Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Jason Hung Vuong

Analyse und Bewertung der Integration von Standard-Architekturen für Cross-Plattform Entwicklungsansätze in mobile Apps

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jason Hung Vuong

Analyse und Bewertung der Integration von Standard-Architekturen für Cross-Plattform Entwicklungsansätze in mobile Apps

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. Stefan Sarstedt

Abgegeben am 27.03.2013

Jason Hung Vuong

Thema der Masterarbeit
Analyse und Bewertung der Integration von Standard-Architekturen für Cross-Plattform Entwicklungsansätze in mobile Apps

Stichworte
Alloy, Android, Appcelerator, Apps, Cordova, Cross-Plattform Tools (CPTs), IBM Worklight, Mobile Applikationen, PhoneGap, Quasar, Sencha Touch, Software-Architektur, Titanium

Kurzzusammenfassung
Smartphones definieren ein neues Zeitalter der Handygenerationen und ihre Marktanteile steigen kontinuierlich an. Einige mobile Betriebssysteme, darunter auch Android und iOS, haben sich auf dem Markt etabliert. Da der Markt um mobile Betriebssysteme stark umkämpft ist, versuchen sich auch viele Andere auf dem Markt zu etablieren.

Durch die Vielfalt an mobilen Betriebssystemen ist die Bereitstellung mobiler Applikationen für mehrere Betriebssysteme mit zusätzlichen Kosten, in Bezug auf Zeit und Geld, verbunden. Der Grund hierfür ist, dass jedes mobile Betriebssystem seine eigene Sprache, API und Entwicklungsumgebung hat und Quellcode somit nicht übergreifend wiederverwendet werden kann. Dies beutet, dass für jedes mobile Betriebssystem eine separate Entwicklung erfolgen muss.

Cross-Plattform-Tools sind Werkzeuge, die eine Betriebssystem übergreifende Entwicklung möglich machen und dadurch den Entwicklungsaufwand, sowie die inkrementellen Kosten pro Plattform auf einem Minimum reduzieren können. Dabei nutzen Cross-Plattform-Tools verschiedene Technologieansätze, um dies zu realisieren.

Um eine gewisse Software-Qualität zu erreichen, ist auch für mobile Anwendungen eine Software-Architektur essentiell. Diese Arbeit untersucht ausgewählte Cross-Plattform-Tools hinsichtlich der Möglichkeit Standard-Architekturen zu integrieren. Als Beispiel wurde hier die von Capgemini (ehemals sd&m) entwickelte Softwarearchitektur „Quasar“ gewählt.

In dieser Arbeit geht es im Speziellen um die Analyse und anschließende Bewertung der Integration der Quasar-Client Dialogkomponenten-Architektur für ausgewählte Cross-Plattform-Tools in mobile Apps. Als Cross-Plattform Tools wurde zum einen PhoneGap in Kombination mit Sencha Touch und zum anderen Titanium mit dem Alloy Framework gewählt.

Die Analyse und Bewertungen ergaben, dass das Model-View-Controller Architekturmuster eine gute Basis für die Integration der Quasar-Client Dialogkomponenten-Architektur bildet. Dadurch ist zwar keine vollständige, aber eine Teilintegration gegeben. Für eine komplette Integration ist ein vollständiges Konzept notwendig, welches in zukünftigen Arbeiten erarbeitet werden kann.

Jason Hung Vuong

Title of the paper
Analysis and evaluation of integrating standard architectures for cross-platform development approaches in mobile apps.

Keywords
Alloy, Android, Appcelerator, Apps, Cordova, Cross-Platform Tools (CPTs), IBM Worklight, Mobile Applications, PhoneGap, Quasar, Sencha Touch, Software-Architecture, Titanium

Abstract
Smartphones define a new era of mobile phone generation and their market shares are continuously on the rise. Some mobile operating systems, including Android and iOS, have become well established on the market. The market is highly competitive for mobile operating systems, so that many competitors try to enter the market as well.

Due to the variety of mobile operating systems, the provision of mobile applications for multiple operating systems is associated with additional costs in terms of time and money. The reason for this is that each mobile operating system has its own language, API and development environment, so that source code cannot be reused across platforms. Consequently, a separate development for each mobile operating system is necessary.

Cross-platform tools enable cross-operating system developments. Thus, development expenses as well as incremental costs per platform will be reduced to a minimum. With this in mind, various technology approaches are used to achieve this.

Software architecture is essential in order to achieve a certain level of software quality. This thesis examines the integration of standard architectures for selected cross-platform tools. “Quasar” is a software architecture, which was developed by Capgemini (formerly sd&m). It was chosen as an example for standard architectures.

This thesis is primarily engaged on the analysis and subsequent evaluation of integrating the Quasar-Client dialog component architecture for selected cross-platform tools in mobile apps. Cross-platform tools were selected based on PhoneGap combined with Sencha Touch and Titanium with the Alloy framework.

According to the analysis and evaluation, it was found that the architectural pattern of Model-View-Controller provides a good foundation for integrating the Quasar-Client dialog component architecture. Using this pattern with the selected cross-platform tools will not allow a complete but only a partial integration. For full integration a complete concept would be necessary which may be developed in future works.

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel beschreibt im Abschnitt 1.1 die Problemstellung und Motivation und definiert in Abschnitt 1.2 die Ziele dieser Arbeit. Der Abschnitt 1.3 grenzt das Thema ein und der abschließende Abschnitt 1.4 gibt einen Überblick über den Aufbau der Arbeit.

1.1 Problemstellung und Motivation

Smartphones haben ein neues Zeitalter der Handygenerationen eingeläutet. Ihr Marktanteil nahm in den letzten 3 Jahren kontinuierlich zu [Joussen 2012]. Der steigende Anteil ist sowohl lokal in Deutschland als auch global zu beobachten [Ho u.a. 2010]. Mit der steigenden Anzahl an Smartphones, steigt auch das mobile Datenaufkommen stetig. Cisco geht davon aus, , dass sich das Datenaufkommen von 2011 auf 2012 auf 1,3 Exabyte verdoppelt hat [Cisco 2012].

Für dieses Datenaufkommen sind nicht nur Smartphones, sondern auch andere mobile Endgeräte wie Tablets verantwortlich [Cisco 2012]. Die BITKOM berichtet, dass im Jahr 2011 allein in Deutschland knapp eine Milliarde App-Downloads verzeichnet wurden und damit einen Umsatz von 210 Millionen Euro generiert wurde [BITKOM 2012]. „App“ ist die Kurzform des englischen Begriffs „Application“ und bedeutet zu Deutsch Anwendung. Dabei handelt es sich genauer gesagt um eine kleine Anwendung mit beschränktem Funktionsumfang [Franke u.a. 2012] [Frei 2012].

Die genannten Zahlen und Fakten machen die hohe Relevanz mobiler Anwendungen für Unternehmen deutlich. Mobile Anwendungen spielen in den Bereichen B2C (Business-to-Consumer / Endkunden), B2E (Business-to-Employee / Mitarbeiter) und B2B (Business-to-Business / Geschäftspartner) eine wichtige Rolle. Jeder dieser Bereiche hat unterschiedliche Ziele. Der B2C Bereich hat das Ziel neue Kundengruppen zu erschließen und neue Geschäftsmodelle mit Mobilitätsaspekten umzusetzen. Das primäre Ziel des B2E Bereichs dagegen, liegt darin Prozesse zu optimieren und somit die Produktivität der Mitarbeiter zu steigern. Dies kann durch den Einsatz von mobilen Geräten erreicht werden, die eine beschleunigte Informations-Bereitstellung und einen beschleunigten Informations-

Austausch ermöglichen. Auch im B2B Bereich ist eine bessere Informationsbereitstellung von Interesse, um Prozesse mit den Geschäftspartnern zu flexibilisieren [Geisler u.a. 2012].
Die Hauptanteile des mobilen Betriebssystemmarktes weltweit nehmen Android am 22. Januar 2013 mit 40,41% und iOS mit 29,45% ein [StatCounter 2012]. Sie zählen unter den Smartphone Plattformen neben Bada (Samsung), Blackberry, Symbian (Nokia) und Windows Phone zu den wichtigsten Betriebssystemen [VisionMobile 2012].

Einzelne native App-Entwicklungen für verschiedene Betriebssysteme benötigen eine hohe Investition an Ressourcen in Bezug auf Zeit und Geld [Jasar 2011]. Dies hat den Grund, dass der Quellcode einer Plattform für andere meist nicht wiederverwendbar ist und jede dieser Codebasen separat gepflegt werden muss. Sogenannte Cross-Plattform-Tools (CPTs) bieten die Möglichkeit Applikationen auf mehreren Plattformen zu distribuieren und gleichzeitig den Codier-Aufwand auf ein Minimum zu reduzieren und somit auch die inkrementellen Kosten pro Plattform zu reduzieren [VisionMobile 2012] Demnach kann mit einer einzelnen Codebasis verschiedene Plattformen gleichzeitig bedient werden.

Um eine Software zu erstellen und vor allem eine gewisse Qualität zu erreichen, ist eine Software-Architektur essentiell. Dies gilt ebenso für mobile Anwendungen, mit denen sich diese Arbeit auseinandersetzt. Die Firma Capgemini (ehemals sd&m) entwickelte die Quasar-Client-Architektur, um Client-Anwendungen selbst bei hoher Komplexität beherrschbar zu halten. Im Rahmen dieser Arbeit wird untersucht in wie fern ausgewählte Prinzipien dieser Standard-Client-Architektur mit dem Cross-Plattform-Tools umgesetzt werden können.

1.2 Zielsetzung

Ziel der Arbeit ist es die Integration ausgewählter Quasar-Client-Architektur Prinzipien für Cross-Plattform-Entwicklungsansätze in mobile Anwendungen zu analysieren und zu bewerten. Dabei wird am Beispiel von Android und ausgewählten Cross-Plattform-Tools eine exemplarische App entworfen, entwickelt und daraufhin rückblickend Erfahrungen dokumentiert und Untersuchungen zum Vergleich der entstandenen Apps durchgeführt.
Vor der Entwicklung wird eine Liste mit funktionalen Anforderungen aufgestellt, die für mobile Applikationen besonders häufig benötigt werden. Die Liste setzt sich dabei zum einen aus den Ergebnissen vorhandener Berichte und zum anderen aus den Resultaten eigener Analysen zusammen. Die ermittelten Anforderungen können für Implementierungsschritte priorisiert werden.

Aufbauend auf der Schulungs-App von Capgemini mit dem Namen „WhatsCapp“ werden zunächst weitere Analysen durchgeführt und Anpassungen vorgenommen, so dass die App sich den noch aufzustellenden Anforderungen annähert. Von der modifizierten App ausgehend werden daraufhin die anderen Apps mit ausgewählten Cross-Plattform-Tools

analog implementiert. Nach Abschluss der Implementierungen werden einige Vergleichsanalysen erläutert, durchgeführt und die Resultate zusammengefasst.

1.3 Abgrenzung des Themas

Der Begriff Cross-Plattform-Tools (CPTs) schließt verschiedene Tools bzgl. der Zielplattform ein. CPTs können als Zielplattformen sowohl mobile Plattformen wie Android oder iOS aber auch Desktop- oder Konsolen-Plattformen wie Microsoft Windows oder Playstation haben. Diese Arbeit befasst sich ausschließlich mit Cross-Plattform-Tools die primär auf mobile Plattformen ausgerichtet sind.

Firmen haben begonnen CPTs in drei unterschiedliche Segmente zu unterteilen. Dazu unterscheiden sie Games-, Enterprise- und Media-Apps voneinander. Entwickler müssen sich in diesen drei Segmenten unterschiedlichen Herausforderungen stellen. Sie arbeiten in gänzlich verschiedenen Umgebungen und benötigen dafür sehr unterschiedliche CPT Lösungen [VisionMobile 2012]. Im Rahmen dieser Arbeit wurde auf die Untersuchung von CPTs des Games-Bereichs verzichtet.

In den folgenden Kapiteln wird klar, dass Quasar weitaus mehr umfasst als man in einer Abschlussarbeit behandeln kann. In dieser Arbeit werden Prinzipien der Quasar-Client-Architektur erläutert und angewandt. Eine vollständige Umsetzung der Quasar-Client-Architektur mit Cross-Plattform-Tools würde den Rahmen dieser Arbeit sprengen.

1.4 Inhaltlicher Aufbau der Arbeit

Mit dem 0. Kapitel werden die Grundlagen für Cross-Plattform-Tools und die Quasar-Client Dialogkomponenten-Architektur (Dialog-Architektur) gelegt. Das darauffolgende Kapitel 3 gibt einen kurzen Überblick zu vergleichbaren Arbeiten und beschreibt welchen Nutzen sie für diese Arbeit haben und worin sie sich unterscheiden.

Kapitel 4 beschreibt das Analysevorgehen und die Ergebnisse in Bezug auf typische Anforderungen einer mobilen App. Die Anforderungen dienen dabei als Orientierung für die zu entwickelnden Apps, sowie als Untersuchungsgrundlage für die Umsetzbarkeit in den ausgewählten Cross-Plattform-Tools des 0. Kapitels.

Nach der Beschreibung der Cross-Plattform-Tool-Auswahl greift das 6. Kapitel diese Tools auf und beschreibt wie die Quasar-Client Dialogarchitektur für diese umgesetzt werden kann. Dabei beschreibt das 6. Kapitel die Integration theoretischer und das 7. Kapitel praxisnaher anhand eines Fallbeispiels.

Das 8. Kapitel beschäftigt sich mit Vergleichsanalysen der gewählten Tools am Beispiel der entstandenen Apps. Es beschreibt aber auch gesammelte Erfahrungen bezüglich verschiedener Kriterien.

Kapitel 0 fasst abschließend alle Ergebnisse noch einmal prägnant zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für die folgenden Kapitel gelegt. Das Unterkapitel 2.1 beschriebt zunächst welche App-Typen existieren und daraufhin wird im Unterkapitel 0 die Marktsituation und Unterschiede mobiler Betriebssysteme erläutert auf denen die Apps unterschiedlicher Typen laufen können. In Unterkapitel 0 wird auch vermittelt aus welchem Grund die native App Entwicklung mit hohen Kosten verbunden sein kann. Eine mögliche Lösung der Problematik wird mit dem Unterkapitel 2.3 beschrieben. Cross-Plattform-Tools werden hinsichtlich mehrerer Aspekte beschrieben und einzelne Tools auch exemplarisch näher erläutert. Das anschließende Unterkapitel 2.4 legt die Grundlagen für die Quasar-Softwarearchitektur und beschreibt insbesondere die komponentenbasierte Client-Architektur. Der abschließende Unterkapitel 2.5 beschreibt das Architekturmuster „Model-View-Controller“, welches in den folgenden Kapiteln 6 und 7 wieder aufgegriffen wird.

2.1 App-Typen

Eine App ist, wie im Einleitungskapitel beschrieben, die Kurzform für "Application" und bedeutet zu Deutsch Anwendung. Diese bezieht sich insbesondere auf kleinere Anwendungen mit einem begrenzten Funktionsumfang [Franke u.a. 2012] [Frei 2012]. Dabei existieren unterschiedliche Typen von Apps.

In diesem Unterkapitel werden die Unterschiede zwischen Web Apps, hybriden und nativen Apps beschrieben. Dabei veranschaulicht die folgende Grafik den Aufbau der App-Typen. Hybride und native Apps sind downloadbar, wohingegen Web Apps nur über den Browser zugänglich sind.

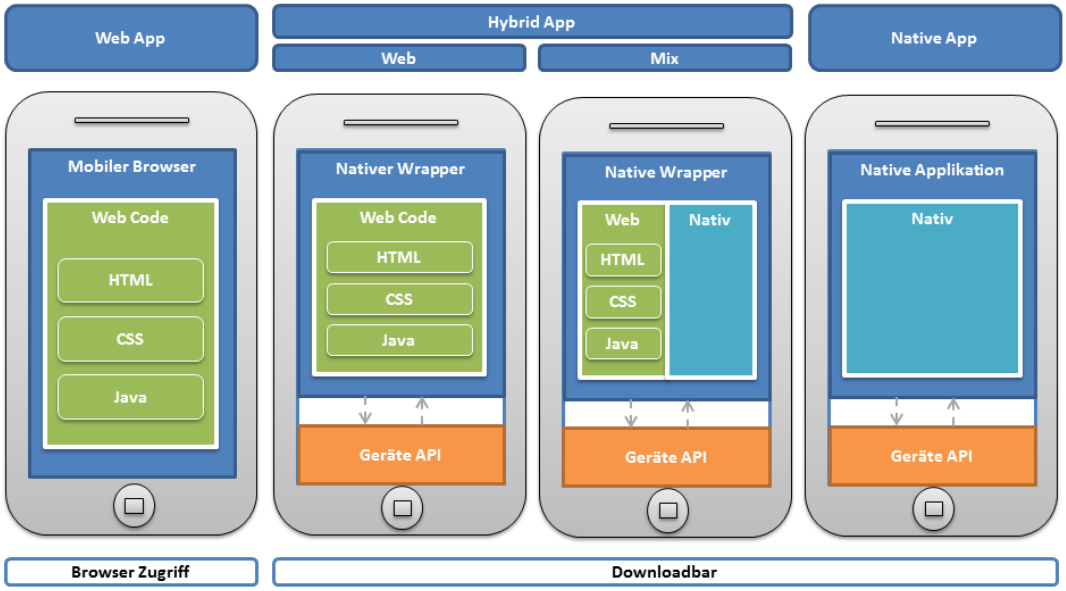

Abbildung 1 App-Typen im Überblick [VisionMobile 2012][Jasar 2011][Worklight 2012].

2.1.1 Native App

Jasar beschreibt in seiner Arbeit native Apps als Applikationen, die auf eine spezielle Plattform angepasst sind und über die spezifischen APIs und Bibliotheken auf Hardwareelemente zugreifen. Sie sind direkt installierbar und werden meist über eine zentrale Stelle, wie beispielsweise dem Apple App Store oder Google Play distribuiert. Jede Plattform hat eine spezifische Sprache mit der die nativen Apps entwickelt werden (siehe Abbildung 1), dadurch haben native Apps Vorteile im Bereich Performance und beim Zugriff auf Hardware- und Software-APIs, wie zum Beispiel dem Zugriff auf die Kamera oder dem Adressbuch. Nachteilig an nativen Apps sind dagegen die verhältnismäßig hohen Entwicklungskosten und die Plattformbindung [Jasar 2011].

VisionMobile erklärt jedoch, dass der Begriff einer nativen App durch das Aufkommen von CPTs etwas unklarer geworden ist. Einige Tools, unter anderem auch Titanium von Appcelerator, erlauben es auch Entwicklern mit Web Technologien „native Applikationen“ zu schreiben [VisionMobile 2012]. Daraus resultiert die Tatsache, dass der Begriff einer nativen App nicht mehr auf die Entwicklung mit der spezifischen nativen Entwicklungssprache begrenzt ist.

2.1.2 Web App

„Eine WebApp ist eine Website, die sich wie ein Programm anfühlt und bedienen lässt“ [Franke u.a. 2012].

Die Navigation durch Web-Apps ähnelt der Navigation wie bei einer nativen App. Sie folgen dem typischen Softwareschema von der Informationseingabe über die Verarbeitung bis hin zur Ausgabe. Das Layout orientiert sich dabei an typischen nativen Apps. Somit unterscheidet sich eine solche mobile App von einer „normalen“ bzw. nicht mobil optimierten Website [Franke u.a. 2012] [Jasar 2011]. Der Begriff ist jedoch nicht auf mobile Webseiten beschränkt, sondern umfasst prinzipiell alle Webseiten. Er wird meist im Zusammenhang mit HTML5 angeführt, womit offlinefähige Webseiten gemeint sind [Jasar 2011]. Technisch gesehen sind die Grundlagen wie bei Webseiten, die kein HTML5 nutzen. Web Apps werden mit Web-Technologien wie HTML, CSS und JavaScript entwickelt. Doch durch die Verwendung von HTML5 ist eine bessere Plattformintegration und eine Erweiterung der Funktionalitäten von WebApps möglich. Neben der Offlinefähigkeit zählt zu den erweiterten Funktionalitäten aber auch die Ermittlung des GPS Standortes. [Jasar 2011].

Zusätzliche Frameworks wie Sencha Touch oder JQuery mobile vereinfachen es Web Apps an das Look and Feel einer nativen App anzupassen. Dabei geht es so weit, dass sich eine Web App durch die UI-Komponenten, Animationen und Hilfsklassen kaum noch von einer nativen unterscheiden lässt [Jasar 2011].Web Apps werden meist von dem Benutzer nicht als App wahrgenommen, da sie in Form einer Webseite angeboten werden und nicht installiert werden müssen.

Der größte Nachteil ist nach Jasar zurzeit der eingeschränkte Zugriff auf Hard- und Software-Funktionen der Plattformen. Ein weiterer Nachteil ist der fehlende Zugriff auf die nativen App Stores und die teilweise schlechtere Performance. Besonders Vorteilhaft dagegen sind die geringen Entwicklungskosten und die von Haus aus plattformübergreifende Eigenschaft, da zurzeit die meisten Geräte auf die Browser-Engine „WebKit“ setzen [Jasar 2011] [Charland u.a. 2011]. Jedes Gerät und Betriebssystem hat in Bezug auf Webkit spezielle Unterschiede. Dabei handelt es sich jedoch nur um geringe Unterschiede [Charland u.a. 2011]. Auch die einfache Distribution ist ein Vorteil [Jasar 2011].

Weitere Vorteile sind, dass keine Installation der App nötig ist und auch keine Updates im eigentlichen Sinne durchgeführt werden müssen [Franke u.a. 2012]. Es findet zwar ein Update durch das erneute Laden der Seite statt, doch da dieses nicht vom App-Store bezogen wird unterscheidet es sich von dem Updateprozess nativer Apps.

2.1.3 Hybrid App

Es stellt sich oft die Frage ob man web oder nativ entwickeln soll. Hybride Apps versuchen das Beste aus beiden Welten zu kombinieren und die Nachteile beider zu kompensieren.
Eine hybride App besteht aus einer nativen Shell bzw. einem Wrapper, der HTML, CSS und JavaScript enthält. Wenn eine hybride App läuft, startet der Wrapper eine Instanz des Webbrowsers, welche man als WebView bezeichnet. Diese WebView lädt daraufhin HTML, CSS und JavaScript. Die Instanz ist für gewöhnlich dabei "chromeless". Dies bedeutet beispielsweise, dass sie keine Browsersteuerelemente besitzt und somit wie eine native Applikation aussehen kann [VisionMobile 2012].

Aus Benutzersicht sind hybride Apps wie native Apps. Sie werden wie native Apps über den nativen App-Stores der jeweiligen Plattform entdeckt und heruntergeladen. Die Installation und das Starten der App funktioniert ebenfalls gleich [VisionMobile 2012].

Auch aus Entwicklersicht existieren Ähnlichkeiten. Der Entwicklungsprozess einer hybriden App ähnelt dem einer nativen Apps mit einer Ausnahme. Entwickler können hybride Apps mit HTML, CSS und JavaScript entwickeln. Da alle mobilen Hauptplattformen eine hybride Webentwicklung unterstützen, bieten hybride Apps den Vorteil, dass sie den Code der genannten Web-Technologien wiederverwenden können [VisionMobile 2012].

IBM beschreibt in seinen Erläuterungen von IBM Worklight den Begriff einer hybriden App noch etwas genauer und unterscheidet dabei zwei Unterarten hybrider Apps. Sie unterscheiden hybride Web Apps von hybriden Mix Apps [Worklight 2012].Hybride Web Apps nutzen ausschließlich Webtechnologien und entsprechen im Prinzip dem, was bisher auch beschrieben wurde.

Hybride Mix Apps dagegen enthalten neben dem Web-Source-Code auch noch plattformspezifischen nativen Code. Der native Code wird dabei von dem Web-Code aufgerufen. Nativen Code zu schreiben macht dann Sinn, wenn man mit Webtechnologien an seine Grenzen stößt. Zurzeit ist es beispielweise noch nicht möglich mit Web-Technologien Funktionalitäten wie Near Field Communication (NFC) oder Argumented Reality (AR) zu nutzen [Worklight 2012]. Durch den Einsatz von nativem Quellcode wird die Nutzung solcher Funktionalitäten dennoch möglich. Mehrere CPTs ermöglichen bereits die Erstellung von hybriden Apps. PhoneGap, Sencha v2, IBM Worklight oder Appcelerator sind nur eine kleine Auswahl der CTPs, die dies ermöglichen [VisionMobile 2012].

2.2 Mobile Betriebssysteme

In diesem Unterkapitel soll deutlich werden wie stark die einzelnen Betriebssysteme global vertreten sind und auf die Problematik eingegangen werden, die bei der Entwicklung von Apps für verschiedene Plattformen entsteht.

2.2.1 Marktsituation

Es existieren verschiedene Betriebssysteme für mobile Endgeräte. In den vergangen 3 Jahren haben sich die Marktanteile verändert. Während SymbianOS im Januar 2010 noch einen Marktanteil von 34,16% hatte, wurde dieser Anteil auf ca. ein Fünftel des damaligen Anteils reduziert. Android hat dagegen mit einem damaligen Marktanteil von 4,54% seinen Anteil auf 40,41% knapp verneunfacht und verzeichnet damit auch den größten Zuwachs in den letzen 3 Jahren [StatCounter].

Bei dieser Gesamtstatistik ist zu beachten, dass regional gesehen die Unterschiede im Vergleich zur hier dargestellten weltweiten Statistik sehr stark ausfallen können. In Afrika nehmen beispielsweise Nokias Series 40 und Symbian den größten Anteil mit knapp 34% und 23% ein und Android belegt nur Platz 3 mit etwa 18%. In Asien liegt der Android Marktanteil ca. 35% über dem iOS Anteil und ist damit vor SymbianOS und Series 40 das verbreiteteste mobile Betriebssystem in Asien. Auf dem europäischen Markt hat Android einen knappen Vorsprung gegenüber iOS mit knapp 5% während iOS in Nordamerika mit 16% noch vor Android liegt und mehr als die Hälfte des Marktes abdeckt [StatCounter].

