Universität Siegen
Naturwissenschaftlich-Technische Fakultät
Lehrstuhl für Compilerbau und Softwareanalyse

Programmieren für das Web 2.x
Ein Überblick über die Webentwicklung im Jahre 2011

Bachelorarbeit

vorgelegt von
Johannes Leers
am 16.03.2012

Gutachter
Dipl-Inf. Benedikt Meurer
Privatdozent Dr. Kurt Sieber

Inhaltsverzeichnis

1 Einleitung

Diese Bachelor-Arbeit entstand im Wintersemester 2011/ 2012 an der Universität Siegen. Sie beschäftigt sich mit wichtigen Aspekten der Programmierung im Internet und zeigt, wie man heutzutage typischerweise vorgeht, um Webanwendungen zu programmieren.

Begonnen wird in Kapitel zwei mit einer kleinen Einführung in die Geschichte des Internets und besonders des World Wide Webs. Ohne die bahnbrechende Errungenschaft rund um Tim Barners Lee gäbe es heute das Internet nicht so, wie wir es kennen und lieben. Um die Jahrtausendwende entstanden die ersten Web-Anwendungen, die sich deutlich von „normalen“ Internetseiten unterschieden. Erstmals versuchte man, den Komfort klassischer Desktop-Anwendungen auf das Internet zu übertragen. Der Übergang von statischen HTML-Seiten hin zu dynamischen Anwendungen im Netz erforderte neue, ausgeklügelte Technologien, welche die HTML-Seiten dynamisch auf dem Server generieren konnten. So entstanden die ersten Web-Sprachen und mit ihnen erste Frameworks, die dem Webworker sich wiederholende Aufgaben abnehmen sollten. Langsam aber sicher wurden die Einschränkungen der statischen HTML-Seiten zu einem Problem. Man entwickelte eine Technologie, die wir heute unter dem Begriff „Ajax“ kennen, mit derer Hilfe es möglich wurde, nur bestimmte Teile einer Seite neu zu laden. Dies war vorher nur bedingt mit Frames möglich. Die Zukunft gehört den WebSockets, welche das Web von einer „Pull“- in eine „Push“-Architektur wandeln werden. Sie wurden entworfen, um eine bidirektionale Verbindung zwischen einer Webanwendung und einem Webserver herzustellen. Ohne WebSockets muss der Client in regelmäßigen Abständen Daten vom Server anfordern, was viel unnötigen Traffic zwischen Client und Server verursacht. Die WebSockets werden zum Zeitpunkt dieser Arbeit allenfalls in ganz neuen Browser-Versionen unterstützt.

Im Kapitel drei wird der Schwerpunkt dieser Arbeit gesetzt. Es wird intensiv auf die Eigenheiten und Besonderheiten des Ruby on Rails Frameworks eingegangen. Natürlich werden aber auch die Alternativen betrachtet. Nicht zu vernachlässigen ist PHP, eine Programmiersprache, die quasi als eine der ersten das Internet prägte, und deshalb im Rahmen dieser Arbeit gleich mit zwei Frameworks vertreten ist (Symfony und FLOW3). Außerdem wird Django betrachtet, welches in Python programmiert wurde und das Google Web Toolkit, welches in Java implementiert ist. Zusätzlich wird noch auf zwei ungewöhnliche Frameworks eingegangen, die sich durch ihre doch exotischen zugrundeliegenden Programmiersprachen auszeichnen: Opa ist Programmiersprache und Web-Framework zugleich, Ocsigen wird in OCaml programmiert. Zuletzt findet sich eine Vergleichstabelle aller vorgestellten Web-Frameworks.

Immer wichtiger wird heute das sogenannte „Mobile Computing“, da Desktop-Rechner oder auch Laptops immer häufiger durch mobile Endgeräte wie Smartphones oder Tablets vom Markt gedrängt werden. Die Herausforderung besteht darin, die Inhalte auf unterschiedlichsten Display-Größen ansehnlich darzustellen und zusätzlich auf Touch-Displays gut bedienbar zu machen. In Kapitel vier werden die unterschiedlichen Ansätze beleuchtet.

Ein wichtiges Thema zukünftiger Webanwendungen und vor allem Web-Spielen wird WebGL sein. Dabei werden Konzepte der 3D-Programmierung ins Internet übertragen. Das Ziel ist, aufwändige Grafiken, welche von der Hardware gerendert werden, in den Webbrowser zu bringen. Kapitel fünf liefert dazu einen kleinen Einblick.

Kapitel sechs gibt einen Überblick über den Themenkomplex des „Cloud Computings“. Interessant ist dabei, wie man heutzutage Aufgaben „in die Wolke“ auslagern kann um so Ausfallsicherheit, Redundanz und Skalierbarkeit mit Kosteneinsparungen zu paaren.

Schließen wird die Arbeit in Kapitel sieben ein kleines Fazit.

2 Präambel

2.1 Die Geschichte des Internets

Als Beginn des Internets bezeichnet man heute das Jahr 1957, als die Union der Sozialistischen Sowjetrepubliken (UdSSR) den ersten Satelliten der Welt (Sputnik) in die Erdumlaufbahn brachten [1]. Als Reaktion gründeten die United States of America (USA) 1958 das „Advanced Research Projects Agency“ (ARPA) um die verlorengegangene Vorherrschaft in Wissenschaft und Technik wieder zu erlangen. Ab 1960 verfügte das US-Militär über die ersten vernetzten Rechner, aber man erkannte auch die Anfälligkeit des Netzes: Wenn ein Knoten zerstört werden würde, wäre das gesamte Netzwerk lahmgelegt. Auch die Universitäten des Landes erkannten den Vorteil, den ihnen diese Vernetzung brachte, und so wuchs das ARPANET sehr schnell. Während das Netz im Jahre 1971 lediglich 15 Knoten besaß, waren es 1977 schon 111 Knoten und im Jahre 1987 hatte das ARPANET bereits 27000 angeschlossene Rechner.

Im März 1989 schrieb Tim Berners-Lee am CERN[1] in Genf die erste Fassung seines Aufsatzes „Information Management: A Proposal“ [3] ein Entwurf für die Entwicklung des späteren World Wide Webs (WWW). Der Grundgedanke war, den Kernphysik-Wissenschaftlern am CERN eine angemessene Kommunikationsplattform zu bieten. Ein wissenschaftliches Dokument, welches sich auf ein anderes bezieht, sollte dieses nicht nur erwähnen, sondern dieses direkt mittels eines einzigen Tastendrucks über einen elektronischen Verweis aus dem Dokument aufrufen können. Benutzer sollten von einer Internetseite zu einer beliebigen anderen springen können, unabhängig von der Entfernung oder der geografischen Lage der Rechner [4]. Anfang der Neunziger startete mit der Freischaltung der ersten privaten Rechner im Internet die Kommerzialisierung.

[1] CERN: **C**onseil **E**uropéenne pour la **R**echerche **N**ucléaire (Europäische Organisation für Kernforschung) [2]

Das militärische ARPANET wurde außer Betrieb genommen. Ab 1993 feierte das WWW endgültig seinen Durchbruch: Es wurde ab nun auch außerhalb des CERN eingesetzt und ersetzte mit der Zeit viele andere etablierte Dienste, unter anderem auch Gopher [5].

Anstelle der örtlich nächstgelegenen Computer konnte man nun diejenigen nutzen, die für die momentane Problemstellung am besten geeignet waren, und mit denjenigen Menschen kommunizieren, die für das aktuelle Problem die besten Lösungsvorschläge anbieten konnten. Der Siegeszug des Netzes machte es Dozenten und Studenten möglich, mit weit entfernt lebenden Kollegen zusammenzuarbeiten, deren Forschungsinteressen sie teilten. So entstand auch die Programmiersprache Lisp als Gemeinschaftsprojekt. Viele geographisch weit verstreut liegende Gruppen arbeiteten bei der Entwicklung zusammen [6]. Heute wird das Internet manchmal sprachlich mit dem World Wide Web gleichgesetzt, was rein technisch nicht ganz korrekt ist, da das Internet noch viele andere Dienste wie FTP oder E-Mail bietet.

2.2 Die Entwicklung des World Wide Webs

Die ersten Ideen des World Wide Webs gehen zurück auf Vannevar Bush, der 1945 in seinem Artikel „As We May Think“ eine Maschine namens Memex erdachte, die als Gehirn-Erweiterung unter anderem Links zwischen Dokumenten folgen konnte [7]. Ted Nelson prägte in den Sechzigern den Begriff des „Hypertexts“ in seinem Artikel „A File Structure for the Complex, the Changing, and the Indeterminate“ [8]. Der Hypertext, das dem WWW zugrunde liegende Konzept, ist bis heute quasi erhalten geblieben und wurde lediglich etwas verändert. Es soll erwähnt sein, dass es auch andere Hypertext-Systeme gab, wie zum Beispiel das Projekt Xanadu [9], welches von Nelson selber entworfen wurde. Dieses konnte sich jedoch nicht gegen das wesentlich einfacher gestaltete WWW durchsetzen. Dies war unter anderem dem freien Protokoll geschuldet,

welches freie WWW-Server und Clients ohne Beschränkungen durch Lizenzen ermöglichte. Marc Andreessen vom NCSA[2] veröffentlichte im Jahre 1993 den Browser „Mosaic for X“ [11], welcher dem gesamten Internet ein explosionsartiges Wachstum bescherte. Andreessen gründete später die Firma „Netscape Communication“ und entwickelte den „Netscape Navigator“. Das gesamte WWW basiert auf der „HyperText Markup Language“ (zu deutsch Hypertext-Auszeichnungssprache, wird oft kurz als Hypertext oder HTML bezeichnet). Dabei hat sich die Grundidee des Webs seit seinen Anfängen nicht sonderlich geändert. Wie Dr. Robert Cailliau, der Co-Entwickler des WWW sagte, geht es darum Links zwischen Dokumenten zu bilden, Formulare auszufüllen und Reaktionen von einer Datenbank zu bekommen [12].

2.3 Die Browserkriege

Was nun aus Sicht der Anwender folgte, ging als „Browserkrieg“ in die Geschichte ein, in dem verschiedene Browser-Hersteller ihre Produkte vertreiben wollten. Während der Netscape Navigator Mitte der Neunziger das WWW nahezu beherrschte und bis zu 80% aller im Internet verzeichneten Zugriffe auf das Konto dieses Browsers gingen [13], schickte sich Microsoft ab diesem Jahr an, Netscape durch den eigenen Browser „Internet Explorer“ [14] Konkurrenz zu machen. Microsoft hatte 1994 eine Generallizenz des Mosaic Browsers erworben und konnte auf dieser aufbauen. Noch bis zur Version 6 trug der Microsoft Internet Explorer den Hinweis „Basiert auf NCSA Mosaic“.

In Redmond befürchtete man, dass der Netscape Navigator sich zu einer sogenannten „Middleware“ entwickeln könnte, auf der mit Hilfe von Java beliebige Programme plattformunabhängig laufen würden. Damit wäre aber Microsofts Betriebssystem-Monopol gefährdet gewesen, und so galt es, den alternativen Browser mit aller Kraft zu bekämpfen und Marktanteile abzunehmen.

[2] NCSA: **N**ational **C**enter for **S**upercomputing **A**pplications [10]

Letztendlich gelang dies Microsoft durch die Bündelung des Internet Explorers mit seinem Betriebssystem „Microsoft Windows“ [15].

Von 1995 bis Anfang des neuen Jahrtausends sank der Marktanteil des Netscape Navigators von über 80% auf unter 4%, während der Marktanteil des Internet Explorers im selben Zeitraum von unter 3% auf über 95% stieg. Ab Januar 1998 gab Netscape den Navigator kostenlos ab und veröffentlichte den Quelltext des Browsers als Open Source. In dem hieraus entstandenen Projekt Mozilla [16] wurde das Programm vollständig neu geschrieben. Die Folgen des Monopols des Browsers von Microsoft waren auf den Internet Explorer „optimierte“ Webseiten, deren Betreiber verschiedene Techniken nutzten, um die vielen Bugs des Browsers auszugleichen. Webstandards, wie sie vom World Wide Web Consortium vorgegeben werden, wurden von den Entwicklern kaum mehr beachtet. Anstatt dessen wurden Webseiten so programmiert, dass sie im Internet Explorer „richtig“ angezeigt wurden. Dies änderte sich erst wieder im sogenannten zweiten Browserkrieg, der Ende 2003 begann. Ausgelöst durch die chronischen Sicherheitsprobleme und der nicht vorhandenen Weiterentwicklung des Internet Explorers wechselten viele Nutzer auf die neuen Aufsteiger. Besonderes Augenmerk durch die ständig steigende Beliebtheit lag auf Mozilla Firefox [17]. Aber auch andere Browser wie Opera, Konqueror oder Safari konnten in den Benutzerstatistiken zulegen [18]. Im Jahr 2011 wird der Internet Explorer weltweit zwar immer noch von den meißten Anwendern benutzt, die Alternativen gewinnen aber immer mehr Einfluss. Auch stellte Google 2008 mit dem „Google Chrome“ einen eigenen Browser vor [19].

Spätestens mit dem Beginn des aktuellen Jahrzehnts nehmen mobile Browser eine ernstzunehmende Stellung im globalen Browsermarkt ein [20]. Laut einer Pressemitteilung von StatCounter verdoppelt sich der Gebrauch mobiler Browser von Jahr zu Jahr [21]. Obwohl bisher erst 10% des globalen Traffics auf mobile Endgeräte zurückgeht, ist das Tempo des Wachstums bemerkenswert.

Auf mobilen Endgeräten sind laut StatCounter die verschiedenen Browser von Opera führend. Außerdem nimmt die Benutzung des Android-Browsers zu, während die des Blackberry- und des Nokia-Browsers abnimmt.

2.4 Die Vorteile der Web-Programmierung

Nun lässt sich natürlich die Frage nach der Motivation stellen, warum die Verlagerung – auch ehemals ausschließlich nativer Programme wie E-Mail-Clients – ins Web, und damit letztendlich die Web-Programmierung, in den letzten Jahren so stark zugenommen hat. Schließlich konnten Programme vorher auch naturbedingt auf den unterschiedlichsten Betriebssystemen betrieben werden. Genau dies ist allerdings auch schon der größte Nachteil der nativen Programme, da sie für viele verschiedene Betriebssysteme einzeln programmiert bzw. kompiliert werden müssen. Webanwendungen verhalten sich wesentlich pflegeleichter: Im Idealfall müssen sie einmalig erstellt werden und können dann auf allen Betriebssystemen, auf jeder Hardware und mit jedem Browser betrieben werden. Durch das verbreitete Aufkommen der mobilen Endgeräte wurde dieser Vorteil noch weiter gestärkt. Natürlich hat auch die Web-Programmierung ihre Nachteile, wie zum Beispiel den Umstand, dass unterschiedliche Browser einige HTML-Elemente unterschiedlich anzeigen.

Ein weiterer Vorteil von Webanwendungen ist das schnelle Deployment. Während native Anwendungen über Update-Mechanismen verfügen müssen, können Webanwendungen transparent im Hintergrund aktualisiert werden, ohne dass der Benutzer etwas davon mitbekommt. Es ist ein recht komplizierter Weg vom Quellcode zu einer fertigen, nativen Anwendung. Das Deployment benötigt eine ganze Reihe von einzelnen Schritten, die unter anderem aus dem Kompilieren und dem Packaging für verschiedene Systeme bestehen können. Auf manchen Systemen, wie z.B. solchen, die dem „Apple-Universum“ angehören, ist zusätzlich ein Freigabe-Prozess nötig, auf den der Entwickler kaum

bis gar nicht Einfluss nehmen kann [22]. So kann es passieren, dass Anwendungen nach vielen Stunden Entwicklung von Apple abgewiesen werden. Außerdem sind alle Freigabeschritte auch nach den kleinsten Updates am Quellcode zu wiederholen.

2.5 Unterschiede zwischen Webseiten und Webanwendungen

Im Laufe der Zeit haben sich die Webseiten der ersten Generationen im Vergleich zu heutigen Webanwendungen stark verändert. In der Anfangszeit des Webs wurden Webseiten oft in sogenannten „What You See Is What You Get“-Editoren (WYSIWYG) erstellt, in denen man, ähnlich zu Textbearbeitungsprogrammen, das Ergebnis sofort betrachten konnte. Der Inhalt war bei jedem Aufruf dieser Seiten gleich und Änderungen konnte nur der Webentwickler durch Editieren des Quelltextes vornehmen. Man sprach weniger vom „Programmieren“, sondern eher vom „Schreiben“ von HTML. Die Seiten waren üblicherweise statisch, d.h. dass keine Bearbeitung durch den Server stattfand sondern dass der HTML-Quelltext so zum Client übertragen wurde wie er auf dem Server abgelegt war. Professionelle Seiten sind üblicherweise aber dynamisch. Erst zur Laufzeit erzeugt ein Programm auf dem Server die Ausgabe, welche an den Client gesendet wird. Als Beispiel sei eine Suchmaschine genannt. Der Inhalt der einzelnen Seiten verändert sich quasi mit jedem Aufruf, da die Ergebnisseiten immer abhängig von der Anfrage sind. Webanwendungen werden dementsprechend programmiert, wohin im Gegensatz erstgenannte statische Webseiten nach dem Baukastenprinzip zusammengesetzt werden. Interaktion mit dem Benutzer wie z.B. Formulare sind auf statischen Seiten nicht möglich, da die gesendeten Daten auf dem Server nicht verarbeitet werden können. Dieser liefert Daten nur aus, kann einkommende Daten, wie sie durch Formulare naturgemäß generiert werden, aber nicht verarbeiten.

