Universität Hamburg
Fachbereich Informatik

B A C H E L O R A R B E I T

Plattformunabhängige Entwicklung mobiler Anwendungen mit Web-Technologien

vorgelegt von

David M. Lammerz
geb. 2. Oktober 1987 in Hamburg
Matrikelnummer: 6053363

eingereicht am 24. September 2012

Betreuer:
Dr. Guido Gryczan,
Dr. Axel Schmolitzky

Erklärung

Hamburg, 24. September 2012

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen - benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht. Ich stimme der Veröffentlichung dieser Arbeit in den Bestand der Bilbiothek des Fachbereichs Informatik in nicht digitaler Form zu.

_____

David M. Lammerz

Danksagung

An dieser Stelle möchte ich mich zuerst bei Dr. Guido Gryzan für die Erstbetreuung und Dr. Axel Schmolitzky für die Zweitbetreuung dieser Arbeit bedanken. Besonderer Dank gilt auch den Mitarbeitern des von dem Unternehmen C1WPS durchgeführten Delegs-Projekts und last but not least meiner Familie und Freunden für das Feedback und die Unterstützung.

Inhaltsverzeichnis

Einleitung

1.1 Motivation

Mit der zunehmenden Verbreitung von Smartphones steigt das Interesse bei Entwicklern und Firmen an mobilen Anwendungen mit dem Ziel, ihre Produkte durch eine mobile Anwendung zu verbessern und vorteilhafter zu vermarkten. Bei der Entwicklung solcher Anwendungen muss zuerst geklärt werden, für welche Plattform eine Anwendung entwickelt werden soll. Um einen größeren Kundenkreis auf dem stark im Wettbewerb stehendem und sich monatlich veränderndem Smartphone-Markt zu erreichen, ist es unumgänglich, für mehrere Plattformen zu entwickeln. Jede Plattform stellt dabei eigene Anforderungen an die Entwickler. Eine Möglichkeit, diesem Problem zu begegnen, kann nach dem Leitsatz „*Write once, run anywhere*“[1] die Nutzung von Webtechnologien sein. Ein solcher Ansatz ist im Rahmen des Möglichen, da die aktuellen Smartphones größtenteils leistungsstarke Web-Browser[2] besitzen, die mehr Schnittstellen zum Betriebssystem bieten.

Ein gutes Beispiel für eine Unternehmen, die stark auf Webanwendungen setzt, ist Google mit ihren eigenen Services wie Google Text&Tabellen oder Google Mail. Diese sind nativer Software oft gleichwertig und können in Googles eigenem Betriebssystem, dem Chrome OS, solche sogar ersetzen. Ein weiteres bekanntes Beispiel ist Windows 8. Für dieses können Anwendungen neben C# auch mit HTML und JavaScript geschrieben werden. Diese Beispiele lassen eine Entwicklung, die den Browser als neues Betriebsystem deklariert, sichtbar werden.

---

[1] Durch Sun Microsystem zur Veranschaulichung der Vorteile der Plattformunabhängigkeit von Java geprägter Ausspruch (siehe [Sun]).

[2] iOS, Android und BlackBerry (drei der größten Plattformen) nutzen WebKit als HTML-Rendering-Engine.

Die Entwicklung einer größeren Anwendung mit Webtechnologien begegnete mir erstmals im Rahmen eines Industriepraktikums bei der C1WPS Ende 2011. Dort konnte ich drei Monate an der Entwicklung des Delegs-Editors[3] mitwirken. Dieser Editor ist eine WYSIWYG-Webanwendung, welche es ermöglicht Dokumente in GebärdenSchrift (siehe *Abschnitt 2.2 GebärdenSchrift*) abzufassen und zu verwalten. Mein Interesse an Webtechnologien und der von den Nutzern geäußerte Wunsch nach einer mobilen Version motivierten mich im Rahmen der Entwicklung einer solche Anwendung das Thema der *„Plattformunabhängigen Entwicklung mobiler Anwendungen mit Web-Technologien“* zu behandeln.

1.2 Aufgabenstellung und Zielsetzung

Im Rahmen dieser Arbeit werden die verschiedenen Ansätze der plattformunabhängigen Anwendungsentwicklung für Smartphones aufgeschlüsselt und verglichen. Als praktisches Beispiel wird die Entwicklung eines mobilen Wörterbuches für die GebärdenSchrift (siehe 2.2 GebärdenSchrift) genutzt. Anhand der eigenen Erfahrungen bei der Entwicklung wird ein Vergleich der Gebrauchstauglichkeit der gängigen Konzepte zur Plattformunabhängigen Anwendungsentwicklung für mobile Plattformen mit Webtechnologien gezogen. Darauf aufbauend wird eine Übersicht über die zukünftige Entwicklung solcher Lösungen, als Motivation für weitere Arbeiten zu diesem Thema, gegeben.

1.3 Aufbau der Arbeit

Kapitel 1 stellt die Arbeit in Grundzügen vor, dabei wird auf die Motivation, Aufgabenstellung und Zielsetzung eingegangen.

Im zweiten Kapitel werden die Grundlagen, auf denen diese Ausarbeitung aufbaut, vorgestellt. Hierfür werden der Begriff der App definiert und die heutigen Anforderungen an Apps beschrieben. Der Abschnitt über die verschiedenen Plattformen soll verdeutlichen, warum das Konzept der plattformunabhängigen Anwendungsentwicklung für mobile Plattformen interessant ist. Darauf folgt eine Übersicht über die verschiedenen Vertriebsmöglichkeiten für Apps. Der Abschnitt über die GebärdenSchrift dient zum Verständnis des späteren Abschnittes zum Anwendungsbeispiel, beschreibt die Grundlagen der GebärdenSchrift und wofür diese eingesetzt wird. Abschließend wird auf die für diese Arbeit wichtigen Webtechnologien eingegangen.

---

3 Zu finden unter http://www.delegs.com/delegseditor/ (geprüft 31.07.2012).

Das dritte Kapitel befasst sich mit einer Gegenüberstellung von WebApps, Wrapper-Hybriden und Interpreter-Hybriden. Mit der Vorstellung der Konzepte sollen die jeweiligen Einsatzmöglichkeiten und die dem schrittweisen Annähern an die nativen Anwendungen zugrunde liegende Logik verdeutlicht werden.

Im vierten Kapitel werden die genutzten Technologien bei der Entwicklung einer Web-App am Beispiel des GebärdenSchrift-Wörterbuchs beschrieben. Es wird genauer auf Anforderungen und Hindernisse eingegangen, die durch die Verwendung von Webtechnologien auf mobilen Geräten entstehen.

Im fünften und letzten Kapitel wird das durch das Anwendungsbeispiel erworbene Wissen auf der Basis der in den vorherigen Kapiteln erarbeiteten Informationen analysiert und ein Fazit zu den genutzten Technologien gezogen. Dafür werden Kriterien definiert anhand derer die Unterschiede der verschiedenen Technologien verdeutlicht werden. Des Weiteren werden die jeweiligen Einsatzmöglichkeiten dieser Technologien dargelegt.

2

Grundlagen

Dieses Kapitel erläutert das Basiswissen, welches zum Verständnis der darauf folgenden Kapitel benötigt wird. *Abschnitt 2.1* behandelt das Thema der mobilen Anwendungen oder auch Apps. Hier wird der Begriff definiert und auf die verschiedenen Plattformen und Vertriebsmöglichkeiten eingegangen. *Abschnitt 2.2* beschäftigt sich mit Web-Technologien und bietet dafür einen kleinen Überblick. Der letzte Abschnitt 2.3 ist eine kurze Einführung in die GebärdenSchrift. Dieses Wissen ist notwendig, um das Anwendungsbeispiel zu verstehen. Den Abschluss bildet Abschnitt 2.4 mit einem Überblick über Webtechnologien, die beim Thema WebApps zur Anwendung kommen.

2.1 Apps

Der Begriff *App* ist alles andere als eine präzise Beschreibung. Zur Klärung soll der folgende Abschnitt eine für diese Arbeit gültige Definition liefern. Ausserdem werden die an Apps gestellten Anforderungen bestimmt und auf die Verbreitung solcher mobilen Anwendungen eingegangen.

2.1.1 Begriffsklärung

Der Begriff App ist die Kurzform für *Application*, (deutsch: Anwendung). Bezeichnet werden damit heutzutage meist Anwendungen, die für Smartphones entwickelt wurden. Dabei wird der Begriff größtenteils in einen Zusammenhang mit Anwendungen gebracht, die über Distributions-Plattformen, wie z.B. dem Apple Appstore oder Android Market, verkauft werden bzw. sich auf den entsprechenden Smartphones installieren lassen. Für diese Ausarbeitung wird eine App als eine Anwendung verstanden, welche auf mobilen Endgeräten wie Smartphones oder Tablets lauffähig ist. Eine weiter Unterteilung wird durch die Begriffe *Native App*, *Web-App* und *Hybrid-App* vollzogen. Von einer *native App* oder auch nativen Anwendung wird dann gesprochen, wenn diese plattformabhängig ist und eine dieser Plattform eigenen Programmiersprache genutzt wird. Die Begriffe *Web-App* und *Hybrid-App* werden in *Abschnitt 3* definiert und weitergehend beschrieben.

2.1.2 Anforderungen an Apps

Neue Plattformen bringen oft neue Bedienkonzepte mit sich. Ein Bedienkonzept, das stark mit den heutigen Smartphones in Verbindung gebracht wird, ist der *Touch* (deutsch: Berührung). Dabei interagiert der Benutzer nicht mehr indirekt mit Objekten, zum Beispiel mittels einer Maus, sondern direkt mit seinen Fingern (vgl. hierzu [Oberquelle2011]). Mit der Möglichkeit, mehrere Berührungspunkte zu erkennen und zu verarbeiten, entstehen neue Systeme zur Gestenerkennung. Mit dem Einzug des *Multi-Touch* wurde zum Beispiel das Größer- und Kleinerziehen von Bildern mit zwei Fingern und der entsprechenden Spreiz- bzw. Kneifgeste sehr beliebt und ist heutzutage auf fast jedem Smartphone als Funktionalität zu finden.

Eine Änderung des Bedienkonzepts bedarf natürlich einer Anpassung der Benutzeroberfläche. Es ist meist nicht praktikabel mit seinem Finger ein Bedienelement zu benutzen, welches für eine indirekte Interaktion entworfen wurde. Zu diesem Zweck gibt es von den jeweiligen Plattformen, so z.B. auch von den drei großen Plattformen iOS, Android oder Windows Phone, *User Interface Guidelines*[4], an denen sich Entwickler orientieren können. In diesen Leitlinien wird z.B festgelegt, wie groß ein Knopf mindestens sein muss, um mit dem Finger bedient werden zu können oder welche Abstände zu anderen Oberflächenelementen eingehalten werden müssen.

Neben den neuen Benutzeroberflächen ist die Performance – also die Leistungsfähigkeit bzw. Effizienz – ein wichtiges Kriterium bei der Entwicklung einer App. Zwar haben die heutigen Smartphones mit ihren Dual-Core-CPUs eine für ihre Größe beachtliche Rechenleistung, doch sind Stromverbrauch, Speicherverwaltung oder Netzwerklatenzen Faktoren, die nicht vernachlässigt werden dürfen. Besonders die Netzwerklatenzen spielen in den Mobilfunknetzen eine große Rolle. Für die Entwickler ist es wichtig, die Faktoren möglicher Probleme im Auge zu halten und, weil hier eine direkte Bearbeitung notwendig wird, gegebenenfalls mit den zu diesem Zweck von den jeweiligen Plattformen angebotenen Analyse- bzw. Optimierungswerkzeugen entgegenzuwirken.

Der Punkt der Optimierung im Hinblick auf die Anwendungs-Performance wird im *Abschnitt 4 - Anwendungsbeispiel* ausführlich behandelt.

---

[4] Apple, Google und Microsoft bieten für ihre Plattformen jeweils eigene *Interface Guide Lines* an. Diese beinhalten Beispiele und Regeln für die jeweilige Plattform. Nur bei Microsoft sind diese Regeln wirklich relevant für den Zulassungsprozess zur Vertriebsplattform.

2.1.3 Plattformen

Besondere Aufmerksamkeit erfordert bei der Anwendungs-Entwicklung für Smartphones das Thema der Plattformvielfalt. Nach den Zahlen der Firma Gartner, welche unter anderem die Marktanteile der einzelnen Plattformen weltweit ermittelt, gibt es derzeit drei große Plattformen. Zu diesen gehören Android (Google), iOS (Apple) und BlackBerry (RIM) (siehe [Gartner2012]).

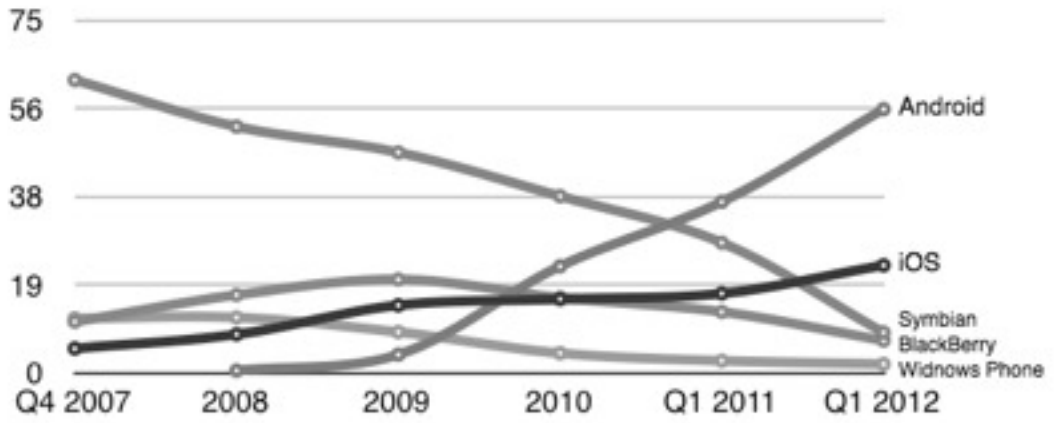

Abbildung 1: Visualisierung des Gartner Worldwide Sales of Mobile Phones Report Q1 2012 (siehe [GARTNER12])

Besonders auffällig ist das Absinken der Plattform Symbian seit Einführung des iPhones im Jahr 2007. Um Apple und Google etwas entgegensetzen zu können, verließ Nokia seine eigene Plattform (Symbian) und kooperierte seit Anfang 2011 mit Microsoft, indem sie Windows Phone als Hauptbetriebsystem für ihre Smartphones einsetzten (siehe [Reuters2011]). BlackBerry besitzt nach den Angaben des Gartner Report von Q1 2012 (siehe [Gartner2012]) noch einen Marktanteil von 6,9%. Ihr Kundenkreis besteht hauptsächlich aus Firmen.

Im folgenden Text werden die folgenden drei Plattformen näher vorgestellt:

- iOS (Apple)
- Android (Google)
- Windows Phone (Microsoft)

Dabei wird auf die Eigenschaften der einzelnen Plattformen und die Möglichkeiten der Entwicklung für diese eingegangen.

