FH Technikum Wien

BACHELORARBEIT

Webseiten-Anpassungen für ältere Browser

ausgeführt von Christoph Kamon

Begutachter: DI Dr. Gerd Holweg

26. 5. 2010

Ausgeführt an der FH Technikum Wien

Studiengang Bachelor Informatik / Computer Science

Eidesstattliche Erklärung

„Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.“

_____ _____

Ort, Datum Unterschrift

Kurzfassung

Bei der Entwicklung von Webanwendungen aller Art stellen ältere Browser immer wieder eine Hürde dar. Mangelnde Unterstützung oder fehlerhafte Umsetzungen von Technologien führen in vielen Fällen zu unbrauchbaren Resultaten. Je älter ein zu unterstützender Browser ist, desto mehr Probleme sind dabei zu erwarten. Auch wenn in diesem Zusammenhang meist nur Internet Explorer 6 genannt wird, ist dieser nicht der einzige betroffene Browser.

Diese Bachelorarbeit befasst sich mit den gängigen Webtechnologien HTML, CSS und JavaScript, sowie dem Grafikdateiformat PNG. Dabei wird jeweils deren Entstehung umrissen und zu erwartende Probleme in gängigen Browsern aufgeführt. Um Probleme zu vermeiden, wird gezielt auf potentielle Problemstellen eingegangen und auch, wie sich diese vermeiden lassen.

Für Fälle, in denen sich Probleme nicht im Vorfeld umgehen lassen, werden mögliche Lösungsansätze für genannte Bereiche erläutert und bewertet. Die Beschreibung einer praktischen Umsetzung, einschließlich nachfolgender Evaluierung, ergänzt die gewonnenen theoretischen Erkenntnisse.

Schlagwörter: HTML, CSS, JavaScript, Browserkompatibilität, Webentwicklung

Abstract

When developing web applications of almost any type, outdated browsers regularly lead to problems. In many cases a lack of support or even buggy implementations of technologies lead to unusable results. The older the browser that is to be supported, the more problems one can expect. Though Internet Explorer 6 is the browser most commonly associated with these problems, there are other browsers that also create problems for web developers.

This bachelor's thesis explores the commonly used web technologies HTML, CSS and JavaScript in addition to the graphics file format PNG. For each of these, an evolutionary history is provided and the expected problems with common web browsers are described. To help minimize problems, this thesis specifically explores potential issues and provides information on how to avoid them.

For cases where it is not immediately possible to avoid these critical areas, approaches for finding solutions are explained and rated. The theoretical knowledge provided in this thesis is then supplemented by a description of a practical implementation, followed by its evaluation.

Keywords: HTML, CSS, JavaScript, browser compatibility, web development

Danksagung

An dieser Stelle möchte ich mich insbesondere bei meinem Betreuer DI Dr. Gerd Holweg bedanken, der mir während der Erstellung dieser Arbeit mit guten Ideen und Hilfestellungen zur Seite stand. Die gute Kommunikationsbasis mit regelmäßigem Feedback wurde von mir als sehr angenehm und motivierend empfunden.

Dank ergeht auch an Paul Kingsbury, welcher wieder so nett war, grammatikalische Fehler im Abstract aufzuspüren und zu korrigieren. Außerdem danke ich meinem Kollegen Philipp Büchler, welcher mir immer wieder Feedback zu meiner Arbeit gegeben und Fehler aller Art aufgedeckt hat.

Anerkennung meinerseits gebührt auch Alexander Germ, welcher die abgabefertige Fassung dieser Arbeit gegengelesen hat und einige suboptimale Formulierungen sowie Schreibfehler ausfindig machen konnte.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Nach langer Vorherrschaft des Internet Explorers kam vor einigen Jahren wieder Bewegung in den Browsermarkt. Jahre im Abseits stehende Browser konnten sich behaupten und stellen nun häufig verbreitete Alternativen dar. Dieser Wandel machte die Notwendigkeit von einheitlichen Webstandards deutlich und führte dazu, dass die Unterstützung für ebenjene immer besser implementiert wurde. Eine standardkonforme Webseite wird in allen aktuellen Browsern größtenteils konsistent dargestellt.

Wesentlich umständlicher gestaltet sich die Situation mit älteren Browsern, die noch immer eine nennenswerte Verbreitung aufweisen. Insbesondere – aber nicht ausschließlich – betrifft das den Internet Explorer 6, welcher zum Lieferumfang von Windows XP gehört. Abgesehen vom Alter dieses Browsers, welches zunehmend zum Problem wird, rendert er Seiten in vielen Fällen nicht korrekt. Um „zerstörte“ Layouts und Formatierungen wieder herzustellen, ist oft viel zusätzliche Arbeit notwendig. Damit Benutzern dieser älteren Browser der Zugang zu einer Webseite nicht verwehrt bleibt, sind solche Anpassungen aber unumgänglich.

Genau so ist auch die Situation bei einem aktuellen Projekt des Autors. Eine Webplattform, die für aktuelle Browser entwickelt wurde beziehungsweise wird, hat teilweise mit erhebliche Darstellungsproblemen zu kämpfen, wenn sie in einem älteren Browser aufgerufen wird. Da solche Besucher aber nicht ausgeschlossen werden dürfen, muss die Plattform entsprechend kompatibel gemacht werden. Durch ausreichendes theoretisches Wissens soll der praktische Aufwand dabei möglichst gering ausfallen.

1.2 Ziele & Nichtziele

Im Rahmen dieser Bachelorarbeit gilt es einige Fragestellungen zu beantworten. Dabei ist zunächst von Interesse, wie gut die einzelnen Browser Cascading Style Sheets (CSS) und HyperText Markup Language (HTML) unterstützen, beziehungsweise welche Probleme in diesem Zusammenhang zu erwarten sind. Auch andere potentielle Problemstellen, wie JavaScript und Grafikdateiformate, werden behandelt.

Desweiteren wird evaluiert, welche Möglichkeiten bestehen, die fehlerhaften Darstellungen in älteren Browsern zu umgehen beziehungsweise zu korrigieren. Dabei werden auch die Nachteile der einzelnen Lösungsansätze erläutert. Abschließend wird das erarbeitete theoretische Wissen auf die bereits angesprochene Webplattform angewandt. Angestrebt wird eine, auch in älteren Browsern, uneingeschränkte Verwendbarkeit. Diese beinhaltet ein korrektes Layout, sichergestellte Lesbarkeit der Inhalte sowie funktionierende JavaScripts.

Erklärtes Nichtziel ist es, eine pixelgenaue Übereinstimmung aller Browser zu erreichen. Dies stellt selbst unter modernen Browsern oftmals eine Hürde dar und würde den Arbeitsaufwand für ältere Browsern drastisch erhöhen. So gut wie nicht mehr verbreitete Browser – beispielsweise Internet Explorer 5.5 oder Firefox 1.0 – sowie Features von CSS3 werden nicht berücksichtigt.

Die Webplattform setzt zum gegenwärtigen Zeitpunkt auf keine Technologien wie Ajax (Asynchronous JavaScript and XML). Potentielle Probleme in diesem Bereich werden daher nicht näher behandelt.

1.3 Methodik

Insgesamt drei verschiedene Lösungsansätze werden verwendet, um die Fragestellungen dieser Arbeit zu beantworten. Den ersten Teil bildet dabei eine Literatur- und Internetrecherche, mit deren Hilfe Grundlagen und Problemstellen der einzelnen Technologien im Webbereich beschrieben werden. Dabei wird auch versucht, potentiell problematische Browser heraus zu filtern und ihre Schwächen zu identifizieren.

Ein Vergleich der unterschiedlichen Methoden, um browserspezifische Korrekturen durchzuführen, stellt den zweiten Ansatz dar. Vorteile, Nachteile und mögliche Nebenwirkungen sowie gegebenenfalls technische Voraussetzungen fließen hier ein.

Die praktisch durchgeführte Behebung von Darstellungsproblemen stellt den dritten Ansatz dar. Dabei wird eine bereits existierende Webseite analysiert und derart angepasst, dass sie in älteren Browsern uneingeschränkt verwendbar und optisch ansprechend ist. Eine genaue Auswahl der „Zielbrowser“ wird im Vorfeld durchgeführt.

2 Selektion der Browser

Da die Bezeichnung „ältere Browser“ zu viel Interpretationsspielraum lässt, ist es wichtig, zunächst zu definieren, welche Vertreter hier von weiterem Interesse sind. Je nach Betrachtungszeitraum und Auswahl der Betriebssysteme, beziehungsweise Plattformen, ist die Anzahl an Browsern unüberschaubar groß.

Wenngleich unzählige mobile Endgeräte mit ihren jeweiligen Plattformen für den Internetzugriff existieren, werden lediglich Browser für herkömmliche Desktop- und Notebook-Computer herangezogen. Dabei liegt das Augenmerk – einerseits, um die Auswahl möglichst repräsentativ zu gestalten, und andererseits, um den Rahmen der Bachelorarbeit nicht zu sprengen – auf folgenden, weit verbreiteten Browsern:

- Internet Explorer 6.0.2900.5512.xpsp_sp3_gdr.091208-2036
- Internet Explorer 7.0.5730.13
- Internet Explorer 8.0.7600.16385
- Firefox 2.0.0.20
- Firefox 3.6.3
- Opera 9.64 Build 10487
- Opera 10.51 Build 3315
- Safari 3.2.2 (525.28.1)
- Safari 4.0.5 (531.22.7)

Die Auswahl ist insofern zu begründen, dass von den vier Browserfamilien jeweils die aktuellste Version gewählt wurde. Da immer wieder Benutzer eine ältere Browserversion im Einsatz haben – beabsichtigt, mangels Möglichkeit zum Upgrade oder aufgrund eines anderen Umstandes – wurde auch die jeweilige Vorversion hinzugefügt. Alle Browser wurden in der

zum Zeitpunkt der Erstellung dieser Arbeit (13. 4. 2010) neuesten verfügbaren Version betrachtet.

Obwohl Internet Explorer 6 bereits deutlich in die Jahre gekommen ist, kann er aufgrund seiner weiterhin großen Verbreitung nicht ignoriert werden. In Firmen und größeren Netzwerken wird, aufgrund zu erwartender Inkompatibilitäten bei älteren Webanwendungen, mit einem Upgrade auf eine neuere Version oftmals gezögert. Auch stehen in den seltensten Fällen zusätzliche Webbrowser zur Verfügung, wodurch die Benutzer weiterhin auf Internet Explorer 6 angewiesen sind.

Ein ebenfalls weit verbreiteter Browser – Google Chrome – befindet sich nicht in der Auflistung. Das hat den Grund, dass Chrome, genauso wie Safari, auf WebKit als Rendering Engine setzt (Google, 2010). Da die somit auftretenden Unterschiede bei der Darstellung von Webseiten minimal bis nicht vorhanden sind (je nach Version der Engine), wurde auf die gesonderte Behandlung von Google Chrome verzichtet.

Die Basistechnologien, auf denen im Allgemeinen Browser aufbauen, umfassen HTML und JavaScript als Unterbau sowie CSS und Grafikdateien als Möglichkeiten zur optisch attraktiven Repräsentation. In den folgenden Kapiteln wird auf diese Technologien eingegangen.

3 HTML & CSS

HTML und CSS bilden die Kerntechnologien, auf denen das WWW (World Wide Web) basiert. Obwohl HTML es durchaus erlaubt, direkt Formatierungen anzubringen (Ducket, 2008, S. 3), ist die Hauptaufgabe – wie bereits durch den Namen Hypertext Markup Language angedeutet – die Auszeichnung von Dokumentabschnitten. Dies bedeutet, dass nur festgelegt wird, wo beispielsweise ein Absatz anfängt und aufhört. Nicht festgelegt wird hingegen, wie dieser Absatz für den Benutzer dargestellt werden soll.

Diese Aufgabe wird von CSS (Cascading Style Sheets) übernommen, mit dessen Hilfe die Repräsentation der HTML-Inhalte geregelt wird. Abgesehen von deutlich erweiterten Möglichkeiten (Ducket, 2008, S. 221f), was die Gestaltung anbelangt, gibt es noch einige weitere Vorteile. Exemplarisch lassen sich hier eine gesteigerte Übersichtlichkeit der HTML-Dateien sowie wiederverwendbare Style Sheets nennen.

3.1 Entstehung von HTML

HTML hat seinen Ursprung im Jahr 1990, als sich Tim Berners-Lee, ein Mitarbeiter von CERN (European Organization for Nuclear Research), vor dem Problem sah, dass sich ein plattformübergreifender Informationsaustausch als kompliziert erwies. Mangels kompatibler Dateiformate wurden meist Ausdrucke weiter gegeben – mit all den damit verbundenen Nachteilen. Durch Kombination der Technologien SGML (Standard Generalized Markup Language) und Hypertext, die beide bereits in den 1960er Jahren geschaffen wurden, entwickelte Tim Berners-Lee die Urfassung von HTML. (Huddleston, 2008, S. 2).

Der große Durchbruch von HTML trat allerdings erst wesentlich später, nämlich Mitte der 1990er Jahre, ein. Ermöglicht wurde dieser durch die Entstehung von fortschrittlichen

Webbrowsern und dem Aufkommen von Webseiten, die für eine große Zielgruppe interessante Inhalte anboten.

Seit seiner Entstehung hat HTML einige Versionssprünge durchlaufen und wurde regelmäßig erweitert. Die erste offizielle Version, die von der Internet Engineering Task Force ab 1993 als „working draft“ geführt wurde, war HTML 2.0. Wenige Jahre danach wurde mit HTML 3.2 die erste populäre Version der Auszeichnungssprache veröffentlicht. Problematisch dabei war allerdings, dass sowohl Netscape als auch Microsoft browserspezifische Zusätze unterstützten. Dies erschwerte es Webdesignern, eine konsistente Darstellung in beiden Browsern zu erreichen. Außerdem führte die Nicht-Trennung von Auszeichnung und Formatierung zu unleserlichem Code. Beide Gründe führten zur Entwicklung von HTML 4 (und in weiterer Folge HTML 4.01), welches zur Formatierung auf Cascading Style Sheets setzt. (Huddleston, 2008, S. 2)

Um abwärtskompatibel zu bleiben, führte HTML 4 insgesamt drei Varianten an Dokumenten ein. Die moderne Form nennt sich „Strict“ und bedeutet, dass sämtliche Formatierungen in CSS-Definitionen ausgelagert sein müssen. Die hauptsächlich für die Übergangszeit geschaffene Variante „Transitional“ ermöglicht die kombinierte Verwendung von CSS und direkten HTML-Formatierungen. Die dritte Variation heißt „Frameset“ und ermöglicht ein Seitenlayout mittels Frames. (Huddleston, 2008, S. 3)

Aktuell steht HTML unter der Verwaltung des W3C (World Wide Web Consortium), welches sich um Weiterentwicklung und Standardisierung kümmert. Neben der Normung zahlreicher anderer Auszeichnungssprachen, die zu einem guten Teil auf XML (Extensible Markup Language) basieren, stellt das W3C auch Validationswerkzeuge zur Verfügung. Mithilfe dieser Werkzeuge lassen sich selbst erstellte Dokumente leicht auf Gültigkeit beziehungsweise Fehlerfreiheit prüfen.

