Ta-da List, Writeboard, Backpack und Campfire, die selbstverständlich alle mit Ruby on Rails entwickelt wurden. 2005 gewann er den Google-O’Reilly Open Source Award als “Best Hacker” für Ruby on Rails und Anfang 2006 den 16th Jolt Product Excellence Award “Software Development” für Ruby on Rails 1.0. Er ist inzwischen ein gefragter Referent auf verschiedenen Konferenzen, wie z.B.

---

1 s. [THOMAS], S.2 und [BLACK], S.xix

der RailsConf 2006 (http://railsconf.org), sowie im Wired Magazin (Ausgabe 14.03, März 2006) mit einem kleinen Artikel vertreten. Das Linux Journal (http://www.linuxjournal.com) widmet ihm in Ausgabe 147 (Juli 2006) gar die Titelseite und ausführliche Berichte über Ruby on Rails, Ruby und RubyGems.

Wir beziehen uns während der Untersuchung von Ruby on Rails hauptsächlich auf das Buch "Agile Web Development with Rails" von Dave Thomas und David Heinemeier Hansson[2]. Zum einen, weil es - neben "Rails Recipes" von Chad Fowler - von den Ruby on Rails Entwicklern offiziell als Einstiegs-Lektüre empfohlen wird, zum anderen, weil wir denken, aus ihm den Umgang mit Ruby on Rails genau so kennen zu lernen, wie ihn sein Erfinder beabsichtigt hat.

Die Frage, wodurch sich Ruby on Rails von anderen Web-Frameworks unterscheidet kann man auf drei verschiedene Arten beantworten:

- die strikte Orientierung am MVC Paradigma (Model-View-Controller bzw. Model 2),
- die Wahl der Programmiersprache Ruby und
- die auf zwei einfachen Konzepten basierende Ruby on Rails Philosophie.

Durch die Art und Weise, wie diese drei Aspekte miteinander kombiniert sind und man ihre Auswirkungen während der Anwendungsentwicklung bemerkt, lässt sich unserer Meinung nach auch die große Zustimmung für das Framework begründen.

Weitere Unterschiede zu anderen Web-Frameworks sind z.B. die Code Generatoren `generate` und `scaffold`, die das Grundgerüst einzelner Klassen oder einer kompletten Anwendung automatisch erstellen, die integrierte Unterstützung für AJAX, das integrierte Unit-Test Framework oder die voneinander isolierten Umgebungen für Entwicklung, Test und Produktivbetrieb einer Anwendung.

Sowohl das Framework selbst, als auch jede Anwendung, die mit Ruby on Rails entwickelt wird, ist in der Sprache Ruby implementiert. Eine Rails Anwendung ist per Definition also auch eine Ruby Anwendung und zumindest Grundlagenkenntnisse in Ruby sind daher unerlässlich, um eine Rails Anwendung erfolgreich entwickeln zu können. Als ein besonderer Vorteil von Ruby on Rails gegenüber anderen Web-Frameworks wird sehr häufig die Geschwindigkeit genannt, mit der man eine Anwendung entwickeln kann (Vgl. z.B. das Zitat von Tim O'Reilly zu Beginn des Kapitels). Diese oder ähnliche Aussagen treffen unserer Meinung nach jedoch nur auf den Standard-Sprachwortschatz von Ruby on Rails zu. Oftmals muss man den Code einer Anwendung jedoch mit Ruby individuell anpassen bzw. erweitern. Für solche Fälle sind dann solide Kenntnisse und Erfahrungen im Umgang mit Ruby unverzichtbar. Ruby ist zwar eine universell einsetzbare, objekt-orientierte Programmiersprache, ist syntaktisch jedoch eher an Lisp als bspw. an Java angelehnt, und dürfte damit vielen unerfahrenen Programmierern zu Beginn ungewöhnlich erscheinen. Wir widmen uns dieser Thematik nochmals zu Beginn von Kapitel 3.

---

2 s. Literaturverzeichnis

## 1.2 Ruby on Rails

Das Ruby on Rails Framework (im Folgenden einfach Rails oder Rails Framework genannt), d.h. die Programme, Programmbibliotheken und -module, die von Rails installiert werden, dienen dazu, individuelle Rails Anwendungen zu erstellen.[3]

Als eine Rails Anwendung sei ein Programm definiert, mit welchem ein Anwender über einen Internetbrowser durch eine Menge bzw. Folge einzelner Webseiten interagiert. Die Anwendungsdomäne, d.h. der Einsatzbereich von Rails Anwendungen, sind interaktive, datenbankgestützte Web Anwendungen. Eine weitere Einschränkung hinsichtlich der Art der Anwendung existiert nicht.

Ein Framework (dt. Klassenrahmen) ist ein Strukturmerkmal vieler Klassenbibliotheken, das sich dadurch von herkömmlichen prozeduralen Bibliotheken unterscheidet, dass hierbei die eigene Klasse nicht einfach vordefinierte Funktionalität aufruft, sondern vielmehr selbst als spezialisierende Erweiterung eines vorgegebenen Klassenrahmens auftritt.[4]

Ein Framework besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wiederverwendbaren Entwurf[5], für eine bestimmte Klasse von Software, die durch die Anwendungsdomäne definiert ist, darstellen. Man spezialisiert ein Framework für eine Anwendung, indem man anwendungsspezifische Unterklassen der (i.d.R. abstrakten) Framework-Klassen bildet. Ein Framework legt die Entwurfsparameter für die Architektur einer Anwendung, wie z.B. die Unterteilung in Klassen und Objekte und ihre Zusammenarbeit, die jeweiligen zentralen Zuständigkeiten und den Kontrollfluss, im voraus fest, so dass sich der Anwendungsentwickler auf die spezifischen Details der Anwendung konzentrieren kann. Als Ergebnis können Anwendungen schneller entwickelt werden, haben aber auch alle eine ähnliche Struktur. Der Anwendungsentwickler verliert zwar dadurch kreative Freiheit, da viele Entwurfsentscheidungen bereits von den Entwicklern des Frameworks getroffen wurden, die Anwendungen sind aber i.d.R. einfacher zu warten und konsistenter.

Der Entwurf eines Frameworks ist - im Vergleich zu Entwürfen für Anwendungen oder Klassenbibliotheken - ungleich komplizierter, da die Architektur des Frameworks für alle Anwendungen aus der Anwendungsdomäne funktionieren muss. Entwürfe für Anwendungen hängen sehr stark vom Framework ab und sind gegenüber Änderungen der Framework-Schnittstellen besonders empfindlich. Ein auf Entwurfsmustern basierendes Framework kann i.d.R. viel leichter einen hohen Grad an Entwurf- und Codewiederverwertung in Anwendungen erreichen, als solche, die nicht auf Entwurfsmustern basieren. Ein typisches Entwurfsmuster zur Strukturierung von Frameworks, die eine Bildschirminteraktion gestalten, ist z.B. das auch in Rails eingesetzte MVC Paradigma (Model-View-Controller).[6]

---

3 s. http://rubyonrails.org/down für die Installationsanleitung von Rails

4 Definition aus [TDI], S.282

5 Ein wiederverwendbarer Entwurf wird auch Entwurfsmuster genannt. Ein Entwurfsmuster (engl. Design Pattern) ist eine bestimmte Vorgehensweise bei der Definition von Klassen und bei der Gestaltung ihrer Zusammenarbeit. (Definition aus [TDI], S.282)

6 Die Erläuterungen zu Frameworks haben wir aus [GoF] S.37f entnommen.

### 1.2.1 Die Rails Philosophie

Die Philosophie von Rails basiert im wesentlichen auf zwei einfachen Konzepten: DRY und Convention over Configuration. Zusammen mit dem MVC Paradigma beeinflussen beide den Entwurf des Frameworks sehr stark.

DRY bedeutet Don't Repeat Yourself[7] - jeder einzelne Aspekt der Anwendungslogik eines Systems sollte auch nur an exakt einer Stelle innerhalb der Anwendung festgehalten werden. In einer Rails Anwendung wird man daher wenig redundante Code-Fragmente finden - man platziert den Code an einer bestimmten - oft durch die MVC Architektur vorgegebenen - Stelle (d.h. in einer bestimmten Komponente oder Datei) und das Framework kümmert sich darum, die einzelnen Komponenten zur Laufzeit miteinander zu verknüpfen.

Convention over Configuration bedeutet, dass Rails sinnvolle und vernünftige Standardeinstellungen für fast alle Aspekte des Zusammenspiels und der Verknüpfung der einzelnen Komponenten einer Anwendung bietet. Folgt man diesen Konventionen, kann man eine komplette Rails Anwendung mit weniger Code schreiben, als "eine Java-basierte Web Anwendung normalerweise für XML-Konfigurationsdateien benötigt".[8] Diese Tatsache ist unserer Meinung nach allerdings ebenso sehr ein Verdienst der Sprache Ruby. Rails sieht natürlich auch vor, diese Standardeinstellungen bei Bedarf außer Kraft zu setzen (z.B. wenn man mit bestehenden Datenbankschemata arbeitet).

### 1.2.2 Die Ruby Programmierumgebung

Rails setzt eine Ruby Installation (Ruby 1.8.4 für Rails 1.0) voraus, die eine Vielzahl an Komponenten, Sprachbibliotheken und Supportdateien enthält. Aus diesem Grund gehen wir im Folgenden zuerst auf einige Punkte ein, welche eine Ruby Installation im Allgemeinen und außerhalb des Rails Kontextes betreffen. Der Kern der Sprache Ruby ist in C geschrieben. Im Verzeichnis des Ruby Quell-Codes befinden sich daher jede Menge C- und C-Header-Dateien. Nicht wenige Programmbibliotheken, die mit Ruby ausgeliefert werden, sind aber auch in Ruby geschrieben (z.B. im Verzeichnis `lib`).

#### 1.2.2.1 Durch die Ruby Installation navigieren

Ruby wird je nach Plattform und/oder verwendetem Paketmanager an unterschiedlichen Stellen im System installiert. Glücklicherweise kann man mit Ruby selbst herausfinden, wo sich die einzelnen Komponenten befinden. Man verwendet dazu den interaktiven Ruby Interpreter `irb` mit der Extension `rbconfig`:

```
$irb -rrbconfig
```

Das Kommando veranlasst `irb` dazu, die Konfigurationsinformationen für die spezielle Ruby Installation dieses Systems zu laden (`-r` steht für "require"). Die Informationen an sich werden mit einem Kommando der Form `Config::CONFIG["term"]` angezeigt, wobei `term` ein spezielles Schlüsselwort ist (Anm.: Wir haben dieses in der folgenden Liste für jeden Punkt am Ende in Klammern gesetzt). Ein Kommando in `irb` hat z.B. die Form:

```
irb(main):001:0> Config::CONFIG["bindir"]
```

---

7 der Begriff wurde durch das Buch "The Pragmatic Programmer: From Journeyman to Master" von Andrew Hunt und Dave Thomas geprägt.

8 Zitat aus [THOMAS], S.2

Wichtige Komponenten der Ruby Installation sind:

- Executables (`bindir`)
  Zeigt das Verzeichnis, in dem sich die ausführbaren Ruby Dateien befinden, z.B. `/usr/local/bin`. Darin enthalten sind u.a. die Programme `erb`, `irb`, `rdoc`, `ri` und `ruby`, sowie aus dem Rails Framework `rails` und `rake`.
- Extensions und Libraries Unterverzeichnisse (`rubylibdir`).
  Zeigt das Verzeichnis der Standard Ruby Libraries und Extensions, z.B. `/usr/local/lib/ruby/1.8`. Man findet darin u.a. die Dateien `erb.rb`, `debug.rb`, sowie die Implementierung von RDoc im Unterverzeichnis `rdoc`. Um die Funktionalität der Libraries und Extensions verwenden zu können, bindet man sie mit dem Schlüsselwort `require` in den eigenen Programmcode ein.
- C Extensions Verzeichnis (`archdir`)
  Dieses Verzeichnis befindet sich i.d.R. eine Ebene unterhalb von `rubylibdir` und enthält architektur-spezifische Libraries und Extensions. Man findet hier u.a. die Implementierung von `rbconfig` (`rbconfig.rb`).
- Das `site_ruby` Verzeichnis (`sitedir`) und seine Unterverzeichnisse (`sitelibdir, sitearchdir`)
  Zeigt das Verzeichnis der 3rd-Party Ruby Libraries und Extensions, z.B. `/usr/local/lib/ruby/site_ruby`. `sitedir` ist wie `rubylibdir` aufgebaut und mittels `require` angeforderte Funktionalität wird auch hier gesucht. Hier befindet sich u.a. der Quell-Code der RubyGems Implementierung.
- Das `gems` Verzeichnis
  Dieses Verzeichnis kann nicht mittels `rbconfig` gezeigt werden, da der RubyGems Paketmanager separat installiert werden muss. I.d.R. befindet sich `gems` jedoch auf der gleichen Ebene wie `sitedir`. Da Rails mit RubyGems installiert wird, befindet sich in den Unterverzeichnissen von `gems` der Rails Quell-Code.

Ein Ruby Programm auszuführen bedeutet, es durch den Ruby Interpreter `ruby` interpretieren zu lassen, oder anders gesagt, das Programm `ruby` mit dem eigenen Programm als Eingabe auszuführen.

Speichert man z.B. die folgenden Zeilen in der Datei `HelloWorld.rb` ab:

```
puts “Hello World!”
puts 2+2
```

und führt das Programm mit folgendem Kommando aus:

```
$ ruby HelloWorld.rb
```

erhält man die (erwartete) Ausgabe:

```
Hello World!
4
```

Standardmäßig werden mit Ruby einige wichtige Tools und Programme installiert, die teilweise auch von Rails verwendet werden. Einige davon werden wir im folgenden kurz vorstellen.

#### 1.2.2.2 irb

`irb` ist ein interaktiver Ruby Interpreter. Anstatt ein vollständiges Programm von `ruby` ausführen bzw. interpretieren zu lassen, führt bzw. wertet `irb` alle eingegebenen Kommandos nacheinander aus und zeigt deren Ergebnisse an:

```
irb(main):001:0> 2+2
=> 4
irb(main):002:0> days = 365
=> 365
irb(main):003:0> hours = 24
=> 24
irb(main):004:0> minutes = 60
=> 60
irb(main):005:0> days*hours*minutes
=> 525600
irb(main):006:0> exit
```

Das Rails Skript `console` (`script/console`) nutzt `irb`, um eine Session mit den Komponenten einer bestimmten Rails Anwendung (z.B. den Model-Klassen) in der Development Umgebung zu starten. Man hat somit die Möglichkeit, auch außerhalb eines Browsers mit der Anwendung zu interagieren.

#### 1.2.2.3 debugger

Der Ruby `debugger` führt je eine Instruktion im Programmcode aus und hält danach an. In diesen Pausen kann man mit einer Eingabe z.B. die Werte von Variablen untersuchen. Die Ausführung wird mit dem Kommando `step` fortgeführt. Um ein Ruby Programm per Kommandozeile zu debuggen verwendet man ein Kommando der Form:

```
$ ruby -rdebug HelloWorld.rb
```

Das Rails Skript `breakpointer` (`script/breakpointer`) nutzt den Ruby `debugger`.

#### 1.2.2.4 ri und RDoc

`ri` (Ruby Index) und RDoc (Ruby Documentation) sind einander ähnliche Tools, mit denen die Dokumentation zu Ruby Programmen erzeugt werden kann.

RDoc ist ein mit Javadoc vergleichbares Dokumentationssystem. Falls ein Programm Kommentare in einem speziellen Format und Markup enthält, kann `rdoc` (das entsprechende Kommandozeilen-Programm) diese extrahieren, anordnen und zur Darstellung visuell aufbereiten. Prinzipiell haben die Kommentare das folgende Aussehen:

```
# This is a comment for
# * a class,
# * a module or
# * a method
#
# Author:: Andreas Retter (mailto:andi@x.y)
#
```

Für weitere Informationen über die Verwendung von RDoc siehe http://rdoc.sourceforge.net/doc.

`ri` macht es möglich, die von RDoc aufbereiteten Kommentare darzustellen. Verwendet man `ri` per Kommandozeile, ist es am ehesten mit dem Linux/UNIX Kommando `man` (für “manual pages”) vergleichbar. Möchte man z.B. die vollständige Dokumentation von `require` ansehen, verwendet man das Kommando:

```
$ ri require
```

`ri` und RDoc sind die Arbeit von Dave Thomas.

#### 1.2.2.5 ERb

ERb (Embedded Ruby) von Seki Masatoshi macht es möglich, Ruby Code in einer HTML Datei zu verwenden. ERb liest dazu eine Datei (in Rails i.d.R. durch die Endung `.rhtml` gekennzeichnet, prinzipiell ist jedoch auch die Endung `.rb` möglich) und führt die darin enthaltenen Ruby Instruktionen aus. In Abhängigkeit der Art der Delimiter (dt. Trennsymbole), wird das Ergebnis der Instruktionen in die resultierende HTML Seite aufgenommen oder nur der Code ausgeführt.

Es gibt zwei Arten von Delimitern, deren Wirkung ähnlich denen der JSP Technologie von Java ist:

- Code, der nur ausgeführt werden soll, steht zwischen `<%` und `%>`
- Code, der ausgeführt und dessen Ergebnis in die Ausgabe aufgenommen werden soll, steht zwischen `<%=` und `%>`

ERb wird in Rails häufig eingesetzt. Die Browseranzeige einer Rails Anwendung ist in den allermeisten Fällen die Ausgabe eines von ERb verarbeiteten Dokuments. In Rails verwendet man z.B. Variablen, um Ergebnisse von Datenbankabfragen zu referenzieren. In Abhängigkeit der Werte dieser Variablen wird ein sog. View-Template (d.h. eine HTML Seite) gerendert, in das unter Einsatz von ERb die Variablen eingefügt bzw. evaluiert werden.

### 1.2.3 Das MVC Paradigma (Model-View-Controller)

Das MVC Paradigma (Trygve Reenskaug, 1979) beschreibt einen Architekturentwurf für interaktive Anwendungen. Dieser wurde erstmals in Smalltalk-80 zur Konstruktion von Benutzerschnittstellen verwendet. Der Entwurf unterteilt die Anwendung in die drei Klassen bzw. Komponenten Model, View und Controller. Dabei stellt das Model-Objekt das Anwendungsobjekt dar, das View-Objekt seine Bildschirmrepräsentation, und das Controller-Objekt bestimmt die Möglichkeit, mit denen die Benutzerschnittstelle (Graphical User Interface, GUI) auf Benutzereingaben reagieren kann. Das MVC Paradigma entkoppelt diese Objekte, um ihre Flexibilität, Modularität und Wiederverwertbarkeit zu erhöhen. Dies wird durch eine strikte Trennung der Verantwortlichkeiten des Codes der Anwendung erreicht. Ein Nebeneffekt dabei ist, dass der Code automatisch verständlicher, sowie einfacher zu warten und weiter zu entwickeln ist.

#### 1.2.3.1 MVC

Im ursprünglichen MVC Paradigma sind Model- und View-Objekte durch ein Protokoll zur Benachrichtigung voneinander entkoppelt. Dabei muss ein View-Objekt sicherstellen, dass seine Darstellung den Zustand des Model-Objektes wiedergibt. Dieser Zustand kann entweder persistent und außerhalb der Anwendung, wie z.B. in einer Datenbank, gespeichert sein, oder er kann sich fortlaufend, z.B. durch die Interaktionen eines Benutzers mit der Anwendung, ändern. In beiden Fällen benachrichtigt das Model die von ihm abhängigen Views, sobald sich seine Daten ändern. Der View hat daraufhin die Möglichkeit, sich selbst in einen konsistenten Zustand zu bringen, indem er auf die Daten des Models zugreift. Dieser Ansatz ermöglicht es, mehrere Views an ein Model zu binden, um verschiedene Repräsentationen der Daten des Models anzubieten (Vgl. Abb. 1). Außerdem kann man weitere Views für ein Model entwicklen, ohne dieses umschreiben zu müssen.

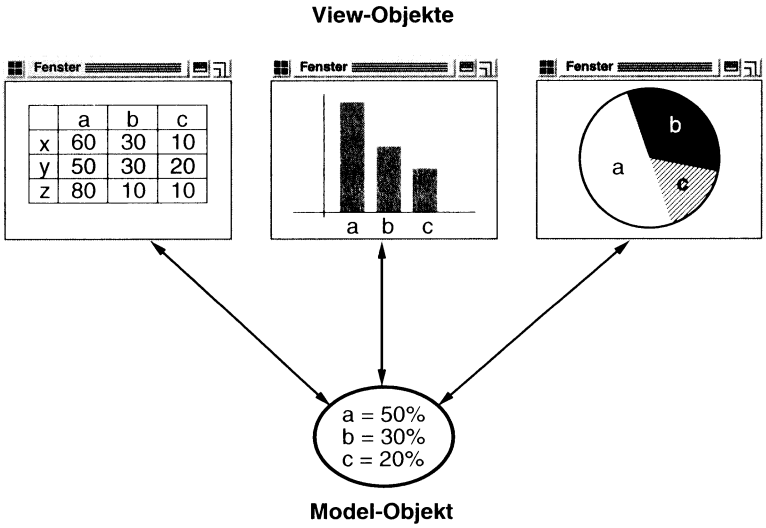


*Abb. 1: Kommunikation und Beziehungen zwischen Model-Objekt und View-Objekten. Die View-Objekte stellen unterschiedliche Repräsentationen der Daten des Models dar.*[9]

Anm.: Der Entwurf, View-Objekte von ihren Model-Objekten zu trennen, ist auf ein allgemeines Problem anwendbar: die Aktualisierung von entkoppelten Objekten. Änderungen eines Objekts können sich auf andere Objekte auswirken, ohne dass das geänderte Objekt die anderen genau kennen muss. Dieser allgemeine Entwurf wird durch das Beobachter Muster (engl. Observer Pattern) beschrieben. Das Observer Pattern wird in Rails im Modul ActiveRecord eingesetzt, um Callbacks in ein Model-Objekt einzubinden, ohne dieses selbst verändern zu müssen. Wir erläutern diese Möglichkeit im Kapitel ActiveRecord.

Das MVC Paradigma ermöglicht es auch, die Reaktion eines Views auf Benutzereingaben zu ändern, ohne seine visuelle Repräsentation zu ändern, indem es den Antwortmechanismus, d.h. eine bestimmte Antwortstrategie bzw. Ablauflogik, in den Controller kapselt. Ein View verwendet ein Exemplar einer Controller-Klasse (oder einer Subklasse, falls eine Klassenhierarchie von Controllern besteht), um eine bestimmte Antwortstrategie zu implementieren.

#### 1.2.3.2 MVC 2 oder Model 2 Architekturen

Für den Einsatz in Web Anwendungen bzw. Web-Frameworks muss das ursprüngliche MVC Pattern modifiziert werden. Insbesondere Zustandsänderungen im Model können nicht mehr direkt an den View (im Falle von Web Anwendungen i.d.R. ein Internetbrowser) kommuniziert werden, da HTTP ein zustandsloses Protokoll ist. Der Browser muss daher Zustandsänderungen mit jedem Request an den Server neu erfragen. Man spricht in diesem Zusammenhang von MVC 2 oder Model 2 Architekturen.

Im Vergleich zum “klassischen” MVC Paradigma existiert unseres Wissens nach keine einheitliche Definition für die MVC 2 oder Model 2 Architektur, der Begriff wird jedoch in erster Linie für Java-

[9] Abbildung aus [GoF], S.6

Frameworks eingesetzt. Je nach (Web-)Framework werden unterschiedliche Modifikationen an der ursprünglichen MVC Architektur vorgenommen und als MVC 2 oder Model 2 Architektur bezeichnet.

Folgt man den Erläuterungen aus Kapitel 4.4 [DEA] entstammt der Begriff einem frühen Entwurf der JSP Spezifikation, die zwei grundlegende Muster für den Einsatz von JSP Seiten beschrieb. Model 1 und Model 2 beziehen sich dabei auf die Ab- respektive Anwesenheit eines Controller-Servlets, welches die eintreffenden Requests des Clients bearbeitet und/oder weiterleitet und entsprechende Views für die Response auswählt.