Die Plattform **iOS** startete 2007 mit der Einführung des iPhones. Zu diesem Zeitpunkt hieß diese noch iPhone OS und gestattete nur die Entwicklung von WebApps. Erst nach größeren Protesten der Mac-OS-X-Entwickler veröffentlichte Apple etwa ein Jahr später, also 2008, das SDK zur Entwicklung von nativen Anwendungen inklusive der Möglichkeit des Verkaufs über den App Store. Durch die Einführung des iPads[5] wurde iPhone OS zu iOS und somit das Betriebsystem für die mobilen Touchgeräte von Apple. Zur Entwicklung für iOS bedarf es in jedem Fall einer aktuellen Mac-OS-X-Installation, die wiederum einen Mac (Hardware)[6] benötigt. Die Programmiersprache Objective-C mitsamt seines Framework Cocoa mit der Entwicklungsumgebung Xcode muss genutzt werden. Um die Anwendung auf dem iPhone sowie auf dem iPad testen und im App Store veröffentlichen zu können, muss eine aktive Entwickler-Lizenz für 99$/Jahr erworben werden. Eine ausführliche Auseinandersetzung mit den Regeln zur Veröffentlichung im Apple App Store wird empfohlen, um einer Ablehnung im Zulassungsprozess zu entgehen (vgl. hierzu [MarkusStäuble2001], S. 308, Kapitel Distribution).

Android wurde Ende 2007 im Rahmen der Gründung der „Open Handset Alliance“ (ein Verbund von 86 Hardware- und Software-Firmen) vorgestellt (siehe [OHA]). Es handelt sich dabei um eine auf Linux basierende Software-Plattform. Entwickelt wird mit Hilfe von Java. Dabei ist es möglich einzelne Module in C zu programmieren. Ausgeführt wird der Code in der DalvikVM, einer selbst entwickelten und auf Android ausgerichteten JavaVM. Android versucht sich von seinen Mitbewerbern durch besondere Offenheit abzuheben. Da Hardwarehersteller Android kostenlos[7] nutzen können, bekommt dieses Betriebssystem eine große Verbreitung. Auch kann Android bei den günstigeren Smartphones große Umsätze verzeichnen. (vgl. hierzu [RetoMeier], S.19). Derzeit hat Android einen Marktanteil von 56,1% (siehe [Gartner2012]). Android läuft wie iOS auf Smartphones und Tablets[8]. Durch die Verwendung von Java als Programmiersprache gibt es weniger Vorgaben für die Anwendungsentwicklung. Es existiert ein Plugin für Eclipse, welches den Android-Emulator integriert und das Debugging erleichtert. Darüber hinaus gibt es noch einige Alternativen, mit denen für Android entwickelt wer-

---

[5] Tablet mit Multi-Touch-Funktionalität und App-Store-Anbindung von Apple.

[6] Mac steht für Macintosh und bezeichnet meist Heimcomputer von der Firma Apple. Die Benutzung des Betriebssystem Mac OS X setzt einen dieser Heimcomputer voraus. Eine Virtualisierung ist nur auf Apple Hardware erlaubt und wird daher von keinem größeren Virtualisierungssoftware-Hersteller für Windows oder Linux angeboten.

[7] Durch Patente müssen dennoch Lizenzgebühren an z.B. Microsoft abgeführt werden (siehe [Heise2011]).

[8] Als Beispiel das Nexus 7 auf dem ein unverändertes Android läuft.

den kann[9]. Bei einer Veröffentlichung der Anwendung über die offizielle Vertriebsplattform bedarf es ihrer Signierung. Zu diesem Zweck wird eine vorherige Registrierung als Entwickler für 25$ vorausgesetzt, um die entsprechenden Zertifikate von Google zu erhalten. Einen expliziten Zulassungsprozess gibt es zur Zeit nicht (vgl. hierzu [RetoMeier2009], S.20).

Eine weitere Plattform, die für diese Ausarbeitung hinzugenommen wird, ist **Windows Phone** von Microsoft. Derzeit hat Windows Phone weltweit zwar nur einen Anteil von 1,9%, doch darf nicht außer Acht gelassen werden, dass es eine neue Plattform ist (vormals Windows Mobile), die sich somit erst etablieren muss. In Deutschland hat Windows Phone einen Marktanteil von 5,4% (siehe [ReutersKantar2012]) und in Frankreich und Italien zwischen 2-3% bei steigender Tendenz. Ein besonderes Merkmal der Windows-Phone-Plattform ist die Verknüpfung der verschiedenen sozialen Netzwerke sowie deren herausgehobene Darstellung auf der Metro-Oberfläche[10]. Die Anwendungsentwicklung für Windows Phone findet ausschließlich auf Windows mit Visual Studio als Entwicklungsumgebung statt. C# ist unter Verwendung von Silverlight die hier meistgenutzte Programmiersprache[11], wobei auch JavaScript und HTML genutzt und folglich Webtechnologien für mobile Anwendungen entwickelt werden können. Für die Entwicklung von Spielen wird das XNA-Framework verwendet, welches im Grunde ein Wrapper[12] um DirectX ist und auch für die XBox360-Spielkonsole von Microsoft zum Einsatz kommt. Zur Veröffentlichung im Windows Phone Market wird ein aktiver Entwickler-Account für 99$/Jahr benötigt. Wie bei iOS gibt es einige zulassungsrelevante Bedingungen, die genutzte Schnittstellen und auch die Oberflächengestaltung[13] betreffen, ein nicht optionaler Zulassungsprozess.

---

[9] Das Android SDK beinhaltet alle nötigen Java-Bibliotheken, Compiler und Debug-Werkzeuge wie die Android Debugging Bridge (ADB). Der Code kann in jeder Java-IDE geschrieben werden. Eclipse liefert durch die Plugin-Integration nur einen Komfortvorteil.

[10] Die Metro-Oberfläche bezeichnet eine auf Kacheln basierte Ansicht, auf der Anwendungen dem Benutzer aktuelle Informationen direkt anzeigen können. Die Kacheln verhalten sich wie Widgets.

[11] Jede Common Language Interface[6] (CLI) basierte Sprache kann für die Windows-Phone-Programmierung genutzt werden. Also C#, F#, VB.NET und C++/CLI.

[12] Wrapper bezeichnet eine Softwarekomponente, die eine andere Softwarekomponente umschliesst. Dies wird gerne genutzt, um eine neue Schnittstelle, die Vorteile gegenüber der vorherigen hat, zu bieten.

[13] Microsoft nimmt bezüglich der Gestaltung der Benutzeroberfläche stärkeren Einfluss. Durch die Einbindung in das Metro-Layout müssen viele Vorgaben erfüllt werden.

	iOS	Android	Windows Phone
Programmiersprache	Objective C	Java	C#, F#, VB.NET, C++/CLI, JS, HTML
Entwicklungsumgebung	Xcode	beliebig (meist Eclipse)	Visual Studio
SDK Plattformen	Mac OS X	Windows, Linux, Mac	Windows
Initialkosten	99$/Jahr	einmalig 25$ für Lizenz	99$/Jahr

Tabelle 1: Übersicht über Vorgaben bei iOS, Android, WindowsPhone (Quellen: [MarkusStäuble2001], [RetoMeier2009], [WindowsPhoneHubFAQ])

In Tabelle 1 werden die Entwicklungsmöglichkeiten für die drei oben beschriebenen Plattformen zusammengefasst. Es zeigt sich, dass Android dem Entwickler am wenigsten Vorgaben macht, wenn es um die Entwicklungsumgebung, das benötigte Betriebssystem und die Vertriebsplattform geht. Nur die Programmiersprache ist auf Java festgelegt (mit NDK kann auch C für performancekritische Anwendungsteile genutzt werden). Windows Phone lässt wiederum bei der Programmiersprache am meisten Spielraum, ist jedoch bei der Entwicklungsumgebung und dem Betriebssystem alternativlos. Die Entwicklung für iOS bietet den geringsten Spielraum. Die Entwicklungsumgebung Xcode ist immer involviert (Testen und Veröffentlichen geht ausschließlich über Xcode) und als Programmiersprache ist Objective-C vorgegeben.

2.1.4 Vertriebsmöglichkeiten

Jede der drei Plattformen hat eine eigene Vertriebsplattform. Die Möglichkeiten und Einschränkungen, die die jeweiligen Plattformen bieten, wird im folgenden Text behandelt.

Der **Apple App Store** wird für den Vertrieb von iOS Anwendungen genutzt. Anwendungen durchlaufen vor Veröffentlichung eine Prüfung durch Apple, in der Schadsoftware herausgefiltert wird. Der Zulassungsprozess von Apple steht schon seit längerer Zeit in der Kritik. Ein häufig gebrauchter Ablehnungsgrund ist die Nutzung von privaten Schnittstellen[14]. Weitere Zulassungsanfragen wurden zurückgewiesen, weil die Anwendung nicht nützlich sei ([AppStoreDrone]) oder bereits existierende Funktionalitäten dupliziert ([Podcaster]). Bei Vertriebsplattformen wie dem Apple App Store wird vielfach die Metapher vom Goldenen Käfig benutzt. Gemeint ist damit eine exklusive Vertriebsplattform, die durch eine zentrale Instanz moderiert wird und keinen anderen Weg

[14] Auch *Private API* genannt. Hierbei handelt es sich um undokumentierte Methoden, die für das Framework zur Verfügung gestellt wurden. Solche Schnittstellen können sich ändern und somit würden Anwendungen nach einem iOS-Update nicht mehr funktionieren. Aus diesem Grund lehnt Apple in seinem Zulassungsprozess Anwendungen ab, die private Schnittstellen benutzen (vgl. hierzu [DaringFireball2008]).

auf die jeweilige Plattform zulässt. Die Anwendungen können kostenlos oder kostenpflichtig sein. Für kostenpflichtige Anwendungen erhält der Entwickler 70% der Einnahmen. Die Zahlungsabwicklung und die Bereitstellung wird von Apple übernommen, sodass der Entwickler keine eigene Infrastruktur bereitstellen muss, um seine Anwendungen zu verkaufen.

Bei **Android** gibt es, anders als bei iOS, keinen Zwang, über die offizielle Vertriebsplattform zu verkaufen (Google Play Store). Entweder können andere Vertriebsplattformen installiert werden (Beispiel: Amazon Appstore for Android) oder die Software kann ohne den Umweg über eine solche Plattform heruntergeladen und installiert werden. Google Play bietet dem Entwickler wie der Apple App Store die Möglichkeit, seine Software entweder kostenlos oder kostenpflichtig anzubieten (wie bei Apple werden 70% der Einnahmen an den Entwickler weitergegeben).

Der **Windows Phone Marketplace** verfährt wie der Apple App Store. Es gibt einen Zulassungsprozess für Anwendungen, in dem die Einhaltung der vorgegebenen Regeln überprüft wird, sodass die Software keinen Schaden verursacht. Anwendungen können kostenlos oder kostenpflichtig veröffentlicht werden. Entwickler erhalten 70% der Einnahmen (siehe [WindowsPhoneHubFAQ]) und müssen sich nicht um die Bezahlungsabwicklung oder die Verteilung kümmern.

Im Vergleich unterscheiden sich die drei Plattformen kaum voneinander. Nur Android verfolgt eine offenere Zielsetzung. Jedoch häufen sich immer wieder auftauchende Nachrichten über Schadsoftware im Google Play Store. Wenn Software über eine dieser Plattformen vertrieben werden soll, ist vor Projektstart intensiv zu prüfen, ob es Konflikte mit den jeweiligen Regelwerken geben könnte. Durch die meist geschlossenen Systeme gibt es kaum eine Möglichkeit, diese Vertriebsplattformen zu umgehen (abgesehen von Android).

2.2 GebärdenSchrift

Abb. 1: GebärdenSchrift

Der folgende Abschnitt hat als Anwendungsbeispiel die Entwicklung eines Wörterbuches für GebärdenSchrift zum Thema. Basierend auf einen Abschnitt meines Industriepraktikumsberichtes (siehe [Industriepraktikumsbericht], Kapitel: GebärdenSchrift), wird hier ein kurzer Einblick in die GebärdenSchrift gegeben. Für umfassendere Informationen verweise ich auf das Handbuch zur GebärdenSchrift von Stefan Wöhrmann[15].

Der Erwerb der deutschen Schriftsprache ist für gehörlose Menschen mit erheblichen Schwierigkeiten verknüpft, da sich die Grammatik der deutschen Schriftsprache und der deutschen Gebärdensprache (DGS) stark unterscheiden. Ein oft genutztes Medium für Gehörlose beim Erwerb der deutschen Schriftsprache sind Videos, in denen in DGS gebärdet wird und die jeweils dazu passenden schriftlichen Vokabel visuell hinzugefügt werden. Die Erstellung solcher Lernmaterialien ist aufwendig und meist nur mit Gewährung von Fördermitteln zu leisten. Die Entwicklung einer Verschriftlichung der DGS ist der nächste logische Schritt, welcher in Ländern wie zum Beispiel den USA und Österreich bereits vollzogen wurde. Dort wurde ein System für die Auswahl von grafischen Darstellungen, die die gebräuchlichsten Bewegungen bei der Darstellung von Gebärden zeigen, entwickelt. Dieses System soll die Gehörlosen und die Lehrkräfte beim Erwerb und der Vermittlung der Schriftsprache unterstützen.

Die Idee der GebärdenSchrift stammt von der Tänzerin Valerie Sutton. Infolge eines Unfalls konnte sie ihren Beruf als Tänzerin nicht mehr ausüben. Sie entwickelte eine Choreographieschrift, um ihr Wissen weiterzugeben. Daraus entwickelte sich die „*Sutton SignWriting*“-Schrift, die zu den vielerorts anerkannten Verschriftlichungen der American Sign Language (ASL) gehört.

Die GebärdenSchrift findet immer mehr Anerkennung in Deutschland. Sie kann auch bei der Vermittlung von DGS an Hörende genutzt werden. Dies bedeutet, dass keine Videos oder digitalen Medien gebraucht werden, um die DGS, beziehungsweise die grammatische Schriftsprache, zu erlernen (Bsp.: Hausaufgaben können in schriftlicher Form gegeben werden).

---

[15] „*Handbuch zur GebärdenSchrift*“, Stefan Wöhrmann, ISBN: 3980900428.

2.3 Webtechnologien

Mit dem Begriff Webtechnologien werden Sprachen und Technologien zusammengefasst, die zur Erstellung von Inhalten für Web-Browser eingesetzt werden. Dabei wird in diesem Abschnitt davon abgesehen, auf die tieferen Schichten der Netzwerkkommunikation wie TCP/IP einzugehen. Hierfür empfiehlt sich als weitergehende Literatur „*Computer Networks and Internets*“ von Douglas E. Comer.

2.3.1 HTML

HTML (Hypertext Markup Language) ist eine textbasierte Auszeichnungssprache, die zur Strukturierung von Texten und anderer Medien genutzt werden kann. HTML-Dokumente bilden das Grundgerüst heutiger Webseiten. Dargestellt werden sie mit Hilfe eines Web-Browsers, der den HTML-Code analysiert und mit seiner Layout-Engine darstellt (vgl. hierzu [AndreasWalterUniKarlsruhe]).

Derzeit ist HTML 4.01 (siehe [W3CHTML4.1]) die aktuell genutzte Spezifikation, welche jedoch in nächster Zeit durch den Nachfolger HTML5 abgelöst wird. Alle derzeit aktuellen Browser unterstützen jetzt schon einen Großteil der HTML5-Spezifikation, die durch die World Wide Web Consortium (W3C) und Web Hypertext Application Technology Working Group (WHATWG) definiert wird. HTML5 basiert auf seinem Vorgänger HTML4 und ist damit abwärtskompatibel. Eine offizielle Empfehlung des W3C soll nach Komplettierung der Spezifikation im Jahr 2014 ausgesprochen werden.

Zu den Neuerungen von HTML5 gehören neue Multimedia-Elemente mit denen Video- und Audio-Medien direkt in Webseiten eingebunden werden können. Das Canvas-Element ermöglicht zweidimensionale Formen mittels definierter Schnittstellen zu zeichnen. Dies bedeutet z.B. Spielentwicklung ohne auf Produkte wie Flash[16] oder Unity[17] zurückgreifen zu müssen. Hinzu kommt die Möglichkeit der lokalen Speicherung von Daten mit Hilfe von LocalStorage und die Erstellung von Offline-Webanwendungen durch das HTML5 Manifest. Diese und weitere Neuerungen ermöglichen den Bau komplexerer Webanwendungen ohne die Nutzung von externen Plugins. (vgl. hierzu [W3CHtml5Differences] und [HTML5]).