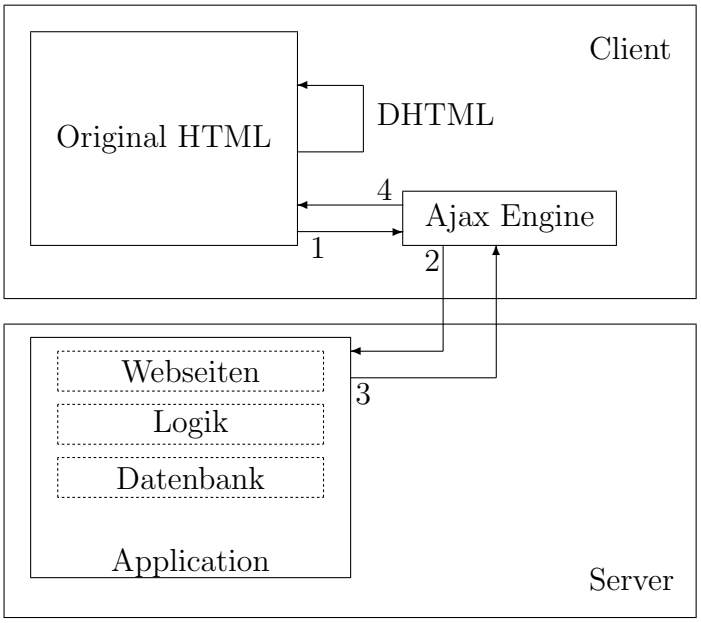

Abbildung 1: Eine DHTML-Anweisung verlässt im Gegensatz zu einem Ajax-Request nie den Client.

Um statischen HTML-Quellcode zu verändern muss zwingend programmiert werden, wozu man JavaScript, welches in seiner Variante als ISO-Norm auch als ECMAScript bezeichnet wird, einsetzt. Das Verändern einer statischen Seite wird weitläufig als „dynamic HTML“, „DHTML“ oder auch „DOM-Scripting“ bezeichnet. Oft wird der Begriff DHTML aber auch als „Erfindung von Marktstrategen“ bezeichnet, da er keinesfalls eine HTML-Erweiterung oder eine neue Sprache darstellt, sondern vielmehr einen Sammelbegriff für verschiedene Lösungen beschreibt, mit der das Aussehen einer Seite nach dem initialen Laden geändert werden kann [23]. Da DHTML in JavaScript programmiert wird, lassen sich, wie man in Abbildung 1 sieht, lediglich optische Effekte erzielen und es kann nicht auf den Server zurückgeschrieben werden.

3 Web-Frameworks

3.1 Das Zusammenspiel von Client und Server

Wie im vorigen Kapitel beschrieben, bestehen moderne Webanwendungen nicht nur aus statischem HTML, welches an den Client ausgeliefert wird. Das HTML wird „on the fly“ generiert und im Browser gerendert. Welches HTML ausgeliefert wird, entscheidet das Programm welches auf dem Server läuft. Typische Sprachen für Server-Programmierung sind PHP, Perl, Java, Python oder Ruby, allerdings setzt man oft auf Frameworks, welche dem Entwickler bei der Programmierung komplexer Webseiten behilflich sein können. Einige Vertreter dieser Art werden später vorgestellt.

Mit reinem HTML ist es nicht möglich die Seite zu verändern die der Server ausliefert. Hierfür braucht man zwingend JavaScript, mit dem man das „Document Object Model“ (DOM) [24] verändern kann. Nur so kann man Elemente verschieben, editieren, löschen oder hinzufügen. JavaScript ist eine eigenständige, objektorientierte Programmiersprache, die es dem Webentwickler erlaubt, clientseitig Quellcode ausführen zu lassen [25]. Dazu setzt man auch in der Client-Programmierung typischerweise auf Frameworks. Als Vorreiter wären jQuery [26], MooTools [27] und Prototype [28] zu nennen, wobei jQuery am häufigsten eingesetzt wird [29] und auf das deshalb im späteren Verlauf auch gesondert eingegangen wird.

Beim Einsatz von JavaScript oder jeglicher JavaScript-Frameworks sollte man sich aber auch immer bewusst sein, dass der Benutzer die Script-Sprache im Browser deaktiveren kann. Deshalb sollte man sich nie darauf verlassen, dass der Programmcode auf den einzelnen Clients ausgeführt werden kann. Während einige Webseiten für diesen Fall eine Fallback-Lösung in reinem HTML anbieten, verzichten andere inzwischen darauf. Sie setzen JavaScript

voraus und verweigern einfach den Dienst sollte dies nicht der Fall sein (z.B. der Google Reader [30]).

Um Daten zurück zum Server zu senden, wird oft „Asynchronous JavaScript and XML“ (Ajax) eingesetzt: Mit Hilfe von JavaScript werden HTTP-Anfragen in einem Hintergrundprozess an den Server gesendet, während die HTML-Seite angezeigt wird. Mit dem Ergebnis kann die Seite mit Hilfe von JavaScript verändert werden, ohne dass sie komplett neu geladen werden muss. Diese Technologie ist also eine Kombination aus Client- und Server-Programmierung. Sie wird verwendet, um ein desktopähnliches Verhalten zu simulieren. Desktop-Applikationen zeichnen sich oft durch ihre besonders gute Reaktionszeit aus. Mithilfe der Ajax-Technologie ist es z.B. möglich, einen Chat zu realisieren, der einzelne Zeilen der Unterhaltung nachladen kann, ohne die ganze Seite neuladen zu müssen.

Ein ganz neuer Ansatz sind die sogenannten WebSockets, die in HTML5 Verwendung finden [31]. Das WebSocket-Protokoll ist ein auf TCP basierendes Netzwerkprotokoll, das entworfen wurde, um eine bidirektionale Verbindung zwischen einer Webanwendung und einem Webserver herzustellen. Ohne WebSockets muss der Client die Daten in regelmäßigen Abständen anfordern, wodurch unnötiger Traffic zwischen Client und Server entsteht. Diese „Pulls“ werden oft in einem Abstand von wenigen Sekunden ausgeführt, so dass ein nicht zu unterschätzender Rechenaufwand sowohl auf Client- als auch Server-Seite entsteht. Eine bidirektionale Verbindung hat den Vorteil, dass Daten vom Server zum Client übertragen werden können. Diese „Pushs“ geschehen also nur auf Aufforderung und eliminieren damit die soeben genannten Nachteile der Pull-Architektur. Die WebSocket-Protokoll-Spezifikation [32] definiert zwei neue URI-Schemen, „ws:“ für unverschlüsselte und „wss:“ für verschlüsselte Verbindungen. Zu Beginn jeder Verbindung führen Server und Client einen sogenannten „Handshake“ durch, welcher vom Aufbau her dem HTTP-Header

ähnelt und zu diesem vollständig kompatibel ist. Dies ermöglicht die Nutzung des Standard-HTTP-Ports zugleich für normale HTTP-Kommunikation als auch für die WebSocket-Nutzung. Die so aufgebaute Verbindung kann im folgenden aktiv vom Server verwendet werden um Daten zum Client zu senden. Der Server muss also nicht mehr Anfragen vom Client abwarten, sondern kann neue Informationen ohne Zeitverzug ausliefern. Als zumindest temporärer Nachteil zählt, dass der Internet Explorer WebSockets erst ab Version 10 unterstützt [33]. Obwohl andere Browser keine Probleme mit der Implementierung haben, wird von einem produktivem Einsatz auf massentauglichen Internet-Seiten zurzeit abgeraten. Anwendungsmöglichkeiten gäbe es indes einige: Neben des klassischen Beispiels des Chats (oder genereller kollaborative Webservices) würden auch Spiele auf HTML5-Basis oder Finanzanwendungen von WebSockets profitieren.

3.2 Die Vorteile von Web-Frameworks

Frameworks haben in den letzten Jahren die Entwicklung großer Webanwendungen sehr geprägt. Letztendlich ist es sogar egal, welches Framework ein Entwicklerteam wählt: Es ist eine Entscheidung persönlicher Vorlieben und des Vorwissens welches bei den Entwicklern vorhanden ist. Mit Hilfe eines Frameworks lassen sich sowohl wiederkehrende Aufgaben vereinfachen, als auch Code leichter wiederverwenden. Außerdem soll die Selbstdokumentation der Software-Entwicklung gefördert werden. Das Selbstdokumentations-Prinzip prägte Bertrand Meyer [34]. Es sagt aus, dass Software-Entwickler danach streben sollten alle Informationen über ein Modul Teil des Moduls selbst werden zu lassen. Damit ist gemeint, dass die Dokumentation von Code direkt im Code stehen sollte und nicht in andere Dateien ausgelagert werden sollte. Gerade bei kleinen Änderungen im Code werde oft die Dokumentation vergessen, wenn sie nicht direkt bei dem betroffenen Code stehe, wodurch es

sehr schnell zu Inkonsistenzen zwischen Code und Dokumentation kommen könne. In der Praxis bedeutet dies aber auch, dass aus diesen Informationen zusätzliche verwandte Dokumente erstellt werden können, die auf dem Quellcode und dessen Dokumentation basieren. So kann z.B. automatisch die Beschreibung einer API in HTML erstellt werden [35].

Entwickler stehen vor der Aufgabe performante und sichere Anwendungen zu erstellen, die einen Komplexitätslevel erreicht haben, der vor Jahren ohne ein Framework nur mit unverhältnismäßig mehr Aufwand erreicht werden konnte. Die Zeiten neigen sich dem Ende zu, in denen Entwickler alle Technologien ihrer Plattform perfekt beherrschten. In nahezu jedem Projekt sind heute verschiedenste Kenntnisse notwendig. Um Anwendungen zu schreiben, die den heutigen Standards genügen, sind auf der Client-Seite tiefgreifende JavaScript- und CSS-Erfahrungen notwendig. Auf der Server-Seite benötigt man Wissen darüber, wie man Anwendungen sicher vor Angriffen schützt, „Session Handling“ und Zugriffe auf Datenbanken effektiv programmiert und Komponenten-Tests für bestimmte Anwendungsteile schreibt. Dabei sollte das Projekt und der Quellcode so strukturiert werden, dass auch nach Monaten und Jahren noch Erweiterungen und Wartungsarbeiten einfach möglich sind. Letztendlich muss all dies in aller Regel auch immer vor dem Hintergrund eines Zeit- und Kostenrahmens für die Entwicklung geschehen. Den meisten Entwicklern sitzt ein Chef oder Kunde im Nacken, der auf die Einhaltung von Zeit- und Kostenplänen drängt. Das Profil eines typischen Webentwicklers hat sich nicht zuletzt deshalb in den letzten Jahren grundlegend geändert.

Obwohl die ersten Frameworks immer schneller zu größeren, unhandlicheren und, auch für den Entwickler, zu unflexiblen Rahmen heranwuchsen, ging aus eben diesen Werken eine neue Generation Web-Frameworks hervor, von denen hier eine Auswahl vorgestellt wird. Diese eignen sich sowohl zur Programmierung in JavaScript für die Client-Seite als auch in unterschiedlichen Sprachen

für die Programmierung des Servers. Oft werden Frameworks für Clients und Server auch gemeinsam benutzt, da sie sich gegenseitig nicht ausschließen. Ganz im Gegenteil benötigen Frameworks für Server oft die Unterstützung eines Client-Frameworks um performante Ajax-Aufrufe umsetzen zu können.

3.3 Ein Framework für Clients: jQuery (JavaScript)

Das klassische Beispiel der Client-Programmierung im Web ist die DOM-Manipulation eines HTML-Dokuments mit Hilfe von JavaScript. Betrachtet werde dazu das in Listing 1 gegebene Ausgangsdokument, welches im Wesentlichen aus einem `h1`-Titel und zwei Paragraphen (`p`) besteht.

```
<html>
  <head>
    <title>jQuery Test</title>
    <script type="text/javascript" src="http://ajax.googleapis.com
        /ajax/libs/jquery/1.6.2/jquery.min.js"></script>
  </head>
  <body>
    <h1>Titel</h1>
    <p>
      <button onclick="$('p').after('<h2>Untertitel</h2>');">
        Untertitel hinzufuegen
      </button>
    </p>
  </body>
</html>
```

Listing 1: jQuery Beispiel

Benutzt wird hier jQuery 1.6.2, aus dem Google „Content Distribution Network“ (CDN) [36]. Der Vorteil dieser Methode ist, dass die Bibliothek nur einmal geladen werden muss und danach vielen anderen Seiten ebenfalls zur Verfügung steht, wenn sie die Bibliothek mit dem gleichen Pfad einbinden.

Wenn man nun auf den Button klickt, wird ein Untertitel (`h2`) nach dem `h1` eingefügt. Die jQuery-Funktion kann mit der `$`-Funktion abgekürzt werden. Sie akzeptiert eine Zeichenkette, welche einen „CSS-Selektor“ beinhaltet, der dann benutzt wird, um eine Gruppe von Elementen zu wählen [37].

```
<html>
  <head>
    <title>JavaScript Test</title>
    <script type="text/javascript">
      function append_h2() {
        var h1 = document.getElementsByTagName('h1')[0];
        var h2 = document.createElement('h2');
        h2.innerHTML = 'Untertitel';
        h1.parentNode.appendChild(h2);
      }
    </script>
  </head>
  <body>
    <h1>Titel</h1>
    <p>
      <button onclick="append_h2();">
        Untertitel hinzufuegen
      </button>
    </p>
  </body>
</html>
```

Listing 2: JavaScript Beispiel

Um die Vorteile eines JavaScript-Frameworks wie jQuery herauszustellen, wird in Listing 2 gezeigt, wie die DOM-Manipulation in reinem JavaScript programmiert wird. Wie man sieht ist diese Implementierung wesentlich aufwändiger. Einer der Vorteile eines JavaScript-Frameworks ist somit, dass viel weniger

Code entwickelt werden muss und dieser damit automatisch weniger Fehler enthält.

```
<script type="text/javascript">
  function browsercheck() {
    var anfrage = null;
    try {
      anfrage = new ActiveXObject("MSXML2.XMLHTTP");
    }
    catch(Error) {
      try {
        anfrage = new ActiveXObject("Microsoft.XMLHTTP");
      }
      catch (Error) {
        try {
          anfrage = new XMLHttpRequest();
        }
        catch (Error) {
          alert("Fehler beim Erzeugen des Objektes");
        }
      }
    }
    return anfrage;
  }
</script>
```

Listing 3: Ajax Beispiel

Mit der Browserunabhängigkeit existiert aber noch ein weiterer, entscheidender Vorteil. Dies kann man in Listing 3 an einem Ajax-Request sehen. Mit reinem JavaScript müsste bei jedem Besucher der aktuell benutzte Browser ermittelt werden, um bei der Benutzung der Ajax-Funktionalität auf seine besonderen Eigenheiten eingehen zu können. Erst nachdem ermittelt wurde, wie Ajax-Requests in dem spezifischen Browser umgesetzt werden, könnte ein Ajax-

Request abgesetzt werden. Nach der Initialisierung könnte ein Request auf der `anfrage`-Variable abgesetzt werden: `anfrage.open("GET","/test.cgi", true);` Die `open()`-Methode bildet ein `XMLHttpRequest`-Objekt, welches dann zusammen mit der `send()`-Methode einen gültigen HTTP-Request ausführen kann.

Wenn man allerdings ein JavaScript-Framework wie jQuery benutzt, fällt dieser Mehraufwand weg. Sobald die jQuery-Bibliothek im HTML-Header eingebunden ist, kann der erste Ajax-Request abgesetzt werden: [38]: `$.get("/test.php");` Die Überprüfung, welcher Browser verwendet wird, und damit auch welcher Request auszuführen ist, findet hier transparent im Hintergrund statt, und muss vom Entwickler nicht jedesmal neu implementiert werden.

3.4 Ein Framework für Server: Ruby on Rails (Ruby)

Kommen wir nun zum Schwerpunkt dieser Arbeit: Ruby on Rails [39] wird als Vertreter aktueller Frameworks für Server ausführlich vorgestellt und auf einzelne Details dieser Software besonders eingegangen. Ruby on Rails (kurz: „RoR“ oder „Rails“) ist ein Web-Framework für die Programmiersprache Ruby [40].

Geschichte

David Heinemeier Hansson, der als der Initiator von Ruby on Rails gilt, war 2003 bei 37signals [41] angestellt und entwickelte dort Basecamp [42], eine webbasierte Projekt-Management-Software. Aus dieser Software extrahierte er 2004 das Rails-Framework. Er veröffentlichte es als Open Source und vergab Commit-Rechte im Februar 2005. Heute steht das Framework unter der MIT-Lizenz. Im Gegensatz zu Frameworks und Spezifikationen, die auf der „grünen

Wiese“ entstehen oder in Gremien erarbeitet werden, hat Rails seine Praxistauglichkeit bereits erwiesen.