Wenn man die globale Grafik betrachtet sollte man diese regionalen Unterschiede im Hinterkopf behalten. Die Statistikwerte wurden von StatsCounter mithilfe von Trackingcode erhoben, die auf mehr als 3 Millionen globalen Seiten verteilt sind [StatCounter].

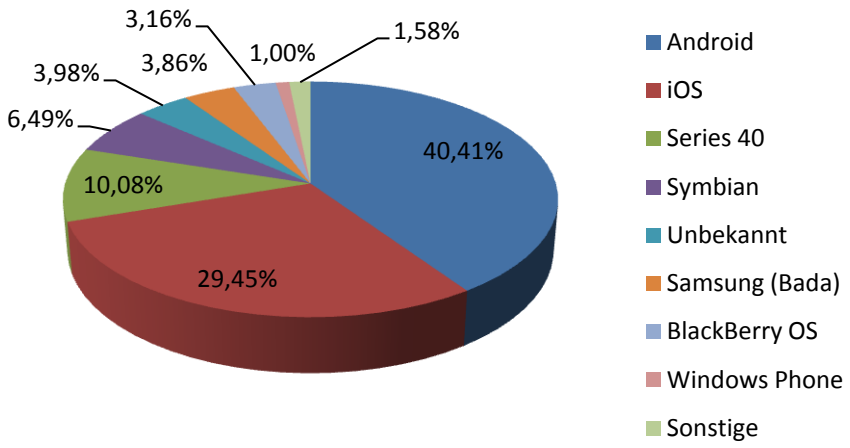

Abbildung 2 Marktanteile mobiler Betriebssysteme weltweit nach Nutzung. Stand vom 22.01.2013 [StatCounter]

Eine weitere Alternative den „Marktanteil“ zu ermitteln ist es Verkaufszahlen heranzuziehen, wie es auch Maximilian Frei in seiner Abschlussarbeit getan hat [Frei 2012]. Der Pressebericht von Gartner beschränkt sich auf die Haupt-Smartphone Betriebssysteme Android, Apples iOS, Blackberry OS von Research in Motion (RIM), Samsungs Bada, Nokias Symbian und Windows Phone [VisionMobile 2012] [Gartner 2012]. Auch hier nimmt Android mit 72,40% die Marktführerposition ein. In dieser Statistik sind wieder regionale Unterschiede von Bedeutung aber auch saisonale. Gartner beschreibt beispielweise, dass das 4. Quartal meist das stärkste für iOS ist [Gartner 2012].

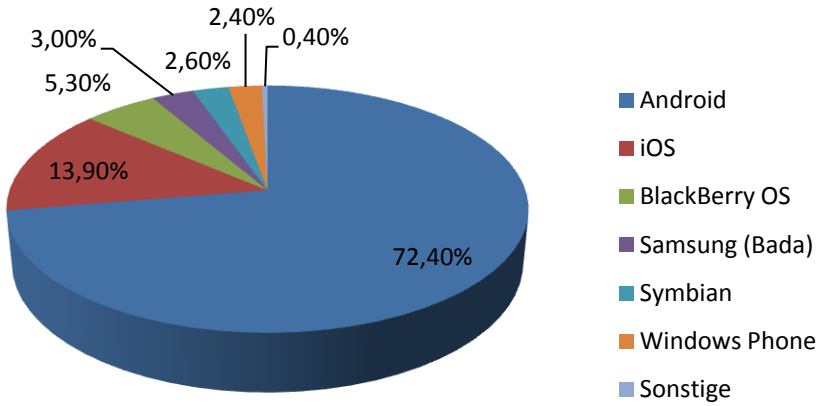

Abbildung 3 Marktanteile mobiler Betriebssysteme weltweit nach Verkaufszahlen des 3. Quartals 2012 [Gartner 2012]

2.2.2 Unterschiede

Die erwähnten Haupt-Smartphone Plattformen bzw. Betriebssysteme unterscheiden sich in mehreren Gesichtspunkten. Jedes Betriebssystem hat ihre eigene Sprache, API, Entwicklungsumgebung und ihren eigenen App Store von dem Apps bezogen werden können. Nach den Verkaufszahlen des 3. Quartals des letzten Jahres sind Android, iOS und Blackberry OS am stärksten abgesetzt worden. Möchte man beispielsweise eine App für diese drei Plattformen entwickeln sind drei unterschiedliche Teams mit unterschiedlichen Fertigkeiten nötig. Diese drei Teams erzeugen drei unterschiedliche Codebasen, in denen Features und Fehlerbehebungen gleichermaßen gepflegt werden müssen. Dies stellt eine Herausforderung dar, die auch ihre Folgen hat. Es kann passieren, dass Apps in den App Stores mit den gleichen Features erst nach mehreren Monaten Verspätungen verfügbar sind. Zudem kann die Qualität und Konsistenz der Apps variieren wenn mehrere Teams involviert sind. Dies ist besonders zu erwarten, wenn der Entwicklungsauftrag für eine der Plattformen an eine außenstehende dritte Partei gegeben wird. Zusätzlich zu diesen möglichen Komplikationen muss auch der Dokumentationsaufwand, die Supportkosten und allgemein die Teamkosten für die drei Plattformen beachtet werden [VisionMobile 2012].

Plattform	Sprache	Entwicklungsumgebung	App Store
Android	Dalvik C und C++ (NDK) Java WebViews	Android Development Tool (ADT) Plugin für Eclipse Andere IDEs wie z.B.: IntelliJ, Netbeans, Visual Studio	Google Play Store
Bada	C++ (proprietäre Erweiterung) HTML, CSS, JS	Bada IDE (Eclipse basierend) CDT und JSDT (JavaScript Development Tools)	Samsung Apps
Blackberry OS	J2ME MIDP 2.0 HTML WebWorks SDK	Eclipse Plugin	Blackberry App World
iOS	Objective C WebViews	Mac mit Xcode	App Store
Symbian	C++ Qt	Carbide C++ IDE	Ovi Store
Windows Phone	C#, VB.NET Silverlight Framework XNA Framework	Visual Studio und Microsoft Tools wie Expression Blend	Windows Phone Marketplace

Tabelle 1 Unterscheidungstabelle der Haupt-Plattformen nach Sprache, Entwicklungsumgebung und App Store [VisionMobile 2012]

Bei der Entwicklung einer App für mehrere Zielplattformen können Cross-Plattform-Tools dabei helfen Kosten zu sparen. Aus diesem Grund ist es sinnvoll sich mit ihnen zu befassen. Der CEO der Firma InRuntime berichtete beispielsweise, dass sie durch den Einsatz von Cross-Plattform-Tools die Zeit bis zum Marktstart einer App durchschnittlich um 70% reduzieren konnten gegenüber einer rein nativen Entwicklung. Es werden nach Ihm auch dann Cross-Plattform-Tools verwendet, wenn eine einzelne App für eine einzelne Plattform entwickelt werden soll [VisionMobile 2012].

2.3 Cross-Plattform-Tools

VisionMobile definiert Cross-Plattform-Tools (CPTs) als Werkzeuge, die es möglich machen Applikationen auf mehreren Plattformen zu distribuieren, gleichzeitig die inkrementellen Kosten pro Plattform zu reduzieren und die Wiederverwendung von Quellcode zu maximieren [VisionMobile 2012]. CPTs können für mobile Plattformen wie iOS oder Android ausgelegt sein, aber auch für Tablets, Desktop-PCs oder Spielekonsolen. CPTs werden häufig auch als Cross-Plattform Frameworks bezeichnet [Frei 2012].

Es existieren über 100 Cross-Plattform-Tools. Jeder dieser Tools verwendet dabei einen Technologieansatz oder auch eine Kombination mehrerer. Sie unterstützen unterschiedlich viele Plattformen und sind dabei auf einen Anforderungsbereich optimiert, die je nach genutzten Ansätzen auf bestimmte Zielgruppen ausgerichtet sind [VisionMobile 2012].

Im Folgenden wird die historische Entwicklung von Cross Plattform Tools betrachtet und daraufhin der App-Lebenszyklus beschrieben, der verschiedene Stufen beinhaltet. Diese einzelnen Stufen werden von Cross-Plattform Tools unterschiedlich stark unterstützt. Auf die Beschreibung der Phasen folgt eine nähere Erläuterung der erwähnten Technologieansätze. Abschließend werden CPTs noch nach Anforderungen unterschieden und einige dieser Tools in Kürze vorgestellt.

2.3.1 Historie

Cross-Plattform-Tools bzw. -Frameworks für mobile Geräte durchliefen eine evolutionäre Entwicklung. Die einheitliche WAP Standard-Technologie machte dabei den Anfang. Erstmals versuchte man Internetinhalte auf mobile Geräte zu übertragen. Zu dieser Zeit nutzte man das langsame Mobilfunknetz zur Übertragung der Inhalte auf mobile Geräte [Frei 2012].

Die Möglichkeit mobile Inhalte übertragen zu können wurde erst mit dem Beginn der Feature Phone Ära (1998-2008) möglich. Bis zu diesem Zeitraum war es nur möglich zu telefonieren, Kurznachrichten zu versenden und kleine Spiele, wie das bekannte Spiel „Snake“, zu spielen. Die vorangehenden Ären wurden von Fling als Brick Ära (1973-1988) und Candy Bar Ära (1988-1998) bezeichnet. Die Feature Phone Ära erweiterte das Telefon um eine Reihe an neuen Anwendungen und Services wie die Wiedergabe von Musik, das Aufnehmen von Bildern oder aber auch den Internetzugang.

Die vierte Ära ist die Smartphone Ära, welche sich im Laufe der Feature Phone Ära seit 2002 bis heute parallel entwickelte. Smartphones können alles, was Feature Phones bisher konnten, doch sie unterscheiden sich dadurch, dass sie ein gemeinsames Betriebssystem verwenden, ein größeres Display besitzen, eine QWERTZ Tastatur oder Eingabestift zur

Eingabe besitzen und Wi-Fi oder über ein ähnlich schnelle Drahtlose Verbindung verfügen. Die fünfte und finale Ära wurde mit der Einführung des iPhones im Januar 2007 eingeläutet [Fling 2009].

Zunehmend schnellere Übertragungsgeschwindigkeiten und leistungsfähigere Hardware brachten die Hersteller dazu normale Webseiten direkt über einen Browser der mobilen Geräte darzustellen. Da die regulären Webseiten nicht für mobile Endgeräte optimiert waren, wurden und werden auch heute noch diese Webseiten durch mobile Internetseiten abgelöst, die speziell für die geringeren Bildschirmgrößen optimiert sind. Aus dem Gedanken der mobil optimierten Internetseiten, resultierte die Idee kleine Anwendungen über Internetseiten bereitzustellen. [Frei 2012].

Durch die Leistungszunahme der mobilen Endgeräte wurden diese dazu fähig aufwändigere Berechnungen direkt auf dem Gerät durchzuführen. Aus der kontinuierlichen Weiterentwicklung der mobilen Webanwendungen resultierten dann kleine Anwendungen, die auf dem Gerät installiert werden konnten [Frei 2012]. Für diese kleinen Anwendungen wurde die Kurzform „App“ verwendet. Im Gegensatz zu Desktop-Anwendungen haben Apps einen stark begrenzten Funktionsumfang und erfüllen meist nur eine bestimmte Aufgabe, wie beispielsweise das Versenden und Empfangen von E-Mails. Statt einem großen Allzweckprogramm werden demnach viele kleine Apps zur Zweckerfüllung installiert. Apple leistete in dieser Hinsicht durch die Einführung des Apple App Stores ein Stück Pionierarbeit. Der App Store stellt eine zentrale Plattform dar über die die Benutzer Apps für ihre iPhones erwerben können. Das neuartige Konzept von Apple verbreitete sich schnell, bald wurde es auf weitere Apple-Betriebssysteme und -Produkte ausgeweitet und von anderen Firmen wie beispielweise Google für Google Play (ehemals Android Market) übernommen [Frei 2012].

Eine native App für jedes Betriebssystem einzeln zu schreiben ist machbar, sofern man über ein Team mit ausreichend vielfältigen Fertigkeiten verfügt. Es ist aber nicht leicht dies zu tun, da die gegenwärtigen Plattform SDKs sich in vielerlei Hinsicht unterscheiden. Für jede Plattform existieren unterschiedliche Werkzeuge, Build Systeme, APIs und Geräte mit unterschiedlichen Möglichkeiten. Die genauen Unterschiede wurden bereits in Kapitel 2.2.2 beschrieben. Das Einzige was alle gemeinsam haben ist ein mobiler Browser [Charland u.a. 2011].

Mobile Webseiten haben daher den Vorteil, dass Sie einmal geschrieben auf allen Geräten angezeigt und genutzt werden können. Die nativen Apps bieten jedoch diesen Vorteil nicht mehr [Frei 2012].

Eine mögliche Lösung, um diesen Nachteil auszugleichen sind die erwähnten Cross-Plattform Tools. Sie bestehen für gewöhnlich aus zwei Komponenten. Die eine Komponente bildet die Entwicklungsumgebung, zu der neben einer Programmierbibliothek oder API auch

die verwendete Programmiersprache und ein Übersetzungstool oder Compiler gehört. Den zweiten Teil bildet die Ausführungsumgebung, welche den abstrakten Code interpretiert und somit als Interpreter („Übersetzungsschicht“) zwischen dem nativen Gerätecode und dem Cross-Plattform Code dient. Sogenannte Cross Compiler stellen einen Sonderfall dar, welche den Code einer abstrakten Sprache direkt in nativen Gerätecode übersetzt und somit keine Ausführungsumgebung nötig ist [Frei 2012]. Abbildung 4 veranschaulicht kurz und prägnant die Entwicklung der CPTs.

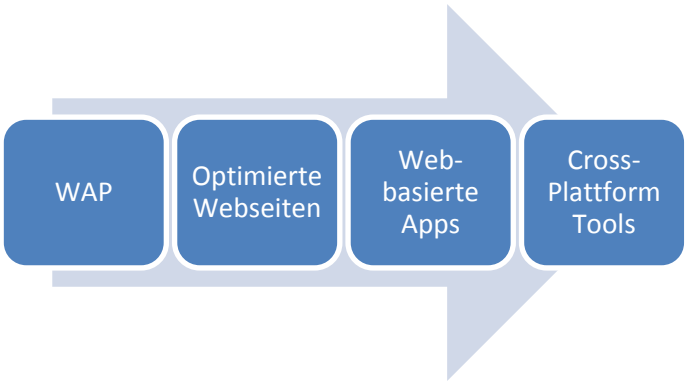

Abbildung 4 Entwicklungsprozessstrahl von Cross-Plattform Anwendungen [Frei 2012]

2.3.2 Technologieansätze

In den Analysen von VisionMobile wurden fünf verschiedene Technologieansätze herausgearbeitet, welche zunächst in der Tabelle 2 dargestellt und daraufhin etwas näher beschrieben werden. Jeder dieser Ansätze ist auf eine spezifische Zielgruppe abgestimmt. Die Zielgruppen reichen von Nicht-Entwicklern bis hin zu erfahrenen Entwicklern. So ist es beispielsweise auch Nicht-Entwicklern möglich mithilfe einiger Tools eine Applikation zu erstellen [VisionMobile 2012].

Typ	Beschreibung	Beispiele	Zielpublikum	Sprache	Anwendungsfälle
JavaScript Framework	JavaScript Code Module bieten zeitsparende User Interfaces oder andere Komponenten	jQuery Mobile, Sencha Touch, Cocos2D, DHTMLX Touch, Zepto JS, ImpactJS, LimeJS, jQuery Mobile, Sencha Touch, Cocos2D, DHTMLX Touch, Zepto JS, ImpactJS, LimeJS, iUI, Wink	Web-Entwickler	JavaScript	Touch UIs erstellen, Cross-Plattform Browser Komptabilität erreichen, Nativen Look and Feel liefern, Komplexe Spielefunktionalitäten liefern

App Factory	Code-freie, visuelle Design Tools zum einfachen Erstellen mobiler Apps	AppMkr, Wix Mobile, Tiggzi, Mobile Nation HQ, Mobjectify, Spot Specific, Red Foundry, Games Salad	Nicht-Entwickler	Visuelle, codefreie Werkzeuge	Eigene App erstellen
Web-To-Nativ Wrapper	Lösung, die Apps im nativen Ausgabeformat liefert bei Verwendung von HTML und JavaScript. Der Webcode wird durch eine native shell gekapselt und Bibliotheken ermöglichen den Zugriff auf native Funktionalitäten.	Adobe (PhoneGap Build),Uxebu (Aparrat.io) Sencha(Touch v2), MoSync (Wormhole)	Web-Entwickler	HTML, JavaScript, CSS	WebApps in Apps mit nativen Ausgabeformat umwandeln, Zugriff auf native Funktionalitäten und Optimierungen
Runtime	Ausführungsumgebung, welche die App von der darunterliegenden Plattform abstrahiert	J2ME, Adobe Air, Ansca Mobile (Corona), AppMobi, Antix, Unity, Appcelerator, Xamarin	Software-Entwickler	Jede unterstütze Sprache	Apps die eine breite Reichweite in Bezug auf Plattformen und Bildschirmen benötigen
Sourcecode Übersetzer	Konvertiert plattformunabhängigen Sourcecode in Code der nativen Plattform oder kompiliert den Sourcecode direkt in Binärcode.	MoSync, Eqela, Marmalade, Bedrock, XMLVM	Software-Entwickler	Jede unterstützte Sprache	Anwendungen mit komplexer Logik und hohen Performance- oder Optimierungs-Anforderungen

Tabelle 2: Übersichtstabelle der 5 Technologieansätze [VisionMobile 2012].

2.3.2.1. **JavaScript Framework**

JavaScript Frameworks sind Code Bibliotheken, die entwickelt wurden um Web-Entwickler bei ihren komplexen Aufgaben zu unterstützen und die Entwicklung zu beschleunigen. Sie vereinfachen beispielsweise die Erstellung von Touchscreen User Interface oder die Umsetzung der Cross-Plattform Browserkompatibilität. Zudem ist es leichter möglich für die jeweilige Plattform das entsprechende native Look and Feel zu erreichen. Einige der Bibliotheken können aber auch die Umsetzung komplexer Spielefunktionalitäten vereinfachen und dadurch Aufgaben dieses Bereichs ebenfalls beschleunigen [VisionMobile 2012].

2.3.2.2. **App Factory**

App Factories sind visuelle Designwerkzeuge, bei denen man ohne Code zu schreiben einfache mobile Applikationen erstellen kann. Sie werden entweder über eine Cloud-basierte Entwicklungsumgebung angeboten oder müssen auf dem Rechner installiert werden. Durch das Durchlaufen eines Wizards mit Hilfe von Vorlagen oder das Zusammenstellen der App per „Drag and Drop“ wird Code generiert. Einige Tools

ermöglichen es den Code zu betrachten und teilweise zu optimieren. Diese Art von Applikationen ist an Nicht-Entwickler gerichtet [VisionMobile 2012].

2.3.2.3. **Runtime**

Eine Runtime oder auch Runtime Environment ist eine Ausführungsumgebung und eine Cross-Plattform Komptabilitätsschicht, die auf dem nativen Betriebssystem aufsitzt. Sie schirmt in erster Linie eine App von den Unterschieden der darunterliegenden Plattform ab. Dabei variieren sie in ihren Möglichkeiten und ihrer Komplexität und verwenden zu diesem Zweck je nach Runtime unterschiedliche Methoden wie Virtualisierung, Interpretation, Just-In-Time Kompilation oder Ahead-Of-Time Kompilation [VisionMobile 2012] [Frei 2011]. Dabei unterscheidet man integrierte Runtimes von Standalone Runtimes. Integrierte Runtimes sind solche, die als ein Teil einer kompilierten App mit ausgeliefert werden. Titanium ist beispielweise einer der CPTs, welches diesen Technologieansatz verwendet. Standalone Runtimes sind im Gegensatz zu den integrierten kein Teil der App und müssen separat installiert werden. Ein bekanntes Beispiel dafür ist Adobe Flex / AIR [Frei 2011].

2.3.2.4. **Sourcecode Übersetzer (source code translator)**

Diese Lösung übersetzt (cross-kompiliert) den Source Code in eine andere Art von Code. Dieser Code kann ein Zwischenbytecode, ein nativer Sprach-Code oder Low-Level Maschinen-Code sein. Source Code Übersetzer werden meist in Kombination mit einem Runtime-Element verwendet. Metismo (nun Software AG) ist ein Beispiel, welches beispielsweise J2ME Applikationen in C++ Code, ActionScript und JavaScript umwandelt und für ARM, MIPS, Power PC und x86 Geräte den Code kompiliert. Andere Beispiele sind Mosync, Marmelade und Xamarin's Mono. Source Code Übersetzer zielen auf erfahrene Softwareentwickler ab, die Cross-Plattform Apps mit einer komplexen Logik und hohen Performanceanforderungen umsetzen wollen [VisionMobile 2012].

2.3.2.5. **Web-To-Native Wrapper**

„Web-To-Native Wrapper“-Lösungen ermöglichen es mit Web-Technologien wie HTML5, CSS und JavaScript Apps mit einem nativen Ausgabeformat zu erstellen [VisionMobile 2012]. Die Technologie, die hier verwendet wird ist die sogenannte „WebView“. Es handelt sich dabei um einen in die native App eingebetteten Browser, der die Darstellung der Web App übernimmt [Jasar 2011]. JavaScript API Erweiterungen ermöglichen dabei den Zugriff auf zusätzliche Funktionalitäten der Plattform. Zu den zusätzlichen Funktionalitäten zählen beispielsweise Benachrichtigungen, Beschleunigungssensoren, Kompass, Geolokalisation und der Zugriff auf das Dateisystem [VisionMobile 2012].

Die zuvor erwähnten JavaScript Frameworks können bei diesem Ansatz verwendet werden. Sie werden durch die zusätzliche Verwendung des Web-To-Native-Technologieansatzes zu einer App mit nativem Ausgabeformat, die dann installierbar ist. [Sencha 2011] Ohne den Web-To-Native Ansatz wird eine auf JavaScript basierende App nicht mit der plattformspezifischen WebView, sondern mit dem Browser des jeweiligen Gerätes angezeigt.

Das Paradebeispiel für ein Tool dieser Art ist PhoneGap, welches am Ende dieses Kapitels nochmal genauer beschrieben wird. Auch dieser Ansatz richtet sich an Webentwickler, vor allem an solche, die Web Apps zu nativen Apps umwandeln wollen oder an Entwickler, die für ihre Apps native Funktionalitäten benötigen [VisionMobile 2012].

2.3.3 Anforderungen

Wie in Kapitel 1.3 beschreiben haben Firmen begonnen CPTs in drei unterschiedliche Segmente zu unterteilen. Dazu unterscheiden sie Games-, Enterprise- und Media-Apps voneinander. Entwickler müssen sich in diesen drei Segmenten unterschiedlichen Herausforderungen stellen. Sie arbeiten in gänzlich verschiedenen Umgebungen und benötigen dafür sehr unterschiedliche CPT Lösungen [VisionMobile 2012].

Enterprise Application Plattformen sind beispielsweise CTPs, die den kompletten Applikationslebenszyklus von der Entwicklung über die Integration und Publizierung bis hin zum Managing unterstützen. Das bedeutet, dass beispielsweise Datenbankanbindungen, Middleware, Cloud Services, App Hosting, Policy Management und Push Messaging von diesen Tools unterstützt wird. Namenhafte CPTs dieser Art sind beispielweise Worklight, Kony, RhoMobile oder Verivo, aber es zählen auch noch einige andere dazu [VisionMobile 2012].

Games haben besondere Anforderungen. Sie werden meist mit Low-Level Sprachen wie C++ entwickelt und werden oft mit Scripting-Sprachen wie Lua für Spielelogiken kombiniert. Außerdem stellt die Game Engine eine besonders wichtige Komponente dar. Marktführer im Bereich für „Advanced 3D Games“ sind Unreal und Unity. An diesen beiden Beispielen sieht man, dass die Anforderungen in den Segmenten unterschiedlich sind und einige CPTs auch auf einzelne Segmente spezialisiert sind. [VisionMobile 2012].

2.3.4 App-Lebenszyklus

Eine App durchläuft in seinem Lebenszyklus die fünf Stufen: Entwicklung, Integration, Build, Veröffentlichung und Managing. Nach VisionMobile kann ein CPT fünf Komponenten enthalten, welche den genannten Stufen des App-Lebenszyklus entsprechen.

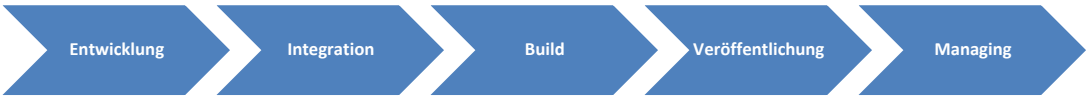

Abbildung 5 Fünf Stufen eines Cross-Plattform App-Lebenszyklus [VisionMobile 2012].

2.3.4.1. **Entwicklung**

In die Stufe bzw. Phase der Entwicklung gehört eine Vielzahl von Elementen. Die angebotenen Sprachen sind einer dieser Elemente. CPTs bieten eine große Auswahl an Sprachen zur Entwicklung an. Dabei können Einstiegssprachen wie LiveCode oder Lua verwendet werden aber auch Web Sprachen wie HTML, CSS und JavaScript bis hin zu höher-sprachlichen Sprachen wie Java oder C# / .NET.