Der nächste Schritt in der Entwicklung heißt HTML5, welches planmäßig im Jahr 2011 vollständig spezifiziert sein wird. Dabei lieg der Fokus auf Webapplikationen, die in vielen Fällen statische Textinhalte ersetzen und den Benutzer interaktiv in die Webseite einbinden. Einer der populären neuen HTML-Tags, die mit HTML5 eingeführt werden, ist `<video>`, welcher das direkte Einbinden von Videodateien ermöglicht. Ein Videoplayer auf Flash-Basis oder vergleichbare Lösungen sind damit nicht mehr erforderlich. Auch direktes Ansprechen von APIs (Application Programming Interface) wird mit HTML5 möglich gemacht. (Zeldman & Marcotte, 2010, S. 136f)

3.2 Entstehung von CSS

Allgemein ist CSS als „moderne Möglichkeit des Webdesigns“ bekannt. Weniger bekannt ist, dass diese Technologie bei weitem nicht so neu ist, wie oft angenommen wird. Bereits Ende des Jahres 1996 (Powers, 2009, S. 3) wurde eine erste Spezifikation von Cascading Style Sheets durch das W3C veröffentlicht. Damit wurde die Basis für zwei essentielle Dinge geschaffen. Einerseits wurden die Möglichkeiten zur optischen Gestaltung von Webseiten deutlich erweitert. Andererseits erlaubt die Nutzung von CSS eine strikte Trennung von Inhalt und dessen Repräsentation.

Um der Forderung, die zur Entstehung von CSS führte, weiter gerecht zu werden, wurde die Technologie durch das W3C intensiv weiter entwickelt und bereits zwei Jahre später erschien

dessen zweite Inkarnation, CSS2. Allerdings führte diese verhältnismäßig schnelle Entwicklung zu mehr Problemen, als sie zu lösen vermochte. Der marktführende Browser im Jahre 1997, Netscape 4, sowie dessen Konkurrent Internet Explorer 4 boten lediglich rudimentäre Unterstützung für CSS (Powers, 2009, S. 4).

In den darauf folgenden Jahren unterlag der Browsermarkt einem starken Wandel. Die Vertreter der Netscape-Familie wurden zunehmen durch Microsofts Browser verdrängt. Zur Jahrtausendwende hatte der Internet Explorer eine deutliche Marktdominanz aufgebaut und Netscape verdrängt. Dennoch waren – damals wie heute – in vielen Bereichen noch veraltete Browser im Einsatz, die den Durchbruch von CSS blockierten.

Internet Explorer 6 war der erste Browser von Microsoft, der eine verhältnismäßig gute Unterstützung für Cascading Style Sheets bot. Verhältnismäßig gut deshalb, weil viele Features unterstützt wurden, aber diese teilweise schlichtweg falsch umgesetzt waren. Obwohl dieser Browser nach wie vor verbreitet eingesetzt wird, wurde er in den Jahren seit seiner Veröffentlichung diesbezüglich nicht korrigiert. Sicherheitslücken wurden wohl geschlossen, aber die Rendering-Engine wurde nicht um Bugs bereinigt. Aufgrund dieses Umstandes strebten die Hersteller anderer Browser nach einer besseren CSS-Unterstützung (Powers, 2009, S. 5).

Diese Strategie war durchaus von Erfolg gekrönt. Mozilla, Opera und Konsorten gelang es so, kontinuierlich Marktanteile von Microsoft zurück zu erobern. Mit der Veröffentlichung von Internet Explorer 7 wurde die CSS-Unterstützung schon deutlich verbessert. Das darauf folgende Release des Microsoft-Browsers, Internet Explorer 8, konnte sogar schon mit einer vollständigen Unterstützung von CSS2 aufwarten. Durch diesen Schritt konnte Internet Explorer den Vorsprung anderer Browser in diesem Bereich größtenteils aufholen. Trotz eventuell auftretender, minimaler Unterschiede in der Darstellung lässt sich somit – ausschließlich Webseitenbesucher mit modernen Browsern vorausgesetzt – ein und dasselbe Style Sheet für alle gängigen Browser verwenden (Powers, 2009, S. 6).

Die aktuelle Version von CSS2 ist „Cascading Style Sheets Level 2 Revision 1 (CSS 2.1)“ und wurde am im September 2009 veröffentlicht. Diese, auf CSS1 aufbauende Version behebt einige Fehler, die in der ursprünglichen Veröffentlichung von CSS2 zu finden waren (World Wide Web Consortium, 2009).

Der evolutionäre Nachfolger, CSS3 genannt, befindet sich aktuell in der Entwicklung. Aufbauend auf CSS1 und CSS2 werden hier neue Funktionen hinzugefügt und in Modulen gruppiert. Der aktuelle Entwicklungsstatus laut Webseite (World Wide Web Consortium, 2010) zeigt, dass einige Module bereits sehr weit fortgeschritten sind, während bei anderen noch viel ausständig ist. Obwohl manche Browserhersteller schon vor mehreren Jahren mit der Implementierung von CSS3 begonnen haben (Storey, 2007), ist diese Technologie nach wie vor mit Bedacht einzusetzen.

Die Gründe hierfür sind einerseits ein inhomogener Implementierungsstand der Browserhersteller (da ja der Standard CSS3 noch in Entwicklung ist) sowie die verhältnismäßig große Verbreitung älterer Browser, auf die ständig Rücksicht genommen werden muss. Aus diesen Gründen werden vermutlich noch einige Jahre vergehen, bis CSS3 allgemein und uneingeschränkt einsetzbar ist.

3.3 XHTML versus HTML

Neben dem bereits angesprochenen HTML wird im Internet immer wieder XHTML (Extensible Hypertext Markup Language), das ebenfalls zur Beschreibung von Webseiten genutzt werden kann, verwendet. Welche der beiden Auszeichnungssprachen die zu Bevorzugende ist, beziehungsweise welche Vor- und Nachteile zu erwarten sind, wird hier kurz erläutert.

XHTML lässt sich als XML-basierte Sprache beschreiben, die in Aussehen und Funktionalität HTML stark ähnelt. Der Unterschied liegt im Detail. Beispielsweise ist es bei XHTML erforderlich, dass jeder öffnende Tag auch einen schließenden Tag besitzt und die Verschachtelung dieser korrekt aufgebaut ist (Wohlgeformtheit). Bei HTML hingegen gibt es Tags, die nicht geschlossen werden genauso wie solche, die geschlossen werden müssen. Bei manchen ist das Schließen sogar optional. Dieser Umstand kann bei der Verarbeitung durch Browser oder auch andere Programme zu Problemen führen (Zeldman & Marcotte, 2010, S. 104f).

Eine weitere Eigenheit von XHTML ist, dass Attribute unter Anführungszeichen gesetzt werden müssen. Bei HTML ist das in vielen Fällen nicht erforderlich. Attribute, die keinerlei Wertzuweisung haben sind in XHTML nicht erlaubt – in Fällen dieser Art ist dem Attribut sein eigener Name als Wert zuzuweisen. Wird beispielsweise einem Radiobutton in HTML-Auszeichnungssprache das Attribut `checked` gesetzt, so lautet diese Zuweisung korrekter Weise in XHTML `checked="checked"`. Diese strengere Syntax von XHTML ist in vielen Details merkbar (Huddleston, 2008, S. 4).

Wie auch bei HTML, existieren bei XHTML ebenfalls drei Variationen des Standards. Diese lauten „XHTML Transitional“, „XHTML Strict“ und „XHTML Frameset“ und entsprechen in Funktionalität beziehungsweise Bedeutung ihren Pendants in HTML.

Als Vorteil von HTML lässt sich zunächst nennen, dass diese Sprache von allen Browsern, inklusive des zugehörigen MIME (Multipurpose Internet Mail Extensions) Types, unterstützt wird. Gerade bei Webseiten, die durch Skripte generiert werden, schleichen sich in der Struktur gerne Fehler ein. Auf diese Fehler reagiert HTML, verglichen mit XHTML, weniger sensibel. Aufgrund der nicht zwingend erforderlichen schließenden Tags kann außerdem Zeit bei der Seitenerstellung sowie Bandbreite bei der Übertragung gespart werden (Zeldman & Marcotte, 2010, S. 109).

XHTML bietet als großen Vorteil, dass der Autor zum sorgfältigeren Arbeiten gezwungen wird. Der resultierende Code ist somit für andere leichter lesbar. Ob ein Dokument valide ist, lässt sich leicht mit einem beliebigen XML-Validator und dem verlinkten Schema beziehungsweise der angegebenen DTD (Document Type Definition) feststellen.

Dadurch, dass XHTML eine Teilmenge von XML ist, lässt es sich von sehr vielen Programmen und Programmierschnittstellen verhältnismäßig einfach verarbeiten. Damit stehen auch Möglichkeiten offen, solche Dokumente maschinell in andere Formate zu übersetzen. Falls in Zukunft ein anderes XML-basiertes Dokumentformat zum Standard wird, lässt sich so die Konvertierung schnell und automatisiert durchführen. Zusätzlich ist mit XHTML die Struktur vollständig von Design getrennt, was in einem „schlankeren“ Code und erhöhter Wiederverwendbarkeit resultiert (Huddleston, 2008, S. 5).

Ob jetzt allgemein auf XHTML oder HTML, welches bald in seiner fünften Revision verfügbar sein wird, gesetzt werden soll, hängt von persönlichen Präferenzen und technischen Anforderungen ab. XHTML bietet „saubereren Code“ und gute Möglichkeiten zur maschinellen Verarbeitung während HTML5 in Richtung „Rich Web Applications“ steuert, die dynamisch mit dem Benutzer interagieren. Die Auswahl ist oft insofern eingeschränkt, dass eine bestehende Webplattform erweitert werden soll. Wurde bisher in XHTML geschrieben, wird dies auch für Anpassungen und Erweiterungen der sinnvollste Weg sein. Sind Strukturen auf HTML-Basis vorhanden, wird meist auf Basis dieser Technologie weiter entwickelt.

Unabhängig von der Wahl der Auszeichnungssprache ist es wichtig, zu wissen, wie Browser die Dokumente interpretieren und wie sie beim Auftreten von Fehlern respektive ungültigen DOCTYPE-Angaben reagieren.

3.4 Rendering-Modi der Browser

Wenn eine Webseite, die beispielsweise im Internet Explorer tadellos aussieht und problemlos verwendbar ist, in anderen Browsern vollkommen anders dargestellt wird, ist ein Betrachten der Rendering-Modi interessant. Obwohl ältere Versionen des Microsoft-Browsers bekannter weise einige Bugs haben, ist das Problem der abweichenden Darstellung oft beim so genannten „Quirks Mode“ zu suchen.

Moderne Browser rendern üblicherweise in einem (möglichst) standardkonformen Modus, der die Vorgaben des W3C bestmöglich einhält. Dies führt zu einer größtenteils konsistenten Darstellung über unterschiedliche Browser. Aus Kompatibilitätsgründen ist in Browsern meist ein Kompatibilitätsmodus (Quirks Mode) enthalten, der das – eigentlich falsche – Verhalten von Netscape 4 sowie Internet Explorer 4 und 5 nachbildet. Dadurch ist es möglich, dass auch alte Webseiten in modernen Browsern korrekt angezeigt werden können (Andrew, 2009, S. 233).

Zur Unterscheidung, ob eine Webseite standardkonform oder im Quirks Mode gerendert werden soll, wird von den meisten Browsern die so genannte DOCTYPE-Angabe am Anfang des Dokuments betrachtet. Sofern diese bekannt gibt, dass es sich um ein valides HTML- oder XHTML-Dokument handelt, wählt der Browser den standardkonformen Modus. Für den Fall, dass die DOCTYPE-Zeile fehlerhaft, unvollständig oder nicht vorhanden ist, geht der Browser davon aus, dass es sich um ein altes Dokument handelt, das im Quirks Mode behandelt werden soll (Schmitt, CSS Cookbook: Third Edition, 2010, S. 8f).

Für Webseiten, die neu entwickelt, angepasst oder erweitert werden, ist es im Allgemeinen ratsam, auf einen korrekten DOCTYPE zu achten und so alle Browser in ihren standardkonformen Modus zu versetzen. Wichtig dabei ist, dass die DOCTYPE-Definition als erste Zeile im Quelltext der Seite steht. Andernfalls wird sie beispielsweise von Internet Explorer 6 ignoriert und es kommt wieder, trotz eventuell gültigem DOCTYPE, der nicht gewollte Quirks Mode zum Einsatz. Unerwartetes Verhalten, falsch berechnete Größenangaben – beispielsweise im Zusammenhang mit dem CSS Box Model – und vieles mehr sind die Folgen. (Andrew, 2009, S. 233)

3.5 Umfang der Unterstützung

Das Vorhandensein von Webstandards – unter anderem betreffend HTML und CSS – macht es sinnvoll und möglich, die Einhaltung dieser überprüfbar zu machen. Der pragmatische Ansatz hier ist es, eine Referenz-Webseite zu schreiben, die im Sinne der W3C-Standards vollkommen valide ist. Anschließend wird das Ergebnis in unterschiedlichen Browsern betrachtet und verglichen. Nachteilig ist dabei, dass die meisten Webseiten nur eine kleine Auswahl der Möglichkeiten ausschöpfen, die CSS zu bieten hat.

Genau auf diesen Ansatz greifen die Browsertests Acid, Acid2 und Acid3 zurück. Diese wurden vom Web Standards Project veröffentlicht[1], um die Standardkonformität von Browsern zu bewerten und zu vergleichen. Das Ziel davon war und ist es, Browserherstellern einen Anreiz zu bieten, gut bei den Tests abzuschneiden und so einen Pluspunkt gegenüber der Konkurrenz zu haben.

Nach einigen Jahren Acid-Geschichte lässt sich festhalten, dass der Plan durchaus aufgegangen ist und alle namhaften Browserhersteller Acid als Referenz sehen. Dabei testet der älteste Acid-Test insbesondere das Box-Modell von CSS. Bei dessen Nachfolger Acid2 liegt der Fokus auf HTML, CSS, PNG-Grafiken sowie Data URIs (Uniform Resource Identifier). Acid3 stellt den aktuellsten Test des Web Standards Project dar und überprüft den Browser – unter anderem – auf Fähigkeiten bezüglich DOM (Document Object Model), CSS 3 und JavaScript.