[16] Plattform von Adobe (ehemals Macromedia) zur Darstellung von multimedialen Inhalten wie Videos und Audio, aber auch oft genutzt zur Entwicklung von Spielen und anderer interaktiver Medien.

[17] Plattform von Unity Technologies, die zur Entwicklung von 3D-Spielen genutzt wird.

2.3.2 CSS

Die Cascading Style Sheets sind eine Definitionssprache für HTML-Element-Eigenschaften. Mit ihrer Hilfe lassen sich Eigenschaften wie z.B. Farbe oder Ausrichtung beeinflussen. Dabei wird ein sogenanntes Format definiert und dieses auf ein oder mehrere HTML-Elemente angewendet. Ein Beispiel ist:

```
1 body{
2    background: #efefef;
3 }
```

Mittels des Selectors, in diesem Fall *body*, wird bei dem Dokument, in das dieses Stylesheet eingebunden wird, auf das Body-Element bzw. dessen Eigenschaft *background* zugegriffen und somit die Hintergrundfarbe auf den Wert *#efefef* (Hex-Farbcode) gesetzt.

Wie bei HTML findet auch bei CSS eine Weiterentwicklung statt. Der derzeit aktuellen Version 2 folgt im Zuge der Einführung von HTML5 die Version CSS3, welche 3D-Transformationen, Schattenwurf und viele andere Neuerungen bietet (siehe [ComMagazinCSS]).

2.3.3 JavaScript

JavaScript ist eine Skriptsprache, die oft in Webseiten eingebettet wird und somit auf Clientseite läuft. Der JavaScriptCode läuft aus Sicherheitsgründen in einer Sandbox[18]. Bis auf ihre syntaktische Ähnlichkeit haben JavaScript und Java nicht viel gemeinsam. Im Gegenteil zu Java handelt es sich bei JavaScript um eine dynamisch getypte Skriptsprache. Dabei basiert bei JavaScript die Objektorientierung nicht auf Klassen, sondern auf Prototypen. Zu den typischen Anwendungsfällen von JavaScript gehört die Manipulation des Document Object Models (DOM), welcher die Schnittstelle für den Zugriff auf das HTML-Dokument darstellt. Die Funktionalitäten der Sprache werden durch den Standard ECMA-262 (ECMAScript) definiert.

Eine weitere neue Entwicklung ist, dass JavaScript durch den Einsatz von z.B. node.js auch auf Serverseite eingesetzt werden kann und somit beide Seiten (Client & Server) in derselben Sprache geschrieben werden können.

---

[18] Übersetzt Sandkasten. Bezeichnet einen kontrollierten Bereich, in dem Aktionen keine Auswirkungen nach draußen haben. Dieser wird oft als Sicherheitsmechanismus gebraucht, um Anwendungen laufen zu lassen, ohne dabei Schaden für das Wirtssystem befürchten zu müssen.

3

Plattformunabhängige Anwendungsentwicklung

In dem vorangegangenem Abschnitt wurden die Grundlagen vermittelt, die für die folgenden Abschnitte benötigt werden. In dem Teil über die verschiedenen Plattformen wurde aufgezeigt, dass die Entwicklung für verschiedene Plattformen mit erheblichen Aufwand verbunden ist. Um diesem Mehraufwand aus dem Weg zu gehen, wird bei der Anwendungsentwicklung für mobile Plattformen mittels der Nutzung von Webtechnologien das Motto „*Write once, run anywhere*“ wiederbelebt. Dabei haben Webtechnologien den Vorteil, einer großen Anzahl von Entwicklern bereits bekannt zu sein. Dieser Umstand kann zu einer Kostenreduktion beitragen (siehe [WebAppKosten]).

Als Beispiel für eine Plattform, die versuchte diesem Weg zu folgen, kann man das iPhone benennen. Zum Zeitpunkt seiner Einführung war die Plattformunabhängigkeit von mobilen Anwendungen noch kein großes Thema, trotzdem war es im ersten Jahr nur möglich, WebApps zu entwickeln. Dieser Ansatz wurde von Steve Jobs (damals CEO von Apple) als „*sweet solution*“ (deutsch: eine schöne Lösung) bezeichnet ([AppleWebApp]). Er argumentierte, dass die Entwickler durch die Nutzung von bekannten Technologien keinen großen Aufwand bei der Entwicklung für neue Plattformen erbringen müssen. Ein weiteres Argument seinerseits war, dass durch die Kommunikation der Anwendung mit dem Gerät mittels der Browser-Schnittstelle (Sandbox) vielen Sicherheitsproblemen aus dem Weg gegangen werden kann. Doch es folgten Proteste seitens der Entwickler, die sich ausgeschlossen fühlten und die WebApps als zu langsam ansa-

hen. Etwa ein Jahr später wurde seitens Apple das SDK veröffentlicht, mit dem sich native Anwendungen für das iPhone entwickeln ließen.

Plattformen wie Android und iOS gestatten es Webseiten, sich auf dem Homescreen[19] zum Zweck der schnelleren Erreichbarkeit zu platzieren. Auf iOS hat das Verlinken auf dem Homescreen noch weitere Vorteile, wie die Möglichkeit die Webseite in den Vollbildmodus zu versetzen oder einen Startbildschirm anzuzeigen. Webseiten die speziell für mobile Geräte optimiert sind, werden gewöhnlich als WebApps bezeichnet. Die Hybrid-Anwendungen[20], hier handelt es sich um einen Oberbegriff unter dem sich verschiedenste Frameworks wie z.B. PhoneGap[21] oder Titanium sammeln, gehen einen Schritt weiter in die Richtung zur nativen Anwendung. Im Rahmen dieser Ausarbeitung wird auf zwei Untertypen eingegangen. Zum einen die Wrapper-Hybriden, welche versuchen Webseiten in native Anwendungen einzubinden und diesen damit die Möglichkeit zu bieten, auf native Schnittstellen zuzugreifen, und zum anderen die Interpreter-Hybriden, wie Titanium[22] von Appcelerator, welche eine eigene Entwicklungsumgebung bieten. Mit Titanium können mittels JavaScript Anwendungen entwickelt werden, wobei sich Titanium darum kümmert, dass aus dem Code eine native Anwendung gebaut wird. Diese von Titanium erzeugte Anwendung interpretiert zur Laufzeit den JavaScript-Code und generiert native Bedienelemente und Zugriffe auf die Geräte-Schnittstellen.

Auf diese Konzepte wird in den folgenden Abschnitte näher eingegangen und ihre Vor- und Nachteile anhand von Beispielen verdeutlicht.

3.1 WebApps

Der Begriff WebApp steht für Web-Application also Web-Anwendung. Web-Anwendungen sind in der heutigen Zeit weit verbreitet. In erster Linie handelt es sich um eine Webseite, die durch Schnittstellen, die der Browser zu Verfügung stellt, versucht, einen Funktionsumfang zu erreichen, den sonst native Anwendungen bieten. Paradebeispiele für eine solche Verzahnung sind z.B. Google Mail oder Google Text&Tabellen. Diese

---

[19] Beim Homescreen handelt es sich um die Hauptansicht auf dem Smartphone, von der aus Anwendungen gestartet werden können (sozusagen eine simplere Version vom Schreibtisch auf dem PC).

[20] Keine native Anwendung, aber auch keine reine WebApp.

[21] PhoneGap ist eine Open-Source-Framework, welches unter phonegap.com zu finden ist.

[22] Titanium ist ein kommerzielles Produkt von der Firma Appcelerator, welches unter appcelerator.com zu finden ist.

bieten einen Funktionsumfang, der den so mancher nativer Anwendung übertrifft. Auch bedarf es keiner herkömmlichen Installation. Weiterentwicklungen der Web-Browser bzw. der HTML-Spezifikationen erlauben es heutigen Web-Anwendungen, Daten lokal zu speichern oder für sich selbst offline verfügbar zu machen. Viele Firmen engagieren sich in dem Bereich und versuchen die Entwicklung der Webtechnologien soweit voran zu treiben, dass die üblichen Alltagsaufgaben[23]in Zukunft mit Webanwendungen absolviert werden können. Der Browser entwickelt sich immer mehr zu einer Art neuen Betriebssystem.

Ein gescheiterter Versuch ein nur auf die Nutzung von Webtechnologien zurückgreifendes Betriebssystem für mobile Geräte zu schaffen, ist WebOS der Firma Palm. Dieses wurde auf der CES 2009 in Las Vegas zusammen mit dem Palm Pre vorgestellt und sollte an die früheren Erfolge der Firma Palm im mobilen Bereich anknüpfen. Anwendungen für WebOS wurden nur mit Webtechnologien wie JavaScript, HTML und CSS entwickelt. Da die Geräte zu wenig Leistung hatten, um sich mit anderen Smartphones messen zu können, scheiterte dieser Versuch, zumal auch das Marketing nicht ausreichend betrieben wurde. Die Plattform war zu diesem Zeitpunkt ihrer Zeit voraus und hätte ein paar Jahre später mehr Chancen auf Erfolg gehabt (vgl. [Palm]). In dieselbe Richtung, nur für Desktops und Laptops, geht die Firma Google. Ihr Betriebssystem Google Chrome OS basiert auf einem Linux-Kern und ihrem Web-Browser Google Chrome. Es zielt auf die ausschließliche Nutzung von sich stark in die Deskopumgebung integrierenden Webanwendungen ab. Über den auch allen anderen Google-Chrome-Benutzern zur Verfügung stehenden Chrome Store lassen sich Webanwendungen kostenlos oder kostenpflichtig vertreiben.

Die erste Ausbaustufe von einer gewöhnlichen Webseite zu einer WebApp enthält z.B. die Anpassung für die Auflösung des Geräts, welches die Webseite anzeigt. Gerade die neuen Technologien HTML5 und CSS3 bieten Webdesignern diese Möglichkeiten. Ein Begriff, der gerne in diesem Zusammenhang genutzt wird, ist das *Responsive Design*. Dieses Konzept ist in der Praxis der plattformunabhängigen Anwendungsentwicklung auf Grundlage von CSS und HTML wichtig und wird nun in Abschnitt 3.1.1 vorgestellt.

[23] Dazu zählen Textverarbeitung, Bildbearbeitung, Verwaltungswerkzeuge und Kommunikationsanwendungen.

3.1.1 Responsive Design

Zu den ersten Reaktionen auf den wachsenden Smartphone-Markt und die damit steigende Anzahl an mobilen Zugriffen übers Internet gehörte die Schaffung separater für die einzelnen Plattformen optimierter mobiler Webseiten. Diese Vorgehensweise hat folgende Nachteile:

1. Fragmentierung der Anwendung wird durch Plattformvielfalt zu Problem
2. Beim Verschicken eines Links an nicht mobile Nutzer (Bsp.: Desktop-PC) werden diese auch auf die mobile Version verwiesen

Die damalige Vorgehensweise entstand aus der fehlenden Möglichkeit, dynamisch auf Auflösung oder andere Parameter zu reagieren. Durch die Weiterentwicklung der Webtechnologien, insbesondere durch HTML5 und CSS3, wurden Vorgehensweisen wie Responsive Designs (deutsch: reagierendes Design) möglich. Unter dem Begriff Responsive Design versteht man ein Vorgehen, bei dem der grafische Aufbau auf das Anzeigemedium (Laptop, Smartphone etc.) optimiert wird (vgl. [ReDe, Kapitel 1]). Zu den großen Verfechtern des Responsive Design gehört unter anderem Ethan Marcotte, der mit seinem Artikel [RWDArtikel] im Jahr 2010 den Begriff prägte und eine praxisnahe Einführung in das Thema veröffentlichte. Nach [ReDe, Kapitel 1] gehören zu den wichtigsten solche Herangehensweise erlaubenden Technologien die Media Queries. Diese bauen auf die in CSS2.1 definierten, bislang wenig verbreiteten Media Types auf und haben sich mit der CSS3-Spezifikation zu einem effektiven Werkzeug für die Entwicklung von sich dem Gerät anpassenden Layouts weiterentwickelt. Media Queries ermöglichen Parameter wie z.B. der Bildschirmgröße und Orientierung auszulesen oder diese gleich als Bedingung für die Verwendung eines CSS-Styles zu definieren. Ein Beispiel für die Verwendung von Media Queries ist folgender Code:

```
1 /* iPads 1 Erkennung im Portrait-Modus ----------- */
2 @media only screen and (min-device-width : 768px)
3                   and (max-device-width : 1024px)
4                   and (orientation : portrait) {
4 ... /* Styles */
5 }
```

Dieser Code gestattet es, Geräte wie das iPad 1 zu erkennen und entsprechend den CSS-Style anzupassen. Dabei wird durch Angabe der Mindestbreite und Maximalbreite in Kombination mit der Orientierung gefiltert. Die Vorgehensweise lässt sich dank der

vielen Möglichkeiten an Abfragen mit Hilfe der Media Queries auf alle möglichen Auflösungen und Gerätetypen übertragen[24].

Als Kritikpunkt an Responsive Design wird häufig genannt, dass bei der Anpassung für mobile Geräte nur eine Neuanordnung oder das Ausblenden von Inhalten stattfindet. Das Benutzer auf mobilen Plattformen einen ganz anderen Anwendungsfall für die Nutzung der Webseite anstreben, wird dabei außer Acht gelassen. Ein anschauliches Beispiel dafür ist die Webseite der deutschen Post. Benutzer, die über die mobile Webseite zugreifen, interessieren sich recht häufig für den Standort des nächstgelegenen Briefkasten oder Bankautomaten. Die Zugriffe von Zuhause auf die normale Webseite forschen eher nach Informationen über Preise oder das Online-Banking. Ob eine eigene mobile Version oder eine sich anpassende Version die bessere Wahl ist, muss auf der Basis der Intention geregelt werden.

Mit Responsive Design wurde ein Vorgehensweise vorgestellt, mit der sich Webseiten dynamisch auf die Anforderungen von Smartphones oder Tablets anpassen lassen. Zusätzlich bieten Plattformbetreiber verschiedene Möglichkeiten, Anwendungen besser in das System zu integrieren. Eins der zur Verfügung gestellten Hilfsmittel sind die sonst zur Optimierung für Suchmaschinen (englisch: Search Engine Optimisation) genutzten Meta-Tags. Mit diesen lassen sich Zusatzinformationen in die Webseite einbetten, die von den Web-Browsern erkannt werden können. Ein kleines Beispiel für ein Meta-Tag[25]:

```
1 /* -----------  Vollbildmodus ----------- */
2 <meta name="apple-mobile-web-app-capable" content="yes">
3 /* -----------  Statusleisten-Farbe ----------- */
4 <meta name="apple-mobile-web-app-status-bar-style" content="black">
```

Durch Angabe des Meta-Tags aus Zeile 2 wird eine in iOS auf dem Homescreen gespeicherte Webseite im Vollbild-Modus gestartet. Dabei verschwindet die typische obere Adressleiste und die untere Aktionsleiste. Dieses und weitere Meta-Tags bieten auf iOS die Möglichkeit eine Anwendung grafisch aus dem Web-Browser-Kontext herauszulösen.

---

[24] Eine Liste der verschiedenen möglichen Abfragen findet sich auf www.w3.org/TR/css3-mediaqueries/

[25] Weitere Meta-Tags und mehr Informationen zu den von den einzelnen Plattformen angebotenen Meta-Tags sind in den entsprechenden Dokumentationen zu finden.