In der Model 2 Architektur befindet sich das Controller-Servlet logisch betrachtet zwischen Browser und weiterführendem Inhalt, in gewissem Sinn zwischen Request und Response. Im Vergleich zu Model 1 entkoppelt Model 2 die einzelnen Views, da diese nicht mehr direkt aufeinander verweisen[10]. Model 2 zentralisiert die Anwendungslogik über die nächsten auszuführenden Schritte und setzt diese um, i.d.R. auf Basis der Auswertung von Request-URL, Eingabeparametern und dem aktuellen Zustand der Anwendung. Die auszuführenden Schritte umfassen sowohl den Aufruf einer Operationen im Model-Objekt als auch - in Abhängigkeit der Rückgabewerte dieser Operationen - die Auswahl und Erzeugung des nächsten darzustellenden Views, sowie seine Übertragung zurück zum Client (dynamische Auswahl des Views). [DEA] bezeichnet diesen Architekturentwurf “Front Controller Pattern”. Der Hauptvorteil liegt in der Zentralisierung, globale Anforderungen wie z.B. Security oder Logging können hier untergebracht werden. Durch die Entkopplung der Views kann außerdem flexibel auf sich verändernde “Page Flows” der Anwendung eingegangen werden. Model-Objekte können nach [DEA] für kleinere Anwendungen als JavaBeans realisiert werden. Controller (Servlet) und View (z.B. JSP) haben dabei über `useBean`, `getProperty()` und `setProperty()` Zugriff auf die Attribute des JavaBeans. Wie oben bereits erwähnt ist das Controller-Servlet bei der Aktualisierung des Views aufgrund einer Zustandsänderung im Model in jedem Fall beteiligt, da jede Zustandsänderung über die nächste Response kommuniziert werden muss; die Response wird durch das Controller Servlet initiiert, so dass der View, im Unterschied zum ursprünglichen MVC Pattern, nur indirekt in Verbindung zum Model steht. Der direkte Zugriff auf die Daten des Models bleibt dagegen auch in der Model 2 Architektur möglich.

In vielen Fällen (wie z.B. bei Struts[11]) wird zusammen mit der Model 2 Architektur auch die Frage diskutiert, wie viel Code und Anwendungslogik im View erlaubt bzw. wünschenswert ist, und ob dafür eine eigene Sprache benötigt wird oder nicht.

Im folgenden Unterkapitel fassen wir kurz die Umsetzung der Model 2 Architektur in Rails zusammen, die den Ideen aus [DEA] in vielen Fällen ähnlich ist.

## 1.3 Implementierung von MVC in Rails

In Übereinstimmung mit seinem MVC Fundament ist das Rails Framework hauptsächlich aus drei separaten Programmbibliotheken aufgebaut (realisiert als Ruby Module); separat in der Hinsicht, dass jede einen eigenen Namen hat und sie ggf. unabhängig voneinander eingesetzt werden können. Eine Standardinstallation von Rails vorausgesetzt, befindet sich ihr Quell-Code im `gems` Verzeichnis der Ruby Installation:

---

10 Eine Anwendung des Mediator Patterns

11 s. http://struts.apache.org

```
$ cd /usr/local/lib/ruby/gems/1.8/gems
gems> ls -l

drwxr-xr-x   9 root  staff  306 18 Mar 20:52 actionmailer-1.1.5
drwxr-xr-x  11 root  staff  374 18 Mar 20:52 actionpack-1.11.2
drwxr-xr-x  11 root  staff  374 18 Mar 20:52 actionwebservice-1.0.0
drwxr-xr-x  10 root  staff  340 18 Mar 20:52 activerecord-1.13.2
drwxr-xr-x   4 root  staff  136 18 Mar 20:52 activesupport-1.2.5
drwxr-xr-x   9 root  staff  306 18 Mar 21:04 fcgi-0.8.6.1
drwxr-xr-x  18 root  staff  612 18 Mar 21:19 mysql-2.7
drwxr-xr-x  16 root  staff  544 18 Mar 20:52 rails-1.0.0
drwxr-xr-x  12 root  staff  408 18 Mar 20:52 rake-0.7.0
drwxr-xr-x   3 root  staff  102 18 Mar 20:45 sources-0.0.1
```

Das Modul ActiveRecord (`activerecord-1-13.2`) entspricht dem Model im MVC Paradigma. Objekte vom Typ ActiveRecord::Base repräsentieren die Anwendungs- und/oder Businessobjekte (engl. entities) der Anwendung und stellen die korrekte und vollständige Umsetzung aller für die Anwendung geltender *Business Rules* sicher. ActiveRecord stellt außerdem ein Interface und ein Binding zwischen den Tabellen einer relationalen Datenbank (RDBMS) und dem Programmcode der Anwendung her. Klassenmethoden aus ActiveRecord setzen dabei z.B. die grundlegenden Operation auf Datenbanktabellen bzw. einzelnen Datensätzen um. Das Binding erfolgt nach dem Prinzip des objekt-relationalen Mappings (O/R-Mapping, ORM), d.h. Tabellen werden durch Klassen, Zeilen durch Exemplare der Klassen und Spalten durch Attribute abgebildet.

ActiveRecord unterscheidet sich jedoch von den meisten auf ORM basierenden Bibliotheken in der Art und Weise der Konfiguration. Im Vertrauen auf vernünftige Konventionen und auf Basis von sinnvoll gewählten Standardeinstellungen minimiert sich der Konfigurationsaufwand erheblich und es sind z.B. keine zusätzlichen XML-Konfigurationsdateien notwendig (Vgl. Convention over Configuration).

Das Modul ActionPack (`actionpack-1.11.2`) besteht wiederum aus den Modulen ActionController und ActionView, die den Controller und den View im MVC Paradigma repräsentieren. Sie sind deshalb im Modul ActionPack zusammengefasst, weil Controller und View i.d.R. tief ineinander greifen und sie typischerweise immer zusammen eingesetzt werden.

Anm.: Das bedeutet jedoch keineswegs, dass Controller- und View-Code einer Rails Anwendung miteinander vermischt sind. Die Gruppierung in einem Modul verdeutlicht nur, dass die Funktionalität aus Controller und View i.d.R. zusammen benutzt wird.

```
$ cd actionpack-1.11.2/lib
lib> ls -l

drwxr-xr-x  34 root  staff  1156 18 Mar 20:52 action_controller
-rw-r--r--   1 root  staff  3067 18 Mar 20:52 action_controller.rb
drwxr-xr-x   3 root  staff   102 18 Mar 20:52 action_pack
-rw-r--r--   1 root  staff  1144 18 Mar 20:52 action_pack.rb
drwxr-xr-x   8 root  staff   272 18 Mar 20:52 action_view
-rw-r--r--   1 root  staff  1417 18 Mar 20:52 action_view.rb
```

Das Modul ActionController koordiniert und kontrolliert die Kommunikation und die Abläufe zwischen Model und View. Es stellt z.B. Möglichkeiten zur Verfügung, Daten des Models zu manipulieren, sie zur Darstellung an den View weiter zu leiten, eintreffende Requests nach einem bestimmten Muster zur Bearbeitung an die Methoden der Controller Klassen zu übergeben und Session-Management oder Caching zu realisieren. Objekte vom Typ ActionController::Base koordinieren die Interaktion zwischen Anwendung und Benutzer, deren Großteil jedoch automatisch durch das Framework ge-

handhabt wird. Der Entwickler kann sich daher auf die Implementierung anwendungs-spezifischer Funktionalität konzentrieren.

Das Modul ActionView ist ein auf ERb basierendes System für die Erstellung von View-Templates zur Darstellung verschiedener Repräsentationen der Daten. In Web-Anwendungen besteht der View im einfachsten Fall aus statischem HTML Code, typischerweise bindet man jedoch auch dynamische Inhalte, die auch den Code des Controllers mit einbeziehen, in ihn ein. Die Verantwortung darüber, eine saubere Trennung zwischen Anwendungslogik und Darstellung einzuhalten, liegt in Rails beim Entwickler der Anwendung.

Alle drei Hauptmodule werden in den nach ihnen benannten Unterkapiteln ausführlich vorgestellt.

Da diese drei Module die komplette MVC Architektur darstellen, stellt sich die Frage, was genau Rails ist. Rails als Framework ist die simultane Bereitstellung der Hauptmodule, einschließlich ihrer Konfiguration und ihrer Verbindung untereinander. Es bezieht dabei den Umgang mit Webserver, CGI Bibliotheken, RDBMS und ERb mit ein.[12]

## 1.4 Hello Rails!

Um den Umgang mit Rails und einen der beiden fundamentalen Aspekte der Rails Philosophie “Covention over Configuration” zu demonstrieren, beschreiben wir im folgenden eine “Hello World!” Anwendung in Rails, “Hello Rails!”. Dabei soll dem Benutzer der Anwendung ein Gruß und das aktuelle Datum in einem Browserfenster angezeigt werden.

Wir werden die Anwendung so entwickeln, dass keine zusätzliche Konfiguration der Komponenten notwendig ist. Das bedeutet, dass wir die von Rails zur Verfügung gestellte Standardarchitektur benutzen, und uns selbst an gewisse Konventionen halten.

### 1.4.1 Eine neue Rails Anwendung anlegen

Ein Teil der Installation des Rails Frameworks ist das Kommandozeilen-Programm `rails`, welches man zum Anlegen jeder neuen Rails Anwendung benutzt. Warum braucht man dieses Programm? Wie bereits angedeutet, ist ein wichtiger Aspekt der Rails Philosophie, funktionierende Anwendungen ohne explizite Konfiguration der Komponenten zu erhalten (d.h. Convention over Configuration). Rails verknüpft die einzelnen Komponenten automatisch miteinander, setzt dafür jedoch u.a. eine bestimmte Verzeichnisstruktur voraus und genau diese wird mit dem Kommando `rails` erzeugt.

Um eine Anwendung anzulegen, öffnen wir ein Terminal, navigieren in das gewünschte Verzeichnis für unser Projekt und legen mit dem Kommando `rails NameDerAnwendung` die Anwendung an (im Beispiel `demo` genannt):

```
Home> cd Sites
Sites> rails demo
```

Das Kommando erzeugt das Verzeichnis `demo` und darin verschiedene Unterverzeichnisse und Dateien:

---

12 Definition aus [BLACK], S.40

```
Sites> cd demo
demo> ls -l

-rw-r--r--   1 andreasr  andreasr  6064  8 May 09:54 README
-rw-r--r--   1 andreasr  andreasr   307  8 May 09:54 Rakefile
drwxr-xr-x   6 andreasr  andreasr   204  8 May 09:54 app
drwxr-xr-x   2 andreasr  andreasr    68  8 May 09:54 components
drwxr-xr-x   8 andreasr  andreasr   272  8 May 09:54 config
drwxr-xr-x   2 andreasr  andreasr    68  8 May 09:54 db
drwxr-xr-x   3 andreasr  andreasr   102  8 May 09:54 doc
drwxr-xr-x   3 andreasr  andreasr   102  8 May 09:54 lib
drwxr-xr-x  10 andreasr  andreasr   340  8 May 09:54 log
drwxr-xr-x  14 andreasr  andreasr   476  8 May 09:54 public
drwxr-xr-x  12 andreasr  andreasr   408  8 May 09:54 script
drwxr-xr-x   7 andreasr  andreasr   238  8 May 09:54 test
drwxr-xr-x   3 andreasr  andreasr   102  8 May 09:54 vendor
```

Die für den Benutzer einer Anwendung sichtbaren Dateien befinden sich in `public`. Die wichtigsten Dateien in `public` sind jedoch die sog. *Dispatcher* (dt. abfertigen, verschicken, versenden), welche die eintreffenden Requests der Anwender nach einem bestimmten Muster an den Code unserer Anwendung weiterleiten. Die *Dispatcher* sind damit auf die korrekte Verzeichnisstruktur der Rails Anwendung angewiesen.

```
demo> cd public
public> ls -l

-rw-r--r--  1 andreasr  andreasr   235  8 May 09:54 404.html
-rw-r--r--  1 andreasr  andreasr   318  8 May 09:54 500.html
-rwxr-xr-x  1 andreasr  andreasr   479  8 May 09:54 dispatch.cgi
-rwxr-xr-x  1 andreasr  andreasr   861  8 May 09:54 dispatch.fcgi
-rwxr-xr-x  1 andreasr  andreasr   479  8 May 09:54 dispatch.rb
-rw-r--r--  1 andreasr  andreasr     0  8 May 09:54 favicon.ico
drwxr-xr-x  4 andreasr  andreasr   136 23 May 07:52 images
-rw-r--r--  1 andreasr  andreasr  7551  8 May 09:54 index.html
drwxr-xr-x  6 andreasr  andreasr   204  8 May 09:54 javascripts
-rw-r--r--  1 andreasr  andreasr    99  8 May 09:54 robots.txt
drwxr-xr-x  5 andreasr  andreasr   170 23 May 11:09 stylesheets
```

Unseren eigenen Code, einschließlich der View-Templates, erstellen wir in `app` bzw. dessen Unterverzeichnissen `models`, `views`, `controllers` und `helpers`.

```
public> cd ../app
app> ls -l

drwxr-xr-x  6 andreasr  andreasr  204 23 May 07:49 controllers
drwxr-xr-x  6 andreasr  andreasr  204 23 May 07:49 helpers
drwxr-xr-x  2 andreasr  andreasr   68  8 May 09:54 models
drwxr-xr-x  7 andreasr  andreasr  238  2 Jun 17:58 views
```

Das Verzeichnis `script` enthält Utilitiy-Skripte, d.h. für den Entwickler der Anwendung hilfreiche Skripte und Kommandozeilen-Programme; eines davon ist `server`, welches den integrierten Ruby

Webserver WEBrick[13] startet. Diesen (oder auch lighttpd[14]) verwendet man i.d.R. in der Development Phase.

```
app> cd ../script
script> ls -l

-rwxr-xr-x  1 andreasr  andreasr   95  8 May 09:54 about
-rwxr-xr-x  1 andreasr  andreasr  102  8 May 09:54 breakpointer
-rwxr-xr-x  1 andreasr  andreasr   97  8 May 09:54 console
-rwxr-xr-x  1 andreasr  andreasr   97  8 May 09:54 destroy
-rwxr-xr-x  1 andreasr  andreasr   98  8 May 09:54 generate
drwxr-xr-x  4 andreasr  andreasr  136  8 May 09:54 performance
-rwxr-xr-x  1 andreasr  andreasr   96  8 May 09:54 plugin
drwxr-xr-x  5 andreasr  andreasr  170  8 May 09:54 process
-rwxr-xr-x  1 andreasr  andreasr   96  8 May 09:54 runner
-rwxr-xr-x  1 andreasr  andreasr   96  8 May 09:54 server
```

Wir starten WEBrick oder lighttpd mit folgendem Kommando und können anschließend unter der angezeigten URL, i.d.R. http://0.0.0.0:3000 oder http://127.0.0.1:3000, sofort testen, ob unsere Anwendung schon verfügbar ist (Vgl. Abb. 2):

```
script> cd ../
demo> ruby script/server
...
=> Rails application started on http://0.0.0.0:3000
...
```

Bem.: Dieses Terminal sollte offen gehalten werden, da in ihm Teile des automatischen Loggings, das Rails zur Verfügung stellt, angezeigt werden. Die vollständige Log-Datei ist `development.log` im Verzeichnis `log`. Der Webserver wird mit `ctrl-c` gestoppt.

13 s. http://www.webrick.org

14 s. http://www.lighttpd.net

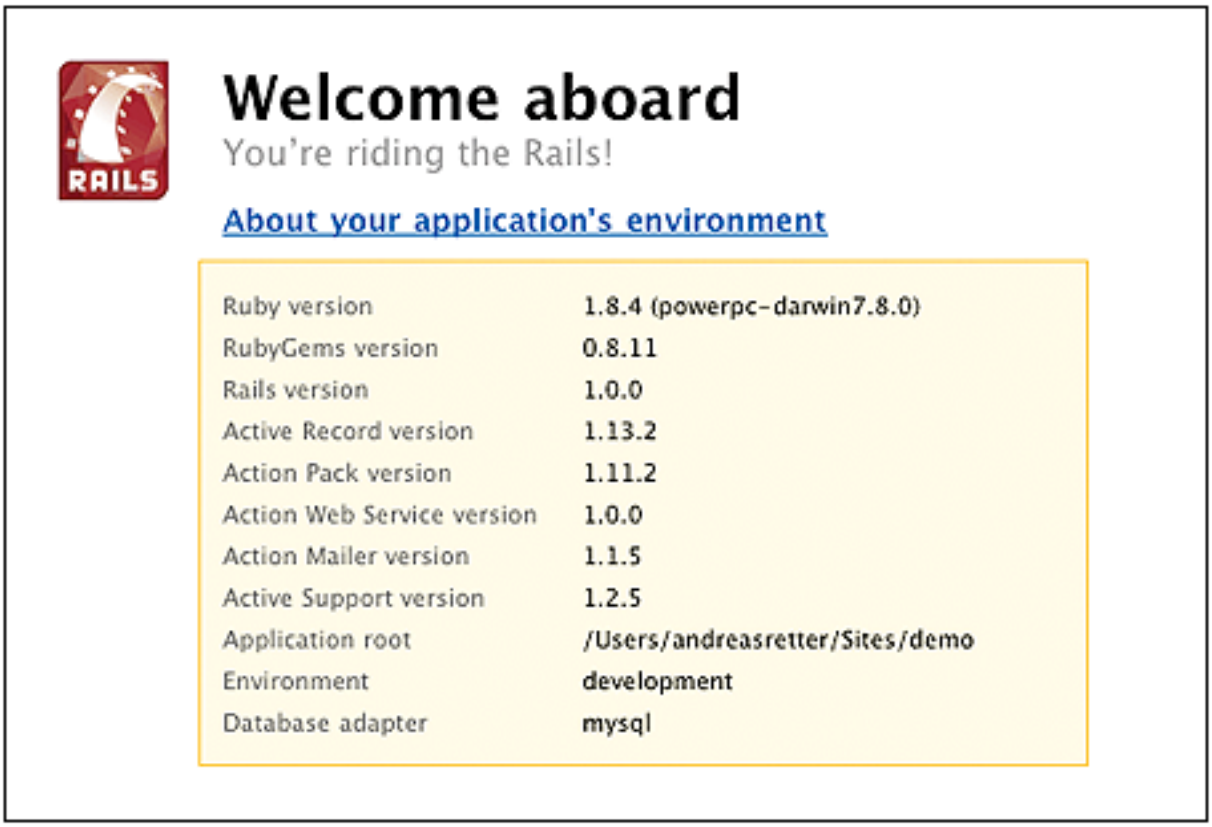


*Abb. 2: "You're riding the Rails!" Im Browser wird die Datei public/index.html angezeigt.*

### 1.4.2 Der Code der Anwendung

Wie bereits erwähnt, ist Rails ein MVC Framework. Rails nimmt die vom Browser ankommenden Requests an, übersetzt sie, um den passenden Controller zu finden, und ruft dann eine Methode (man spricht in diesem Zusammenhang auch von einer Action oder Action Methode) in diesem Controller auf. Die Methode interagiert i.d.R. mit einer Model-Klasse, dann wird ein bestimmter View bzw. ein View-Template aufgerufen, das im Browser als Ergebnisseite dargestellt wird.

In unserem einfachen Beispiel brauchen wir zunächst nur Code für einen Controller und für einen View, jedoch nicht für ein Model, da wir keine (persistenten) Daten verarbeiten. Den Ablauf definieren wir wie folgt: Ein Controller mit Namen `HelloRails` nimmt den Request entgegen und ruft durch die Methode `hello()` den View mit der passenden Ausgabe "Hello Rails!" auf.

Zusätzlich zur intakten Verzeichnisstruktur müssen in Rails auch die Dateien, die den Code der Anwendung enthalten, nach einem bestimmten Muster benannt und abgelegt sein.[15] Analog zum Programm `rails` zur Erzeugung der Verzeichnisstruktur existiert dafür das Skript `generate`.

15 s. Kapitel 2.2 Verzeichnisstruktur

Wir verwenden `generate` mit der Option `controller`, um eine Controller-Klasse zu erzeugen. Für unser Beispiel übergeben wir `generate` den Namen der Controller Klasse (`HelloRails`) und den Namen der Action Methode (`hello()`) als Parameter:

```
demo> ruby script/generate controller HelloRails hello
      exists  app/controllers/
      exists  app/helpers/
      create  app/views/hello_rails
      exists  test/functional/
      create  app/controllers/hello_rails_controller.rb
      create  test/functional/hello_rails_controller_test.rb
      create  app/helpers/hello_rails_helper.rb
      create  app/views/hello_rails/hello.rhtml
```

Rails erstellt beim Ausführen des Kommandos automatisch noch weitere Dateien, wie z.B.

- zu jedem als Parameter übergebenen Methodennamen ein korrespondierendes View-Template
- ein Helper Modul für den Controller im Verzeichnis `app/helpers`
- einen speziellen Controller für die Test Phase

Die Datei `hello_rails_controller.rb` enthält die Klassendefinition unseres `HelloRails` Controllers. Noch bevor wir diese bearbeiten, testen wir die URL, unter der der Benutzer später die Grußseite aufrufen können soll, http://0.0.0.0:3000/HelloRails/hello. `generate` erzeugt für jedes generierte View-Template den in Bild 3 zu sehenden Inhalt. Wir sehen damit zum einen, wo wir das Template finden, zum anderen, dass die Anwendung intakt ist und Rails alle benötigten Komponenten finden und miteinander verknüpfen kann. Deutlich erkennbar ist die Korrespondenz zwischen URL und Controller- und Actionname. Diesen in Rails schlicht routing genannten Mechanismus erläutern wir beginnend mit Kapitel 1.4.3 Rails und Request-URLs.


*Abb. 3: "Find me in app/views/hello_rails/hello.rhtml"*

Das Logging in lighttpd gibt u.a. Aufschluss über den aktuellen Controller, die aufgerufene Action und das gerenderte Template:

```
Processing HelloRailsController#hello (for 127.0.0.1 at 2006-05-08 10:03:12) [GET]
  Parameters: {"action"=>"hello", "controller"=>"hello_rails"}
Rendering hello_rails/hello
Completed in 0.01511 (66 reqs/sec) | Rendering: 0.00902 (59%) | 200 OK
[http://0.0.0.0/HelloRails/hello]
```

Öffnen wir die Klassendefinition des Controllers in `app/controllers/hello_rails_controller.rb`. Erstaunlicherweise enthält die Klasse bis auf die leere Methode `hello()` keinen weiteren Code:

```
class HelloRailsController < ApplicationController
  def hello
  end
end
```

Für das Beispiel reicht also alleine das Vorhandensein einer leeren Methode mit Namen `hello`, um Rails zu sagen, dass bei Aufruf dieser Methode ein View mit gleichem Namen, d.h. `hello.rhtml`, in einem Unterverzeichnis von `app/views` aufgerufen und als Ergebnisseite an der Browser gesendet werden soll. Wie in Bild 3 zu sehen, wird die Methode `hello()` durch den Aufruf der URL

http://0.0.0.0:3000/HelloRails/hello getriggert. Die Klasse `HelloRailsController` selbst erbt das Standardverhalten eines Controllers über die Klasse `ApplicationController` von ActionController::Base:

```
class ApplicationController < ActionController::Base
end
```

Anm.: Vererbung ist in Ruby durch das Zeichen "<" realisiert (entspricht `extends` in Java). Die beiden Doppelpunkte ": :" bedeuten, dass sich eine Klasse `Base` im Modul ActionController befindet. Module kann man in Ruby u.a. zur Definition von Namespaces verwenden, um Klassen mit demselben Namen gleichzeitig verwenden zu können (Vgl. `package` in Java):

```
module Tools
  class Hammer
  end
end

module Piano
  class Hammer
  end
end

tool = Tools::Hammer.new
piano = Piano::Hammer.new
```

Öffnen wir nun das View-Template `app/views/hello_rails/hello.rhtml`. In dieses Template schreiben wir einfach die gewünschte HTML Ausgabe; dynamischen Inhalt fügen wir ein, indem wir Ruby Code in das Template einbetten (deshalb auch die Dateiendung `.rhtml`. Damit erkennt Rails, dass der Inhalt mit dem ERb Filter (Embedded Ruby Filter) in reguläres HTML transformiert werden soll):

```
<html>
  <head>
    <title>Hello Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
    <p>The time is <%= Time.now %>.</p>
  </body>
</html>
```

Wir speichern die Datei und laden die Seite im Browser neu. Wie die folgende Abbildung zeigt, sind wir damit schon mit der Hello Rails! Anwendung fertig.