Auf den für HTML und CSS besonders interessanten Acid2-Test wird im folgenden Kapitel theoretisch eingegangen und anschließend der Test in den gewählten Browsern durchgeführt.

3.5.1 Acid2 im Überblick

Aufgabe des Acid2-Tests ist es, Browser auf die Einhaltung von Webstandards zu testen. Das Ergebnis wird dabei rein visuell beziehungsweise subjektiv durch den Benutzer mit dem Referenz-Rendering verglichen. Eine automatisierte Überprüfung des Testergebnisses oder Ausgabe eines Punktestandes ist, im Gegensatz zum Nachfolger Acid3, nicht möglich.

Getestet werden Features von HTML und CSS, Unterstützung für das PNG (Portable Network Graphics) Grafikformat sowie Data URLs. Diese Auswahl ist insofern zu begründen, dass diese Technologien die Basis des World Wide Web bilden und auch in Zukunft weiterhin von Bedeutung sein werden. Ausgegangen wird von standardgemäßen Browsereinstellungen, 100% Zoomstufe sowie einer Bildschirm-Pixeldichte von 96 ppi (points per inch).

Selbstverständlich ist es nicht möglich, in einem einzigen Test die gesamte Spezifikation von CSS2 und HTML abzudecken. Acid2 versucht, eine möglichst sinnvolle Auswahl an häufig verwendeten Elementen zu treffen. Unter anderem getestet werden transparente PNGs, Positionierung (`relative`, `absolute`, `fixed`), das CSS Box Model, Margins und vieles mehr (The Web Standards Project, 2006).

Die Webseite, die während des Acid2-Tests gerendert wird, zeigt einen gelben Smiley mit schwarzer Umrandung. Der grundlegende Aufbau entspricht einer Matrix mit 14x14 Feldern, wobei jede Zeile innerhalb dieser bestimmte Features testet. In einer auf der Projektwebseite

[1] http://www.acidtests.org/
[2] http://www.webstandards.org/action/acid2/guide/

zur Verfügung stehenden Tabelle[2] lässt sich die genaue Bedeutung jeder Matrizenzeile ablesen. Falsch positionierte, fehlende oder rot eingefärbte Elemente sind als Darstellungsfehler zu werten.

3.5.2 Acid2: Resultate & Auswertung

Um herauszufinden, welche Browser sich gut an die vorgegebenen Webstandards halten – und in weiterer Folge, bei welchen Browsern Probleme zu erwarten sind – wurden alle neun ausgewählten Browser diesem Test unterzogen. Als Testplattform diente ein zum Zeitpunkt der Durchführung (13. 4. 2010) vollständig gepatchtes Windows XP.

Für die Durchführung des Acid2-Tests wurde auf die Testseite[3] navigiert und der Link zum Starten betätigt. Anschließend wurde jeweils ein Screenshot angefertigt. Zur besseren Vergleichbarkeit wurden diese pixelgenau übereinander gelegt. Das heißt, dass sich der Smiley, unabhängig von Fenstergröße des Browsers und unterschiedlich hohen Symbolleisten, immer an derselben Position des Bildes befindet.

Aufgrund von teilweise unterschiedlichen Kantenglättungsverfahren, die bei der Schriftdarstellung zum Einsatz kommen, sieht der Schriftzug auf den folgenden Screenshots – je nach Browser – geringfügig anders aus. Auch minimale Abweichungen des Abstandes zwischen Schriftzug und Smiley werden nicht als Fehler betrachtet, da auch hier Glättung und etwaige Rundungsfehler beim Rasterisieren von Vektorschriftarten ihren Einfluss haben.

Internet Explorer

Als erste der vier zu untersuchenden Browserfamilien wird Microsofts Internet Explorer dem Test unterzogen. Nachfolgend die Rendering-Ergebnisse in den drei getesteten Versionen.

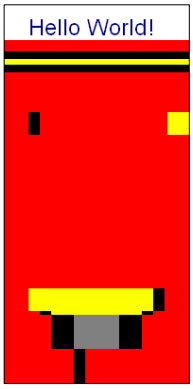

Abbildung 1: Acid2 im Internet Explorer 6.0

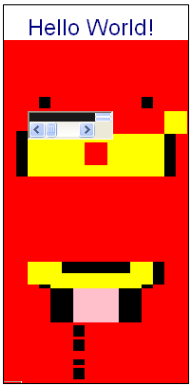

Abbildung 2: Acid2 im Internet Explorer 7.0

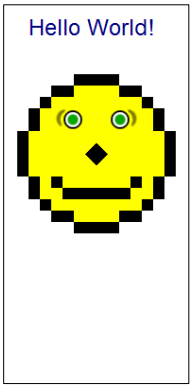

Abbildung 3: Acid2 im Internet Explorer 8.0

Während das Ergebnis von Internet Explorer 8 (Abbildung 3) dem Referenz-Rendering entspricht, sind die Resultate von Internet Explorer 6 (Abbildung 1) und Internet Explorer 7 (Abbildung 2) weit davon entfernt. Dennoch lässt sich hier differenzieren. Abgesehen vom Schriftzug „Hello World!“ ist kein einziges Element im Internet Explorer 6 korrekt positioniert.

---

[2] http://www.webstandards.org/action/acid2/guide/
[3] http://www.webstandards.org/files/acid2/test.html

Hingegen beim Internet Explorer 7 ist zu erkennen, dass zumindest ein paar der Blöcke, die den Smiley formen, korrekt positioniert und eingefärbt sind. Somit lässt sich festhalten, dass die Konformität des Microsoft-Browsers sukzessive verbessert wurde. Größere, die Darstellung betreffende, Probleme sind aufgrund dieser Ergebnisse nur bei Version 6 und Version 7 von Internet Explorer zu erwarten.

Firefox

Bei der zweiten zu testenden Browserfamilie handelt es sich um Mozilla Firefox. Die Ergebnisse der zwei getesteten Versionen sehen folgendermaßen aus:

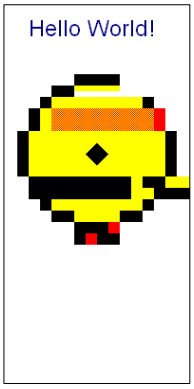

Abbildung 4: Acid2 im Firefox 2.0

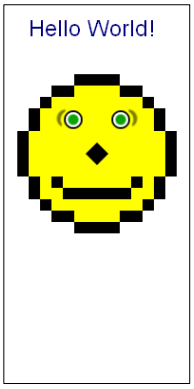

Abbildung 5: Acid2 im Firefox 3.6

Das Ergebnis in Firefox 3.6 (Abbildung 5) entspricht in allen Belangen der Referenz. Ein wenig davon abweichend rendert Firefox 2.0 (Abbildung 4). Trotz einer von der Referenz abweichenden Darstellung, fällt das Ergebnis verhältnismäßig gut aus (insbesondere im Direktvergleich mit dem von Internet Explorer 7). Auch bei diesem Browser sind gewisse Probleme einzukalkulieren, die gegebenenfalls zusätzliche Anpassungen erforderlich machen.

Opera

Die Opera-Familie stellt den dritten Testkandidaten dar. Nachfolgend die beiden Screenshots.

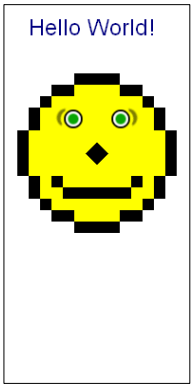

Abbildung 6: Acid2 in Opera 9.64

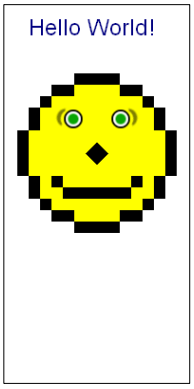

Abbildung 7: Acid2 in Opera 10.51

Da die Ergebnisse von sowohl Opera 9.64 (Abbildung 6) als auch 10.51 (Abbildung 7) der Referenz durchgängig gerecht werden, sind bei diesen Browsern keinerlei Probleme zu erwarten.

Safari

Die vierte und letzte Testgruppe setzt ihren Fokus auf Safari. Da, wie bereits erwähnt, diese Browserfamilie auf dieselbe Rendering-Engine wie Google Chrome setzt, wird dieser Browser auch gleich mit abgedeckt. Zusammen mit den anderen drei getesteten Browserfamilien ist somit der überwiegende Großteil an verbreiteten Browsern berücksichtigt.

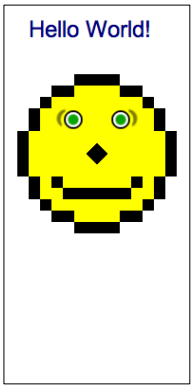

Abbildung 8: Acid2 in Safari 3.2

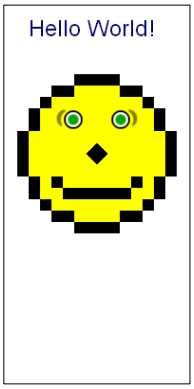

Abbildung 9: Acid2 in Safari 4.0

Sowohl Safari 3.2 (Abbildung 8) als auch Safari 4.0 (Abbildung 9) rendern den Acid2-Test der Referenz entsprechend und ohne auffällige Abweichungen. Eine Kleinigkeit, die erst bei näherer Betrachtung auffiel ist, dass Safari im Allgemeinen die „Nase“ des Smileys ein wenig kleiner darstellt, als es die anderen Browser tun. Trotz dieser Tatsache ist davon auszugehen, dass die korrekte Darstellung einer Standardkonformen Webseite mit Safari ohne Probleme möglich ist.

3.6 Probleme einzelner Browser

Aufgrund der Evaluierung der Standardkonformität haben sich drei Browser heraus kristallisiert, die zu potentiellen Problemen führen können. Auf die expliziten Mängel beziehungsweise Schwächen betreffend HTML- und CSS-Verarbeitung wird im Folgenden – für jeden Browser separat – eingegangen.

Aufgrund der Tatsache, dass die restlichen sechs Browser den Acid2-Test – von nicht nennenswerten Abweichungen abgesehen – problemlos bewältigen, wird auf diese hier nicht detailliert eingegangen. Das bedeutet allerdings nicht automatisch, dass diese frei von Fehlern sind. Lediglich die Korrektheit der Features, die von Acid2 getestet werden, ist sicher gestellt.

3.6.1 Internet Explorer 6.0

Wie bereits beim Acid2-Test zu erkennen ist, weist Internet Explorer 6 eine nennenswerte Anzahl an Problemen bei der CSS-Verarbeitung auf. Deren Auswirkungen reichen in der Praxis von leicht verschobenen Layouts bis hin zu komplett unbrauchbaren Webseiten. Wie gravierend die Abweichungen ausfallen, hängt maßgeblich davon ab, wie viele Features verwendet werden, die falsch implementiert sind oder überhaupt nicht unterstützt werden. Nachfolgend sind exemplarisch häufige Probleme aufgezählt.

Eines der häufigsten Probleme im Zusammenhang mit Internet Explorer 6 ist der so genannte „Box Model Bug“. Die CSS-Spezifikation sieht insgesamt vier Größenangaben für jedes Blockelement vor – Größe des Inhalts (Content Area), Padding (Innenabstand), eine Rahmenlinie (Border) mit beliebiger Stärke sowie Margin (Außenabstand) lassen sich festlegen. Abbildung 10 zeigt die Bereiche, auf die sich die einzelnen Angaben auswirken.

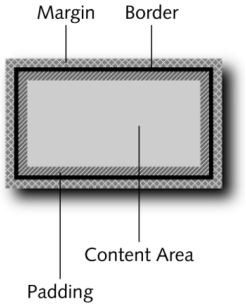

Abbildung 10: CSS Box Model
(Lowery, 2005, S. 38)

In standardkonformen Browsern wird die Content Area mit den angegebenen Maßen erstellt und anschließend mit Padding, Border und Margin nach außen hin erweitert. Bei vielen älteren Browsern – und auch beim Internet Explorer 6, sofern er im Quirks Mode läuft, ist das allerdings nicht so. In diesen Fällen werden Border und Padding von der Box subtrahiert. Das heißt, dass der Inhalt weniger Platz hat und die Box insgesamt kleiner ist, als sie sein sollte (Schmitt, Dominey, Li, Marcotte, Orchard, & Trammell, 2008, S. 41f).

Das zweite Problem in demselben Zusammenhang (fehlender oder falscher DOCTYPE) ist der „Doubling Margin Bug“. Elemente, deren Positionierung auf **`float`** gesetzt wurde, erhalten im Quirks-Modus ein Margin, das dem Doppelten des Festgelegten entspricht (Teague, 2009, S. 322).

Um diese beiden Fehler zu umgehen, ist es wichtig, dass der richtige DOCTYPE gesetzt ist, um den Browser im W3C-konformen Modus rendern zu lassen (siehe Kapitel 3.4).

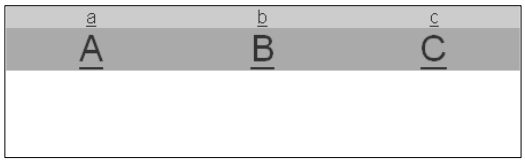

Abbildung 11: Korrekte Darstellung
in Firefox 2.0

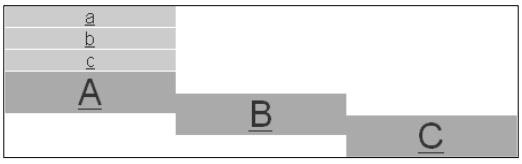

Abbildung 12: Fehlerhafte Darstellung
im Internet Explorer 6 („Treppeneffekt“)

Ein weiterer Bug im Internet Explorer 6 betrifft per **`float`** nebeneinander ausgerichtete Block-Elemente. Selbst, wenn die Elemente eine idente Höhenangabe aufweisen, werden diese nicht waagrecht nebeneinander angeordnet. Abbildung 11 zeigt die korrekte Darstellung einer solchen Situation. In Abbildung 12 ist der „Treppeneffekt“, der im Internet Explorer auftritt, gut zu erkennen. Umgehen lässt sich dieses Problem oft, indem das gemeinsame Elternelement mit **`display: inline`** formatiert wird (hasLayout.net, 2009).

Neben den Angaben **`width`** und **`height`** sieht CSS noch die Angaben **`min-height`**, **`max-height`**, **`min-width`** und **`max-width`** vor, um das Wachsen und Schrumpfen eines Elements

innerhalb von gegebenen Grenzen möglich zu machen. Internet Explorer in Version 6 kann mit diesen Attributen nichts anfangen und ignoriert sie folgedessen. Anzumerken ist, dass der Browser dafür `height` so interpretiert, als wäre es `min-height`. Das heißt, dass ein Container mit fixer Höhenangabe in Internet Explorer 6 trotzdem in der Höhe expandiert, wenn es der Inhalt erforderlich macht (Teague, 2009, S. 323).