Damit auch andere Plattformen solche Möglichkeiten nutzen können, haben einige Web-Frameworks wie jQuery Mobile und Sensa Touch es sich zur Aufgabe gemacht, Hilfsmittel für die Entwicklung von auf mobile Plattformen spezialisierten Anwendungen zur Verfügung zu stellen. Die Zielrichtung der meisten dieser Frameworks ist, mit Hilfe von HTML, CSS und JavaScript dem Funktionsumfang von GUI-Frameworks nativer Anwendungen (bei iOS z.B. Cocoa) nahezukommen. Der folgende Abschnitt beschäftigt sich mit dieser Art von Frameworks und stellt die verschiedenen Konzepte vor.

3.1.2 GUI-Frameworks

Bei der Entwicklung von Webseiten wird heutzutage meistens auf die die Funktionalität erweiternden JavaScript-Frameworks zurückgegriffen. In diesem Abschnitt werden zwei unterschiedliche Entwicklungsansätze nutzende Frameworks vorgestellt. Diese Frameworks werden bevorzugt frequentiert und übernehmen gewissermaßen eine Patenrolle für die meisten anderen Frameworks. Als erstes Framework wird jQuery vorgestellt. Nach[BuildWith] wird dieses bei mehr als 58% der von Quantcast als erfolgreichsten zehntausend eingeschätzten Webseiten eingesetzt. Für den Einsatz auf mobilen Geräten wurde das auf die Anforderungen an den mobilen Einsatz optimierte und auf jQuery aufbauende jQuery Mobile entwickelt. Dieses bietet eine auf HTML5 basierende Benutzeroberfläche, die für einen Großteil der mobilen Plattformen optimiert ist. Zu den damit unterstützten Plattformen gehören neben Apple iOS, Android, Windows Phone, BlackBerry und anderen auch die fünf großen Desktop-Web-Browser Chrome, Safari, Firefox, Internet Explorer und Opera ([jQueryBrowser]). Eingesetzt wird jQuery Mobile von Organisationen wie Disney World, Box.net und der Universität Standford ([jQueryResources]).

Die Definition der Benutzeroberfläche findet in jQuery Mobile anhand HTML-Elementen, denen Rollen zugewiesen werden, statt. So lässt sich die Benutzeroberfläche komplett in HTML, ohne den Einsatz von JavaScript, beschreiben. Weil das komplette Neuladen der Inhalte durch hohe Latenzen und langsame Verbindungen zu viel Zeit kostet, werden in jQuery Mobile üblicherweise alle Ansichten in einer Datei definiert und durch JavaScript entsprechend ein- bzw. ausgeblendet. Auf die Methode des dynamischen Nachladens wird im *Abschnitt 4.4.2 Performance-Optimierung* genauer eingegangen. Das dynamische Nachladen führt zu der Aufhebung der üblichen Trennung von Inhalten in verschiedene HTML-Dokumente. Der Grundaufbau einer in jQuery Mobile hergestellten WebApp sieht folgendermaßen aus. Es wird zum Zweck

der besseren Übersichtlichkeit auf die Imports der jQuery-Mobile-Bibliotheken und <html>-Tags verzichtet:

```
1  <div data-role="page">
2     <div data-role="header">
3      ...
4     </div>
5     <div data-role="content">
6      ...
7     </div>
8     <div data-role="footer">
9      ...
10    </div>
11 </div>
```

In Zeile 1, 2, 5 und 8 wird jeweils durch das Attribut `data-role` der Typ des Div-Elements angegeben und somit die Struktur der Anwendung definiert. Für jede dieser data-roles ist in den Templates für die jeweiligen Plattformen eine Position und ein Style definiert. Zur Veranschaulichung, wie ein solches Div-Element mit Inhalt gefüllt werden kann und wie es später aussieht, wird dem Header nun ein Zurück-Button und ein Label hinzugefügt:

```
1     <div data-role="header">
2         <a href="index.html" data-icon="back">Zurück</a>
3         <h1>Testanwendung</h1>
4     </div>
```

Das folgende Bild zeigt das Ergebnis des obigen Codes:

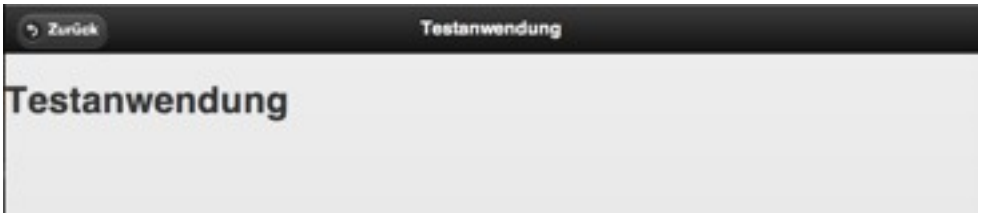

Abbildung 2: jQuery Mobile Testanwendung

Das zweite vorgestellte Framework Sencha Touch vertritt einen anderen Ansatz bei der Definition des Layouts. Im Gegensatz zu jQuery setzt Sencha Touch nicht auf die Auszeichnungssprache HTML zur Definition der Benuzteroberfläche, sondern auf die Erstellung der Bedienelemente durch JavaScript.

Im folgenden Beispielcode wird eine Seite mit einem Header definiert:

```
1  Ext.application({
2     requires: ['Ext.Toolbar'],
3     launch: function() {
4               Ext.Viewport.add({
5                    items:    [{   title: 'Test',
6                                   xtype: 'toolbar',
7                                   docked: 'top',
8                                   items: [{ text: 'Zurück'}],
9                              }],
10              });
11   },
12 });
```

Bei der Programmierung mit Sencha Touch werden elemente direkt in JavaScript definiert dabei wird die JavaScript Object Notation (JSON) genutzt. Das folgende Bild zeigt das Ergebnis des obigen Codes:

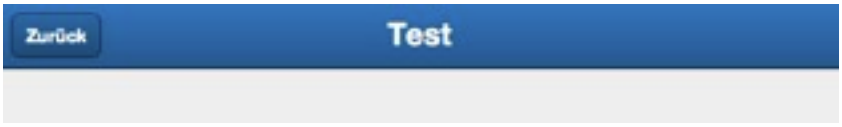

Abbildung 3: Sencha Touch 2 Beispielanwendung

Bei der Darstellung der Elemente versucht Sencha Touch, ein „*nativ wirkendes Bedienerlebnis zu schaffen*“[26]. Ein Versuch, der durch das Festhalten an den Bedienkonzepten der iOS-Plattform nicht vollständig gelingt. Ein Beispiel wäre das Imitieren des typischen iOS-TabPanel auch auf Android. Dieses Vorgehen bringt kein „*natives Erlebnis*“ zur Geltung. Nur von wenigen GUI-Frameworks wird die Transformation des Layouts für die verschiedenen Plattformen ganz vollzogen. Nicht nur eine Änderung des angewendeten Styles auf den Elementen, sondern auch das Bereitstellen eigener Bedienelemente mit entsprechender Positionierung für die jeweiligen Plattformen ist dafür notwendig. Eines dieser Frameworkst ist mgwt, auf welches im *Abschnitt 4.3.2 - MGWT* näher eingegangen wird.

[26] Originaltext: „Sencha Touch enables you to quickly and easily create HTML5 based mobile apps that work on Android, iOS and Blackberry devices and produce a native-app-like experience inside a browser.“ aus www.sencha.com/learn/getting-started-with-sencha-touch-2/

3.1.3 Schnittstellen zum Gerät

Vorab war das Thema die WebApps und deren Versuch, den nativen Anwendungen über die grafische Darstellung nahe zu kommen. Wie in den Grundlagen zum Thema HTML5 erwähnt wurde, haben WebApps durch die Schnittstellen der Browser auch die Möglichkeit, sich auf der Ebene der Funktionalität anzunähern. Dieser Abschnitt stellt nun eine durch HTML5 spezifizierte Schnittstelle vor, um die Gestaltung der Interaktion mit dem Gerät zu veranschaulichen.

Ortungsdienste sind in vielen Bereichen eine nützliche Erweiterung des Funktionsumfangs einer Webseite. Der Vorteil des Ortens im Vergleich zur manuellen Eingabe des derzeitigen Aufenthaltsorts liegt auf der Hand. Zu diesem Zweck kann anhand eines einfachen JavaScript-Codes die Ortung nach vorheriger Zustimmung des Benutzers ausgelöst werden.

```
1  navigator.geolocation.getCurrentPosition(
2     //Callback bei erfolgreicher Positionsermittlung
3     function(pos){
4       'Lati.: '+pos.coords.latitude+'/ Longi.: '+pos.coords.longitude;
5     },
6     //Callback bei Fehler
7     function(){
8       ...
9     }
10 );
```

In Zeile 3 bis 9 werden die beiden Callbacks[27] definiert, dabei wird die erste Funktion nach erfolgreicher Positionsermittlung aufgerufen. Die zweite Funktion wird aufgerufen, wenn ein Fehler bei der Positionsermittlung auftrat. Das von der Funktion `getCurrentPosition` übergebene `Position`-Objekt enthält die Koordinaten. In Zeile 4 ist die Art des Auslesens zu sehen. Neben der Position lassen sich auch Geschwindigkeit, Höhe und Ausrichtung abfragen ([GeoLocW3C]).

[27] Eine Callbackfunktion (deutsch: Rückruffunktionen) wird einer anderen Funktion übergeben. Diese ruft gegebenenfalls die Callback-Funktion auf und kann ihr dabei Daten als Parameter übergeben.

3.2 Wrapper-Hybride

Im vorangegangen Abschnitt wurden reine WebApps vorgestellt. Es wurde auf die grafische Darstellung und die Möglichkeit des Zugriffs auf hardwarenahe Funktionalität wie die Verbindung zu Ortungsdiensten eingegangen. Die HTML5-Spezifikation wird mit der Zeit um immer mehr Zugriffsarten erweitert. Dieser Prozess der Spezifikation geht vielen Entwicklern zu langsam. Um nun auf mehr oder sogar alle Schnittstellen, die in der nativen Anwendungsentwicklung zur Verfügung stehen, Zugriff zu erlangen, bedarf es eines neuen Ansatzes. Dieser wird durch die Wrapper-Hybriden möglich. Dabei handelt es sich um eine native Anwendung, die auf der Darstellungsebene nur aus einem WebView[28] besteht. Diese eigentlich simple Idee wird in verschiedensten Ausprägungen umgesetzt. Die einfachste Variante ist das Verpacken einer Webseite in eine native App. Firmen, die sich keine eigene native App leisten können oder wollen, gelangen so in die jeweiligen Distributionsplattformen[29]. Die für diese Ausarbeitung wirklich interessanten Wrapper-Hybriden setzen die Entwicklung fort, indem sie ihren beherbergten WebViews eigene Schnittstellen zur Hardware bieten.

3.2.1 PhoneGap/Cordova

Jedes moderne mobile Betriebssystem erlaubt seinen Entwicklern, auf Ereignisse innerhalb eines eingebunden WebViews zu reagieren und einen JavaScript-Code zu injizieren. Ein sich diesen Umstand zunutze machendes Open-Source-Framework ist PhoneGap[30]. Hier wird WebApps die Möglichkeit geboten, über eine einheitliche JavaScript-Schnittstelle auf native Schnittstellen zuzugreifen. PhoneGap bietet für alle größeren Plattformen[31] jeweils ein als Brücke zwischen dem JavaScript-Code und den nativen Schnittstellen fungierendes SDK an. Erweiterbar ist dieses SDK mit in der jeweiligen nativen Programmiersprache geschriebenen Plugins ([PhoneGapDocPlugin]).

Die Funktionsweise ist einfach strukturiert. Das SDK erzeugt eine native Anwendung mit nur einem Oberflächen-Element, einem WebView. In diesem wird die in das Projekt

---

[28] WebView bezeichnet eine Komponente, die dem Entwickler zur Verfügung steht, um Webseiten anzuzeigen.

[29] Die Distributionsplattformen sind ein einfacher Weg, die gewünschte Zielgruppe zu erreichen und bilden damit auch ein Marketing-Instrument.

[30] Ursprünglich wurde PhoneGap von der Firma Nitobi entwickelt. Durch die Übernahme durch Adobe wurde das PhoneGap-Projekt als Apache-Incubator-Projekt mit dem Namen Apache Cordova veröffentlicht (vgl. [t3nPhoneGap]). http://incubator.apache.org/cordova

[31] iOS, Android, Windows Phone, BlackBerry, Bada, Symbian, webOS. Für eine aktuelle Übersicht siehe http://docs.phonegap.com/en/2.0.0/guide_getting-started_index.md.html

eingebundene WebApp geöffnet. Der Zugriff auf hardwarenahe Schnittstellen wird mit Hilfe von Proxy-Objekten zur Verfügung gestellt. Die Proxy-Objekte fungieren als Brücke, indem sie die in JavaScript erzeugten Aufrufe an den entsprechenden nativen Code weiterleiten. Der Rückkanal wird durch das Injizieren von JavaScript-Code in den WebView bewerkstelligt. Dabei werden Ergebnisse oder andere Daten als Parameter an eine beim Aufruf übergebene Rückruffunktion (Callback) übergeben. Der entsprechende WebApp-Code wird mit dem vom SDK generierten Code zu einer Anwendung verpackt und kann danach über die jeweiligen Distributionsplattformen vertrieben werden.

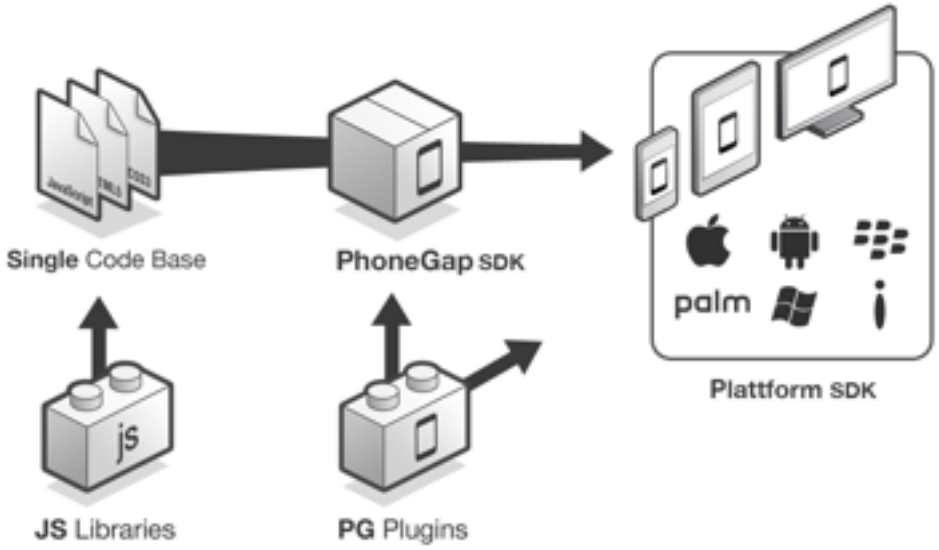

Abbildung 4: PhoneGap-Funktionsweise

Bei der Entwicklung der WebApp kommen alle im *Abschnitt 3.1* beschriebenen Methoden zum Einsatz. Für die WebApp können beliebige JavaScript-Bibliotheken bzw. Frameworks benutzt werden. Die Nutzung von GUI-Frameworks wie Sencha Touch oder jQuery in Kombination mit PhoneGap ist eine übliche Vorgehensweise. Zur Speicherung von Daten lassen sich neben den beschriebenen HTML5-Features auch Schnittstellen zum App-Speicher nutzen. Durch das Verpacken der WebApp in die native Anwendung beschleunigen sich die Ladezeiten, da weniger Daten aus dem Internet nachgeladen werden müssen. Auch die Kommunikation mit anderen Diensten vereinfacht sich enorm. In PhoneGap gespeicherte WebApps werden aus dem lokalen Speicher heraus geöffnet und fallen damit nicht unter die *Same origin policy* (mehr dazu in *Abschnitt 4.4.1*). So können Inhalte ohne großen Aufwand von anderen Servern, unter anderem mit Hilfe von REST, JSON oder WebSockets, nachgeladen werden.

	iOS 3GS+	Android	Blackberry OS 6.0+	Windows Phone 7	Symbian	Bada
Beschleunigungsmesser	X	X	X	X	X	X
Kamera	X	X	X	X	X	X
Kompass	X	X	-	X	-	X
Kontakte	X	X	X	X	X	X
Dateien	X	X	X	X	-	-
Position	X	X	X	X	X	X
Medien	X	X	-	X	-	-
Netzwerk	X	X	X	X	X	X
Benachrichtigungen	X	X	X	X	X	X
Speicher	X	X	X	X	X	-

Tabelle 2: Die von PhoneGap unterstützten Schnittstellen zu den jeweiligen Geräten ([PhoneGapSupport])

Um den Zugriff über die PhoneGap-JavaScript-Brücke zu veranschaulichen, wird im folgenden Beispiel für iOS aus der PhoneGap-Dokumentation der Beschleunigungsmesser ausgelesen ([PhoneGapAcceleration]):

```
1  function onSuccess(acceleration)(
2       alert( 'Acceleration X: ' + acceleration.x + '\n' +
3              'Acceleration Y: ' + acceleration.y + '\n' +
4              'Acceleration Z: ' + acceleration.z + '\n' +
5              'Timestamp: '      + acceleration.timestamp + '\n');
6  };
7  function onError(){
8    ...
9  }
10
11 navigator.accelerometer.getCurrentAcceleration(onSuccess, onError);
```

Durch den Aufruf von *getCurrentAcceleration* am von PhoneGap zur Verfügung gestellten Objekt *accelerometer* wird intern über die *GapBridge* auf die native Klasse CDVAccelerometer zugegriffen und zwei Callback-Funktionen übergeben. Nachdem in CDVAccelerometer die aktuelle Beschleunigung bestimmt wurde, werden die Daten in ein Acceleration-Objekt gespeichert und mit Hilfe der Methode *stringByEvaluating-JavaScriptFromString* des WebViews in denselbigen injiziert. Dabei werden die Daten über die Callback-Funktionen im JSON-Format übergeben.