Ruby

Wie schon erwähnt, basiert Ruby on Rails auf Ruby, einer Programmiersprache, die Mitte der neunziger Jahre von Yukihiro Matsumoto entworfen wurde [43]. Bereits 1995 vermischte der Japaner Teile seiner Lieblingssprachen (Perl, Smalltalk, Eiffel, Ada und Lisp) und formte daraus eine neue Programmiersprache, in der funktionale und imperative Programmierung ausbalanciert sind. Damit ist die Sprache so alt wie Java, allerdings rückte sie erst 2001 in den internationalen Fokus als die ersten englischsprachigen Artikel erschienen, die sich mit der Sprache befassten. Sie zeichnet sich durch ihre verständliche Syntax und erwartungskonforme Semantik aus. Programme können mit wenig Code geschrieben werden und dennoch einen großen Funktionsumfang aufweisen. Der geschriebene Quellcode ist daher auch noch Monate später zu lesen und zu verstehen. Da Ruby dynamisch getypt ist, entfällt die Zeit für Übersetzung und Deployment. Ein unmittelbares Feedback jeder Änderung ist das Ergebnis, welches aber auf Kosten einer fehlenden statischen Typsicherheit erkauft werden muss. Die Sprache erlaubt die nachträgliche Änderung von bestehenden Klassen, wodurch gezielt das Verhalten von Core-Komponenten verändert werden kann. Dieses Verhalten wird als „Monkey patching“ bezeichnet. Bis 2004 mit dem Erscheinen von Ruby on Rails fristete die Sprache allerdings ein Nischendasein. Die Erfindung von Rails wird heute als Wendepunkt in der Ruby-Geschichte angesehen.

Model, View, Controller-Architektur

Wie man in Abbildung 2 sehen kann, basiert Rails auf einer „Model, View, Controller“-Architektur (MVC) [44]. Diesen Begriff führte Trygve Reenskaug

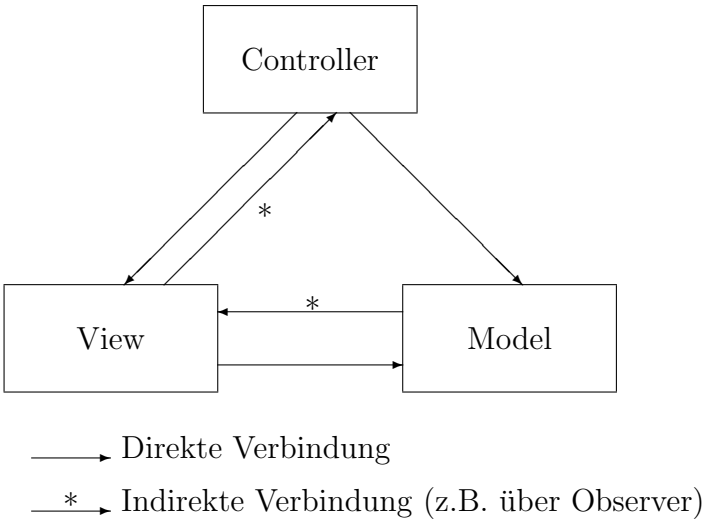

Abbildung 2: Model, View, Controller-Architektur

1979 in dem Paper „Applications Programming in Smalltalk-80: How to use Model–View–Controller“ [45] ein, als er bei Xerox PARC an Smalltalk arbeitete. Die Trennung der einzelnen Schichten führt automatisch zu einer klaren Trennung der Verantwortlichkeiten („Separation of Concerns“) und zu einer Verringerung von Abhängigkeiten im Code, so soll z.B. die Business-Logik vom User-Interface getrennt werden. Nur wenn klar definiert ist wohin bestimmte Code-Teile gehören, kann man langfristig wartbare Applikation erstellen. Das „M“ steht in MVC für „Model“, also die Datenbankabstraktionsschicht. Ein Model repräsentiert die Daten, die die Anwendung bearbeiten kann und legt fest, wie sie diese manipulieren kann. Im Normalfall repräsentiert dabei ein Model der Applikation eine Tabelle der Datenbank. Der Großteil der Business Logik ist konzentriert in den Models zu finden. Rails liefert dazu „ActiveRecord“ mit, welches später noch genauer betrachtet wird. Das „V“ steht für „View“ und wird im Rails-Framework zusammen mit dem „C“ für „Controller“ in der „ActionPack“-Bibliothek ausgeliefert. Views repräsentieren das User-Interface der Applikation. In Rails bestehen die Views oft aus HTML-

Dateien mit eingebettetem Ruby Code. Aufgabe der Views ist es aber auch, mit Hilfe von Formularen Daten vom Webbrowser zurück an den Webserver zu leiten. Natürlich sind Views auch in anderen Ausgabeformaten wie JSON oder XML denkbar [46, 47, 48]. Controller liefern den „Glue-Code“ zwischen Models und Views, also die Verbindung zwischen ihnen. Sie steuern in gewissem Sinn den Kontrollfluss innerhalb der Anwendung: Der Applikations-Server hat die Aufgabe jeder einkommenden Anfrage anhand der URL einen eindeutigen Controller zuzuweisen, der wiederum die Models nach Daten fragt und die Informationen zur Repräsentation in den Views aufbereitet.

Convention over Configuration

Rails ist von Haus aus so ausgelegt, dass jede Anwendung ohne umfangreiche Konfiguration auskommt. Dieses Paradigma nennt man „Convention over Configuration“ (englisch für „Konvention vor Konfiguration“). Man setzt u.a. auf Namenskonventionen. So werden z.B. die anzuzeigenden HTML-Dateien anhand der aufgerufenen URL automatisch bestimmt [49]. Außerdem stehen in jedem Rails-Projekt die Zugangsdaten für die Datenbank im selben Pfad in der gleichen Datei [50]. Den Ursprung dieses Konzepts zu lokalisieren gestaltet sich schwierig. Sicher ist, dass es schon in den Wurzeln einiger Java-Bibliotheken vorhanden ist. Als Beispiel sei auf das Kapitel „Simplicity“ der JavaBeans-Spezifikation in der Version 1.01 aus dem Jahr 1997 verwiesen [51]. Die Vorteile dieser Methodik liegen auf der Hand: Neue Entwickler finden sich recht zügig auch in schon gewachsenen Projekten zurecht. Sobald sie die Konventionen verstanden und verinnerlicht haben, können sie am Projekt mitarbeiten ohne sich Gedanken um Konfigurationen machen zu müssen.

Don't Repeat Yourself

Rails setzt das „Don't Repeat Yourself"-Prinzip (DRY) in allen Bereichen um. Es besagt, dass Wissen nur eine einzige, eindeutige Repräsentation in einem System hat. Geprägt haben dieses Prinzip David Thomas und Andrew Hunt [52]. Rails braucht zum Beispiel für eine Datenbanktabelle weder korrespondierende Attribute noch müssen Getter- und Setter-Methoden im Domain-Objekt definiert werden, wie dies in anderen objektorientierten Sprachen wie Java üblich ist. Diese Redundanz entfällt, da Rails entsprechende Attribute und Methoden automatisch erzeugt. Die Unterschiede zwischen der beiden Prinzipien „Convention over Configuration" und „Don't Repeat Yourself" sind hierbei fließend, denn das eine schließt das andere mit ein.

Representational State Transfer

Der „Representational State Transfer" (REST) ist ein Architekturstil für das World Wide Web, welcher im Jahre 2000 von Roy Thomas Fielding beschrieben wurde [53]. Schnittstellen zwischen Systemen können so auf eine überschaubare und – bezüglich des zu erwartenden Verhaltens – standardisierte Menge von Aktionen zurückgeführt werden. Welche Aktionen das sind, legt Fielding nicht fest, allerdings haben sich im Web bestimmte Verben durchgesetzt. So benutzt der Client einen `GET`-Request um eine Ressource vom Server anzufordern, einen `POST`-Request um eine Ressource anzulegen, einen `PUT`-Request um eine Ressource zu ändern und einen `DELETE`-Request um eine Ressource zu löschen. Eine Ressource ist als eine URL-adressierbare Entität zu verstehen, mit der über HTTP interagiert werden kann. Leider haben sich diese Verben in der Praxis nicht durchsetzen können: So wurden vom „World Wide Web Consortium" (W3C) sowohl in der HTML-Spezifikation in Version 4.01 [54] als auch in der neuen Version 5 [55] nur die beiden Verben `GET` und `POST` vorgesehen. Es hat sich daher unter den Entwicklern eingebürgert, diese zwei erlaubten

Verben zu nutzen und die restlichen Verben durch ein „Hidden Field“ umzusetzen. Dies bedeutet, dass Rails ein `PUT`- und `DELETE`-Verhalten emulieren kann, indem es in dem betreffenden `POST`-Formular ein verstecktes Feld namens `_method` mitsendet, welches das eigentlich gewünschte Verb wie z.B. `PUT` enthält. Wenn Rails `POST`-Daten verarbeitet, wertet es zuallererst diese besondere Variable aus und verhält sich dann so als wäre der aktuelle Request so wie in dieser Variable vorgegeben [56].

Create, Read, Update, Delete

Mit Hilfe des „Create, Read, Update, Delete“-Ansatzes (CRUD), welcher oft fälschlicherweise mit REST gleichgesetzt wird, kann man „schöne URLs“ erstellen. CRUD besagt, dass man anhand der URL erkennen kann, welche Aktion auf dem Server ausgeführt werden wird. URLs genügen immer dem gleichen Format, d.h. eine URL beginnt mit dem Controller, welchem die Action und dann eventuell die ID der referenzierten Ressource folgt. So lauten URLs z.B. `/posts/new`, wobei hier `posts` der Controller und `new` die Action ist. Ein anderes Beispiel ist `/posts/edit/12345`, wobei `posts` wieder der Controller und `edit` die Action ist. Zusätzlich wird mit der `12345` eine bestimmte Ressource referenziert. Sowohl REST als auch CRUD lösen die Fragen, welche Ressource in der Anfrage bearbeitet werden soll, und außerdem was mit dieser Ressource geschehen soll. Der wirkliche Unterschied besteht darin, dass CRUD dieses Verhalten in der URL implementiert, und REST dies in dem Request durch das Senden eines Verbs umsetzt [57]. Natürlich lassen sich diese Ansätze sehr gut vermischen, was die Rails-Entwickler auch taten. Man begann darüber nachzudenken, wie man die riesigen Actions in den Controllern ausdünnen konnte. Wenn man sich beispielsweise einen Blog-Controller anschaut, der die Methoden `new_post`, `edit_post`, `new_comment` und `edit_comment` mitbringt, kann man diesen in zwei Controller `Post` und `Comment` aufteilen. Diese neuen

Controller bekämen nach CRUD-Standard jeweils eine `new`- und eine `edit`-Methode. Dies war ein klarer Ansatz um dem Controller-Design seine Komplexität zu nehmen. Durch CRUD-orientierte Controller verschmolzen Controller und Ressource zu einer Einheit, so dass jeder Controller für die Manipulation von genau einem Ressourcentyp zuständig ist.

Scaffolding

Gerade im Anfangsstadium einer Anwendung ist es wichtig, schnell zu präsentierbaren Ergebnissen zu kommen. Hier spielt das Scaffolding eine große Rolle, welches in Rails-Anwendungen gerne benutzt wird [58]. Dabei werden Datenbank-Modelle, Controller und Views generiert, die zusammen die Erzeugung, Anzeige, Bearbeitung und Speicherung von Modellen ermöglichen. Nur in den seltensten Fällen bleiben diese Gerüste bis zur Auslieferung bestehen, da sie nur sehr rudimentäre Funktionen liefern können. Es ist daher üblich die Anwendung sukzessiv um individuelle Funktionalitäten zu erweitern und so die Anwendung zu jedem Zeitpunkt in einem voll funktionsfähigen Zustand vorzufinden. Obwohl das Konzept des Scaffoldings von Rails geprägt wurde, finden sich die Wurzeln in Konzepten wie „Rapid Prototyping“ oder „Rapid Application Development“.

Active Record

Active Record ist ein eigenständiges Architekturmuster für objektorientierte Software, die Daten in einer relationalen Datenbank speichert. Der Begriff wurde von Martin Fowler [59] geprägt. Er definiert einen „Active Record“ als ein Objekt welches einer Zeile einer Datenbanktabelle entspricht. Der Datenbank-Zugriff wird von diesem Objekt abgekapselt und eine Domain-Logik zu diesem Objekt hinzugefügt. So stellt jedes Objekt Schnittstellen zum Einfügen, Ändern und Löschen bereit, welche dies an die darunterliegende Datenbank

weitergibt. Neben der Ruby-Implementierung „ActiveRecord“[3] [60] existieren weitere für viele verschiedene Programmiersprachen und je nach Implementierung verstehen sich diese mit den unterschiedlichsten Datenbanken. So hat die Ruby-Implementierung Adapter für MySQL, PostgreSQL, SQLite, Oracle, SQLServer und DB2. Ruby on Rails macht exzessiven Gebrauch von Active Record und erweitert den Ansatz Fowlers um einige zusätzliche Aspekte wie Validations, Callbacks, Migrations und viele andere mehr.

Migrations

Ein Problem vieler Entwickler-Teams ist es, mit einer konsistenten Datenbankstruktur zu arbeiten. Wenn ein Entwickler der Datenbank ein Feld hinzufügt oder ein vorhandenes Feld ändert, müssen diese Änderungen sowohl zu allen anderen Entwicklern als auch auf das Live-System übertragen werden, ohne dass die Systeme inkonsistent werden. Rails löst dies mit Hilfe sogenannter „Migrations“ [61], die von ActiveRecord bereitgestellt werden. Dass diese Migrations in Ruby programmiert werden hat den Vorteil, dass Migrations im Großen und Ganzen Datenbanksystem-unabhängig sind: Anweisungen, die auf einem MySQL-System geschrieben wurden, werden damit auch auf einem PostgreSQL-System laufen. Die Migrations anderer Entwickler werden z.B. über ein Versionskontrollsystem auf das eigene System übertragen und auf der lokalen Datenbank ausgeführt. In jeder Transformation gibt es sowohl eine `up`- als auch eine `down`-Methode. Die `down`-Methode macht die Änderungen der `up`-Methode rückgängig. Eine `up`-Anweisung kann z.B. lauten: Benenne das Feld `bar` in der Tabelle `foo` in `baz` um. Die zugehörige `down`-Methode wäre dann: Benenne das Feld `baz` in der Tabelle `foo` in `bar` um. Aber auch neue Tabel-

[3] Im Gegensatz zu der von Fowler geprägten Schreibweise von „Active Record“ schreibt sich die Ruby-Bibliothek „ActiveRecord“ ohne Leerzeichen.

len können mit Migrations erstellt und selbst Datensätze können verändert werden.

Unit Test

Rails bietet mit der Unit Test-Bibliothek eine Infrastruktur, die alle MVC-Ebenen testen kann. Während sich die Unit-Tests für Modelle [62] und die Functional-Tests für Controller [63] ziemlich unspektakulär verhalten, sind die Integration-Tests sozusagen die Königsklasse. Während die Functional-Tests nur einen einzelnen Controller testen, können Integration-Tests gleich über mehrere Controller hinweg auch die Views bis hin zum zurückgelieferten HTML testen [64]. Es können also ganze Szenarien oder Benutzerflüsse wie z.B. eine Benutzer-Registrierung getestet werden. Wie granular man seine Anwendung testen möchte muss jedes Entwickler-Team selbst beantworten. Oft ist es aber so, dass zu granulare Tests keinen Sinn mehr ergeben, da der Aufwand, die Tests anzupassen, zu groß wird, wenn man nur eine Kleinigkeit im Code geändert hat. Selbst zum Testen von JavaScript- und Ajax-Funktionalitäten gibt es Bibliotheken wie Jasmine [65], die zum Testen in das Framework eingebunden werden können. Zum eigentlichen Testen der Modelle bietet es sich an, Test-Daten zu erstellen, was in Rails durch sogenannte „Fixtures“ umgesetzt wird [66]. Dies sind vom Programmierer vorgegebene Daten, die vor jedem Test in eine extra angelegte Test-Datenbank geladen werden. So ist sichergestellt, dass jeder Test auf der gleichen Datenbasis arbeitet.

Umgebungen

Beim Anlegen eines Rails-Projekts werden insgesamt mindestens drei Umgebungen vorgegeben: Production, Development und Test [67]. Je nachdem, in welcher Rolle sich die Applikation befindet, wird die entsprechende Umgebung genutzt. Dies kann man über globale Variablen steuern. Die richtige Umge-

bungsvariable sorgt unter anderem dafür, dass in den Tests stets die Test-Datenbank genutzt wird. Ein anderes Beispiel ist das deaktivierte Caching in der Development-Umgebung: Auf Kosten von Performance wird auf das Cachen von Klassen verzichtet, damit aber erreicht, dass Änderungen am Code sofort reflektiert werden können. Manche Entwickler nutzen zusätzlich eine Staging-Umgebung, die live auf den Produktiv-Servern läuft, dort allerdings nur von einer relativ kleinen, klar abgesteckten Nutzergruppe verwendet wird, die Bugs sofort an die Entwickler melden können.

Rubygems

Einer der Vorteile von Ruby on Rails ist es, dass es von anderen, älteren Web-Frameworks oder auch Web-Sprachen wie PHP lernen konnte. Man erkannte recht früh, dass es Aufgaben gibt, die in vielen Anwendungen sehr ähnlich ablaufen. Als Beispiel sei eine Benutzerverwaltung genannt. Sehr viele Webanwendungen setzen in irgendeiner Form einen angemeldeten Benutzer voraus der mit der Anwendung interagieren kann. Anstatt das Rad jedesmal neu zu erfinden, bietet es sich an, diese Funktionalität in eine Bibliothek auszulagern und in das Projekt einzubinden. In der „Ruby-Welt“ heißen diese externen Bibliotheken Rubygems (kurz nur „Gems“, englisch für „Edelsteine“, in Anlehnung an Ruby) [68]. Gems sind das offizielle Paketsystem der Programmiersprache Ruby und werden unter anderem von RubyGems.org [69] gehostet. Rubygems ist im Verhalten sehr ähnlich zu APT, portage, yum oder auch CPAN (Comprehensive Perl Archive Network). Selbst Ruby on Rails besteht aus einem Gem, welches andere Gems wie z.B. ActiveRecord als Abhängigkeit einbezieht.