*Abb. 4: "Hello from Rails!"*

Zur Wiederholung noch einmal die notwendigen Schritte:

- Erzeugen der Anwendung mit `rails demo`
- Wechseln ins Verzeichnis mit `cd demo`
- Starten des Webservers mit `ruby script/server`
- Erstellen der Controller-Klasse und einer Action Methode mit `ruby script/generate ...`
- Bearbeiten des View-Templates `app/views/hello_rails/hello.rhtml`

### 1.4.3 Rails und Request-URLs

Wie jede andere Web-Anwendung auch, erscheint unsere Rails Anwendung dem Benutzer als Folge verschiedener URLs. Der Aufruf einer URL ist daher eine Interaktion mit unserem Programmcode, der eine Antwort für den Benutzer generiert.

Rails gewinnt die Information, was mit einem eintreffenden Request zu tun ist (d.h. welcher Code den Request bearbeiten soll) aus der Pfadangabe der Request-URL. Das routing Subsystem stellt fest, wie der eintreffende Request verarbeitet werden soll; routing identifiziert dazu den Controller und i.d.R. auch die Action Methode des Controllers, die ausgeführt werden soll. Dazu verwendet routing einen Algorithmus, der in unserem Beispiel die folgenden drei Regeln umsetzt:

- Der erste Teil der URL definiert die Anwendung (im Beispiel http://0.0.0.0:3000)
- Der nächste Teil der URL wählt den Controller aus (im Beispiel `HelloRails`)
- Der letzte Teil der URL identifiziert die aufzurufende Action Methode (im Beispiel `hello()`)

Man ist natürlich nicht dazu gezwungen, URLs der Form http://www.domain.com/anwendung/controller/action zu bilden, d.h. man kann den Algorithmus frei definieren. Die Interpretation des Algorithmus übernimmt ActionController.

### 1.4.4 Erweiterung der Anwendung

#### 1.4.4.1 Instanzvariablen und Environment des Controllers

Wie bereits erwähnt, erbt die Klasse `HelloRails` das Standardverhalten eines Controllers von ActionController:: Base. Dadurch sind u.a. zwei Dinge möglich:

- private Instanzvariablen des Controllers sind in allen View-Templates des Controllers verfügbar
- die View-Templates haben Zugriff auf das sog. Environment des Controllers.[16]

Wir erweitern die Anwendung beispielhaft, um zu illustrieren, dass Rails im Hintergrund die Zusammenarbeit zwischen den beiden Komponenten Controller und View tatsächlich selbständig organisiert und damit dem Entwickler einiges an Arbeitsaufwand abnimmt. Zum einen definieren wir dazu in der Methode `hello()` eine Instanzvariable `@time` und verwenden diese statt `Time.now` im View-Template, zum anderen lassen wir uns Informationen über die aktuelle Controller-Instanz (im Beispiel unten den Klassennamen) über das Attribut `controller` anzeigen:

```
class HelloRailsController < ApplicationController
  def hello
    @time = Time.now
  end
end
```

Der Code im View-Template lautet dann:

```
<html>
  <head>
    <title>Hello Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
    <p>The time is <%= @time %>.</p>
    <p>Controller class name is <%= controller.controller_class_name() %>.</p>
  </body>
</html>
```

Anm.: In Ruby beginnen die Namen von Instanzvariablen immer mit "@". Instanzvariablen sind außerdem implizit `private`, d.h. ihre Sichtbarkeit ist auf das Objekt, zu dem sie gehören, reduziert. Der Zugriff erfolgt über Accessor Methoden.

---

16 zum Begriff Environment s. Kapitel 2.5.2.1 Das Controller Environment

#### 1.4.4.2 Die Methode *link_to()*

Abschließend erstellen wir noch ein weiteres View-Template `goodbye.rhtml` und verlinken es mit `hello.rhtml`. `goodbye.rhtml` speichern wir ebenfalls im Verzeichnis `app/views/hello_rails`, da es durch Actions desselben Controllers aufgerufen wird. Der Code lautet:

```
<html>
  <head>
    <title>Goodbye from Rails!</title>
  </head>
  <body>
    <h1>Goodbye from Rails!</h1>
  </body>
</html>
```

Wir wissen bereits, dass Rails einen Algorithmus verwendet, um eine Request-URL in den Aufruf einer Action Methode zu "übersetzen". Ein View-Template hat also i.d.R. immer eine korrespondierende Action Methode in der Controller-Klasse[17]:

```
class HelloRailsController < ApplicationController
  def hello
    @time = Time.now
  end
  def goodbye
  end
end
```

Die URLs unserer Anwendung haben die Form /controller/action. Anstatt jedoch nach diesem Schema gebildete URLs direkt in die View-Templates einzufügen (wie z.B. `<a href="/hello_rails/goodbye">Goodbye!</a>`), verwendet man die Methode `link_to()`, welche einen Hyperlink auf eine Action erzeugt. `link_to()` ist eine sog. Helper Methode aus dem Modul ActionView::Helpers::URLHelper. Ein Vorteil ist, dass wir mit diesem generischen Ansatz keine Annahmen über die Bildung von URLs machen müssen und flexibler gegenüber Änderungen sind. `link_to()` übernimmt als ersten Paramter den Text, der als Hyperlink dargestellt werden soll und als zweiten Parameter ein Hash aus key => value Paaren, mit dem (im einfachsten Fall) festgelegt wird, welche Action Methode aufgerufen werden soll.[18]

```
<html>
  <head>
    <title>Hello Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
    <p>The time is <%= @time %>.</p>
    <p>Controller class name => <%= controller.controller_class_name() %></p>
    <p><%= link_to "Goodbye!", :action => "goodbye" %></p>
  </body>
</html>
```

Anm.: Ein Hash ist ein assoziatives (auch: inhaltsadressiertes) Array. Darunter versteht man eine Datenstruktur, in der die Daten über einen die Daten kennzeichnenden Suchschlüssel abgelegt und aufge-

---

17 zwingend notwendig ist dies allerdings nicht, s Kapitel 2.5.2 Action Methoden

18 weitere Informationen zu link_to() s. Kapitel 2.6.3 Erzeugung von Hyperlinks

funden werden.[19] Die wichtigsten Operationen auf dieser Datenstruktur sind das Einfügen eines neuen Elements, das Suchen eines Elements zu einem Suchschlüssel, das Entfernen eines Elements und das iterative Durchlaufen aller Elemente. In Ruby ist die Datenstruktur in der Klasse `Hash` realisiert. Die Java Klassenbibliothek definiert die abstrakten Operationen im Interface `java.util.Map`. Assoziative Arrays werden in Rails sehr oft als Methoden-Parameter verwendet. Anstatt Hash wird auch häufig der Begriff Dictionary verwendet.

19 Definition aus [TDI], S.226

# 2. Die Hauptbestandteile von Ruby on Rails

Kapitel 2 untersucht die wichtigsten Bestandteile des Frameworks, die Module ActiveRecord, ActionController und ActionView. In Bezug zu den Erläuterungen über den Entwurf von Frameworks in Kapitel 1.2 wollen wir zeigen, wie Rails den Anwendungsentwickler bei seiner Arbeit unterstützt und welche Vorgaben er dabei berücksichtigen muss.

## 2.1 Namens-Konventionen

Namens-Konventionen tragen einen sehr großen Teil zur Erfüllung des fundamentalen Aspekts der Rails Philosophie "Convention over Configuration" bei. Konkret bedeutet dies, dass Rails Dinge automatisch benennt, ähnlich benannte Dinge automatisch miteinander in Verbindung bringt, oder der Anwendungsentwickler Dinge nach bestimmten Regeln benennen muss. Zu Beginn sind diese Regeln teilweise ungewöhnlich, z.B. dann, wenn Rails eine Model-Klasse `Person` automatisch in Bezug zu einer Datenbanktabelle `people` setzt.

Dieses Unterkapitel dokumentiert die vorgegebene Namensbildung in Rails und ihre Auswirkungen. Die beschriebenen Regeln sind die von Rails verwendeten Standardkonventionen. Sie können alle durch entsprechende Deklarationen in den Klassendefinitionen überschrieben werden.

Nach Konvention werden Variablennamen in Ruby klein geschrieben und mehrere Wörter werden durch einen Unterstrich "_" voneinander getrennt (z.B. `my_variable`). Modul- und Klassennamen werden dagegen anders gebildet: Unterstriche werden nicht verwendet, sondern das erste Zeichen des Namens und das erste Zeichen jedes neuen Wortes wird groß geschrieben (gemischte Groß- und Kleinschreibung, z.B. `HelloRails`, `ActiveRecord`, `ActionController`).

Diese Ruby Konventionen werden auf zwei Arten erweitert. Rails nimmt an, dass Dateinamen und Namen von Datenbanktabellen wie Variablennamen gebildet werden. Für Datenbanktabellen gilt zusätzlich, dass die Namen immer aus der Pluralform des Namens der korrespondierenden Model-Klasse hergeleitet sind (z.B. `orders` statt `order`, `titles` statt `title`, usw.).

Rails verwendet die Kenntnisse über Namens-Konventionen, um Namen automatisch zu konvertieren und miteinander in Verbindung zu bringen. Dazu ein Beispiel: In einer Anwendung sollen Buchtitel zu einer Liste hinzugefügt werden können, um diese später z.B. ausleihen oder kaufen zu können. Die Liste enthält also eine Menge von Titeln als Listenelemente. Die Model-Klasse könnte dazu `ListElement` genannt werden. Aus diesem Namen würde Rails automatisch folgern, dass

- die korrespondierende Datenbanktabelle den Namen `list_elements` hat
- die Klassendefinition von `ListElement` in der Datei `list_element.rb` unter `app/models` steht.

Für Controller gelten zusätzlich weitere Konventionen, die sich ebenfalls am besten anhand eines Beispiels erklären lassen: Die Verwaltung der Buchtitel soll durch den Controller `Bookadmin` erfolgen. Daraus schließt Rails automatisch, dass

- die Klasse den Namen `BookadminController` hat und in der Datei `bookadmin_controller.rb` unter `app/controllers` definiert ist
- ein Helper Modul `BookadminHelper` in der Datei `bookadmin_helper.rb` unter `app/helpers` definiert ist

- die View-Templates für den Controller im Verzeichnis `app/views/bookadmin` abgelegt werden
- die View-Templates in die Layout-Datei `bookadmin.rhtml` unter `app/views/layouts` eingebunden werden

Alle Konventionen lassen sich wie folgt gruppieren und zusammenfassen. Details werden uns in den folgenden Unterkapiteln immer wieder begegnen:

Model:

| | |
|---|---|
| Tabelle | `list_element` |
| Klasse | `ListElement` |
| Datei | `app/models/list_element.rb` |

Controller:

| | |
|---|---|
| URL | http://.../HelloRails/hello |
| Klasse | `HelloRailsController` |
| Datei | `app/controllers/hello_rails_controller.rb` |
| Methode | `hello()` |
| Layout | `app/views/layouts/hello_rails.rhtml` |

View:

| | |
|---|---|
| URL | http://.../HelloRails/hello |
| Datei | `app/views/hello_rails/hello.rhtml` |
| Helper | Modul `HelloRailsHelper` |
| Datei | `app/helpers/hello_rails_helper.rb` |

In konventionellem Ruby Code müssten die Quelldateien für inkludierte Module oder Klassen vor deren Verwendung mit dem Schlüsselwort `require` in die Datei eingebunden werden. Da Rails jedoch die Beziehungen zwischen Datei- und Klassennamen kennt, ist die Verwendung von `require` in einer Rails Anwendung nicht notwendig. Bei erstmaliger Referenzierung einer unbekannten Klasse oder eines unbekannten Moduls übersetzt Rails deren Namen mit Hilfe der Namens-Konventionen in Dateinamen und lädt diese im Hintergrund automatisch nach. Eine Umsetzung dieses Prinzips findet man z.B. bei den Helper Modulen für einen View.

Einzig Klassen, die in Sessions verwendet werden, müssen einem Controller bzw. der gesamten Anwendung aus Gründen, die im Kapitel ActionController erläutert sind, explizit “vorgestellt” werden. Man verwendet jedoch auch hier nicht `require`, sondern eine konfigurations-artige Anweisung der Art `model :symbol` (im Vorgriff auf Kapitel 3 sei an dieser Stelle schon darauf hingewiesen, dass es sich hierbei jedoch auch um einen regulären Methodenaufruf handelt.)

```
class ApplicationController < ActionController::Base
  model :list_item
end
```

Anm.: Symbole sind Objekte der Ruby Klasse `Symbol`. Diese Klasse besitzt einen sog. *Literal Constructor*, d.h. anstatt `new` (z.B. `Object.new`) verwendet man eine spezielle Notation, um ein Objekt zu erzeugen. Die Klasse `String` verwendet z.B. Anführungszeichen (z.B. `"string"` oder `'string'`), `Array` verwendet eckige Klammern (z.B. `[1,2,3]`). Symbole verwenden einen voran gestellten Doppelpunkt (z.B `:symbol`).

In Ruby dienen Symbole neben Strings der Repräsentation von Text, dabei gibt es jedoch zwei entscheidende Unterschiede. Zum einen kann nur ein einziges Objekt der Klasse `Symbol` zu einem be-

stimmten Text existieren, d.h. jedes mal, wenn ein bestimmtes Symbol eingesetzt wird, hat man eine Referenz auf dasselbe `Symbol` Objekt. Zwei Strings sind dagegen zwei String-Objekte, egal, ob sie gleich sind oder nicht:

```
>> :symbol.equal?(:symbol)
=> true
>> “string”.equal?(”string”)
=> false
>> “string” == “string”
=> true
```

Zum anderen sind Symbole *immutable*, d.h. sie sind nach ihrer Erzeugung nicht mehr veränderbar. Strings können dagegen auch nach ihrer Erzeugung verändert werden:

```
>> str = “Hi “
=> “Hi ”
>> str + “there!”
=> “Hi there!”
```

Symbole sind sinnvoll, wenn man ein Wort immer und immer wieder benutzt, um etwas anderes zu repräsentieren, z.B. einen Schlüssel in einem Hash. In der Methode `link_to()` sind `:controller` und `:action` Symbole. Wenn ein Anwendung Hunderte von Hyperlinks, oder zumindest Hunderte von Referenzen auf verschiedene Controller und Actions hat, ist es deutlich effizienter, Symbole anstatt Strings zu verwenden, um den Speicherbedarf zu reduzieren.

## 2.2 Verzeichnisstruktur

Um fehlerfrei zu funktionieren, setzt Rails zur Laufzeit eine bestimmte Verzeichnisstruktur voraus, deren wichtigste Aspekte im folgenden beschrieben werden. Wie im Kapitel 1.4 Hello Rails! bereits beschrieben, wird mit dem Kommando `rails` eine neue Anwendung erzeugt. Dieses Kommando legt nun im Grunde genau diese Verzeichnisstruktur an und reichert sie mit etwas Ruby Code an.

Der größte Teil der Entwicklungsarbeit findet in den Verzeichnissen `app`, `test` und `public` statt. Unter `app` befinden sich die Verzeichnisse `models`, `views`, `controllers` (in direkter Analogie zum MVC Pattern) und `helpers`.

Der Webserver sieht `public` als Wurzelverzeichnis der Anwendung, in welchem sich daher die nach außen sichtbaren Dateien der Anwendung befinden. Automatisch generiert finden sich darin u.a. Dateien wie `index.html` (als Startseite der Applikation), `404.html` und `500.html` (als Standard-Fehlerseiten), sowie die Unterverzeichnisse `images`, `javascripts` und `stylesheets`, deren Bedeutung selbsterklärend sein dürfte.

In `doc/app` befindet sich die HTML-Dokumentation, welche mit RDoc (Ruby Documentation) über das Kommando `rake appdoc`[20] erzeugt wird; die Begrüßungsseite der Dokumentation ist `doc/README_FOR_APP`. Diese kann beliebig gestaltet werden.

20 rake ist vergleichbar mit GNU make (http://www.gnu.org/software/make)

Die von Rails automatisch erzeugten Log-Dateien (`development.log` protokolliert z.B. die vom Webserver erzeugten Meldungen und eignet sich daher gut für eine erste Fehlersuche) befinden sich unter `log`. Logging wird von Rails direkt unterstützt, indem sämtlichem Code der Anwendung ein Logger-Objekt zur Verfügung steht. Logger, ein einfaches Logging Framework, ist fester Bestandteil von Ruby, mit dem sich z.B. Warnmeldung verschiedener Stufen (`info`, `warn`, `error` und `fatal`) erzeugen lassen.

`lib` und `vendor` enthält Code, welcher nicht exklusiv für die Anwendung geschrieben wurde (und daher nicht unter `app` zu finden sein sollte).

`script` enthält die Programme und Utilities, welche für den Entwickler der Anwendung wichtig sind. Das meist gebrauchte ist hier sicherlich der Code Generator `generate`, sein Gegenstück `destroy`, sowie `server` (startet die integrierten Webserver WEBrick oder lighttpd). `console` ermöglicht die direkte Interaktion mit der Anwendung via `irb` (dem Ruby Interpreter) und `breakpointer` das Debugging der Anwendung.

## 2.3 Rails Konfiguration

Die Laufzeitkonfiguration einer Rails Anwendung wird durch die Dateien im Verzeichnis `config` festgelegt. Diese setzen das Konzept der Laufzeitumgebung um.

Verschiedene Laufzeitumgebungen sind wichtig, da die Bedürfnisse eines Anwendungsentwicklers sich je nach Entwicklungsphase (Entwicklung, Test oder Produktivbetrieb) erheblich voneinander unterscheiden. In der Entwicklungsphase möchte man z.B. ausführliches Logging, ausführliche Fehlermeldungen oder automatisches Nachladen von geändertem Code zur Laufzeit, während im Produktivbetrieb z.B. die Performance der Anwendung im Vordergrund steht.

Rails unterstützt dieses Konzept durch die Laufzeitumgebungen Development, Test und Production, die jeweils über eine eigene Menge von Konfigurationsparametern verfügen. Die anwendungsweite Konfiguration geschieht in der Datei `config/environment.rb`, die korrespondierenden Konfigurationsdateien zu den drei unterschiedlichen Laufzeitumgebungen befinden sich unter `config/environment`.

Man legt die Laufzeitumgebung beim Start des Webservers WEBrick mit folgender Anweisung fest:

```
demo> ruby script/server -e development    # or test, or production
```

Verwendet man Apache oder lighttpd wird sie durch die RAILS_ENV Umgebungsvariable festgelegt. Wichtig dabei ist, dass die Konfigurationsdateien außerhalb des Anwendungscode liegen, dieser damit also nicht geändert werden muss. lighttpd wird über die Datei `config/lighttpd.conf` konfiguriert, der folgende Auszug zeigt die Festlegung der Laufzeitumgebung:

```
# Default configuration file for the lighttpd web server
fastcgi.server = ( ".fcgi" => ( "localhost" => (
        "min-procs" => 1,
        "max-procs" => 1,
        "socket"    => "log/fcgi.socket",
        "bin-path"  => "public/dispatch.fcgi",
        "bin-environment" => ( "RAILS_ENV" => "development" ) ) ) )
```

Die anwendungsweite Laufzeitumgebung führt zusätzlich i.d.R. folgende Aufgaben aus:

- anwendungsweit benötigte Ressourcen erzeugen (z.B. ein Exemplar von Logger)
- den Ruby load path, durch den die Anwendung die zur Laufzeit benötigten Komponenten findet, anlegen. Dieser enthält i.d.R. die Verzeichnisse `app`, `app/models`, `app/controllers`, `app/helpers`, `components`, `config`, `lib`, `vendor` und `vendor/rails` und alle Verzeichnisse unter `app/models` und `components`, die mit einem Unterstrich oder einen Kleinbuchstaben beginnen

### 2.3.1 config/database.yml

Die Datei `database.yml` legt die Verbindungsparameter für einen bestimmten Datenbankadapter fest. Sie wird mit dem Kommando `rails` zum Anlegen einer Anwendung automatisch erzeugt und muss i.d.R. nur noch um den Datenbankname, Benutzername und Passwort ergänzt werden. Standardmäßig wird der MySQL Adapter ausgewählt. Jede neue Sektion beginnt mit dem Namen der Laufzeitumgebung, gefolgt von den speziellen Verbindungsparametern für den Datenbanktyp bzw. -hersteller.[21]

Für die Anwendung `books`, für die wir in den folgenden Unterkapiteln Beispiele entwickeln werden, hat `database.yml` die Form:

```
# MySQL (default setup)
# ...
development:
  adapter: mysql
  database: books_development
  username: andreasretter
  password:
  socket: /tmp/mysql.sock

test:
  adapter: mysql
  database: books_test
  username: andreasretter
  password:
  socket: /tmp/mysql.sock

production:
  adapter: mysql
  database: books_production
  username: andreasretter
  password:
  socket: /tmp/mysql.sock
```

Anm.: Jede Rails Anwendung arbeitet nach Konvention mit drei unterschiedlichen Datenbanken, `_development`, `_test` und `_production`. Für die Beispiele verwenden wir `_development`.

21 s. Kapitel 2.4.4 Verbindung zur Datenbank

## 2.4 ActiveRecord

Das Modul ActiveRecord ist die ORM-Ebene in Rails und die umfangreichste Komponente innerhalb des Frameworks. Übertragen auf das MVC Paradigma repräsentiert ActiveRecord das Model.

Objekt-relationales Mapping (O/R-Mapping, ORM, engl. object-relational mapping) beschreibt eine Lösung zur Abbildung von Daten, die in relationalen Datenbanken gespeichert sind, auf die Konzepte der objekt-orientierten Programmierung. ORM Systeme sind z.B. als Klassenbibliotheken oder Frameworks implementiert, die dieses Mapping (größtenteils) automatisch durchführen und dem Anwendungsentwickler ein Interface anbieten, mit dem er wie gewohnt in einer objekt-orientierten Umgebung umgehen kann. Die Tabellen, Zeilen und Spalten der Datenbank werden dabei i.d.R. durch Klassen, Objekte und Attribute repräsentiert. Aus der Perspektive des Anwendungsentwicklers unterscheiden sich diese Objekte nicht von beliebigen anderen Objekten, da die ORM-Ebene der Anwendung dafür verantwortlich ist, sie auf Datensätze der Datenbank abzubilden und z.B. die Integritätsbedingungen für relationale Datenbanken zu gewährleisten.

Unter dem Begriff Active Record versteht man auch ein Enwurfsmuster für die Architektur von Enterprise Applikationen. Martin Fowler beschreibt Active Record wie folgt:

"*An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data*. An object carries both data and behavior. Much of this data is persistent and needs to be stored in a database. Active Record uses the most obvious approach, putting data access logic in the domain object. This way all people know how to read and write their data to and from the database."[22]

Das Modul ActiveRecord orientiert sich eng am Standard ORM-Modell und am Active Record Pattern, d.h Tabellen werden auf Klassen, Zeilen auf Objekte und Spalten auf Objektattribute abgebildet. Convention over Configuration äußert sich im Zusammenhang von ActiveRecord dadurch, dass durch die Verwendung von Standardkonventionen, z.B. bei der Benennung von Tabellen und Klassen, ein Großteil der Konfiguration entfällt.

ActiveRecord verfügt über eine Reihe von Klassenmethoden, die tabellen-weite Operation ausführen und kann auch unabhängig vom Rails Framework eingesetzt werden. Das folgende Beispiel kann daher von `ruby` direkt interpretiert werden:

```
require "rubygems"
require_gem "activerecord"

ActiveRecord::Base.establish_connection(:adapter => "mysql", :host => "localhost",
:database => "books", username => "test", password => "test")

class Title < ActiveRecord::Base
end

title = Title.find(1)
title.publisher = "Addison-Wesley"
title.save
```

ActiveRecord ist sehr gut in das Framework integriert, und kann z.B. aus einem HTML-Formular modell-spezifische Daten extrahieren und Validierungen auf diesen durchführen. Auf der anderen Seite

22 Zitat aus [FOWLER]

kann ein View Fehlermeldungen mit nur einer einzigen Zeile Code aus dem Model extrahieren und darstellen.