3.3 Interpreter-Hybride

Besondere Aufmerksamkeit bei der Entwicklung von WebApps und demzufolge auch bei Wrapper-Hybriden muss für die Darstellung der Benutzeroberfläche aufgebracht werden. Nicht ohne Grund entstehen immer mehr GUI-Framworks mit der Zielrichtung, die Entwickler bei dieser Arbeit zu unterstützen. Ist ein natives Aussehen und Verhalten der Bedienelemente angestrebt, verspricht die Nutzung nativer Elemente den größten Erfolg. Zu diesem Zweck bieten sich Interpreter-Hybriden an. Dieses Kapitel behandelt das hierfür Pate stehende Framework Titanium von Appcelerator mit dem dahinterstehendem Konzept.

3.3.1 Appcelerator Titanium

Titanium ist ein auf Open Source aufbauendes Framework, welches in seiner Grundfunktionalität kostenlos genutzt werden kann. Mit Titanium Studio stellt Appcelerator eine auf Eclipse basierende Entwicklungsumgebung zur Verfügung. Unterstützt werden als Ziel-Plattformen iOS und Android. Neben dem SDK und der Entwicklungsumgebung bietet Applecerator auch in kleinster Ausführung Cloud-Services in dem kostenlosen Entwickleraccount an. Kostenpflichtig sind Erweiterungen für das SDK, größere Cloud-Services wie Push und die Buchung von mehr Speicherplatz .

Bei Titanium handelt es sich nicht, wie oft behauptet, um einen Cross-Compiler. Vielmehr wird die in JavaScript definierte Programm-Logik und Oberfläche zur Laufzeit innerhalb einer nativen Anwendung interpretiert (vgl. [Titanium]), d.h. entsprechende native Elemente werden in den aktuellen View eingebunden und mit Inhalt gefüllt. Die Nutzung nativer Elemente hat den Vorteil, dass keine aufwändige Nachahmung des Verhaltens nativer Bedienelemente abgefasst werden muss.

Abbildung 5: Aufbau einer Titanium-Anwendung

Abbildung 5 zeigt den grundlegenden Aufbau einer mit Titanium erzeugten Anwendung. Der Anwendungsquellcode in JavaScript bedient sich der vom Titanium-Framework zur Verfügung gestellten Schnittstellen. Zur Laufzeit wird dieser Quellcode von

dem in die Anwendung eingebundenen Interpreter – auf iOS JavaScript Core und auf Android Rhino (vgl. [Titanium]) – verarbeitet und aus den jeweiligen Definitionen werden die gewünschten Bedienelemente erzeugt. Zur Veranschaulichung ein kleines Beispiel:

```
1   var tab_group = Titanium.UI.createTabGroup();
3
4   var window = Titanium.UI.createWindow({
5                   title:“Testled Status“,
6                   backgroundColor:“#fff“,
7                   tabBarHidden: false,
8   });
9
10  var tab = Titanium.UI.createTab({
11       title: "Test",
12       icon: "images/KS_nav_ui.png",
13       window: window
14  });
15
16  tab_group.addTab(tab);
17  tab_group.open();
```

In Zeile 1 wird ein TabPanel erstellt und der Variable tab_group zugewiesen. In dieses wird in Zeile 16 das in Zeile 10 bereitgestellte Tab hinzugefügt. Diesem Tab wurde bei der Erzeugung in Zeile 10-14 die Attribute für seinen Titel, sein Bild und seinem Inhalt mitgegeben. Abschließend wird in Zeile 17 noch der einzige Tab geöffnet.

Aus dem Titanium Studio heraus lässt sich die Anwendung direkt in den jeweiligen Simulatoren bzw. Emulatoren öffnen. Diese müssen separat installiert sein, z.B im Fall von iOS wird ein MacOS X vorausgesetzt.

Abbildung 6: Screenshot der Beispielanwendung auf iOS

4

Anwendungsbeispiel WebApp

Dieser Abschnitt behandelt die Entwicklung des GebärdenSchrift-Wörterbuchs im Rahmen des Delegs-Projekts. Dabei wird besonders auf die genutzten Technologien und die damit verbundenen Hindernisse eingegangen. Das GebärdenSchrift-Wörterbuch basiert auf einer WebApp, die mit Hilfe von Mobile GWT[32](GUI-Framework) für Android, iOS und BlackBerry ein natives Erscheinungsbild bekommt. Die Erfahrungen mit dieser Handlungsweise werden im nächsten Abschnitt als Basis für ein Fazit genutzt.

4.1 Anforderungen und Vorgaben

Im Rahmen des Delegs-Projekts wurde von den fachlichen Mitarbeitern der Wunsch nach einer mobilen Anwendung zum Nachschlagen von GebärdenSchrift-Vokabeln bekundet. Der gewünschte Funktionsumfang wurde wie folgt definiert:

- Nachschlagen von deutschen Wörtern
- Autovervollständigung beim Nachschlagen
- GebärdenSchrift-Alternativen zu dem nachgeschlagenen Wort
- Speichermöglichkeit von häufig benutzten Vokabeln
- Kostenlos und schnell verfügbar

Diese zu entwickelnde Anwendung soll sowohl auf Android als auch auf iOS funktionieren. Da Webtechnologien im Entwicklungsprozess eine große Rolle spielen, soll, um auch eine gute Wartung durch das Entwicklerteam zu garantieren, das Mobile-GebärdenSchrift-Wörterbuch auf diesen Technologien basieren.

[32] Offiziell mgwt genannt und auf http://www.m-gwt.com/ zu finden.

4.2 Benutzte Technologien

Das mobile GebärdenSchrift Wörterbuch wird in seiner ersten Ausbaustufe eine Web-App, da die Vorgabe der Nutzung von Webtechnologien die native Anwendungsentwicklung ausschließt. Der Funktionsumfang bleibt übersichtlich und es werden keine speziellen Schnittstellen zum Gerät benötigt. Außerdem wird keine kostenpflichtige Distribution angestrebt. Zu einem späteren Zeitpunkt lässt sich die WebApp, falls gewünscht, problemlos zu einem Wrapper-Hybriden mit PhoneGap verpacken.

4.3.1 LocalStorage

Eine der benötigten HTML5-Funktionalitäten ist der LocalStorage. Dabei handelt es sich um einen einfachen Key Value Storage[33], der mit Hilfe der vom Browser angebotenen JavaScript API angesprochen werden kann. Der Speicherplatz variiert je nach Plattform zwischen 5MB und 25MB[34]. Dies soll die gespeicherten GebärdenSchrift-Vokabeln sichern.

Die Ansteuerung des LocalStorage verläuft recht simpel:

```
1 /* -----------  Speichern eines Eintrags ----------- */
2 localStorage.setItem(„key“, „wert“);
3 /* -----------  Auslesen eines Eintrags ----------- */
4 localStorage.getItem(„key“);
```

Zum Speichern von mehreren Werten unter einem Key, also Indexwert, kann die JSON (Java Script Object Notation) Repräsentation eines JavaScript-Arrays genutzt werden.

```
1 var array = [„foo“, „bar“];
2 var output = JSON.stringify(array); // output: "["foo","bar"]"
3 var input = JSON.parse(output);     // input: ["foo","bar"]
```

Dabei wird das JavaScript-Array beim Speichern unter Nutzung der Funktion `Stringify` (Zeile 2) in einen String umgewandelt. Um von der String-Repräsentation wieder zurück zu einem JavaScript-Array zu kommen, wird die Methode `Parse` (Zeile 3) genutzt.

[33] Auf Skalierbarkeit orientierte Datenbank mit Indexierung nach dem Schema f(Key) = Value (vgl. hierzu [LocalStorage]).

[34] Auf Android ab Version 2.0, iOS ab Version 3, BlackBerry ab Version 5.0 und Windows Phone ab 7.

4.3.2 Mobile GWT

Das mgwt (Mobile GWT) Framework erlaubt die Anwendungsentwicklung für mobile Plattformen mit dem Google Web Toolkit[35]. Entwickelt wird mgwt von Daniel Kurka. Er ist auch an der Entwicklung von GWT als *GWT Steering Committee*[36]-Mitglied beteiligt.

Mobile GWT ermöglicht nativ aussehende Anwendungen auf Grundlage einer einzigen Codebasis zu entwickeln. Zu diesem Zweck gibt es für jedes Bedienelement Designs in der Art von iOS, Android und BlackBerry. Diese Designs sind größtenteils in CSS implementiert, brauchen somit keinen Einsatz von pixelbasierten Formaten. Diese Form der Darstellung ist speicherschonender und ermöglicht dadurch eine Performancesteigerung. Gerade die Übertragung von Daten in mobilen Netzen spielt bei der Performance-Optimierung eine große Rolle und wird im *Abschnitt 4.4.2* genauer behandelt. Neben den verschiedenen Designs bietet mgwt auch eine Gestenerkennung und die Optimierung auf Touch-Bedienung.

Eine weitere angebotene Funktionalität ist die Generierung des HTML5 Offline Manifests. Dieses ermöglicht eine Webanwendung ohne Netzanbindung zu nutzen. Zusätzlich bietet Mobile GWT die Integration von Phonegap, wie in *Abschnitt 3.2.1* beschrieben, um auf die nativen Schnittstellen zuzugreifen (vgl. hierzu [MGWT]).

4.4 Hindernisse bei der Implementierung

Bei der Bewertung der verschiedenen Möglichkeiten der plattformunabhängigen Anwendungsentwicklung für mobile Plattformen mit Webtechnologien gibt es einige unbedingt zu beachtende Einschränkungen im Bereich der WebApp-Entwicklung. Hin und wieder sind es ganz allgemein Hindernisse bei der Webentwicklung wie der *Same Origin Policy*, wohingegen das Thema Performance einen direkten Bezug zu den Einschränkungen mobiler Geräte hat.

---

[35] Das Google Web Toolkit ist ein Framework, welches Entwicklern gestattet, komplexe Webanwendungen komplett in Java zu schreiben. Dabei wird der klientenseitige Code durch den Java-zu-JavaScript-Compiler in JavaScript überführt. Es ist somit möglich Klientenseite und Serverseite in Java zu entwickeln.

[36] Das GWT Steering Comittee ist eine von Google ausgewählte Gruppe von Entwicklern, die sich um die Zukunft des Google Web Toolkits kümmern. Sie verwalten die Codebasis und kümmern sich um die Veröffentlichung neuer Versionen.

4.4.1 Same origin policy (SOP)

Die *Same origin policy* ist ein von den Web-Browsern implementierter Schutzmechanismus. Die SOP verhindert das Nachladen von Inhalten mit JavaScript, CSS oder ActionScript aus Quellen, die von der origin (deutsch: Ausgangspunkt) abweichen. Bei SOP handelt es sich um einen der elementaren Sicherheitsmechanismen heutiger moderner Web-Browser, mit dem Angriffe, wie z.B. Cross Site Scripting (XSS)[37], verhindert werden sollen.

Um die Funktionsweise zu erläutern, muss zuerst die Zugehörigkeit zu einer origin geklärt werden. Ein Beispiel aus [RFC6454] für zu derselben *origin* gehörenden Adressen ist:

http://example.com/
http://example.com:80/
http://example.com/path/file

Wohingegen folgende Adressen nicht zu derselben *origin* gehören:

http://example.com/
http://example.com:8080/
http://www.example.com/
https://example.com/
http://example.org/

In der Praxis bedeutet dies für Webanwendungen, dass die Konfiguration der Webserver mit der Konfiguration der Domains und Ports an die Anforderungen der einzelnen Webanwendungen angepasst werden müssen[38].

Um nun jedoch Schnittstellen (APIs), wie sie von den meisten größeren Webseiten angeboten werden, zu nutzen, entstanden zwischenzeitlich einige Umgehungslösungen wie zum Beispiel *JSON with Padding*. Bei JSONP wird anstatt reiner Daten JavaScript-Code zurückgegeben. Dieser ruft eine beim Aufruf angegebene Funktion auf und übergibt ihr die angeforderten Daten ([JSONP]).

---

[37] Nach [XSSHeise] handelt es sich bei XSS um einen Angriff, bei dem versucht wird, mit Hilfe von injiziertem Skriptcode die Webanwendung derart zu manipulieren, dass z.B. falsche Informationen angezeigt werden. Ziel ist häufig, an Passwörter oder andere sensible Daten zu gelangen.

[38] Allgemein ausgeschlossen sind von der SOP-Regelung Webseiten die lokal, also durch das File://-Protokoll aufgerufen werden.

Diese Methode der Umgehung der SOP wird am Beispiel des GebärdenSchrift-Wörterbuches veranschaulicht. Dieses benötigt für die Übersicht, in der nach Gebärden gesucht werden kann, für ein gegebenes Präfix eine Liste von dieses Präfix beinhaltenden Gebärden. Um die Datenmenge klein zu halten, gibt die vom Webserver des Delegs-Projekts zur Verfügung gestellte Schnittstelle ein Array von Strings in seiner JSON-Repräsentation zurück. Ein Aufruf der Schnittstelle mit dem Präfix `„hall“` führt somit zu der Rückgabe von:

```
["Halle","Hallo","Hallodri","Halloween","halluzinieren"]
```

Um nun die Daten nutzen zu können, werden diese Daten in einen Funktionsaufruf verpackt. Dafür übergibt man der Web-Schnittstelle neben dem Präfix noch einen Funktionsnamen. Zum Beispiel mit dem Funktionsnamen *rufMichAuf* würde die Rückgabe wie folgt aussehen:

```
rufMichAuf(["Halle","Hallo","Hallodri","Halloween","halluzinieren"])
```

Nun wird dieser Aufruf als Script in die Web-Anwendung eingebaut:

```
<script type=“text/javascript“
  src=“http://delegs.com/.../signitems?prefix=hall&callback=rufMichAuf“
</script>
```

Der letzte Schritt ist das Schreiben der die Daten verarbeitenden Funktion *rufMichAuf*:

```
function rufMichAuf(data){
   ... }
```

Neben JSONP gibt es einige andere Methoden zur Umgehung der SOP. Durch *Posting messages*[39] soll auch in HTML5 eine offizielle Methode zur Klienten-Fremdserver-Kommunikation spezifiziert werden.