Neben der direkten Auswirkung, dass eine CSS-Box mit festgelegter Höhe trotzdem noch expandiert, resultiert das in einem zweiten Problem. Das Attribut `overflow: auto`, das einen Container mit Scrollbalken versieht, sobald der Inhalt dessen Abmessungen übersteigt, ist somit nicht nutzbar.

Neben diesen beschriebenen Fehlern weist Internet Explorer 6 noch eine große Anzahl anderer Schwachstellen auf, die das Rendering betreffen. Neben nicht unterstützten Features sorgen dabei besonders Bugs, die nur unter Verkettung von bestimmten Umständen und eher selten auftreten, dafür, dass oft Nacharbeiten an einer Webseite notwendig sind.

3.6.2 Internet Explorer 7.0

Mit dieser Version behob Microsoft eine große Anzahl der Bugs, die in Internet Explorer 6 zu finden waren. Obwohl das Ergebnis des Acid2-Tests, auch bei diesem Browser, auf den ersten Blick sehr schlecht wirkt, wurden doch viele Fehler behoben. Die schlechte Darstellung des Tests liegt zu großen Teilen an weiterhin nicht unterstützten CSS-Features.

Insbesondere betreffen diese Verbesserungen das CSS Box Model, das in diesem Browser den W3C-Vorgaben entspricht. Die Größenberechnung funktioniert ab Version 7 des Microsoft-Browsers somit ident zu der in anderen, standardkonformen Browsern. Außerdem werden ab dieser Version CSS-Selektoren für Child (`html>p`) und Sibling (`td+td`) unterstützt (Mielke & Massy, 2006).

CSS-Hacks, die browserspezifische Bugs nutzen um gewisse Formatierungen nur an bestimmte Browser zu vermitteln, wurden und werden immer wieder gerne eingesetzt. Problematisch dabei ist allerdings, dass Internet Explorer 7 einige dieser Hacks anders interpretiert, als es Internet Explorer 6 tat. Detaillierte Informationen hierzu werden direkt von Microsoft bereit gestellt[4].

Eine Übersichtsseite[5] auf der Webplattform „The StyleWorks“ bietet einen guten Einblick in die einzelnen Problemstellen, die auf diesen Browser zutreffen. Dabei stellen unvorhersehbare Prozentangaben, Probleme im Zusammenhang mit Rahmenlinien bei Boxen sowie einige kleine Schwächen bei der CSS-Pseudoklasse `:hover` die größten Mängel dar. Teilweise tritt auch der bei Internet Explorer 6 schon angesprochen „Treppeneffekt“ auf (The StyleWorks).

3.6.3 Firefox 2.0

Verglichen mit den beiden Testergebnissen von Internet Explorer 6 und Internet Explorer 7 kommt das Ergebnis dieses Browsers ziemlich nahe an die Referenz heran. Ungeklärt ist allerdings, woher die Abweichungen stammen.

---

[4] http://msdn.microsoft.com/en-us/library/bb250496%28VS.85%29.aspx
[5] http://www.thestyleworks.de/tut-art/ie7.shtml

> *„Firefox is designed as a standards-compliant browser, which means that if you create your page following the rules of XHTML and CSS set forth by the W3C, your page should display correctly in Firefox. […] Firefox is currently in version 2.0 […].“* (Huddleston, 2008)

Kern dieser Aussage ist, dass bei der Entwicklung von Firefox 2 die bestmögliche Einhaltung von Webstandards im Fokus lag. Die Ursache für die abweichende Darstellung ist folgedessen weniger in fehlerhaften Implementierungen zu suchen, sondern vielmehr in CSS-Features, die schlichtweg nicht unterstützt werden.

Eine häufig verwendete und in Firefox 2 unzureichend implementierte Formatierung ist `display: inline-block`. Diese Darstellung steht zwar über das Mozilla-spezifische Attribut `display: -moz-inline-block` zur Verfügung – das Ergebnis ist aber trotzdem von der Spezifikation leicht abweichend und nicht immer zufrieden stellend (Koch, 2009).

4 JavaScript

Die bisher beschriebenen Technologien ermöglichen lediglich die Darstellung von statischen Webseiten. Erst mit der Einführung von JavaScript wurde ein direktes und clientseitiges Manipulieren von Webseiten ermöglicht.

Nach einem kleinen geschichtlichen Abriss über die Entstehung werden die Kompatibilität der einzelnen Browser bezüglich JavaScript untersucht und potentielle Problemstellen erläutert.

4.1 Entstehung

Bereits 1992 wurden die Grundlagen für die heutige Form von JavaScript geschaffen. Allerdings hieß die Sprache damals noch C-minus-minus und war dafür gedacht, Makros zu ersetzen. Die dahinter stehende Firma Nombas erkannte kurz darauf die steigende Popularität des Netscape Navigators und brachte unter dem Namen ScriptEase eine Version der Sprache heraus, die sich in Webseiten eingebettet verwenden ließ.

Die Anforderung, trotz verhältnismäßig langsamer Internetverbindungen (analoge Telefonmodems) eine schnelle Validierung von Formulardaten zu ermöglichen, machte die Notwendigkeit einer clientseitigen Browser-Skriptsprache deutlich. Mit der Veröffentlichung von Netscape Navigator 2 im Jahr 1995 sollte LiveScript eingeführt werden. Da die Entwicklung des Browsers unter Zeitdruck stattfand, wurde Sun als Partner engagiert und half maßgeblich bei der Fertigstellung von LiveScript mit. Um von der gerade aufkommenden Popularität der Programmiersprache Java zu profitieren, wurde die Sprache schlussendlich in JavaScript umbenannt.

Aufgrund der großen Popularität von JavaScript im Netscape-Browser sprang 1996 auch Microsoft auf den Zug auf und entwickelte eine eigene, nicht kompatible Version von JavaScript unter dem Namen JScript. Damit existierten drei parallele Standards, die jeweils ihren eigenen (Syntax-)Regeln folgten. Aufgrund dieser Entwicklung wurde erkannt, dass es sinnvoll ist, eine herstellerunabhängige Skriptsprache für Browser zu entwickeln und zu standardisieren. 1997 wurde die erste Version, JavaScript 1.1, als Standard vorgeschlagen und

schließlich als ECMA-262 (European Computer Manufacturers Association) Standard veröffentlicht. (Zakas, 2009, S. 1f)

JavaScript umfasst dabei noch mehr, als nur die Skriptsprache ECMA-262 alias ECMAScript selbst. Zusätzlich werden noch DOM und BOM (Browser Object Model) als Teilmengen von JavaScript angesehen. Über die beiden Letztgenannten wird die Interaktion des Skripts mit dem Browserfenster sowie der dargestellten Webseite ermöglicht (Zakas, 2009, S. 3).

Aus Sicherheitsgründen läuft JavaScript generell in einer Sandbox. Das heißt, dass der Zugriff auf lokale Dateien (mit Ausnahme von Cookies) und gewisse Formularfelder nicht möglich ist. Das trifft insbesondere auf das Formularfeld für den Dateiupload zu, bei dem es nicht möglich ist, per JavaScript den gewählten Pfad zu ändern. Dies schützt den User davor, dass eine Datei hochgeladen wird, die er nicht ausgewählt hat (Edwards & Adams, 2006, S. 3f).

Im Web 2.0, das massiv auf dynamische Inhalte und Interaktionen mit dem Benutzer setzt, ist JavaScript nicht mehr weg zu denken. Einerseits wegen seiner grundlegenden Aufgabe, clientseitige Validierungen durchzuführen, andererseits, weil es die Basis für Ajax bildet. Erst diese asynchrone Methode zur Datenübertragung macht hochgradig dynamische Webanwendungen überhaupt möglich.

4.2 Umfang der Unterstützung

Browser	ECMAScript Edition	DOM Level 1	DOM Level 2	DOM Level 3
Internet Explorer 6	3rd	teilweise	nein	nein
Internet Explorer 7	3rd	teilweise	nein	nein
Internet Explorer 8	3rd	teilweise	teilweise	nein
Firefox 2.0	3rd	ja	teilweise	nein
Firefox 3.6	3rd	ja	teilweise	nein
Opera 9.64	3rd	ja	ja	teilweise
Opera 10.51	3rd	ja	ja	teilweise
Safari 3.2.2	3rd	ja	ja	nein
Safari 4.0.5	3rd	ja	ja	nein

Tabelle 1: Browservergleich bezüglich DOM[6] und JavaScript (ECMAScript)

Da der Unterstützungsgrad der einzelnen Browser stark unterschiedlich ist, sind zunächst einige Fakten in Tabelle 1 zusammen gestellt. Die Auswahl der Browser entspricht jener, die am Beginn dieser Arbeit getroffen wurde.

Anzumerken ist, dass die Unterstützung von DOM Level 1 in Internet Explorer 8 deutlich verbessert wurde. Dennoch ist sie weiterhin nicht vollständig, weswegen einer der Tests des W3C-Checkers fehlschlägt.

Ein Vergleich der JavaScript-Unterstützung ist insofern schwierig, da JavaScript in jeder Engine anders versioniert wird. Auch die interne Bezeichnung ist oft sehr unterschiedlich –

[6] DOM-Kompatibilität laut W3C-Checker (http://www.w3.org/2003/02/06-dom-support.html)

bei Mozilla wird die Technologie unter JavaScript geführt, bei Microsoft unter JScript und bei Opera unter ECMAScript.

Als Gemeinsamkeit ist zu nennen, dass die eben erwähnten Technologien alle auf der 3rd Edition der ECMAScript-Sprache aufbauen. Die einzelnen Implementierungen der Browserhersteller weisen jedoch teilweise unterschiedliche Funktionalitäten auf. In vielen JavaScripts finden sich daher Abfragen, die am Vorhandensein eines Objekts oder einer Methode den verwendeten Browser ermitteln.

Noch inkonsistenter als bei JavaScript gestaltet sich der Sachverhalt beim BOM. Das Problem hier ist, dass es weder Standards noch Richtlinien gibt, was hier unterstützt werden soll. Dennoch bieten die meisten Browser eine gewisse Standardpalette an BOM-Objekten an, wie beispielsweise das Objekt `window`, um auf die Eigenschaften des aktuellen Fensters zuzugreifen (Zakas, 2009, S. 11).

4.3 Probleme einzelner Browser

Aufgrund der Ergebnisse des W3C-Checkers ist festzustellen, dass bezüglich JavaScript und DOM hauptsächlich Probleme mit Browsern der Internet Explorer Familie zu erwarten sind. Insbesondere, aber nicht ausschließlich, betrifft das die Versionen 6 und 7 des Browsers. In Internet Explorer 8 wurde die Unterstützung für DOM Level 1 deutlich verbessert und entspricht nun fast der Spezifikation.

Einige Methoden, die in der Praxis sehr häufig eingesetzt werden, beispielsweise `document.getElementsByName()` oder `Array.indexOf()` funktionieren allerdings in keiner der drei Browserversionen. Microsoft hat angekündigt, diese Schwächen bei der kommenden Version, Internet Explorer 9, auszumerzen. Zusätzlich sollen auch die DOM-Level 2 und 3 besser unterstützt werden (Hachamovitch, 2010).

5 PNG Grafikformat

Neben HTML, CSS und JavaScript stellen Bitmap-Grafiken eine Basistechnologie im Webbereich dar. Traditioneller weise wurde lange auf die Formate GIF (Graphics Interchange Format) und JPEG (Joint Photographic Experts Group), mit ihren jeweiligen Vor- und Nachteilen, gesetzt. Speziell in den letzten Jahren erfreute sich das PNG-Format steigender Beliebtheit.

Die Gründe hierfür sind zweierlei. Einerseits ist PNG ein freies Bildformat, das an keine Lizenzgebühren oder ähnliches gebunden ist. Andererseits ist es mit PNG möglich, die von GIF-Bildern bekannte Transparenz zu nutzen. Der große Vorteil hier ist allerdings, dass sich die Transparenz nicht nur binär maskieren, sondern per Alpha-Channel abgestuft für jedes einzelne Pixel festlegen lässt.

5.1 Entstehung

Zu Beginn des Jahres 1995 wurde von Unisys und CompuServe verkündet, dass künftig für GIF-Implementierungen Lizenzgebühren fällig werden könnten. Grund dafür war, dass das Bildformat auf einen Kompressionsalgorithmus von Unisys setzt. Im Endeffekt wurde die

Drohung nie verwirklicht, aber die Entwicklung von PNG wurde damit ins Leben gerufen (Roelofs, 2009).

Ziel dieser Entwicklung war es, ein Format zu schaffen, das mit GIF mithalten kann. Das heißt, dass es Bilder verlustfrei komprimieren soll (abgesehen von der Farbreduktion auf maximal 256 Farben ist GIF verlustfrei!) und diese transparente Bereiche aufweisen können. Da sich ein Klon des GIF-Formats ohne Mehrwert schlecht am Markt hätte etablieren können, bietet PNG wesentlich mehr Optionen.

Neben dem 256-Farben-Modus, der von GIF-Bildern bekannt ist, lassen sich auch Echtfarben-Bilder (24 oder 48 Bit) im PNG-Format speichern. Zusätzlich ist es – wie schon angesprochen – auch möglich, Transparenz-Informationen für jeden Pixel festzulegen. Diese Information wird im Alpha-Channel gespeichert und liegt üblicherweise mit 8 Bit vor. Das heißt, dass sich für jeden Pixel sehr genau festlegen lässt, mit wie viel Deckkraft er gegenüber dem Untergrund dargestellt werden soll.

Das PNG-Format speichert nicht nur Bitmap-Bilder, vielmehr werden diese auch noch verlustfrei komprimiert. Die Kompressionsrate übersteigt dabei in den meisten Fällen jene, die beim GIF-Format zu erreichen ist.

5.2 Umfang der Unterstützung

Seit seiner Einführung hat sich das PNG-Format gut etabliert und wird von allen gängigen Bildbearbeitungsprogrammen, Browsern und vielen anderen Anwendungen unterstützt. Aufgrund der möglichen Alpha-Transparenz und der verlustfreien Komprimierung ist es in vielen Belangen JPEG und GIF überlegen und wird deswegen auch häufig eingesetzt. Nicht zuletzt auf Webseiten, wo PNG-Grafiken allgegenwärtig sind.