Die Kernelemente dieser Phase sind neben der Entwicklungssprache die IDE (Integrated Development Environment), der Emulator und der Debugger. Es zählen aber auch Elemente wie Source Control, Kollaborations- und Workflowtools dazu. Ein weiterer Aspekt dieser Phase stellen Marktplätze dar, über die Designer und Entwickler Komponenten oder Assets anbieten können, um anderen die Möglichkeit zu bieten ihren Entwicklungsprozess zu beschleunigen [VisionMobile 2012].

2.3.4.2. **Integration**

Diese Phase beinhaltet sowohl die Integration nativer Funktionalitäten als auch die Integration von Cloud APIs und Datenbanken in die App. Um Funktionalitäten, wie die Kamera zu integrieren, wird oft JavaScript in Kombination mit PhoneGap und anderen Bibliotheken genutzt. Einige Plattformen bieten auch Services an, die das Management von Datenbankverbindungen unterstützen und die insbesondere bei Datensynchronisationen relevant sind [VisionMobile 2012].

2.3.4.3. **Build**

Die Phase beschäftigt sich mit dem „Build“ der App. Die verschiedenen Technologieansätze wurden bereits in den vorherigen Kapiteln etwas näher erläutert. Dabei kann der Code in

eine native Hülle eingebettet und von einer Ausführungsumgebung ausgeführt werden oder aber auch direkt in native Binärdaten kompiliert werden [VisionMobile 2012].

2.3.4.4. **Veröffentlichung**

Diese Phase beinhaltet alles in Bezug auf die Veröffentlichung. Sie beinhaltet sowohl das Einreichen der App in einen nativen App Store wie dem Google Play Store oder auch dem Apple App Store als auch das Hosting. Eine App kann dabei auch intern veröffentlicht werden. CTPs, wie RhoMobile oder Worklight bieten zum Beispiel die Möglichkeit Apps in einem privaten App Store zu hosten [VisionMobile 2012].

2.3.4.5. **Managing**

Das Managing beinhaltet viele Bereiche. Updateprozesse müssen verwaltet und Analysen durchgeführt werden. Analysetools helfen dabei Performanceaussagen zu treffen. Ebenfalls zu diesem Bereich zählen das Push Messaging, Datenflussmanagement, Remote Deinstallationen, das Managing kommerzieller Adds und Inventory Management und Policy Management [VisionMobile 2012].

2.3.5 Tools

Diese Arbeit widmet sich nicht dem „Games“ Segment, da sich die Anforderungen wie bereits erwähnt, je Segment stark unterscheiden. Dabei scheiden bei den folgenden Betrachtungen und Analysen bestimmte CPTs und Apps schon automatisch aus. Außerdem beschränkt sich die Arbeit auf mobile Applikationen, womit weitere CPTs ausscheiden, die keine mobilen Apps unterstützen.
Im Folgenden werden ausgewählte Tools bezüglicher ihrer Entwicklung, Technologie und ihrer Marktsituation und Positionierung kurz vorgestellt. Die Gründe für die Wahl der CPTs werden im Kapitel 0 im Detail beschrieben.

2.3.5.1. **PhoneGap**

Das PhoneGap Projekt entstand 2008 in Rahmen eines iPhoneDevCamp Ereignisses. Daraus bildete sich kurz darauf das amerikanische Unternehmen Nitobi. Im Oktober 2011 wurde Nitobi von Adobe übernommen und in Apache Cordova umbenannt.

Die zuvor geltende MIT Lizenz wurde durch die Apache 2.0 Lizenz ersetzt. Bei dem Einsatz von PhoneGap ist man dadurch verpflichtet der entwickelten Software eine Kopie der Lizenzvereinbarung und einen Hinweis auf die Apache 2.0 Lizenz beizufügen. PhoneGap ist heute einer der bekanntesten Cross-Plattform Tools [VisionMobile 2012] [Frei 2012].

Technologie
PhoneGap richtet sich an Web Entwickler, die native Smartphone Apps für diverse Plattformen wie iOS, Android, Blackberry, Symbian, Bada oder Windows Phone entwickeln wollen und diese über die nativen App Stores distribuieren möchten. Dazu können Standardentwicklungsumgebungen wie Eclipse oder Dreamweaver eingesetzt werden.

PhoneGap verwendet von den fünf vorgestellten Technologieansätzen den Web-To-Native Wrapper-Ansatz. Als Paradebeispiel für ein Tool dieser Art unterstützt PhoneGap eine Reihe von nativen Funktionalitäten. Es werden aber nicht alle Funktionalitäten auf allen Plattformen gleichermaßen abgedeckt [VisionMobile 2012] [Frei 2012]. Der Kompass wird bei Blackberry Betriebssystemen und auf Geräten mit Symbian beispielsweise nicht unterstützt [CordovaDoc 2013].

In dem Build Prozess einer PhoneGap App wird der geschriebene Web-Code, zusammen mit den PhoneGap Bibliotheken, in einen nativen Applikations-Wrapper eingepackt. Nach erfolgter Installation auf einem Gerät werden die User Interfaces von der nativen JavaScript Engine und der Browser-Komponente gerendert. Bei den Applikationen, die mit PhoneGap erstellt werden, handelt es sich demzufolge um hybride Web Apps.

Marktsituation und Positionierung
Im November 2011 wurde PhoneGap mehr als 600.000 Mal heruntergeladen und bereits tausende Apps damit entwickelt. Darunter fällt zum Beispiel auch die App des bekannten Social Business Netzwerks LinkedIn. In dem Bericht von VisionMobile wurde durch Umfragen festgestellt, dass PhoneGap das meist verwendete Cross-Plattform-Tool ist. Dabei wurden Kriterien ermittelt, die für Entwickler bei der Tool-Wahl besonders ausschlaggebend waren [VisionMobile 2012].

Eine der wichtigsten Gründe für die Entscheidung eines Tools war, dass die Zielplattform unterstützt wird. PhoneGap bedient alle wichtigen Smartphone Plattformen und verursacht dazu auch geringe Kosten, was ebenfalls ein wichtiger Grund für die Tool-Wahl war [VisionMobile 2012]. Das können Gründe sein, die zu einer so hohen Verbreitung beitragen. Das hybride Web App Paradigma von PhoneGap wurde eine Kernkomponente vieler anderer Cross-Plattform-Tools wie beispielweise von AppMobi, Worklight, aber auch noch von vielen anderen [VisionMobile 2012].

Kriterien	Beschreibung
Besonderheiten	minimalistisches Framework
Dokumentation	Ausführlich, große Entwicklergemeinde
Firma	Nitobi (2011 von Adobe übernommen)
Gegründet	2008

Input	HTML5, CSS, JS
Entwicklungsumgebung	Web-Entwicklungstools
Lizenz	Apache 2.0 (open source)
Min. Veröffentlichungskosten	N/A
Output	Hybride Web Apps
Tool-Typ / Technologieansatz	Web-To-Native Wrapper
Unterstütze Smartphone Plattformen	Android, Bada, Blackberry, iOS, Symbian, Windows Phone 7, WebOS (Dabei eingeschränkt: Blackberry, WebOS, Symbian und Bada)
Ziel Anwendung	Rich Internet Anwendungen

Tabelle 3 Übersicht der Informationen zu PhoneGap [Frei 2012][VisionMobile 2012][CordovaDoc 2013]

2.3.5.2. **Appcelerator (Titanium)**

Appcelerator wurde 2006 in Kalifornien gegründet. Das Hauptprodukt Titanium beinhaltet eine IDE und durch eine eigene Runtime eine Betriebssystems-Abstraktionsschicht. Es wurde 2008 für Windows, Mac und Linux Plattformen veröffentlicht und 2009 kamen die mobilen Plattformen Android und iOS hinzu [VisionMobile 2012].

Mit den neuesten drei Übernahmen wurde der Markt durch Appcelerator verdichtet. Appcelerator übernahm im Januar 2011 Aptana, wodurch sie eine Eclipse-basierte IDE mit 1,6 Millionen Usern erwarben. Im Oktober desselben Jahres übernahmen sie auch Particlecode mit dem Ziel die Plattformbandbreite in Bezug auf Spiele und dem mobilen Web zu erweitern.

Die jüngste Übernahme von Cocoafish bewerkstelligte Appcelerator im Februar 2012, um zahlreiche Cloud-basierte Dienste zu integrieren. Zu den Diensten gehören unter Anderem die Datenspeicherung, Push Benachrichtigungen und Messaging. Das Titanium SDK fällt wie PhoneGap unter die Apache 2.0 Open Source Lizenz [VisionMobile 2012].

Technologie

Titanium Applikationen werden mit HTML, JavaScript und CSS geschrieben und bieten Support für PHP, Ruby und Python. Die Appcelerator APIs bieten Zugriff auf native Funktionalitäten, Benutzerschnittstellen und optionale Module. Optionale Module sind beispielsweise im Bereich Analyse vorhanden. Das JavaScript wird mit nativen Bibliotheken verbunden und daraufhin in Bytecode kompiliert. Daraufhin erstellt der Compiler des jeweiligen SDKs aus allem das Package für die Zielplattform. Nach Angaben von Appcelerator kann eine gemeinsame Code-Basis von 80% bei mobilen Plattformen erreicht

werden. Die gemeinsame Codebasis bietet Vorteile, doch müssen für jede Zielplattform die Nutzeroberflächen individuell geschrieben werden [VisionMobile 2012] [Frei 2012].

Die Output-Applikation beinhaltet hauptsächlich nativen Code. Zusätzlich beinhaltet die Applikation eine integrierte JavaScript Runtime und eine Webkit Rendering-Engine, welche den Code rendert und ausführt [VisionMobile 2012] [Frei 2012].

Appcelerator bietet sowohl native, hybride als auch Web Apps als Output-Formate. Dabei ist Titanium bei mobilen Plattformen auf Android und iOS begrenzt. Blackberry befindet sich zum derzeitigen Zeitpunkt noch im Beta-Stadium [VisionMobile 2012].

Marktsituation und Positionierung

Nach Appcelerator wurden bereits über 35.000 Apps veröffentlicht und auf über 40 Millionen Geräten deployed. Sie besitzen einen Kundenstamm von 250.000 Mobile-Entwicklern und gewannen durch die Übernahmen von Aptana 1,6 Millionen Webentwickler hinzu. Ein Beispiel für eine Titanium-App ist die NBC IPad App.

Appcelerator ist nicht für aufwändige grafische Spiele ausgelegt sondern eher für interaktive und Rich-Information Applikationen [VisionMobile 2012]. Appcelerators Zielgruppe sind Web Entwickler. Im Gegensatz zu hybriden Lösungen, wie sie beispielsweise mit PhoneGap oder Sencha erstellt werden, sollen Titanium Apps eine bessere Performance bieten [VisionMobile 2012].

Durch die Serie an Übernahmen hat Appcelerator eine mobile Entwicklungsplattform für Unternehmen geschaffen, die anderen Mobile App Plattformen wie Worklight ähnelt. Appcelerator ist jedoch auf Rich Anwendungen ausgerichtet und weniger auf große Enterprise Apps [VisionMobile 2012].

Kriterien	Beschreibung
Besonderheiten	Zugriff auf gerätespezifische Elemente für die Gestaltung der Nutzungsoberfläche
Dokumentation	Umfangreich aber teilweise nicht aktuell
Firma	Appcelerator
Gegründet	2006
Input	HTML, CSS, JavaScript
Entwicklungsumgebung	Eclipse basierte IDE Titanium Studio (übernommen von Aptana) oder andere IDEs
Lizenz	Apache 2.0 (open source)
Min. Veröffentlichungskosten	Kostenlos (limitierte APIs)
Output	Native (hauptsächlich), Hybrid and Web Apps (beta)
Tool-Typ / Technologieansatz	Runtime
Unterstütze Smartphone Plattformen	Android, iOS, Blackberry (beta)
Ziel Anwendung	Rich-Information und Rich Media Apps für B2C, B2B and B2E

Tabelle 4 Übersicht der Informationen zu Appcelerator (Titanium) [Frei 2012] [VisionMobile 2012]

2.3.5.3. IBM (Worklight)

Die Firma Worklight wurde 2006 von Shahar Kaminitz gegründet. Bis zur Veröffentlichung der mobilen Enterprise Application Plattform im Jahr 2009 sind weitere Produkte der Firma veröffentlicht worden. Im Februar 2012 wurde Worklight dann von ihrem strategischen Sales Partner IBM übernommen [VisionMobile 2012].

Technologie

Worklight ist eine Plattform zum Managen von Enterprise Apps. Die Anwendungen werden mit den Webtechnologien HTML, CSS und JavaScript entwickelt und können daraufhin sowohl als Desktopanwendungen auf Windows, Mac und Linux deployed werden als auch als Webanwendung (inkl. Facebook) oder aber auch als mobile native Anwendungen für iOS, Android, Blackberry und Windows Phone [VisionMobile 2012].

Bei der Entwicklung können JavaScript Frameworks wie Sencha oder JQuery Mobile ergänzt werden. Die Entwicklung ist bei Worklight aber nicht auf Web-Technologien beschränkt. Es kann auch nativer Code geschrieben werden. Dies kann entweder in Kombination mit den Web-Technologien geschehen, wodurch hybride-Mix Apps entstehen, oder es können gänzlich native Apps geschrieben werden, die dann aber auch nur plattformspezifisch sind [Worklight 2012] [VisionMobile 2012]. Es können folglich sowohl Web-Apps, hybride als auch native Apps mit Worklight entwickelt werden [Worklight 2012].

Hybride Apps, die durch Worklight entstehen, werden einer anpassbaren nativen Runtime-Shell verpackt. Die Runtime beinhaltet dabei eine Web-To-Native Device API Brücke und Runtime Libraries. Worklight setzt auf PhoneGap und verwendet eine qualitätsgesicherte Untermenge der PhoneGap Libraries, um native Funktionalitäten zu nutzen [Worklight 2012].

Worklight beinhaltet ein umfassendes Angebot an verschiedenen Werkzeugen und Unterstützungen. Es beinhaltet eine Entwicklungsumgebung, Back-End Integration, Managementunterstützung in Bezug auf Entwicklung, Runtime sowie die Möglichkeit Analysen durchzuführen [VisionMobile 2012].

Marktsituation und Positionierung
Worklight bedient eine breite Masse an Kunden. Die Kunden stammen sowohl aus dem Finanzbereich, Gesundheitswesen, Technologiebereich als auch aus der Energieindustrie. Das CPT hat als Zielgruppe Firmen im Bereich System Integration und Softwarehersteller die B2B Apps herstellen. Es konkurriert mit anderen CPTs wie RhoMobile oder Xamarin, die ebenfalls auf Enterprise Anwendungen ausgerichtet sind, aber auch mit anderen auf Web-Code basierenden Lösungen, wie Sencha. Da Worklight sich um eine Mobile Application Plattform handelt, die viele zusätzliche Unterstützungen anbieten, konkurriert es aber hauptsächlich auch mit den Mobile Application Plattformen anderer Hersteller wie Verivo oder Kony [VisionMobile 2012].

Wie zuvor erwähnt bietet Worklight verschiedenste Möglichkeiten Apps zu entwickeln. Die Möglichkeiten reichen von Web-Apps über hybriden Web Apps und hybriden Mix Apps bis hin zu nativen Apps. Diese Besonderheit ermöglicht es Projekte flexibler an die Fähigkeiten eines Unternehmens und deren Mitarbeiter anzupassen [VisionMobile 2012].

Worklight	
Kriterien	**Beschreibung**
Besonderheiten	Unterstützt jede App-Typ Entwicklung
Dokumentation	Verschiedene PDF Dokumente stehen zur Verfügung. Online Dokumentation wäre praktikabler
Firma	Worklight, Übernommen von IBM im Februar 2012
Gegründet	2006
Input	HTML, CSS, JS plus PhoneGap APIs und nativer Code
Entwicklungsumgebung	Eclipse-basierte IDE
Lizenz	Produktabhängig: Developer Edition (no-cost), Consumer und Enterprise Edition (kostenpflichtige Lizenzen)
Min. Veröffentlichungskosten	N/A
Output	Native, Hybrid, Web
Tool-Typ / Technologieansatz	Enterprise Application Plattform
Unterstütze Smartphone Plattformen	Android, Blackberry, iOS, Windows Phone 7
Ziel Anwendung	Hauptziel sind B2C, B2B und B2E Enterprise Apps und Rich Internet Apps

Tabelle 5 Übersicht der Informationen zu IBM Worklight [VisionMobile 2012][Worklight 2012]

2.3.5.4. **Sencha**

Sencha entstand 2007 als Gemeinschaftsinitiative um drei bekannte UI Bibliotheken Ext JS, jQToch und Raphaël zu kombinieren. Erst 3 Jahre später, im Juni 2010, wurde aus ihr die Firma Sencha. Sie unterteilt sich in den kommerziellen Teil Sencha und der Non-Profit Entität Sencha Labs. Dabei ist Sencha Labs im Besitz des intellektuellen Guts für Open Source Softwareelemente [VisionMobile 2012].

Technologie
Sencha stellt verschiedene Werkzeuge bereit, die Java Script Entwicklern die Erstellung vom Rich Web Applikationen ermöglicht. Dabei hat Sencha als Zielgruppe, sowohl Entwickler als auch Designer. Für jede Zielgruppe existiert eine Produktlinie [VisionMobile 2012].

Die erste Produktsparte ist an Entwickler gerichtet und bietet Frameworks für JavaScript Entwickler. Dazu zählt das verwendete Sencha Touch Framework für den Bereich mobile, Ext JS für den Bereich Desktop und Ext GWT für Browser Apps, die auf dem Google Web Toolkit basieren [VisionMobile 2012].

Die zweite Produktlinie beinhaltet unter anderem den Sencha Animator, der es Designern ermöglicht CSS3 basierte Animationen für diverse Geräte zu entwickeln [VisionMobile 2012].

Sencha Touch beinhaltet verschiedene Komponenten um Web Apps oder aber auch hybride Apps zu erstellen. Dazu gehört die Unterstützung von Touch-Interaktionen, Animationen und Datenmanipulationen. Es werden die nativen Plattformen Blackberry OS, Android und iOS unterstützt. Dabei ist zu beachten, dass Sencha 1.0 basierte Apps in einem Browser laufen und somit nur Zugriff auf die Schnittstellen der jeweiligen Plattformbrowser haben. Die Sencha Touch 2.0 basierten Apps unterstützen dagegen ähnlich wie PhoneGap es tut ein Wrapping von Web Apps. Somit unterstützt Sencha Touch 2.0 von anderen CPTs unabhängig hybride Apps. Senchas Produkte sind nach Verwendungszweck unter verschiedenen Lizenzen verfügbar [VisionMobile 2012].

Marktsituation und Positionierung
Sencha hat als Basis ca. 1,6 Millionen JavaScript Entwickler und ca. 300.000 registrierte Gemeinschaftsmitglieder [VisionMobile 2012].

Die Sencha Desktop Tools richten sich an den Unternehmer Markt, während die mobilen Produkte sich eher an den Konsumenten richten. Die geringen Kosten und die vielfältigen UI Möglichkeiten sind meist ein Grund für die Auswahl dieser Frameworks. Kritisiert werden die Performance und die schwache Unterstützung von nativen Benutzeroberflächen. Performanceschwächen sind meist auf den Gerätebrowser zurückzuführen [VisionMobile 2012].

Sencha	
Kriterien	**Beschreibung**
Besonderheiten	Sowohl als Framework als auch eigenständig nutzbar (Version 2)
Dokumentation	Für die 1.X Versionen sehr umfangreich und für die 2.X Versionen ausreichend gut (Kapitel 8.2.1.3)
Firma	Sencha
Gegründet	2007
Input	JavaScript
Entwicklungsumgebung	Beliebige IDE, Visuelle Designer
Lizenz	Produkte unter den Lizenzen: Zero Cost Lizenz (für kostenlose Apps), Eine Kommerzielle zahlungspflichtige Lizenz und eine Open Source Lizenz (GPL3)
Min. Veröffentlichungskosten	Kostenlos (Framework)
Output	Web, Hybrid
Tool-Typ / Technologieansatz	JavaScript Framework, Web-To-Native Wrapper (Version 2)
Unterstütze Smartphone Plattformen	Android, Blackberry, iOS
Ziel Anwendung	Enterprise Apps und Rich Web Apps

Tabelle 6 Übersicht der Informationen zu Sencha [VisionMobile 2012]

2.4 Quasar

Dieses Unterkapitel befasst sich mit der von Capgemini (ehemals sd&m) entwickelten Softwarearchitektur „Quasar“. Quasar steht für Qualitätssoftwarearchitektur und beschreibt und präzisiert besonders wichtige Regeln und Mechanismen der Softwaretechnik und erklärt diese auf folgenden vier Ebenen zum Standard [Siedersleben 2004].

- **Ideen und Konzepte:** Diese Ebene beinhaltet Grundprinzipien der Softwaretechnik wie die Trennung von Zuständigkeiten und das Denken in Schnittstellen und Komponenten [Siedersleben 2004].
- **Begriffe:** Definiert technikneutrale Begriffe, um eine Kommunikation über Softwarearchitektur ohne die Bindung an technische Details zu ermöglichen [Siedersleben 2004].
- **Standardarchitektur und Standardschnittstellen:** Definition von Standardschnittstellen für verschiedene Themen, wie beispielsweise für Datenbankzugriff [Siedersleben 2004].
- **Standardkomponenten:** Projektübergreifende Standardkomponenten werden entwickelt und stehen bereits zur Verfügung. Bei diesen Komponenten steht jedoch die Schnittstelle im Vordergrund, um Implementierungen leichter austauschbar zu halten [Siedersleben 2004].

Die Softwarearchitektur eines Systems hat direkte Auswirkungen auf die Qualität und die Kosten des Systems, daher werden aufbauend auf den Regeln von Quasar die entstehenden Anwendungen in Hinblick auf die Softwarearchitektur betrachtet. Die beschriebenen vier Ebenen zeigen auf, dass Quasar weitaus mehr ist als das, was im Rahmen dieser Arbeit behandelt werden kann. Im Folgenden werden ausgewählte Aspekte von Quasar beschrieben, die in Bezug auf diese Arbeit eine besondere Relevanz haben.

2.4.1 Komponenten

Jedes System besteht aus unterschiedlichen Bestandteilen, die man auch als Komponenten bezeichnet. Nur mit ihnen ist es möglich große Systeme zu entwerfen, zu bauen und zu betreiben [Siedersleben 2004].
Der Komponentenbegriff ist einer der wichtigsten Begriffe der Softwarearchitektur. Eine Komponente zeichnet sich nach Siedersleben durch sechs Merkmale aus [Siedersleben 2004]:

1. Eine Komponente exportiert eine oder mehrere Schnittstellen, die im Vertragssinne garantiert sind.

2. Sie importiert andere Schnittstellen und wird erst dann lauffähig, wenn alle importierten Schnittstellen zur Verfügung stehen.
3. Eine Komponente versteckt ihre Implementierung und ist daher durch eine Komponente mit denselben exportierten Schnittstellen austauschbar.
4. Sie eignet sich als Einheit der Wiederverwendung und macht nur minimale Annahmen zur Umgebung in der sie läuft.
5. Eine Komponente kann andere Komponenten enthalten, so dass neue Komponenten durch die Komposition mehrerer entstehen können.
6. Sie ist neben der Schnittstelle die wesentliche Einheit des Entwurfs, der Implementierung und damit auch der Planung.

2.4.2 Schnittstellen

Wie im vorangehenden Kapitel erwähnt, ist die Schnittstelle neben der Komponente die wesentliche Einheit des Entwurfs-, Implementierungs- und Planungsprozesses einer komponentenbasierten Software-Architektur [Siedersleben 2004].
Sie hat die Aufgabe Komponenten miteinander oder Komponenten mit dem Benutzer zu verbinden. Dementsprechend spricht man bei einer Schnittstelle zwischen Komponente und Benutzer von „Benutzerschnittstelle“ und bei einer Verbindung zweier Komponenten von einer „Programmschnittstelle“. Programmschnittstellen werden mit einer Menge von Methoden einschließlich folgender Eigenschaften definiert [Siedersleben 2004]:

- **Syntax:** Methodensignatur mit Angaben zum Rückgabewert, Argumenten, in/out und deren Typen
- **Semantik:** Beschreibt die Funktionalität der Methode
- **Protokoll:** Beschreibt, ob die Ausführung synchron oder asynchron stattfindet
- **Nichtfunktionale Eigenschaften:** Beschreibt Eigenschaften, wie Performance, Robustheit, Verfügbarkeit und bei Web-Anwendungen eventuelle Kosten.

Programmschnittstellen beinhalten für Programmierer alles Relevante in Bezug auf die Nutzung einer Komponente, so dass er sie versteht und weiß was er zu beachten hat.
Jede Komponente exportiert mindestens eine Schnittstelle, da sie sonst nicht verwendbar wäre [Siedersleben 2004].

2.4.3 Software Kategorien

Software-Systeme befassen sich mit der fachlichen Anwendung und verwenden eine technische Basis, wie beispielsweise ein Betriebssystem oder Datenbanken. Die Trennung von Zuständigkeiten ist eine der Leitlinien für eine gute Software-Architektur. Nach Siedersleben setzt sich ein System aus unterschiedlichen Komponenten zusammen, die

jeweils einer Kategorie zugeordnet werden können [Siedersleben 2003] [Siedersleben 2004]:

- **O-Software:** Sie ist Anwendung- und Technikneutral. Dazu zählen beispielsweise Klassenbibliotheken, die sich mit Strings befassen. Komponenten dieser Kategorie sind ideal wiederverwendbar aber alleine nutzlos.
- **A-Software:** Sie ist anwendungsbestimmt, aber unabhängig von Technik und kennt fachliche Begrifflichkeiten
- **T-Software:** Sie ist anwendungsunabhängig und durch Technik bestimmt. T-Software kennt mindestens eine technische Schnittstelle, wie beispielsweise JDBC zum Datenbankzugriff.
- **AT-Software:** Sie ist sowohl durch die Anwendung als auch die Technik bestimmt. Sie ist schwer zu warten, schwer änderbar und kaum wiederverwendbar und soll daher idealerweise vermieden werden.
- **R-Software:** Sie transportieren fachliche Objekte in externe Repräsentationen und wieder zurück. Ein XML-Parser ist ein Beispiel für eine R-Software wohingegen XML ein Beispiel für die externe Repräsentation ist.