Deployment

Das Deployment (englisch im Sinne von „zum Einsatz bringen“) erledigt man typischerweise durch ein Deployment Tool. In Rails-Projekten nutzt man oft

Capistrano, welches sich darum kümmert, dass die Datenbank auf dem Server oder den Servern migriert, und die Codebasis aktualisiert wird [70]. Dabei kann man in einem sogenannten „Capistrano Recipe“ verschiedene Rollen definieren, und damit genau festlegen, welcher Server welche Änderungen durchzuführen hat. So muss ein Datenbankserver üblicherweise lediglich die Datenbankstruktur aktualisieren, der Quellcode aber wird ignoriert, da er auf diesem Server sowieso nicht ausgeführt wird. Andererseits läuft auf einem Fileserver meist keine Datenbank und so muss hier auch nichts migriert werden. Auch zum Aufsetzen eines neuen Servers eignet sich Capistrano, da es spezielle Setup-Rezepte bietet, welche die notwendigen Abhängigkeiten der Anwendung auf dem Server installieren, Datenbanken anlegen oder Verzeichnisse erstellen können.

Rack

Seit Version 2.3 ist Rails kompatibel zu Rack, einem minimalen, modular aufgebauten und erweiterbaren Interface, welches es ermöglicht, Webanwendungen in Ruby zu entwickeln. Indem es HTTP-Requests und -Antworten auf eine sehr einfache Art kapselt, vereinheitlicht und konzentriert Rack eine API für Webserver, Web-Frameworks und Middleware in einem einzelnen Methoden-Aufruf [71]. Der Vorteil ist, dass viele verschiedene Webserver wie z.B. Mongrel, FCGI, Phusion Passenger (ein Modul für Apache und nginx) von Haus aus unterstützt werden. Außerdem werden neben Ruby on Rails auch andere Ruby-Web-Frameworks wie Sinatra [72] oder Camping [73] unterstützt. Rack dient also als kleinster gemeinsamer Nenner all dieser Ruby-Projekte. Da die Entwickler des Rails-Frameworks Rack-kompatibel programmieren, sind alle Rails-Projekte automatisch auf diversen Webservern lauffähig. Auch lassen sich Middleware-Projekte laden, die sich dann z.B. um das Bereitstellen statischer Dateien wie Grafiken oder CSS- oder JavaScript-Dateien kümmern. Wie

es bei Ruby-Projekten üblich ist, wird auch Rack als Rubygem zur Verfügung gestellt [74].

Rake

Rake ist das Ruby-Pendent zu dem aus der Unix-Welt bekannten Programm „make“. Rake-Files werden komplett in Ruby geschrieben [75]. Jedes Rails-Projekt wird mit diversen vordefinierten rake-Tasks erstellt, die typischerweise für administrative Zwecke gebraucht werden. Beispielsweise gibt es rake-Tasks mit denen sich die Datenbank migrieren, oder mit denen sich das oben erwähnte Scaffolding einleiten lässt [76]. Den rake-Tasks können Argumente übergeben werden und sie können andere Tasks einbinden oder aufrufen. Seit Ruby 1.9 ist rake Bestandteil der Standard-Ruby-Bibliothek, für ältere Ruby-Versionen wird es als Rubygem bereitgestellt.

Konsole

Mit Hilfe der Rails-Konsole [77] kann direkt auf der Datenbank gearbeitet werden kann. Wenn unabhängig von der eigentlichen Anwendung Änderungen an der Datenbank vorgenommen werden sollen, muss hierfür nicht wie sonst üblich mit der SQL-Konsole oder Hilfsmitteln wie phpMyAdmin gearbeitet werden. Stattdessen startet man die Konsole und hat dort Zugriff auf alle Models. Die Rails-Konsole erweitert dazu einfach eine IRB (Interactive Ruby Shell) [78] um Rails-spezifische Bibliotheken wie z.B. ActiveRecord. Es ist sogar möglich die Konsole in einem Sandbox-Modus zu starten, der alle Änderungen der aktuellen Session rückgängig macht, wenn man die Konsole verlässt.

3.5 Weitere Frameworks für Server

Aufgrund der riesigen Anzahl verfügbarer Web-Frameworks ist es nahezu unmöglich im Rahmen dieser Arbeit alle diese Projekte vorzustellen. Stattdes-

sen wird neben Ruby on Rails auf vier weitere besonders bekannte (Symfony, FLOW3, Django und Google Web Toolkit) und zwei exotische Frameworks (Opa und Ocsigen) eingegangen.

3.5.1 Symfony (PHP)

Wie die meisten anderen PHP-Frameworks ist auch Symfony [79] ein noch recht junges Produkt, das im Jahr 2005 das erste Mal der breiten Öffentlichkeit unter der MIT-Lizenz zugänglich gemacht wurde [80]. Die eigentlichen Wurzeln reichen aber bis ins Jahr 2003 zurück, als Fabien Potencier unbeachtet von der Allgemeinheit mit der Entwicklung eines eigenen Frameworks für die französische Internetagentur Sensio begann. Als er das Projekt im Jahr 2005 schließlich der Öffentlichkeit übergab, befand es sich bereits in einem benutzbaren und ausreichend stabilen Zustand. Dies war nicht zuletzt auch ein Grund, warum sich schnell Entwickler fanden, die sich bereiterklärten sich mit dem Framework auseinanderzusetzen und eigene Anwendungen damit zu entwickeln. Während andere Frameworks nur im Ansatz versuchen, bekannte und ausgereifte Lösungen elegant in das Framework zu integrieren, verfolgt Symfony diesen Ansatz sehr konsequent, wie am Beispiel von Doctrine [81] zu sehen ist:

Obwohl Doctrine nahtlos in Symfony integriert wurde, ist es dennoch vollständig getrennt zu betrachten, da es ein eigenständiges Projekt ist [82]. Die Bibliothek basiert auf dem Active Record-Ansatz [83], welcher schon im Ruby on Rails-Kapitel vorgestellt wurde, und hilft demnach dabei Objekte auf die Datenbank abzubilden. Es ist möglich viele verschiedene Datenbank-Systeme als Backend zu nutzen, unter anderem MySQL, PostgreSQL oder Microsoft SQL. Die Syntax ist dabei sehr ähnlich zu der Syntax der Ruby-Bibliothek.

Historisch bedingt gehört die Trennung der einzelnen Aufgabenbereiche, wie Steuerungslogik, Datenbankabfrage und Ausgabe, nicht gerade zu den Stärken

von PHP. Da dies inzwischen aber nach Meinung vieler Softwareentwickler Voraussetzung ist, um den Überblick im Code zu wahren, nutzt auch Symfony hierfür den Model, View, Controller-Ansatz.

3.5.2 FLOW3 (PHP)

Das FLOW3-Framework [84] stellten die TYPO3-Entwickler nach fünf Jahren Entwicklung im August 2011 vor [85]. Obwohl es ursprünglich als technisches Fundament für das TYPO3-Content-Management-System gedacht war, reifte das Projekt zu einem eigenständigen Framework heran. Anders als Symfony versteht sich FLOW3 nicht als Komponenten-Framework, aus dem man sich ein paar einzelne Komponenten herauspicken könnte. FLOW3 ist eher eine „Web Application Platform“ und gibt ein übergreifendes Konzept vor, wie man moderne, gute Software bauen kann. FLOW3 vereint viele Best-Practice-Ansätze unter einem Dach und ermöglicht es dem Entwickler auf diese Weise, Webanwendungen auf höchstem Niveau zu entwickeln.

Der Code, den man mit dem Framework entwickelt, ist sehr gut gekapselt und hat so nur minimale API-Abhängigkeiten zu FLOW3. Dieses Entwurfsmuster, welches auch als „Dependency Injection“ bezeichnet wird, dient in einem objektorientierten System dazu, die Abhängigkeiten zwischen Komponenten oder Objekten zu minimieren. Es überträgt dafür die Verantwortlichkeit für das Erzeugen und Verknüpfen von Objekten an ein extern konfigurierbares Framework, entsprechend einem Komponentenmodell (in diesem Fall wäre es FLOW3). Dadurch wird der Code der Objekte unabhängig von ihrer Umgebung und es könnten so auch andere Frameworks als „Drop-In-Replacement“ eingesetzt werden. So kann man zum Beispiel eine Symfony-Komponente nehmen und sie direkt in FLOW3 benutzen. Die Bezeichnung der Dependency Injection führte Martin Fowler 2004 ein [86]. FLOW3-Entwickler Robert Lemke sagt in einem Interview, dass das Framework es grundsätzlich fördert, dass Ent-

wickler nicht zu viel Code schreiben brauchen, gleichzeitig aber auf möglichst wenig „Magic“ vertrauen müssen [87]. Die Entwicklung mit FLOW3 soll so durchschaubar bleiben, dass der Entwickler jederzeit den roten Faden erkennen kann, der sich durch das Framework zieht.

Das FLOW3-Framework nutzt Doctrine2 für das Persistenzmanagement, da dort laut der FLOW3-Entwickler viele Probleme bereits gut gelöst sind und man deutlich von der Leistungsfähigkeit von Doctrine2 profitieren kann [88].

Obwohl es bereits einige Template-Engines auch speziell für PHP gibt, wurde für FLOW3 eine komplett neue Engine namens „Fluid“ entwickelt, da keine der existierenden Engines den Anforderungen der Entwickler genügte. Innerhalb der Templates ist reiner PHP-Code verboten, Variablen, die im Template verarbeitet werden sollen, müssen im entsprechenden PHP-Code an Fluid übergeben werden.

Ähnlich zur Rails-Konsole gibt es auch bei FLOW3 ein Command Line Interface, in dem man Unix-artige Befehle eingeben kann. So lassen sich viele Aufgaben automatisieren und effizient erledigen.

Das FLOW3-Framework verfolgt zeitgemäße, objektorientierte Ansätze, wie das der aspektorientierten Programmierung (AOP). Dieses Konzept wurde von Gregor Kiczales und seinem Team bei der Firma Xerox PARC entwickelt. Im Jahr 2001 wurde dort auch die erste AOP-Sprache AspectJ vorgestellt [89]. Dieses Programmierparadigma wurde entwickelt um generische Funktionalitäten über mehrere Klassen hinweg zu verwenden. Dabei werden einzelne Aspekte eines Programms von der eigentlichen Geschäftslogik getrennt. Eine bestehende Klasse wird mit einer anderen so verwoben, dass diese sich, abweichend von ihrem ursprünglichem Entwurf, völlig anders verhält. Das Besondere dabei ist, dass kein Eingriff in den Code der Ursprungsklasse erforderlich ist.

FLOW3 setzt auf Domain-Driven Design, was bedeutet, dass der Schwerpunkt des Softwaredesigns auf der Fachlichkeit und der Fachlogik liegt [90]. Bei

dem von Eric Evans geprägten Begriff handelt es sich um eine Sammlung von Prinzipien und Mustern, die Entwicklern beim Entwurf eleganter Objektsysteme helfen. Es soll so zu Softwareabstraktionen führen, die als Domänenmodelle bezeichnet werden. Diese Modelle kapseln komplexe Geschäftslogik und schließen die Lücke zwischen Geschäftsrealität und Code [91].

3.5.3 Django (Python)

Auch Django [92] setzt etliche Design-Philosophien um von denen hier einige vorgestellt werden. Das nach dem Jazz-Gitarristen Django Reinhardt benannte Python-Framework wurde „from scratch“ neu entwickelt. Es entstand ab 2004 bei „The World Company“, einer Firma aus Lawrence, Kansas, die mehrere Nachrichtenseiten im Internet betreibt [93]. Im Juli 2005 wurde das Framework unter BSD-Lizenz freigegeben.

Obwohl auch Django im Grunde eine Model, View, Controller-Architektur nutzt, gibt es doch einige Unterschiede zu anderen Web-Frameworks [94], die bei Umsteigern von anderen Frameworks leicht zu Verwirrung führen können. So heißen die Controller bei Django „Views“ und die Views heißen „Templates“. Die Django-Entwickler argumentieren, dass in ihrer Interpretation die Views die Daten beschreiben, die dem Benutzer präsentiert werden. Sie beschreiben nicht, wie die Daten auszusehen haben, sondern welche Daten man überhaupt bekommt. Demnach ist ein View im „Django-Universum“ eine Python Callback-Funktion für eine bestimmte URL, weil die URL bestimmt, welche Daten überhaupt präsentiert werden. Außerdem muss man zwischen Inhalt und Präsentation unterscheiden. Während die Views zuständig sind zu entscheiden welche Daten angezeigt werden, kümmern sich Templates darum wie diese Daten präsentiert werden. Die Controller wiederum sind bei Django das Framework selbst, nähmlich der Algorithmus im Hintergrund, der einen Request anhand der URL an den korrekten View sendet. Scherzhaft bezeichnen

die Entwickler Django auch als „MTV“-Framework, was für Model, Template, View steht.

Auch Django setzt auf das „Don’t Repeat Yourself“-Prinzip (DRY). Redundanz wird als schlecht betrachtet und Normalisierung soll gefördert werden. Das Django-Framework soll möglichst viel Logik aus möglichst wenig Code ableiten können. Als Beispiel führt man den Y2K-Bug an: Das größte Problem an diesem Bug bestand darin, dass es nicht nur eine einzige Abstraktion innerhalb eines Systems gab, sondern dass das Wissen über Daten (als Mehrzahl von Datum) weit verteilt wurde. Deshalb war es nicht möglich den Bug an einer Stelle zu beheben [95].

Im Gegensatz zu z.B. Ruby on Rails versuchen die Django-Entwickler nicht zu viel „Magic“ zu benutzen. Dies ist bereits ein Python-Prinzip, welche im „Zen of Python“ festgelegt wird [96] und von den Django-Entwicklern übernommen wurde [97]: „Explicit is better than implicit.“ Der Grundsatz sagt aus, dass Code von Anwendern sofort verstanden werden sollte. Dies sieht man auch daran, dass Datenbank-Felder kein bestimmtes Verhalten nur aufgrund ihres Namens annehmen [98].

Django verfolgt wie Ruby on Rails den Active Record-Ansatz von Martin Fowler. Die Models kapseln demnach alle Aspekte der Objekte ab [99]. Der Object-Relational Mapper von Django ist dabei sehr mächtig. So kann man mit Hilfe weniger Zeilen im Code in den Models eine Datenbankstruktur anlegen und Django erzeugt automatisch das SQL für die Erstellung der Tabellen in verschieden Datenbankmodellen wie MySQL oder PostgreSQL und erstellt dann die entsprechenden Tabellen. Migrations, wie man sie von Ruby on Rails kennt, sind demzufolge bei Django nicht unbedingt nötig.

3.5.4 Google Web Toolkit (Java)

Das Google Web Toolkit (GWT) wird, wie der Name schon vermuten lässt, von Google entwickelt und wurde im Mai 2006 als freie Software unter einer Apache-Lizenz der Öffentlichkeit übergeben. Das Framework verfolgt einen gänzlich anderen Ansatz als z.B. Ruby on Rails oder Django, welche Hochsprachen wie Ruby oder Python nutzen, um die Geschäftslogik abzubilden. GWT wiederum ist im Kern ein Java-nach-JavaScript-Compiler, der es ermöglicht nahezu die gesamte Entwicklung auf Basis von Java durchzuführen [100]. GWT kompiliert dazu die Anwendung und erstellt browserunabhängigen HTML-, CSS- und JavaScript-Code. Anstatt JavaScript oder Ajax-Anweisungen in HTML-Code zu schreiben, fügt der Entwickler Java-Programme ein, die das Toolkit selbstständig kompiliert. Nach dem Kompilieren ist GWT zum Betrieb der Anwendung nicht mehr nötig. Es kann auf einen Pool von Bibliotheken z.B. für Layouts oder Formularelemente zurückgegriffen werden. Viele Produkte von Google wurden in GWT umgesetzt, u.a. das in Südamerika sehr erfolgreiche Orkut und die neue Version von AdWords [101].

Dabei kann GWT neben einer Desktop-Anwendung auch gleich Versionen kompilieren, die für mobile Endgeräte optimiert sind. Google wirbt damit, dass der kompilierte Code bezüglich der Geschwindigkeit optimiert ist. Dies gelte sowohl für die Performance im Browser als auch in Hinsicht der Download-Geschwindigkeit. Der Code wird aufgeteilt, so dass selten benötigter Code für spezielle Bereiche der Seite erst geladen werden, wenn die Startroutine bereits ausgeführt wird.

Google bietet eine fertige Infrastruktur rund um Google Web Toolkit an. So ist es z.B. möglich, GWT-Applikationen auf Google App Engine [102] zu hosten. Neben einer niedrigen Downtime ist die einfache Skalierbarkeit zu nennen. Sobald eine Anwendung auf der aktuellen Hardware an ihre Grenzen

stößt, kann ein weiterer Server dazugebucht werden und so der Weiterbetrieb gewährleistet werden.