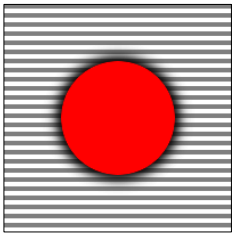

Abbildung 13: Korrekte Transparenz

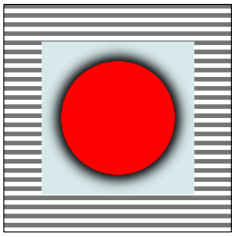

Abbildung 14: Fehlerhafte Transparenz im Internet Explorer 6

Die Aussage, dass PNG in allen gängigen Browsern unterstützt wird, stimmt zwar (alle neun Browser aus der anfänglich getroffenen Auswahl können dieses Format verarbeiten), allerdings weist Internet Explorer 6 ein Problem auf, das bei PNGs mit Alpha-Transparenz auftritt.

Anstatt die Transparenz entsprechend dem Alpha-Channel anzupassen und den Hintergrund entsprechend farblich zu verändern (siehe Abbildung 13), füllt Internet Explorer 6 den Hintergrund der Grafik einfach mit hellgrauer Farbe (siehe Abbildung 14). Neben der schlichtweg falschen Farbe bedeutet das auch, dass dahinter liegende Elemente verdeckt werden.

Das angesprochene Problem tritt ausschließlich bei Bildern auf, die Alpha-Transparenz verwenden. PNGs, die eine indizierte Farbe für die Transparenz-Information setzen (so, wie das bei GIF der Fall ist), werden korrekt dargestellt.

In den Nachfolgern des Browsers, also ab einschließlich Internet Explorer 7, ist dieses Problem behoben und auch PNGs mit Alpha-Transparenz werden korrekt dargestellt.

6 Probleme lösen & umgehen

Nachdem Entstehungsgeschichte, Aufgabenbereiche und Problemstellen der einzelnen Technologien behandelt wurden, geht es in diesem Kapitel um das Beseitigen der jeweiligen Probleme.

Die erste wichtige Anforderung in diesem Zusammenhang ist ein fundiertes Wissen über potentielle Fehlerquellen und Browserbugs. Dadurch ist es bereits während der Entwicklung und noch vor dem Testen einer Webplattform möglich, Fehler zu umgehen oder auszubessern. Beispielsweise gibt es bei CSS-Layouts und JavaScript-Funktionen oft mehrere Wege, die zum Ziel führen. Ist gute Browserkompatibilität gefordert, sollte gleich jene Methode gewählt werden, die die wenigsten Probleme mit sich bringt.

Natürlich bedeuten solche „kompatiblen Lösungsansätze“ oft einen zunächst größeren Arbeitsaufwand oder wirken sich negativ auf die Performance von JavaScripts aus. In vielen Fällen relativieren sich diese Nachteile allerdings schnell, wenn dadurch bei der Anpassung an die verschiedenen Browser weniger Zeit benötigt wird. Auch die Notwendigkeit von CSS-Hacks und anderer unsauberer Lösungen kann so im Rahmen gehalten werden.

Sinnvoll ist es auch, die zu unterstützenden Browser bereits zu Beginn eines Projektes festzulegen. Somit ist es möglich, bereits während der Entwicklung mit allen Zielbrowsern zu testen. Auftretende Probleme werden so nach und nach aufgedeckt und können meist innerhalb kurzer Zeit behoben werden. Die Webplattform bleibt somit während der gesamten Entwicklungszeit in sämtlichen Browsern zum Testen zur Verfügung.

Wird zur Entwicklung hingegen lediglich ein Browser, beispielsweise Safari 4, als Testplattform verwendet, bringt das einige Nachteile mit sich. Sämtliche CSS-Formatierungen und JavaScripts funktionieren wohl in Safari (und mit hoher Wahrscheinlichkeit auch in anderen modernen Browsern), aber über die Kompatibilität zu älteren Browsern lassen sich nur Mutmaßungen anstellen.

Mit gegebenenfalls großen Problemen ist zu rechnen, wenn bei dieser Herangehensweise seitens der Softwareentwicklung später ein zusätzlicher Browser als Kompatibilitätskriterium hinzu kommt. Beim ersten Test, beispielsweise mit Internet Explorer 6, ist oft eine unbrauchbare Webseite vorzufinden. Das heißt, dass das Layout „zerstört“ ist, gewisse Elemente gar nicht auf dem Bildschirm zu sehen sind und vieles mehr.

Das größte Problem hier ist allerdings, dass JavaScripts meistens erst ausgiebig getestet und repariert werden können, nachdem das Layout (zumindest notdürftig) in Ordnung gebracht wurde. Durch Umgehen dieser Wartezeit – wie angesprochen durch frühes Spezifizieren der Browserauswahl und regelmäßiges Testen in allen diesen Browsern – lässt sich die Effizienz im Projektteam steigern und Frust minimieren.

6.1 Vermeiden von Inkompatibilitäten

Wie bereits angesprochen ist es oft der bessere Ansatz, Probleme zu umgehen, anstatt sie nachträglich mühsam zu beheben. Oft bedeutet das zwar einen erhöhten anfänglichen Arbeitsaufwand, spart aber später Zeit bei der Problemlösung.

Bei allen drei angesprochenen Kerntechnologien – das heißt HTML mit CSS, JavaScript und Bilddateien – ist es wichtig, den kleinsten gemeinsamen Nenner zu kennen. Das heißt, im Vorfeld bereits zu wissen, welche Browser unterstützt werden sollen und welche Fähigkeiten diese haben. Dabei sollten natürlich auch nicht vorhandene Funktionalitäten und Bugs der einzelnen Browser bekannt sein.

Während der Entwicklung sollte ständig darauf Rücksicht genommen werden, an welchen Stellen Probleme zu erwarten sind. Das heißt, dass im Idealfall keine nicht unterstützten oder fehlerhaften Browserfeatures verwendet werden. Falls dies doch notwendig ist, muss evaluiert werden, wie die Funktionalität in problematischen Browsern trotzdem wieder hergestellt werden kann.

6.1.1 HTML & CSS

Ein erster wichtiger Schritt in Richtung Browserkompatibilität ist es, die Style-Angaben für möglichst alle Elemente zurück zu setzen. Ist keine CSS-Datei angegeben, werden trotzdem Überschriften der verschiedenen Ebenen und Links unterschiedlich formatiert. Verantwortlich dafür ist das Default-Stylesheet des jeweiligen Browsers.

Wird ein eigenes Stylesheet in die Webseite eingebunden, so überschreiben die darin angegebenen Formatierungsanweisungen die des Default-Stylesheets teilweise. Genau dieses teilweise Überschreiben stellt aber oft ein Problem dar beziehungsweise führt zu Inkonsistenzen.

Es hängt vom Browser ab, wie viele Pixel Abstand nach einer Überschrift eingehalten werden. Bei eingerückten Absätzen (beispielsweise für Aufzählungen) wird die Einrückung bei manchen Browsern über `margin` geregelt, bei anderen über `padding`. Dies darf nicht als Fehler gesehen werden, da es eben nur ein Standard-Stylesheet ist, um Seiten notdürftig darstellen zu können. Werden diese Werte nicht anfangs explizit angegeben, kann es passieren, dass sich `padding` und `margin` addieren und Elemente verschoben dargestellt werden oder Schriften in falscher Größe gerendert werden (Schmitt, CSS Cookbook: Third Edition, 2010, S. 268f).

Ein zweiter wichtiger Punkt in diesem Zusammenhang betrifft Größenangaben von DIV-Elementen und Text. CSS sieht zahllose Möglichkeiten vor, dies zu beeinflussen. Zentimeter, Millimeter, Zoll, px (Pixel), Pica, Punkt, `em` (Höhe des Buchstaben „M“), `x` (Höhe des Buchstaben „x“) sowie Prozent sind dabei gültige Einheiten. Welche der gebotenen Möglichkeiten beim Layout an welcher Stelle sinnvoll einzusetzen sind, hängt maßgeblich von dessen Konzept ab. Handelt es sich um ein fixes Layout, kommen eher absolute Angaben, wie beispielsweise Pixel, in Frage. Ist es ein Layout, das sowieso die gesamte Fensterbreite ausfüllen soll, sind Prozentangaben ratsam (Huddleston, 2008, S. 86ff).

Beim Spezifizieren der Schriftgröße gilt grundsätzlich die Regel, dass auf relative Angaben zurück gegriffen werden sollte. Der Grund dafür ist, dass Benutzer mit visuellen Einschränkungen so von der Möglichkeit Gebrauch machen können, die Basisschriftgröße zu

verändern. Wurden etwa Pixel oder Punkt als Größenangabe verwendet, macht dies in manchen Browsern eine Skalierung durch den Benutzer unmöglich.

Um trotz weiterhin möglicher Skalierung pixelgenaue Angaben machen zu können, ist empfohlen, dem `body`-Element `font-size: 62.5%` zuzuweisen. Standardmäßig entspricht die Schriftgröße von `1em` umgerechnet `16px`. Nach dieser Zuweisung entspricht `1em` ziemlich genau `10px`. Somit lässt sich komfortabel in `em` rechnen, während (100% Zoomstufe vorausgesetzt) trotzdem pixelgenaue Angaben möglich sind (Schmitt, CSS Cookbook: Third Edition, 2010, S. 121f).

Von Schriftgrößenangaben in Punkt oder Millimeter sollte – sofern es sich um eine Bildschirmausgabe und keine Druckansicht handelt – Abstand genommen werden, da diese in jedem Browser beziehungsweise Betriebssystem unterschiedlich in Bildschirmpixel umgerechnet werden.

Während sich CSS-Bugs in vielen Fällen mit Hacks und dergleichen umgehen lassen, ist das bei Eigenschaften oder Selektoren, die nicht unterstützt werden, schon komplizierter. Deswegen sollte nach Möglichkeit auf solche Formatierungen verzichtet werden, die von einem der gewünschten Browser überhaupt nicht unterstützt werden. Als Beispiel lässt sich hier das bereits erwähnte `display: inline-block` nennen, das in Firefox 2 nicht ordnungsgemäß funktioniert.

6.1.2 JavaScript

Hauptproblem bei der Verwendung von JavaScript in unterschiedlichen Browsern ist deren inhomogene Unterstützung des DOM. Während neuere Opera-Versionen bereits DOM Level 3 teilweise unterstützen, hapert es beim Internet Explorer oft noch an den Basisfunktionalitäten von DOM Level 1. Zwar ist Internet Explorer 8 diesbezüglich schon deutlich besser als seine beiden Vorgänger, aber erst bei Internet Explorer 9 ist mit einer guten DOM-Unterstützung zu rechnen.

Um Problemen aus dem Weg zu gehen, empfiehlt sich oft der Einsatz einer JavaScript Library. Je nach Einsatzzweck und Grad der Spezialisierung gibt es hier eine große Auswahl. Wird eine solche als Basis für die Webapplikation gewählt, sind schon viele Probleme behoben. Der Grund dafür ist, dass die Libraries eine vom Browser unabhängige und einheitliche Programmierung ermöglichen. Funktionen, die von einem Browser nicht direkt unterstützt aber häufig benötigt werden, werden in der Regel von den Libraries selbst implementiert, um trotzdem zur Verfügung zu stehen. Bekannte Libraries für JavaScript sind – unter anderem – das Dojo Toolkit und jQuery (Zakas, 2009, S. 759ff).

Für Funktionen, die nicht auf der gewählten Library aufbauen (oder für den Fall, dass keine Library eingesetzt wird) ist es wichtig, regelmäßig mit den unterschiedlichen Browsern zu testen. Insbesondere ist auf Internet Explorer 6 und 7 aufzupassen, da hier die DOM-Unterstützung größere Defizite aufweist. Aufgrund der allgemeinen JavaScript-Inhomogenität sollten aber auch alle anderen Browser ausreichend in die Tests mit einbezogen werden.

6.1.3 PNG-Bilder

Aus Rücksicht auf den Internet Explorer 6 und das Problem, dass dieser PNG-Bilder mit Alpha-Kanal nicht korrekt darstellt, ist es Weg des geringsten Widerstandes, auf solche Bilder zu verzichten. Stattdessen sollten Grafiken im GIF- oder PNG-Format mit indizierter

Transparenz und nur 8 Bit Farbtiefe verwendet werden. In vielen Fällen ist es so möglich, eine korrekte Darstellung der Webseite zu erreichen, ohne Abstriche machen zu müssen.

Natürlich ist dies nur dort möglich, wo sich die Transparenz vorab schon mit dem „Untergrund“ verrechnen lässt. Das bedeutet, dass sie entweder auf einem einfarbigen Untergrund oder immer an derselben Stelle der Webseite dargestellt werden müssen.

Desweiteren ist es erforderlich, dass es sich dabei um Grafiken handelt, die auch mit 256 Farben noch akzeptabel aussehen. Das heißt, dass dies für kleinere Icons durchaus zutrifft, während das Ergebnis bei großen Fotos nicht allzu befriedigend ausfallen wird.

6.2 Beheben von CSS-Problemen

Viele potentielle Probleme lassen sich durch eine gut gewählte Herangehensweise bereits im Vorfeld vermeiden beziehungsweise umgehen. Trotzdem kommt es so gut wie immer zu einigen Darstellungsunterschieden zwischen den Browsern, die behoben werden müssen. Möglichkeiten hierfür werden in diesem Kapitel, mit ihren jeweiligen Vor- und Nachteilen, erläutert.

Da, je nach Browser unter Umständen noch eine Vielzahl anderer Methoden zur Verfügung steht, handelt es sich dabei lediglich um eine Auswahl häufig angewandter Lösungswege. In jedem Fall wird davon ausgegangen, dass die Webseite vom Browser im standardkonformen Modus gerendert wird und das Default-CSS ausreichend überschrieben ist.

6.2.1 Browserweichen

Eine naheliegende und verhältnismäßig häufig eingesetzt Möglichkeit, um Darstellungsfehler auszubessern, sind so genannte Browserweichen. Die Grundidee dahinter ist, für jeden zu unterstützenden Browser eine eigene CSS-Datei bereit zu stellen. Möglichkeiten zur Browsererkennung und Wahl der zu verwendenden Stylesheets existieren sowohl serverseitig als auch clientseitig.

Soll bereits am Webserver entschieden werden, welche CSS-Datei zu verwenden ist, bietet sich die Auswertung des User Agent Strings an. Bei jedem Request des Browsers befindet sich dieser im HTTP-Header und enthält Informationen über den verwendeten Browser. Eine am Server verfügbare Skriptsprache wie PHP (PHP Hypertext Preprocessor) oder ASP (Active Server Pages) verarbeitet diesen String, ermittelt den Browser und wählt die zu sendende Stylesheet-Datei aus. Tabelle 2 zeigt anhand der auch bisher verwendeten Browserauswahl, wie die einzelnen User Agent Strings aussehen.