### 2.4.1 Tabellen und Klassen

Jede Subklasse von ActiveRecord::Base kapselt eine Datenbanktabelle. Als Konvention nimmt Rails an, dass der Tabellenname immer die Pluralform des korrespondierenden Klassennamens ist (Klassenname `Title` => Tabellenname `titles`). Diese Konvention kann mit folgender Zeile in der Datei `config/environment.rb` übergangen werden:

```
ActiveRecord::Base.pluralize_table_names = false
```

Auch die direkte Abbildung von Klassenname auf Tabellenname ist veränderbar, und zwar durch die folgende Anweisung in der Klassendefinition:

```
class Person < ActiveRecord::Base
  set_table_name “persons”    # overrides default “people”
end
```

### 2.4.2 Spalten und Attribute

Objekte vom Typ ActiveRecord korrespondieren mit den Zeilen einer Datenbanktabelle. Attribute dieser Objekte korrespondieren hinsichtlich des Namens und des Datentyps mit den Spalten einer Datenbanktabelle. Über die Accessor Methoden der Attribute sind die Werte der einzelnen Zellen abfrag- und veränderbar. In Rails enthalten die Klassendefinitionen von Model-Klassen jedoch weder Attribute, noch Accessor Methoden, da ActiveRecord diese zur Laufzeit dynamisch aus dem Datenbankschema gewinnt.

#### 2.4.2.1 Zugriff auf Attribute

Attributwerte sind mittels Accessor Methoden abfrag- und veränderbar. Man verwendet hierbei die in objekt-orientierten Sprachen übliche Punkt-Notation. Auf die Attribute wird hierbei tatsächlich über Accessor Methoden zugegriffen und nicht direkt, wie es beim ersten Betrachten der Syntax den Anschein hat. Man verwendet in Rails üblicherweise eine “shortcut” Notation (s. folgende Anmerkung):

```
publisher = title.publisher          # return current value
title.publisher = “Addison-Wesley”   # set value
```

Der Rückgabewert wird von ActiveRecord in einen passenden Ruby Datentyp gecastet. Hängt man an die Getter Methode das Suffix `_before_type_cast` an, erhält man den Wert mit dem in der Datenbank verwendeten Typ.

Die Überschreibung der generierten Accessor Methoden, z.B. zur Ergänzung zusätzlichen Verhaltens, ist möglich. Das folgende Codefragment löst z.B. ein bestimmte Exception aus, falls `value` niedriger als ein (konstant) festgelegter Wert ist:

```
class Product
  MINIMUM_PRICE = 0.01
  def price=(value)
```

```
    raise PriceTooLow if value < MINIMUM_PRICE
    self[:price] = value
  end
end
```

Anm.: Rails verwendet die durch Ruby ermöglichte “shortcut” Notation. Diese erlaubt es, Methodennamen, die mit einem Gleichheitszeichen “=” enden, zu definieren. Die äquivalente Notation der Methode `price=()` wäre:

```
def set_price(value)
  # ...
end
```

... und ihre Verwendung:

```
p = Product.new
p.set_price(1.99)
```

Der Aufruf der Methode mit Gleichheitszeichen ist dagegen folgender:

```
p = Product.new
p.price=(1.99)
```

Nun erlaubt der Ruby Interpreter jedoch, dass zwischen `price` und `=` ein Leerzeichen stehen darf. Zusammen mit der Tatsache, dass die Klammerung in Ruby prinzipiell optional ist, ergibt sich somit folgende Notation:

```
p = Product.new
p.price = 1.99
```

Damit wäre geklärt, warum es in Rails den Anschein hat, man würde auf Attribute direkt, statt über Accessor Methoden, zugreifen.

### 2.4.3 Primärschlüssel und IDs

Alle Datenbanktabellen, die über eine Subklasse von ActiveRecord repräsentiert werden sollen, müssen nach Konvention einen künstlichen Primärschlüssel mit dem Namen id vom Typ Integer besitzen. Man kann diese Konvention zwar mit der folgenden Anweisung in der Klassendefinition außer Kraft setzen, übernimmt damit aber gleichzeitig die Verantwortung über die Vergabe eindeutiger Werte für den Primärschlüssel (eine Aufgabe, die andernfalls ActiveRecord übernimmt):

```
class Book < ActiveRecord::Base
  set_primary_key “isbn”
end
```

Etwas gewöhnungsbedürftig ist die Tatsache, dass der neue Primärschlüssel weiterhin über den Accessor `id()` gesetzt werden muss, obwohl der Model-Klasse der Spaltenname des tatsächlichen Primärschlüssels bekannt ist und dieser für die Abfrage von Werten sogar verwendet werden muss. Es bleibt abzuwarten, ob diese Inkonsistenz in zukünftigen Rails Versionen verbessert wird:

```
book = Book.new
book.id =”0-12345-6789”
```

```
book.title = “My Book Title”
book.save

query_book = Book.find(:first, :conditions => “isbn = ‘0-12345-6789’”)
puts query_book.attributes
```

Ergibt die Ausgabe `{ “isbn” => “0-12345-6789”, “title” => “My Book Title” }`.

### 2.4.4 Verbindung zur Datenbank

ActiveRecord abstrahiert das Konzept der Datenbankverbindung über die Delegation der Details der Verbindung an eine Menge datenbank-spezifischer Adapter, welche den Umgang mit den Verbindungsparametern für einen speziellen Datenbanktyp bzw. -hersteller übernehmen. Derzeit existieren Datenbankadapter für MySQL, Postgres, Oracle, DB2, SQLite und SQL Server.[23]

Verbindungen sind mit Model-Klassen assoziiert und jede Klasse erbt die Verbindung ihrer Superklasse. Eine Verbindung für ActiveRecord::Base (wie im einführenden Beispiel zu sehen) definiert somit die Standardverbindung aller vom Anwendungsentwickler erstellten Model-Klassen.

In Rails Anwendungen verwendet man üblicherweise jedoch nicht die Methode `establish_connection()`, sondern legt die Verbindungsparameter in der Datei `config/database.yml` ab. Dadurch werden zum einen Konfigurationsinformationen nicht mit dem Code der Anwendung vermischt, sondern stehen zentral zu Verfügung (Vgl. Dont’t Repeat Yourself), zum anderen ist `database.yml` fest in die unterschiedlichen Umgebungen für Entwicklung, Test und Produktivbetrieb integriert.[24]

### 2.4.5 Create, Retrieve, Update, Delete (CRUD)

Die Basisoperationen auf Datenbanktabellen INSERT INTO (Create), SELECT FROM (Retrieve), UPDATE (Update) und DELETE FROM (Delete) sind als Klassenmethoden `create()` bzw. `save()`, `find()`, `update()` und `delete()` in ActiveRecord::Base implementiert und stehen somit über die Vererbungshierarchie allen Model-Klassen zur Verfügung. Wir illustrieren ihre Verwendung anhand der Tabelle `titles` mit folgender SQL Definition:

```
CREATE TABLE titles (
   id int NOT NULL auto_increment,
   title varchar(100) NOT NULL,
   publisher varchar(100) NOT NULL,
   year int DEFAULT '0' NOT NULL,
   PRIMARY KEY (id)
);
```

... und der korrespondierenden Klasse `Title`:

```
class Title < ActiveRecord::Base
end
```

---

23 s. http://rubyforge.org für die Details zu Datenbankadaptern

24 s. Kapitel 2.3 Rails Configuration

Anm.: Diese ist Teil einer kleinen Anwendung zur Verwaltung von Büchern, die wir in mehreren Iterationen ausbauen werden. Die Anwendung soll den Namen `books` haben und wird (wie im Beispiel in Kapitel 1.4 Hello Rails! beschrieben) mit dem Komando `rails books` angelegt.

#### 2.4.5.1 Erzeugen neuer Datensätze (Create)

Im ORM Paradigma steht die Erzeugung neuer Tabellenzeilen (d.h. Datensätzen) in direkter Verbindung zur Erzeugung neuer Objekte. Objekte werden in Ruby mit der Methode `new()` erzeugt, anschließend können die Attribute mit Werten belegt werden, dann werden sie mit der Rails Methode `save()` in der Datenbank gespeichert:

```
title = Title.new
title.title = “Agile Web Development with Rails”
title.publisher = “The Pragmatic Programmers”
title.year = 2005
title.save
```

Die Vergabe des Primärschlüssels übernimmt in diesem Fall ActiveRecord, da in der SQL-Definition von `titles` eine Spalte `id` vom Typ `int` vorhanden ist. Der Wert lässt sich nachträglich feststellen:

```
puts “The id of this title is #{ title.id }”
```

Die Methode `create()` führt die Erzeugung, die Belegung mit Werten und die Speicherung in einem Schritt aus. Sie erwartet als Parameter ein Hash, dessen Schlüssel sog. *Keyword Parameter* für die Attribute sind:

```
Title.create(
  :title => “Agile Web Development with Rails”,
  :publisher => “The Pragmatic Programmers”,
  :year => 2005)
```

#### 2.4.5.2 Abfragen von Datensätzen (Retrieve)

Die einfachste Möglichkeit, bestimmte Datensätze abzufragen, ist, sie anhand ihres Primärschlüssels zu finden. Die Methode `find()` erwartet als Parameter einen einzelnen Wert oder eine Menge an Werten für Primärschlüssel und gibt entweder ein einzelnes Objekt oder ein Array aus Objekten vom Typ der aufrufenden Model-Klasse zurück:

```
# Find title with given id = 1
a_title = Title.find(1)

# Get a list of title ids from a html form
# then print each titles’ publisher
titles = params[:title_ids]
Title.find(titles).each { | title | p title.publisher }  # ‘p’ = ‘puts’
```

Rails enthält zwei Varianten der Methode `find()`. Die Erste nimmt als ersten Parameter das Symbol `:first`, die Zweite `:all` entgegen. `:first` gibt den ersten Datensatz, der bestimmte Bedingungen erfüllt, zurück; `:all` alle Datensätze, welche diese Bedingungen erfüllen. Alle folgenden Beispiele funktionieren sowohl mit `:first`, als auch mit `:all`.

Die Bedingungen werden durch den Parameter `:conditions` festgelegt. Dieser ist das ActiveRecord-Äquivalent zur WHERE Clause in einem SQL-Statement:

```
# query database for titles published by Addison Wesley in 2005
titles = Title.find(:all, :conditions => "publisher = 'Addison Wesley' and year =
2005")
```

`:conditions` ist auch mit Platzhaltern verwendbar, die z.B. Werte aus einem HTML Formular annehmen können. Damit ist auf einfache Weise dynamisches SQL möglich:

```
# Get a publishers' name from a html form
# then find all of its records
publ = params[:publisher_name]
titles_by_publ = Title.find(:all, :conditions => [ "publisher = ?", publ ])
```

Optionale Parameter von `find()` sind `:order` und `:limit`. Dazu ein Beispiel:

```
# Find all titles published by Addison Wesley,
# sort by year desc, then by title alphabetically
# limit result to the first 100 records found
titles = Title.find(:all, :conditions => "publisher = 'Addison Wesley'", :order =>
"year DESC, title", :limit => 100)
```

Alle Varianten von `find()` werfen die Exception `RecordNotFound`, falls kein passender Datensatz gefunden werden konnte.

#### 2.4.5.3 Aktualisieren von Datensätzen (Update)

Die Aktualisierung von Datensätzen geschieht zum einen implizit durch den Aufruf der Methode `save()` auf einem Objekt, welches einem bereits bestehenden Datensatz repräsentiert (identifiziert über den Primärschlüssel), zum anderen explizit mit den Methoden `update_attribute()`, `update_attributes()`, `update()` und `update_all()`.

Der Aufruf von `update_attribute()` bzw `update_attributes()` ändert ein bzw. mehrere Attribute und speichert den Datensatz erneut in der Tabelle:

```
title = Title.find(:first, :conditions => "title = 'Agile Web Development with
Rails'")
title.update_attributes(:title => "Agile Web Development with Rails, 2nd Edition",
year => 2006)
```

Ist der Primärschlüssel bekannt, kann die Methode `update()` verwendet werden:

```
# Update title with id = 1
Title.update(1, :title => "Agile Web Development with Rails, 2nd Edition", year =>
2006)
```

#### 2.4.5.4 Löschen von Datensätzen (Delete)

ActiveRecord unterstützt drei Arten zur Löschung von Datensätzen. Zum einen die Klassenmethoden `delete()` bzw. `delete_all()` und `destroy()` bzw. `destroy_all()`, die tabellen-weit agieren, zum anderen die Instanzmethode `destroy()`, die den korrespondierenden Datensatz zu einem *in-memory* Objekt löscht.

`delete()` erwartet als Parameter eine einzige `id` oder ein Array aus `ids`:

```
Title.delete(1)
Title.delete([1, 5, 23, 77])
```

`delete_all()` löscht alle Datensätze, die den als Parameter übergebenen Bedingungen entsprechen, oder alle Datensätze, falls die Methode ohne Parameter aufgerufen wird.

```
# Get a publishers' name from a html form
# then delete all of its records
publ = params[:publisher_name]
result = Title.delete_all([ "publisher = ?", publ ])
render(:text => result.to_s)    # renders the number of affected rows
```

Der Rückgabewert `result` hängt vom verwendeten Datenbanktyp bzw. -hersteller ab, ist jedoch i.d.R. ein Integer mit der Anzahl der gelöschten Datensätze.

Die Instanzmethode `destroy()` löscht den korrespondierenden Datensatz zu einem *in-memory* Objekt und macht dieses Objekt *immutable*, damit die Attributwerte nicht mehr verändert werden können.

Anm.: In Ruby werden unveränderliche (engl. immutable) Objekte mit der Methode `freeze()` erzeugt. Eine nachträgliche Modifikation erzeugt den Fehler `TypeError`. Dazu ein Beispiel mit einem einfachen Array:

```
a = ["a", "b", "c"]
a.freeze
a.frozen?  # => true
a << "z"   # produces: '<<': can't modify frozen array (TypeError)
```

Die Klassenmethoden `destroy()` und `destroy_all()` rufen alle Callback und Validierungs Methoden[25] von ActiveRecord auf, während `delete()` bzw. `delete_all()` diese umgehen. Um eine Datenbank also hinsichtlich der in den Model-Klassen implementierten *Business Rules* konsistent zu halten, sollte man i.d.R. `destroy()` verwenden.

### 2.4.6 Beziehungstypen zwischen Tabellen

Die meisten Anwendungen arbeiten mit mehreren Datenbanktabellen und im Normalfall werden zwischen manchen dieser Tabellen Beziehungen bestehen, z.B. wird ein Titel einen oder mehrere Autoren haben und von genau einem Verlag veröffentlicht sein. Innerhalb der Datenbank werden diese Beziehungen durch Verweise auf Basis von Primärschlüsseln zwischen den beteiligten Tabellen ausgedrückt, z.B. wird die Tabelle der Titel eine Spalte besitzen, die den Wert des Primärschlüssels des korrespondierenden Verlages aus der Tabelle der Verlage enthält. Im Kontext von Datenbanken sagt man dazu, dass der Titel einen Fremdschlüssel als Referenz auf den Verlag besitzt.

Im Rahmen einer Rails Anwendung wollen wir jedoch mit Objekten und deren Beziehungen zueinander umgehen, anstatt uns um Primär- und Fremdschlüssel auf Datenbankebene kümmern zu müssen. ActiveRecord ermöglicht es uns deshalb, fast ohne zusätzliche Konfiguration intuitive Abfragen, wie z.B.:

```
publ_name = titles.publisher.name
```

---

25 s. Kapitel 2.4.7 Validierung und Kapitel 2.4.8 Callback Methoden und Observer

... zu formulieren, statt zuerst die Beziehung mit Hilfe des Fremdschlüssels zu erfragen:

```
publ_id = titles.publisher_id
publisher = Publisher.find(publ_id)
publ_name = publisher.name
```

Um die folgenden Deklarationen erklären zu können, gehen wir von einer SQL-Definition für die Tabellen `titles`, `publishers` und `authors` aus:

```
CREATE TABLE titles (
   id int NOT NULL auto_increment,
   title varchar(100) NOT NULL,
   publisher_id int NOT NULL,
   year int DEFAULT '0' NOT NULL,
   constraint fk_titles_publs foreign key (publisher_id) references publishers(id),
   PRIMARY KEY (id)
);

CREATE TABLE publishers (
   id int NOT NULL auto_increment,
   name varchar(60) NOT NULL,
   PRIMARY KEY (id)
);

CREATE TABLE authors (
   id int NOT NULL auto_increment,
   name varchar(60) NOT NULL,
   PRIMARY KEY (id)
);

CREATE TABLE authors_titles (
   author_id int NOT NULL,
   title_id int NOT NULL,
   constraint fk_at_authors foreign key (author_id) references authors(id),
   constraint fk_at_titles foreign key (title_id) references titles(id),
   PRIMARY KEY (author_id, title_id)
);
```

Zusätzlich zu der Namens-Korrespondenz zwischen Klassen und Datenbanktabellen sehen wir hier eine weitere Rails Konvention in diesem Zusammenhang. Die Spalte des Fremdschlüssels wird nach dem Namen der Klasse der Zieltabelle, ergänzt um das Suffix `_id`, benannt (z.B. Name des Fremdschlüssels `publisher_id` => Tabelle `publishers` => Klasse `Publisher`).

Anm.: Rails benötigt nur die Namens-Konvention und nicht die Angabe der *foreign key constraints*, um die Beziehungstypen aus den Tabellendefinitionen zu gewinnen.

#### 2.4.6.1 Beziehungstypen definieren

ActiveRecord unterstützt drei Arten von Beziehungen zwischen Tabellen: `1:1`, `1:n` und `m:n`, oder in englischer Sprache, *one-to-one*, *one-to-many* und *many-to-many*.[26] In den Model-Klassen verwendet man hierfür die Deklarationen `has_one`, `has_many`, `belongs_to` und `has_and_belongs_to_many`.

[26] aus den englischen Bezeichnungen wird der Bezug zu den im folgenden verwendeten Deklarationen deutlicher.

Verlage und Titel haben eine *one-to-many*-Beziehung. Eine beliebige Anzahl an Titeln kann von einem bestimmten Verlag veröffentlicht werden. In Rails erstellen wir dafür die folgenden Klassendefinitionen:

```
class Publisher < ActiveRecord::Base
  has_many :titles
end

class Title < ActiveRecord::Base
  belongs_to :publisher
end
```

Ein einziger Titel hat entweder einen oder mehrere Autoren und ein Autor kann der Verfasser eines oder mehrerer Titel sein. Dies ist ein Beispiel für eine *many-to-many*-Beziehung. In Rails definiert man diese wie folgt:

```
class Title < ActiveRecord::Base
  # ...
  has_and_belongs_to_many :authors
end

class Author < ActiveRecord::Base
  has_and_belongs_to_many :titles
end
```

Die Deklarationen fügen zusätzlich eine Menge an Methoden, welche die Navigationsmöglichkeiten zwischen den beteiligten Model-Objekten ermöglichen, zu den Klassendefinitionen hinzu.[27]

#### 2.4.6.2 *One-to-many*-Beziehungen

Dieser Beziehungstyp wird über die Deklarationen `has_many` und `belongs_to` definiert. Die Deklaration `belongs_to` muss dabei in der Klasse für das Model, das den Fremdschlüssel enthält, definiert sein (im Beispiel `Title`).

#### Die *belongs_to*-Deklaration

`belongs_to` definiert, dass eine Klasse ein *Parent*-Beziehung zu der Klasse, welche die Deklaration enthält, hat. Diese Formulierung ist ungewöhnlich, aber die Konvention ist, dass die Tabelle, welche den Fremdschlüssel enthält, zu der Tabelle, die damit referenziert wird, gehört (engl. *belongs to*).

Die Deklaration erzeugt automatisch eine Menge an Instanzmethoden, um die Verbindung zu verwalten. Alle Namen dieser Instanzmethoden beginnen mit dem Namen der Verbindung (z.B. `publisher`). Im folgenden Beispiel verwenden wir die Methoden `publisher()` und `publisher=()`[28], um Zugriff auf das Publisher-Objekt zu haben und es zu aktualisieren. ActiveRecord hält die Datenbank dabei automatisch konsistent. Der neue Verlag wird mit der Aktualisierung des Titels gespeichert und der Wert des Fremdschlüssels wird angepasst:

```
title = Title.find(1)
puts “Current publisher for ‘#{title.title}’ is #{title.publisher.name} with id
     #{title.publisher.id}”
title.publisher = Publisher.new(:name => “Addison Wesley”)
title.save!
```

---

[27] s. dazu das einführende Beispiel.

[28] die verwendete Notation erläuterten wir in Kapitel 2.4.2.1 Zugriff auf Attribute.

```
puts “New publisher for ‘#{title.title}’ is #{title.publisher.name} with id
     #{title.publisher.id}”
```

Das Code-Beispiel könnte die folgende Ausgabe erzeugen:

```
Current publisher for ‘Contributing to Eclipse’ is Prentice Hall with id 1
New publisher for ‘Contributing to Eclipse’ is Addison Wesley with id 2
```

#### Die *has_many*-Deklaration

`has_many` definiert ein Attribut (im Beispiel `titles`) als ein Array aus *Child*-Objekten. Über dieses Array kann man iterieren, um z.B. ein bestimmtes Objekt zu finden oder, wie im folgenden Beispiel, bestimmte Attribute von allen enthaltenen Datensätze auszugeben:

```
publisher = Publisher.find(1)
puts “Titles published by #{publisher.name}:”
publisher.titles.each do | t |
  puts “#{t.title} was published in #{t.year}”
end
```

Auch `has_many` fügt noch weitere Methoden zu der die Deklaration enthaltenden Host-Klasse hinzu, die ebenfalls alle mit dem Namen der Verbindung (z.B. `titles`) beginnen. Stellvertretend sei `titles.find()` genannt, die einen regulären Aufruf der Methode `find()`[29] auslöst, deren Ergebnisse jedoch auf die Titel beschränkt sind, die mit dem korrespondierenden Verlag assoziiert sind. Eine detaillierte Beschreibung findet sich in der Rails Framework Dokumentation für das Modul ActiveRecord::Associations::ClassMethods.

### 2.4.6.3 *Many-to-many*-Beziehungen

*Many-to-many*-Beziehungen sind symmetrisch aufgebaut; beide beteiligte Klassen definieren ihre Verbindung mit der Deklaration `has_and_belongs_to_many`. Innerhalb der Datenbank ist diese Art der Beziehung durch eine sog. Join-Tabelle repräsentiert (im Beispiel die Tabelle `authors_titles`). Nach Konvention nimmt ActiveRecord an, das der Tabellenname aus dem Zusammenschluss der Namen beider beteiligten Tabellen (im Beispiel `authors` und `titles`), in alphabetischer Reihenfolge, gebildet wird. Die Join-Tabelle besitzt als Primärschlüssel die Kombination beider beteiligten Fremdschlüssel, deren Namen wiederum den eingangs erwähnten Konventionen zur Bildung der Namen für Fremdschlüssel genügen.

#### Die *has_and_belongs_to_many*-Deklaration

`has_and_belongs_to_many` (für die folgenden Erläuterungen mit `habtm` abgekürzt) verhält sich in vielerlei Hinsicht wie `has_many`. `hatbm` definiert je ein Attribut (im Beispiel `authors` und `titles`) als Array aus korrespondierenden Objekten und unterstützt ebenfalls ähnliche Methoden. Um z.B. zusätzlich alle Autoren für die oben formulierte Abfrage auszugeben, genügt die folgende Ergänzung:

```
publisher = Publisher.find(1)
puts “Titles published by #{publisher.name}:”
publisher.titles.each do | title |
  puts “#{title.title} was published in #{title.year}”
  puts “and was written by “
  title.authors.each do | author |
    puts “#{author.name}”
  end
end
```

---

29 s. Kapitel 2.4.5.2 Abfragen von Datensätzen

Als Einführung in die möglichen Beziehungen zwischen Klassen und Tabellen und vor allem die daraus resultierenden Methoden sollen diese Erläuterungen genügen. Da die Methoden dynamisch aus den Namen der Beziehungen generiert werden, müssen sie je nach Anwendung einzeln betrachtet und verstanden werden. Ein geeigneter Einstiegspunkt ist die Dokumentation für das Modul ActiveRecord::Associations::ClassMethods.

### 2.4.7 Validierung

ActiveRecord kann die Inhalte von Model-Objekten automatisch validieren[30], z.B. bevor ein Objekt in der Datenbank gespeichert wird. Der Anwendungsentwickler kann dazu zum einen die Methoden `validate()`, `validate_on_create()` und/oder `validate_on_update()` aus dem Modul ActiveRecord::Validations implementieren (d.h. überschreiben), zum anderen die vielfältigen Validation Helper Methoden benutzen, die ihm das Framework durch das Modul ActiveRecord::Validations::ClassMethods zur Verfügung stellt.