[39] Weitere Informationen zu den Posting Messages werden durch die WHATWG auf http://whatwg.org/specs/web-apps/current-work/multipage/web-messaging.html#posting-messages zur Verfügung gestellt.

4.4.2 Performance-Optimierung

Das Thema der Performance wird meist erst thematisiert, wenn diese auffällig schwach ist. Mobile Geräte sind durch ihre Akkus und weniger leistungsfähige Rechenkerne eingeschränkt und neigen viel eher als Heimrechner zu Performance-Einbrüchen. Gerade bei der Entwicklung von nicht nativen Anwendungen kommt es häufig zu ruckelnden Animationen oder nicht reagierenden Bedienelementen. Auch Themen wie Datenverarbeitung und die Entwicklung von 3D-Anwendungen sind keine unbedeutenden Probleme bei der plattformunabhängigen Anwendungsentwicklung mit Webtechnologien. Ein Beispiel für eine große Anwendung, die wegen Performance-Problemen nun als native Anwendung neu entwickelt werden soll, ist bei der Firma Facebook zu finden. Bei der Facebook-App klagten die Nutzer in den 21.000 Ein-Stern-Bewertungen von insgesamt 38.000 Bewertungen für die iPhone-Version über ruckelnde Benutzeroberflächen und häufige Abstürze (vgl. [FacebookApp]). Facebook setzte auf eine selbst entwickelte Lösung, die wie PhoneGap die Webinhalte darstellte und Zugriff auf die Kamera und andere Schnittstellen ermöglichte.

Der Begriff Performance beschreibt üblicherweise die Leistungsfähigkeit einer Software in Bezug auf die zur Verfügung stehende Rechenleistung, also wie viele Rechenschritte in einem vorher definierten Zeitraum abgearbeitet werden können. Auf mobilen Plattformen nutzen Anwendungen vermehrt das Internet als Datenquelle oder Datenspeicher. Dieser Sachverhalt bedingt, dass der Leistungsfähigkeit der Netzwerkverbindung eine größere Rolle zukommt. Aus diesem Grund wird der Abschnitt der Performance-Optimierung in die zwei Unterabschnitte der *Rendering-Optimierung*[40] und der *Netzwerkoptimierung* unterteilt.

Bei der **Rendering-Optimierung** für Webanwendungen ist der entscheidende Faktor die Verarbeitungszeit für das darzustellende Dokument durch den Web-Browser – also die Zeit, die der Web-Browser benötigt, um dieses anzuzeigen. Zur Optimierung dieser Zeit gibt es einige in der Praxis bewährte Maßnahmen, die im folgenden Text beschrieben werden. Eine der ersten Maßnahmen besteht darin, die CSS-Definitionen in den <head>-Tag des HTML-Dokuments zu schreiben. Da dem Web-Browser alle Styles nach dem Laden des Kopfbereichs (englisch: head) bekannt sind, gibt es kein unnötiges Umorganisieren der Seitenelemente ([Greslehner, S.5]). Im weiteren Verlauf sollte bei jedem <img>-Tag eine Breite und Höhe definiert werden. Der Web-Browser erzeugt beim Rendern des Dokuments eine Vorabversion, in welche die jeweiligen Elemente

[40] Rendering bedeutet in diesem Zusammenhang das Darstellen der durch das HTML und CSS definierten Elemente und Styles im Browser.

eingefügt werden. Sind die Abmesssungen zu diesem Zeitpunkt nicht bekannt, löst das spätere Einfügen des Bildes eine Neuorganisation der Elemente aus und somit ein Neuzeichnen (englisch: repaint) der gesamten Oberfläche ([Greslehner, S.5]). Eine weitere Vorgehensweise, die durch die Weiterentwicklung von CSS möglich wird, ist die Nutzung von CSS-Transitions statt JavaScript für Animationen. Die CSS-Transitions sind von den jeweiligen Web-Browsern in nativen Algorithmen implementiert und können somit von den Grafikbausteinen (Hardware) beschleunigt werden. Von Google 2011 durchgeführte Tests ergaben, dass jedes Kilobyte JavaScript-Code auf aktuellen mobilen Geräten ungefähr 1 Millisekunde an Verarbeitungszeit benötigt. Hochgerechnet auf die übliche Größe der JavaScript-Dateien sind das 250-300ms, die nur für die Verarbeitung des JavaScript-Codes benötigt werden. Aus diesem Grund werden zum Beispiel bei der Mail-Web-Anwendung von Google einige funktionale Teile erst verarbeitet (englisch: parsed), wenn diese genutzt werden (vgl. [GoogleSpeed]). Diese Methode des verzögerten Auswertens gehört zu den neueren Vorgehensweisen. Dabei wird JavaScript-Code als Kommentar oder in String-Literalen innerhalb der JavaScript-Datei gespeichert und zu einem späteren Zeitpunkt mit dem Befehl `eval()` ausgewertet bzw. ausgeführt.

Bei der Umsetzung des GebärdenSchrift-Wörterbuchs brauchten diese Aspekte nicht berücksichtigt werden, da der von dem GWT-Compiler generierte JavaScript-Code schon stark optimiert ist. Der Compiler bedient sich dabei der oben beschriebenen Methoden und einiger sich auf die Optimierung von JavaScript-Code beziehender Methoden, die in Büchern wie „*High Perfomance JavaScript*“ von Nicholas C. Zakas beschrieben werden. Um die gute Performance des GWT generierten Codes zu veranschaulichen, kann der Benchmark [GWTBench] herangezogen werden. Dieser vergleicht die Performance üblicher Frameworks wie jQuery mit GWT in Bezug auf die CSS-Selector-Performance, die bestimmte Seitenelemente mit Hilfe von CSS-Selectoren identifiziert. Diese Methode HTML-Elemente zu identifizieren und umzuorganisieren wird häufig genutzt. Ein bislang üblicher Seitenwechsel von einem HTML-Dokument zu einem anderen wird immer seltener. Meist wird durch einen Klick auf einen Link nur der Inhalt durch einen nachgeladenen Inhalt ausgetauscht. Dieses Konzept des Nachladens bezeichnet man als Ajax[41]. Je nach Leistungsfähigkeit des Systems liegen die Ergebnisse des Benchmarks unterschiedlich stark auseinander, in jeder Konstellation ist jedoch GWT auf dem ersten Platz.

[41] *Asynchronous JavaScript and XML* (Ajax) bezeichnet ein Konzept, bei dem asynchron Inhalte nachgeladen werden ohne ein Neuladen der Webseite auszulösen.

Im Bereich der **Netzwerkoptimierung** gibt es noch einige weitere Optimierungsmöglichkeiten. Bei der Netzwerkoptimierung wird die Lade- oder Reaktionszeit (beim Nachladen) bei Netzwerkzugriffen von Webanwendungen optimiert. Zuallererst werden geeignete Methoden zur Messung der Performance benötigt, um eventuelle Probleme aufzuzeigen. Eines der effektivsten Tools stellen dafür viele Web-Browser[42] selbst zur Verfügung. Dies ist das entsprechende Entwicklertool, das neben der üblichen Ansicht des Quellcodes auch Werkzeuge für die Ressourcen-Übersicht, Netzwerkverkehr-Analyse, CPU-Lastanalyse und CSS-Selector-Analyse bietet. Zusätzlich können mit manchen Browsern, wie z.B. Chrome Audits (deutsch: Überprüfungen) durchgeführt werden. Hierbei wird die Webseite im Rahmen einer vom Browserhersteller definierten Vorgabe getestet und dann zielgerichtete Maßnahmen zur Optimierung vorgeschlagen.

Eines der größten Probleme für Webanwendungen sind die durch die genutzten Übertragungstechnologien hervorgerufenen Latenzen. Diese Verzögerungen treten bei jeder Anfrage an den Server auf.

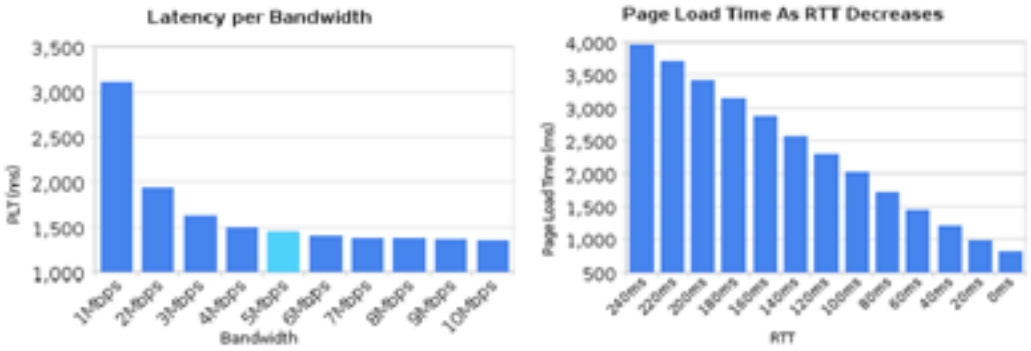

Abbildung 7: Latenzen in Abhängigkeit zur Übertragungsgeschwindigkeit (Quelle: [Latency])

Die Betrachtung des Latenzverhaltens in Abhängigkeit zur Übertragungsgeschwindigkeit auf Abbildung 7 ergibt, dass der Faktor Übertragungsgeschwindigkeit ab der gebräuchlichen Breitbandgeschwindigkeit von 5-6 Mbit/s fast keinen Einfluss mehr auf die Latenzen nimmt. Durch die immer schneller werdende Vermittlungstechnik wird der begrenzende Faktor die Lichtgeschwindigkeit[43] (vgl. hierzu [Latency]).

Ein anderer Sachverhalt liegt bei Technologien wie EDGE oder UMTS vor. Beides sind bei fast allen aktuellen Geräten unterstützte Mobilfunkstandards.

---

[42] Chrome, Safari, Internet Explorer, Firefox und Opera bieten die Entwicklertools direkt in der Standardinstallation an. Diese müssen meistens nur über die Einstellungen aktiviert werden.

[43] Die Datenübertragung durch Lichtwellenleiter wird für transatlantische Verbindungen aber neuerdings auch für die Anbindung direkt beim Kunden genutzt.

Der zu Sprint/USA gehörende Mobilfunkprovider Virgin Mobile schreibt zum Thema Latenzen auf seiner Webseite:

> On the Sprint 3G network users can expect to experience average speeds of 600Kbps - 1.4Mbps download and 350Kbps - 500Kbps upload with an average latency of 400ms. (Quelle: [VirginMobile])

Ihr UMTS-Netz weist also eine mittlere Latenz von 400ms auf und dies auch nur, wenn die Funktechnik des Geräts nicht im Standby-Modus ist. Der Aufwachvorgang mit Kanalsuche dauert etwa 1000-2000ms (vgl. hierzu [Latency]). Die jeweiligen Zeiten bis das Gerät in den Standby-Modus geht, hängen von den jeweiligen Providern ab. Die Abbildung 5 von AT&T zeigt die für ihr Netz gültigen Werte von 5 Sekunden bis Halb-Standby und 12 Sekunden bis Standby. Es ist anzunehmen, dass diese Werte sich nicht stark von den Werten der hiesigen Anbieter unterscheiden.

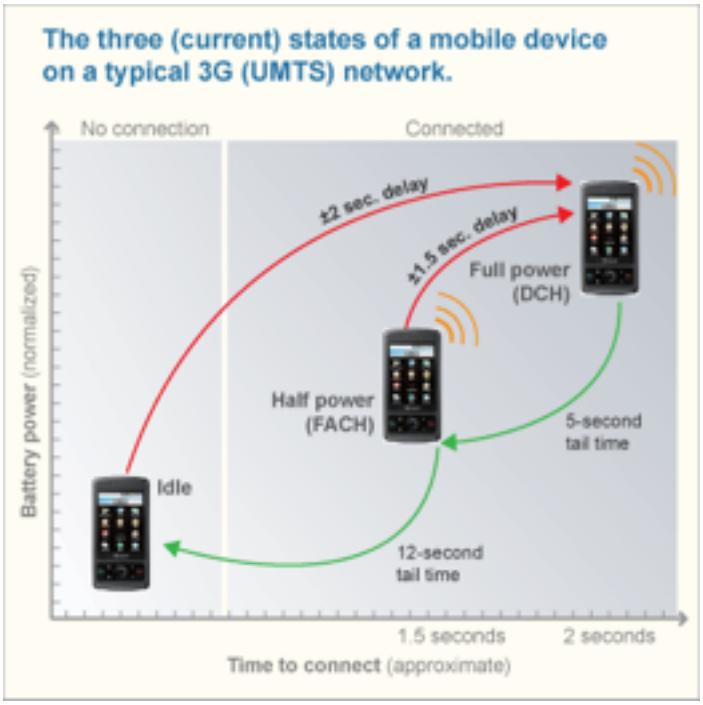

Abbildung 8: Zustände eines mobilen Geräts im UMTS-Netz (Quelle: [UMTS])

Ein Gerät im Halb-Standby hat dementsprechend etwa 1500ms zusätzliche Latenz bis es eine Verbindung wiederhergestellt hat. Um dem Problem der hohen Latenzen entgegenzuwirken, wird versucht, die Anzahl der Anfragen an den Webserver zu reduzieren.

Aus diesem Grund haben mobile Webanwendungen meist nur ein HTML-Dokument. Die Navigation und andere Änderungen werden mit Hilfe von Ajax, also dem dynamischen Nachladen unter Nutzung von JavaScript, umgesetzt. Wenn eine große Anzahl von Bildern geladen werden, können diese zu einem großen Bild zusammengefasst und

mit Hilfe von CSS Sprites wieder einzeln dargestellt werden ([YahooPerformance]). Auch ist es möglich, Bilder als Base64-kodierte Daten in das HTML Dokument einzubinden und somit eine weitere Abfrage beim Webserver einzusparen. Ebenso kann das CSS in das HTML-Dokument eingebunden werden (englisch: inlinen). Dieses Verfahren des Zusammenfassen findet häufig im Zuge des Minifizierungs-Vorgangs statt. Dieser Vorgang wird zur Minimierung von CSS und JavaScript-Dateien durchgeführt. Dabei werden überflüssige Inhalte wie Kommentare, Leerzeilen oder zu lange Variablennamen zwecks Einsparung von einigen Bytes entfernt.