Die genannten fünf Kategorien formalisieren die Trennung der Zuständigkeiten. Jede A- und T-Komponente sollte sich demnach auch nur mit einem Anwendungsthema bzw. einer technischen API befassen. Durch die Trennung der Zuständigkeiten werden die Abhängigkeiten reduziert, was zur Verbesserung des Systems beiträgt [Siedersleben 2003] [Siedersleben 2004].

2.4.4 Softwarearchitekturen

Die Trennung von A-Software und T-Software wirkt sich auf die Softwarearchitektur und den Entwicklungsprozess aus. Man leitet aus der Anwendungsspezifikation die Architektur der Anwendung ab und implementiert diese als A-Softwarekomponenten gegen Schnittstellen der Kategorie 0. Die Implementierung dieser Schnittstellen ist Aufgabe der T-Komponente, die im Rahmen der T-Architektur definiert wird [Siedersleben 2004].
Die Architektur eines Informationssystems bezeichnet man als „Systemarchitektur“. Diese ist von verschiedenen Perspektiven aus betrachtbar [Siedersleben 2004]:

- **TI-Architektur:** TI steht für technische Infrastruktur. Die TI-Architektur beschreibt die technische Infrastruktur des Informationssystems. Dabei umfasst sie sowohl physische Geräte (Rechner, Netzleistung, Drucker usw.), die installierte Systemsoftware (Betriebssystem, Anwendungsserver usw.), die verwendeten Programmiersprachen als auch das Zusammenspiel von Hard- und Systemsoftware [Siedersleben 2004].
- **Softwarearchitektur:** Sie beschreibt das Zusammenspiel der Schnittstellen und Komponenten aus denen das System besteht. Dabei unterscheidet man zwischen zwei Architektursichten [Siedersleben 2004]:

- **A-Architektur:** A steht für Anwendung. Die Anwendungsarchitektur strukturiert die Software aus der Anwendungssicht. Dabei werden die Funktionen und das Komponenten Zusammenspiel ohne Bezug zur Technik entworfen. Die Strukturierung findet dabei mit genauen Kenntnissen über die fachlichen Abläufe statt [Siedersleben 2004].
- **T-Architektur:** T steht für Technik. Die Technikarchitektur ist neben der Anwendungsarchitektur eine Architektur, die jedes Informationssystem besitzt. Sie stellt den sicheren Umgang mit der Technik sicher und verbindet die A-Architektur mit der TI-Architektur. und beschreibt alle Komponenten eines Systems, die Anwendungsunabhängig sind. Dazu zählen die Zugriffschicht, Fehlerbehandlung und vieles mehr. Außerdem beschreibt die T-Architektur noch die Verteilung der Software und die Einheiten (.jar-Dateien, DLLs), in denen das fertige System geliefert wird [Siedersleben 2004].

2.4.5 Komponentenbasierte Client-Architektur

„Die Komplexität von Clients wird meist unterschätzt. Ein Client besteht eben nicht nur aus ein paar Masken. Es steckt noch eine Menge Funktionalität unter der „Oberfläche". Eine geeignete Strukturierung hilft die Komplexität beherrschbar zu machen." [Haft u.a. 2007]

Clients in Client-Server Systemen weisen meist eine hohe Komplexität auf, die unterschätzt wird. Die Unterschätzung der Komplexität ist darauf begründet, dass Clients bei der Betrachtung nur auf die visuelle Darstellung begrenzt werden und die innere Struktur in der Betrachtung außen vor gelassen wird. Dabei nehmen Clients häufig mehr als 50% des Gesamtentwicklungsaufwandes ein. Durch die Unterschätzung entstehen komplexe Client-Monolithen, die schlecht angepasst, erweitert und wartbar sind. Die komponentenbasierten Client-Architektur macht die Komplexität durch eine Strukturierung und Aufteilung der Clientfunktionalitäten in Komponenten beherrschbarer. Somit vermeidet sie die Entstehung solcher Monolithen [Däschlein u.a. 2010] [Haft u.a. 2007].

2.4.5.1. **Client**

Der Client wird als oberste Schicht der „Drei-Schichten Architektur" verstanden. Sie stellt dem Benutzer die Bedienoberfläche zur Verfügung und greift auf dem Anwendungsserver zu. Diese Schicht wird auch oft als Benutzerschnittstelle oder Präsentationsschicht bezeichnet, doch diese Begriffe reduzieren die Aufgaben des Clients nur auf einzelne Aspekte und werden dem Client Begriff daher nicht gerecht [Haft u.a. 2007] [Haft u.a. 2005].

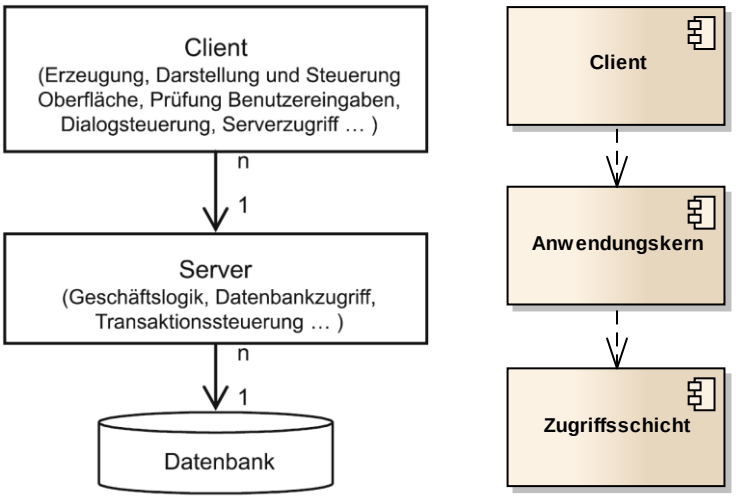

Abbildung 6 Drei-Schichten Architektur links: [Haft u.a. 2007] rechts: [Haft u.a. 2005].

Die Aufgaben der Client-Schicht bzw. des Clients umfasst [Däschlein u.a. 2010]:

- Das Anzeigen und erstellen der visuellen Darstellung
- Die Anzeige von Daten und die Bereitstellung von Aktionen in der visuellen Darstellung
- Entgegennahme und syntaktische Validierung von Benutzereingaben
- Steuerung der visuellen Darstellung (Dialognavigation)

Der Client-Begriff umfasst dabei mehrere Aspekte der TI-Architektur, wie die Nutzung eines Anwendungsservers, die Ausführung auf einem zentralen oder dezentralen Rechner und die Nutzung einer GUI-Bibliothek zur visuellen Darstellung [Haft u.a. 2005]
Des Weiteren wird im Client in der Regel der nicht-persistente Zustand der Benutzer-Sitzung gehalten [Däschlein u.a. 2010].

2.4.5.2. **Dialog**

„Ein Dialog bildet eine Einheit in der Mensch/Maschine-Kommunikation derart, dass darin für den Benutzer zusammenhängende Daten und Funktionen verfügbar sind. Der Dialog ist die Bearbeitungseinheit des Anwenders. Jeder Dialog unterstützt einen oder mehrere Anwendungsfälle und präsentiert sie in geeigneter Form (z.B. Fenster oder HTML-Seite) am Bildschirm.“ [Siedersleben 2004] [Däschlein u.a. 2010]

Diese Definition beschreibt nicht was ein sinnvoller Dialog ist sondern nur, dass ein Dialog ein Teil der Benutzeroberfläche abbildet. Wie auch Haft u.a., definiert Siedersleben den Dialog als das wesentliches Element zur Strukturierung bzw. Planung und Entwurf des Clients [Siedersleben 2004] [Haft u.a. 2005]. Diese Strukturierung in Dialoge führt zu einer beherrschbareren Komplexität des Clients.

Die Benutzeraktionen zwischen Benutzer und Daten können sehr verschieden sein und sind auch unterschiedlich komplex. Die vielfältigen und komplexen Aufgaben spiegeln sich dabei auch in der Architektur des Clients wieder [Haft u.a. 2005].

2.4.5.3. **Dialogkomponente**

Dialoge werden nach Quasar in Form von Anwendungskomponenten umgesetzt, die als Dialogkomponente bezeichnet werden. Es handelt sich dabei genauer um einen fachlich abgeschlossenen Teil der Benutzeroberfläche, die Benutzerinteraktionen entgegennimmt und die Kommunikation mit dem Benutzer steuert. Sie sind in sich abgeschlossen, so dass sie neben den Oberflächen auch die zugehörige Logik, Daten und Zustände beinhalten. Sie können zudem auf Dienste anderer Komponenten zugreifen [Däschlein u.a. 2010].

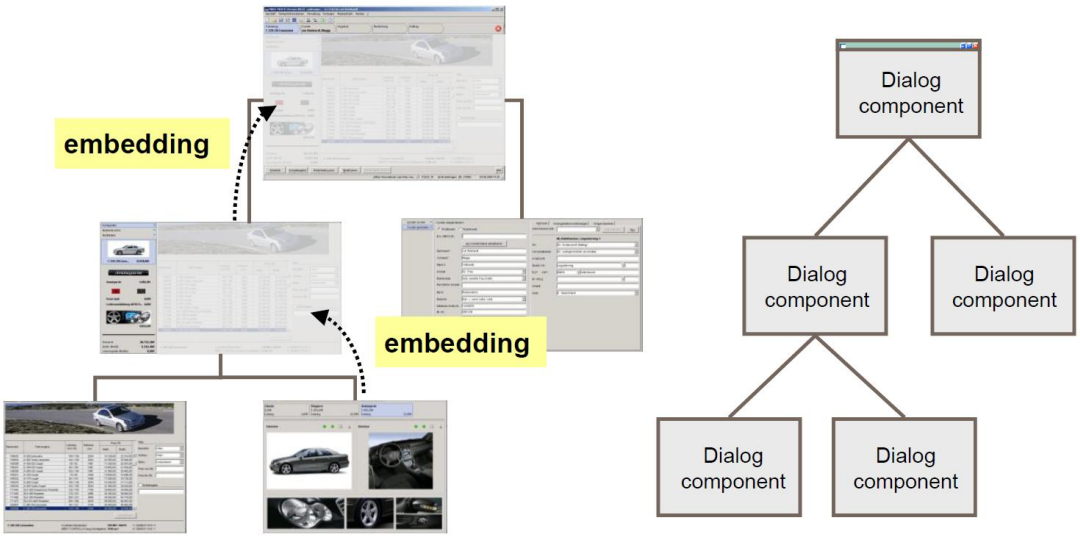

Abbildung 7 Unterteilung eines Fahrzeugkonfigurators in einzelne Dialogkomponenten die visuell in einer eingebettet werden [Däschlein u.a. 2010].

Abbildung 7 zeigt wie Benutzeroberflächen durch Dialogkomponenten strukturiert und umgesetzt werden sollten. Oberflächen können, wie in diesem Beispiel, visuell unterteilt werden. Die Unterteilung kann dabei hierarchische Strukturen annehmen. Findet eine solche Unterteilung statt müssen diese Teile auch wieder zusammengefügt werden. Diese

Zusammenfügung bezeichnet man als visuelle Einbettung. Dabei muss nicht zwangsweise eine visuelle Unterteilung wie im Beispiel stattfinden. Die Dialogkomponenten können beispielsweise auch in einem eigenen Fenster dargestellt werden. In den Ausarbeitungen von Däschlein und Haft wird die Hierarchie der Dialogkomponenten noch etwas ausführlicher unterteilt und beschrieben.

2.4.5.4. **Dialogkomponenten-Architektur**

Dialoge können wie bereits erwähnt unterschiedlich stark komplex sein. Es existiert aus diesem Grund nicht eine einzelne Architekturlösung für alle Dialogkomponenten sondern mehrere, aus denen ein Client-Architekt die Spezifische für den jeweiligen Dialog wählt [Haft u.a. 2005].

In Abbildung 8 ist ein möglicher Aufbau einer Dialogkomponente dargestellt. Er beinhaltet typische Unterkomponenten die im Folgenden genauer erläutert werden.

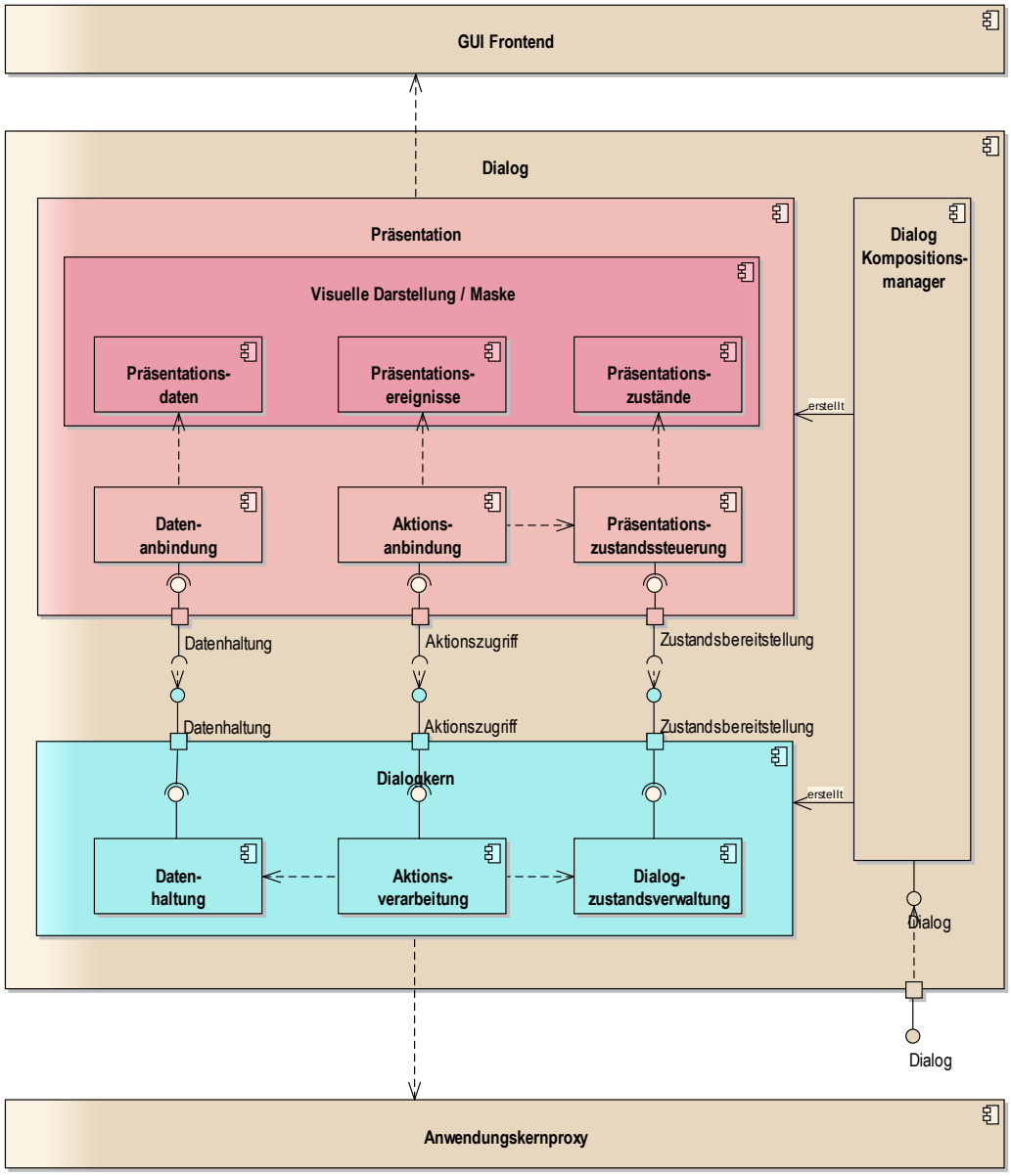

Abbildung 8 T-Architektur von Dialogen mit Dialogkompositionsmanager [Haft u.a. 2005] [Däschlein u.a. 2010] [Haft u.a. 2012].

2.4.5.5. **Präsentation**

Die Präsentation ist für die Bereitstellung der visuellen Darstellung verantwortlich. Sie erzeugt und stellt die Benutzeroberfläche dar, nimmt die Benutzerinteraktionen entgegen und bindet diese an den Dialogkern. Die Präsentation ist dadurch immer abhängig zu der

verwendeten UI-Bibliothek. Die Daten und die Oberfläche werden lokalisiert und internationalisiert angezeigt. Außerdem finden mit eingeschränktem Umfang syntaktische und semantische Validierungen statt [Däschlein u.a. 2010].

Däschlein und Haft beschreiben neben der ersten Aufgabenkategorie der Bereitstellung der visuellen Darstellung eine weitere Kategorie, die für die Anbindung der visuellen Darstellungen an den Dialogkern verantwortlich ist. Die Anbindung ist dabei in die drei Bereiche Datenanbindung, Präsentationszustandssteuerung und der Präsentationsaktionssteuerung aufgeteilt [Däschlein u.a. 2010] [Haft u.a. 2005].

Datenanbindung

Man kann zwei Arten der Datenhaltung in einem Dialog unterscheiden. Die Dialogdatenhaltung befindet sich im Dialogkern und wird im Folgenden als „Datenhaltung“ bezeichnet. Unterscheiden tut sich diese Datenhaltung von der Datenhaltung der Präsentation. Die Daten aus beiden Bereichen können sich beispielsweise aus ortspezifischen Datenformats-Gründen unterscheiden.

Die Datenanbindung (Data Binding) ist dabei für die Synchronisation der Daten zwischen den beiden Datenhaltungen verantwortlich. Ist ein Observer Pattern in der Dialogdatenhaltung implementiert, kann sich die Datenbindung als Observer registrieren und die Präsentationsdaten sofort bei Änderungen aktualisieren. Dies gilt auch umgekehrt. Ein Beispiel für ortspezifisch unterschiedliche Datenformate ist das Datumsformat. Während die Dialogdaten eine neutrale Darstellung haben (z.B. Datumsformat: „yyyymmtt“) können die Präsentationsdaten lokal auf Deutschland bezogen das Datumsformat „tt.mm.yyyy“ haben. Um dies zu erreichen sind verschiedene Vorgänge notwendig. Die Adaption, die Konvertierung der Daten durch Formatierung und parsen, aufspalten oder zusammenfassen sind ebenfalls Aufgaben der Datenanbindung. Auch die syntaktische Validierung zählt zu den Aufgaben, da syntaktisch korrekte Eingaben Voraussetzung für das erfolgreiche Parsen von Daten sein können [Haft u.a. 2005].

Aktionsanbindung

Die Aktionsanbindung (Action Binding) soll Benutzerinteraktionen empfangen und falls nötig Aktionen in der Dialogkernschicht auslösen. Benutzerinteraktionen, die durch Präsentationsereignisse empfangen werden, können auf verschiedene Weisen ausgelöst werden. Ein Buttonklick oder das Ändern von Werten sind Beispiele dafür. Sind durch die Benutzeraktion Anpassungen nur in der Präsentationsschicht notwendig, können diese direkt in der gleichen Schicht durch die Präsentationsaktionsverwaltung verarbeitet werden. Alle anderen werden zur Dialogkernschicht weitergegeben. Dort sind Aktionen auf fachliche Funktionen abgebildet. Da allerdings nicht immer Aktionen auf Funktionen abgebildet sind, wie beispielsweise bei einem Speichern von nicht geänderten Daten, können Aktionen mit einem Ausführbarkeitszustand versehen werden. Die Anbindung des

Ausführbarkeitszustandes wird dabei nicht durch die Aktionsanbindung sondern durch die Präsentationszustandsteuerung geregelt.

Die Präsentationsaktionssteuerung fasst die Präsentationsaktionen zusammen, die keine Zustände ändern. Beispiele dafür wären die Sortierung der Präsentationsdaten oder eine Umgestaltung der Oberfläche. Sie umfasst also die Verarbeitung von Ereignissen, die nicht vom Dialogkern verarbeitet werden, sondern alleinig von der Präsentation. Dabei sind präsentationsinterne Verarbeitungen gemeint, die auch nicht von GUI-Libraries oder der Präsentationszustandssteuerung übernommen werden [Haft u.a. 2005].

Präsentationszustandssteuerung
Die Präsentationszustandssteuerung regelt den Präsentationszustand. Der Präsentationszustand umfasst alle Zustände der visuellen Darstellung. Dabei leitet sich der visualisierte Zustand aus den verschiedenen Zuständen des Dialogkerns und der Präsentation selbst ab. Dazu zählen die Zustände der fachlichen Daten, der Dialogdatenhaltung, der Ereignisverarbeitung und explizit definierte Zustände der Ereignisverarbeitung des Dialogkerns und die Zustände bestimmter Präsentationsdaten sowie Ergebnisse der syntaktischen Validierung. Die Anbindung an die Zustände des Dialogkerns sollten nach Möglichkeit über das Observer-Pattern realisiert werden, dabei würde sich die Präsentation bei dem Dialogkern registrieren. Diese Komponente wird auch als State Binding bezeichnet [Haft u.a. 2005].

Aktionen werden sowohl bei der Präsentationsaktionssteuerung als auch, der Präsentationszustandssteuerung durch die Aktionsanbindung ausgelöst, die beispielsweise bei dem Betätigen eines Buttons eine Aktion im Dialogkern auslöst.

2.4.5.6. **Dialogkern**

Der Dialogkern enthält die Fach- und Steuerungslogik, sowie den fachlichen Zustand des Dialogs. Er ist zudem, anders als die Präsentation, unabhängig von der GUI-Library und daher technikneutral. Die Komponententeile des Dialogkerns kommunizieren mit den Schnittstellen des Anwendungskerns über einen technischen Serverzugriffdienst [Däschlein u.a. 2010]. Däschlein u.a. beschreiben die Aufgaben des Dialogkerns in folgender Form detaillierter:

Datenhaltung
Die Datenhaltung stellt Daten der Dialogkernschicht zum Lesen und Ändern bereit. Im einfachsten Fall handelt es sich bei der Datenhaltung beispielsweise um eine Schnittstelle, vergleichbar mit einer Map. Die hier gespeicherten Daten müssen rein fachlicher Natur sein und dürfen somit keine Präsentationsdaten bereitstellen. So sollten bei der Auswahl von Checkboxen in der Präsentation eine List der fachlichen Objekte in der Datenhaltung hinterlegt sein und keine List mit booleschen Werten. Die Form der hier hinterlegen Daten

entspricht der Transportobjektform, die in der Schnittstelle des Anwendungskerns spezifiziert wurde. Damit unterscheiden sich die Daten der Präsentation und der Datenhaltung. Die Konvertierung der Daten zwischen der Präsentation und dem Dialogkern ist nicht Aufgabe der Datenhaltung, sondern Aufgabe der Datenanbindung in der Präsentationsschicht [Däschlein u.a. 2010].

Dialogzustandsverwaltung
Im Dialogkern können verschiedene Zustände vorhanden sein. Der Dialogzustand setzt sich aus all diesen Zuständen zusammen. Dabei können Zustände beispielweise Ausführbarkeitszustände sein, die das Ausführen von Aktionen nur ermöglicht, wenn eine vorherige Bedingung erfüllt ist. Ein Beispiel für einen Zustand der Dialogdatenhaltung wäre „Daten sind geladen“. Im einfachsten Fall werden Zustände im Dialogkern implizit verwaltet. Dies geschieht durch das Abfragen verschiedener Eigenschaften, woraufhin entsprechender Code ausgeführt wird. Eine explizite Zustandsverwaltung ist in so einem Fall nicht nötig. Daher ist die Zustandsverwaltung optional. In einigen Fällen wird aber eine explizite Zustandsverwaltung empfohlen. Einer der Beispiele ist, wenn das Abfragen von Eigenschaften überhandnimmt und es zu größeren If-Else-Kaskaden kommt [Däschlein u.a. 2010].

Aktionsverarbeitung
Die Aktionsverarbeitung (Action Processing) bietet in verschiedenen Formen die Ausführung von fachlichen Aktionen an. Die beiden gängigsten Varianten sind die Ausführung über Methoden oder über Kommandos. Bei der ersten Variante werden Methoden direkt von der Aktionsanbindung aufgerufen. Bei der zweiten Variante werden Aktionen über das Command Pattern angeboten. Dabei holt sich die Aktionsanbindung ein Kommando aus der Dialogkernschicht und ruft es auf. Der Vorteil besteht bei Kommandos darin, dass sie einen Ausführbarkeitszustand besitzen können, der über die Präsentationzustandsteuerung an Oberflächenelemente gebunden werden können [Däschlein u.a. 2010].

Dialogsteuerung
Die Dialogsteuerung ist für die Abfolge, das Erzeugen und Schließen von Dialogen verantwortlich. Die Steuerung kann entweder zentral in einer sogenannten Einbettungskomponente programmiert sein oder aber implizit. Bei der impliziten Steuerung der Dialogabläufe bestimmt jeder Dialog seinen Folgedialog auf Grund seines Zustands und der eigenen Daten. Dazu wird der Folgedialog erzeugt und initialisiert bevor der auslösende Dialog selbst geschlossen wird. Der implizite Weg reicht für die meisten Fälle aus [Däschlein u.a. 2010].