Er kann den Zustand des Objekts jedoch auch jederzeit programmatisch überprüfen; dazu verwendet er die Methode `valid?()`, welche die beiden Methoden, die normalerweise durch ein `save()` Operation ausgelöst werden, unabhängig von dieser triggert.

Die Methode `validate()` wird vor jeder `save()` Operation ausgelöst (INSERT INTO und UPDATE), `validate_on_create()` nur vor dem Einfügen eines neuen Datensatzes (INSERT INTO) und `validate_on_update()` nur vor der Aktualisierung eines bestehenden Datensatzes (UPDATE). Man muss dies bei der Prüfung der Eindeutigkeit eines bestimmten Wertes berücksichtigen, wie z.B. im Beispiel zu Kapitel 2.4.3 Primärschlüssel und IDs die Prüfung von ISBN, falls man den künstlichen Primärschlüssel `id` beibehält. Verwendet man `validate()` schlägt die Prüfung fehl, da `save()` auch bei der Aktualisierung verwendet wird und die ISBN in diesem Fall natürlich schon in der Datenbank vorhanden ist.[31] Richtig ist dagegen der Einsatz von `validate_on_create()`:

```
class Book < ActiveRecord::Base
  # ...
  def validate_on_create
    if self.find_by_isbn(isbn)
      errors.add(:isbn, “is already being used!”)
    end
  end
end
```

Falls bei der Überprüfung ein Problem auftritt fügen alle Varianten von `validate()`, einschließlich der Validation Helper, mittels der Methode `add()` eine Nachricht zur “Fehlerliste” `errors` des korrespondierenden Model-Objekts hinzu. Wie wir im Kapitel 2.6.5 Fehlerbehandlung für Model-Objekte sehen werden, kann man diese Liste in den Views äußerst komfortabel verwenden, um z.B. falsche oder fehlende Eingaben in einem HTML Formular darzustellen. Der zweite Parameter der Methode `add()` repräsentiert die Fehlermeldung, die dabei für das entsprechende Attribut (erster Parameter von `add()`) ausgegeben wird.

---

30 Validierung = Gültigkeitsprüfung

31 s. Kapitel 2.4.5.3 Aktualisieren von Datensätzen

Im Anwendungscode selbst hat man mit dem Aufruf `errors.on()` Zugriff auf die Fehlerliste (z.B. in der Form `errors.on(:isbn)`). `errors.clear()` löscht den Inhalt der Liste. Für die vollständige Klassendokumentation verweisen wir auf ActiveRecord::Errors in der Rails Framework Dokumentation.

#### 2.4.7.1 Validation Helper

Manche Validierungen treten so häufig auf, dass Rails für sie vorgefertigte Hilfsmethoden anbietet. Man nennt diese Klassenmethoden Validation Helper. Sie beginnen alle mit dem Präfix `validates_` und nehmen als Parameter eine Liste der zu überprüfenden Attribute, sowie ein optionales Dictionary aus key => value Paaren zur weiteren Konfiguration der Methode entgegen. Die Auflistung aller Methoden findet sich in der Dokumentation zu ActiveRecord::Validations::ClassMethods. Stellvertretend seien `validates_format_of()`, `validates_numericality_of()` und `validates_presence_of()` genannt. Auch für das obige Beispiel existiert ein Validtion Helper, `validates_uniqueness_of()`. Der Code für dieselbe Funktionalität wie in der Methode `validate_on_create()` lautet dann:

```
class Book < ActiveRecord::Base
  validates_uniqueness_of(:isbn,
                          :on => :create,
                          :message => “is already being used”)
  # ...
end
```

Die Option `:on` bestimmt, wann die Überprüfung stattfinden soll und akzeptiert die Werte `:save`, `:create` und `:update`. Die Option `:message` legt die Fehlermeldung fest und überschreibt die Standardfehlermeldung der Validation Helper.

Anm.: Weitere Information zu Helper Methoden in Rails finden sich im Kapitel 2.6.2 Helper.

### 2.4.8 Callback Methoden und Observer

#### 2.4.8.1 Callback Methoden

ActiveRecord kontrolliert und überwacht den Lebenszyklus, d.h. den Zustand und die Zustandsänderungen von Model-Objekten - ActiveRecord erstellt, speichert, findet, aktualisiert und löscht sie (Create, Retrieve, Update, Delete). Unter Einsatz von Callback Methoden hat der Anwendungsentwickler die Möglichkeit, in diesen Prozess aktiv einzugreifen, indem er Code schreibt, der zu bestimmten Ereignissen im Lebenszyklus eines Objekts aufgerufen wird. Die Callback Methoden auf Model-Objekten sind vergleichbar mit Datenbank-Triggern, die in Rails zwar außerhalb der Datenbank gehalten werden, aber im Prinzip dieselbe Funktionalität bieten.

Analog zu den SQL Operationen INSERT INTO, UPDATE und DELETE gibt es Callback Methoden für `create()`/`save()`, `update()`/`save()` und `destroy()`, jeweils unterteilt in verschiedene `before_` und `after_` Varianten. Zusätzlich stehen mit `after_find()` (analog zu SELECT) und `after_initialize()` zwei besondere Callback Methoden ohne `before_` Pendant zur Verfügung; insgesamt definiert ActiveRecord 20 unterschiedliche Callbacks.

Grundsätzlich erlaubt Rails die Implementierung von Callbacks auf zwei verschiedene Arten. Zum einen die direkte Implementierung einer Callback Methode als Instanzmethode einer Subklasse von ActiveRecord::Base (Identifizierung durch den Methodennamen), zum anderen durch die Übergabe

entweder einer Referenz auf eine beliebige Methode (per Symbol) oder eines Blocks an die Klassenmethoden aus ActiveRecord::Callbacks.[32]

Im Falle der direkten Implementierung schreibt der Anwendungsentwickler den Code, der vor oder nach der Änderung des Objektzustands ausgeführt werden soll, in den Methodenrumpf der Callback Methode. Alle neu erzeugten und gespeicherten Datensätze können z.B. mit folgendem Code geloggt werden:

```
class Title < ActiveRecord::Base
  # ...
  def after_create
    logger.info "Title #{ self.id } created"
  end
end
```

Der zweite Fall, d.h. die Registrierung der Callbacks durch die Macros in ActiveRecord::Callbacks, bietet den Hauptvorteil, dass das zur Ausführungskette hinzugefügte Verhalten in der Vererbungshierarchie der Model-Klassen erhalten bleibt.[33] Nehmen wir folgende Klassenhierarchie als Beispiel:

```
class A < ActiveRecord::Base
  before_destroy :method_x
end

class B < A
  before_destroy :method_y
end
```

Beim Aufruf von A#destroy wird nur `method_x()` ausgelöst, dagegen wird beim Aufruf von B#destroy sowohl `method_x()` als auch `method_y()` automatisch ausgelöst. Wären beide als überschriebene Methoden implementiert (Fall 1), könnte die Klasse B mit dem Schlüsselwort `super` entscheiden, ob die geerbte Callback Methode getriggert werden soll oder nicht.

Es besteht auch die Möglichkeit, die Callback Methoden in eine eigene Klasse auszulagern, um sie wiederverwendbar zu machen. Die Datei mit der Klassendefinition muss sich in diesem Fall unter `app/models` befinden. Die Klassenmethoden aus ActiveRecord::Callbacks erhalten dann, statt einer Referenz auf eine Methode, eine Referenz auf ein neu erzeugtes Objekt der Klasse als Parameter. Das folgende Beispiel protokolliert das Datum der Erzeugung und jeder Änderung für einen Titel-Datensatz (Die Tabelle `titles` sei hierzu um zwei Spalten vom Typ `datetime` ergänzt):

```
class Title < ActiveRecord::Base
  before_create TimestampCallbacks.new
  before_update TimestampCallbacks.new
end

# in app/models/timestamp_callbacks.rb
class TimestampCallbacks
  def before_create(model)
    model.created_at ||= Time.now
  end
  def before_update(model)
```

32 s. before und after Filter in 2.5.5 Filter und Verifikation

33 Single Table Inheritance ist mit ActiveRecord möglich und wird unterstützt. Wir werden dieses Thema im Rahmen der Ausarbeitung jedoch nicht näher betrachten.

```
    model.updated_at ||= Time.now
  end
end
```

Anm.: Wir benutzen hier Rubys bedingten Zuweisungs Operator (engl. conditional assignment operator) "`||=`". Eine Erläuterung dazu findet sich im Kapitel 2.5.4.2.

Der Code lässt sich weiter generalisieren, indem die Information über den Spaltennamen dynamisch gesetzt wird. Die einzige Voraussetzung zur Verwendung der Klasse ist dann, dass die korrespondierenden Spalten von Typ `datetime` sind, was sich jedoch gut in der Klassendokumentation festhalten lässt (und zusätzlich durch eine Exception abgefangen werden kann, die mit der Methode `columns_hash()` den Datentyp prüft):

```
# in app/models/timestamp_callbacks.rb
class TimestampCallbacks
  def initialize(field)
    @curr_field = field
  end
  def before_create(model)
    model[@curr_field] ||= Time.now
  end
  def before_update(model)
    model[@curr_field] ||= Time.now
  end
end
```

In der Klasse `Title` wird das gewünschte Attribut dann bei der Erzeugung angegeben:

```
class Title < ActiveRecord::Base
  before_create TimestampCallbacks.new(:created_at)
  before_update TimestampCallbacks.new(:updated_at)
end
```

#### 2.4.8.2 Observer

Callbacks sind zwar ein mächtiges Werkzeug, führen aber manchmal dazu, dass eine Model-Klasse Aufgaben erledigt, für die sie aus Sicht des Anwendungsdesigns nicht verantwortlich ist.

Zu diesen Aufgaben gehört z.B. Logging, wie wir es im ersten Beispiel des Kapitels 2.4.8.1 Callback Methoden eingesetzt haben. Ein ActiveRecord Observer (Klasse ActiveRecord::Observer) vermeidet diese Vermischung der Verantwortlichkeiten, indem er sich für die Callback Methoden der Model-Klasse registriert, ohne jedoch eine Änderung in ihr selbst zu verursachen (der Observer ist transparent). Observer werden standardmäßig auf die Model-Klasse abgebildet, mit der sie den Namen teilen und werden unter `app/models` gespeichert. Im Widerspruch zu der in Rails üblichen "Autokonfiguration" müssen Observer in der Datei `config/environment.rb` aktiviert werden, da sie sonst nicht aufgerufen werden:

```
config.active_record.observers = :title_observer  # append more observers with ","
```

Anm.: Alternativ kann man auch die Deklaration `observer` in einer Controller-Klasse verwenden, um einen Observer zu aktivieren.[34]

---

34 s. Kapitel 2.5 ActionController

Um einen Observer für das Logging-Beispiel zu verwenden, entfernen wir zuerst die Methode `after_create()` aus der Klasse `Title` und definieren dann die Observer-Klasse in der Datei `app/models/title_observer.rb`:

```
class TitleObserver << ActiveRecord::Observer
  def after_create(title)
       title.logger.info "Title #{ title.id } created"
  end
end
```

## 2.5 ActionController

ActionController bildet zusammen mit ActionView das Modul ActionPack. ActionPack ist dafür verantwortlich, eintreffende Requests zu verarbeiten und ausgehende Responses zu erzeugen. Zusammen mit ActiveRecord bildet ActionPack die Basis jeder Rails Anwendung. Übertragen auf das MVC Paradigma repräsentiert ActionController - wie der Name bereits andeutet - den Controller.

Ein Großteil der Konfiguration für das Zusammenspiel der Module ActiveRecord, ActionController und ActiveRecord wird von Rails automatisch gehandhabt (Vgl. Convention over Configuration). Wenn z.B. ein Request über die URL http://0.0.0.0:3000/HelloRails/hello eintrifft, macht Rails folgendes:

- Die Datei `hello_rails_controller.rb` aus dem Verzeichnis `app/controllers` laden. Dieser Ladevorgang findet in der Production Umgebung einmalig, in der Development Umgebung dagegen für jeden Request an den Controller statt, um Änderungen im Quell-Code ohne Verzögerungen wiederzugeben.
- Ein Objekt der Klasse `HelloRailsController` erzeugen
- Prüfen, ob die Datei `hello_rails_helper.rb` im Verzeichnis `app/helpers` existiert. Falls ja, diese Datei laden und als Modul `HelloRailsHelper` in das Controller-Objekt einbinden.
- Prüfen, ob die Datei `HelloRails.rb` im Verzeichnis `app/models` existiert. Falls ja, diese Datei ebenfalls laden und ein Exemplar der Klasse `HelloRails` erzeugen.

Gelegentlich muss man dieses Standardverhalten erweitern, z.B. wenn Klassen in Sessions verwendet werden. Denkbar ist auch, ein Helper Modul in unterschiedlichen Controller-Klassen zu verwenden oder Model-Klassen im voraus zu laden. In der Controller-Klasse stehen dem Anwendungsentwickler dazu verschiedene. von ActionController geerbte Deklarationen zur Verfügung:

```
class BookadminController < ApplicationController
  model :titles, :authors, :publishers
  helper :date_format    # includes app/helpers/date_format_helper.rb
  ...
end
```

Anm.: Nach demselben Prinzip können mit der Deklaration `observer` auch ActiveRecord Observer eingebunden werden (z.B. mit `observer :title_observer`).

### 2.5.1 Routing Requests

Wie bereits im Verlauf des Kapitels 1.4 Hello Rails! erwähnt, ist die Information darüber, welcher Code der Anwendung einen bestimmten Request verarbeiten soll in der URL selbst enthalten und wird mit Hilfe des routing Subsystems decodiert. Dabei identifiziert routing den Controller, die auszuführende Action Methode, sowie den an diese Methode zu übergebenden Parameter aus der gegebenen URL.

Ein Request einer kleinen Bibliotheksverwaltung kann z.B. die Form http://0.0.0.0:3000/bookadmin/list_titles_by_publisher/1 haben. Die Anwendung interpretiert dies als Aufruf der Action Methode `list_titles_by_publisher()` der Klasse `BookadminController`. Der Methode wird als Parameter die `1` übergeben, was bedeutet, alle Titel des Herausgebers mit der `id=1` aufzulisten. Über die Datenstruktur `params` hat die Action Methode Zugriff auf die `id`:

```
def list_titles_by_publisher
  publ_id = params[:id]
  @titles = Title.find(:all, :conditions => [”publisher_id = ?, publ_id])
end
```

Sobald der Controller identifiziert ist, wird eine neues Objekt erzeugt und auf diesem die `process()` Methode ausgeführt, welche über die Details des Requests sowie ein Response-Objekt verfügt. Dann ruft der Controller die Action Methode auf, welche den Request bearbeitet soll. Falls die Action Methode keinen expliziten Aufruf an die Methode `render()` enthält, versucht der Controller ein View-Template mit gleichem Namen wie die Action Methode zu rendern. Falls keine Action Methode mit entsprechendem Namen gefunden wurde, rendert der Controller dieses Template umgehend - man benötigt also nicht unbedingt eine Action, um ein Template zu rendern.

### 2.5.2 Action Methoden

Das Controller-Objekt ruft für den zu bearbeitenden Request i.d.R. eine Action Methode (im folgenden auch einfach Action genannt) auf. Als auszuführende Actions kommen alle öffentlichen (engl. public) Instanzmethoden der Controller-Klasse in Frage. Falls keine Methode für die Action gefunden wird, die Klasse jedoch die Methode `method_missing()` implementiert, wird diese aufgerufen. Als Parameter erhält sie den Namen der ursprünglich aufgerufenen Action. Wenn beide Fälle nicht zutreffen, versucht die Controller-Klasse ein nach der Action benanntes View-Template zu rendern. Bleibt auch das ohne Erfolg, wird die Exception `UnknownAction` geworfen (Vgl. Abb. 5).


*Abb. 5: "Action Controller: Exception caught"*

Anm.: In Kapitel 1.4 Hello Rails! könnte also die leere Action `hello()` auch weggelassen werden, da ein benanntes View-Template `hello.rhtml` vorhanden ist (in der zweiten Iteration brauchen wir `hello()` jedoch für die Initialisierung der Instanzvariablen `@time`).

#### 2.5.2.1 Das Controller Environment

Der Controller legt ein Environment für "seine" Actions und - wie wir später sehen werden - auch für die durch die Actions aufgerufenen View-Templates an. Dabei handelt es sich um verschiedene Objekte, die über Attribute des Controller-Exemplars referenziert werden; ihre Eigenschaften können mittels Accessor Methoden abgefragt werden. Alle Environment Objekte sind in ActionController::Base definiert, wie z.B.:

`request`
Repräsentiert den eintreffenden Request; sobald eine Action identifiziert worden ist, stehen die verbleibenden Request Parameter, die Session und das vollständige Request-Objekt `request` der Action über Instanzvariablen zur Verfügung. `request` wird hauptsächlich zur Abfrage von HTTP Header Informationen verwendet. Diese Abfragen erfolgen über den Zugriff auf das Dictionary `env`, z.B. durch:

```
language = request.env["HTTP_ACCEPT_LANGUAGE"]
host = request.env["REMOTE_HOST"]
ip = request.env["REMOTE_IP"]
```

`request` ist ein Objekt der Klasse ActionController::AbstractRequest; weiter nützliche Methoden sind u.a.

- `method()` liefert den Typ des Requests (`:get`, `:post`)
- `get?()` und `post?()` liefern je nach Typ des Requests `true` oder `false`

`params`
Eine Datenstruktur vom Typ Dictionary, die alle Request Parameter, egal ob vom Typ `GET`, `POST` oder aus der URL enthält. Beispiele: Für eine Action der Form /bookadmin/list?category=all&limit=20 enthält `params { :category => "all", :limit => 5 }`. Für den durch das folgende HTML Formular ausgelösten Request:

```
<%= form_tag %>
  User name: <%= text_field('user', 'name', 'size' => 40) %>
  Email address: <%= text_field('user', 'email', 'size' => 40) %>
  <%= submit_tag(' LOGIN ')
<%= end_form_tag %>
```

... enthält `params` z.B. `{ :user => { :name => "Andi", :email => "andi.retter@web.de" }}`. Werte werden über den Schlüssel, der entweder als Ruby Symbol oder String übergeben werden kann, abgefragt.

`cookies`
Die Cookies, die dem Request zugeordnet sind. Das Setzen von Werten in `cookies` veranlasst automatisch, dass es mit der nächsten Response an den Browser gesendet wird. `cookies` ist ein Objekt der Klasse ActionController::Cookies.[35]

`session`
Ebenfalls eine Datenstruktur vom Typ Dictionary, welche die Session-Daten enthält. Jeder beliebige Objekttyp kann in ihr gespeichert werden, vorausgesetzt das Objekt ist *marshall*-bar (d.h. serialisierbar). Ein Objekt wird zu `session` durch einen eindeutigen Schlüssel hinzugefügt:

```
session[:list] = List.new
```

... und wieder gefunden:

```
List l = session[:list]
```

Um Objekte aus `session` zu entfernen genügt die Zuweisung von `nil` zu einem bestimmten Schlüssel:

```
session[:list] = nil
```

Die Löschung aller Objekte in `session` und die Erzeugung eines neuen Session-Exemplars geschieht durch die Methode `reset_session()` aus ActionController::Base. Ein ausführlicheres Beispiel über die Verwendung und den Umgang mit Session in Rails folgt in Kapitel 2.5.4 Sessions.

---

35 s. Kapitel 2.5.3 Cookies

#### 2.5.2.2 Dem Benutzer antworten

Neben der Bearbeitung eintreffender Requests und der Interaktion mit dem Model ist der Controller auch dafür verantwortlich, dem Benutzer zu antworten (d.h. die Interaktion zwischen Benutzer und Anwendung zu koordinieren).[36] Dabei stehen ihm prinzipiell drei Möglichkeiten zur Verfügung:

- ein Template (d.h. den View im MVC Paradigma) rendern
- einen String an den Browser übergeben (ohne "Umweg" über einen View)
- beliebige Daten an dem Browser senden (z.B. ein PDF Dokument)

Der Controller antwortet dem Benutzer pro Request genau einmal (Request/Response Paradigma). Das bedeutet, dass man während der Verarbeitung eines Requests die Methoden `render()` aus ActionController::Base bzw. `send_data()` oder `send_file()` aus ActionController::Streaming nur einmal explizit aufrufen darf, da man sonst die Exception `DoubleRenderError` erhält. Weil der Controller andererseits aber auch genau einmal pro Request antworten muss, prüft er am Ende der Verarbeitung, ob schon eine Antwort erzeugt wurde. Ist dies nicht der Fall, rendert er - wie oben bereits erwähnt - ein nach der Action benanntes Template automatisch (Anm.: I.d.R. arbeitet man mit diesem "impliziten" Rendering-Automatismus).

##### Templates rendern

Ein Template legt den Inhalt - im Sinne von Texten, Bildern, usw. - für eine Antwort an den Benutzer fest. Rails unterstützt zwei Formate für Templates: RHTML und Builder. RHTML ist HTML mit eingebettetem Ruby Code, Builder benutzen eine XML-artige Syntax.

Nach Konvention befindet sich das Template für die Action `action` des Controllers `controller` in der Datei `app/views/controller/action.rhtml` (oder `action.rxml` für Builder Templates). Der Standardanteil des Pfades `app/views` kann anwendungsweit mit der Konfigurationsoption `template_root` in der Anweisung:

```
ActionController::Base.template_root = path/to/template_directory
```

... überschrieben werden. Unterhalb dieses Pfades ist die Konvention, für jeden Controller ein separates und nach dem Controller benanntes Verzeichnis mit den Views des Controllers zu haben.

`render()` ist die zentrale Methode beim Rendering von Templates. Sie nimmt Optionen in einem Dictionary aus `key => value` Paaren entgegen, mit denen spezifiziert wird, was gerendert werden soll und wie es gerendert werden soll. Einige dieser Optionen bei der Verwendung im Controller sind z.B.

`render(:text => string)`
Übergibt den String `string` als *Plain Text*, d.h. ohne Einbeziehung eines Views, direkt an den Browser, z.B.:

```
def hello
  render(:text => "Hello Rails!")
end
```

`render(:action => action_name)`
Rendert das korrespondierende Template `action_name.rhtml` unter `app/views/controller_name` zur Action `action_name` dieses Controllers, z.B.:

---

36 s. Kapitel 1.2.3.2 MVC 2 oder Model 2 Architekturen

```
def display_list
  if @list.empty?
    render(:action => “index”)
  else
    render(:action => “display_list”)
  end
end
```

Anm: In manchen Fällen ist zu beachten, ob anstatt `render(:action)` nicht `redirect_to(:action)` verwendet werden muss.[37]

`render(:file => path, [:use_full_path => true | false])`
Rendert das durch `path` spezifizierte Template. Falls die Option `:use_full_path` gleich `true` ist, wird der Standardanteil des Pfades (nach Konvention `app/views`) vor `path` eingefügt. Wir verwenden `render(:file)` in Zusammenhang mit Builder Templates im Kapitel 2.6.1.2.1 Builder Templates.

`render(:partial => name, options)`
Rendert ein sog. Partial Page Template.[38]

`render()`
Ohne weiteren Parameter rendert `render()` das Standard Template zur korrespondierenden Action im aktuellen Controller. Das folgende Codefragment rendert also das Template `app/views/hello_rails/ hello.rhtml` :

```
class HelloRailsController < ApplicationController
  def hello
    render
  end
end
```

Wie oben bereits erwähnt, prüft Rails am Ende jeder Action Methode, ob ein expliziter Aufruf an `render()` erfolgt. Falls dies nicht der Fall ist, wird das zur Action korrespondierende Template automatisch gerendert (”implizites” Rendering); vereinfacht lautet der Code daher:

```
class HelloRailsController < ApplicationController
  def hello
  end
end
```

Zusätzlich ist es nicht notwendig, explizit eine Action Methode zu definieren, wenn diese - wie im Beispiel `hello()` - keine weitere Aufgabe zu erfüllen hat. Der Controller kann anhand den Informationen aus der Request-URL das Template auch automatisch identifizieren; die größtmögliche Vereinfachung ist daher:

```
class HelloRailsController < ApplicationController
end
```

Alle Varianten von `render()` nehmen optional die Parameter `:status` und/oder `:layout` entgegen. `:status` setzt den Status Header der HTTP Response. Dieser ist standardmäßig konstant auf “`200 OK`” gesetzt und muss i.d.R. nicht explizit gesetzt werden.