Des Weiteren veröffentlichte Google 2011 ein Eclipse Plugin unter einer Open Source Lizenz [103]. Eclipse ist gerade unter Java Entwicklern eine beliebte Entwicklungsumgebung. Mit Hilfe des Plugins ist es Java-Entwicklern möglich, cloud-basierte Anwendungen mit dem Google Web Toolkit zu designen, entwickeln, optimieren und anzuwenden.

3.5.5 Opa (Opa)

Relativ neu ist Opa [104], welches 2010 vorgestellt, und im Juni 2011 als Open Source veröffentlicht wurde. Entwickelt wurde Opa von MLstate [105], einem Zusammenschluss verschiedener Forscher auf dem Gebiet der Online-Softwareentwicklung. Es unterscheidet sich grundsätzlich von anderen Web-Frameworks die in dieser Arbeit vorgestellt werden. Im Grunde ist Opa nicht einmal ein Web-Framework im klassischem Sinne, sondern es umfasst eine neue Programmiersprache, einen neuen Webserver, eine neue Datenbank und einen neuen verteilten Anwendungsserver. Die Architektur der Sprache ist weitestgehend immun gegen SQL-Injections und XSS-Angriffe und darüber hinaus sind in Opa entwickelte Applikationen sehr gut skalierbar. Laut der Entwickler unterscheidet sich die Entwicklung von Webanwendungen heutzutage kaum von der Softwareentwicklung in den neunziger Jahren da ein Großteil der Zeit darauf verwendet wird, wenig kompatible Technologien miteinander zu verknüpfen (das Prinzip wird auch als „Impedance Mismatch“ bezeichnet). Im Sinne der Web-Programmierung bedeutet das, dass der Entwickler verschiedene Browser, diverse Serverkomponenten und Datenbanken vereinen muss. Dieses Problem wollen die Opa-Entwickler lösen.

Code wird in Opas homogener Umgebung ausschließlich in Opa programmiert. SQL-Anweisungen sucht man folglich genauso vergebens wie JavaScript-

Einschübe. Opa-Code wird, ähnlich wie Code welcher mit dem Google Web Toolkit programmiert wurde, kompiliert. Der Unterschied ist, dass die Ausgabe keine HTML-, JavaScript- und CSS-Dateien sind, sondern dass der Compiler eine ausführbare Datei zurückliefert. Diese Datei enthält nicht nur den server- als auch den clientseitigen Code, sondern auch die komplette Datenbank und den Applikationsserver [106]. Der Aufwand für das Deployment, Datenbank-Management, das Warten von Frameworks oder sonstiger Middleware entfällt.

Die Sprache Opa ist funktional und mit Typinferenz statisch getypt. Die Syntax kommt dabei bekannt vor: Beeinflusst wurde die Sprache von OCaml und Erlang. Außerdem finden sich Ähnlichkeiten zu C und JavaScript. Opa unterstützt von Haus aus MongoDB und CouchDB. Für neue Projekte bietet sich aber an, die Opa-eigene Datenbank zu nutzen, was keine Konfiguration benötigt. Opa implementiert des Weiteren Sessions welche einen imperativen Status kapseln können und Nachrichten ähnlich zu Erlang-Prozessen austauschen. Die Sprache liefert viele Datenstrukturen und Funktionen, die stark auf die Webentwicklung gemünzt sind. Aufgrund dieser starken Bindung der Sprache an das Web-Framework ist es nicht vorgesehen die Sprache in Anwendungen zu nutzen, die keinen Web-Kontext haben, so wie dies z.B. bei Desktop-Anwendungen der Fall ist.

3.5.6 Ocsigen (OCaml)

Ocsigen ist ein Web-Framework, welches ab 2007 während eines Forschungsprojekts der Diderot Universität [107] in Paris implementiert wurde. Heute ist es Teil von IRILL[4], einer französischen Einrichtung um freie Software zu erforschen und zu unterstützen [109].

[4] **I**nitiative de **R**echerche et **I**nnovation sur le **L**ogiciel **L**ibre (Forschung und Innovation auf Basis Freier Software) [108]

Ocsigen macht es ebenso wie Opa möglich, Webanwendungen als ein einziges Programm zu schreiben. So wird die gleiche Sprache (OCaml [110]) benutzt, um die Server- und Client-Programme zu schreiben, es können Server-Variablen im Client-Code genutzt werden, und serverseitige Funktionen aus dem Client-Code aufgerufen werden. Der Vorteil dieser Architektur ist, dass die gleichen Datentypen, Bibliotheken usw. genutzt werden können. „Server-zu-Client"-Kommunikation ist ebenso möglich, so kann der Client informiert werden, wenn ein Ereignis auf dem Server stattgefunden hat (z.B. wenn eine neue Nachricht eingetroffen ist) [111].

Welche Aktion der Server ausführen soll, wird durch sogenannte „Services" bestimmt. Welcher dieser Services aufgerufen werden soll, entscheidet sich anhand der URL, der mitgesendeten Parameter und der aktuellen Session. Interessant sind hier „dynamische Services", welche auf vorangegangen Interaktionen mit dem Benutzer basieren. Dies hilft dabei voneinander abhängige Seitenaufrufe zu identifizieren. Um dies korrekt zu implementieren ist es üblicherweise nötig, alle Informationen über jeden einzelnen Schritt jedes Benutzers zu speichern, wofür in traditionellen Frameworks serverseitige Datenbanken genutzt werden. Mit Ocsigen kann man sehr komfortabel einen dynamischen Service für jeden „Checkpoint" im Interaktionsbaum erstellen [112]. Diese Methodik ist auch als „Continuation Based Programming" bekannt [113], und ist dem zustandslosem HTTP geschuldet.

Die statische Typisierung von Ocsigen bringt einige Vorteile mit sich. So können bereits während der Kompilierung viele Bugs ausgeschlossen werden. Das Typsystem garantiert sogar, dass wohlgetypte Programme niemals abstürzen. Ebenfalls kümmert es sich darum, dass Form-Elemente und Links passend zu den Services sind die sie aufrufen, z.B. muss ein Formular eine Checkbox beinhalten, wenn der dazugehörige Service einen boolschen Wert

erwartet. Außerdem wird während des Kompilierens geprüft ob das Ergebnis-HTML valide (also W3C-konform) ist [114].

Wie schon erwähnt, wird sowohl Ocsigen selber, als auch die Programme die der Anwender damit entwickelt, in OCaml programmiert. Der OCaml-Code für den Client wird dann von der Bibliothek „Js_of_ocaml“ [115] von OCaml-Bytecode zu JavaScript kompiliert. Außerdem existieren Bindings um andere JavaScript-Bibliotheken wie „Google Closure“ einzubinden. Ocsigen ist, wie viele andere Frameworks auch, in Komponenten unterteilt, die man auch einzeln nutzen kann [116].

3.6 Vergleichstabelle aller vorgestellten Frameworks für Server

Zusätzlich zu den textuellen Beschreibungen findet sich für die bessere Übersicht auf der nächsten Seite eine Tabelle aller vorgestellten Web-Frameworks.

	Rails	**Symfony**	**FLOW3**	**Django**	**GWT**	**Opa**	**Ocsigen**
Sprache	Ruby	PHP	PHP	Python	Java, JavaScript	Opa	OCaml
aktuelle Version	3.1.3	2.0.6	1.0	1.3.1	2.4.0	1311	2.0.2c
veröffentlicht am	07.10.2011	16.11.2011	20.10.2011	09.09.2011	08.09.2011	14.02.2012	05.01.2012
Lizenz	MIT / Ruby	MIT	LGPL	BSD	Apache	AGPL	LGPL
ORM	ActiveRecord	Propel, Doctrine	Doctrine2	Django ORG	Java Persistence API	unbekannt	Mirage ORM
i18n & l10n	ja	ja	ja	ja	ja	nein	nein
MVC	ja	ja	ja	ja	nein	nein	nein
Test-Framework	ja	ja	ja	ja	externe Bibliotheken	nein	nein

Abbildung 3: Vergleichstabelle aller vorgestellten Web-Frameworks (Stand: Februar 2012)

3.7 WebSocket-Framework: Socket.IO

Wie bereits erwähnt reicht im Gegensatz zu reinen HTTP-Verbindungen beim WebSocket-Protokoll, dass der Client eine einzelne Verbindung öffnet. Diese kann im Folgenden dann vom Server genutzt werden um Informationen ohne Zeitverzug auszuliefern. Technisch betrachtet startet das WebSocket-Protokoll wie ein gewöhnlicher HTTP-Request/ Response-Zyklus. Nach der Übertragung der HTTP-Request- und Response-Header bleibt die zugrundeliegende TCP-Verbindung allerdings bestehen und wird dadurch für Übertragungen in beide Richtungen freigehalten.

Socket.IO [117] ist eine JavaScript-Bibliothek, die zu Beginn von Guillermo Rauch entwickelt wurde. Inzwischen haben sich etliche weitere Entwickler gefunden und sich das Ziel gesetzt, WebSockets möglichst allen Browsern zur Verfügung zu stellen. Ähnlich wie die jQuery-Bibliothek die Ajax-Requests vereinfacht, kümmert sich Socket.IO um WebSockets, indem Sockets nur einmal programmiert werden müssen, und dieser Code dann in allen Browsern lauffähig ist. Das Besondere an Socket.IO ist, dass für Webbrowser, die WebSockets nativ nicht unterstützen, Fallback-Lösungen implementiert werden. Nachdem festgestellt wurde, dass WebSockets in dem aktuellen Browser nicht zur Verfügung stehen, kann auf Adobe Flash Sockets ausgewichen werden. Falls Flash nicht installiert ist oder aus anderen Gründen ausfällt, kommen noch vier weitere Techniken zur Anwendung. Alle diese Techniken haben natürlich ihre Nachteile, die Socket.IO aber transparent umschifft, und sie auf einen gemeinsamen Befehlssatz zurückführt, der von allen diesen Techniken ausgeführt werden kann. Dank dessen kommt die Bibliothek sogar mit dem Internet Explorer ab Version 5.5 und vielen mobilen Browsern zurecht [118].

4 Mobile Computing

4.1 Warum es wichtig ist, Smartphone-Frontends zu haben

Mobile Web-Applikationen sind Webseiten, die für mobile Endgeräte wie Smartphones oder Tablet-Computer aufbereitet sind und sich dank der Internet-Standards HTML5 und CSS3 sowie der Scriptsprache JavaScript Eigenschaften nativer Anwendungen aneignen. Sie werden gewöhnlich mit dem internen Browser des Endgerätes bedient. Dabei werden browserbasierte mobile Anwendungen zunehmend zu echten Alternativen für native Anwendungen. Weil mobile Geräte heute fast immer online sind und durch die vielen Möglichkeiten, die moderne Browser z.B. mit HTML5 bieten, lassen sich mobile Websites entwickeln, die vom „Look & Feel“ und Funktionsumfang her in vielen Fällen nativen Applikationen gleichkommen.

Die Nutzung des mobilen Internets nimmt immer weiter zu. Auch werden mobile Internetinhalte immer wichtiger. So stellten laut einer Statistik des Statistischen Bundesamtes Deutschlands im Jahr 2010 16% der Internetnutzer die Verbindung zum Internet über ein mobiles Gerät her. Dies ist eine Steigerung zum Vorjahr um 78% [119]. Laut eigener Aussage benutzen mehr als 350 Millionen Nutzer (44%) Facebook mit mobilen Geräten. Menschen, die Facebook mit mobilen Endgeräten nutzen sind doppelt so aktiv wie andere Nutzer [120]. Auch Google sieht einen signifikanten Wechsel im Nutzerverhalten: Inzwischen benutzen mehr Konsumenten ein „Feature Phone“[5] oder Smartphone als einen Computer [121]. Google verfolgt deshalb sogar eine „Mobile First“-Strategie. „Mobil“ ist hier aber nicht der kleine Bruder von „Online“, bei dem es lediglich gilt, ein kleineres Display zu berücksichtigen. Oft sind die Erwartungen der Be-

[5] Als **Feature Phone** wird ein Mobiltelefon bezeichnet, welches nicht zu den Smartphones zählt, trotzdem aber wesentlich mehr Funktionalitäten als die GSM-Services bietet.

nutzer an ein mobiles Interface anders definiert und es gilt die wechselseitigen Abhängigkeiten der verschiedenen Kanäle zu nutzen.

4.2 Vier Ansätze von mobilen Frontends

Es gibt verschiedene Ansätze ein mobiles Interface umzusetzen, von dem natürlich jeder Ansatz seine eigenen Vor- und Nachteile mit sich bringt: Ein Vorteil der Web-App-Programmierung ist der einfache Einstieg vor allem für Webentwickler, da sie keine neuen Programmiersprachen erlernen müssen, sondern ihr bisheriges Wissen in die Entwicklung der mobilen Webseiten einfließen lassen können. Eine native Anwendung zeichnet sich dadurch aus, dass die „User Interface Widgets“ des zugrundeliegenden Betriebssystems genutzt werden können. So kann eine Web-Anwendung manchmal wie ein Fremdkörper wirken, da sie eine eigene Bedienstruktur mitbringt. Dabei sollte die Entscheidung ob eine plattformunabhängige Web-Anwendung oder eine native Anwendung entwickelt wird, abhängig sein von der Komplexität und den Eigenschaften, die die mobile Anwendung aufweisen soll.

4.2.1 Cascading Style Sheets

Die einfachste Möglichkeit ist sicherlich der „Cascading Style Sheets“ (CSS) basierte Ansatz. Dabei wird der Internetseite ein Stylesheet hinzugefügt, welches die Seite für mobile Endgeräte optimiert. Der Vorteil liegt auf der Hand: Ein solches Stylesheet ist im Idealfall in wenigen Minuten zusammengesetzt. Obwohl keine Patentlösung existiert, gibt es doch einige Grundregeln die beim Erstellen des Stylesheets zu beachten sind. Aus HTML4 und CSS2 ist das `media`-Attribut für Stylesheets bereits bekannt [122]. Mit diesem kann man einzelne CSS-Files nur für bestimmte Endgeräte wie Beamer (`media='projection'`) oder Mobiltelefone (`media='handheld'`) einbinden und so auf die Eigenheiten dieses Gerätes eingehen. CSS3 bietet mit den sogenannten „Media Que-

ries“ zusätzlich die Möglichkeit unter anderem auf die minimale und maximale Breite oder die Orientierung (`portrait` oder `landscape`) des Gerätes einzugehen: `media='handheld, only screen and (max-device-width: 320px) and (orientation:portrait)'` [123]. So kann man z.B. für Tablets und Telefone unterschiedliche Styles einbinden.

Zu den Nachteilen dieses Ansatzes zählt unter anderem die nicht optimierte Bandbreiten-Nutzung der Anwendungen: Oft werden HTML-Teile, die auf den mobilen Endgeräten nicht angezeigt werden sollen mit `display:none` ausgeblendet, der betroffene HTML-Code wird allerdings trotzdem übertragen. Während der eigentliche HTML-Code nur wenige Byte groß ist, summiert es sich schnell zu einigen hundert Kilobyte wenn auch Bilder im HTML-Code referenziert werden, die zwar übertragen, aber nicht angezeigt werden. Zusätzlich werden bei der Nutzung dieser Methode Bilder nicht für kleine Bildschirme optimiert und müssen daher in ihrer vollen Größe übertragen werden.

4.2.2 Mobile Web-Frameworks

Da man mit Anpassungen der eigenen Seiten über Stylesheets ziemlich schnell an seine Grenzen stößt, gehen viele Entwickler einen Schritt weiter und entwickeln mobile Anwendungen mit eigens dafür entworfenen Frameworks wie „Sencha Touch“ [124]. Sencha Touch ist ein „mobiles Web Framework“. Man benutzt es um webbasierte, mobile Anwendungen zu erstellen. Man hat dabei keinen Zugriff auf die nativen APIs des Endgerätes, da lediglich mit JavaScript gearbeitet wird. Anwendungen, die mit Sencha Touch erstellt werden, bleiben Webanwendungen, ein Einzug in die App-Stores der verschiedenen Smartphones bleibt ihnen daher verwehrt.

Neben Sencha Touch gibt es natürlich noch eine ganze Reihe anderer Frameworks, die diese Aufgabe ebenso gut erledigen. Eines der bekanntesten dürfte jQuery Mobile sein [125], welches durch die flache Lernkurve hervorsticht. Das

Framework baut laut eigener Aussage auf HTML5 auf und unterstützt alle wichtigen mobilen Geräte. Als Basis dienen die „Mutter“-Frameworks jQuery und jQuery UI.

Neben einer guten Übersicht über alle aktuellen mobilen Frameworks findet sich im „Mobile Frameworks Comparison Chart“ [126] eine Entscheidungshilfe, mit derer Hilfe aus den über 30 Frameworks das gewünschte gefunden werden kann. Die oben bereits angesprochenen strukturbedingten Nachteile haben aber alle reinen Web-Frameworks gemein: Ihnen fehlt der Zugriff auf Gerätefunktionen wie der Kamera oder des Mikrofons und der fehlende standardisierte Distributionskanal ist ein großer kommerzieller Nachteil. Während native Anwendungen einfach in den entsprechenden App-Stores gesucht und gefunden werden können, ist dies bei reinen Web-Anwendungen nicht möglich.