Anwendungskernzugriff
Fachliche Funktionen der Anwendungskernkomponente werden über Schnittstellen bereitgestellt, die aus Performance-Gründen auf den Client zugeschnitten sein sollte. Die Anwendungskernkomponentenschnittstellen werden über einen technischen Serverzugriffsdienst aufgerufen. Dieser Dienst sollte als Anwendungskernproxy implementiert werden, sofern es möglich ist. Die Nutzung dieses Proxys hat den Vorteil, dass Entwickler gegen die fachlichen Schnittstellen des Anwendungskerns implementieren, unabhängig von der dahinter liegenden Implementierung und der technischen Kommunikation zwischen Server und Client. Die Infrastruktur und Übertragungstechnologie sind für den fachlichen Entwickler transparent und auch austauschbar [Däschlein u.a. 2010].

2.4.5.7. **Dialog-Kompositionsmanager**

Wie auch die anderen Komponenten der Dialogkomponente, hat der Dialog-Kompositionsmanager mehrere Aufgaben. Er implementiert das Dialog-Interface des Dialograhmens und repräsentiert den Dialog gegenüber den Dialograhmen. Dabei ist die Architektur des Dialogs ein Geheimnis [Haft u.a. 2005]. Der Dialograhmen stellt die Ablaufumgebung für die Dialoge dar und bietet die grundlegende Infrastruktur zur Steuerung und Kommunikation von Dialogen [Haft u.a. 2007]. Außerdem instanziiert der Dialog-Kompositionsmager den Dialogkern und die Präsentation und konfiguriert sie mit den erforderlichen Ressourcen. Er ist darüber hinaus für das Fehlermanagement der Dialogkomponente verantwortlich. Dabei werden Fehlerbehandlungen übernommen, die nicht innerhalb der anderen Komponenten selbst erfolgen können. Sie ermöglichen eine Diagnose und Reparatur innerhalb des Dialoges oder reichen diesen an den Dialograhmen weiter. Der Dialogkompositionsmanager fungiert folglich als Sicherheitsfassade [Haft u.a. 2005].

2.4.5.8. **GUI-Frontend**

Bei dem GUI-Frontend handelt es sich um eine Komponente die Klassen aus GUI-Bibliotheken enthält. Eine GUI Bibliothek ist zum Beispiel Java Swing. Sie stellen Elemente zur Gestaltung der Benutzeroberfläche bereit [Siedersleben 2004].

2.4.5.9. **Anwendungskernproxy**

Der Anwendungsproxy verbirgt die Verteilung bei Client-Server-Anwendungen. Er hat die Aufgabe die Nutzung eines Dienstes eines Anwendungskerns auch in einem anderen Prozess eines anderen Rechners zu ermöglichen. Da er die technische Umsetzung der Client-Server-Kommunikation umsetzt, wird der Dialog technisch unabhängig [Däschlein u.a. 2010].

2.5 Architekturmuster

2.5.1 Model-View-Controller (MVC)

Model-View-Controller (MVC) beschreibt ein Architekturmuster [Veit u.a. 2003], welches das Ziel hat eine losere Kopplung zu erreichen und dadurch die Flexibilität und Wiederverwendbarkeit von Objekten zu erhöhen [Lahres u.a. 2006] [Veit u.a. 2003].
MVC wurde gemeinsam mit der objektorientieren Programmiersprache Smalltalk eingeführt. MVC ist in drei Objekte bzw. Komponenten Model, View und Controller aufgeteilt [Lahres u.a. 2006].

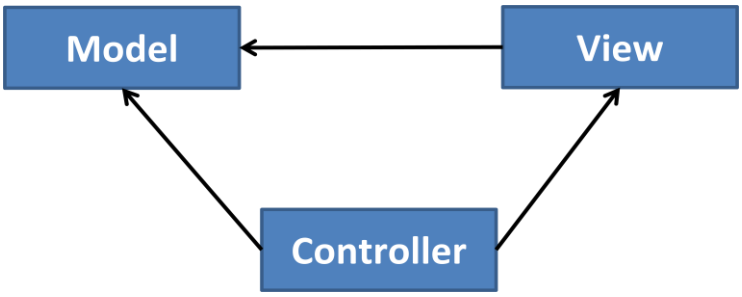

Abbildung 9 Model-View-Controller Abhängigkeiten [Microsoft 2013].

Model
Das „Model“ oder zu Deutsch „Modell“ ist für die Daten- und Zustandsverwaltung verantwortlich und unabhängig von dem Controller und der View [Lahres u.a. 2006].

View
Der „View“ oder zu Deutsch die Präsentation ist für die Darstellung des Models verantwortlich und sollte nach Möglichkeit keine fachliche Logik beinhalten. Benutzerinteraktionen werden entgegen genommen, aber nicht in der View weiterverarbeitet [Lahres u.a. 2006].

Controller
Eine Controller-Komponente oder zu Deutsch eine Steuerungskomponente nimmt die Benutzerinteraktionen entgegen und führt nach der Auswertung entsprechende Methoden aus [Lahres u.a. 2006]. Außerdem ist der Controller dafür verantwortlich das Model und/oder den View zu informieren, falls eine Aktualisierung notwendig ist [Microsoft 2013].

3 Vergleichbare Arbeiten

Dieses Kapitel befasst sich ausschließlich mit vergleichbaren Arbeiten. Es beschreibt sie hinsichtlich ihres Inhaltes, ihrer Ergebnisse und ihrem Nutzen. Außerdem wird beschrieben inwiefern sie sich von der eigenen Arbeit unterscheiden.
In Abschnitt 3.1 wird eine Arbeit beschrieben, die sich mit der komponentenbasierten Entwicklung nach Quasar Client auf Basis von Android befasst hat und in Abschnitt 3.2 wird eine beschrieben, die Ähnliches auf Basis von iOS untersucht. Der Abschnitt 3.3 beschreibt dagegen eine Arbeit, die verschiedene CPTs miteinander vergleicht.

3.1 Android und Quasar

Die Arbeit dieses Kapitels trägt den Titel „Entwurf einer komponentenbasierten Dialogarchitektur mobiler Endgeräte auf Basis von Quasar Client und Android“. Sie wurde von Heiko Maier im Rahmen seiner Masterarbeit an der Hochschule Karlsruhe Technik und Wirtschaft im November 2011 eingereicht [Maier 2011].

3.1.1 Inhalt und Ergebnisse

Maier hat einen Leitfaden zur komponentenbasierten Entwicklung nach Quasar Client auf Basis von Android entwickelt. Zu diesem Zweck wurden Quasar Client Prinzipien auf die mobile Plattform Android angewandt. Die Arbeit stellt Lösungsvorschläge für typische Probleme vor, die bei der Entwicklung mit Android auftauchen. Ein Beispiel für ein Problem wäre die Strukturierung einer Activity. Zudem wurde ein Konzept für die Umsetzung einer Dialogarchitektur beschrieben inkl. der Dialogsteuerung und der Kommunikation zwischen Dialogen. Abschließend versuchte Maier prototypisch an einer bestehenden App die Dialogarchitektur ansatzweise umzusetzen [Maier 2011].

Das Ergebnis dieser Arbeit ist, dass sich nicht alle aber viele Paradigmen der Quasar Client Architektur umsetzen lassen.

3.1.2 Abgrenzung und Nutzen

Wie auch in der eigenen Arbeit, geht es in Maiers Arbeit um die komponentenbasierte Quasar-Client-Dialogarchitektur. Sein Fokus liegt hierbei auf der Umsetzung der Dialogarchitektur bezüglich des Android Frameworks.

In der eigenen Arbeit wird das Android Framework als Vergleichsobjekt für die Untersuchungen und Bewertungen von den ausgewählten Cross-Plattform-Tools verwendet. Da Android auch in dieser Arbeit aufgegriffen wird, können Erfahrungen verglichen und ergänzt werden. Somit bietet Maiers Arbeit eine gute Teilgrundlage auf welche diese Arbeit aufsetzen kann.

Darüber hinaus beschäftigt sich diese Arbeit auch mit typischen Anforderungen für eine mobile App und bewertet die Möglichkeiten der Umsetzungen.

3.2 iOS und Quasar

Die Arbeit dieses Kapitels trägt den Titel „Konzeption und Realisierung einer komponentenbasierten Client-Architektur für mobile Anwendungen nach Quasar für die iOS-Plattform“. Sie wurde von Florian Ullrich im Rahmen seiner Bachelorarbeit an der Hochschule für Angewandte Wissenschaften Hamburg im August 2012 eingereicht [Ullrich 2012].

3.2.1 Inhalt und Ergebnisse

Die Arbeit von Ullrich beschäftigt sich wie die Arbeit von Maier und die eigene mit der komponentenbasierten Client-Architektur für mobile Anwendungen aber untersucht diese speziell für Apples iOS. Ullrich spezifizierte zu diesem Zweck eine Beispielapplikation, entwarf nach der Quasar Client Architektur eine entsprechende Architektur und untersuchte daraufhin Bestandteile der Quasar Clientarchitektur bezüglich der Umsetzbarkeit. Das Ergebnis seiner Arbeit ist eine T-Architektur für iOS-Anwendungen, welche die Komplexität von Clientanwendungen beherrschbar machen [Ullrich 2012].

3.2.2 Abgrenzung und Nutzen

Ullrichs Arbeit hat mit der eigenen Arbeit, wie bereits erwähnt, die komponentenbasierte Client-Architektur gemeinsam. Er beschäftigt sich ausschließlich mit iOS, während in der eigenen Arbeit Android und CPTs im Mittelpunkt stehen.

Diese Arbeit hat zum Lernerfolg der Quasar Client Architektur beigetragen und ermöglicht nach Abschluss der eigenen Arbeit einen Vergleich mit Android und CPTs bezüglich der Architekturumsetzbarkeit.

3.3 Cross-Plattform Frameworks

Die Arbeit dieses Kapitels ist in Englisch verfasst und trägt den Titel „Comparison and evaluation of cross-platform frameworks for the development of mobile business applications“. Sie wurde von Andreas Sommer im Rahmen seiner Masterarbeit an der Technischen Universität München im Oktober 2012 eingereicht und überschnitt sich somit auch in der Bearbeitungszeit dieser Arbeit [Sommer 2012].

3.3.1 Inhalt und Ergebnisse

Sommers Arbeit vergleicht und bewertet verschiedene CPTs miteinander anhand unterschiedlicher Kriterien wie Gerätefunktionen, Entwicklungsunterstützung und -werkzeug, Verlässlichkeit, Performanz, Features der Benutzeroberfläche, Kosten, Auslieferung, Portierbarkeit und Wartbarkeit. Zu diesem Zweck wurde eine kleine Anwendung mit ausgewählten CPTs und zwei nativen SDKs entwickelt [Sommer 2012].
Das Ergebnis von Sommers Arbeit ist, dass native SDKs immer noch in einigen Bewertungskriterien vorne liegen aber CPTs auch einen vergleichbaren Reifegrad erreicht haben [Sommer 2012].

3.3.2 Abgrenzung und Nutzen

Es stehen wie in der eigenen Masterarbeit Cross-Platform-Tools bzw. Frameworks im Mittelpunkt. Sommer verwendet und testet mehrere CPTs und zwei native SDKs. In der eigenen Arbeit werden zwar nur ein natives SDK und auch nur Teile der Bewertungskriterien herausgegriffen doch der Fokus der eigenen Arbeit liegt auch auf der Integration von Quasar in CPTs. Außerdem befasst sich die eigene Arbeit intensiver mit der Anforderungsanalyse und analysiert zudem auch die Architektur einer bestehenden größeren Applikation. Diese Arbeit kann als Vergleich für einzelne Bewertungskriterien dienen.

4 Anforderungsanalyse

Dieses Kapitel beschreibt zunächst im Unterkapitel 4.1 aus welchem Grund Anforderungen analysiert werden sollten und beschreibt daraufhin in Unterkapitel 4.2 das Vorgehen für die durchgeführten Analysen. Das abschließende Unterkapitel 4.3 dieses Kapitels beschreibt anschließend die Ergebnisse der Analysen.

4.1 Wozu Anforderungen analysieren?

Native Apps können auf alle nativen Funktionalitäten zugreifen sofern die Apps es vorsehen und die Geräte auf denen sie laufen mit der entsprechenden Hardware ausgestattet sind.
Bei der Entwicklung von Apps mit Cross-Plattform-Tools (CPTs) haben Entwickler den Wunsch annähernd so viele Funktionalitäten nutzen zu können [VisionMobile 2012].

Möchte man eine App entwickeln stellt sich die Frage, welche Funktionalitäten sie haben soll. Daraus entsteht eine Anforderungsliste, die umgesetzt werden soll. Diese Anforderung müssen an das jeweilige verwendete CPT gestellt werden. Möchte man beispielsweise eine Galerie App entwickeln, ist es zwingend erforderlich durch das CPT einen Zugriff auf das lokale Dateisystem zu erhalten.

Im Rahmen dieser Arbeit wurde eine bzw. mehrere Apps mit den gleichen Anforderungen umgesetzt. Um zielgerichtet eine App zu entwickeln, erschien es sinnvoll vorab besonders typische Anforderungen für eine mobile App zu ermitteln. Für die Ermittlung wurden eigene Untersuchungen angestellt, um diese daraufhin mit Ergebnissen aus anderen Berichten abzugleichen.

4.2 Vorgehen der Analyse

Ein gangbarer Weg stark gefragte Anforderungen zu ermitteln ist es bestehende Apps zu untersuchen. Dabei muss beachtet werden, dass je nach App Kategorie sich auch die Anforderungen unterscheiden können. Eine Wetter App könnte beispielsweise einen Zugriff auf den aktuellen Standort benötigen, um standortbezogene Wetterdaten abzurufen, wohingegen eine App die Comics anzeigt darauf verzichten kann.

Der Funktionsumfang einer App sollte sich nach Möglichkeit auf verschiedenen Betriebssystemen nicht unterscheiden. Beispielsweise sollte eine Wetter App, sowohl unter iOS als auch Android, Zugriff auf den aktuellen Standort haben um ohne manuelle Konfigurationen positionsbezogene Wetterdaten zu ermitteln. Es macht demnach keinen

Unterschied in welchem und in wie vielen App Stores Apps untersucht werden. Es wurde exemplarisch nur einer der App Stores ausgewählt, um Apps zu untersuchen. Da Android die größten Marktanteile besitzt, fiel die Wahl auf den Google Play Store.

Zur Anforderungsanalyse wurden erfolgreiche Apps herangezogen, um aussagekräftige Ergebnisse zu erhalten. Als erfolgreiche Apps wurden die Apps definiert, die eine gute Platzierung im App Store besitzen. Es wurde pro vorhandene Kategorie jeweils die bestplatzierteste App gewählt. Die Anforderungen wurden an den zusätzlich geforderten Berechtigungen der jeweiligen App festgemacht. Sofern die bestplatzierteste App keine besonderen Berechtigungen erforderte trat stellvertretend die zweitbestplatzierte App ein.
Die am häufigsten geforderten Berechtigungen ergaben sich dabei durch die Summierung der Berechtigungen aller Apps. Diese Berechtigungen wurden als Anforderungen verwendet.

Parallel zu dieser Analyse wurden Recherchen angestellt, um die Ergebnisse anderer Studien zu ermitteln. Vision Mobile hat auch dies untersucht. Es wurden Entwickler befragt welche Features sie sich wünschen [VisionMobile 2012]. Es wurde daraufhin versucht die genannten Features den Anforderungen aus der eigenen Analyse zuzuordnen.

4.3 Analyseergebnisse

Es folgen in diesem Kapitel zunächst die Ergebniszusammenfassung der eigenen Analysen und daraufhin die Ergebnisse aus einem Bericht von VisionMobile.

4.3.1 Play Store Analyseergebnisse

Die zusammengefasste Tabelle 7 zeigt die summierte Anzahl der geforderten Berechtigungen sortiert nach absteigender Häufigkeit. Der Maximalwert beträgt 24, da in 24 Kategorien jeweils eine App untersucht wurde. Der uneingeschränkte Internetzugriff erreicht als einzige Berechtigung den Maximalwert und stellt somit das Top-Kriterium dar.
Der uneingeschränkte Internetzugriff gehört zu der Berechtigungskategorie Netzkommunikation. Jede Berechtigung ist bei Android Apps genau einer Berechtigungskategorie zugeordnet.
Ebenfalls oft geforderte Berechtigungen sind der Speicherzugriff, der Zugriff des Netzwerkstatus um beispielsweise den Zugriff auf das Internet prüfen zu können, die Steuerung des Vibrationsalarms und ein Zugriff auf Standort durch verschiedene Methoden. Die ausführlichen Auswertungen sind der beiliegenden CD zu entnehmen.

Berechtigungskategorien	Berechtigungen / Features	Summe	Anforderungs-Rang
Netzkommunikation	Uneingeschränkter Internetzugriff	24	1

Speicher	Inhalt des USB-Speichers und der SD-Karte ändern/löschen	21	**2**
Netzkommunikation	Netzwerkstatus anzeigen	19	**3**
Hardware-Steuerelemente	Vibrationsalarm steuern	13	**4**
Standort	Ungefährer (netzwerkbasierter) Standort	11	**5**
Standort	Genauer (GPS-) Standort	10	**6-7**
System-Tools	Automatisch nach dem Booten starten	10	**6-7**
Ihre Konten	Bekannte Konten suchen	9	**8-9**
Telefonanrufe	Telefonstatus lesen und identifizieren	9	**8-9**
Netzkommunikation	WLAN-Status anzeigen	8	**10-11**
System-Tools	Standby-Modus des Tablets deaktivieren Standby-Modus des Telefons deaktivieren	8	**10-11**
Hardware-Steuerelemente	Bilder und Videos aufnehmen	7	**12-14**
Ihre persönlichen Daten	Kontaktdaten lesen	7	**12-14**
Netzkommunikation	Daten aus dem Internet abrufen	7	**12-14**
Hardware-Steuerelemente	Audio aufnehmen	6	**15-17**
Ihre Konten	Informationen zur Authentifizierung eines Kontos verwenden	6	**15-17**
Ihre Konten	Kontoliste verwalten	6	**15-17**
System-Tools	Aktive Apps abrufen	5	**18**

Tabelle 7 Top 18 Anforderungen aus den Play Store Analyseergebnissen basierend auf den Analysedaten vom 13.09.2012

4.3.2 Berichtergebnisse

In dem Bericht Cross Plattform Developer Tools 2012 von VisionMobile gaben befragte Entwickler an, dass fast die Hälfte der Befragten sich einen Zugriff auf das Dateisystem und das Benachrichtigungssystem wünschen [VisionMobile 2012]. Die 10 gefragtesten Features sind der folgenden Tabelle zu entnehmen. Eine klare Platzierung war den Angaben leider nicht zu entnehmen, weswegen teilweise eine Rangbandbreite angegeben ist.

Vision Mobile Rang	Anforderungen nach VisonMobile	Berechtigungs-kategorien	Berechtigungen / Features / Anforderungen
1-2	Zugriff auf das Dateisystem unabhängig vom App-Typ (Kategorie)	Speicher	Inhalt des USB-Speichers und der SD-Karte ändern/löschen
1-2	Zugriff auf das Benachrichtigungssystem unabhängig vom App-Typ (Kategorie)		
3	GPS	Standort	Genauer (GPS-) Standort
			Ungefährer (netzwerkbasierter) Standort
4	Hintergrundabläufe		
5	Multitouchscreen-Eingabe		
6-10	Native Medienwiedergabe und -aufnahme	Hardware-Steuerelemente	Bilder und Videos aufnehmen
			Audio-Einstellungen ändern
			Audio aufnehmen
6-10	SMS/MMS senden und empfangen	Kostenpflichtige Dienste	SMS senden
		Ihre Nachrichten	SMS empfangen
			SMS oder MMS lesen
			SMS oder MMS bearbeiten
6-10	Zugriff auf den Beschleunigungssensor		
6-10	Zugriff auf das Adressbuch und Anrufprotokoll (Call Log)	Ihre persönlichen Daten	Kontaktdaten lesen
			Kontaktdaten schreiben
6-10	Möglichkeit andere Apps aufzurufen		

Tabelle 8 Top 10 der am häufigsten gewünschten Features bzw. Anforderungen

Wie im Kapitel 4.3.1 erwähnt sind funktionale Anforderungen bei Android durch Berechtigungen beschrieben. Aus diesem Grund wurde weitestgehend versucht eine Zuordnung von Berichtsanforderungen zu den Android Berechtigungen vorzunehmen. Die oben aufgeführte Tabelle zeigt die Ergebnisse der Berichtsanalyse. In den ersten beiden Spalten sind die aufgestellten Anforderungen absteigend nach Wichtigkeit sortiert. In der letzten Spalte sind dagegen Berechtigungen bzw. Features aufgeführt, die dem jeweiligen Kriterium entsprechen. Die Berechtigungen können dabei Berechtigungskategorien zugeordnet werden, welche man in der dritten Spalte wiederfindet.

Der Zugriff auf das Benachrichtigungssystem, Hintergrundabläufe, Multitouchscreen-Eingabe und den Beschleunigungssensor sowie die Möglichkeit andere Apps aufzurufen sind Anforderungen, die nicht direkt auf Android Berechtigungen abgebildet werden können. Sie können nämlich ohne zusätzliche Berechtigung unter Android genutzt werden. Sie sind aber keineswegs irrelevant, da es nicht zwangsweise bedeutet dass auch CPTs sie problemlos umsetzen können.

4.3.3 Analysezusammenfassung

Die Beiden Ergebnisse bilden zusammen immer noch eine sehr umfangreiche Anforderungsliste, die hier in der finalen Zusammenfassung noch weiter auf die wichtigsten Anforderungen gekürzt wurden. Zu dieser Liste gehören die Top-10 Anforderungen der Google Play Analyse und die nicht zugeordneten Anforderungen aus den Top-10 der Vision Mobile Ergebnisse. Hinzu kamen noch die an die Top-10 angrenzenden Anforderungen mit der Rangbandbreite 12-14, die auch unter den Top-10 der VisionMobile Ergebnisse ist. Zusammen bilden diese Anforderungen die Top-18 Anforderungen dieser Arbeit.

Anforderungs-ID	Berechtigungskategorien	Berechtigungen / Features / Anforderungen	Anforderungs-Rang	VisionMobile Rang
1	Netzkommunikation	Uneingeschränkter Internetzugriff	1	
2	Speicher	Inhalt des USB-Speichers und der SD-Karte ändern/löschen	2	1-2
3	Netzkommunikation	Netzwerkstatus anzeigen	3	
4	Hardware-Steuerelemente	Vibrationsalarm steuern	4	
5	Standort	Ungefährer (netzwerkbasierter) Standort	5	3
6	Standort	Genauer (GPS-) Standort	6-7	3
7	System-Tools	Automatisch nach dem Booten starten	6-7	
8	Ihre Konten	Bekannte Konten suchen	8-9	
9	Telefonanrufe	Telefonstatus lesen und identifizieren	8-9	
10	Netzkommunikation	WLAN-Status anzeigen	10-11	
11	System-Tools	Standby-Modus des Tablets deaktivieren Standby-Modus des Telefons deaktivieren	10-11	
12	Hardware-Steuerelemente	Bilder und Videos aufnehmen	12-14	6-10
13	Ihre persönlichen Daten	Kontaktdaten lesen	12-14	6-10

Anforderungs-ID	Zusätzliche VisionMobile Anforderungen	VisionMobile Rang
14	Zugriff auf das Benachrichtigungssystem unabhängig vom App-Typ (Kategorie)	1-2
15	Hintergrundabläufe	4
16	Multitouchscreen-Eingabe	5
17	Zugriff auf den Beschleunigungssensor	6-10
18	Möglichkeit andere Apps aufzurufen	6-10

Tabelle 9 Zusammengefasste Liste der funktionalen Anforderungen

5 Cross-Plattform-Tool Auswahl

Dieses Kapitel beschäftigt sich mit der Auswahl von Cross-Plattform-Tools und beschreibt im Kapitel 5.1 die Auswahl im Allgemeinen. Anschließend wird in Kapitel 5.2 das Vorgehen und die Ergebnisse dieser Arbeit erläutert.

5.1 Cross-Plattform-Tool Auswahl im Allgemeinen

Es existieren mehr als 100 Cross-Plattform-Tools (CPTs) [VisionMobile 2012]. Dabei wurde aus dem Grundlagenkapitel klar, dass sie sich unterscheiden. Sie nutzen unterschiedliche Technologieansätze, produzieren unterschiedliche App-Typen, unterstützen unterschiedliche Plattformen und unterschiedliche Stufen des App-Lebenszyklus, sie sind unterschiedlich stark in Entwicklungsumgebungen integriert und stellen unterschiedliche Werkzeuge bereit. Sie zielen zudem auf unterschiedliche Entwicklergruppen ab und haben sonst auch noch viele weitere Differenzen.

Stellt man sich die Frage welches Tool man wählen soll, muss man sich zunächst einmal die Frage stellen was man genau entwickeln möchte und was die App können soll womit wir wieder beim Kapitel 4 der Anforderungsanalyse wären. Außerdem soll man sich über seine Fertigkeiten und über den verfügbaren Zeitraum zur Entwicklung der App im Klaren sein.
VisionMobile erstellte zur erleichterten Tool-Wahl eine mögliche Entscheidungstabelle, die hilfreich bei der Wahl sein kann.