Dabei ist zu erkennen, dass neben Browsername und Version auch diverse Informationen über das Betriebssystem, die Sprache des Browsers, Kompatibilitäten zu anderen Browsern und vieles mehr in diesem String enthalten ist. Was dabei preisgegeben wird, und in welcher Form die Informationen vorliegen, ist dabei von Browser zu Browser sehr unterschiedlich.

Problematisch bei dieser Vorgehensweise ist allerdings, dass sich der User Agent String leicht verändern lässt. Beispielsweise Opera lässt dies sogar über ein Menü der grafischen Oberfläche zu, aber auch bei den meisten anderen Browsern ist dies mit geringem Aufwand möglich. Das heißt, dass die Informationen, die sich über diesen String gewinnen lassen, nicht unbedingt stimmen müssen.

Browser	User Agent String
Internet Explorer 6	Mozilla/4.0 (compatible; **MSIE 6.0**; Windows NT 5.1; SV1)
Internet Explorer 7	Mozilla/4.0 (compatible; **MSIE 7.0**; Windows NT 5.1)
Internet Explorer 8	Mozilla/4.0 (compatible; **MSIE 8.0**; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)
Firefox 2.0	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.20) Gecko/20081217 **Firefox/2.0.0.20**
Firefox 3.6	Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9.2.3) Gecko/20100401 **Firefox/3.6.3**
Opera 9.64	**Opera/9.64** (Windows NT 5.1; U; de) Presto/2.1.1
Opera 10.51	**Opera**/9.80 (Windows NT 5.1; U; de) Presto/2.5.22 Version/**10.51**
Safari 3.2.2	Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE) AppleWebKit/525.28 (KHTML, like Gecko) Version/**3.2.2 Safari**/525.28.1
Safari 4.0.5	Mozilla/5.0 (Windows; U; Windows NT 5.1; de-DE) AppleWebKit/531.22.7 (KHTML, like Gecko) Version/**4.0.5 Safari**/531.22.7

Tabelle 2: User Agent Strings der ausgewählten Browser

Als Vorteil der serverseitigen Browserweiche ist zu erwähnen, dass diese für den Benutzer unsichtbar und vom Browser weitgehend unabhängig funktioniert. Außerdem wird Bandbreite gespart, da der Benutzer nur so viel CSS- beziehungsweise HTML-Code übertragen bekommt, wie letzten Endes für die Darstellung notwendig ist.

Clientseitig bietet sich an, den Browser per JavaScript auf vorhandene Methoden und Objekte zu testen. Aus den unterschiedlichen Kombinationen von Unterstützung und Nichtunterstützung lässt sich der verwendete Browser gut bestimmen. Vorteil dieser Methode ist, dass sie auch bei einem veränderten User Agent String problemlos funktioniert. Nachteilig ist allerdings, dass für diese Überprüfung zusätzlicher Quellcode übertragen werden muss und diese von JavaScript abhängig ist (Lowery, 2005, S. 72).

Eine weitere Möglichkeit, die allerdings nur im Internet Explorer funktioniert, ist die Verwendung von so genannten „Conditional Comments“ im HTML-Code. Aufgrund der Tatsache, dass diese Anweisungen in der HTML-Syntax eigentlich Kommentare sind, werden die darin enthaltenen Codeteile von anderen Browsern schlichtweg ignoriert und das Dokument bleibt weiterhin validierbar.

Innerhalb der Internet Explorer Familie lässt sich auch noch festlegen, für welche Browserversionen der im Conditional Comment enthaltene Code eingebunden werden soll. Nachfolgendes Beispiel zeigt eine mögliche Anwendung.

```
<!--[if lte IE 6]>
    <link rel='stylesheet' type='text/css' href='old_ie.css'>
<![endif]-->

<!--[if IE 7]>
    <link rel='stylesheet' type='text/css' href='ie7.css'>
<![endif]-->
```

Diese beiden Weichen bewirken, dass für Internet Explorer 6 oder älter die Stylesheet-Datei „old_ie.css“ verwendet wird (`lte` = lower than or equal). Für Internet Explorer 7 wird stattdessen „ie7.css“ eingebunden. Beide Dateien enthalten Korrekturen, die an den jeweiligen Browser angepasst sind. Die zur Verfügung stehenden Operatoren zur Browserdifferenzierung sind in der Online-Dokumentation[7] von Microsoft aufgelistet.

Im Allgemeinen stellt die Verwendung von Browserweichen eine gute Lösung dar. HTML- beziehungsweise CSS-Dateien sind weiterhin valide und jeder Browser bekommt nur den Code ausgeliefert, den er für die Darstellung der Webseite benötigt. Aufgrund der Tatsache, dass mehrere Stylesheet-Dateien gepflegt werden müssen, ist allerdings auch mit einem gewissen Mehraufwand zu rechnen.

6.2.2 CSS-Hacks

Insbesondere für kleinere Korrekturen und Fälle, in denen weder JavaScript noch eine serverseitige Browserweiche in Frage kommen, bieten sich so genannte „CSS-Hacks“ an. Diese nützen Bugs und Lücken in der CSS-Verarbeitung der einzelnen Browser aus, um gewisse Codeteile zwar für den Zielbrowser zugänglich zu machen, sie aber vor allen anderen Browsern zu verstecken. Je nach Browser und Version stehen dabei sehr unterschiedliche Hacks zur Verfügung. Die Herausforderung dabei ist es, diese derart zu kombinieren, dass sie sich gegenseitig nicht unerwünscht beeinflussen (Lowery, 2005, S. 8ff).

Um die Funktionsweise und das Aussehen von Hacks zu verdeutlichen, wird nachfolgend ein kleiner Ausschnitt aus einer CSS-Datei gezeigt und für ausgewählte Problembrowser jeweils ein CSS-Hack vorgestellt. Die allgemein gültige CSS-Deklaration sieht wie folgt aus:

```
div.info {
    width: 200px;
}
```

Diese Anweisung wird zunächst von allen Browsern interpretiert. Anschließend folgen die diversen Hacks, die die Unzulänglichkeiten der Problembrowser beheben. Folgender Hack ist auch als „Star-HTML-Hack“ bekannt und wird von Internet Explorer bis einschließlich Version 6 ausgewertet:

```
* html div.info {
    width: 190px;
}
```

Dieser Hack basiert auf dem vermeintlich existierenden Elternelement von `html`. Auch wenn dieses nicht existiert, wird es im Internet Explorer 6 korrekt angewandt. Die CSS-Datei validiert trotz dieses Hacks weiterhin problemlos (Bongard, 2007).

Für Korrekturen, die ausschließlich auf Internet Explorer 7 angewandt werden sollen, gibt es einen Hack, der ebenfalls auf einem nicht existierenden und `html` übergeordneten Element basiert. Er sieht folgendermaßen aus:

```
*:first-child+html div.info {
    width: 195px;
}
```

[7] http://msdn.microsoft.com/en-us/library/ms537512%28VS.85%29.aspx

Da es sich bei Firefox 2 um einen W3C-konformen Browser handelt, sollte die Verwendung von Hacks in den meisten Fällen nicht notwendig sein. Allerdings ist es durchaus möglich, über Mozilla-spezifische Attribute Formatierungen vorzunehmen, die über die offizielle CSS-Unterstützung nicht möglich sind.

```
span.box {
    width:    100px;
    height:   20px;
    display:  -moz-inline-block;
    display:  inline-block;
}
```

In obigem Codebeispiel wird die eigentlich von Firefox 2 nicht unterstützte Darstellungsvariante **`inline-block`** über ein Mozilla-spezifisches Attribut umgangen. Eine CSS-Datei, die diesen Hack beziehungsweise dieses Workaround enthält, ist allerdings nicht mehr valide.

Bei den vorgestellten Hacks handelt es sich, wie bereits angesprochen, nur um eine kleine Auswahl, um deren Funktionsweise zu erläutern und grundsätzlich zu zeigen, wie sie verwendet werden. In der Praxis bieten Hacks oft den Vorteil, dass nur eine CSS-Datei vorhanden ist, die gewartet werden muss. In dieser befinden sich meist die Browserhacks gleich nach den „gewöhnlichen" Style-Angaben. Verglichen mit mehreren Stylesheet-Dateien führt dies zu einer erhöhten Übersichtlichkeit.

Nachteilig an Hacks ist vor allem die Tatsache, dass sie oftmals eingesetzt werden, wo dies gar nicht nötig wäre. In vielen Fällen lässt sich deren Einsatz durch fundierte Kenntnisse über CSS im Allgemeinen und die zu unterstützenden Browser im Speziellen vermeiden. Außerdem führen viele Hacks auch dazu, dass eine Validierung des Stylesheets fehlschlägt.

6.2.3 JavaScript-Lösung mit IE7.js

Obwohl Internet Explorer 6 zum Zeitpunkt seines Erscheinens ein verhältnismäßig guter Browser war, führte seine lange Lebensdauer zunehmend zu Problemen für Web Designer. Aus diesem Grund entschied sich Dean Edwards, eine JavaScript-Library zu entwickeln, die diese Probleme umgeht und behebt. Die offizielle Beschreibung der Library lautet:

> *„IE7.js is a JavaScript library to make Microsoft Internet Explorer behave like a standards-compliant browser. It fixes many HTML and CSS issues and makes transparent PNG work correctly under IE5 and IE6."*
> (Edwards D. , 2009)

Zwischenzeitlich wurde der Support dahingehend ausgedehnt, dass auch Features von Internet Explorer 8 unterstützt werden. Natürlich kann dieses Skript keine Wunder vollbringen und sämtliche Schwachstellen beheben. Allerdings ist es dadurch ohne zusätzlichen Aufwand beziehungsweise Hacks möglich, Attribute wie **`position: fixed`** und **`max-height`** im Internet Explorer 6 zu verwenden. Außerdem werden einige CSS-Selektoren ergänzt, die ursprünglich nicht unterstützt wurden.

Die komplette Liste an unterstützten Features respektive behobenen Fehlern findet sich auf der Webseite des Projekts in Form einer Browser-Testseite[8].

Der größte Vorteil dieser Lösung ist ohne Zweifel, dass sie verhältnismäßig viele Probleme beseitigt und schnell in eine bestehende Webseite integriert werden kann. Nachteilig ist, wie auch schon bei Browserweichen auf CSS-Basis, dass der Zielbrowser über aktiviertes JavaScript verfügen muss. Da auch das Skript selbst an den Client geschickt werden muss, steigt die zu übertragende Datenmenge (beim Laden der Webseite) geringfügig an.

6.3 Beheben von JavaScript-Problemen

Wie auch bei CSS gibt es auch im Bereich von JavaScript immer wieder Fälle, in denen sich Probleme nicht wirklich umgehen lassen und gelöst werden müssen. Dies ist speziell dann der Fall, wenn der Einsatz von externen Libraries wie jQuery nicht möglich oder unerwünscht ist.

Basis der meisten JavaScript-Probleme sind nicht vorhandene Methoden und Objekte. Fehlende Methoden lassen sich von Hand nachbilden oder, für den Fall, dass eine Methode fehlerhaft ist, durch eine eigene Implementierung ersetzen. Nachfolgender Codeauszug zeigt exemplarisch, wie sich eine im Internet Explorer nicht korrekt implementierte Methode überschreiben lässt.

```
document.getElementsByName = function(name, tag) {...}
```

Zu erwähnen ist hier, dass es sinnvoll ist, den Browser abzufragen (beispielsweise über Conditional Comments) und die Methode nur dann zu überschreiben, wenn sie nicht schon vorhanden ist. Der Grund dafür ist, dass selbst geschriebene Methoden in der Regel eine schlechtere Performance aufweisen als solche, die vom Browser direkt bereitgestellt werden.

6.4 Transparente PNGs & Internet Explorer 6

Während Probleme mit CSS und JavaScript in diversen Browsern auftreten, betrifft dieses Problem ausschließlich Internet Explorer 6. Zwar lässt sich in vielen Fällen die Verwendung von PNG-Bildern mit Alpha-Transparenz umgehen, aber nicht in allen. Speziell bei unregelmäßigen Hintergründen oder in Fällen, wo mehr als 256 Farben für eine befriedigende Darstellung notwendig sind, ist eine andere Lösung gefragt.

Microsofts Browser bietet seit Version 4 eine Vielzahl an Filtern, die diverse Bildmanipulationen und Animationen im Browser ermöglichen. Einer davon trägt die Bezeichnung AlphaImageLoader und bietet die Möglichkeit, PNGs mit Alpha-Transparenz korrekt darzustellen. Folgender Codeauszug zeigt, wie sich dieser Filter dazu verwenden lässt, das Hintergrundbild einer DIV-Box korrekt zu laden (Schmitt, Dominey, Li, Marcotte, Orchard, & Trammell, 2008, S. 216ff).

```
<div style="filter:progid:DXImageTransform.Microsoft.
      AlphaImageLoader(src='back.png',sizingMethod='scale');">
   ...
</div>
```

[8] http://ie7-js.googlecode.com/svn/test/index.html

Nachteilig in diesem Zusammenhang ist, dass diese Style-Angabe nicht standardkonform ist. Außerdem wird – insbesondere, wenn viele Bilder einer Seite auf diese Art eingebettet werden – die Zeit, die zum Laden und Darstellen benötigt wird, merkbar länger. Wichtig ist auch, diesen Code per Conditional Comment auszuliefern, da andere Browser als der Internet Explorer damit nichts anfangen können.

Viele JavaScript-Libraries wie beispielsweise jQuery (über ein Plugin[9]) oder das vorher angesprochene IE7.js enthalten bereits Methoden, die das Laden von PNG-Grafiken über den Internet Explorer Filter durchführen, sofern dies erforderlich ist. Oft ist es dabei notwendig, eine transparente Ersatzgrafik im GIF-Format einzubinden oder die Dateien nach einem bestimmten Schema zu benennen. Der große Vorteil bei der Verwendung von Libraries ist, dass sie sich schnell einbauen und einfach verwenden lassen.

7 Konkrete Umsetzung

Auf Basis der bisher theoretisch erläuterten Ansätze zur Lösung und Umgehung von Problemen widmet sich dieses Kapitel der praktischen Umsetzung. Dazu wurde eine in Entwicklung befindliche eLearning-Plattform analysiert und derart angepasst, dass sie in älteren Browsern einwandfrei verwendbar ist. Da das Projekt zum Zeitpunkt dieser Anpassungen bereits verhältnismäßig weit fortgeschritten war, musste mit erheblichem Aufwand kalkuliert werden.