4.2.3 Hybride Anwendungen

Im Gegensatz dazu zählen Anwendungen, die mit PhoneGap [127] erstellt wurden als native Anwendungen und können damit in die App-Stores eingestellt werden. Dabei darf man PhoneGap und Sencha Touch nicht gleichsetzen, da beide Frameworks ziemlich unterschiedliche Ansätze verfolgen. PhoneGap dient als eine Art Wrapper für Web-Applikationen. Es sieht nach außen aus, als handle es sich um native Anwendungen, letztlich besteht es aber lediglich aus HTML und JavaScript. Es ist deshalb auch möglich Sencha Touch (und andere Frameworks) innerhalb von PhoneGap zu benutzen [128]. Ein weiterer großer Vorteil von PhoneGap besteht darin, dass das Framework die nativen APIs der Endgeräte ansteuern kann um so Hardwarekomponenten wie z.B. Kameras, GPS-Empfänger oder Beschleunigungssensoren von Smartphones anzusprechen. Man muss als Entwickler auch keine Fallunterscheidungen für verschiedene Geräte einbauen, da dies alles intern von PhoneGap geregelt wird. Ursprünglich entwickelt wurde PhoneGap von Nitobi [129], welches inzwischen

von Adobe aufgekauft wurde. Um auf lange Sicht den Open-Source Gedanken von PhoneGap zu sichern, haben die Entwickler den PhoneGap-Quellcode als Apache Projekt bereitgestellt. PhoneGap wird neben Adobe auch von anderen großen Unternehmen wie z.B. Cisco, IBM, Time Warner oder Logitech eingesetzt und teils weiter entwickelt.

Einen ähnlichen Ansatz verfolgt AppCelerator Titanium [130], welches eine Art „Hybrid-App“ ermöglicht. Programmiert wird mit Hilfe typischer Web-Sprachen wie JavaScript, Ruby oder Python. Trotzdem können die nativen Widgets wie Tabellen- und Scroll-Ansichten, native Buttons, Schalter, Tabs, Popovers und andere benutzt werden. Dabei gewährt Titanium Zugriff auf über 300 APIs.

4.2.4 Native Anwendungen

Im Gegensatz dazu stehen komplett native Anwendungen, wie die Anwendungen der Tagesschau [131] oder des Spiegels [132]. Auf Android-Geräten werden diese Anwendungen typischerweise mit Java, auf iOS-Geräten mit Objective-C programmiert. Genau hier liegt aber auch das Problem dieses Ansatzes. Auftraggeber mobiler Anwendungen wollen oft, dass deren Anwendungen auf den unterschiedlichsten Endgeräten laufen, wollen aber nicht für jede einzelne Plattform zahlen. Vor ein paar Jahren galt noch Java ME (Java Micro Edition) als der heilige Gral der plattformunabhängigen Entwicklung, der es ermöglichen sollte, eine Anwendung einmal zu entwickeln und dann auf jeder beliebigen Umgebung ausführen zu können. In der Praxis ist dies allerdings Wunschdenken geblieben. Neben Fragmentierungsproblemen gab es einfach zu viele Parteien mit unterschiedlichen Interessen, die bei der Standardisierung mitmischten und eine zügige Fortentwicklung verhinderten. Schnell wanderten die Entwickler von dieser Plattform ab [133].

In diese Bresche springen nun wieder andere Frameworks. Mit Rhodes [134] z.B. können native, mobile Anwendungen in Ruby entwickelt werden. Diese Anwendungen sind explizit keine Webanwendungen. Neben der Plattformunabhängigkeit (iPhone, Android, RIM, Windows Mobile und Windows Phone 7) gilt das integrierte Model, View, Controller-Design als ein großer Vorteil.

5 WebGL

5.1 Einführung in WebGL

Bereits in den neunziger Jahren gab es Bestrebungen, 3D im Webbrowser verfügbar zu machen. Im Jahr 1994 begann die Arbeit an der „Virtual Reality Markup Language“, kurz VRML. Gründe für das Scheitern dieser Technologie gab es einige. So waren in den Neunzigern weder die Bandbreite der Internetanschlüsse noch die Leistung der Grafikhardware in gewöhnlichen PCs ausreichend um VRML zu rendern. Außerdem musste der Benutzer ein Plugin für seinen Browser laden und selbst dann war die Kommunikation zwischen Browser und 3D-Szene nicht wirklich komfortabel [135].

WebGL [136] (Web Graphics Library) bezeichnet eine Bibliothek die fester Bestandteil neuerer Webbrowser ist und hardwarebeschleunigte 3D-Grafiken auf OpenGL-Basis im Browser ermöglicht. Dank offizieller Spezifikation ist der Standard plattformunabhängig und lizenzfrei. Die Grafiken können direkt im Browser dargestellt werden, ohne zusätzliche Plugins wie Adobe Flash installieren zu müssen. Stattdessen können sie durch das HTML5 `canvas`-Element im Webbrowser zur Verfügung gestellt werden. Entwickelt wird WebGL seit 2009 von der Khronos Group und Mozilla. Inzwischen sind alle großen Browser-Hersteller (außer Microsoft) Mitglieder der WebGL Working Group. Außerdem beteiligen sich viele andere IT-Unternehmen u. a. AMD, Ericsson, Nvidia an der Entwicklung des Projektes. Die aktuelle Version 1.0 wurde im März 2011 verabschiedet.

Insgesamt unterstützen Ende 2011 außer des Internet Explorers alle großen Browser die WebGL-Erweiterungen [137]. Microsoft weigert sich WebGL in den Internet Explorer einzubinden, da man Sicherheitsbedenken hat. Laut Microsoft stelle WebGL Hardwarefunktionen zu freizügig im Web bereit. Daher würde die Sicherheit von WebGL in starkem Maße von den Grafikkartentrei-

bern abhängen und das würde deren Entwickler vor Probleme stellen, die sie bisher nicht kannten [138]. Microsoft wurde für diese Verweigerungspolitik sogar aus den eigenen Reihen gerügt [139]. Trotzdem sollte man in nächster Zeit nicht mit einer WebGL-Implementierung im Internet Explorer rechnen. Zum Nachrüsten der 3D-Unterstützung muss dazu entweder Chrome Frame [140] oder das Plugin IEWebGL [141] installiert werden. Beide laufen unter dem Internet Explorer 6 bis 9.

Die Spezifikation [142] ist sehr nahe an der Original-OpenGL-Spezifikation angelegt, musste aber an einigen Stellen vereinfacht werden, um den begrenzten Möglichkeiten von JavaScript entgegenzukommen. WebGL ist keine 3D-Engine im klassischen Sinne sondern eine standardisierte Low-Level-3D-Grafik-Programmierschnittstelle für Webbrowser, die auf OpenGL ES in Version 2.0 basiert und mit JavaScript zusammenspielt. Da WebGL eine DOM API ist, kann sie eigentlich aus allen DOM-kompatiblen Programmiersprachen wie JavaScript, Java oder Objective C verwendet werden. Im Internet dient JavaScript als Bindeglied zwischen Browserumgebung und Grafikhardware. Da es in dieser Programmiersprache aber keine Möglichkeit für den direkten Zugriff auf Binärdaten gibt, mussten Vorkehrungen getroffen werden, damit die Bereitstellung von großen Datenmengen z.B. in Form von Texturen oder Polygondaten nicht zum Engpass wird. Aus diesem Grund wurde neben WebGL eine weitere Spezifikation erarbeitet, welche sich „Typed Arrays“ nennt. In dieser wird ein Interface für den Zugriff auf Binärdaten definiert [143]. Grafiker können Inhalte mit Tools wie Blender [144] oder Maya [145] erstellen und die 3D-Szenen dann nach WebGL exportieren.

Da HTML als Markup-Sprache keine Unterstützung für freies Zeichnen innerhalb einer Webseite bietet, gab es bereits 2001 Bemühungen diesen Mangel zu beseitigen. Joe Hewitt, der spätere Entwickler von Firebug, programmierte als erster einen einfachen Prototyp für ein `canvas`-Element [146]. Doch erst

mit der Konkretisierung des HTML5-Standards kam ein offener Dialog über die Umsetzung eines solchen Elements zu Stande. Propagiert wird in der offiziellen Spezifikation zum `canvas`-Element ein sogenannter 2D-Kontext zum Zeichnen im Bereich des `canvas`-Elements [147]. Die Entwickler rund um die Khronos Group setzen für WebGL das `canvas`-Element nun äquivalent als 3D-Kontext ein.

Bei WebGL handelt es sich um einen prozeduralen Ansatz (immediate mode), bei welchem Grafikaufrufe direkt ausgeführt werden. Es ist nicht möglich, eine Liste von Objekten zu speichern und diese dann pro Frame zu zeichnen oder nur die geänderten Objekte neu zu zeichnen. Dieser Ansatz wird als deklarativ bezeichnet (retained mode) und er wurde bei VRML verwendet. Auch die Frameworks, welche bereits für WebGL existieren, funktionieren nach diesem Prinzip.

5.2 WebGL Frameworks

Eine Kritik, die immer wieder in Bezug auf WebGL auftaucht, ist der enorme Programmieraufwand, dem sich Entwickler zu stellen haben. Einarbeitung in die OpenGL Shading Language (GLSL) [148] als eigene Programmierumgebung und mathematisches Know-how sind unerlässlich. Obwohl WebGL eine relativ junge Technologie ist, haben sich bereits unzählige Frameworks etabliert. Im Folgenden werden einige wichtige Frameworks zur WebGL-Programmierung vorgestellt. Es kann aufgrund der großen Anzahl vorhandener Frameworks nur auf einen Bruchteil dieser Werke eingegangen werden. Eine vollständige Liste aller Frameworks findet sich auf der Website von Khronos [149].

5.2.1 three.js

Das Ziel von three.js ist es, eine 3D Engine zu entwickeln, die sich durch eine sehr niedrige Lernkurve auszeichnet. Obwohl das Hauptaugenmerk auf WebGL liegt, ist es mit dem Framework auch möglich 3D-Szenen in `canvas` oder `svg` auszugeben. Ins Leben gerufen wurde das Framework im April 2010 von einem Entwickler namens „Mr. doob“. Inzwischen arbeiten mehr als 30 Entwickler an dem Framework mit.

5.2.2 Jax

„Jax is a game-changer“. Mit diesem Slogan bewirbt Colin MacKenzie IV sein WebGL-Framework an dem er seit März 2011 arbeitet [150]. Laut MacKenzie IV sei Jax [151] nicht nur ein weiteres Framework, das einen WebGL-Kontext beinhalte und dann vom Entwickler erwarte, dass dieser den größten Teil der Entwicklung selber erarbeitet. Stattdessen soll Jax als Komplettpaket verstanden werden. Obwohl es ursprünglich nicht als ein Ruby on Rails-Adapter gedacht war, so weist es doch heute einige Ähnlichkeiten zu dem Web-Framework auf. Laut Entwickler soll Jax die Produktivität der Rails-Entwicklung in die JavaScript- und im Besonderen auf die WebGL-Welt übertragen. In der Tat kann man im Quellcode der Jax-Beispiele zahllose Ähnlichkeiten zur Rails-Entwicklung finden. Dies beginnt damit, dass Jax selber als Rubygem veröffentlicht wird, und so in ein bestehendes Rails-Projekt eingebunden werden kann, ohne aber Rails zwingend als Abhängigkeit zu erwarten [152].

Das Framework ist eine Kombination von JavaScript- und Ruby-Quellcode, das Ergebnis aber immer reines JavaScript. Dies macht es möglich ein Jax-Programm in jedem Web-Projekt zu nutzen. Jax übernimmt dabei Philosophien von Ruby on Rails wie „Don’t repeat yourself“ oder „Convention over Configuration“. Außerdem wurde das Model, View, Controller-Architekturmodell übernommen, was die Codebasis generell leichter wartbar

macht. Die Models repräsentieren Container die Daten und Regeln beinhalten. Diese beschreiben wie Daten verändert werden können. Sobald feststeht, dass eine bestimmte Aktion ausgeführt werden soll, ist es Aufgabe des Models die zugehörigen Daten zu verändern. Models beinhalten demzufolge den Großteil der Business-Logik. Die Views arbeiten als Präsentations-Layer. Sie bearbeiten keine Daten, nehmen keine Benutzereingaben entgegen und beinhalten auch keine Logik, die nicht für die Präsentation notwendig wäre. Stattdessen rendern sie die aktuelle Szene und können so z.B. visuelle Effekte zu einer Szene hinzufügen. Die Controller teilen den Models mit wann sie was machen sollen. In Jax sind sie dafür verantwortlich die aktuelle Szene auf- und später auch wieder abzubauen. Außerdem verarbeiten sie alle Benutzereingaben [153].

5.2.3 PhiloGL

PhiloGL [154] ist ein WebGL-Framework zur Daten-Visualisierung und Spiel-Entwicklung, das kreatives Coding unterstützt. Laut der Feature-Liste wurde es mit dem Hintergedanken entwickelt, so nah wie möglich an den GL-Aufrufen zu bleiben um eine klare aber trotzdem enge Abstraktion für WebGL zu sein. Dabei werden gute Ansätze und Idiome von JavaScript übernommen um so eine ausdrucksstarke und mächtige API bereitstellen zu können. PhiloGL ist modular aufgebaut, so dass man benötigte Module z.B. für Ajax-Requests einfach nachladen kann. Entwickelt wurde das Framework seit Februar 2011 von Nicolas Garcia Belmonte unter der Schirmherrschaft von Sencha Inc.

5.2.4 GLGE

Auch GLGE ist eine JavaScript-Bibliothek mit der Intention die Benutzung von WebGL zu vereinfachen: „GLGE is WebGL for the lazy“ [155]. Das Ziel von GLGE ist es, das involvierte WebGL zu maskieren und so dem Entwickler mehr Zeit zum Erstellen der Inhalte zu geben. Wie eigentlich alle hier vorgestellten

Frameworks ist auch GLGE noch immer in der Entwicklung, und so kann es sein, dass Features von heute auf morgen hinzukommen oder gestrichen werden.

6 Cloud Computing

6.1 Was man unter der Cloud versteht

Die Idee des Cloud Computing ist eigentlich recht alt, die Cloud hieß vor etlichen Jahren lediglich noch Mainframe. Die Unternehmensphilosophie von Sun Microsystems, in den Achzigern ein führender Anbieter von Mainframes, lautete damals „The Network is the Computer“. Die Vision war, dass leistungsfähige Großrechner die Arbeit übernehmen, während Nutzer an einfachen und preiswerten Rechnern, sogenannten Thin Clients, arbeiten. Deren Rechenleistung wird nur für die Benutzerschnittstelle gebraucht. Da die Software in der Cloud läuft, muss man keine Gedanken an die Hardware verschwenden: Speicherplatz, Rechenleistung und Anwendungen liegen in der Cloud, jederzeit nutzbar, ohne Installation auf dem lokalen Rechner. Auch Updates nach jeder neuen Sicherheitslücke werden automatisch ohne das Zutun des Anwenders eingespielt.

Laut einer Studie aus dem Frühjahr 2011 [156] arbeiten 70% der 235 befragten Unternehmen mit mehr als 250 Mitarbeitern an einer Cloud-Strategie. 29% der Unternehmen wollen Cloud Services in so vielen Bereichen wie möglich nutzen und beabsichtigen eine relativ breit angelegte Herangehensweise. Es gibt unterschiedlichste Motivationen, warum ein Unternehmen einen cloud-basierten Ansatz verfolgt. Die beiden wichtigsten Antriebsfaktoren für die Nutzung von Cloud-Diensten hängen eng miteinander zusammen: „Keine zusätzliche Anschaffung von IT-Infrastruktur“ (48%) und „Reduzierung von Kosten“ (35%). Neben den wegfallenden Investitionen in neue IT-Infrastruktur sollten zudem Einsparungen in den Bereichen Administration, Wartung, Upgrades, Betrieb und Gewährleistung der Verfügbarkeit einberechnet werden. Weitere Einsparungen ergeben sich bei Softwarelösungen zudem durch den Wegfall der Einmalzahlungen von Lizenzkosten. Dafür müssen aber Kosten der Nutzung nach Verbrauch (z.B. pro User/ Monat) einkalkuliert werden.

6.2 Das Betriebssystem in der Cloud

Der Weg für Cloud-Anwendungen ist also seit Jahrzehnten vorgezeichnet, nur das Internet war noch nicht so weit: Was lange fehlte waren Breitbandanschlüsse für jeden, um ganze Applikationen auf die Rechner der Nutzer und deren Daten wieder zurück ins Netz transportieren zu können. Die kritische Masse an Breitbandanschlüssen und Nutzern ist erreicht, so dass sich Clouds auch für deren Anbieter rechnen. Heute übernimmt die Cloud immer mehr zentrale Aufgaben. Das lokale Betriebssystem wird zusätzlich immer unwichtiger. Und obwohl sowohl Microsoft als auch Apple Cloud-Dienste immer weiter integrieren, klammern sie sich doch an den lokalen Betriebssystemen fest. Immerhin verdienen die beiden Firmen mit dem Verkauf dieser Software sehr viel Geld. Und auch der Wunsch, die Nutzer an sich zu binden spielt eine große Rolle. Wer sich einmal in die Fänge einer Cloud begeben hat, wird so schnell nicht in eine andere wechseln. Es wird von Tag zu Tag schwieriger, all seine Daten wieder aus der Cloud zu holen und sie der Konkurrenz anzuvertrauen, bei der alles ganz anders funktioniert und die im schlechtesten Fall für spezifische Anwendungen gar keine Clouds anbietet.