Sie sind ein/e…	Und Sie wollen…	Dann benutzen Sie…
CIO, der Enterprise Applikationen mobil machen möchte	Bestehende C# und .NET Anwendungen wiederverwenden	Xamarin (MonoTouch / Droid), iFactr, Expanz
Endverbraucher und App Entwicklung ist Neuland für Sie	Einfach Ihre erste App erstellen	App Factories wie Spot Specific, Tiggzi, Mobile Nation HQ
Entwickler Neuling	2D Spiele erstellen	Games Salad, Corona, AppMobi
Verlagsgesellschaft	Ihre Webinhalte auf alle Haupt-Smartphone Plattformen ausweiten	App Factories oder Web-To-Native Wrappers wie beispielsweise PhoneGap, Uxebu oder Presspad
Klein bis mittelständisches Unternehmen	Eine Enterprise Applikation erstellen	Appcelerator (Titanium), Verivo, Netbiscuits, DragonRad, Expanz, Xamarin (MonoTouch)
Softwareentwickler	Ideen mit einem schnellen Entwicklungszyklus verwirklichen	RunRev LiveCode, Proto.io
Softwareentwickler	2D Spiele schreiben	Marmalade, SiO2, EDGELIB, Cocos2D
Erfahrener Spieleentwickler	3D Spiele erstellen	Unity 3D, Unreal, Marmalade

Designer einer digitalen Agentur	2D Spiele	Gamesalad, Cocos2D, AppMobi, ImpactJS, LimeJS
Designer einer digitalen Agentur	Rich Multimedia Erlebnis deployed über einen App Store	Adobe AIR, Corona, Appcelerator

Tabelle 10 Tool-Wahltabelle nach VisionMobile [VisionMobile 2012]

Möchte beispielsweise eine Verlagsgesellschaft Ist seine Webinhalte auf allen Haupt-Smartphone Plattformen ausweiten, sollte man beispielsweise App Factories nutzen oder Web-To-Nativ Wrapper, wie PhoneGap. Möchte man dagegen 2D Spiele Entwickeln kann man der Tabelle entnehmen, welche Tools nach Erfahrungsgrad die möglich richtigen sind.

5.2 Cross-Plattform-Tool Auswahl in dieser Arbeit

In dieser Arbeit wurden die CPTs nicht nach der Umsetzbarkeit der funktionalen Anforderungen gewählt, die im vorangegangen Kapitel 4 beschrieben wurden. Ausschlaggebend für die Wahl der CPTs war insbesondere die Verbreitung der Tools. Aus diesem Grund werden im Folgenden Unterkapitel zunächst Analyseergebnisse bzgl. der Nutzung vorgestellt.

5.2.1 Nutzungsanalysen

Es wurden in einer Untersuchung 2.406 Entwickler aus 91 Ländern befragt welche CPTs sie nutzen. Dabei gaben knapp ein Drittel der Befragten an PhoneGap zu nutzen. Damit ist PhoneGap das am meistgenutzte CPT. Sencha Touch bzw. jQTouch belegen mit 30% den zweiten Platz. Xamarins „MonoTouch“ für die iOS-Entwicklung bzw. „Mono for Android“ für die Entwicklung mit Android, werden mit 26% von etwas mehr als ein Viertel der Befragten benutzt. Appcelerator und Adobe Flex belegen Platz vier und fünf mit 24% und 22% [VisionMobile 2012].

Neben den Befragungen welche Tools die befragten Entwickler nutzen, wurden sie zudem befragt welche Tools sie planen einzusetzen und welche sie verwerfen wollen. 23% wollen PhoneGap in Zukunft nutzen. Fast genauso viele planen Mono und Unity einzusetzen. Appcelerator und Sencha Touch sind ebenfalls unter den Top 5. Wenn es um die Verwerfung von CPTs geht, ist Adobe Flex auf Platz eins mit 42%.

PhoneGap und Appcelerator werden häufig verwendet, weil sie kostenlos zur Verfügung stehen. Sie werden allerdings auch nicht selten verworfen, wie man den Umfragen entnehmen kann. Zwei der Hauptverwerfungsgründe sind Schwierigkeiten beim Debugging und Performanceschwächen. Corona und Mono belegen jeweils mit 12% gemeinsam den fünften Platz bei den geplanten Verwerfungen [VisionMobile 2012].

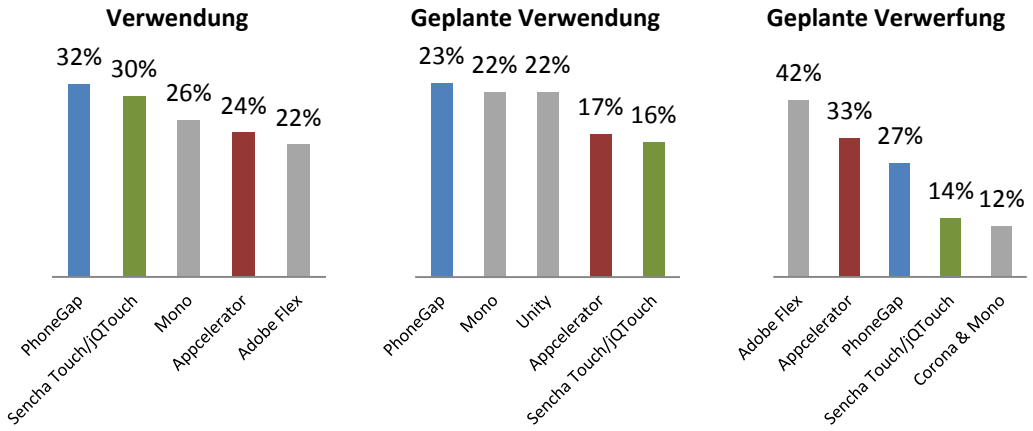

Abbildung 10 Die Top 5 der am häufigsten genannten CPTs bzgl. der Verwendung, geplanten Verwendung und der geplanten Verwerfung [VisionMobile 2012]

5.2.2 Gewählte Tools

Das Ziel war es, für diese Arbeit zwei CPTs auszuwählen. Die CPTs sollten sich dabei in den produzierten App-Typen und ihrer verwendeten Technologie unterscheiden. Da PhoneGap in den Umfragen sowohl in der Verwendung als auch der geplanten Verwendung weit vorne liegt, wurde entschieden PhoneGap oder ein darauf aufbauendes CPT zu wählen. PhoneGap produziert hybride Web Apps (Kapitel 2.1.3)und nutzt dazu die Web-To-Native Wrapper Technologie (Kapitel 2.3.2.5). Sencha Touch lag in den Umfragen hinter PhoneGap, nutzt u.a. den gleichen Technologieansatz und produziert den gleichen App-Typ. PhoneGap wurde demnach Sencha Touch vorgezogen.

Appcelerator ist ebenfalls in der Verwendung und der geplanten Verwendung unter den Top 5. Dieses CPT nutzt im Gegensatz zu PhoneGap den Runtime-Technologieansatz und produziert native Apps. Die Wahl für das zweite Tool fiel aus diesem Grund auf Appcelerators Titanium.

Xamarins „Mono“-Lösungen produzieren ebenfalls native Apps und nutzen ebenfalls die Runtime-Technologie und zusätzlich noch die Sourcecode Übersetzer Technologie und liegen in den Umfragen auch vor Appcelerator. Da dieses CPT aber nicht kostenlos ist, wurde darauf verzichtet, sich näher damit zu beschäftigen. Auf Adobe Flex dagegen wurde verzichtet wegen der hohen Werte bzgl. der CPT Verwerfung und der etwas schlechteren Positionierung gegenüber Appcelerator.
Unity ist ein CPT, welches für Spiele ausgelegt ist. In den Abgrenzungen wurde darauf hingewiesen, dass CPTs zur Spieleentwicklung einen eigenen Bereich darstellen, der nicht näher betrachtet wird.

In den folgenden Kapiteln werden Tools, sofern sie verwendet wurden, in Kürze beschrieben. Zu den CPTs, die getestet und verworfen wurden, folgen kurze Beschreibungen zu verschiedenen Kriterien und eine abschließende Begründung der Verwerfung.

5.2.2.1. **IBM Worklight**

Für diese Arbeit hat der Autor beschlossen PhoneGap oder ein auf PhoneGap basierendes Tool zu verwenden. Ein Tool, welches sehr interessant erschien war IBM Worklight. Es setzt auf PhoneGap auf und bietet zusätzlich eine IDE und die Möglichkeit Apps jeden Typs zu entwickeln. Nach einiger Verwendungszeit wurde dieses Tool jedoch verworfen.

Installation
Die aktuelle Version ist zurzeit nicht mit der aktuellen Eclipse Juno Version kompatibel, weswegen man darauf achten muss eine ältere Version (z.B. Indigo) von Eclipse zu verwenden. Die Installation erfolgt durch ein Plug-In, das direkt in Eclipse über eine URL installiert werden kann.

Integration in die IDE
Es ist möglich direkt Worklight Projekte zu erstellen und dabei zwischen nativen, hybriden oder nativen Apps auswählen. Zusätzliche Buttons in der Toolbar ermöglichen es mit drei Klicks zusätzliche Plattformen bzw. wie sie hier genannt werden „Environments“ (Android, IOS usw.) hinzufügen. Durch das Hinzufügen weiterer Environments wird jeweils ein Unterordner erzeugt. Der Unterordner hat dabei immer eine Struktur wie der von allen Environments geteilten Gemeinschaftsordner. Diese Unterordner für die Environments ermöglichen eine individuelle Anpassung an die Plattformen, so dass die Inhalte sich plattformabhängig unterscheiden können. Außerdem ist es möglich nativen Code dort zu schreiben, falls man mit den von Haus gelieferten Worklight- und PhoneGap-Bibliotheken an seine Grenzen stoßen sollte. Somit werden aus hybriden Web Apps, hybride Mix Apps.

Deploying
Möchte man die App auf einem Gerät deployen ist vorab ein Build notwendig, welcher ein neues spezifisch plattformangepasstes Projekt erzeugt. Der Vorteil daran ist, dass der Buildprozess vollautomatisiert geschieht, solange man nichts an den Ordnerstrukturen verändert. Es wurde versucht eine andere Projektstruktur zu integrieren. Dazu wurde auch eine entsprechende XML-Datei angepasst. Bei dem Deployen traten aber dann schwer nachvollziehbare Probleme auf. Es wird vermutet, dass die Projektstruktur nicht so flexibel ist. Unterstützend für diese Vermutung ist, dass auch bei einer Änderung der Projektstuktur des gemeinsamen Ordners, sich neu eingerichtete Environments sich nicht an der neuen Struktur orientieren.

Einarbeitung
Worklight bietet zurzeit nach eigenem Befinden verhältnismäßig wenige Anlaufstellen um zu lernen, wie man damit effektiv entwickelt. Es gibt zwar eine zentrale Anlaufstelle, wo das meiste in einzelnen PDF Dateien beschrieben ist, aber wenn man mal Probleme hat findet man im Vergleich zu Sencha Touch und Appcelerator eine bemerkbar kleinere Community.
Was auch anfangs etwas hinderlich war, sind das BasisProjekte aus den Tutorials nicht ohne weiteres lauffähig waren. Umgehen konnte man dieses Problem indem man ein neues Projekt aufsetzte und die Source-Dateien einfach in das neue Projekt kopierte.

Bevor man die App auf einem echten Gerät testen kann, muss man sich entweder erst einmal mit dem Worklight Server beschäftigen, der standardmäßig von der App automatisch aufgerufen wird oder die Verbindung zum Server deaktivieren. Die Verbindung zum Worklight-Server funktioniert auf einem getesteten Android-Virtual-Device problemlos aber auf dem nativen Gerät gab es bei mir immer Fehlermeldungen mit denen man sich anfangs ungern beschäftigt, da dies beim Lernen irritierend und eher hinderlich ist.

Was positiv auffiel ist die bereits erwähnte gut funktionierende automatische Generierung von Environments und der entsprechenden Projekte, wenn man sich an die vorgegebene Struktur hält.

Implementierung
Bei der Implementierung fiel nichts Negatives auf. Allerdings wurden mit IBM Worklight auch nur die ersten Schritte gegangen. Es gilt hierbei vermutlich alles was für Webprojekte sonst auch gilt.

Verwerfungsbegründung
Da die Unterstützung bzgl. Worklight im Web nicht so stark gegeben ist und auch Beispielprojekte nicht auf Anhieb liefen, war dies ein mögliches Anzeichen dafür, dass die Dokumentationen nicht besonders aktuell sind und dass beim Auftreten von Problemen man nur schwer weiterkommt. Probleme schwer lösen zu können ist allein schon ein guter Grund sich gegen Worklight zu entscheiden. Zudem ist Worklight nicht mit den neusten Entwicklungsumgebungen kompatibel und bringt Features mit sich, welche nicht zwangsweise für die Zwecke dieser Arbeit benötigt werden, wie beispielsweise den Worklight Server. Bevor man mit der Implementierung beginnen kann, ist es notwendig sich mit Worklight Strukturen wie dem genannten Server zu beschäftigen. Dies kostet zu Beginn unnötig Zeit.
Zusammengefasst lässt sich sagen, dass Worklight sehr vieles bietet, aber für die Zwecke dieser Arbeit darin ein zu hohes Risiko gesehen wurde es einzusetzen. Aus diesem Grund fiel die Wahl doch direkt auf PhoneGap.

5.2.2.2. **PhoneGap & Sencha Touch**

Nach dem Verzicht auf Worklight, wurde PhoneGap an sich für den Einsatz ausgewählt. PhoneGap wird häufig in anderen CPTs als Basis genutzt wie auch in Worklight. Daher erschien PhoneGap als die richtige Wahl.

Sencha Touch lag ebenfalls in den Umfragen unter den Top-5 und unterlag bei der Vorauswahl PhoneGap. PhoneGap verfügt jedoch über kein eigenes User Interface (UI) Framework. Aus diesem Grund wurde beschlossen PhoneGap in Kombination mit Sencha Touch zu nutzen.

5.2.2.3. **Appcelerator (Titanium)**

Titanium ist ein ebenfalls stark verbreitetes CPT, welches nativen Code erzeugt und zudem auch noch kostenlos ist. Eine bekannte Applikation, die mit Titanium entwickelt wurde ist Wunderlist von der 6Wunderkinder GmbH. Bei Wunderlist handelt es sich um eine Todo-Listen App [6Wunderkinder 2013]. 6Wunderkinder verwarfen Titanium nach Medienberichten mit der neuen Version Wunderlist 2 aus verschiedenen Gründen [Golem.de 2013].

Es ist besonders interessant sich mit Titanium zu befassen, da trotz der hohen Verwerfungszahlen immer noch neue Nutzer planen Titanium einzusetzen.

6 Analyse der Integration der Quasar Client Dialogkomponenten-Architektur

Dieses Kapitel beschreibt zunächst in Abschnitt 6.1 die Vorgehensweise der Analyse und nimmt dann in Abschnitt 6.2 direkten Bezug auf die Integration der Quasar Dialogkomponenten-Architektur in Android. Anschließend wird in Abschnitt 6.3 das MVC Architekturmuster aus Kapitel 2.5.1 aufgegriffen und beschrieben wie dieses als Basis zur Integration der Dialogkomponenten-Architektur dienen kann. Der Abschnitt 6.4 analysiert die Integration in PhoneGap & Sencha Touch und der Abschnitt 0 beschreibt selbiges in Bezug auf Titanium.

6.1 Art der Analyse

Diese Arbeit befasst sich mit der Integration von Standardarchitekturen am Beispiel von Quasar. Der Begriff Quasar deckt ein sehr umfassendes Spektrum ab, welches im Rahmen dieser Arbeit nicht vollständig behandelbar ist. Aus diesem Grund konzentriert sich diese Arbeit auf die Dialogkomponentenarchitektur des Quasar Clients. Es geht insbesondere um die Aufteilung der Präsentation und des Dialogkerns, welche im folgenden Ausschnitt noch einmal aufgegriffen werden.

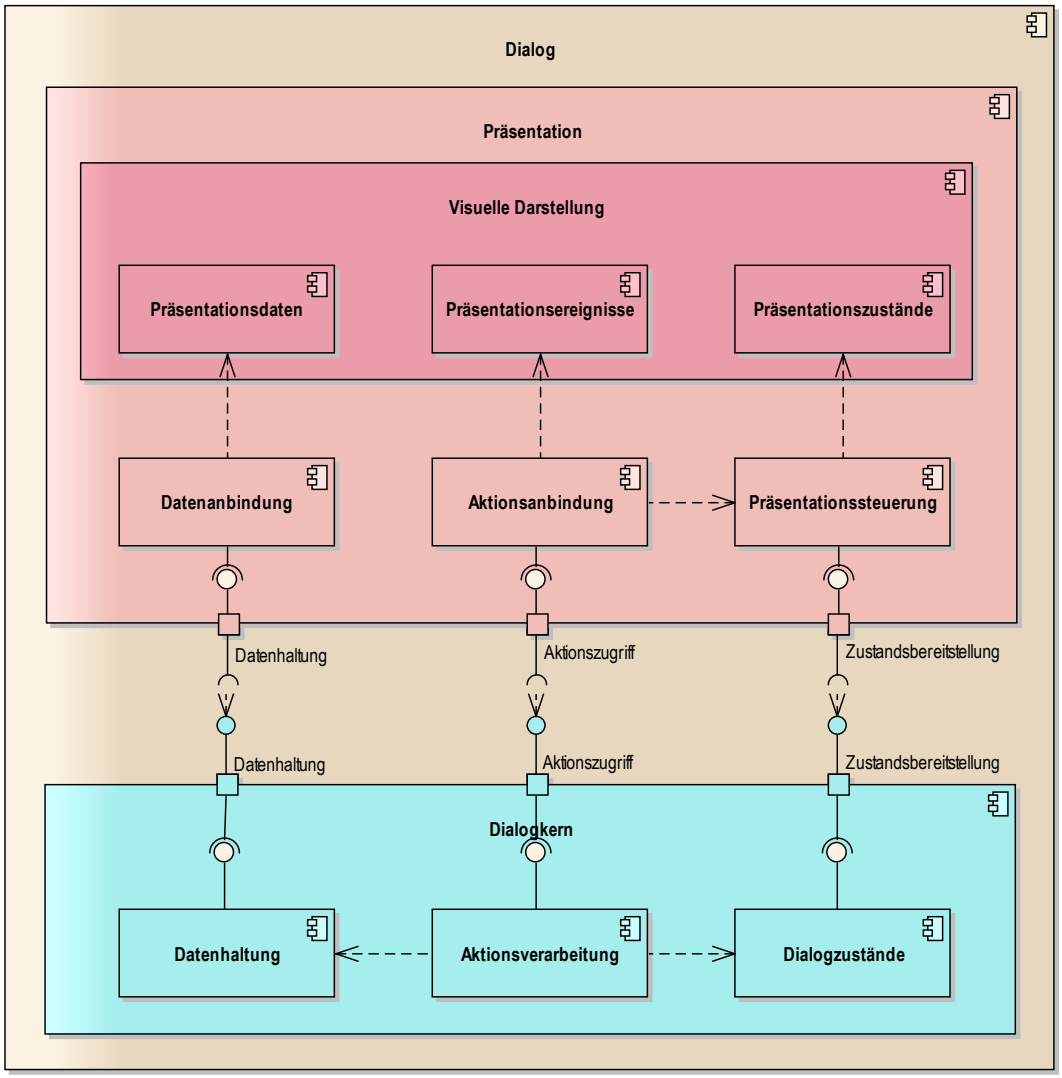

Abbildung 11 Struktur einer Quasar Dialogkomponente [Haft u.a. 2012]

Die Präsentation ist für die Bereitstellung der visuellen Darstellung der Oberfläche verantwortlich, wohingegen der Dialogkern für die Datenhaltung, Aktionsverarbeitung und die Verwaltung der Dialogzustände verantwortlich ist. Über Schnittstellen kann die Präsentation auf Dienste des Dialogkerns zugreifen. In der folgenden Abbildung 12 ist die Dialogkomponente in vereinfachter Form dargestellt.

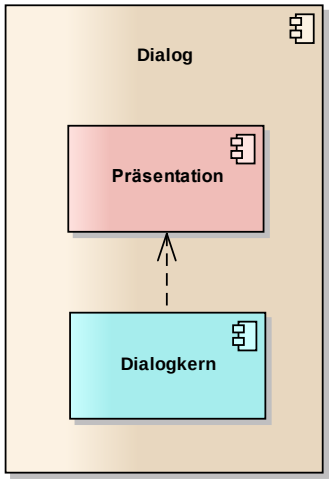

Abbildung 12 Vereinfachte Darstellung einer Quasar-Dialogkomponente

Es wird in den folgenden Kapiteln beschrieben, ob und wie die Dialogkomponente mit den ausgewählten CPTs umgesetzt werden kann. Um eine native Entwicklung als Vergleich zu haben wurde Android herangezogen. Es wird mit Android in Kapitel 6.2 begonnen und mit PhoneGap & Sencha Touch in Kapitel 6.4 und Titanium mit den Alloy-Framework in Kapitel 0 fortgefahren.

6.2 Android

Android ist hinsichtlich der Verbreitung die führende Plattform. Aus diesem Grund wurde Sie als Referenzplattform gewählt. Das aktuelle Betriebssystem vom März 2013 ist Android 4.2.2. alias „Jelly Bean“.

Kurz gefasst lässt sich sagen, dass Android Apps aus einem Set von lose gekoppelten Screens bestehen, die durch Activity Klassen repräsentiert werden und durch Service Klassen, die Hintergrundprozesse ausführen [Xamarin 2013].

6.2.1 Quasar und Android

Nach den Quasar-Begrifflichkeiten entspricht eine Activity einem Dialog, da eine Activity eine oder mehrere Anwendungsfälle für den Anwender bereitstellt [Maier 2011]. Für Activities bestehen keine Vorgaben zur Einhaltung des Prinzips Separation of Concerns wie es bei Dialogen nach Quasar der Fall ist. Umfangreiche Dialoge können dazu führen, dass

Activity-Klassen sehr groß und schwer überschaubar werden, wodurch auch die Wartung schwieriger wird [Maier 2011].

In Android sind Dialogkern und Präsentation größtenteils miteinander verschmolzen, was auch Heiko Maier in seinen Ergebnissen bestätigt. Nur die visuelle Darstellung, des essentiellen Teils der Präsentation, wird separat angeboten. Weitere Aufgaben der Präsentation, wie beispielsweise die Datenanbindung oder Konvertierung sind nicht vorhanden. Ein Teil des Dialogkerns befindet sich in der Activity und andere fehlen. Der Dialogkern ist damit unvollständig was in Kapitel 6.4.2 detaillierter beschrieben wird [Maier 2011].

Wie bereits beschrieben entspricht ein Dialog einer Activity und Präsentation und Dialogkern sind komplett miteinander verschmolzen [Maier 2011]. Man könnte es allerdings etwas präzisieren und es so sehen, dass ein Teil der Präsentation einem XML Layout entspricht, die in Android partiell für die visuelle Darstellung verantwortlich ist. Der andere technische Teil der Präsentation ist in der Activity mit der Fachlichkeit des Dialogkerns vermischt. Es wird beispielsweise aus der Activity direkt auf die Präsentationschicht zugegriffen und Inhalte gesetzt, was eigentlich auf Präsentationsschichtebene geschehen sollte. Zusammen mit diesen technischen Aufgaben findet beispielsweise auch die fachliche Aktionsverarbeitung statt, die durch Useraktionen ausgelöst wurden.

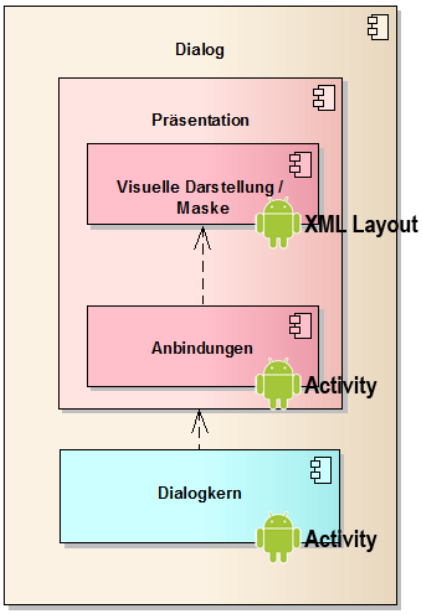

Abbildung 13 Abbildung der Android Bestandteile auf die Quasar-Client-Dialogarchitektur

6.2.2 Realisierungsmöglichkeit einer Dialogkomponente

6.2.2.1. **Umfang einer Dialogkomponente**

Eine Dialogkomponente besteht in der Umsetzung aus mehreren Elementen. Hier soll erklärt werden, was alles zu einer Dialogkomponente in Android gehört.
Mit dem Android Framework ist für jede visuelle Darstellung eine Activity vorgesehen. Dies wird in diesem Konzept beibehalten. Daher enthält jeder Dialog eine Activity. Da zur Darstellung einer Maske meistens mehre Teile nötig sind, wird hier beschrieben welche Teile dazu zählen [Maier 2011].

Es zählen folgende Teile zu einer Dialogkomponente[Maier 2011]:

- Activity Klasse und der zugehörige Controller, der in dem Konzept von Maier beschrieben wird
- Fragmente
- Klassen zur Darstellung von User Interface Elementen
- Benötigte Ressourcen Dateien wie
    - Layouts
    - Bilder
    - Menus
    - Werte wie Strings, Farben, Integers usw.
    - Animationen
    - Farbzustände
- Klassen zur Ausführung eines Anwendungskernzugriffs
- Eindeutig dem Dialog zuordnungsbare Klassen

Teile, die in mehreren Dialogen verwendet werden und somit nicht eindeutig einer Dialogkomponente zugeordnet werden können, kann man auf zwei Weisen behandeln. Man kann einerseits den Teil duplizieren und jeder Dialogkomponente somit eine eigene Kopie zuordnen oder auch einfach keine Zuordnung vornehmen. Im ersten Fall würden Duplikate zu einer sauberen Trennung der Dialoge führen, was aber auch zusätzlichen Speicher erfordern würde. Der zweite Fall wäre vorzuziehen, wenn das entsprechende Teil sehr häufig verwendet wird [Maier 2011].

6.2.2.2. **Initiale Projektstruktur**

Eine Projektstruktur, welche die Dialogkomponentenarchitektur wiederspiegelt, verschafft einem unter anderem eine leichtere Orientierung.