---

37 s. Kapitel 2.5.2.3 Redirects

38 s. Kapitel 2.6.6.1 Partial Page Templates

Anm.: Der Standard-Statusheader als Konstante ist in ActionController::Base `DEFAULT_RENDER_STATUS_CODE = "200 OK"` definiert.

Der :layout Parameter legt fest, ob die Ausgabe des Templates in ein Rails Layout verpackt wird oder nicht. Ist er auf `true` oder `nil` gesetzt (`nil` bedeutet hier, dass er nicht angegeben ist), wird ein Layout angewandt, jedoch nur, falls es mit der aktuellen Action assoziiert ist. Da dies jedoch i.d.R. der Fall ist (Vgl. Standardkonventionen für Controller), muss er z.B. bei Verwendung der AJAX-Technologie explizit auf `false` gesetzt werden, da hier ja nur ein Teil der HTML Seite aktualisiert wird. Weitere Informationen zu `render(:layout)` finden sich im Kapitel 2.6.6.3 Layouts und im Kapitel 3.2 AJAX on Rails.

#### 2.5.2.3 Redirects

Ein HTTP Redirect wird vom Server zum Client als Antwort auf einen Request gesendet. Die Antwort enthält die neue URL für den Client und einige Zusatzinformationen, z.B. ob diese Umleitung dauerhaft (Statuscode 301) oder temporär (Statuscode 307) ist. Nötig sind Redirects z.B. nach der Reorganisation einer Website, falls ein Besucher über eine veraltete URL auf der Site eintrifft.

Aus Sicht der Anwendung macht ein Redirect jedoch nichts anderes als ein Besucher, wenn dieser eine neue URL in den Browser eintippt oder einen Hyperlink auf der Seite anklickt. Genau deshalb kann er für Web-Anwendungen nützlich sein.

Bsp.: Im später folgenden Sessions Beispiel definieren wie eine Methode `add_to_list()`:

```
def add_to_list
  title = Title.find(params[:id])
  @list = find_list
  @list.add_title(title)
  redirect_to(:action => "display_list")
end
```

Die Absicht dabei ist klar: Nach dem hinzufügen des Titels zur Liste soll diese mit `redirect_to(:action => "display_list")` angezeigt werden. Es wäre zwar auch der Aufruf `render(:action => "display_list")` möglich, aber aus Sicht des Browsers wäre die URL auch danach noch `/bookadmin/add_to_list/some_id`. Aktualisiert der Benutzer - aus welchen Gründen auch immer - die Seite über den Refresh Button des Browsers, löst dies einen weiteren Request für die Methode `add_to_list()` aus, und derselbe Titel würde erneut zur Liste hinzugefügt werden.

`redirect_to()` leitet zu einer Action eines beliebigen Controllers um, oder nimmt eine relative oder absolute Pfadangabe entgegen. In der `:action` Variante wird die resultierende URL mit der Methode `url_for()` erzeugt und kann somit alle Vorteile von Rails routing nutzen.

Anm.: Mit `url_for()` können URLs in beliebiger Form erzeugt werden. Die im Rahmen dieser Ausarbeitung verwendete Form /controller/action/id ist die von Rails verwendete Standardvariante. `url_for()` ist in ActionController::Base dokumentiert. Anpassungen bzw. Änderungen am anwendungsweit geltenden routing Algorithmus werden in der Datei `config/routes.rb` vorgenommen.

### 2.5.3 Cookies

Cookies sind Zeichenfolgen, die vom Web Server per HTTP an den Browser gesendet und dort abgelegt werden. Sie können vom betreffenden Web Server auch wieder angefordert werden. Dadurch kann eine Zustandsspeicherung des Dialogs zwischen Browser und Server erfolgen.[39]

Rails abstrahiert Cookies durch das einfach zu bedienende Interface der Datenstruktur `cookies` aus dem Controller Environment. Beim Empfang des Requests wird `cookies` mit den Namen der vom Browser gesendeten Cookies und ihren korrespondierenden Werten initialisiert. Die Anwendung kann jederzeit neue `key => value` Paare zu `cookies` hinzufügen, die dann mit der nächsten Response automatisch an den Browser zurück gesendet werden.

Als Beispiel betrachten wir einen Request der URL http://0.0.0.0:3000/CookiesController/action_one. Die Action Methode `action_one()` initialisiert `cookies`, erzeugt ein Cookie, indem sie einen Wert unter dem Schlüssel `the_time` ablegt und leitet anschließend zur Action `action_two()` weiter. Hier wird der Wert ausgelesen und ausgegeben:

```
class CookiesController < ApplicationController
  def action_one
    cookies[:the_time] = Time.now.to_s
    redirect_to(:action => 'action_two')
  end
  def action_two
    cookie_value = cookies[:the_time]
    render(:text => "The Value is #{cookie_value})
  end
end
```

Anm.: Die Methode `to_s()` der Ruby Klasse `Time` erzeugt eine String Repräsentation des Objektes, da der Wert eines Cookies vom Typ String sein muss.

Browser speichern ein kleine Menge an Zusatzinformationen zu jedem Cookie. Diese sind als Optionen `value`, `path`, `domain`, `expires` und `secure` bei der Initialisierung von cookies verfügbar:

```
cookies[:the_time] = { :value => Time.now.to_s,
                       :expires => 30.days.from_now,
                       :secure => false,
                       :path => "/bookadmin" }
```

Anm.: `cookies` wird dabei mit einer zweiten Collection vom Typ Dictionary als Wert zum Schlüssel `the_time` initialisert (s. Dokumentationen der Klasse `Hash` in der Ruby 1.8.4 Corelib Dokumentation, sowie ActionController::Cookies in der Rails Framework Dokumentation).

Cookies müssen i.d.R. für Rails Anwendungen aktiviert sein, da Rails verschiedene Sessions über den Wert des Cookies `_session_id` identifiziert.

39 Definition aus [TDI] S.386

### 2.5.4 Sessions

Sessions werden in Web-Anwendungen gebraucht, damit ihr Informationen über einen bestimmten Benutzer über mehrere Seitenaufrufe hinweg zur Verfügung stehen. Man löst damit das Problem, dass HTTP ein zustandsloses Protokoll ist. Eine Ebene der Anwendung versucht, einen eintreffenden Request zu auf dem Server gespeicherten Session-Daten zuzuordnen. Wenn nun bestimmte Session-Daten allen Requests eines bestimmten Browsers zugeordnet werden können, ist es möglich, die Interaktion eines Benutzers mit der Anwendung nachzuvollziehen.

In diesem Kapitel beschreiben wir die Umsetzung von Sessions in Rails anhand einer Liste, zu der ein Benutzer über mehrere Seitenaufrufe hinweg verschiedene Titel-Datensätze hinzufügen kann. Während er die Anwendung benutzt bleibt die Liste durch den Einsatz einer Session eindeutig mit ihm assoziiert.

Rails identifiziert eine bestimmte Session über eine 32-stellige Hexadezimalzahl, die unter dem Schlüssel `_session_id` in einem Cookie im Browser des Benutzers gespeichert wird. Alle nachfolgenden Requests können damit über die Auswertung der `_session_id` diesem Browser zugeordnet werden. Auf der Serverseite werden die Session-Daten als serialisierte Ruby-Objekte unter derselben Session id gespeichert. Trifft ein neuer Request ein, findet Rails das Ruby-Objekt anhand der Session id, deserialisiert es und übergibt dem `session` Objekt des Controller Environments eine Referenz auf das Objekt. Die Session-Daten stehen damit der Anwendung wieder zur Verfügung.

Falls Model-Klassen in einer Session gespeichert werden sollen, müssen `model` Deklarationen in der Controller-Klasse verwendet werden. Damit werden die Klassen im voraus geladen und sind verfügbar, wenn Rails die gespeicherten Objekte deserialisiert. Damit sie der ganzen Anwendung zur Verfügung stehen, fügt man die Deklaration üblicherweise der Klasse `ApplicationController` (`app/controllers/ application_controller.rb`) hinzu.

#### 2.5.4.1 Vorbereitungen

Als Basis für das folgende Beispiel verwenden wir die Datenbanktabellen aus Kapitel 2.4.6 Beziehungstypen zwischen Tabellen und die Rails Anwendung `books`. Wir wollen die Möglichkeit, Titel zur Ausleihe zu reservieren, realisieren.

In der Session speichern wir eine Liste (Klasse `List`), zu der ein Datensatz über den für jeden Titel angezeigten Textlink “Add to List” hinzugefügt werden kann. In der Liste werden die einzelnen Datensätze, erweitert um zwei Attribute, als Listenpositionen (Klasse `ListItems`) gehalten. Der Reservierung soll dann über die Listenpositionen und eine weitere Klasse `Reservation` realisiert werden.

Jeder Titel-Datensatz kann gegen eine individuelle Gebühr reserviert und geliehen werden. Die Tabellendefinition für `titles` aus Kapitel 2.4.6 erweitern wir deshalb um das Attribut `fee` (dt. Gebühr):

```
CREATE TABLE titles (
   id int NOT NULL auto_increment,
   title varchar(100) NOT NULL,
   publisher_id int NOT NULL,
   year int DEFAULT '0' NOT NULL,
   fee decimal(3,2) NOT NULL,
   constraint fk_titles_publs foreign key (publisher_id) references publishers(id),
   PRIMARY KEY (id)
);
```

Da ActiveRecord die Informationen aus Datenbanktabellen automatisch zur Laufzeit gewinnt, ist an der korrespondierenden Klassendefinition `Title` keine Anpassung notwendig. Lediglich bereits existierende Views müssen - bei Bedarf - angepasst werden, um die Reservierungsgebühr zu jedem Titel darzustellen, wie z.B:

```
<% for title in @titles %>
  ...
  Leihgebühr: <%= number_to_currency(title.fee, :unit=>'&euro;', :precision=>2) %>
  ...
<% end %>
```

#### 2.5.4.2 Liste, Teil 1

Im Controller hält Rails die Datenstruktur `session`. Alle `key => value` Paare, die während eines Requests darin gespeichert werden, sind für alle nachfolgenden Requests desselben Browsers wieder verfügbar. Wir wollen ein neues Listen-Objekt zur Session hinzufügen, wenn es das erste mal gebraucht wird (d.h. wenn der erste Titel zur Liste hinzugefügt wird) und diese Liste dann bei weiteren Requests wieder finden. Dazu implementieren wir die Methode `find_list()` in der Klasse `BookadminController`:

```
private
def find_list
  # conditional assignment operator: the same as a = a op b
  # So count ||= 0 gives count the value 0 if it doesn't already have a value
  @list = session[:list] ||= List.new
end
```

Anm.: Wir benutzen hier Rubys bedingten Zuweisungs-Operator (engl. conditional assignment operator) "`||=`". Falls `session` einen Wert zum Schlüssel `list` hat, wird dieser zurückgegeben (d.h. ein Objekt der Klasse `List`). Andernfalls wird eine neues Objekt von `List` erzeugt, zu `session` hinzugefügt und diese dann anschließend zurückgegeben.

#### 2.5.4.3 Listenpositionen

Als nächstes benötigen wir Model-Klassen für die Liste und die Listenpositionen; beginnen wir mit den Listenpositionen für die folgende SQL Tabellendefinition:

```
create table list_items (
  id int not null auto_increment,
  title_id int not null,
  quantity int not null default 1,
  unit_fee decimal(3,2) not null,
  constraint fk_items_titles foreign key (title_id) references titles(id),
  primary key (id)
);
```

Für die Datenbanktabelle erstellen wir - nach den Namens-Konventionen von Rails - das Model `ListItem`:

```
ruby script/generate model ListItem
      exists  app/models/
      exists  test/unit/
      exists  test/fixtures/
      create  app/models/list_item.rb
      create  test/unit/list_item_test.rb
      create  test/fixtures/list_items.yml
```

Den Beziehungstyp (1:1) zwischen Titel und Listenpositionen müssen wir in den Klassen explizit angeben (Vgl. Kapitel 2.4.6 Beziehungstypen zwischen Tabellen):

```
class ListItem < ActiveRecord::Base
  belongs_to :title
end

class Title < ActiveRecord::Base
  # ...
  has_one :list_item
end
```

#### 2.5.4.4 Liste, Teil 2

Es folgt die Klasse für die Liste. Logisch betrachtet speichert diese Klasse Daten, ist also ein Model. Da die Liste jedoch nicht in der Datenbank gespeichert werden soll (die Speicherung einer Reservierung samt Listenpositionen und Adressdaten des Benutzers soll später die Klasse `Reservation` übernehmen), erbt die Klasse `List` nicht von ActiveRecord::Base.

Unser Design sieht vor, dass in der Liste die Titel als Listenpositionen gehalten werden. Dafür benötigen wir ein Array `@list_items`. Ferner soll es möglich sein, die Gesamtgebühr zu erhalten, wir speichern diese in `@total_fee`:

```
class List
  # attr_reader creates instance variables and corresponding
  # methods that return the value of each instance variable ...
  attr_reader :list_items, :total_fee
  def initialize
    @list_items = []
    @total_fee = 0.0
  end
end
```

#### 2.5.4.5 Titel zur Liste hinzufügen

Die Detailseite zu jedem Titel `show_title.rhtml` enthält einen Textlink "`Add to list`". Die Rails Syntax zur Erzeugung von Hyperlinks ist:

```
<%= link_to 'Add to list', { :action => 'add_to_list', :id => title.id } %>
```

Anm.: Der Hyperlink zeigt auf die Action `add_to_list()` der Klasse `BookadminController` und übergibt dieser die Titel-id als Parameter. Durch die Verwendung der id identifizieren wir den Titel, der zur Liste hinzugefügt werden soll, eindeutig.

Die Action `add_to_list()` implementieren wir in `BookadminController`. Dafür benötigen wir zuerst das zur id korrespondierende Titelexemplar (d.h. einen Datensatz aus der Tabelle `titles`) und die Liste aus `session`. Anschließend fügen wir den Titel zur Liste hinzu und zeigen die Liste an:

```
def add_to_list
  title = Title.find(params[:id])
  @list = find_list
  @list.add_title(title)
  redirect_to(:action => 'display_list')
end
```

Daraus ergeben sich die nächsten Schritte: Die Methode `add_title()` in der Klasse `List` und die Action Methode `display_list()` in der Klasse `BookadminController`, sowie den zugehörigen View `display_list.rhtml`. Zunächst die Methode `add_title()`:

```
def add_title(title)
  item = @list_items.find { |i| i.title_id == title.id }
  if item
    item.quantity += 1
  else
    @list_items << ListItem.for_title(title)
  end
  @total_fee += title.fee
end
```

`for_title()` ist eine Klassenmethode der Klasse `ListItem`, die ein Objekt vom Typ `ListItem` aus dem Titel erzeugt:

```
def self.for_title(title)
  item = self.new
  item.quantity = 1
  item.title = title
  item.unit_fee = title.fee
  item  # returns this item
end
```

#### 2.5.4.6 Anzeigen der Titel

Schließlich kommen wir zur Anzeige der Titel der Liste. In der Methode `add_to_list()` verwendeten wir `redirect_to(:action => 'display_list')`. Das bedeutet, dass wir zumindest einen korrespondierenden View `display_list.rhtml` benötigen (Vgl. Kapitel 2.5.2.2.1 Templates rendern). Weil wir den Inhalt der Liste, also die Listenpositionen, darstellen wollen, brauchen wir aber auch eine Action Methode `display_list()` in der Klasse `BookadminController`, die diese Funktionalität implementiert:

```
def display_list
  @list = find_list
  @list_items = @list.items
end
```

Das View-Template `display_list.rhtml` verwendet die oben definierte Instanzvariable `@list_items`. Das für Rails Verhältnisse umfangreiche Listing entsteht nur durch die Verwendung einer HTML Tabelle:

```
<table border="0" cellpadding="5" cellspacing="0">
  <tr>
    <td rowspan="2">Qty.</td>
    <td rowspan="2">Title</td>
    <td colspan="2">Fee</td>
  </tr>
  <tr>
    <td>Each</td>
    <td>Total</td>
  </tr>
  <%
    for item in @list_items
      title = item.title
  -%>
```

```
  <tr>
    <td><%= item.quantity %></td>
    <td><%= h(title.title) %></td>
    <td align="right">
      <%= number_to_currency(item.unit_fee, :unit => '&euro;', :precision => 2) %>
    </td>
    <td align="right">
      <%= number_to_currency(item.unit_fee * item.quantity,
                             :unit => '&euro;', :precision => 2) %>
    </td>
  </tr>
  <% end -%>
  <tr>
    <td colspan="3" align="right"><b>Total</b></td>
    <td align="right">
      <%= number_to_currency(@list.total_fee, :unit => '&euro;', :precision => 2)%>
    </td>
  </tr>
</table>
<ul>
  <li><%= link_to 'Continue', :action => 'list' %></li>
</ul>
```

### 2.5.5 Filter und Verifikationen

Filter ermöglichen es, Methoden zu definieren, die vor und/oder nach dem Aufruf einer Action ausgeführt werden. Sie können entweder nur durch eine bestimmte Menge oder durch alle Actions der korrespondierenden Controller-Klasse (und ihrer Subklassen) ausgelöst werden.

Die Filter des Controllers sind in ActionController::Filters::ClassMethods definierte Klassenmethoden, die neben den Callback Methoden aus ActiveRecord die zweite Familie von Methoden bilden, für die ein Anwendungsentwickler bei Bedarf Code schreiben kann.[40] Im Einzelnen werden before, after und around Filter unterstützt, die i.d.R. eine Referenz auf eine als protected oder private deklarierte Methode innerhalb der Controller Vererbungshierarchie als Parameter erwarten. Diese Referenz wird überlicherweise als Symbol übergeben. Filter haben Zugriff auf die `request` und `response` Objekte des Controllers, sowie auf alle Instanzvariablen, die während der Ausführungskette durch andere Filter oder - im Falle von after Filtern - während der Abarbeitung der Action gesetzt werden. Sie können zudem aktiv in die Behandlung des Requests eingreifen, z.B. kann ein before Filter durch die Rückgabe von `false` oder einen Aufruf von `render()` bzw. `redirect()` den Aufruf der eigentlich vorgesehenen Action unterbinden.

Bsp.: Ein before Filter kann prüfen, ob ein Benutzer beim Aufruf einer durch Authentifizierung[41] geschützten Seite (z.B. http://0.0.0.0:3000/bookadmin/edit_title) auch tatsächlich im System angemeldet ist, und, falls dies nicht der Fall ist, den Zugriff auf diese Seite verhindern (z.B. mit einer Umleitung auf die Seite http://0.0.0.0:3000/login/login). Dazu definieren wir in der korrespondierenden Controller-Klasse die Methode `authorize()`, die durch den before Filter getriggert werden soll. Wir überprüfen, ob unter dem Schlüssel `user_id` ein Wert im `session` Objekt vorhanden ist und erlauben nur in diesem Fall den Zugriff:

---

40 s. Kapitel 2.4.8 Callback Methoden und Observer

41 Authentifizierung = Überprüfung der Identität

```
class BookadminController < ApplicationController
  # ...
  protected
  def authorize
    unless session[:user_id]
      flash[:notice] = “Please log in!”
      redirect_to(:controller => “login”, :action => “login”)
    end
  end
end
```

Mit folgender Anweisung legen wir dann `authorize()` als before Filter für alle Action Methoden fest, d.h. `authorize()` wird vor dem Aufruf jeder Action dieses Controllers ausgelöst:

```
class BookadminController < ApplicationController
  before_filter :authorize
  # ...
end
```

Die Auslösung von `authorize()` kann mit den Optionen `:only` oder `:except` auf bestimmte Actions reduziert werden. Übertragen auf das Beispiel wäre die folgende Überlegung sinnvoll: Da die Actions, welche eine Authentifizierung erfordern, im Vergleich zu allen Actions des Controllers in der Unterzahl sind, verwenden wir `:only`. Das vollständige Listing könnte somit lauten:

```
class BookadminController < ApplicationController
  before_filter :authorize, :only => [ :edit_title, :delete_title, :new_title ]
  def edit_title
    # ...
  end
  def delete_title
    # ...
  end
  def new_title
    # ...
  end
  # more Action Methods ...
  # ...
  protected
  def authorize
    unless session[:user_id]
      flash[:notice] = “Please log in!”
      redirect_to(:controller => “login”, :action => “login”)
    end
  end
end
```

Die Verwendung von `after_filter()` erfolgt analog zu `before_filter()`. Die Reihenfolge der durch die Filter getriggerten Methoden kann durch das Präfix `prepend_` für alle drei Filtertypen angepasst werden. Zusätzlich steht das Präfix `skip_` für after und before Filter zur Verfügung. Sinnvoll ist `skip_` z.B. für Subklassen einer Vererbungshierarchie an Controllern, die einen oder eine bestimmte Menge an Filtern übergehen wollen.

Ein around Filter ist durch eine Klasse gekapselt, welche die Instanzmethoden `before(controller)` und `after(controller)` implementieren muss. Er wird im Gegensatz zu den anderen Filtertypen nicht durch ein Symbol, sondern durch die Übergabe eines Exemplars der Klasse bei der Callback Methode registriert:

```
class BookadminController < ApplicationController
  around_filter MyAroundFilter.new
  # ...
end

class MyAroundFilter
  def before(controller)
  # ...
  end
  def after(controller)
  # ...
  end
end
```

Die Klassenmethode `verify()` im Modul ActionController::Verification::ClassMethods ist ein besonderer before Filter, mit der Vorbedingungen in manchen Fällen prägnanter formuliert werden können. Mit `verify()` können z.B. die Bedingungen und die Ablauflogik, die im Beispiel oben in die `authorize()` Methode ausgelagert wurden, direkt angegeben werden:

```
verify     :only => [ :edit_title, :delete_title, new_title ],
           :session => :user_id,
           :add_flash => { :notice => “Please log in!” },
           :redirect_to(:controller => “login”, :action => “login”)
```

Mit der Option `:params` kann zusätzlich überprüft werden, ob ein für die Ausführung der Action benötigter Schlüssel oder eine Menge an Schlüsseln in der `params` Datenstruktur vorhanden sind.

## 2.6 ActionView

Wie das routing Subsystem einen Controller identifiziert und wie der Controller eine Action auswählt, haben wir im Kapitel ActionController erläutert. I.d.R. hat die Verarbeitung einer Action und die Erzeugung einer Antwort für den Benutzer das Rendering eines View-Templates zur Folge; die zentrale Methode dabei ist `render()`.

Das Modul ActionView kapselt die für das Rendering von Templates notwendige Funktionalität. ActionView bildet neben ActionController und ActiveRecord den dritten Teil der MVC Architektur.

### 2.6.1 Templates

Die Views im MVC Paradigma werden in Rails Templates genannt. Templates enthalten sowohl statische als auch dynamische Inhalte. Für das Verständnis von Templates sind die folgenden drei Punkte wichtig:

- wie Templates gefunden werden und wo sich Templates befinden können (s. Kapitel 2.5 ActionController)
- das Template Environment
- die Inhalte der Templates

#### 2.6.1.1 Das Template Environment

Die dynamischen Inhalte der Templates sind, wie die gesamte Anwendung, ebenfalls in Ruby geschrieben. Dieser Code hat Zugang zu denen vom Controller bereitgestellten Daten, dem Controller Environment. Das bedeutet, dass

- der Code Zugriff auf alle Instanzvariablen des Controllers hat
- das aktuelle Controller-Objekt über das Attribut `controller` (definiert in ActionView::Base) verfügbar ist. Damit ist z.B. der Zugriff auf alle öffentlichen Instanzmethoden des korrespondierenden Controllers und seiner Oberklasse ActionController möglich.