Einen sehr konsequenten Weg beschreitet Google mit seinem Chromebook [157], einem Netbook[6] in Formvollendung. Dessen Betriebssystem „Chrome OS“ [158] besteht fast nur aus dem hauseigenen Chrome-Browser. Es arbeitet ohne lokal installierte Software komplett webbasiert. Sowohl Programme als auch Daten kommen aus der Cloud, neue Software gibt es im Chrome Web Store [159]. Um Viren, Würmer und Sicherheitsupdates kümmern sich stattdessen Googles Rechenzentren.

[6] Ein Netbook ist kleiner und günstiger als ein Notebook, bringt aber auch weniger Rechenleistung mit. Da es als tragbarer Internet-Client konzipiert ist, verfügt es üblicherweise über integriertes WLAN, teilweise sogar über ein integriertes Mobilfunk-Modem.

Einen ähnlichen Ansatz verfolgt Mozilla mit „Boot2Gecko“ (B2G) [160], welches Ende Februar 2012 auf dem Mobile World Congress in Barcelona vorgestellt wurde. Die Basis bildet in diesem Projekt ein Linux-Kernel und einige Teile des Android-Projekts. Diese Basis-„Schicht“ heißt „Gonk“ und Mozilla braucht sie hauptsächlich für die Treiberunterstützung mobiler Endgeräte. Die mittlere Schicht ist die von Firefox bekannte Rendering-Engine „Gecko“, welche mit vielen APIs stark aufgebessert wurde. Auf diesen beiden technischen Fundamenten liegt schließlich „Gaia“, die Nutzeroberfläche des Betriebssystems. Gaia wird ausschließlich in bekannten Web-Technologien, wie HTML und JavaScript, programmiert. Während der Browser das Betriebssystem ersetzt, verdrängen Online-Programme auf Basis von HTML5 die nativen Programme. Auf HTML5 basierend werden sich nicht alle Anwendungen reproduzieren lassen, die man vom Android- oder Apple-Markt her kennt. Doch von rund 80% geht Mozilla-Vizepräsident Jay Sullivan in einem Interview mit der New York Times [161] schon aus. Es ist vorgesehen, dass ein App-Store für B2G entsteht [162], auf dem auch kostenpflichtige Anwendungen angeboten werden können.

6.3 Beispiel Cloud-Programmierung: Deltacloud

Ein großes Problem der immer weiter wachsenden Zahl der Cloud-Anbieter sind deren unterschiedlichen Schnittstellen, die man als Programmierer ansprechen muss, da jede dieser Clouds seine eigene API bereitstellt und diese nicht genormt sind. Dieses Problems hat sich Deltacloud [163] angenommen, das eine REST-basierte API für möglichst viele verschiedene Cloud-Anbiter bereitstellen möchte. Jede einzelne Cloud wird durch einen Adapter (englisch „driver“) kontrolliert. Zusätzlich werden Client-Bibliotheken für verschiedene Programmiersprachen wie C, C++ und Ruby bereitgestellt. Sollte sich die API eines Cloud-Anbieters ändern, muss vom Programmierer lediglich der zugehörige Adapter aktualisiert werden, welcher die API-Änderungen abdeckt.

Im Winter 2011 werden unter anderem die Cloud-Angebote von Red Hat, Amazon EC2, Eucalyptus, GoGrid, IBM, Microsoft, OpenStack, OpenNebula und Rackspace unterstützt. Die vollständige Liste aller unterstützten Clouds findet sich auf der Projektseite [164]. Ursprünglich wurde Deltacloud von Red Hat entwickelt, im Mai 2010 aber in die Hände der Apache Software Foundation übergeben. Inzwischen ist es dem Apache Incubator Projekt entwachsten und somit ein Apache Software Foundation TLP (Top-Level Project).

6.4 Software as a Service: Amazon Dienste

Amazon bietet unter dem Namen „Amazon Web Services“ (AWS) eine große Palette an Cloud-Services an, die die unterschiedlichsten Bereiche abdecken [165]. Gestartet wurden die Amazon Web Services im Juli 2002 als Dienst für Amazon-fremde Webseiten oder Clientseitige Anwendungen. Der größte Teil der Anwendungen wird über HTTP transportiert. Laut eigenen Angaben waren im Februar 2009 bereits 490000 Entwickler für die Nutzung der Amazon Web Services registriert [166]. Dabei richten sich die Angebote Amazons nicht direkt an Endkunden, sondern an andere Entwickler, die die Services in ihre Produkte integrieren. Stellvertretend für andere Anbieter wird hier eine Auswahl der Amazon-Services vorgestellt.

Der bekannteste Dienst der Amazon Web Services ist die Amazon Elastic Compute Cloud (EC2). Dieser Dienst existiert seit 2006 und ist eine Art virtueller Host. Es handelt sich um ein „Infrastructure as a Service“-Angebot (IaaS), bei der Rechenleistung „on demand“ (englisch für „bei Bedarf“) gemietet werden kann. Dies bietet einige Vorteile gegenüber dem klassischem Hardware-Kauf. So können Belastungsspitzen sehr gut abgefangen werden, da bei Bedarf einfach zusätzliche Rechenkapazität hinzugebucht werden kann und die Hardware so sehr gut skaliert. Im Gegensatz dazu können Ressourcen, die nicht mehr gebraucht werden, sofort wieder abgegeben werden. Da durch Vir-

tualisierungstechnologie viele verschiedene Systeme laufen können, bietet sich EC2 zum Beispiel zum Softwaretest an. Dazu können sowohl vorgegebene sogenannte „Amazon Machine Images“ (AMI) ausgewählt werden, als auch eigene erstellt werden. Diese AMIs enthalten ein bestimmtes Betriebssystem, verschiedene Software oder installierte Datenbanken. Zu bedenken ist dabei, dass eine solche Instanz von Haus aus nicht persistent ist. Nach dem Beenden einer Instanz werden alle Änderungen, die während der Laufzeit durchgeführt wurden, rückgängig gemacht. Zur Speicherung von Zuständen kann wiederum ein anderer Amazon Web Service, der Amazon Elastic Block Store, verwendet werden.

Der Amazon Simple Storage Service (S3) ist ein File Hosting Service, welcher Daten in sogenannten Buckets organisiert. Diese sind amazonweit einzigartig und können über eine URL adressiert werden. Neben der eigentlich gespeicherten Datei werden auf S3 auch noch diverse Metadaten wie beispielsweise Content-Type oder Datum der letzten Veränderung gespeichert. Der Zugriff auf die Daten ist über SOAP, REST oder sogar BitTorrent möglich. Es existieren fertige Open Source-Bibliotheken, mit denen ein Zugriff auf den Storage Service sehr komfortabel möglich ist.

7 Fazit

Aufgrund der Schnelllebigkeit ist es fast unmöglich einen „Ist“-Zustand der Web-Programmierung zu ergreifen. Die Entwicklung von Web-Frameworks z.B. schreitet in einer Geschwindigkeit voran, die es selbst für Entwickler schwierig werden lässt, Schritt zu halten. Kaum hat man sein Projekt an die Neuigkeiten der letzten Framework-Version angepasst, folgt schon die nächste Version des zugrundeliegenden Frameworks. Außerdem ändern sich sie Popularitätswerte der einzelnen Frameworks rasant, so dass man als Entwickler immer einen Blick über den eigenen Tellerrand wagen sollte, um nicht den Anschluss zu verlieren. Der Schwerpunkt dieser Arbeit liegt auf dem Ruby on Rails Framework, welches besonders intensiv betrachtet wird. Es wird sich mit wichtigen Konzepten der Softwareentwicklung wie z.B. der „Model, View, Controller“-Architektur beschäftigt, die auch in anderen Frameworks wie Symfony oder FLOW3 Anwendung findet.

Das Thema des „mobile Computing“ ist eine vergleichsweise recht junge Disziplin, was sich auch darin begründet, dass mobile Endgeräte erst seit ein paar Jahren ein sprunghaftes Wachstum verzeichnen konnten. Da durch sinkende Preise für Online-Tarife diese Geräte auch unterwegs oft mit dem Internet verbunden sind, wuchs das Interesse der Anwender an Webseiten, die für Mobilgeräte optimiert sind. Diese Umstände haben bei den Anbietern von Webanwendungen Strategien für das Online-Geschäft entstehen lassen, welche mobile Webseiten beinhalteten. Zügig entstanden damit auch die ersten mobilen Frameworks, um die neuen Bedürfnisse der Entwickler abzudecken. Diese Arbeit gibt einen Überblick über Frameworks, die zum Zeitpunkt dieser Arbeit aktuell sind. Interessant sind die recht unterschiedlichen Ansätze, die die verschiedenen Projekte verfolgen. Während z.B. Sencha Touch oder jQuery Mobile versuchen, die verschiedenen APIs der unterschiedlichen mobilen Endgeräte zu

kapseln, agiert PhoneGap als eine Art „Wrapper“ um Webanwendungen in die verschiedenen App-Stores zu bringen.

Obwohl bereits in den neunziger Jahren versucht wurde, 3D-Inhalte im Internet populär zu machen, ist eine weite Verbreitung auch im Jahre 2011 noch nicht gelungen. Wenigstens existiert vom World Wide Web Consortium nun ein Standard und mit Hilfe verschiedenster Frameworks ist es einfacher geworden WebGL-Inhalte komfortabel umzusetzen. Spannend wird die Frage wie sich Microsoft letztendlich im Bezug auf die Unterstützung von WebGL im Internet Explorer verhält. Damit diese Technologie endgültig im Massenmarkt ankommen kann, ist die Kooperation der Firma aus Redmond unbedingt nötig, da deren Browser nach wie vor große Marktanteile hält. Diese Arbeit versucht einen Beitrag zum WebGL-Verständnis zu liefern, indem verschiedene Frameworks verglichen werden.

Kaum ein Thema ist so stark mit sowohl positiven wie auch negativen Emotionen verbunden wie das des Cloud Computings. Während man sich auf der einen Seite Zeit- und damit Kosteneinsparungen verspricht, muss man auf der Anderen neben des Totalausfalls der Cloud vor allem den fehlenden Datenschutz fürchten. Es gilt ein gesundes Mittelmaß an Cloud-Nutzung zu finden und immer abzuwägen, ob man die zu verarbeitenden oder zu speichernden Daten einer Cloud anvertrauen möchte. Die Cloud-Anbieter sind in der Pflicht größtmögliche Transparenz zu schaffen. Im Rahmen dieser Arbeit wird auf einige Amazon-Dienste genauer eingegangen. Außerdem werden Ansätze wie „Google Chrome“ oder das sehr aktuelle „Boot2Gecko“ beleuchtet, die versuchen das ganze Betriebssystem in die Cloud auszulagern.

Literatur

[1] Jochen Musch. *Die Geschichte des Netzes: ein historischer Abriß*, 2010. http://www.uni-duesseldorf.de/home/Fakultaeten/math_nat/WE/Psychologie/abteilungen/ddp/Dokumente/Publications/1997.Musch.Die_Geschichte_des_Netzes.html.

[2] CERN. *Website des CERN*, 2011. http://www.cern.ch.

[3] Tim Berners-Lee. *Information Management: A Proposal*, 1989. http://www.w3.org/History/1989/proposal.html.

[4] Hassan El Moussaoui und Klaus Zeppenfeld. *Ajax - Geschichte, Technologie, Zukunft*. Springer, 2008.

[5] Tim Berners-Lee. *Statement concerning CERN W3 Software release into Public Domain*, 1993. http://tenyears-www.web.cern.ch/tenyears-www/Welcome.html.

[6] Richard P. Gabriel and Guy L. Steele Jr. *The Evolution of Lisp*. http://www.dreamsongs.com/NewFiles/Hopl2.pdf.

[7] Vannevar Bush. *As We May Think*, 1945. http://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/3881/.

[8] Ted Nelson. *Complex information processing: a file structure for the complex, the changing and the indeterminate*, 1965. http://dl.acm.org/citation.cfm?id=806036.

[9] Project Xanadu. *Xanandu Website*, 1965. http://www.xanadu.com/.

[10] University of Illinois. *NCSA Website*, 2011. http://www.ncsa.illinois.edu/.

[11] University of Illinois. *About NCSA Mosaic*, 2011. http://www.ncsa.illinois.edu/Projects/mosaic.html.

[12] Dr. Robert Cailliau. *Co-developer of the WWW: part one*, 2008. http://dl.acm.org/citation.cfm?id=1408626.1408636.

[13] John Borland. *Browser wars: High price, huge rewards*, 2003. http://www.zdnet.com/news/browser-wars-high-price-huge-rewards/128738.

[14] Microsoft Corporation. *Internet Explorer Website*, 2011. http://windows.microsoft.com/en-US/internet-explorer/products/ie/home.

[15] U.S. District Judge. *United States of America vs. Microsoft Corporation - The middleware threats*, 1999. http://www.justice.gov/atr/cases/f3800/msjudgex.htm#iv.

[16] Mozilla Foundation. *Mozilla.org Website*, 2011. http://www.mozilla.org/.

[17] Mozilla Foundation. *Mozilla Firefox Website*, 2011. http://www.mozilla.org/en-US/firefox/.

[18] Heise Zeitschriften Verlag GmbH & Co. KG. *Microsofts Internet Explorer verliert gegenüber Mozilla/Firefox Anteile*, 2004. http://heise.de/-105259.

[19] Sundar Pichai. *Blog-Post von Google der den Browser Chrome ankündigt*, 2008. http://googleblog.blogspot.com/2008/09/fresh-take-on-browser.html.

[20] StatCounter. *StatCounter Global Stats - Top 9 Browsers from December 2008 to February 2012*, 2012. http://gs.statcounter.com/#mobile_browser-ww-monthly-200812-201203.

[21] StatCounter. *Mobile internet usage is doubling year on year*, 2012. http://gs.statcounter.com/press/mobile-internet-usage-is-doubling-year-on-year.

[22] Paul Golding. *Connected Services: A Guide to the Internet Technologies Shaping the Future of Mobile Services and Operators*. John Wiley & Sons, 2011.

[23] SELFHTML e.V. *Allgemeines zu Dynamischem HTML*, 2007. http://de.selfhtml.org/dhtml/intro.htm.

[24] W3C DOM IG. *Document Object Model (DOM)*, 2005. http://www.w3.org/DOM/.

[25] Jens Behrendt und Klaus Zeppenfeld. *Web 2.0*. Springer, 2008.

[26] The jQuery Project. *jQuery: The Write Less, Do More, JavaScript Library*, 2010. http://jquery.com/.

[27] Valerio Proietti. *MooTools - a compact javascript framework*, 2009. http://mootools.net/.

[28] Prototype Core Team. *Prototype JavaScript framework: Easy Ajax and DOM manipulation for dynamic web applications*, 2007. http://www.prototypejs.org/.

[29] BuiltWith.com. *JavaScript Libraries and Functions Distribution*, 2011. http://trends.builtwith.com/javascript.

[30] Google Inc. *Google Reader*, 2011. http://www.google.com/reader/view/.

[31] Ian Hickson. *The WebSocket API - Editor's Draft*, 2011. http://dev.w3.org/html5/websockets/.

[32] Internet Engineering Task Force. *The WebSocket Protocol*, 2011. http://tools.ietf.org/html/rfc6455.

[33] Alexis Deveria. *When can I use - WebSockets*, 2011. http://caniuse.com/#search=websocket.

[34] Bertrand Meyer. *Object-oriented software construction*. PRENTICE-HALL INTERNATIONAL SERIES IN COMPUTER SCIENCE. Prentice Hall PTR, 1997.

[35] Oliver Vogel, Ingo Arnold, Arif Chughtai, and Timo Kehrer. *Software Architecture: A Comprehensive Framework and Guide for Practitioners*. Springer, 2011.

[36] The jQuery Project. *CDN Hosted jQuery*, 2011. http://docs.jquery.com/Downloading_jQuery#CDN_Hosted_jQuery.

[37] The jQuery Project. *jQuery API*, 2011. http://api.jquery.com/jQuery/.

[38] The jQuery Project. *jQuery API - jQuery.get()*, 2011. http://api.jquery.com/jQuery.get/.

[39] 37signals LLC. *Rails Website*, 2011. http://www.rubyonrails.org.

[40] Ralf Wirdemann und Thomas Baustern. *Rapid Web Development mit Ruby on Rails*. Hanser Fachbuchverlag, 2008.

[41] 37signals, LLC. *37signals: Web-based collaboration apps for small business*, 2011. http://37signals.com/.

[42] 37signals, LLC. *Project management software, online collaboration: Basecamp*, 2010. http://basecamphq.com/.

[43] Ruby Visual Identity Team. *Über Ruby*. http://www.ruby-lang.org/en/about/.

[44] Mikel Lindsaar, Mike Gunderloy, et al. *The MVC Architecture*, 2011. http://guides.rubyonrails.org/getting_started.html#the-mvc-architecture.

[45] Trygve Reenskaug. *MVC XEROX PARC 1978-79*, 1979. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

[46] The Internet Society. *The application/json Media Type for JavaScript Object Notation (JSON) RFC*, 2006. http://tools.ietf.org/html/rfc4627.

[47] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition) RFC*, 2008. http://www.w3.org/TR/REC-xml/.

[48] Mike Gunderloy et al. *JSON, XML und andere Formate in Rails rendern*, 2010. http://guides.rubyonrails.org/layouts_and_rendering.html#rendering-text.

[49] Mike Gunderloy et al. *Convention Over Configuration in Action*, 2010. http://guides.rubyonrails.org/layouts_and_rendering.html#rendering-by-default-convention-over-configuration-in-action.