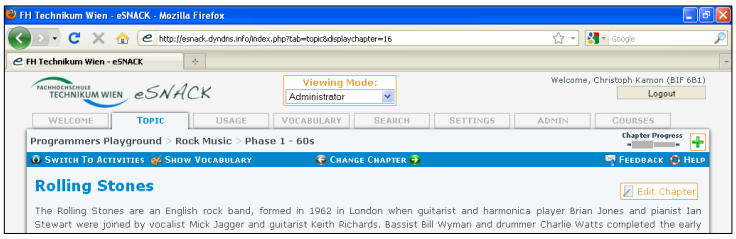

Abbildung 15: Ausgangssituation in Firefox 3.6

Schrittweise wurden zuerst die Ausgangssituation analysiert, Darstellungsprobleme behoben und anschließend Fehlerquellen seitens JavaScript ausgemerzt. Die detaillierte Durchführung dieser Schritte ist in den folgenden Kapiteln beschrieben und mit Screenshots veranschaulicht.

7.1 Ausgangssituation

Ursprünglich wurde davon ausgegangen, dass die Webplattform ausschließlich in modernen und standardkonformen Browsern funktionsfähig sein muss. Dementsprechend wurde ausschließlich mit den aktuellen Versionen von Internet Explorer, Firefox, Opera und Safari getestet. Abbildung 15 zeigt, wie die Seite in einem solchen Browser aussieht.

Als bereits etwa die Hälfte des Projekts umgesetzt war, kam seitens des Kunden die Anforderung, dass auch ältere Browser mit der Webseite zurechtkommen müssen. Begründet wurde das damit, dass die User Agent Strings am Webserver des Kunden ausgewertet werden

[9] http://jquery.andreaseberhard.de/pngFix/

und noch ein erheblicher Anteil der Besucher einen älteren Browser verwendet. Tabelle 3 zeigt die exakten Zahlen, gruppiert nach Browserfamilie.

Internet Explorer			Firefox				Opera	Safari	Andere
6.0	7.0	8.0	2.0	3.0	3.5	3.6			
12%	11%	14%	1%	3%	22%	21%	3%	7%	6%

Tabelle 3: Browserverteilung am Webserver des Kunden

Aufgrund dieser Fakten wurde klar, dass es keine Option ist, Benutzer von Internet Explorer 6 auszuschließen. Ausschließen deshalb, weil die Webseite in ihrem Ausgangszustand aufgrund eines stark „zerstörten“ Layouts schlichtweg unbenutzbar war. In Internet Explorer 7 stellte sich die Situation deutlich besser dar, obwohl eine verschobene Navigationsleiste und andere Kleinigkeiten die Optik empfindlich störten.

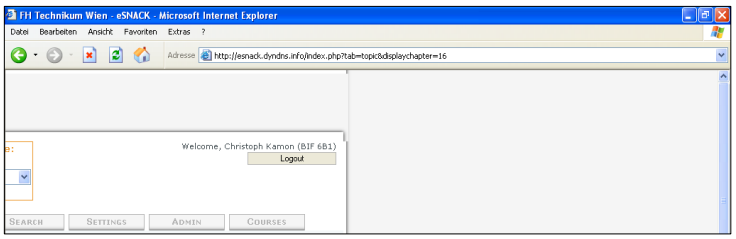

Abbildung 16: Ausgangssituation in Internet Explorer 6

Abbildung 16 zeigt das angesprochene „zerstörte“ Layout, das eine Bedienung der Seite weitgehend unmöglich machte.

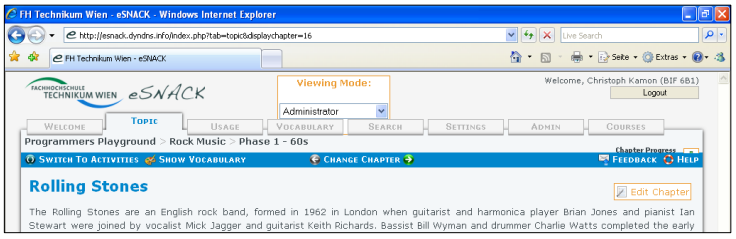

Abbildung 17: Ausgangssituation in Internet Explorer 7

In Abbildung 17 ist die Situation in Internet Explorer 7 zu sehen. Die Anzeige für den „Chapter Progress“ rechts oben ist, ebenso wie die Navigationsleiste, verschoben. Auch die orange Box oben in der Mitte ist deutlich zu hoch und am unteren Ende abgeschnitten.

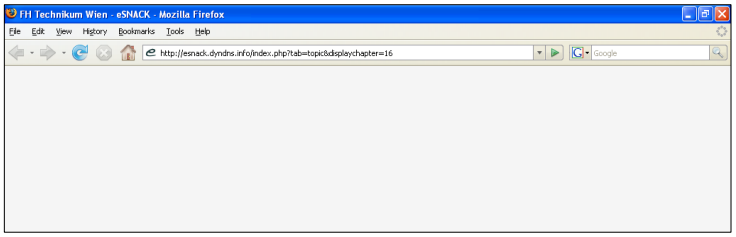

Abbildung 18: Ausgangssituation in Firefox 2.0

Noch gravierender gestaltete sich die Situation im Firefox 2.0. Aufgrund zahlreicher nicht unterstützter CSS-Formatierungen zeigte der Browser schlichtweg nichts außer der Hintergrundfarbe an (siehe Abbildung 18).

Dadurch, dass das Layout in zwei der Fälle sehr stark beeinträchtig war, konnten hier noch keine Aussagen zu anderen Problemen getroffen werden. Über weitere Probleme betreffend Seiteninhalte und JavaScripts konnten somit lediglich Mutmaßungen angestellt werden.

7.2 Darstellungsprobleme

Während der Projektdurchführung wurde das Seitenlayout einmal grundlegend verändert. Das heißt, dass von einem gewöhnlichen Layout, bei dem die gesamte Seite gescrollt wird, auf ein fixiertes Layout umgestellt wurde, bei dem nur noch der Hauptinhalt gescrollt wird – Navigation und dergleichen verbleiben an ihrer Position. Da bei diesem Umbau möglichst viele Strukturen respektive DIV-Boxen beibehalten werden sollten, fanden sich zahlreiche Container im Layout, die eigentlich ohne Funktion waren. Außerdem waren diese teilweise übermäßig ineinander verschachtelt.

Im ersten Schritt stand eine „Entwirrung“ des Layouts auf dem Plan. Alle fix positionierten DIVs wurden ohne Verschachtelung direkt im `body` platziert und positioniert. Die Anzahl der für das Layout verantwortlichen Container konnte so auf etwa die Hälfte reduziert werden. Durch Wegfall der gegenseitigen Beeinflussung verschachtelter Elemente konnten potentielle Fehlerquellen auf ein Minimum begrenzt werden.

Direkt nach Durchführung dieser ersten Maßnahme war das Layout bereits in allen drei gezeigten Problembrowsern deutlich besser. Um den Seiteninhalt in einem DIV-Element scrollen zu lassen, ist es notwendig `position: fixed` zu setzen und `overflow` zu aktivieren. Da Internet Explorer 6 das von Haus aus nicht beherrscht, wurde diese Funktionalität durch Einbindung von IE7.js nachgerüstet.

Im nächsten Schritt wurden `margin` und `padding` des Default Stylesheets für alle Elemente überschrieben. Wie in Kapitel 6.1.1 beschrieben, ist dies oft eine Ursache für Inkonsistenzen bei der Darstellung. Die wenigen PNG-Bilder mit Alpha-Transparenz – welche ausnahmslos auf einfarbigem Hintergrund dargestellt werden – wurden, um Probleme im Zusammenhang mit Internet Explorer 6 zu umgehen, durch Versionen mit 256 Farben und binärer Transparenz ersetzt. Da es sich nur um kleine Icons handelt, musste dabei kein Qualitätsverlust in Kauf genommen werden.

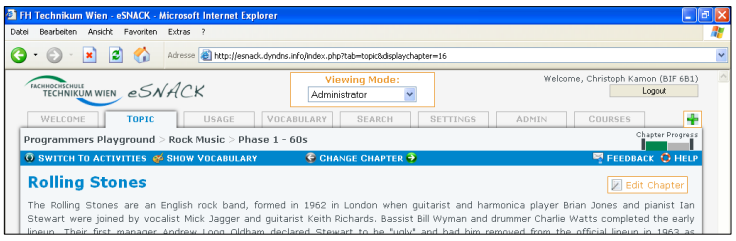

Abbildung 19: Korrigierte Darstellung in Internet Explorer 6

Das Seitenlayout an sich konnte zu diesem Zeitpunkt bereits als konsistent bezeichnet werden. Abbildung 19 zeigt die Darstellung in Internet Explorer 6, nach Durchführung der beschriebenen Schritte. Desweiteren sind hier bereits die Icons durch Versionen ohne Alpha-

Transparenz ersetzt. Für die Fußzeile (nicht am Screenshot zu sehen) wurde der Star-HTML-Hack eingesetzt, um deren Position zu korrigieren. Abgesehen davon, kommt die Webseite ohne CSS-Hacks und Browserweichen aus.

Abbildung 20: Statusbalken in unterschiedlichen Browsern (rechts unten = Referenz)

Da die Unterstützung für **`display: inline-block`** äußerst unterschiedlich ausfällt, wurden solche Elemente im nächsten Schritt durch Grafiken ersetzt. Da es sich dabei um verhältnismäßig einfache und gut komprimierbare Bilder handelt, wirkt sich das nur marginal auf die an den Besucher zu übertragende Datenmenge aus. Abbildung 20 zeigt die unterschiedlichen Darstellungen – je nach Browser – vor dieser Maßnahme. Die „Chapter Progress" Anzeige (rechts oben im Hauptbereich der Webseite) wurde ebenfalls korrigiert, indem Grafiken anstelle von Blockelementen eingesetzt wurden.

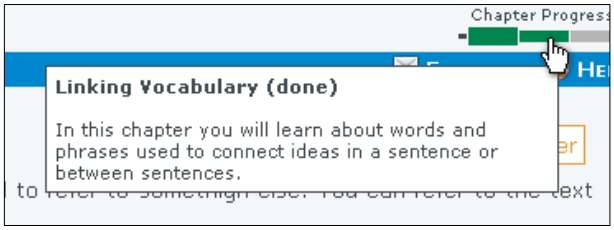

Abbildung 21: JavaScript / CSS Quick Info

Als letztes gravierendes Problem erwiesen sich „Quick Infos", die mittels CSS-Pseudoklasse **`:hover`** eingeblendet wurden. Weder in Internet Explorer 6 noch 7 war die Darstellung dieser zufriedenstellend. Da es sich dabei um ein Feature handelt, das vom älteren der beiden Browser überhaupt nicht unterstützt wird, war ein gänzlich anderer Ansatz notwendig. Dazu wurde auf eine JavaScript-basierte Lösung[10] zur Einblendung solcher Quick Infos gesetzt, die in allen Browsern zufriedenstellend funktioniert.

Abbildung 21 zeigt exemplarisch, wie die beschriebene Quick Info aussieht. Durch entsprechendes Anpassen der Stylesheets wurde erreicht, dass CSS- und JavaScript-Lösung optisch nicht zu unterscheiden sind.

7.3 Probleme mit JavaScript

Nachdem Layout und Benutzbarkeit der Webplattform wieder hergestellt waren, wurde der Fokus bei der Fehlerbehebung auf die verwendeten JavaScript-Komponenten gerichtet. Durch ausgiebiges Testen konnten dabei insgesamt drei größere Probleme diagnostiziert werden, die es zu beheben galt.

[10] http://www.dynamicdrive.com/dynamicindex5/dhtmltooltip.htm

Obwohl die Webplattform zwischenzeitlich teilweise auf jQuery setzt, war sie ursprünglich nicht dafür konzipiert. Aus diesem Grund wird – anstatt die Elemente via jQuery zu selektieren – an zahlreichen Stellen die JavaScript Funktion `getElementsByName()` aufgerufen. Das Problem dabei ist, dass Internet Explorer (bis einschließlich Version 8) diese Funktion fehlerhaft implementiert hat und statt über den Namen trotzdem über die ID selektiert. Somit funktionieren Scripts nicht korrekt, die auf diese Funktion angewiesen sind.

Sämtliche Funktionsaufrufe im Projekt zu suchen und auf `getElementById()` umzubauen hätte viel Zeit gekostet. Deswegen wurde für Internet Explorer Browser die fehlerhafte Funktion mit folgendem Code überschrieben:

```
document.getElementsByName = function(name, tag) {
    if(!tag) tag = '*';
    var elems = document.getElementsByTagName(tag);
    var res = [];

    for(var i=0; i<elems.length; i++) {
        att = elems[i].getAttribute('name');

        if(att == name)
            res.push(elems[i]);
    }
    return res;
}
```

Zu beachten ist dabei, dass die Performance dieser selbst geschriebenen Funktionen im Allgemeinen schlechter ist, als von jenen, die direkt vom Browser zur Verfügung gestellt werden. Dies ist bei modernen Rechnern oft zu vernachlässigen, sollte aber dennoch bedacht werden, falls die Funktion sehr oft aufgerufen wird.

Das zweite große Problem wurde durch einige kleinere Fehler in JavaScripts verursacht. Die meisten Browser geben zwar beim Auftreten von Fehlern eine Meldung in der Fehlerkonsole aus, führen aber den Rest des Scripts trotzdem aus. Anders ist die Situation beim Internet Explorer, der nach dem Auftreten eines Fehlers die Ausführung des restlichen Scripts verweigert. Zwar lässt sich hier streiten, welcher Ansatz („Komfort trotz unsauberer Programmierung“ oder „erzwungener sauberer Code“) der Bessere ist, aber Rücksicht darauf muss in jedem Fall genommen werden.

Um die Problematik zu beseitigen, empfiehlt es sich, alles auf `null` zu überprüfen, das potentiell diesen Wert zurück liefern könnte. Außerdem sollten heikle Codepassagen in try-catch-Blöcken ausgeführt werden, um auf Fehler entsprechend reagieren zu können. Speziell durch Letzteres konnten so gut wie alle Problemstellen beseitigt werden.

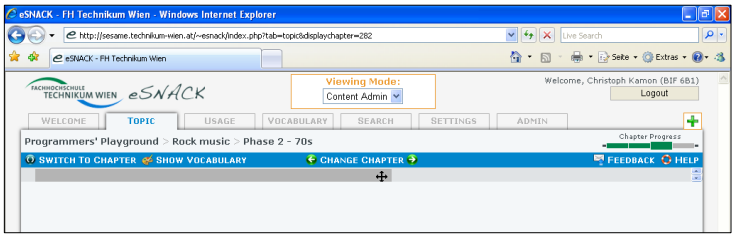

Abbildung 22: Kollabierter DIV in Internet Explorer 7

Ein weiteres auftretendes Problem, das allerdings ausschließlich Internet Explorer 7 betrifft, trat durch die Kombination aus einem DIV-Element mit variabler Höhe und einer jQuery Draggable List auf. Letztere ermöglicht es, die Reihenfolge der Listeneinträge innerhalb eines `<ul>` (Unordered List) per Drag & Drop zu verändern. Wird diese Sortierung innerhalb des angesprochenen DIVs durchgeführt, „kollabiert“ der DIV auf eine Höhe von wenigen Pixel und lässt sich nicht wieder vergrößern. Abbildung 22 zeigt das Problem.