  Bsp.: Erweitern wir das Template `hello.rhtml` aus Kapitel 1.4 Hello Rails!, um einige Meta-Informationen anzuzeigen. Die verwendeten Methoden sind in ActionController::Base definiert:

```
<html>
  <head>
    <title>Hello Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
    <p>Controller class name => <%= controller.controller_class_name() %></p>
    <p>Controller name => <%= controller.controller_name() %></p>
    <p>Session enabled? => <%= controller.session_enabled? %></p>
  </body>
</html>
```

- die Objekte aus dem Environment des Controllers (`headers`, `params`, `request`, `response` und `session`) über Accessor Methoden verfügbar sind. I.d.R. sollte man die Verantwortung über diese Objekte dem Controller überlassen. Beim Debugging kann es jedoch nützlich sein, auf sie direkt zuzugreifen, um z.B. ihren Inhalt auszugeben

  Bsp.: Erweitern wir das Template `hello.rhtml` aus Kapitel 1.4 Hello Rails! erneut, dieses mal um die Methode `debug()` aus ActionView::Helpers::DebugHelper:

```
<html>
  <head>
    <title>Hello Rails!</title>
  </head>
  <body>
    <h1>Hello from Rails!</h1>
    <p>Params: <p>
    <p><%= debug(params) %></p>
    <p> </p>
    <p>Session: </p>
    <p><%= debug(session) %></p>
  </body>
</html>
```


*Abb. 6: Verwendung des Attributs controller in einem Template.*

#### 2.6.1.2 Die Inhalte der Templates

ActionView unterstützt zwei Formate für Templates:

- RHTML Templates sind eine Mischung aus HTML und eingebettetem Ruby (ERb)
- RXML Templates benutzen die Builder Library, um Antworten in XML zu erzeugen

**Builder Templates**

Ein Builder Template enthält Ruby Code, der die Builder Library benutzt, um XML zu erzeugen. Builder unterstützt u.a. Namespaces, Entites, Processing-Instructions u.v.m. Über das folgende Beispiel hinaus werden wir jedoch im Rahmen dieser Arbeit nicht weiter auf die Builder Library eingehen. Detaillierte Informationen zu Builder sind in der Builder Dokumentation (http://builder.rubyforge.org) zu finden.

Mit Builder wollen wir die Ausgabe aller Titel eines bestimmten Verlages in XML erzeugen. Dazu benutzen wir - ebenso wie für das Template `list_titles_by_publishers.rhtml` - die Action Methode `list_titles_by_publishers()`. Der zusätzliche Aufruf der Methode `render()` verhindert das automatische Rendering des RHTML Templates, indem statt dessen das Builder Template `titles.rxml` gerendert wird:

```
def list_titles_by_publisher
  pub_id = params[:id]
  @titles = Title.find(:all,
                  :conditions => ["publisher_id = ?", pub_id], :order=>"title ASC")
  render(:file => "bookadmin/titles", :use_full_path => true)
end
```

Bemerkenswert ist, dass Builder die Namen von Methoden in XML Tags konvertiert. So erzeugt z.B. der Aufruf `xml.author(author.name, :id => author.id)` das Tag `<author>`, dessen Inhalt durch den ersten Parameter gegeben ist (der Name des Autors) und dessen Attribut id mit dem Wert `author.id()` durch den zweiten Paramter, eine Collection vom Typ Dictionary, erzeugt wird.

Der vollständige Code in `titles.rxml` ist:

```
xml.div do
  for title in @titles
    xml.title(:id => title.id) do
      xml.title(title.title)
      xml.publisher(title.publisher.name)
      xml.year(title.year)
      title.authors.each do |author|
        xml.author(author.name, :id => author.id)
      end
    end
  end
end
```

Der Aufruf der URL http://0.0.0.0:3000/bookadmin/list_titles_by_publisher/6 für den Verlag Pragmatic Bookshelf (`id=6`) erzeugt die Ausgabe:

```
<div>
  <title id="20">
    <title>Agile Web Development with Rails</title>
    <publisher>Pragmatic Bookshelf</publisher>
    <year>2005</year>
    <author id="18">Thomas, David</author>
    <author id="21">Heinemeier Hansson, David</author>
  </title>
  <title id="18">
    <title>Programming Ruby. The "Pick Axe" Book, 2nd Ed.</title>
    <publisher>Pragmatic Bookshelf</publisher>
    <year>2004</year>
    <author id="17">Hunt, Andrew</author>
    <author id="18">Thomas, David</author>
    <author id="20">Fowler, Chad</author>
  </title>
  <title id="19">
    <title>Rails Recipes</title>
    <publisher>Pragmatic Bookshelf</publisher>
    <year>2006</year>
    <author id="20">Fowler, Chad</author>
  </title>
</div>
```

Natürlich besteht auch die Möglichkeit, ein Tag frei zu benennen. Man verwendet dazu die Methode `xml.tag!()`, z.B. auch, um Namens-Konflikte gleich lautender Tags zu vermeiden:

```
xml.tag!("publisherID", title.publisher.id)
xml.tag!("authorID", author.id)
```

**RHTML Templates**

Ruby Code kann in Templates - ganz ähnlich wie Java in JSPs - als sog. Inline-Expressions oder Scriplets verwendet werden. Nach der Verarbeitung des Templates liegt eine reguläre HTML Datei vor.

Als Inline-Expressions bezeichnet man in Templates verwendeten Code, der zwischen den Delimitern `<%=` und `%>` steht. Dieser Code wird ausgewertet, das Ergebnis wird mit der Ruby Methode `to_s()` in einen String konvertiert und in der Ergebnisseite statt der Inline-Expression eingesetzt.

In Templates verwendeter Code, der keinen direkten Output erzeugen soll, wird durch die Delimiter `<%` und `%>` eingeschlossen. Im Kontext von JSPs spricht man bei dieser Art der Einbettung von Scriplets. Ohne das Gleichheitszeichen "=" am ersten Delimiter wird der enthaltene Code zwar ausgeführt, aber nicht in das Template eingefügt. Der Code kann die Ausgabe jedoch sehr wohl beeinflussen, wie man z.B. an der for-Schleife `<% for title in @titles %>` im Template `list_titles_by_publisher.rhtml` sieht. Der Schleifenrumpf wird dort für jedes Objekt im Array `@titles` ausgeführt.

```
<% for title in @titles %>
  <b>ID for Title: <b><%= h(title.id) %><br />
  <%= h(title.title) %><br />
  ...
<% end %>
```

Anm.: Der in Scriplets ebenfalls verwendbare Delimiter `-%>` verhindert, dass in der resultierenden HTML Ausgabe eine Leerzeile nach der Ausführung und Entfernung des Scriptlet-Codes bestehen bleibt.

In Inline-Expressions sollte man außerdem immer die Ruby Utility Methode `h(s)` - ein Alias für `html_escape(s)` - verwenden. Diese konvertiert mögliche HTML Tags im String `s` in reguläre HTML Schreibweise

### 2.6.2 Helper

Bei der Einbindung dynamischer Inhalte in den View stellt sich in jeder MVC Architektur schnell die Frage, wie viel Anwendungs- und Businesslogik in diesen Inhalten erlaubt bzw. wünschenswert ist. Sicherlich kann diese Frage nicht allgemein gültig beantwortet werden; guter Programmierstil ist jedoch, die von der MVC Architektur beabsichtigte Trennung von Anwendungslogik und Darstellung in möglichst vielen Fällen einzuhalten.

In Rails bietet es sich z.B. an, im View auf Instanzvariablen des Controllers zuzugreifen, anstatt Berechnungen direkt in ihm auszuführen, da alle Instanzvariablen automatisch und ohne weitere Konfiguration im View verfügbar sind. Rails unterstützt damit den Anwendungsentwickler bei der Einhaltung der "Best Practices" in der MVC Architektur. Schon in der "Hello Rails!" Anwendung findet sich dafür ein ganz einfaches Beispiel:

```
class HelloRailsController < ApplicationController
  def hello
    @time = Time.now
```

```
  end
end

<!-- Benutzung im View -->
<p>The time is <%= @time %>.</p>
```

Im Widerspruch zum MVC Paradigma ist es in vielen Web-Anwendungen auch durchaus üblich, die Validierung von Formulareingaben eines Benutzers client-seitig - d.h. also im View - z.B. mittels JavaScript vorzunehmen. Auch hier sieht Rails eine strengere Trennung vor. Üblicherweise verwendet man in Rails zur Validierung die geerbten Methoden aus dem Modul ActiveRecord::Validations::ClassMethods - sog. Validation Helpers - in der Klassendefinition des korrespondierenden Models. Man hat dadurch auch den Vorteil, auf vordefinierte Fehlermeldungen zurückgreifen zu können.[42] Zur Überprüfung, ob ein Benutzer z.B. in zwei gegebenen Formularfeldern mit Namen `name` und `phone_number` eine Eingabe gemacht hat, und ob es sich bei `phone_number` um einen Zahlenwert handelt, benutzt man in Rails die Methoden `validates_presence_of()` und `validates_numericality_of()`:

```
class Title < ActiveRecord::Base
  validates_presence_of :name, :phone_number
  validates_numericality_of :phone_number
  # ...
end
```

Eine weitere Möglichkeit, Anwendungslogik auszulagern und zu modularisieren hat man, indem man sog. Helper verwendet. Helper sind Module, welche das Verhalten von Templates erweitern, d.h. sie assistieren den Views bei der Erzeugung von HTML bzw. XML. Nach Konvention verfügt jeder Controller über ein eigenes Helper Modul, welches im Verzeichnis `app/helpers` abgelegt ist. Die Namens-Bildung für die Datei, welche das Helper Modul enthält, folgt der Konvention, das Suffix `_helper` an den Namen des korrespondierenden Controllers anzuhängen. Im Beispiel des `HelloRails` Controllers (`hello_rails_controller.rb`) lautet der Name also `hello_rails_helper.rb`.

Anm.: Das `generate controller` Skript erstellt automatisch ein Helper Modul zu jedem Controller. Methoden, die im Modul `ApplicationHelper` definiert sind (`app/helpers/application_helper.rb`) stehen allen View-Templates der Anwendung zur Verfügung.

Eine sinnvolle Helper Methode wäre z.B. eine Methode, welche Preisangaben auf einer HTML Seite abhängig von der Währung visuell unterschiedlich darstellt. Ruby bietet dafür die Methode `sprintf()` an, die einen String in Abhängigkeit des Parameters `format_string` formatiert zurück gibt. `sprintf()` kann man nun - entweder für einen bestimmten Controller oder anwendungsweit - in eine Methode kapseln, damit

- sie für alle View-Templates gleichermaßen zur Verfügung steht,
- eine mögliche Änderung der visuellen Darstellung zentral an einer Stelle vorgenommen werden kann und
- die (relativ) komplexe Signatur der `sprintf()` Methode dem Designer der View-Templates verborgen bleibt.

Die Methode zur Währungsdarstellung kann wie folgt definiert werden:

---

42 s. Kapitel 2.4.7 Validierung und 2.6.5 Fehlerbehandlung für Model-Objekte

```
module ApplicationHelper
  def format_dollars(amount)
    sprintf("US $%0.2f", amount)
  end
end
```

Sie kann dann in jedem beliebigen View der Anwendung verwendet werden, z.B. wie in folgendem Code Fragment dargestellt:

```
<table>
<% for title in @titles -%>
  <tr>
    <td><%= h(title.title) %></td>
    <td><%= format_dollars(title.fee) %></td>
  </tr>
<% end -%>
</table>
```

Rails stellt eine Reihe von Helper Modulen zur visuellen Formatierung, deren Methoden in allen Views einer Anwendung verwendbar sind, standardmäßig zur Verfügung. (Anm.: Darunter befindet sich im Modul ActionView::Helpers::NumberHelper auch eine Methode `number_to_currency()`, welche dieselbe Aufgabe wie unsere selbstdefinierte Methode `format_dollars()` erledigt).

In Kombination mit Layouts und Partial Page Templates kann der Einsatz von Helpern zu einer weiteren Modularisierung des Codes und zu einer noch stärkeren Verringerung von Redundanzen im Code führen.

#### 2.6.2.1 Wiederverwendbarkeit von Helpern

Außer der Möglichkeit, Helper Methoden im Modul ApplicationHelper zu definieren, um sie für die gesamte Anwendung verfügbar zu machen, kann man ein Helper Modul auch über einen Aufruf den Form `helper :helper_name` in einen Controller einbinden.

Hat man ein Modul aus selbstgeschriebenen Helper Methoden, z.B. zur Formatierung von unterschiedlichen Währungsangaben, zusammengestellt, so kann es mit:

```
class ApplicationController < ActionController::Base
  helper :currency_formatter
end
```

in jede beliebige andere Anwendung einbinden, vorausgesetzt die entsprechende Datei `currency_formatter_helper.rb` befindet sich im Verzeichnis `app/helpers` dieser Anwendung.

### 2.6.3 Erzeugung von Hyperlinks

Die Module ActionView::Helpers::AssetTagHelper und ActionView::Helpers::UrlHelper enthalten Helper Methoden, die Verweise auf außerhalb des View-Templates liegende Ressourcen ermöglichen. Die meist verwendete ist sicherlich die Methode `link_to()`, die i.d.R. einen Hyperlink auf eine beliebige Action innerhalb der Anwendung erzeugt. Beispiele für ihre Verwendung finden sich an vielen Stellen in dieser Ausarbeitung. Hier sei einzig darauf hingewiesen, dass durch die Verwendung einer Methode, die auf eine Action zeigt, statt der direkten Verknüpfung der Views (z.B. mit `<a`

`href="..."`>), zwei Aspekte der Model 2 Architektur, nämlich die Entkopplung der Views voneinander und die Verwendung des Front Controller Patterns umgesetzt sind.[43]

Ebenfalls in diesen Modulen sind die Methoden `image_tag()` zur Einbindung von Bildern in einen View, `stylesheet_link_tag()` zur Einbindung von Stylesheets, sowie `mail_to()` zur Erstellung einer Email Nachricht definiert sind.

### 2.6.4 Form Helper

Zur Erstellung von HTML Formularen verwendet man in View-Templates die sog. Form Helper. Unter diesem Begriff versteht man eine Menge von Methoden, welche das Formular Tag an sich (`<form>`) und die Formularfelder wie z.B. `<input>`, `<option>`, `<select>`, usw. erzeugen.

Interessant ist dabei vor allem, wie das Zusammenspiel der Komponenten Model, View und Controller die Erzeugung und Änderung von Datensätzen einer Datenbanktabelle unterstützt.

Die Attribute einer Model-Klasse stehen dem Controller wie auch dem View zur Verfügung. Das folgende Beispiel erläutert anhand eines HTML Formulars zur Aktualisierung Titel-Datensatzes (bestehend aus `id`, `title`, `publisher` und `year`), wie Form Helper diese Eigenschaft nutzen können. Der Ablauf skizziert, wie die Attribute von Model zu Controller, von View-Template zur HTML Seite, und über den Controller zurück zum Model gereicht werden:

- Die Anwendung empfängt einen Request über die Änderung eines bestimmten Titel-Datensatzes (z.B. über die URL http://0.0.0.0:3000/bookadmin/edit_title/123 den Titel mit der id=123). Der Controller legt den Datensatz in der Variablen `@title` - ein Exemplar der Model-Klasse `Title` - ab (über einen an `Title` delegierten Datenbankzugriff):

```
class BookadminController < ApplicationController
  def edit_title
    @title = Title.find(params[:id])
  end
end
```

- Das `edit_title.rhtml` View-Template wird (implizit) aufgerufen. In diesem benutzen die Form Helper Methoden (z.B. `text_field()`) das `@title` Objekt:

```
<html>
  <head>
    <title>Datensatz aktualisieren</title>
  </head>
  <body>
    <%= start_form_tag({ :action => 'update_title', :id => @title.id }) %>
    <%= text_field('title', 'title', :size => 20) %>
    <%= text_field('title', 'publisher', :size => 20) %>
    <%= text_field('title', 'year', :size => 4) %>
    <%= submit_tag('Datensatz aktualisieren!') %>
    <%= end_form_tag %>
  </body>
```

---

43 s. Kapitel 1.2.3.2 MVC 2 oder Model 2 Architekturen

```
</html>
```

- Die erzeugte HTML Seite enthält regulären HTML Code:

```
<form action="/myapp/bookadmin/update_title/123">
<input type="text" id="title_title" name="title[title]" value="#{@title.title}"
                   size="20">
<input type="text" id="title_publisher" name="title[publisher]
                   value="#{@title.publisher}" size="20" >
<input type="text" id="title_year" name="title[year] value="#{@title.year}"
                   size="4" >
<input type="submit" value="Datensatz aktualisieren!">
</form>
```

- Nach der gewünschten Änderung der Daten und betätigen des Submit-Buttons werden die Parameter in das `params` Objekt übernommen:

```
@params = {:id => 123, :title => {:title => '..', :publisher=>'..', :year=>'..' }}
```

- Die `update_title()` Methode verwendet `params`, um den Datensatz zu identifizieren und zu aktualisieren:

```
# class BookadminController
def update_title
  a_title = Title.find(params[:id])
  if a_title.update_attributes(params[:title])
    flash[:notice] = 'Titel erfolgreich aktualisiert!'
  else
    # Error Handling ...
  end
end
```

Alle Form Helper Methoden zur Erzeugung der speziellen HTML Formularfelder (wie z.B. `<input>`) übernehmen mindestens zwei Parameter. An erster Stelle den Namen einer Instanzvariablen (die typischerweise ein Model-Objekt referenziert - im Beispiel oben `@title`), danach den Namen des Attributs dieses Objektes, welches beim Setzen des Wertes des Feldes abgefragt werden soll. Zusammen bilden diese beiden Parameter auch das `name` Attribut des resultierenden Formulartags (z.B. `<input name="title[publisher]">`). Beide Parameter sind entweder Ruby Strings oder Symbole.

Mit dem optionalen Paramter `options` - einer Datenstruktur vom Typ Dictionary - kann dem HTML Formulartag eine CSS Eigenschaft (mittels `{ class => '...' }`) zugewiesen werden.

#### 2.6.4.1 Date and Time Fields

Zwei sehr hilfreiche Form Helper Methoden sind `date_select()` und `datetime_select()`. Sie erzeugen Auswahllisten (mit den HTML Tags `<select>` und `<option>`), durch die auf Datums- und Zeitangaben in Attributen von Model-Klassen zugegriffen werden kann. In MySQL sollte die Spalte der korrespondierenden Datenbanktabelle den Datentyp `date` oder `datetime` haben, damit die gemachten Auswahlen gespeichert werden können.

`date_select(:variable, :attribute, options)` liefert eine Menge von `<select>` Tags (je eines für Jahre, Monate und Tage - `datetime_select()` zusätzlich noch für Stunden und Minuten) für den Zugriff auf ein datums-basiertes Attribut `:attribute` des Objekts `:variable`. Mit `options` - einer

Collection vom Typ Dictionary - können die Auswahllisten angepasst werden. Der voreingestellte Wert der Auswahllisten wird aus `@variable.attribute` gewonnen.

Im Beispiel des Titel-Datensatzes könnte das Datum einer Änderung wie folgt protokolliert werden (vorausgesetzt die SQL Tabellendefinition ist um die Spalte `last_edited datetime not null` erweitert worden)

```
# edit_title.rhtml
...
<%= datetime_select('title', 'last_edited',
                    :start_year => 2001,
                    :order  => [:day, :month, :year, :hour, :minute])
...
```

### 2.6.5 Fehlerbehandlung für Model-Objekte

Die Überprüfung von Formulareingaben erfolgt in Rails nicht client-seitig, d.h. im View, sondern durch die Validation Helper in der Model-Klasse (Vgl. Kapitel 2.4.7 Validierung). Die Attribute der Model-Klasse stehen dem View jedoch zur Verfügung. Zu diesen Attributen gehört auch `errors`, wodurch die Fehlerbehandlung im View, z.B. durch das Hervorheben von Formularfeldern, welche die Überprüfung nicht bestanden haben, einfach zu realisieren ist.

Falls die Überprüfung fehl schlägt, fügen alle Validation Helper eine Nachricht zu `errors` des korrespondierenden Model-Objektes hinzu. In einer Rails Anwendung erhält die Action, welche die Interaktion mit dem Model leitet, dann den Rückgabewert `false` und stellt das HTML Formular erneut dar. Das View-Template für diese Action verwendet die Methode `error_messages_for()`, um `errors` darzustellen. Die Form Helper Methoden rufen dazu für jedes Formularfeld, das aus einem Attribut einem Model-Objekt erzeugt wurde, die Methode `errors.on()` auf.

`error_messages_for(object_name, options = {})` aus dem Modul ActionView::Helpers:: ActiveRecordHelper gibt ein `<div>`-Tag, welches alle Fehlermeldungen für das durch `object_name` referenzierte Model-Objekt enthält, zurück. Standardmäßig hat dieses `<div>` die CSS Eigenschaft `class="fieldWithErrors"`. Mit einem entsprechenden Stylesheet können also alle fehlerhaften Formularfelder visuell hervorgehoben werden. Die `scaffold` Option des `generate` Utility-Skripts (s. folgende Anmerkung) unterlegt diese z.B. mit einem roten Hintergrund:

```
<!-- public/stylesheets/scaffold.css -->
.fieldWithErrors {
  padding: 2px;
  background-color: red;
  display: table;
}
```

Anm.: `scaffold` erzeugt das Code Gerüst für einen bestimmten Teil einer Anwendung automatisch, z.B. auch HTML Formulare (insbesondere Eingabemasken) für Model-Klassen.

Die Methode `on(attribute)` aus der Klasse ActiveRecord::Errors gibt `nil` zurück, falls keine Fehler mit `attribute` assoziiert sind, oder eine Fehlermeldung bzw. ein Array aus Fehlermeldungen, falls ein oder mehrere Fehler mit `attribute` assoziiert sind. `attribute` referenziert je ein Attribut des Model-Objektes.

Beispiel-Code werden wir im Zusammenhang mit Partial Page Templates zeigen.

### 2.6.6 Partial Page Templates, Layouts und Components

Bisher haben wir View-Templates als voneinander isolierte Code-Fragmente betrachtet. In einer Menge von Webseiten gibt es jedoch potentiell viele Stellen, an denen sich Code wiederholen kann, z.B. befindet sich auf jeder einzelnen Webseite eine Navigationsleiste, sowie je eine Kopf- und Fußzeile und teilweise dieselben inhaltlichen Bestandteile. Auch dieselbe Funktionalität wird i.d.R. auf mehr als einer Webseite verwendet. Rails stellt mit Partial Page Templates (auch einfach Partials genannt), Layouts und Components drei Möglichkeiten zur Verfügung, um redundanten Code in View-Templates zu vermeiden. Wir werden im folgenden diese Möglichkeiten anhand einfacher Beispiele vorstellen.

#### 2.6.6.1 Partial Page Templates

Viele weitgehend identische Bestandteile müssen i.d.R. auf mehr als nur einer einzigen Seite darstellt werden. Im Kapitel 2.6.4 Form Helper erstellten wir z.B. das View-Template `edit_title.rhtml` zur Änderung eines bestimmten Titel-Datensatzes. Man benötigt nun jedoch ein in weiten Teilen identisches HTML Formular sowohl zur Änderung von Datensätzen, als auch um neue Datensätze anzulegen.

Der Partial Page Template Mechanismus vermeidet in diesem Fall die Duplizierung des Formular Codes auf zwei View-Templates. Ein Partial ist in einer eigenen RHTML Datei definiert, die nach Konvention ebenfalls im Verzeichnis `app/views/controller_name` abgelegt ist, deren Namen jedoch mit einem Unterstrich beginnen muss.

Die identischen Teile der Formulare können z.B. in die Datei `app/views/bookadmin/_form.rhtml` ausgelagert werden:

```
<%= text_field('title', 'title', :size => 20) %>
<%= text_field('title', 'publisher', :size => 20) %>
<%= text_field('title', 'year', :size => 4) %>
```

Die Datei `edit_title.rhtml` wird unter Verwendung der Methode `render(:partial)` angepasst. Der `:partial` Parameter gibt den Namen des zu rendernden Partial Page Templates - jedoch ohne den Unterstrich zu Beginn - an. Als Ergänzung und Nachtrag zu Kapitel 2.6.5 Fehlerbehandlung für Model-Objekte stellen wir zusätzlich mit dem Aufruf der Methode `error_messages_for()` die bei der Validierung auftretenden Fehlermeldungen dar:

```
<html>
  <head>
    <title>Datensatz aktualisieren</title>
  </head>
  <body>
    <%= error_messages_for :title %>
    <%= start_form_tag({ :action => 'update_title', :id => @title.id }) %>
    <%= render(:partial => 'form' %>
    <%= submit_tag('Datensatz aktualisieren!') %>
    <%= end_form_tag %>
  </body>
</html>
```

Nach demselben Prinzip wird die Datei `app/views/bookadmin/new_title.rhtml` angelegt:

```
<html>
  <head>
    <title>Neuen Datensatz erstellen</title>
  </head>
  <body>
    <%= error_messages_for :title %>
    <%= start_form_tag({ :action => 'new_title' }) %>
    <%= render(:partial => 'form' %>
    <%= submit_tag('Neuen Datensatz erstellen!') %>
    <%= end_form_tag %>
  </body>
</html>
```

#### 2.6.6.2 Components

Partials erlauben es, ein Code Fragment aus einem View-Template auszulagern, um es dann an mehreren Stellen wieder verwenden zu können. Häufig trifft man jedoch auch auf den Fall, dass sowohl Teile eines Views als auch eine bestimmte Funktionalität oder Anwendungslogik wiederholt verwendet wird. Ein Einkaufswagen wird z.B. während des Einkaufsvorgangs regelmäßig nach dem Hinzufügen eines Produktes angezeigt, sowie auch in der Zusammenfassung von Produkt- und Kundendaten gegen Ende des Bestellvorgangs.