[50] Mike Gunderloy et al. *Configuring a Database*, 2011. http://guides.rubyonrails.org/getting_started.html#configuring-a-database.

[51] Sun Microsystems Inc. *JavaBeans Dokumentation*, 1997. http://www.cs.vu.nl/~eliens/documents/java/white/beans.101.pdf.

[52] Andrew Hunt und David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Pearson Education, 1999.

[53] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[54] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. *HTML 4 Spezifikation*, 1999. http://www.w3.org/TR/html4/interact/forms.html#h-17.13.1.

[55] Ian Hickson. *HTML 5 Spezifikation*, 2011. http://dev.w3.org/html5/spec/Overview.html#attr-fs-method.

[56] Mislav Marohnić et al. *How do forms with PUT or DELETE methods work?*, 2011. http://guides.rubyonrails.org/form_helpers.html#how-do-forms-with-put-or-delete-methods-work.

[57] Nick Sutterer. *Rails Misapprehensions: CRUD is not REST!*, 2010. http://nicksda.apotomo.de/2010/10/rails-misapprehensions-crud-is-not-rest/.

[58] Mike Gunderloy et al. *Getting Up and Running Quickly with Scaffolding* , 2011. http://guides.rubyonrails.org/getting_started.html#getting-up-and-running-quickly-with-scaffolding.

[59] Martin Fowler. *Patterns of enterprise application architecture*. The Addison-Wesley signature series. Addison-Wesley, 2003.

[60] David Heinemeier Hansson, Jeremy Kemper, et al. *ActiveRecord*, 2008. http://ar.rubyonrails.org/.

[61] Frederick Cheung et al. *Einführung in Migrations*, 2011. http://guides.rubyonrails.org/migrations.html.

[62] Akshay Surve, Mike Gunderloy, et al. *Unit Testing your Models*, 2010. http://guides.rubyonrails.org/testing.html#unit-testing-your-models.

[63] Akshay Surve, Mike Gunderloy, et al. *Functional Tests for Your Controllers*, 2010. http://guides.rubyonrails.org/testing.html#functional-tests-for-your-controllers.

[64] Akshay Surve, Mike Gunderloy, et al. *Integration Testing*, 2010. http://guides.rubyonrails.org/testing.html#integration-testing.

[65] Pivotal Labs. *BDD for JavaScript*, 2011. http://pivotal.github.com/jasmine/.

[66] Akshay Surve, Mike Gunderloy, et al. *What are Fixtures?*, 2010. http://guides.rubyonrails.org/testing.html#what-are-fixtures.

[67] Akshay Surve, Mike Gunderloy, et al. *The Three Environments*, 2010. http://guides.rubyonrails.org/testing.html#the-three-environments.

[68] Ruby Visual Identity Team. *Ruby Bibliotheken*. http://www.ruby-lang.org/en/libraries/.

[69] *Rubygems.org*, 2011. http://rubygems.org/.

[70] *Capistrano Projektseite*, 2011. https://github.com/capistrano/capistrano.

[71] The Rack Core Team. *Rack Dokumentation*, 2010. http://rack.rubyforge.org/doc/.

[72] Blake Mizerany. *Sinatra Website*, 2011. http://www.sinatrarb.com/.

[73] *Camping, a Microframework*. http://camping.rubyforge.org/.

[74] Christian Neukirchen. *Rack: a Ruby Webserver Interface*, 2010. http://rack.rubyforge.org/.

[75] Jim Weirich. *Rake Website*, 2004. http://rake.rubyforge.org/.

[76] Ruby on Rails Team. *Rake Beispiele für Rails*, 2011. http://guides.rubyonrails.org/command_line.html#rake.

[77] Ruby on Rails Team. *Die Rails Konsole*, 2011. http://guides.rubyonrails.org/command_line.html#rails-console.

[78] *Interactive Ruby Shell*. http://ruby-doc.org/docs/ProgrammingRuby/html/irb.html.

[79] Sensio Labs. *High Performance PHP Framework for Web Development - Symfony*, 2011. http://symfony.com/.

[80] Timo Haberkern. *Das Symfony Framework - Enterprise-Anwendungen mit PHP*. entwickler.press, 2008.

[81] Jonathan H. Wage, Guilherme Blanco, Benjamin Eberlei, Bulat Shakirzyanov, Juozas Kaziukenas, et al. *Doctrine - PHP Object Persistence Libraries and More*, 2011. http://www.doctrine-project.org/.

[82] Jonathan H. Wage, Guilherme Blanco, Benjamin Eberlei, Bulat Shakirzyanov, Juozas Kaziukenas, et al. *Doctrine-Einbindung in Symfony*, 2011. http://symfony.com/doc/current/book/doctrine.html.

[83] *Doctrine Explained*. http://www.doctrine-project.org/projects/orm/1.2/docs/manual/introduction/en#doctrine-explained.

[84] TYPO3 Association. *FLOW3 Enterprise PHP Framework*, 2011. http://flow3.typo3.org/.

[85] TYPO3 Association. *FLOW3 1.0 Release Notes*, 2011. http://flow3.typo3.org/download/release-notes/flow3-1-0.html.

[86] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*, 2004. http://martinfowler.com/articles/injection.html.

[87] Robert Lippert and Robert Lemke. *Robert Lemke im Interview zu FLOW3*, 2011. http://www.heise.de/developer/artikel/Robert-Lemke-im-Interview-zu-FLOW3-1362225.html.

[88] Benjamin Eberlei. *Write your own ORM on top of Doctrine2*, 2011. http://www.doctrine-project.org/blog/your-own-orm-doctrine2.

[89] Tzilla Elrad, Mehmet Aksit, Gregor Kiczales, Karl Lieberherr, and Harold Ossher. *Discussing aspects of AOP*, 2001. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.6253&rep=rep1&type=pdf.

[90] TYPO3 Association. *FLOW3 – The Definitive Guide, Part I: Introduction and Fundamentals, Domain-Driven Design*. http://flow3.typo3.org/documentation/guide/parti/conceptsofmodernprogramming.html.

[91] Inc. Domain Language and contributors. *What is Domain-Driven Design?*, 2011. http://www.domaindrivendesign.org/resources/what_is_ddd.

[92] Django Software Foundation. *Django - The Web framework for perfectionists with deadlines*, 2011. https://www.djangoproject.com/.

[93] The World Company. *The World Company Website*, 2011. http://www2.ljworld.com/about/.

[94] Django Software Foundation. *MVC-Aspekte in Django*, 2011. http://bit.ly/xwIfmV.

[95] *Das DRY-Prinzip in Django*, 2011. http://c2.com/cgi/wiki?DontRepeatYourself.

[96] Tim Peters. *The Zen of Python*, 2004. http://www.python.org/dev/peps/pep-0020/.

[97] Django Software Foundation. *Explicit is better than implicit*, 2011. https://docs.djangoproject.com/en/1.3/misc/design-philosophies/#explicit-is-better-than-implicit.

[98] Django Software Foundation. *Explicit is better than implicit (database related)*, 2011. https://docs.djangoproject.com/en/1.3/misc/design-philosophies/#id7.

[99] *Django's Active Record-Ansatz*. https://docs.djangoproject.com/en/1.3/misc/design-philosophies/#include-all-relevant-domain-logic.

[100] Google Inc. *Google Web Toolkit Overview*, 2011. http://code.google.com/webtoolkit/overview.html.

[101] Google Inc. *Google Web Toolkit Productivity for developers, performance for users*, 2011. http://code.google.com/webtoolkit/.

[102] Google Inc. *Google App Engine - Run your web apps on Google's infrastructure*, 2011. http://code.google.com/appengine/.

[103] Eric Clayberg. *Google Plugin for Eclipse (GPE) is Now Open Source*, 2011. http://googlewebtoolkit.blogspot.com/2011/11/google-plugin-for-eclipse-gpe-is-now.html.

[104] mlstate.com. *Opa - the cloud language*, 2011. http://opalang.org/.

[105] mlstate.com. *MLstate - the Company behind Opa*, 2011. https://mlstate.com/.

[106] Robert Lippert. *Auch Opa ist für Cloud-Anwendungen*, 2011. http://www.heise.de/developer/meldung/Auch-Opa-ist-fuer-Cloud-Anwendungen-1288588.html.

[107] Université Paris Diderot. *Université Paris Diderot*, 2011. http://www.univ-paris-diderot.fr/.

[108] IRILL founding members. *IRILL - Research and Innovation on Free Software*, 2011. http://www.irill.org/.

[109] Ocsigen development team. *The Ocsigen project is aimed at proposing clean and safe tools for developing and running client/server Web 2.0 applications*, 2011. http://ocsigen.org/.

[110] INRIA. *The Caml language*, 2011. http://caml.inria.fr/index.en.html.

[111] Ocsigen development team. *Client/server Web applications as a single program*, 2011. http://ocsigen.org/overview/eliomapplications.

[112] Ocsigen development team. *Service based programming*, 2011. http://ocsigen.org/overview/services.

[113] John C. Reynolds. The Discoveries of Continuations. 1993. ftp://ftp.cs.cmu.edu/user/jcr/histcont.pdf.

[114] Ocsigen development team. *Write reliable Web applications - The compiler helps you to remove bugs*, 2011. http://ocsigen.org/overview/typing.

[115] Ocsigen development team. *Js_of_ocaml is a compiler of OCaml bytecode to Javascript.*, 2011. http://ocsigen.org/js_of_ocaml/.

[116] Ocsigen development team. *A complete framework to create and run Web applications*, 2011. http://ocsigen.org/overview/framework.

[117] Guillermo Rauch. *Socket.IO website*, 2011. http://socket.io.

[118] Guillermo Rauch. *Socket.IO Supported transports and Supported browsers*, 2011. http://socket.io/#browser-support.

[119] Statistische Bundesamt, Wiesbaden. *Pressemitteilung Nr.060 vom 14.02.2011*, 2011. http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/Internet/DE/Presse/pm/2011/02/PD11__060__63931,templateId=renderPrint.psml.

[120] Facebook, Inc. *Statistik - Mobile*, 2011. https://www.facebook.com/press/info.php?statistics.

[121] Nicole Leverich. *New research: Global surge in smartphone usage, UK sees biggest jump with 15% increase*, 2012. http://googlemobileads.blogspot.com/2012/01/new-research-global-surge-in-smartphone.html.

[122] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. *Media types*, 2011. http://www.w3.org/TR/CSS2/media.html.

[123] Håkon Wium Lie, Tantek Çelik, Daniel Glazman, and Anne van Kesteren. *Media Queries*, 2010. http://www.w3.org/TR/css3-mediaqueries/.

[124] Sencha Inc. *Sencha Touch - Mobile JavaScript Framework for Developing HTML5 Web Apps*, 2011. http://www.sencha.com/products/touch.

[125] The jQuery Project. *jQuery Mobile: Touch-Optimized Web Framework for Smartphones & Tablets*, 2011. http://jquerymobile.com.

[126] Markus Falk. *Mobile Frameworks Comparison Chart*, 2011. http://www.markus-falk.com/mobile-frameworks-comparison-chart/.

[127] Adobe Systems Inc. *PhoneGap is an HTML5 app platform that allows you to author native applications with web technologies and get access to APIs and app stores*, 2011. http://phonegap.com/.

[128] Beautyindesign.com. *Titanium, PhoneGap, Sencha Touch, jQuery Mobile – Clearing up confusion*, 2010. http://bit.ly/bzahWT.

[129] Adobe Systems Incorporated. *Nitoby Website*, 2011. http://www.nitobi.com/.

[130] Appcelerator Inc. *Titanium Mobile*, 2011. http://www.appcelerator.com/products/titanium-mobile-application-development/.

[131] tagesschau.de, Norddeutscher Rundfunk, Anstalt des öffentlichen Rechts. *Die Tagesschau - auch als Smartphone App*, 2011. http://www.tagesschau.de/app/index.html.

[132] SPIEGEL-Verlag Rudolf Augstein GmbH & Co. KG. *Unterwegs immer auf dem Laufenden*, 2010. http://www.spiegel.de/dienste/0,1518,675525,00.html.

[133] Kay Glahn. *Aus Web wird Native.* iX Developer, Programmieren heute. heise Developer, 2012.

[134] Rhomobile, Inc. *Use your web skills to write NATIVE apps once and build for ALL smartphones with Rhodes 3.0.*, 2011. http://www.appcelerator.com/products/titanium-mobile-application-development/.

[135] Gerhard Völkl. *3D-Darstellungen im Webbrowser*, 2010. http://www.heise.de/ix/artikel/In-die-Tiefe-gehen-1108267.html.

[136] Khronos Group. *WebGL - OpenGL ES 2.0 for the Web*, 2012. http://www.khronos.org/webgl/.

[137] Khronos Group. *Getting a WebGL Implementation*, 2012. https://www.khronos.org/webgl/wiki/Getting_a_WebGL_Implementation.

[138] Microsoft Corporation. *WebGL Considered Harmful*, 2011. http://blogs.technet.com/b/srd/archive/2011/06/16/webgl-considered-harmful.aspx.

[139] Avi Bar-Zeev. *Why Microsoft and Internet Explorer need WebGL (and vice-versa)*, 2011. http://www.realityprime.com/articles/why-microsoft-and-internet-explorer-need-webgl.

[140] Google Inc. *Google Chrome Frame Website*, 2012. http://tools.google.com/chromeframe.

[141] IEWebGL Team. *IEWebGL - WebGL for Internet Explorer*, 2011. http://iewebgl.com/.

[142] Khronos Group. *WebGL Specification*, 2012. http://www.khronos.org/registry/webgl/specs/latest/.

[143] David Herman and Kenneth Russell. *Typed Array Specification*, 2011. http://www.khronos.org/registry/typedarray/specs/latest/.

[144] Stichting Blender Foundation. *Blender is the free open source 3D content creation suite*, 2011. http://www.blender.org/.

[145] Autodesk Inc. *Autodesk Maya*, 2011. http://www.autodesk.de/adsk/servlet/pc/index?siteID=403786&id=14657512.

[146] Joe Hewitt. *(canvas) create xul canvas tag for custom painting control. Bug Report*, 2001. https://bugzilla.mozilla.org/show_bug.cgi?id=102285.

[147] Ian Hickson. *The canvas element*, 2012. http://dev.w3.org/html5/spec/Overview.html#the-canvas-element.

[148] Khronos Group. *OpenGL Shading Language*, 2012. http://www.opengl.org/documentation/glsl/.

[149] Khronos Group. *WebGL Frameworks*, 2011. http://www.khronos.org/webgl/wiki/User_Contributions#Frameworks.

[150] Colin MacKenzie IV. *Erster Commit von JaxGL bei Github*, 2011. https://github.com/sinisterchipmunk/jax/commit/028c4c1ffd1b2267b7faeb36fe375d608dac1c7e.

[151] Colin MacKenzie IV. *JaxGL Website*, 2011. http://jaxgl.com/.

[152] Colin MacKenzie IV. *Jax on Rails*, 2011. http://blog.jaxgl.com/2011/09/jax-on-rails/.

[153] Colin MacKenzie IV. *Getting Started with Jax*, 2011. http://guides.jaxgl.com/getting_started.html.

[154] Nicolas Garcia Belmonte. *PhiloGL Website*, 2011. http://www.senchalabs.org/philogl/.

[155] Paul Brunt. *GLGE: WebGL for the lazy*, 2010. http://www.glge.org/.

[156] S.M.A.R.T. Consult Ltd. & Co. KG. *IDC-Studie: Cloud Computing in Deutschland 2011*, 2011. http://www.idc.de/press/presse_mc_cloud2011.jsp.

[157] Google Inc. *Chromebook Website*, 2011. http://www.google.com/intl/en/chromebook/.

[158] Google Inc. *Chromium OS*, 2012. http://www.chromium.org/chromium-os.

[159] Google Inc. *Chrome Web Store*, 2012. https://chrome.google.com/webstore.

[160] Mozilla Foundation. *Boot to Gecko*, 2012. https://wiki.mozilla.org/B2G.

[161] Steve Lohr. *Why Mozilla Is Entering the Smartphone War*, 2012. http://bits.blogs.nytimes.com/2012/02/23/why-mozilla-is-entering-the-smartphone-war/.

[162] Mozilla Foundation. *Coming Soon: Mozilla Marketplace - Your apps on all your devices*, 2012. https://www.mozilla.org/en-US/apps/.

[163] Apache Software Foundation. *Deltacloud - Many clouds. One API. No Problem*, 2011. http://deltacloud.apache.org/.

[164] Apache Software Foundation. *Driver functionality and Credentials*, 2011. http://deltacloud.apache.org/drivers.html#providers.

[165] Amazon Web Services LLC. *AWS Products & Solutions*, 2012. http://aws.amazon.com/products/.

[166] Amazon Web Services LLC. *AWS Blog*, 2009. http://aws.typepad.com/aws/2009/02/aws-links-wednesday-february-25-2009.html.

Erklärung

Hiermit versichere ich, dass ich die vorliegende schriftliche Bachelorarbeit selbstständig verfasst und keine anderen als die von mir angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinne nach entnommen sind, wurden in jedem Fall unter Angabe der Quellen (einschließlich des World Wide Web und anderer elektronischer Text- und Datensammlungen) kenntlich gemacht. Mir ist bewusst, dass jedes Zuwiderhandeln als Täuschungsversuch zu gelten hat, der die Anerkennung der Bachelorarbeit ausschließt und weitere angemessene Sanktionen zur Folge haben kann.

Siegen, den 16.03.2012