Zur Lösung wurde per Conditional Comment der Browser abgefragt und, für den Fall, dass eine Draggable List initialisiert wird, die Höhe des DIVs per JavaScript fixiert – zu sehen im nachfolgenden Codeausschnitt.

```
toBeFixed = document.getElementById('funquiz');
toBeFixed.style.height = ($(document).height() - 173) + 'px';
```

Obwohl dies die Einschränkung mit sich bringt, dass sich bis zum neu Laden der Seite der DIV nicht mehr dynamisch an Größenänderungen des Browsers anpassen kann, überwiegt der Vorteil der sichergestellten Verwendbarkeit.

7.4 Endergebnis

Nach Durchführung aller Korrekturmaßnahmen lässt sich die Webplattform mit Internet Explorer 6, Internet Explorer 7 und Firefox 2.0 genauso gut verwenden, wie mit modernen und standardkonformen Browsern. In allen drei Browsern deckt sich das Seitenlayout exakt mit der Referenz. Desweiteren wurden sämtliche JavaScripts funktionsfähig gemacht sowie insgesamt eine gute Bedienung sicher gestellt.

Nicht weiter beachtet wurde der Administrationsbereich der Webseite. Der Grund hierfür ist, dass nur eine sehr kleine Anzahl an Benutzern sich den administrativen Aufgaben widmen wird. Von diesen kann – im Gegensatz zu gewöhnlichen Besuchern der Webseite – verlangt werden, dass sie einen modernen Browser verwenden. Für den Notfall funktioniert zwar auch der Administrationsbereich in den „Problembrowsern“, aber es ist mit kleineren Problemen bei der Darstellung zu rechnen.

Eine weitere Einschränkung betrifft Benutzer des Internet Explorer 7. Falls eine per Drag & Drop veränderbare Liste in der aktuellen Ansicht vorkommt, skaliert der Hauptbereich der Seite nicht mehr dynamisch mit dem Browserfenster mit. Aufgrund der Tatsache, dass Internet Explorer 7 eine rückläufige Verbreitung hat, wurde dies – auch vom Kunden – in Kauf genommen.

8 Diskussion

Zum Abschluss wird erneut auf die zu Beginn formulierten Ziele und die aus dieser Arbeit gewonnenen Erkenntnisse eingegangen. Ergänzt wird das ganze um einige persönliche Eindrücke, die während des Verfassens gewonnen werden konnten.

8.1 Ergebnisse

Als allgemeine Erkenntnis lässt sich festhalten, dass es auch im Bereich der Webentwicklung äußerst wichtig ist, vor Beginn der Projektumsetzung ausreichend zu spezifizieren. Im

Konkreten bedeutet das, dass die zu unterstützenden Browser von Anfang an festgelegt sein sollten. Neben der Tatsache, dass sich unerwartete Probleme bei der Projektabnahme so vermeiden lassen, bringt diese Vorgehensweise weitere Vorteile mit sich.

Ist die Browserauswahl fixiert, ist es möglich, einen „kleinsten gemeinsamen Nenner“ zu finden. Das heißt, dass versucht wird herauszufinden, welche Funktionen von möglichst allen dieser Browser problemlos unterstützt werden. Durch Umgehen von nicht unterstützten Funktionalitäten wird der Anpassungsaufwand für einzelne Browser möglichst gering gehalten. Außerdem ermöglicht die festgelegte Browserauswahl kontinuierliches Testen während der gesamten Umsetzung. Probleme können so in kleinen Schritten identifiziert und behoben werden. Andernfalls besteht das Risiko, dass auf einen Schlag eine unüberschaubare Menge an Korrekturen vorzunehmen ist.

Die Unterstützung von CSS und HTML ist insbesondere beim Internet Explorer 6 als schlecht zu bezeichnen. Obwohl der Browser zum Zeitpunkt seines Erscheinens mit einer verhältnismäßig guten CSS-Unterstützung aufwarten konnte, führen zahlreiche Bugs und sein hohes Alter immer wieder zu Problemen. Wenngleich zahlreiche Bugs mit Internet Explorer 7 behoben wurden, fehlen auch dieser Version einige wichtige CSS-Features.

Firefox 2 wurde als W3C-konformer Browser konzipiert und rendert im Allgemeinen korrekt. Sein Problem sind insbesondere nicht unterstützte CSS-Attribute, wodurch die Darstellung von jener moderner Browser abweichen kann. Da es sich bei HTML5 und CSS3 um verhältnismäßig junge Technologien handelt, bieten alle drei Browser hier schlechte oder gar keine Unterstützung.

Zur Behebung von CSS- und HTML-bezogenen Problemen empfiehlt sich der Verzicht auf schlecht unterstützte Features im Vorfeld. Ist dies nicht möglich, lässt sich oft mittels Browserweichen, CSS-Hacks und JavaScript-Lösungen eine befriedigende Darstellung – auch in älteren Browsern – erreichen.

Bei JavaScript ist die Erkenntnis wichtig, dass es dabei hier um eine „inkonsistente Technologie“ handelt. Zwar stehen die meisten gängigen Funktionen in allen Browsern zur Verfügung, aber Garantie dafür gibt es keine. Der Grund dafür ist, dass zwar die Syntax normiert ist, nicht aber die zur Verfügung stehenden Funktionen. Somit obliegt es dem Browserhersteller, was letztendlich zur Verfügung steht.

Probleme sind außerdem beim DOM zu erwarten, da dieses unterschiedlich gut implementiert ist. Während manche Browserhersteller bereits die Unterstützung für DOM Level 3 ausbauen, wird selbst von Internet Explorer 8 nur DOM Level 1 (unvollständig) unterstützt. Um diese Inhomogenität von JavaScript auszugleichen, empfiehlt sich der Einsatz einer JavaScript-Library wie beispielsweise Dojo oder jQuery. Diese ermöglichen eine für den Entwickler transparente Verwendung von Funktionalitäten und bilden gegebenenfalls Funktionen nach, die vom Browser eigentlich gar nicht zur Verfügung gestellt werden.

Ein spezielles Problem von Internet Explorer 6 tritt bei der Verarbeitung von PNG-Bildern auf, die über einen Alpha-Kanal verfügen – transparente Bereiche werden schlichtweg grau gefüllt. Umgehen lässt sich das Problem im Idealfall durch Verzicht auf solche Grafiken. Alternativ stehen in den Microsoft-Browsern zahlreiche Filter bereit, wovon auch einer zum korrekten Laden der PNG-Grafiken benutzt werden kann. Ein erhöhter

Implementierungsaufwand, sowie eine längere Ladedauer der fertigen Webseite, ist bei Einsatz dieser Lösung zu bedenken.

Insgesamt ist es als Ziel zu sehen, dass eine Webseite in möglichst allen verbreiteten beziehungsweise gewünschten Browsern ordnungsgemäß angezeigt wird. Dabei sollte der Fokus allerdings nicht auf einer pixelgenauen Übereinstimmung liegen, sondern vielmehr auf ansehnlicher Darstellung und einwandfreier Funktionalität.

Aktuelle Sorgenkinder der Webentwickler sind zweifelsohne Internet Explorer 6 und 7, sowie in manchen Belangen Firefox 2. Die Verbreitung dieser Browser wird in den nächsten Jahren ohne Frage zurück gehen. Auch versuchen die meisten Browserhersteller, sich immer besser an gängige Webstandards zu halten. Trotzdem werden die in dieser Arbeit beschriebenen Probleme weiter existieren – die Browser und problematischen Technologien werden in Zukunft lediglich andere sein.

8.2 Persönliche Stellungnahme

Während der Erstellung dieser Arbeit konnte ich viel neues Wissen erlangen. Auch meine Meinung bezüglich Internet Explorer 6 hat sich dabei ein wenig verändert. Ursprünglich war ich der Auffassung, dass dessen CSS-Unterstützung sehr schlecht ist. In Wirklichkeit war der Browser aber zu seiner Erscheinung diesbezüglich recht fortschrittlich und unterstützte viele Features. Verglichen mit der damaligen Konkurrenz konnte er sich durchaus als guter Browser behaupten. Das Hauptproblem, das der Browser heute hat, ist – neben einigen Bugs – hauptsächlich sein Alter.

Weit schlimmer finde ich persönlich, dass die DOM-Unterstützung im Internet Explorer sehr schlecht ausfällt. Immerhin sind für Version 9 diesbezüglich Verbesserungen angekündigt. Aufgrund von Browsertests wie Acid2/3 und dergleichen streben alle Browserhersteller danach, sich bestmöglich an Webstandards zu halten und diese Tests zu bestehen. Auch wenn HTML5 und CSS3 teilweise noch in den Kinderschuhen stecken, ist wenigstens für die alten „Basistechnologien“ mit einer immer konsistenteren Unterstützung zu rechnen.

Da ich von einigen Leuten gefragt wurde, wieso ich zwar über Acid2, nicht aber über Acid3 geschrieben habe, möchte ich dies kurz erklären. Acid2 verwendete eine Auswahl an häufig verwendeten CSS-Funktionen und kann meiner Meinung nach als recht praxisgerecht angesehen werden. Außerdem testet er wirklich mit Schwerpunkt auf CSS.

Beim Acid3-Test ist die Situation eine völlig andere. Neben JavaScript und DOM werden auch noch unzählige andere Browserfunktionen, wie beispielsweise die Darstellung von Vektorgrafiken, getestet. Obendrein steht Acid3 in der Kritik[11], nicht annähernd praxisgerecht zu sein sondern sich vielmehr auf ausgewählte und weniger sinnvolle Features zu beschränken. Aufgrund dieses Umstandes und der Tatsache, dass er eben weit mehr als JavaScript testet, habe ich darauf verzichtet, ihn für den JavaScript-Teil dieser Arbeit zu verwenden.

[11] http://meyerweb.com/eric/thoughts/2008/03/27/acid-redux/

Literatur- & Internetquellen

Andrew, R. (2009). *The CSS Anthology: 101 Essential Tips, Tricks & Hacks (3rd Edition).* Collingwood, Australia: SitePoint.

Bongard, D. (14. 5 2007). *Browserunterscheidung mit CSS-Hacks*. Abgerufen am 10. 5 2010 von http://www.bongard.net/blog/2007/05/14/css_stern-html-hack/

Ducket, J. (2008). *Beginning Web Programming with HTML, XHTML, and CSS (Second Edition).* Indianapolis, USA: Wiley Publishing.

Edwards, D. (6. 3 2009). *ie7-js*. Abgerufen am 10. 5 2010 von http://code.google.com/p/ie7-js/

Edwards, J., & Adams, C. (2006). *The JavaScript Anthology: 101 Essential Tips, Tricks & Hacks.* Collingwood, Australia: SitePoint.

Google. (2010). *Google Chrome FAQ for web developers*. Abgerufen am 14. 4 2010 von http://www.google.com/chrome/intl/en/webmasters-faq.html#insidechrome

Hachamovitch, D. (18. 3 2010). *IEBlog*. Abgerufen am 28. 4 2010 von http://blogs.msdn.com/ie/archive/2010/03/16/html5-hardware-accelerated-first-ie9-platform-preview-available-for-developers.aspx

hasLayout.net. (19. 7 2009). *Staircase Bug - hasLayout.net*. Abgerufen am 27. 4 2010 von http://haslayout.net/css/Staircase-Bug

Huddleston, R. (2008). *HTML, XHTML, and CSS: Your visual blueprint for designing effective Web pages.* Indianapolis, USA: Wiley Publishing.

Koch, P.-P. (2009). *Quirksmode.org*. Abgerufen am 27. 4 2010 von http://www.quirksmode.org/css/display.html

Lowery, J. (2005). *CSS Hacks and Filters: Making Cascading Style Sheets Work.* Indianapolis, USA: Wiley Publishing.

Mielke, M., & Massy, D. (26. 9 2006). *Cascading Style Sheet Compatibility in Internet Explorer 7*. Abgerufen am 26. 4 2010 von http://msdn.microsoft.com/en-us/library/bb250496%28VS.85%29.aspx

Powers, D. (2009). *Getting StartED with CSS.* New York, USA: Springer-Verlag.

Roelofs, G. (14. 3 2009). *Portable Network Graphics*. Abgerufen am 28. 4 2010 von http://www.libpng.org/pub/png/#history

Schmitt, C. (2010). *CSS Cookbook: Third Edition.* Sebastopol, USA: O'Reilly Media.

Schmitt, C., Dominey, T., Li, C., Marcotte, E., Orchard, D., & Trammell, M. (2008). *Professional CSS: Cascading Style Sheets for Web Design (Second Edition).* Indianapolis, USA: Wiley Publishing.

Storey, D. (22. 1 2007). *Upcoming CSS3 support in Opera*. Abgerufen am 16. 4 2010 von http://my.opera.com/dstorey/blog/show.dml/701902

Teague, J. C. (2009). *Speaking in Styles: Fundamentals of CSS for Web Designers.* Berkeley, USA: New Riders.

The StyleWorks. (kein Datum). *Internet Explorer 7 — Hacks und Neuerungen*. Abgerufen am 26. 4 2010 von http://www.thestyleworks.de/tut-art/ie7.shtml

The Web Standards Project. (21. 7 2006). *Acid2: The Guided Tour*. Abgerufen am 18. 4 2010 von http://www.webstandards.org/action/acid2/guide/

World Wide Web Consortium. (8. 9 2009). *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*. Abgerufen am 16. 4 2010 von http://www.w3.org/TR/CSS2/

World Wide Web Consortium. (24. 3 2010). *CSS: Current Work*. Abgerufen am 16. 4 2010 von http://www.w3.org/Style/CSS/current-work#CSS3

Zakas, N. C. (2009). *Professional JavaScript for Web Developers (2nd Edition).* Indianapolis, USA: Wiley Publishing.

Zeldman, J., & Marcotte, E. (2010). *Designing with Web Standards (Third Edition).* Berkeley, USA: New Riders.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
CERN	European Organization for Nuclear Research
CSS	Cascading Style Sheets
DTD	Document Type Definition
ECMA	European Computer Manufacturers Association
GIF	Graphics Interchange Format
HTML	HyperText Markup Language
JPEG	Joint Photographic Experts Group
MIME	Multipurpose Internet Mail Extension
PHP	PHP Hypertext Preprocessor (rekursives Acronym!)
PNG	Portable Network Graphics
ppi	Points per Inch
px	Pixel
SGML	Standard Generalized Markup Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WWW	World Wide Web
XHTML	Extensible Hypetext Markup Language
XML	Extensible Markup Language