Mit der Methode `render_component()` kann der Anwendungsentwickler eine Action aus einem View-Template heraus aufrufen. Die Action Methode wird ausgeführt und ihr korrespondierendes View-Template wird gerendert. Diese Ausgabe wird dann in die Ausgabe des aufrufenden Views automatisch integriert.

Bsp.: Im Kapitel 2.5.4 Sessions definierten wir u.a. die Methode `display_list()` zur Anzeige der Listenpositionen. Wir erweitern das Beispiel um eine Seite `checkout.rhtml`, auf welcher der Benutzer seine persönlichen Daten für den Leihvorgang angeben kann. Auf dieser Seite möchten wir die Listenpositionen ebenfalls anzeigen, ohne jedoch den Code dafür zu duplizieren. Wir erreichen dies mit folgendem Code-Fragment:

```
<!-- First, display the current list: -->
<%= render_component(:action => "display_list") %>
<!-- Then show a form to the customer: -->
<h1>Bitte geben Sie Ihre persönlichen Daten an: </h1>
<%= start_form_tag(:action => "save_order") %>
<table>
  <tr>
    <td>Name: </td>
```

`render_component()` ruft die angegebene Action `display_list()` auf und setzt deren Ausgabe in das aktuelle View-Template `checkout.rhtml` ein.

Ein Problem ergibt sich dadurch, dass beide View-Templates in ein Layout eingebettet sind (Layouts werden im nächsten Unterkapitel erläutert). Wir wollen erreichen, dass `display_list.rhtml` ohne Layout dargestellt wird, falls die korrespondierende Action Methode mittels `render_component()` aufgerufen wird. Dazu fügen wir zuerst mit der Option `:params` der Methode `render_component()` ein neues `key => value` Paar in die `params` Datenstruktur des Controllers ein. Wir müssen lediglich darauf achten, dass der Schlüssel eindeutig ist; wir wählen in diesem Beispiel `context`:

```
<%= render_component(:action => "display_cart", :params=>{:context=>:checkout}) %>
```

Danach passen wir die Methode `display_list()` an, indem wir den Wert von `context` prüfen und ggf. das Layout mit `render(:layout => false)` deaktivieren:

```
def display_list
  @list = find_list
  @list_items = @list.items
  if @list_items.empty?
    redirect_to(:action => “list”)
  end
  if params[:context] == :checkout
    render(:layout => false)
  end
end
```

#### 2.6.6.3 Layouts

Wie am Beispiel aus Kapitel 2.6.6.1 Partial Page Templates zu erkennen ist, reduzieren Partials zwar die Menge an redundantem Code, dennoch treten weiterhin Wiederholungen auf (z.B. der gesamte Kopfbereich der HTML Seite). Hier kommen nun Layouts zum Einsatz. Rails erlaubt es, gerenderte Templates in andere, ebenfalls gerenderte Templates zu schachteln (engl. Nested Templates). Üblicherweise wird diese Eigenschaft dazu verwendet, die Ausgabe einer beliebigen Action in ein anwendungsweit einheitliches Layout, z.B. bestehend aus Navigationsleiste, sowie Kopf- und Fußzeile, einzubinden. Nach Konvention ist das Layout-Template an einen bestimmten Controller gebunden und wird für alle Action Methoden dieses Controllers eingesetzt, sofern diese die `:layout` Option der Methode `render()` nicht explizit auf `false` setzen (wie im Beispiel zu Kapitel 2.6.6.2 Components).

Bsp.: Das Layout Template für die Klasse `BookadminController` ist nach Konvention `app/views/layouts/ bookadmin.rhtml`. In ihm definieren wir die auf jeder Seite gleichbleibenden Elemente, wie z.B.:

```
<html>
  <head>
    <title>Books Anwendung => <%= @page_title %></title>
    <%= stylesheet_link_tag “books_styles”, media => “all” %>
  </head>
  <body>
    <div id=”top”>
      <%= @page_title || “Willkommen bei der Books Anwendung” %>
    </div>
    <div id=”cols”>
      <div id=”side”>
        <%= link_to “Home”, :action => “index” %>
        <%= link_to “List”, :action => “list” %>
      </div>
      <div id=”main”>
        <%= @content_for_layout %>
      </div>
    </div>
    <div id=”footer”>
      <%= link_to “Contact”, :action => “contact” %>
    </div>
  </body>
</html>
```

Die Ausgabe aller Actions, die durch die Methode `render()` implizit oder explizit gerendert werden, wird automatisch der Instanzvariablen `@content_for_layout` zugewiesen, die folglich im Layout-

Template mit `<%= @content_for_layout %>` nur noch evaluiert werden muss. Alle View-Templates des korrespondierenden Controllers vereinfachen sich auf die tatsächliche Ausgabe ihrer Actions, wie z.B. `edit_title.rhtml`:

```
<% @page_title = "Datensatz aktualisieren!" -%>
<%= error_messages_for :title %>
<%= start_form_tag({ :action => 'update_title', :id => @title.id }) %>
<%= render(:partial => 'form' %>
<%= submit_tag('Datensatz aktualisieren!') %>
<%= end_form_tag %>
```

Layout-Templates haben denselben Zugriff auf Daten wie konventionelle View-Templates auch. Zusätzlich stehen Layout-Templates alle Instanzvariablen, die in den einzelnen View-Templates gesetzt werden, zur Verfügung. Jedes View-Template kann so z.B. einen individuellen Seitentitel festlegen (im Beispiel über `@page_title`).

# 3. Weiterführende Themen

## 3.1 Rails aus der Perspektive von Ruby

Wie eingangs bereits erwähnt wurde, unterscheidet sich Rails von anderen Web-Frameworks u.a. auch deshalb, weil es in der Sprache Ruby implementiert ist. Wir wollen uns in diesem Kapitel dieser Besonderheit erneut widmen und das Rails Framework aus der Perspektive von Ruby betrachten. Gleichzeitig wird sich zeigen, dass ein Rails Entwickler von umfangreichen Kenntnissen und Erfahrungen mit Ruby sehr profitiert, da er nur so die Möglichkeiten, die Rails zu bieten hat, voll ausschöpfen kann.

Kenntnisse in der Sprache Ruby nutzen dem Rails Entwickler auf folgende Arten:

- er erkennt, was der Code der Anwendung - einschließlich dem Rails Standardcode - macht
- er hat die Möglichkeit, Rails Anwendungen durch den Einsatz von Ruby beliebig zu erweitern. Das Framework sieht diese Erweiterungen ausdrücklich vor und bietet bestimmte Erweiterungspunkte an (z.B. Helper Module und Hook bzw. Callback Methoden).
- er hat die Möglichkeit, den Rails Quell-Code zu lesen und zu verstehen und kann Rails Standardcode von Ruby Code unterscheiden. Dadurch gewinnt er einen Einblick in das Zusammenspiel von Ruby und Rails und kann Weiterentwicklungen und/oder Änderungen am Framework nachvollziehen oder sich sogar selbst an der Weiterentwicklung beteiligen (s. folgende Anmerkung).
- mit Ruby hat er die Möglichkeit, administrative Aufgaben im Umfeld der Anwendungsentwicklung zu erledigen, z.B. durch die Verwendung von `irb`.

Anm.: Die am 28. März 2006 vorgestellte Rails Version 1.1 brachte über 500 Erweiterungen und/oder Veränderungen mit sich.[44] Beispiele für die Weiterentwicklung des Frameworks durch die Anwendungsentwickler finden sich z.B. unter http://wiki.rubyonrails.org/rails/pages/AvailableGenerators.

### 3.1.1 Rails als domänen-spezifische Programmiersprache

Das Rails Framework vermittelt dem Entwickler in gewisser Weise das Gefühl, nicht im eigentlichen Sinne zu programmieren, sondern die Anwendung stattdessen anhand einer Menge von gegebenen Parametern zu konfigurieren. Zieht man die Anwendungsdomäne, also interaktive, datenbankgestützte Web-Anwendungen, mit in Betracht, kann Rails als eine domänen-spezifische Programmiersprache, d.h. eine Programmiersprache, die Anforderungen für bestimmte Problemstellungen besonders gut erfüllt, gesehen werden.

Eine DSL (domain specific language, dt. domänen-spezifische Programmiersprache) ist eine Sprache, die entworfen wurde, um eine Aufgabe oder eine bestimmte Menge an Aufgaben in einem bestimmten Umfeld, Kontext oder einer bestimmten Anwendungsdomäne zu lösen, mit der Intention, die fachlichen Aspekte der Programmerstellung von den technischen zu trennen. Als Hintergrund können die kontinuierlichen Bemühungen in der Informatik, die Komplexität von Software durch die Kapselung von Teilaspekten zu beherrschen, gesehen werden. Der Befehlssatz oder Standard-Wortschatz einer

44 s. http://weblog.rubyonrails.com/articles/2006/03/28/rails-1-1-rjs-active-record-respond_to-integration- tests-and 500-other-things

DSL ist i.d.R. (sehr) klein. Im Gegensatz zu DSLs stehen die universell einsetzbaren Programmiersprachen (engl. general purpose programming languages), zu denen auch Ruby gehört.

Da Ruby eine geeignete Host-Sprache für DSLs ist und sich das Rails Framework diese Eigenschaft bewusst zu Nutze macht, hat der Entwickler also oft das Gefühl, entweder das System anhand gegebener Parameter zu konfigurieren oder zumindest eine eigene, auf Ruby aufbauende Sprache zu verwenden. Dennoch ist die verwendete Sprache Ruby und der Entwickler erstellt eine objekt-orientierte Anwendung in Ruby. Er muss daher genau beachten, wie diese beiden Ebenen der Programmierung verbunden sind, und welche Rolle Ruby selbst darin einnimmt, wenn manche Rails Standardcode-Fragmente und -idiome teilweise nicht mehr an Programmierung erinnern.[45]

Es sind hauptsächlich zwei Gründe, die Ruby als geeignete Host-Sprache für DSLs auszeichnen:

- Ruby erlaubt es, Sprachkonstrukte zu einem großen Teil zu redefinieren
- die Ruby Syntax erlaubt es auch unerfahrenen Programmierern mit einer wohldefinierten Untermenge an Sprachkonstrukten effizient umzugehen

Rails zeigt, besonders für einen unerfahrenen Ruby Programmierer, seine Besonderheiten auf zwei Ebenen: Zum einen in der Syntax, zum anderen in der Terminologie [46]

Besonderheiten in Relation zur Syntax treten an vielen Stellen innerhalb des Framework auf. Stellvertretend wollen wir die folgenden nennen:

- Fehlende bzw. optionale Klammerung bei Methodenaufrufen, z.B. `has_many :titles`
- Häufige Verwendung von Ruby Symbolen, z.B. als Keyword Parameter
- Dictionaries als Methodenparameter
- Accessor Methoden in Ruby “Short Cut” Notation

Die Besonderheiten in Relation zur Terminologie stehen natürlich auch in direktem Zusammenhang mit der Anwendungsdomäne von Rails Anwendungen. In Hinblick auf die Architektur sei die standardmäßige Benennung der Verzeichnisse `app/models`, `app/controllers` und `app/views` in direkter Analogie zum MVC Paradigma genannt. Auf der Ebene der Implementierung fallen die Methodennamen für die Beziehungstypen zwischen Datenbanktabellen, wie z.B. `has_one()`, `has_many()`, `belongs_to()` und `has_and_belongs _to_many()`, auf, oder die Methodennamen der Form Helper, wie z.B. `start_form_tag()`, `text_field()`, `submit_tag()`, usw..

Beide Besonderheiten unterstützen den oben bereits angedeuteten Eindruck: Eine Rails Anwendungen erstellen erinnert an Konfiguration eines Systems unter bestimmten Bedingungen. Dazu einige Beispiele:

Eine häufig eingesetzte Anweisung ist z.B.:

```
has_many :titles
```

45 In [BLACK] wird dieser Sachverhalt mit “ configuartion-like programming and programming thats actually configuration” beschrieben.

46 Terminologie = die Menge aller Begriffe und Benennungen einer Fachsprache

Man hat - bedingt durch die Syntax - den Eindruck, eine Konfiguration der Form `eine_option = ein_wert` vorzunehmen, was für einen unerfahrenen Entwickler oder jemanden, der Rails neu kennen lernt, einen einfacheren und übersichtlicheren Eindruck als ein typischer Methodenaufruf macht.

Es ist jedoch trotzdem ein Methodenaufruf in regulärer Ruby Syntax. Die Klammerung um die Methodenparameter ist in Ruby optional, und statt eines Strings kann in den meisten Fällen ein Symbol verwendet werden. Eine andere Schreibweise wäre daher:

```
has_many(”titles”)
```

Ruby erlaubt es, mit der Syntax relativ frei umzugehen, und die Designentscheidung des Rails Erfinders war eindeutig, sich der Variante zu bedienen, die am wenigsten an traditionelle Programmierung erinnert und die Absicht hinter einem Methodenaufruf am besten dokumentiert. Deutlicher wird dies vielleicht anhand des folgenden Beispiels: Die Methode `link_to()` erzeugt einen Hyperlink in einem View-Template und hat viele (optionale) Parameter, die jedoch durch die Verwendung eines Dictionaries mit Symbolen als Schlüssel trotzdem verständlich bleiben:

```
<%= link_to “Add to list”,
          { :controller => “bookadmin”
            :action     => “add_to_list”
            :id         => “title.id },
            :class      => “regular_textlink” %>
```

... v.a. wenn man einen typischen Methodenaufruf in einer anderen Programmiersprache gegenüberstellt:

```
add_to_list(”Add to list”, bookadmin, add_to_list, title.id, “regular_textlink”);
```

Programmcode kann daher wie ein Auszug aus einer Konfigurationsdatei aussehen und Vorteile hinsichtlich der Übersichtlichkeit, der Auffassung der dahinter liegenden Programmlogik, der Dokumentation und der Kommunikation zwischen Entwicklern haben. Ein Anwendungsentwickler sollte sich bei der Erstellung von eigenem Ruby Code innerhalb von Rails immer an diesen Stil halten, obwohl dies keinesfalls durch das Framework erzwungen wird, sondern lediglich eine Konvention darstellt.

## 3.2 AJAX on Rails

AJAX (Asynchronous JavaScript and XML) ist der Oberbegriff für eine Technologie zur Entwicklung von interaktiven Web-Anwendungen. Die Grundidee dabei ist, dass nicht ständig neue Seiten dargestellt werden, sondern Inhalte asynchron vom Server nachgeladen werden können. Eine Folge davon ist, dass Webseiten ansprechbarer reagieren und eine bessere Interaktion mit dem Benutzer möglich ist. Der Begriff selbst wurde von Jesse James Garret bzw. Adaptive Path geprägt.[47]

Das Rails Framework unterstützt AJAX durch:

- die Einbindung des Prototype Frameworks und der darauf aufbauenden script.aculo.us Libraries
- das Rails Modul JavaScriptHelper, welches die Verwendung von JavaScript in Views ermöglicht.

47 s. http://www.adaptivepath.com/publications/essays/archives/000385.php

### 3.2.1 Prototype JavaScript Framework

Prototype[48], entwickelt von Sam Stephenson, ist ein JavaScript Framework, dessen Ziel es ist, die Entwicklung dynamischer Web 2.0 Anwendungen zu vereinfachen. Sam Stephenson gehört zum Rails "core team of comitters" und veröffentlichte die erste Prototype Version im Februar 2005. Prototype bildet die Basis für die Unterstützung von AJAX in Rails.

Anhand der Beispiele in diesem Kapitel werden wir einige Aspekte von Prototype näher erläutern.

### 3.2.2 script.aculo.us JavaScript Libraries

script.aculo.us[49] ist ebenfalls die Entwicklung eines Rails "core committers", Thomas Fuchs. Die cross-browser fähige JavaScript Library unterstützt u.a. anspruchsvolle User-Interface-Kontrolle (z.B. drag-and-drop) und visuelle Effekte. Nach Thomas Fuchs' Meinung haben Browser und Web-Standards mittlerweile ein Niveau erreicht, das anspruchsvolle, mit JavaScript realisierte Effekte endlich möglich macht. Web-Anwendungen müssen den Möglichkeiten klassischer Desktop-Anwendungen in nichts mehr nachstehen und können diese sogar ohne den Einsatz von Plug-Ins erreichen, welche Usability und Barrierefreiheit (engl. Accessibility) als wichtige Web Standards untergraben würden.

### 3.2.3 JavaScriptHelper

Das Modul JavaScriptHelper (ActionView::Helpers::JavaScriptHelper) ermöglicht die Verwendung von beliebigen JavaScript Code in View-Templates. Aufrufe von JavaScript Funktionen werden dabei in Ruby Code gekapselt. Die Methode `button_to_function(name, function, html_options)` erzeugt z.B. einen Hyperlink `name`, der die JavaScript Funktion `function` mittels eines onclick Event Handlers triggert:

```
button_to_function("Greeting", "alert('Hello world!')")
```

Um JavaScript Frameworks, Libraries oder Dateien in ein View-Template einzubinden, verwendet man die Methode `<%= javascript_include_tag 'filename' %>` aus dem Modul ActionView::Helpers:: AssetTagHelper. In diesem Modul sind weitere Methoden z.B. zur Einbindung von Stylesheets (`stylesheet_link_tag()`), als auch von Bildern (`image_tag()`), definiert. Sowohl Prototype als auch script.aculo.us werden mit dem Kommando `rails` zur Erzeugung einer neuen Rails Anwendung automatisch im Verzeichnis `public/javascripts` eingefügt. Der Aufruf:

```
<%= javascript_include_tag :defaults %>
```

im HEAD Teil eines RHTML Templates referenziert beide, so dass damit die Funktionalität beider Bibliotheken innerhalb dieses Templates verwendet werden kann. Alternativ kann auch z.B. nur Prototype eingebunden werden:

---

48 s. http://prototype.conio.net

49 s. http://script.aculo.us

```
<%= javascript_include_tag ‘prototype’ %>
```

Der Methodenaufruf erzeugt die Zeile `<script src="/javascripts/prototype.js" type="text/javascript"></script>` im View-Template.

### 3.2.4 PrototypeHelper

Das Modul PrototypeHelper (ActionView::Helpers::PrototypeHelper) stellt eine Menge von Helper Methoden für die Verwendung der JavaScript Funktionen aus dem Prototype Framework zur Verfügung.

Diese Funktionalität beinhaltet auch den Aufruf von Methoden unter Verwendung von AJAX. Das bedeutet, dass man Action Methoden eines Controllers aufrufen kann, ohne eine Aktualisierung der kompletten Seite zu verursachen. So ist z.B. das Hinzufügen und Entfernen von Punkten zu einer To-do-Liste über ein Formularfeld ohne vollständiges Aktualisieren der Seite möglich.

Ein einzelner Eintrag der Liste wird über eine Instanz der Model-Klasse `Item` repräsentiert. Ohne uns um eine Datenbankanbindung zu kümmern, verwenden wir für diese Beispiel ein Array zur “Speicherung” der Einträge. Ein Eintrag besteht aus einem Text (`body`) und dem Datum des Eintrags (`posted_on`). In der Model-Klasse benötigen wir keine AJAX Unterstützung:

```
class Item
  attr_reader :body, :posted_on
  DATABASE = []
  def initialize(body)
    @body = body
    @posted_on = Time.now
    DATABASE.unshift(self)
  end
  def self.find_items
    DATABASE
  end
  # Populate with initial items
  new(”Do this!”)
  new(”Do that!”)
end
```

Anm.: Die Ruby Methode `unshift(self)` fügt hier die neu erzeugte `Item` Instanz (referenziert über `self`) an erster Position in das Array ein. `self.find_items` ist eine Klassenmethode, die das Array zurückgibt.

Das View-Template `app/views/todo_list/show_list.rhtml` enthält eine Liste zur Darstellung der Einträge und ein Formular, dessen Methode `form_remote_tag()` einen neuen Eintrag via XMLHttpRequest hinzufügt. Die Liste wird aktualisiert (`:update => “items”`), indem der neue Eintrag an erster Position der Liste eingefügt wird (`:position => :top`):

```
<ul id="items">
  <%= render(:partial => 'item', :collection => @items) %>
</ul>
<%= form_remote_tag(:url => { :action => "add_item" },
                    :update => "items",
                    :position => :top) %>
```

```
<%= text_field_tag('item_body') %>
<%= submit_tag("Add Item") %>
<%= end_form_tag %>
```

Jedes Formular einer Rails Anwendung kann AJAX einsetzen, indem man statt der Methode `form_tag()` die Methode `form_remote_tag()` aus dem Modul PrototypeHelper verwendet. `form_remote_tag()` überträgt das Formular unter Verwendung des XMLHttpRequest-Objekts im Hintergrund (anstatt der regulären Übertragung mit `action="POST"`). Obwohl JavaScript zur Serialisierung der Formularelemente eingesetzt wird, ändert sich aus Sicht der Empfängerseite, d.h. aus der Sicht der Action Methode bzw. des Controllers, nichts: die Elemente sind auch hier über die `params` Datenstruktur im Controller verfügbar. Partial Page Templates unterstützen das "Don't Repeat Yourself"-Prinzip: Die Action Methode teilt sich das Partial (im unserem Beispiel `_item.rhtml`) mit dem View. Der View benutzt das Partial, um die initiale Liste dazustellen, der Controller für alle neu erzeugten Einträge:

```
class TodoListController < ApplicationController
  def show_list
    @items = Item.find_items
  end
  def add_item
    item = Item.new(params[:item_body])
    render(:partial => 'item', :object => item, :layout => false)
  end
end
```

Das Partial Page Template `app/views/todo_list/_item.rhtml` vervollständigt die Liste:

```
<li>
  <p>
    <%= item.posted_on.to_s %>
    <%= h(item.body) %>
  </p>
</li>
```

Nicht vergessen dürfen wir, die Prototype Library einzubinden, nach Konvention am besten in das Layout-Template `app/views/layouts/todo_list.rhtml`:

```
<html>
  <head>
    <title>AJAX Examples: <%= controller.action_name %></title>
    <%= javascript_include_tag "prototype" %>
  </head>
  <body>
    <%= @content_for_layout %>
  </body>
</html>
```

Über die URL http://0.0.0.0:3000/TodoList/show_list kann man das Beispiel nun ausprobieren.

# A. Literaturverzeichnis

[THOMAS] Agile Web Development with Rails - A Pragmatic Guide
Dave Thomas, David Heinemeier Hansson, et al
©2005 The Pragmatic Programmers LLC.

[BLACK] Ruby for Rails - Ruby Techniques for Rails Developers
David A. Black
©2006 Manning Publications

[TDI] Taschenbuch der Informatik; 5., neu bearbeitete Auflage
Uwe Schneider, Dieter Werner, et al
©2004 Carl Hanser Verlag München Wien

[GoF] Design Patterns: Elements of Reusable Object-Oriented Software
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
©1995 Addison-Wesley Publishing Company

[DEA] Designing Enterprise Applications with the J2EE Platform, Second Edition
Inderjeet Singh, Beth Stearns, Mark Johnson, et. al.
©2002 Addison-Wesley Professional
(http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e)

[FOWLER] Patterns of Enterprise Application Architecture
Martin Fowler
©2002 Addison-Wesley Professional

## A.1 Quellen im Internet

Rails Rails Framework Documentation:
http://api.rubyonrails.com/

Ruby Ruby 1.8.4 Core Library Documentation:
http://corelib.rubyonrails.org/