Eine weitere Optimierungsmöglichkeit ist das Komprimieren der Daten mit Hilfe von gzip. Diese Optimierung muss serverseitig am Webserver konfiguriert und vom Web-Browser unterstützt werden[44]. Durch das Komprimieren der Daten konnte beim GebärdenSchrift-Wörterbuch eine Reduzierung des Datenaufkommens von durchschnittlich 40% erreicht werden. Die Verlagerung des JavaScripts an das Ende und der CSS-Stylesheets an den Anfang des HTML-Dokuments beschleunigt das initiale Laden. Meist wird der JavaScript-Code nicht für die Darstellung der Startseite benötigt. Vielmehr ist seine Aufgabe, auf Aktionen zu reagieren. Durch die Verlagerung ans Ende kann die Seite grafisch fertig aufgebaut werden, sodass der Benutzer schneller eine funktionstüchtig aussehende Webseite erhält. Auch das asynchrone Nachladen von Inhalten, die für zukünftige Aktionen benötigt werden, kann helfen, die gefühlte Performance zu erhöhen. Unter dem Begriff des *anticipated preloading* (deutsch: vorweggenommenes Vorladen) verbirgt sich eine Vorgehensweise, bei der versucht wird, zukünftige Aktionen des Benutzers vorherzusehen und dafür entsprechende Daten vorzuladen. Dieses Verfahren ist eine explizite Netzwerkoptimierung und muss von dem im vorherigen Abschnitt beschriebenen Verfahren des verzögerten Auswertens unterschieden werden. Ein Beispiel ist die Webseite von XING, bei der durch ein Fokus-Event auf dem Login-Feld einige der für die nächste Seite benötigten Daten asynchron vorgeladen werden. Dies führte zu einer Reduktion der Ladezeit von 10% (vgl. hierzu [XINGPrefetching]). Zum Schluss bleibt noch die Möglichkeit, die Entfernung zum Benutzer, also die Hops[45], zu reduzieren. Dies kann entweder manuell, also durch das Anmieten von Servern in den jeweiligen Regionen und einem Loadbalancer, der sich um die Weiterleitung der Benutzer an den richtigen Server kümmert, oder durch sogenannte Content Delivery Net-

[44] Alle aktuellen Web-Browser auf Desktop und Mobile-Geräten unterstützen gzip und somit die HTTP-Kompression

[45] Nach [RSchreiner, S. 83] bedeutet ein Hop soviel wie „Etappe“ und beschreibt die Anzahl der Zwischenstationen die ein Datenpaket weitergeleitet wird bis es sein Ziel erreicht.

works (CDN)[46] erreicht werden. Die CDN bieten als Service an, sicherzustellen, dass die Daten einen kurzen Weg zum Empfänger zurücklegen.

Auch im Bereich der Netzwerkoptimierung nimmt mgwt bzw. GWT einige Arbeit ab. Bis auf das Packen auf Serverseite und das Vorladen von Daten auf Grundlage von vorhergesagtem Nutzerverhalten kümmert sich GWT meist automatisch um jeden der oben benannten Optimierungsschritte.

4.4.3 Fehlersuche

Fehlersuche (englisch: Debugging) beschreibt die Handlungsweise, mit der ein Programm, um Fehler aufzufinden, mit Hilfe geeigneter Werkzeuge analysiert wird. Für die Entwicklung von Webanwendungen stellen die Web-Browser-Hersteller mit ihren Entwicklertools für die Desktop-Version ihrer Web-Browser einige nützliche Werkzeuge zur Verfügung (siehe *Abschnitt 4.4.2*). Bei der Fehlersuche, also dem Testen der mobilen Anwendung, führt kein Weg an der entsprechenden Hardware vorbei. Simulatoren und Emulatoren können in Kombination mit Netzwerkfehler-Simulatoren[47] einen ersten Eindruck von dem Verhalten der Webanwendung in verschiedenen Netzwerksituationen vermitteln, ersetzen jedoch keinen Einsatz unter realen Bedingungen. Besonders bei Webanwendungen ist der Test unter realen Bedingungen wichtig, da Mobilnetzprovider Optimierungen zur Reduzierung der genutzten Datenmenge anstreben. Ein solcher Provider in Deutschland ist zum Beispiel Vodafone. Hier werden die Adressen in den IMG-Tags derart geändert, dass auf eigene Server verwiesen wird, auf denen die Bilder in geringerer Qualität vorliegen. Auch am Quellcode der Seite wird optimiert. Wie bei der Minifizierung werden Leerzeilen und andere unnötige Teile entfernt, teilweise wird sogar ein eigener JavaScript-Code injiziert (vgl. hierzu [ZDNetMobile]). Diese Eingriffe in die Daten einer Anwendung führen in vielen Fällen zu einem unvorhersehbaren Verhalten einer Webanwendung und somit ist ein Test auf den entsprechenden Geräten und mobilen Netzen unvermeidlich.

---

[46] Die CDN-Knoten sind großflächig verteilt, oft sitzen sie nah an den Basisnetzen (Backbones) der Internetprovider und sind durch Caching und anderer Mechanismen auf eine hohe Performance ausgelegt. Zu den größten CDN gehört Akamai, welcher mit über 100.000 Servern in 78 Ländern vertreten ist (siehe [Akamai, S.13 und S.25]).

[47] Viele Entwicklungswerkzeuge bieten solche Simulatoren, die es erlauben Netzwerkeigenschaften wie Paketverluste oder Bandbreite selbst zu definieren, um den realen Netzwerkbedingungen nahe zu kommen.

Um eine Webanwendung auf dem Gerät testen zu können, bieten Hersteller wie Apple und Android mit ihren Entwicklungswerkzeugen entsprechende Tools an[48]. Wird eine Lösung für alle Plattformen benötigt, bieten sich Lösungen wie z.B. *Weinre* (Abkürzung für „**We**b **In**spector **Re**mote“) an. Weinre gestattet es, mit Hilfe eines HTTP-Servers die mobile Webanwendung zu erproben. Dabei wird JavaScript-Code, welcher mit dem HTTP-Server kommuniziert, in die Anwendung eingebaut und so wird es möglich, auf einer Entwicklungstools ähnelnden Oberfläche mit der Webanwendung zu interagieren.

Bei der Entwicklung des mobilen GebärdenSchrift-Wörterbuches wurde mit dem GWT-Debugger gearbeitet. Dieser bietet durch seine Web-Browser-Erweiterung (für die Desktop-Version der Web-Browser) die Möglichkeit, direkt mit dem Java-Code zu arbeiten. Zum Testen auf mobilen Geräten wurde Weinre genutzt.

4.5 Benutzertests

Bei den Benutzertests war ein beherrschendes Thema die Performance. Die kleinsten Verzögerungen wurden von den Benutzern bemerkt und beanstandet.

Dabei standen als Testgeräte ein Motorola Defy, Apple iPhone 3GS, Apple iPhone 4 und ein Apple iPhone 4S zur Verfügung. Die zu testende WebApp lief auf dem Defy und iPhone 3GS stockend, auf dem iPhone 4 flüssig und auf dem iPhone 4S sehr flüssig. Auf dem iPhone 3GS wurde die Anwendung als kaum zu nutzen bezeichnet, wohingegen bei der iPhone 4 und iPhone 4S Version fast nur zusätzliche Funktionalitäten gewünscht wurden.

Dies Beispiel zeigt wie anspruchsvoll Benutzer bei der Performance von Anwendungen sind. Doch zeigen auch die Performanceunterschiede zwischen dem iPhone 3GS, iPhone 4 und iPhone 4S, die immer ein Jahr nach dem vorherigen Gerät veröffentlich wurden, wie schnell der Abstand zwischen den verschiedenen Entwicklungsansätzen, im Bezug auf die Perfomance, durch die sich verbessernde Hardware verringert wird.

[48] Android stellt die Funktionalität mit Hilfe seiner Android Debugging Bridge zur Verfügung, bei iOS ist dies ab Version 6 mit XCode möglich.

5

Zusammenfassung

5.1 Zusammenfassung der Möglichkeiten zur plattformunabhängigen Entwicklung von mobilen Anwendungen

Im Verlauf der vorangegangenen Abschnitte wurden WebApps und die hybriden Ansätze vorgestellt und in Beziehung zu den nativen Anwendungen gebracht. Um die Vorzüge und Nachteile im direkten Vergleich zu betrachten, werden diese Ansätze zur mobilen Anwendungsentwicklung nun den folgenden Kriterien gegenübergestellt. Zugrunde gelegt werden Plattformunabhängigkeit, Distributionswege, Performance, nutzbare Schnittstellen, Bedienbarkeit und Entwicklungskosten. Jedes Kriterium wird reflektiert und so eine Entscheidungshilfe bei der Entwicklung einer mobilen Anwendung zur Verfügung gestellt.

Kriterium	WebApp	Wrapper-Hybrid	Interpreter-Hybrid	Native Anwendung
Plattformunabhängigkeit	+	+	0	-
+: Es kann für einen Großteil der mobilen Plattformen mit derselben Codebasis entwickelt werden 0: Es kann für mind. zwei der drei großen Plattformen mit derselben Codebasis entwickelt werden - : Der Code kann nur für eine Plattform genutzt werden				
Distributionswege	-	+	+	+
+: Kann über die jeweiligen Distributionsplattformen vertrieben werden - : Kann nicht über die jeweiligen Distributionsplattformen vertrieben werden				
Performance	-	-	0	+
+: Bedienelemente reagieren schnell, flüssige Animationen, komplexe Berechnungen möglich 0: Bedienelemente reagieren schnell, flüssige Animationen - : Bedienelemente reagieren langsam, ruckelnde und unterbrechende Animationen				

Kriterium	WebApp	Wrapper-Hybrid	Interpreter-Hybrid	Native Anwendung
Schnittstellen	0	+	+	+
+: Zugriff auf alle Schnittstellen zur Hardware 0: Zugriff auf wenige der Schnittstellen zur Hardware - : Kein Zugriff auf die Schnittstellen zur Hardware				
Bedienbarkeit	-/0	0	+	+
+: Anwendung verhält sich wie auf der jeweiligen Plattform üblich 0: Anwendung verhält sich meist wie auf der jeweiligen Plattform üblich - : Anwendung weist ein eigenes Bedienkonzept auf oder scheitert an der Nachahmung				
Entwicklungskosten	+	+	0	-
+: Anwendung wird bei jeder weiteren unterstützten Plattformen kaum teurer 0: Anwendung wird bei jeder weiteren unterstützten Plattform moderat teurer - : Anwendung wird bei jeder weiteren unterstützten Plattform signifikant teurer				

Tabelle 3: Kriterienkatalog zur Entwicklung von mobilen Anwendung mit Webtechnologien

5.1.1 Kriterium: Plattformunabhängigkeit

Die Plattformunabhängigkeit steht in Zusammenhang mit der Arbeit, die geleistet werden muss, um die Anwendung für mehrere Plattformen zur Verfügung zustellen. Ein hierfür wichtiger Faktor ist die Wiederverwendbarkeit des Programmcodes (englisch: *code reuse*). WebApps lassen sich, abgesehen von minimalen Anpassungen für den jeweiligen Web-Browser, mit geringem Aufwand wieder verwenden.

Dies gilt auch bei den Wrapper-Hybriden. Diese benötigen nur eine Änderung des äußeren Containers, die beinhaltete WebApp muss nicht verändert werden. Interpreter-Hybriden abstrahieren wie Wrapper-Hybride von der jeweiligen Plattform, sodass der Anwendungscode nicht speziell angepasst werden muss. Jedoch werden im Fall des derzeit bekanntesten Interpreter-Hybriden Titanium weniger Plattformen, nur iOS und Android, unterstützt. Native Anwendungen sind per Definition auf ihre Plattform festgelegt und es lassen sich bestenfalls einzelne Module auf einer anderen Plattform wieder benutzen. Die Abbildung 4 veranschaulicht die Einordnung der verschiedenen Ansätze unter dem Kriterium Plattformunabhängigkeit.

Abb. 4: Einordnung von WebApp, Wrapper-Hybride, Interpreter-Hybride

5.1.2 Kriterium: Distributionswege

Die Verteilung von mobilen Anwendungen über die entsprechenden Distributionsplattformen ist in vielen Marketingstrategien ein wichtiger Bestandteil geworden. Vielen Nutzern ist die Platzierung von WebApps auf den Homescreen unbekannt. In welchem Umfang eine WebApp in der Praxis regelmäßig genutzt wird ist fraglich. Dieses Problem haben Wrapper-Hybriden und Interpreter-Hybriden nicht, da die von ihnen produzierten nativen Anwendungen über die Distributionsplattformen vertrieben werden können.

5.1.3 Kriterium: Performance

Gegenwärtig gehört die Performance neben den benötigten Schnittstellen zu den wichtigsten Auswahlkriterien für den Ansatz bei der Entwicklung einer mobilen Anwendung. Die Entwicklung von aufwändigen Spielen oder von viel Leistung benötigenden Anwendungen ist mit WebApps kaum zu bewerkstelligen. Auch die Hybriden können hierbei nicht mithalten. Allerdings leisten die Interpreter-Hybriden mit der Nutzung von nativen Bedienelementen gute Arbeit bei der Verbesserung der Performance von Listen und anderen typischen Benutzeroberflächen-Elementen. Die Wrapper-Hybriden und WebApps ähneln sich im Bereich der Performance und weisen beide eine geringere Leistungsfähigkeit gegenüber der von nativen Anwendungen auf. Gerade die Wrapper-Hybriden haben es im Bereich der Performance auf iOS besonders schwer. Da Webinhalte innerhalb von nativen Anwendungen nicht mit der schnelleren Nitro-Webengine[49] verarbeitet werden, kann es vorkommen, dass eine mit Hilfe von Safari geöffnete WebApp schneller ist als eine in einen Wrapper-Hybriden verpackte WebApp. Die Möglichkeit, Daten in die Anwendung einzubinden und damit lästiges Nachladen zu verhindern, kann diesem Umstand etwas entgegensetzen. Die Weiterentwicklung der mobilen Geräte mit immer leistungsfähigeren Multi-Rechenkernen und Grafikbausteinen schafft andere Voraussetzungen und kann den Leistungsunterschied zwischen WebApp und nativer App im Bezug auf die Performance verringern.

5.1.4 Kriterium: Schnittstellen

Die benötigten Schnittstellen definieren in der Praxis die zu verwendende Technologie. Zum Beispiel kann zurzeit beim Zugriff auf die Kamera keine reine WebApp benutzt werden. Zwar gibt es schon erste Spezifikationen für HTML5, hierfür eine API bereitzustellen, jedoch ist der Zeitpunkt einer Umsetzung noch nicht absehbar. Die derzeit auf jeder mobilen Plattform funktionierende Funktionalität ist LocalStorage zum Speichern

[49] Die Nitro-Webengine beschleunigt die Verarbeitung von JavaScript-Code nach [Nitro] um den Faktor zwei. Ihre Nutzung kann allerdings auch zu Sicherheitsproblemen führen und wird aus diesem Grund nicht innerhalb von nativen Apps genutzt.

von Daten und die Ermittlung der derzeitigen Position über die Geolocation-Schnittstelle. Für Wrapper-Hybriden und Interpreter-Hybriden stehen theoretisch alle Schnittstellen zur Verfügung. Das Framework muss nur die entsprechenden Module enthalten, die diese Schnittstellen ansprechen und an die Kommunikations-Schnittstelle des Frameworks anbinden. Diese Module können durch die pluginbasierte Architektur von Frameworks wie PhoneGap oder Titanium von den Entwicklern selbst entwickelt werden. Häufig wird eine solche eigene Entwicklung erforderlich, da die Frameworks keine Garantie, weder für das Funktionieren jeder Schnittstelle noch für eine zeitnahe Implementierung geben. Entwickler, die native Anwendungen programmieren, haben in diesem Punkt einen Vorteil und können leichter auf die Änderungen der einzelnen Plattformen reagieren.

5.1.5 Kriterium: Bedienbarkeit

Viele der GUI-Frameworks werben mit nativ aussehenden Oberflächen. Der maßgebende Faktor ist jedoch deren Bedienbarkeit (englisch: usability). Gerade WebApps und Wrapper-Hybriden scheitern bei ihrer Imitation von nativen Anwendungen oft an Kleinigkeiten wie der Positionierung von Elementen oder der nicht flüssigen Reaktion auf Benutzereingaben. Es erfordert viel Arbeit, um das Verhalten einer nativen Anwendung mit dem Gebrauch von Webtechnologien nachzubilden. Aus diesem Grund bieten Frameworks wie jQuery Mobile und Sencha Touch annähernd nativ erscheinende, aber nicht komplett nachgebildete Bedienoberflächen. Interpreter-Hybriden nutzen die nativen Elemente der jeweiligen Plattformen und weisen somit das für die jeweilige Plattform erwartbare Bedienoberflächen-Verhalten auf.