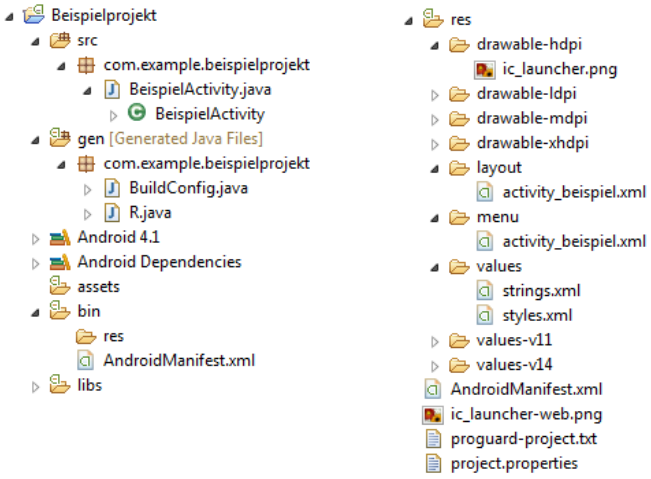

Abbildung 14 Initiale Android 4.1. Projektstruktur

Da Android nicht nach der Quasar Client Architektur umgesetzt wurde, ist es nachvollziehbar, dass sich diese Struktur nicht in der initialen Projektstruktur von Android wiederspiegelt.

6.2.2.3. **Umstrukturierung der Projektstruktur**

Java Dateien

Um Java-Dateien zu organisieren und übersichtlich nach Dialogen zu unterteilen, bieten sich Pakete an. Dies ist eine naheliegende und auch in vergleichbaren Arbeiten angewandte Möglichkeit.

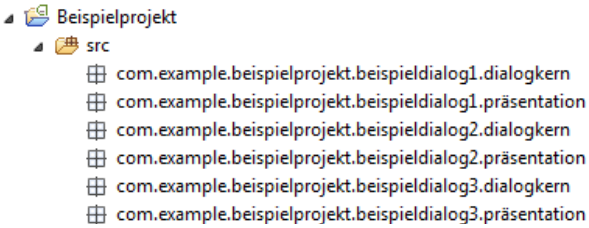

Abbildung 15 Organisation von Java Dateien nach Dialogkomponenten

Es wird für jede Dialogkomponente ein Unterpaket erzeugt. Sofern man eine Trennung des Dialogkerns und der Präsentation vornimmt, sollte das erstellte Paket wiederum in die Unterpakete Dialogkern und Präsentation untergliedert werden. Dies bedeutet, dass hier eine zuerst fachliche und danach technische Aufteilung vorzunehmen ist.

Ressourcen Dateien

Es wird im Folgenden auf die Dateien der Ressourcen Typen Layouts, Bilder, Werte und Menus näher eingegangen. Ressourcen Dateien werden in dem jeweiligen Unterordner des Ressourcenordners „res“ abgelegt und liegen somit getrennt von den Java-Dateien. Um eine direkte Zuordnung sichtbar zu machen, ist es generell sinnvoll die Dateien mit dem Namen des Dialoges zu versehen. Falls mehrere Dateien notwendig sind für einen Dialog und somit der Dialogname nicht ausreicht, ist der Dialogname als Präfix zu nutzen. Dies ist am Beispiel der Bilder des Beispieldialogs1 zu sehen.

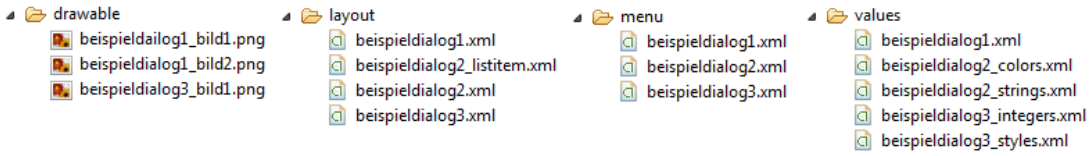

Abbildung 16 Ressourcen von links nach rechts Bilder, Layouts, Menus, Werte

Meistens hat jeder Dialog ein Hauptlayout. Es ist aber auch möglich, dass mehrere Layouts benötigt werden, wie beispielsweise bei dem Verwenden von Android Fragmenten. Ein weiteres Beispiel wäre die Erstellung eines speziellen ListItem-Layouts, welches innerhalb einer ListView angezeigt wird und somit ein untergeordneter Bestandteil der ListView ist [Maier 2011].

Es kann sehr viele Werte in einem Dialog geben. Dabei existieren verschiedene Werte. Einige dieser Beispiele sind in der Grafik zu sehen. Beispieldialog1 hat zum Beispiel nur eine XML Datei. Dies bietet sich an, wenn sehr wenige Werte benötigt werden. Werden dagegen mehrere Werte benötigt, wie bei den Beispieldialogen 2 und 3, wäre es sinnvoll sie nach Typen mit vorangestelltem Dialognamen aufzutrennen.

6.2.3 Dialogaufteilung

Dieses Unterkapitel beschreibt, ob und wenn ja wie Präsentation und Dialogkern eines Dialoges in Android getrennt werden kann. Dabei basieren die Informationen der folgenden beiden Unterkapitel, Dialogkern und Präsentation, hauptsächlich auf den Ergebnissen von Heiko Maiers Abschlussarbeit.

Zur Trennung in die Sub-Komponenten Dialogkern und Präsentation werden alle Klassen den zugehörigen Paketen zugeordnet. Sehr einfache Dialoge müssen dabei nicht in Subkomponenten aufgeteilt werden [Maier 2011].

Eine Acitvity ist der zentrale Teil jedes Dialoges. Nach Maier ist bei einer Trennung eines Dialoges, analog zu der Quasar Client Architektur, eine Aufspaltung nötig. In seiner Ausarbeitung hat er zu diesem Zweck einige Klassen entwickelt, welche einen Controller

des Dialogkerns für jede Activity erzeugt [Maier 2011]. Unterstützend verwendet er Android Binding. Android Binding ist ein Open Source Projekt, welches die Aufteilung einer Activity nach dem MVC oder MVVM Entwurfsmuster erleichtert [Android-Binding 2013].

Mit seinem Ansatz wird unter anderem eine räumliche Trennung vorgenommen. Es sei angemerkt, dass es nach Quasar Client nicht zwingend notwendig ist eine räumliche Trennung vorzunehmen. Trennt man die Fachlichkeit von der Technik innerhalb einer Activity, würde dies den Quasar Client Gedanken nicht verletzen. Es muss aber so geschehen, dass eindeutig erkennbar ist, was zum Dialogkern und was zur Präsentation gehören würde. Eine Möglichkeit wäre es beispielsweise ebenfalls mit den Präfixen „Kern“ und „Präsentation“ zu arbeiten um die Bestandteile innerhalb einer Activity sofort visuell zuordnen zu können. Da dies jedoch etwas unübersichtlich werden kann, wäre auch die räumliche Trennung vorteilhafter.

6.2.3.1. **Dialogkern**

Alle Klassen, die Aufgaben für die Datenhaltung, Aktionsverarbeitung oder der Dialogzustände übernehmen, sollten in das Paket des Dialogkerns. Hinzukommen würde der entsprechende Controller aus Maiers Ansatz. Wie der Controller im Detail umgesetzt wurde, kann man seiner Arbeit entnehmen.

Datenhaltung
Daten können in Android auf verschiedene Weisen abgelegt werden. Primitive Datentypen können in Android über SharedPreferences als Key-Value-Paare abgelegt werden. Außerdem sind Daten sowohl auf dem internen Speicher des Gerätes, als auch auf einem externen Speichermedium ablegbar.

Für größere Datenmengen kann eine SQL-Datenbank verwendet werden, die dann jedoch nicht mehr Teil des Dialogkerns wäre, sondern Teil des Anwendungskerns. Android unterstützt hierfür SQLite. Daten können allerdings auch extern auf einem Server gespeichert werden [Developer Android 2013].

Aktionsverarbeitung
Aktionen werden durch eine Methode im Controller verarbeitet. Aktionen können Aufrufe zum Anwendungskern oder aber auch zum Beispiel der Aufruf eines anderen Dialoges sein.

Dialogzustandsverwaltung
Eine Zustandsverwaltung ist nicht direkt in Android integriert. Sie lässt sich aber durch Adapter-Klassen in das Android Binding Framework integrieren.

6.2.3.2. **Präsentation**

In das Präsentationspaket gehört die eigentliche Activity in der auch der entsprechende Controller von Maiers Ansatz erzeugt wird. In der Activity findet die Datenanbindung, Aktionsanbindung und die Präsentationszustandssteuerung statt. Zusätzliche Klassen, die Funktionalitäten für die Oberfläche bereitstellen, gehören ebenfalls in dieses Paket. Layouts setzen meist die visuelle Darstellung um, können aber, falls nötig, auch in der Activity umgesetzt werden.

Datenanbindung
Android bietet nach Heiko Maier keine Datenanbindung an. Dies hat zur Folge, dass alle Werte manuell aktualisiert werden müssen. Das bereits erwähnte Android Binding bietet aber als Alternative eine bessere Lösung an.

Aktionsanbindung
Aktionen werden als Command über Android Binding angebunden. Sind einzelne Aktionsanbindungen noch nicht direkt unterstützt durch Android, kann auf die Standard Android Lösung (beispielsweise durch die Erstellung eines OnClickListeners und anschließender Registrierung am Element) zurückgegriffen werden.

Präsentationszustandssteuerung
Auch die Präsentationszustandssteuerung wird durch das Android Binding Framework umgesetzt. Benutzerereignisse, wie die Auswahl einer Checkbox, werden über Variablen des Android Bindings umgesetzt. Zustände können aber auch umgekehrt auf die Anzeige wirken. So kann eine Deaktivierung eines Buttons genau so über die Variablen dargestellt werden.

6.3 Quasar Client Dialogarchitektur und MVC

Ein Entwurfsmuster, welches dabei helfen kann Ansätze des Quasar Clients umzusetzen, ist das Model-View-Controller (MVC) Architekturmuster. Es lässt sich ohne größere Anpassungen partiell auf die Quasar-Client-Dialogarchitektur abbilden, wie in Abbildung 17 und Abbildung 18 zu sehen ist.

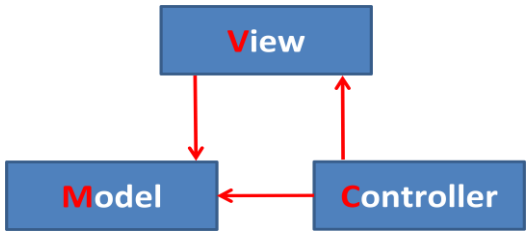

Abbildung 17 Model-View-Controller Abhängigkeiten

Die partielle Projektion auf die Quasar Client Dialogarchitektur weist allerdings keine klare Trennung zwischen der Maske und den Anbindungen auf. Dennoch unterstützt die Verwendung dieses Architekturmusters die Integration von Quasar Client in Dialogarchitekturen. Deshalb wurde dieser Ansatz als Grundlage für die genutzten CPTs ausgewählt und verwendet.

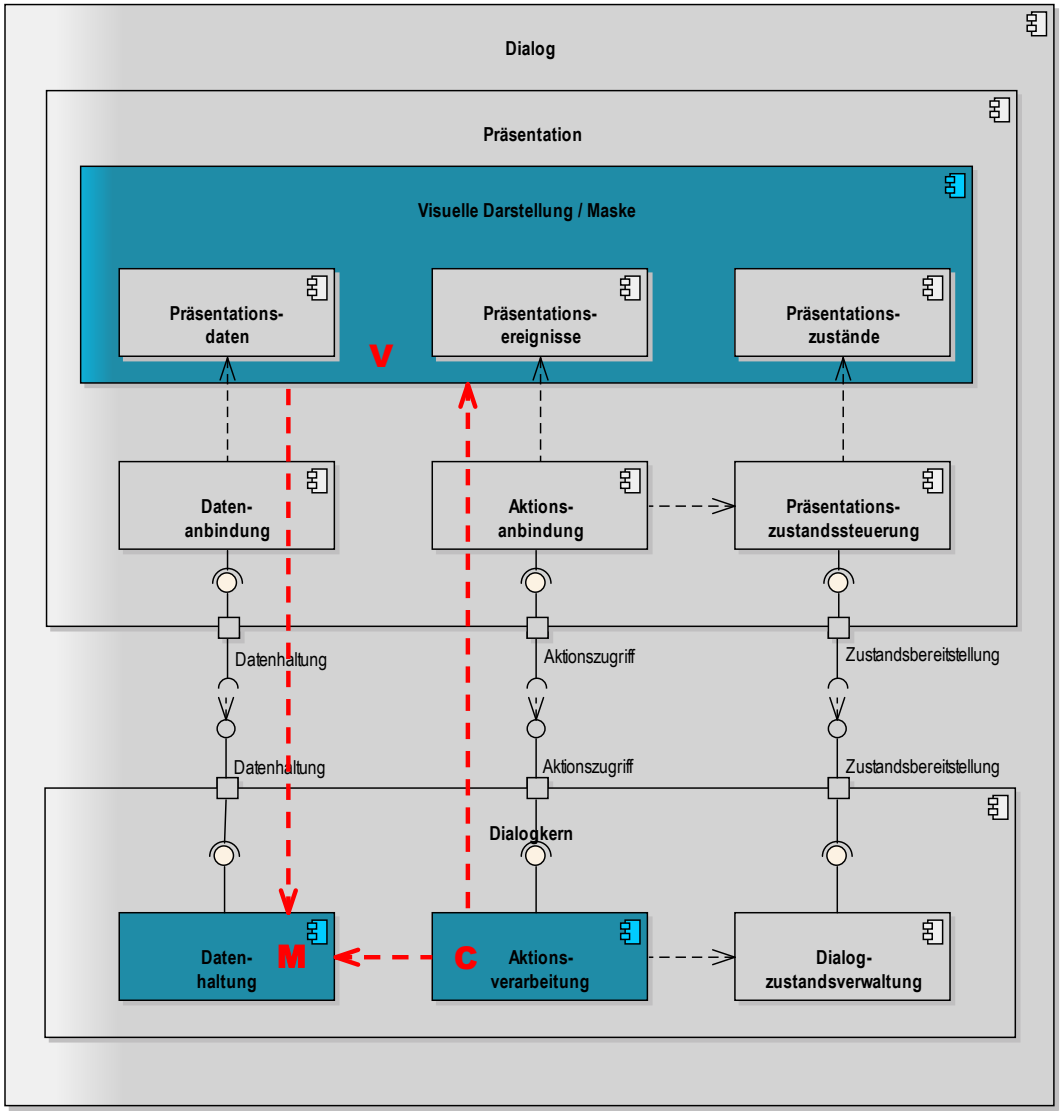

Abbildung 18 Projektion des MVC Architekturmusters auf die Dialogarchitektur des Quasar Clients [Haft u.a. 2012]

6.4 PhoneGap & Sencha Touch

Die aktuelle Version im Februar 2012 von PhoneGap bzw. Apache Cordova ist die Version 2.4.0 [CordovaDoc 2013]. Sencha Touch liegt aktuell bei der Version 2.1.1 [SenchaDoc 2013]. Im Rahmen dieser Arbeit wurde mit Cordova 2.2.0 gearbeitet, die zu Beginn dieser Arbeit am aktuellsten war. Diese beiden Tools wurden in Kombination exemplarisch in einem Android Projekt eingesetzt.

In Kapitel 6.4.1 wird zunächst die Realisierungsmöglichkeit einer Dialogkomponente beschrieben. Dazu wird beschrieben was in Bezug auf Sencha Touch-Framework zu einer Dialogkomponente gehört und wie die Bestandteile auf ein Projekt abgebildet werden können, so dass die Dialogkomponenten-Architektur sich auch visuell in der Projektstruktur wiederspiegelt. Das Kapitel 6.4.2 beschäftigt sich dagegen mit ausgewählten Subkomponenten der Dialogarchitektur und beschreibt wie und ob diese mit Sencha Touch umgesetzt werden können.

6.4.1 Realisierungsmöglichkeiten einer Dialogkomponente

Bei der Entwicklung von Apps mit Sencha Touch verwendet man ausschließlich JavaScript. Entwickler können selber bestimmen wie ihre Architektur aussehen soll. In den Sencha Touch Dokumenten gibt es allerdings Empfehlungen, wie eine App strukturiert werden kann [SenchaDoc 2013].

Im vorangegangenen Kapitel wurde gezeigt, dass die Nutzung des MVC Architekturmusters ein gangbarer Weg ist um sich an die Dialogarchitektur nach dem Quasar Client anzunähern. In den genannten Empfehlungen wird ebenfalls MVC bei der Entwicklung von Sencha Touch App empfohlen [SenchaDoc 2013].

6.4.1.1. **Umfang einer Dialogkomponente**

Bei der Entwicklung mit PhoneGap in Kombination mit Sencha Touch können folgende Elemente zu einer Dialogkomponente gehören:

- JavaScript Dateien [SenchaDoc 2013]
  - **Model:** Repräsentiert ein Datenobjekt
  - **Store:** Verantwortlich für das Laden von App-Daten und die Versorgung von Komponenten mit Daten wie z.B. Listen, die Daten des Stores anzeigen.
  - **View:** Verantwortlich für die Darstellung von Daten

    - **Controller:** Behandelt Interaktionen, durch das Lauschen auf Benutzerinteraktionen mit entsprechender Reaktion
    - **Profile:** Ermöglicht, falls erforderlich, Models, Views und Controller individuell an unterschiedliche Gerätegrößen anzupassen.
- Benötigte Ressourcen
    - Bilder
    - Stylesheets (falls benötigt)
    - Zusätzliche andere Dateien, die vom Dialog verwendet werden

Für einen Dialog sind nicht alle Elemente notwendig. Individuelle CSS-Stylesheets werden beispielsweise für einen Dialog nur bedingt benötigt. Durch eine vordefinierte app.scss, die zu einer app.css kompiliert wird, ist für die gesamte App bereits das Standard Sencha Touch 2 Theme vordefiniert. Daher ist es nicht nötig zusätzliche Stylesheets zu nutzen.

6.4.1.2. **Initiale Projektstruktur**

Die initiale Projektstruktur eines Sencha Touch & PhoneGap Projektes baut in diesem Beispiel auf der Rahmen-Projektstruktur eines Android Projektes auf, da diese Arbeit exemplarisch auf Android ausgerichtet ist. Die wichtigen Dateien, die eigentlich PhoneGap & Sencha Touch ausmachen, befinden sich dabei in dem assets Ordner des Android Projektes. In diesem erstellt man manuell die App Struktur. Es können allerdings auch Kommandozeilenbefehle des Sencha Cmd verwendet werden, um eine initiale Struktur zu generieren.

In Abbildung 19 ist die vorgeschlagene Projektstruktur zu sehen, die bei der Kombination von Sencha Touch und PhoneGap verwendet wurde. Sie ist auf das wichtigste begrenzt und stimmt größtenteils mit der MVC Projektstruktur von Sencha Touch überein. Abweichungen von Senchas Standardprojektstruktur sind in der Organisation der Bibliotheken zu erkennen.

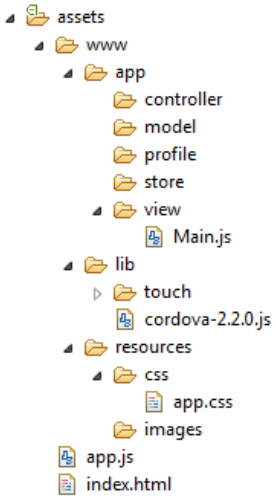

Abbildung 19 Initiale PhoneGap&SenchaTouch Projektstruktur nach MVC

Die View „Main“ stellt im Regelfall die Hauptview dar. In dem Ordner „lib“ ist zum einen die Cordova und Sencha Touch Bibliothekhinterlegt. Diese Struktur teilt die App in erster Linie technisch auf nach MVC. Eine fachliche Aufteilung nach Dialogen innerhalb der entsprechenden Ordner ist daraufhin möglich. Die hier erwähnte technische und fachliche Aufteilung bezieht sich nicht auf die Aufteilung nach dem fachlichen Dialogkern und der technischen Präsentation, sondern darauf, ob und wann eine Aufteilung nach Dialogen stattfindet.

6.4.1.3. **Umstrukturierung der Projektstruktur**

JavaScript Dateien

Die JavaScript Dateien kann man unterschiedlich organisieren. Der einfachste Weg ist es sich an der vorhandenen MVC Struktur zu orientieren, wie links in der Abbildung 20. Dabei ist es vorteilhafter die Controller und Models mit einem Postfix im Camel-Case Notation zu versehen, der die Zuordnung zur Aufgabe im Quasar Client Modell beschreibt. Dies hat den Vorteil, dass jede Datei anders heißt und man anhand des Dateinamen schon erkennen kann, wohin diese Datei gehört.

Alle Models, Views, Controller, Stores und Profile müssen in der app.js-Datei registriert werden, um Zugriff auf diese zu haben.

Wird eine App sehr groß, kann man, um das Projekt übersichtlicher zu halten, auch Unterordner für die Models, Views und Controller erstellen [SenchaDoc 2013]. Sie müssen in der app.js-Datei entsprechend angepasst deklariert werden. Das Anlegen von Unterordnern für jede Dialogkomponente würde beispielweise bei mehreren View Elementen einer Maske eine bessere Übersicht verschaffen. Controller und Model

Unterordner sollten, wie auch im linken Beispiel der Abbildung 20, bei den JavaScript Dateien mit einem Postfix versehen werden. Das mittlere Bild der Abbildung 20 zeigt ein Beispiel hierfür.

Es ist möglich, außerhalb des „app“-Ordners, Dialogkomponenten anzulegen. Die Dialogkomponenten sind dabei in erster Linie fachlich getrennt und auf zweiter Ebene erst technisch. Ein Beispiel dafür ist in der Abbildung 20 rechts zu sehen. Es handelt sich dabei nicht um den empfohlenen Standardweg von Sencha, da dieser Ansatz für einzelne Komponenten verwendet wird, die in mehreren Apps wiederverwendet werden [SenchaDoc 2013].

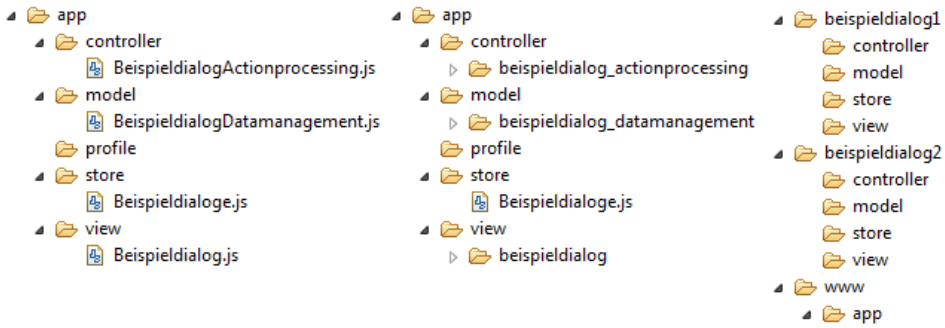

Abbildung 20 Links: Standard MVC Struktur, Mitte: Geschachtelte MVC Struktur, Rechts: Externe Dialogkomponenten

Ressourcen Dateien

Für die Ressourcen Dateien kann man auch hier das gleiche Prinzip wie bei Android anwenden. Man stellt den entsprechenden Ressourcennamen ebenfalls dem Dialognamen als Präfix voran und trennt diese mit einem Unterstrich. So ist die Zuordnung sofort erkennbar.

Übersichtlicher als die erste Alternative ist es einen Unterordner pro Dialog zu erzeugen und darin die entsprechenden Ressourcen abzulegen. In Abbildung 21 ist ein Beispiel dafür skizziert.