5.1.6 Kriterium: Entwicklungskosten

Die Entwicklungskosten setzen sich aus zwei Faktoren zusammen, den Initialkosten, d.h. den Kosten für die komplette Entwicklung einer mobilen Anwendung und den Portierungskosten, die für das Überführen der Anwendung auf eine andere Plattform fällig werden. Nach den Zahlen von [WebAppKosten] kostet die Entwicklung einer WebApp etwa 60-70% im Verhältnis zu einer nativen App. Die Portierung auf eine weitere Plattform beträgt bei nativen Apps etwa 30% ihrer Initialkosten. Grundsätzlich richten sich die Initialkosten nach der Komplexität der Anwendung. Eine simple native Informations-App, die kaum Funktionalität aufweist, startet etwa bei 2.500€. Für WebApps und die Hybriden fallen die Portierungskosten nicht komplett weg, sind jedoch minimal.

5.1.7 Zusammenfassung und Ausblick

Die verschiedenen Methoden zur plattformunabhängigen Anwendungsentwicklung für mobile Plattformen unter Verwendung von Webtechnologien wurden in dieser Arbeit verglichen. Die dafür benötigtenTechnologien wurden in *Abschnitt 3* vorgestellt und auf die jeweiligen Eigenheiten eingegangen. Grundlegend offensichtlich wurde, dass WebApps hauptsächlich Webanwendungen sind, die durch Meta-Tags und entsprechende GUI-Frameworks für mobile Geräte angepasst werden. Die Absicht, eine native Oberfläche überzeugend zu imitieren, kann von keinem dieser GUI-Frameworks realisiert werden. Am erfolgreichsten sind Frameworks wie jQuery Mobile, die einen eigenen Stil, die Vorteile von vielen Plattformen einbeziehend, entwickelt haben und auf den Versuch, für jede Plattform den nativen Stil zu imitieren, dementsprechend verzichten. Danach wurden die Wrapper-Hybriden begutachtet. Diese funktionieren wie ein Container, der eine WebApp umschliesst und ihr über eine JavaScript-Schnittstelle Funktionen der jeweiligen Hardware zur Verfügung stellt. Einen zusätzlichen Vorteil bietet die Möglichkeit, die auf diesem Wege entstandene App über die für die jeweilige Plattform üblichen Distributionswege zu verteilen. Zuletzt wurden die Interpreter-Hybriden vorgestellt. Bei diesen ist der einzige Zusammenhang mit der üblichen Webprogrammierung die Nutzung von JavaScript als Programmiersprache. In dieser wird mit nativen Elementen gearbeitet und so ein Performancevorteil bei Bedienelementen wie z.B. Listen erzielt.

Die Benutzertests aus *Abschnitt 4.5* zeigten, dass die Anwender gerade Performance-Probleme sehr negativ bewerten. Somit wird die Beseitigung bzw. Verhinderung solcher Probleme vorrangig. Hierfür wurden auch in *Abschnitt 4.4.2* viele Performance-Optimierungen vorgestellt. Eine solche Fokussierung auf die Performance ist bei der nativen Anwendungsentwicklung meist nicht erforderlich.

Die Anwendungsfälle für die jeweiligen Technologien können wie folgt zusammengefasst werden. Eine WebApp reicht für die meisten kleinen Apps aus. Werden mehr Zugriffsmöglichkeiten auf das Gerät und die Nutzung der Distributionsplattformen gebraucht, ermöglicht diese PhoneGap, indem es einen Container um die WebApp erzeugt und damit die Kommunikation mit nativen Plugins erlaubt. Wird darüber hinaus mehr Performance bei der Darstellung von Benutzeroberflächen benötigt, können Interpreter-Hybriden wie Titanium verwendet werden. Rechenintensive Anwendungen wie z.B. Spiele werden jedenfalls noch für längere Zeit nicht mit WebApps oder Hybriden möglich sein. Native Anwendungen haben im Bereich der Performance, Schnittstellen und Bedienbarkeit entscheidende Vorteile. Mit der Verbesserung der Rechenleistung von

Smartphones und der Erweiterung der Schnittstellen wird der Leistungsunterschied geringer und so finden WebApps und Hybriden immer größere Verbreitung und Akzeptanz durch die Benutzer. Dennoch ist am jeweiligen Einsatzzweck abzuklären, ob eine native App nicht sinnvoller ist. Zur Entscheidung dieser Frage bieten die oben beschriebenen Kriterien eine Grundlage.

Die Entwicklung des GebärdenSchrift-Wörterbuchs hat ein akzeptables Resultat geliefert. Auch wenn die Benutzer von älteren Geräten Unzufriedenheit mit der Performance äußern, ist die Anwendung auf allen nach 2009 auf den Markt gekommenen Geräten gut zu benutzen. Als maßgebend für die Akzeptanz der Benutzer hat sich das Thema Performance gezeigt. Eine grundlegende Analyse und Wertung von Faktoren, die Einfluss auf die Zufriedenheit von Benutzern mobiler Geräte haben, ist ein interessantes Thema für eine nachfolgende Arbeit.

Literaturverzeichnis

[Akamai] Akamai.com: Annual Report 2011
www.akamai.com/dl/investors/akamai_annual_report_11.pdf (Abgerufen 01.09.2012)

[AppleWebApp] macworld.com: Apple gives developers some iPhone access (Juni 2007)
http://www.macworld.com/article/1058344/iphone_dev.html (Abgerufen 01.09.2012)

[AppStoreDrone] nytimes.com: Apple Rejects App Tracking Drone Strickes (Aug. 2012)
bits.blogs.nytimes.com/2012/08/30/apple-rejects-app-tracking-drone-strikes/
(Abgerufen 01.09.2012)

[BlackBerryRevenue] distimo.com: BlackBerry App World Research (Feb. 2012)
http://www.distimo.com/appstores/app-store/20-BlackBerry_App_World
Abgerufen 30.08.2012

[BuildWith] Builtwith.com: jQuery Usage Trends
http://trends.builtwith.com/javascript/jQuery (Abgerufen 01.09.2012)

[ComMagazinCSS] com-magazin.de: Das ist neu bei HTML5 und CSS3 (Feb. 2012)
http://www.com-magazin.de/praxis/detail/artikel/das-ist-neu-bei-
firefox-10/4/das-ist-neu-bei-html5-und-css3.html?no_cache=1 (Abgerufen 28.08.2012)

[DaringFireball2008] Daringfireball.net: Private (Dez. 2008)
http://daringfireball.net/2008/12/private Abgerufen 30.08.2012

[FacebookApp] nytimes.com: Facebook Plans to Speed Up its iPhone App (Juni 2012)
http://bits.blogs.nytimes.com/2012/06/27/facebook-plans-to-speedup-its-iphone-app/
(Abgerufen 23.08.2012)

[Gartner2012] Gartner.com: Gartner Says Worldwide Sales of Mobile Phones Declined 2 Percent in First Quarter of 2012; Previous Year-over-Year Decline Occurred in Second Quarter of 2009 (2012) http://www.gartner.com/it/page.jsp?id=2017015, Abgerufen 29.08.2012

[GeoLocW3C] w3.org: Geolocation API Specification (Mai 2012)
http://dev.w3.org/geo/api/spec-source.html (Abgerufen 01.09.2012)

[GoogleSpeed] developers.google.com: Optimize for mobile
developers.google.com/speed/docs/best-practices/mobile (Abgerufen 01.09.2012)

[Greslehner] Richard Greslehner: Diplomarbeit zum Thema Performanceoptimierung einer Webanwendung am Beispiel der Livewetten von bet-at-home.com, September 2010, Fachhochschule Digitale Medien in Hagenberg
http://theses.fh-hagenberg.at/system/files/pdf/Greslehner10.pdf (Abgerufen 01.09.2012)

[GWTBench] googlecode.com: GwtQuery Benchmark
http://gwtquery.googlecode.com/svn/trunk/demos/gwtquery.samples.GwtQueryBench
/GwtQueryBench.html (Abgerufen 01.09.2012)

[Heise2011] Heise.de: Bericht - Microsoft verdient knapp eine halbe Milliarde an Android (Sept. 2011)
http://www.heise.de/newsticker/meldung/Bericht-Microsoft-verdient-knapp-eine-
halbe-Milliarde-an-Android-1351900.html, Abgerufen 29.08.2012

[HTML5] Klaus Förster & Bernd Öggl: HTML5 - Leitfaden für Webentwickler, 2011, 1. Auflage, Addison-Wesley

[Industriepraktikumsbericht] David Lammerz: Industriepraktikumsbericht zum Praktikum bei der C1WPS, Universität Hamburg, 2012

[IO12Souders] developers.google.com: Google IO 2012 - High Perfromance HTML5 mit Steve Souders `developers.google.com/events/io/sessions/gooio2012/219/` (Abgerufen 01.09.2012)

[jQueryBrowser] jQueryMobile.com: Mobile Graded Browser Support for jQuery Mobile 1.1.0 `http://jquerymobile.com/gbs/` (Abgerufen 01.09.2012)

[jQueryResources] jQueryMobile.com: Ressources `http://jquerymobile.com/resources/` (Abgerufen 01.09.2012)

[JSONP] json-p.org: Defining Safer JSON-P http://json-p.org/ (Abgerufen 01.09.2012)

[Latency] Ilya Grigorik: Latency - The New Web Perfromance Bottleneck (Jul. 2012) `http://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/` (Abgerufen 01.09.2012)

[LocalStorage] Simon Bin, Jonas Brekle: Key Values Stores (CouchDB, BigTable, Hadoop, Dynamo, Cassandra), Universität Leipzig Abteilung Datenbanken, `http://dbs.uni-leipzig.de/file/seminar_0910_Bin_Brekle.pdf` (Abgerufen 29.08.2012)

[MarkusStäuble2011] Markus Stäuble: Programmieren für iPhone und iPad, dpunkt.verlag, 2011

[MobileConnection] Steve Souders: Making a mobile connection (Sept. 2011) `http://www.stevesouders.com/blog/2011/09/21/making-a-mobile-connection/` (Abgerufen 01.09.2012)

[Nitro] mobilexweg.com: Is Apple trying to hinder PhoneGap and other HTML5 Frameworks[...]? `http://www.mobilexweb.com/blog/apple-phonegap-html5-nitro` (Abgerufen 10.09.2012)

[Oberquelle2011] Folien zu Interaktiondesign-Vorlesung-Kapitel-2: Interaktionsformen, Horst Oberquelle, Universität Hamburg, SoSe 2011

[OHA] OpenHandsetAlliance.com: Open Handset Alliance Release Android SDK (Nov. 2007) `http://www.openhandsetalliance.com/press_111207.html`, Abgerufen 29.08.2012

[Palm] Hardware.net: Palm-Manager - WebOS von Anfang an zum Scheitern verurteilt (Jan. 2012) `http://www.hartware.net/news_53692.html` (Abgerufen 28.08.2012)

[PhoneGapAcceleration] phonegap.com: PhoneGap Documentation - Accelerometer for Version 2.0.0 `http://docs.phonegap.com/en/2.0.0/cordova_accelerometer_accelerometer.md.html` (Abgerufen 01.09.2012)

[PhoneGapDocPlugin] phonegap.com: PhoneGap Documentation - Plugin Development for Version 2.0.0 `http://docs.phonegap.com/en/2.0.0/guide_plugin-development_index.md.html` (Abgerufen 01.09.2012)

[PhoneGapSupport] phonegap.com: Supported Features for Version 2.0.0 `http://phonegap.com/about/feature` (Abgerufen 01.09.2012)

[Podcaster] cnet.com: Apple to Podcaster - No App Store for you (Sept. 2008) `http://news.cnet.com/8301-13579_3-10041572-37.html` (Abgerufen 01.09.2012)

[ReDe] Responsive Web Design, Ethan Marcotte, A Book Apart, 2011

[RetoMeier2009] Reto Meier: Professional Android Application Development, wrox, 2009

[Reuters2011] Reuters.com: Nokia and Microsoft join forces in smartphone war (Feb. 2011) http://uk.reuters.com/article/2011/02/11/uk-nokia-idUKTRE71A10O20110211, Abgerufen 29.08.2012

[ReutersKantar2012] Reuters.com: Google's Android gains share in smartphones-survey (Mai. 2012) www.reuters.com/article/2012/05/15/cellphones-survey-idUSL5E8GEILV20120515 Abgerufen 30.08.2012

[RSchreiner] Rüdiger Schreiner: Computer Netzwerke, 3. Auflage, Hanser Verlag

[RWDArtikel] Ethan Marcotte: Responsive Web Design (Mai 2010) http://www.alistapart.com/articles/responsive-web-design/ (Abgerufen 01.09.2012)

[Sun] Developer.com: „Write once, run anywhere?“ (Feb. 1998), http://www.developer.com/java/other/article.php/601861/Write-once-run-anywhere.htm Abgerufen 12.09.2012

[t3nPhoneGap] Nibuja Steubberg: PhoneGap - Mobile Apps mit HTML5 und JavaScript (Feb. 2012) http://t3n.de/news/phonegap-mobile-apps-html5-368490/ (Abgerufen 01.09.2012)

[Titanium] Kevin Whinnery: Titanium Guides Project - JS Environment (Dez. 2010) developer.appcelerator.com/blog/2010/12/titanium-guides-project-js-environment.html (Abgerufen 01.09.2012)

[UMTS] Alexandre G., Subhabrata S., Oliver S.: A Call for More nergy-Efficient Apps (Apr. 2011) http://www.research.att.com/articles/featured_stories/2011_03/201102_Energy_efficient (Abgerufen 01.09.2012)

[VirginMobile] virginmobileusa.com: Frequently Asked Questions About Network Management http://www.virginmobileusa.com/networkmanagement (Abgerufen 01.09.2012)

[W3CHTML4.1] w3.org: HTML4.01 Specification (Dez. 1999) http://www.w3.org/TR/html401/ (Abgerufen 26.08.2012)

[W3CHtml5Differences] w3.org: HTML5 differences from HTML4 Documentationt http://dev.w3.org/html5/html4-differences/ (Abgerufen 23.08.2012)

[WebAppKosten] suptra.de: Die Vorteile von Web Apps am Beispiel der Prechtl oHG Web App (Juni 2011) app-entwickler-verzeichnis.de/faq-app-entwicklung/10/71-was-kostet-die-entwicklung-einer-app/ (Abgerufen 05.09.2012)

[WindowsPhoneHubFAQ] msdn.com: Windows Phone Dev Center - Submit and sell http://msdn.microsoft.com/en-us/library/windowsphone/help/ff699067 (Abgerufen 11.08.2012)

[XINGPrefetching] Björn Kaiser: Prefteching at XING.com (Feb. 2010) http://devblog.xing.com/frontend/prefetching/ (Abgerufen 01.09.2012)

[XSSHeise] Christiane Rütten, Tobias Glemser: Gesundes Misstrauen - Sicherheit von WebAnwendungen (Jan. 2007) http://www.heise.de/security/artikel/Sicherheit-von-Webanwendungen-270870.html (Abgerufen 30.08.2012)

[YahooPerformance] developer.yahoo.com: Best Practices for Speeding Up Your Web Site http://developer.yahoo.com/performance/rules.html (Abgerufen 01.09.2012)

[ZDNetMobile] Christoph Hochstätter: Internet per UMTS - So fälschen deutsche Provider Webinhalte (Okt. 2009) zdnet.de/41515603/internet-per-umts-so-faelschen-deutsche-provider-webinhalte/ (Abgerufen 01.09.2012)