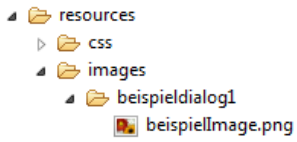

Abbildung 21 Organisation der Ressourcen mit Unterordnern

6.4.2 Dialogaufteilung

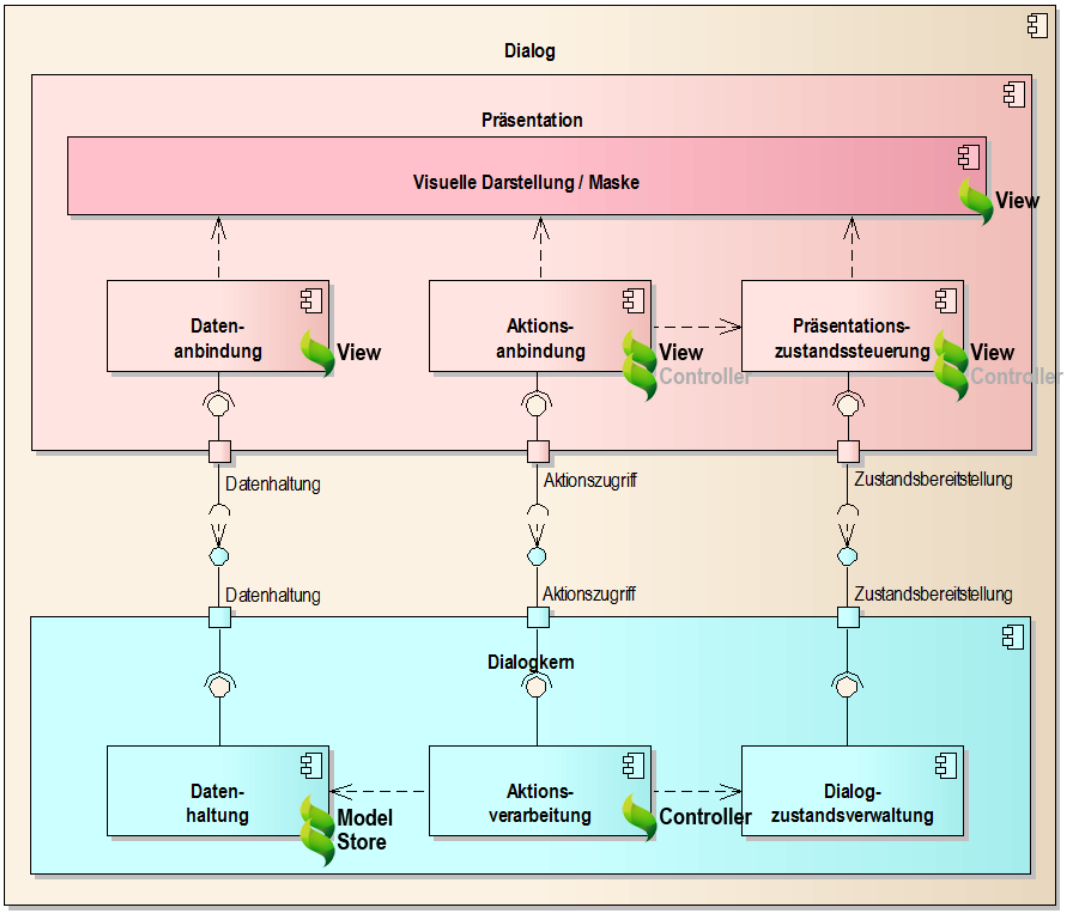

Abbildung 22 Abbildung der Sencha Touch-Bestandteil auf die Quasar Client Dialogarchitektur

Die Abbildung 22 zeigt die Projektion der Sencha-Touch Bestandteile auf die Quasar Clientarchitektur. Je nach Umsetzung kann sich ein Teil der Aktionsanbindung und der Präsentationszustandssteuerung im Sencha Controller befinden. Verwendet man Events zur Anbindung von Aktionen ist auf Controller-Seite ein Dispatcher, der die Aktion an eine spezifische Methode bindet. Der Dispatcher wurde in dieser Arbeit als Teil der Aktionsanbindung angesehen. Auch ein Teil der Präsentationszustandsteuerung kann sich im Controller befinden, wenn vom Controller aus die Eigenschaften der View-Elemente verändert werden.

Im Folgenden wird noch etwas genauer auf die einzelnen Komponenten in Verbindung mit Sencha Touch eingegangen.

6.4.2.1. **Dialogkern**

Den Dialogkern bilden jeweils alle Model, Controller und verwendete Stores. Beispiel-Quellcode des Dialogkerns wird am Beispiel vom WhatsCappPhoneGap-Projekt in Kapitel 7.3.3.2 gezeigt.

Datenhaltung
Daten können auch mit PhoneGap und Sencha Touch auf verschiedene Weisen abgelegt werden.

Der gängigste Weg in Sencha Touch Daten zu speichern ist einen Sencha Touch Store zu verwenden. Sie entkapseln Model Objekte und werden durch die Verwendung von Proxys befüllt. Die Proxys sind für das Laden und die Speicherung von Model Daten verantwortlich [CordovaDoc 2013]. Aus diesem Grund bildet ein Sencha Touch Model zusammen mit einem Store die Datenhaltung einer Dialogkomponente. Das Model spezifiziert die Struktur der Datenobjekte und der Store stellt die Daten bereit. Der Teil, den der Store übernimmt kann auch im Model implementiert werden. In so einem Fall würde das Model alleine die Datenhaltung darstellen. Darüber hinaus ist der Store bzw. das Model auch für die Anwendungsanbindung verantwortlich, was nicht mehr zur Datenhaltung zählt denn es wird an dieser Stelle die Datenquelle spezifiziert.

Es existieren zwei Hauptproxytypen. Clientseitige und serverseitige Proxys. LocalStorage Proxys entsprechen dem Web Storage aus PhoneGap, da sie ebenfalls Daten innerhalb des Browsers ablegen. MemoryProxys gehören ebenfalls zu den clientseitigen und speichern die Daten im Speicher, doch dieser geht beim neu Laden einer Seite verloren. Die serverseitigen Proxys laden und speichern ihre Daten auf einem entfernten Server [CordovaDoc 2013].

Seit Sencha Touch 2.1.0 existiert auch die Möglichkeit einen SQL Proxy einzusetzen, der eine Web SQL Datenbank nutzt, um Daten über SQL zu persistieren [CordovaDoc 2013].

Aktionsverarbeitung
Aktionen werden im Controller, wie im analogen Android Kapitel, durch eine Methode verarbeitet.

Dialogzustandsverwaltung
Eine Dialogzustandsverwaltung ist durch Sencha Touch selbst nicht direkt gegeben. Es können aber manuell Zustände implizit verwaltet werden, indem einzelne bzw. mehrere Eigenschaften abgefragt werden. Solche Abfragen würden separat oder im Controller stattfinden.

6.4.2.2. **Präsentation**

Die Präsentation findet sich in den JavaScript View Dateien wieder. Beispiel-Quellcode der Präsentation wird am Fallbeispiel WhatsCapp für PhoneGap & Sencha Touch im Kapitel 7.3.3.1 gezeigt.

Datenanbindung
Bei der Verwendung von Sencha Touch Stores können die Daten der Präsentation über den Namen von UI-Elementen an die Storedaten angebunden werden. Dazu muss der Name der UI-Elemente dem im Model definierten Attributnamen entsprechen. Das tatsächliche Laden der Daten und damit auch die Aktualisierung der Daten wird dabei manuell durch eine Aktionsverarbeitung ausgelöst.

Aktionsanbindung
Es können Maskenelemente der View mit einem Handler oder Listener versehen werden. Diese lösen daraufhin eine Methode in der View selbst aus. Die Aktionsanbindung ist demnach prinzipiell schon durch Sencha Touch selber gegeben.
Da eine Methode in so einem Fall direkt in der View ausgelöst wird, könnte man es so sehen, dass sich ein Teil des Dialogkerns in der View befindet. Verarbeitet man in dieser Methode nur Aufgaben, die ausschließlich die Präsentationsschicht betreffen, könnte man es so interpretieren, dass diese Methode der Präsentationsaktionsverarbeitung entspricht. Innerhalb dieser Methode kann dann auch ein Methodenaufruf in der Dialogkernschicht initiiert werden. Man bindet entweder Benutzerereignisse direkt an Methoden des Controllers oder nutzt stattdessen Events. Bei der Nutzung von Events wird ein selbst definierter Event in der Präsentationsmethode gefeuert, welcher dann von dem verantwortlichen Controller abgefangen und an eine Methode gebunden werden kann. Daraufhin findet die entsprechende Aktionsverarbeitung statt.

Präsentationszustandssteuerung
Die Präsentationszustandssteuerung wird durch Sencha Touch umgesetzt. Benutzerereignisse, wie die Auswahl einer Checkbox, werden über Events gesteuert, die dann gefeuert werden. Diese Events können wiederum Aktionen durchführen, die entsprechende Eigenschaftsvariablen setzen. Auch hier gilt, wie beim Android Binding, der Umkehrschluss. Es kann beispielsweise auch hier ein deaktivierter Button durch die genannten Eigenschaftsvariablen abgebildet werden, was sich dann auf die Darstellung auswirkt.

6.5 Titanium

Die aktuellste SDK-Version aus dem Februar 2013 ist Titanium 3.0.2. Der Majorrelease Titanium 3.0 wurde im Dezember 2012 veröffentlicht. Es bringt ein MVC-Framework mit sich, was eine saubere Trennung der visuellen Darstellung von der Businesslogik ermöglicht. Das MVC-Framework trägt den Namen Alloy [AppceleratorBlog 2013]. Die App, die im Rahmen dieser Abschlussarbeit entwickelt wurde, wurden mit dem SDK 3.0.0 entwickelt.

In Kapitel 6.5.1 wird analog zum PhoneGap & Sencha Touch in Kapitel 6.4 als erstes die Realisierungsmöglichkeit einer Dialogkomponente in Bezug auf Titanium und dem verwendeten Alloy Framework des neuen SDKs beschrieben. Das Kapitel 0 beschreibt daraufhin für ausgewählte Subkomponenten der Dialogarchitektur ob und wodurch diese mit dem Alloy-Framework umgesetzt werden können.

6.5.1 Realisierungsmöglichkeiten einer Dialogkomponente

6.5.1.1. **Umfang einer Dialogkomponente**

Wie auch die Dialogkomponenten der anderen Tools bzw. Frameworks besteht sie auch bei Titanium aus mehreren Elementen. Folgende Beschreibungen beziehen sich auf ein Titanium Alloy Projekt. Eine Dialogkomponente setzt sich zusammen aus:

- Model (*.js Datei)
- View (*.xml Datei)
- Controller (*.js Datei)
- Style (*.tss Datei)
- Benötigte Ressourcen
    - Bilder
    - Zusätzliche andere Dateien, die vom Dialog verwendet werden

In der View wird die Struktur der Benutzeroberfläche per XML deklariert, wobei die Style-Datei die Formatierung der View übernimmt. Die View stellt die Daten dar und interagiert mit dem Model. Die *.tss Datei ähnelt *.css Dateien sehr. Models stellen die Businesslogik bereit und beinhalten Regeln, Daten und Zustände der Applikation. Der Controller beinhaltet die Präsentationslogik und verarbeitet somit Aktionen. Benötigte Ressourcen werden hier unter Assets untergeordnet [AppceleratorDoc 2013].

6.5.1.2. **Initiale Projektstruktur**

Die initiale Projektstruktur wird komplett über die IDE Titanium Studio generiert. Dabei kann man bei der Projekterstellung zwischen zwei Projektvarianten wählen. Es besteht die Wahl zwischen einem klassischen Titanium Projekt oder aber einem auf MVC basierenden Alloy Projekt.

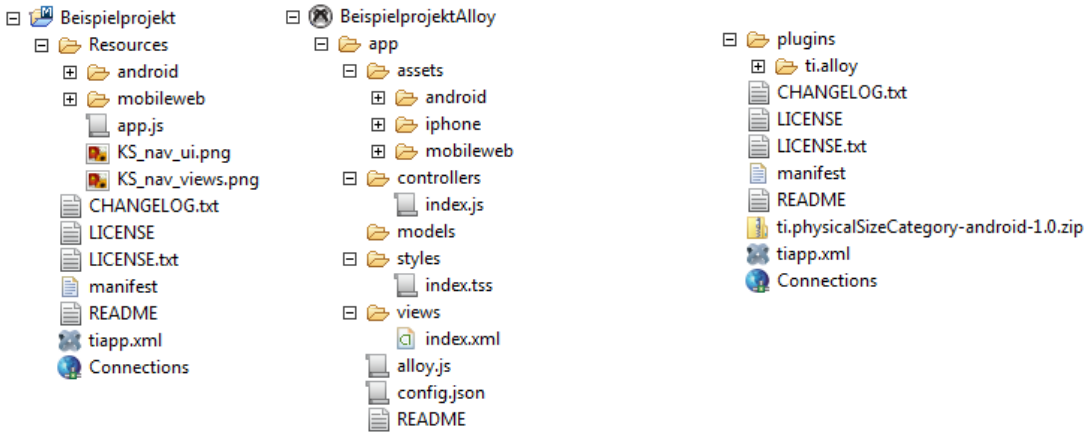

Abbildung 23 Links: Initiale klassische Titanium Projektstruktur. Rechts: Initiale Alloy Projektstruktur

Wie zu sehen ist, ist ein Alloy Projekt von sich aus stärker strukturiert als eine klassische Titanium App. Eine ähnliche Struktur wie bei Alloy wäre auch für das klassische Titanium Projekt denkbar.

6.5.1.3. **Umstrukturierung der Projektstruktur**

Dateien

Alloy ist in seinen Strukturen unflexibel. Die Dateien des Models, der View, des Controller und der Styles setzen voraus, dass die Dateien alle den gleichen Namen haben wie in Abbildung 24 visualisiert ist. Aus diesem Grund kann die vorgeschlagene Nutzung von Postfixen aus dem vorangehenden Sencha Touch Kapitel nicht übernommen werden. Die Dateien werden hier einfach einheitlich mit dem Dialognamen beschrieben.

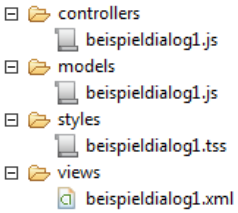

Abbildung 24 Organisation der MVC-Dateien und Titanium Style Sheets

Ressourcen Dateien

Da die übrigen Dateien etwas unflexibel in der Benennung sind, sollte man die Ressourcen Dateien ähnlich strukturieren, um das Projekt in der Benennung konsistent zu halten. Das bedeutet, dass auch bei den Ressourcen keine Präfixe genutzt werden sollten. Stattdessen ist es sinnvoll die Ressourcen wie z.B. Bilder nach Dialogunterordnern zu unterteilen (Abbildung 25).

Es ist möglich für Android, iOS und für das MobileWeb jeweils unterschiedliche Ressourcen zu verwenden. Dazu sieht man in Abbildung 23 unter dem Ordner „assets“ drei verschiedene Ordner. Für diese Ordner wird diese Datenorganisation empfohlen.

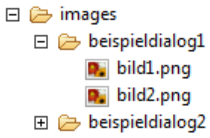

Abbildung 25 Ressourcenorganisation in Alloy am Beispiel von Bildern.

6.5.2 Dialogaufteilung

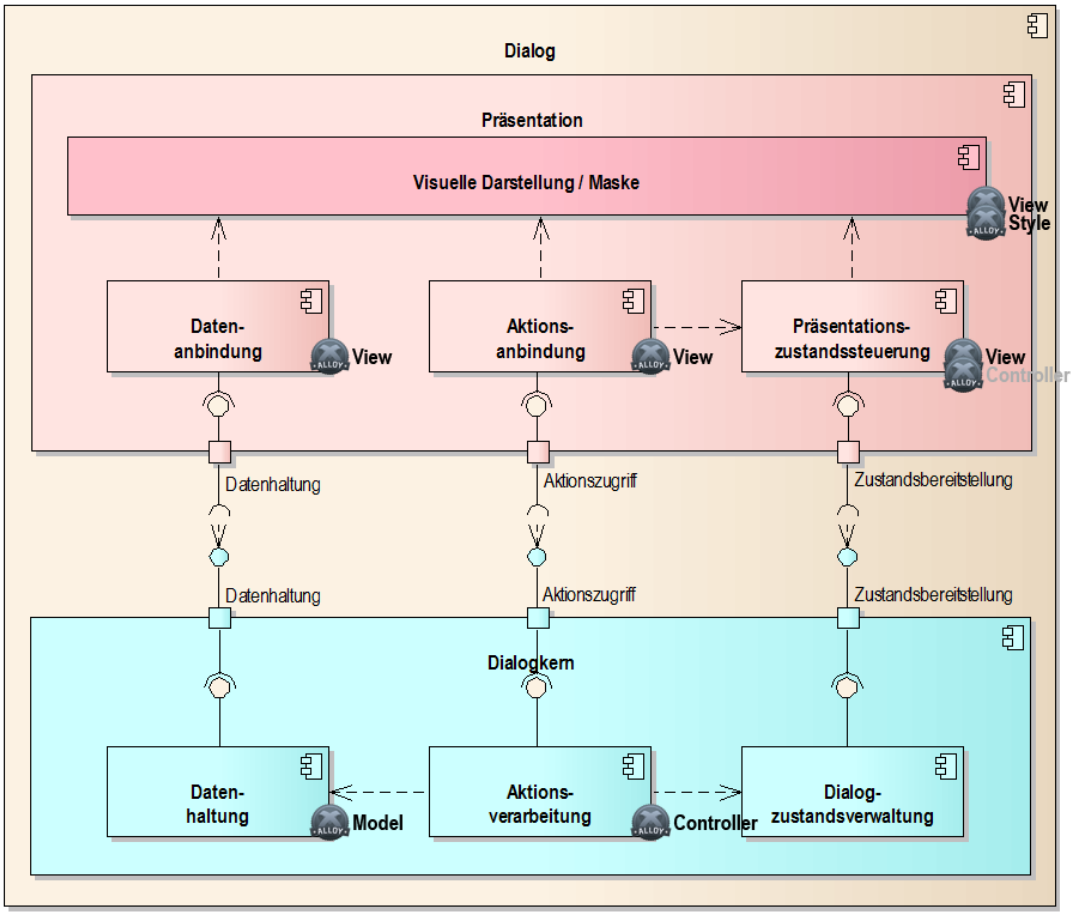

Abbildung 26 Abbildung der Titanium Alloy-Bestandteil auf die Quasar Client Dialogarchitektur

Die Abbildung 26 zeigt im Groben die Projektion der Titanium Alloy Bestandteile auf die Quasar-Client-Architektur. Je nach Umsetzung kann sich ein Teil der Präsentationszustandssteuerung im Alloy Controller befinden. Wenn Eigenschaften der View-Elemente aus dem Controller verändert werden, befinden sich Teile der Präsentationszustandsteuerung vermischt mit der Aktionsverarbeitung im Controller, der zum Dialogkern gehört.
Im Folgenden wird noch etwas genauer auf die einzelnen Komponenten in Verbindung mit Alloy eingegangen.

6.5.2.1. **Dialogkern**

Den Dialogkern bilden jeweils alle Models und Controller in Titanium. Aus dem WhatsCappTitanium-Projekt werden Quellcode-Beispiele des Dialogkerns mit dem Alloy Framework im Kapitel 7.4.3.2 gezeigt.

Datenhaltung
Das Model in Alloy beinhaltet Regeln, Daten und Zustände einer Applikation [AppceleratorDoc 2013]. Daten können mit Titanium über Eigenschaften, Datenbanken oder dem Dateisystem abgelegt werden was aber alles zum Anwendungskern gehört. Die Eigenschafts- (Properties-) API bietet einen leichtgewichtigen Key-Value Store an und eignet sich aus diesem Grund zum Speichern von kleineren Datenmengen, wie beispielsweise Applikationseinstellungen. Für größere Datenmengen sollte eine Datenbank eingesetzt werden. Die Titanium Database API realisiert zu diesem Zweck den Zugriff auf lokale SQLite3 Datenbanken, die sowohl von Android als auch iOS unterstützt werden. Die Dateisystem API ermöglicht es binäre Daten wie Bilder im Dateisystem des Gerätes abzulegen [AppceleratorWiki 2013].

Aktionsverarbeitung
Aktionen werden, wie auch bei SenchaTouch&PhoneGap beschrieben, ebenfalls durch die Ausführung einer Methode im Controller verarbeitet.

Dialogzustandsverwaltung
Eine Dialogzustandsverwaltung ist wie bei Sencha Touch auch bei Titanium von sich aus selbst nicht direkt gegeben. Eine implizite Dialogzustandsverwaltung wäre auch hier analog zu Sencha Touch (Kapitel 6.4.2.1) umsetzbar.

6.5.2.2. **Präsentation**

Aus Abbildung 26 ist ersichtlich, dass sich die Präsentation aus den View und den Style-Dateien zusammensetzt. Beispiel-Quellcode der Präsentation wird im Kapitel 0 aus dem WhatsCappTitanium-Projekt gezeigt.

Datenanbindung
Um Daten zwischen der Datenhaltung und der Präsentation synchron zu halten, bietet sowohl Alloy mit dem Model-View Binding als auch das bereits erwähnte MVC Framework Backbone eine Möglichkeit der Datenbindung an [AppceleratorDoc 2013].

Aktionsanbindung
Die Aktionsanbindung kann durch Listener umgesetzt werden. Es können in der View die XML-Elemente mit Listenern versehen werden, um bei entsprechender Aktion im Controller die entsprechende Methode auszuführen. Ein konkretes Beispiel wäre ein Button, der mit einem onClick-Listener versehen auf Knopfdruck eine Methode im Controller auslöst.

Präsentationszustandssteuerung
Die Präsentationszustandssteuerung wird durch Titanium umgesetzt. Benutzerereignisse werden wie bei Sencha Touch durch Events gesteuert. Beim Eintreten eines definierten Ereignisses werden diese Events gefeuert. Als Reaktion auf die Events können wiederum entsprechende Eigenschaftsvariablen gesetzt werden. Auch hier gilt, wie beim Android Binding, der Umkehrschluss. Ein deaktivierter Button kann durch eine Konfigurationsvariable abgebildet werden, was in der visuellen Darstellung auch entsprechend angezeigt wird.

7 Umsetzung einer Fallstudie

Dieses Kapitel stellt nach der erfolgten CPT Wahl des 0. Kapitels und Analyse der Integration des 6. Kapitels die Umsetzung einer Fallstudie dar und beschreibt inwiefern die funktionalen Anforderungen, die in Kapitel 4 ermittelt wurden, umgesetzt werden können.
Das Unterkapitel 7.1 beschreibt zunächst die Fallstudie „WhatsCapp“, die als native Android Entwicklung zur Verfügung stand und die Grundlage für weitere Implementierungen bildete. Anschließend wird im Unterkapitel 0 die Realisierungsanalyse und Architekturanalyse der bestehenden Android-App beschrieben.

Im Unterkapitel 7.3 wird daraufhin die Umsetzung der Fallstudie mit PhoneGap & Sencha Touch beschrieben und im Unterkapitel 7.4 selbiges mit Titanium.

7.1 WhatsCapp

WhatsCapp ist eine auf Android basierende App, die für Technologie-Schulungen bei Capgemini zum Einsatz kommt. Der Name lehnt sich an die Kommunikations-App WhatsApp an, die einen Austausch von Nachrichten und Medien ermöglicht [WhatsApp 2013].

Mit WhatsCapp werden Mitarbeiterdaten bereitgestellt, die aus einer lokalen Datei geladen werden. Die Mitarbeiterdaten enthalten Informationen wie beispielsweise Name, Telefonnummern, Büroraumnummern, aber auch Daten wie die Niederlassungszugehörigkeit. Die App deckt mehrere Anwendungsfälle ab, wie das Ändern eines Mitarbeiterbildes, das Anzeigen des eigenen Standortes und den der Mitarbeiter sowie den Austausch von Chatnachrichten über einen gemeinsamen Kanal.

7.1.1 Anwendungsfalldiagramm

Im folgenden Diagramm sind Anwendungsfälle dargestellt, die von der WhatsCapp App umgesetzt werden sollen. Der Hauptzweck der App ist es als Nachschlagewerk für firmeninterne Mitarbeiterdaten zu dienen. Da es sich nicht zwangsweise um

Mitarbeiterdaten handeln muss, wird im Diagramm der allgemeinere Begriff „Kontakte“ verwendet.

Bei einer größeren Anzahl von Mitarbeitern macht es Sinn auch nach einzelnen Personen suchen zu können. Dem Benutzer soll es demnach möglich sein durch eine Suche Kontakte zu finden. Zu jedem Kontakt gehört ein Bild, welches vom Benutzer durch die Aufnahme eines neuen Bildes ersetzt werden kann. Zusammen mit dem Bild sollen Kontaktdatendetails angezeigt werden und auch direkt Kontakte angerufen werden können.

Auch die Angabe des derzeitigen geografischen Standortes des Kontaktes kann vom Benutzer auf einer Karte eingesehen werden. Außerdem soll auch der eigene Standort auf einer Karte sichtbar sein. Dadurch sind Kontakte um den aktuellen Standort identifizierbar. Darüber hinaus kann der Benutzer durch Textnachrichten in einem zentralen Chat mit anderen Kontakten kommunizieren. Vorab ist allerdings das Registrieren mit einem Benutzernamen notwendig. Nach der Registrierung kann der Benutzer sich auch wieder abmelden.

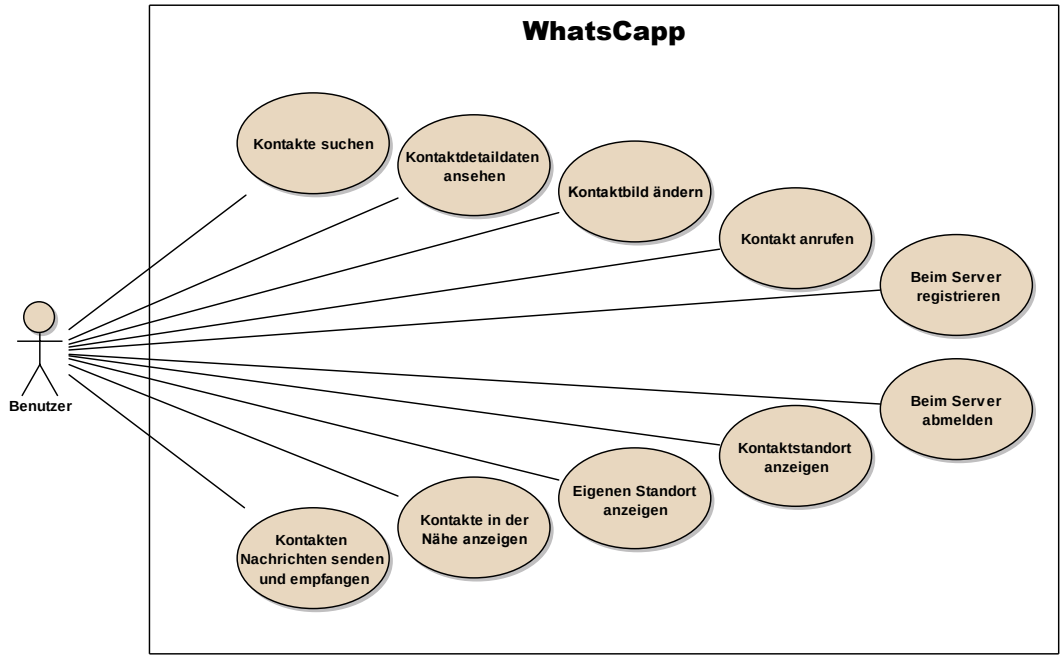

Abbildung 27 Anwendungsfalldiagramm von WhatsCapp

7.1.2 WhatsCapp-App

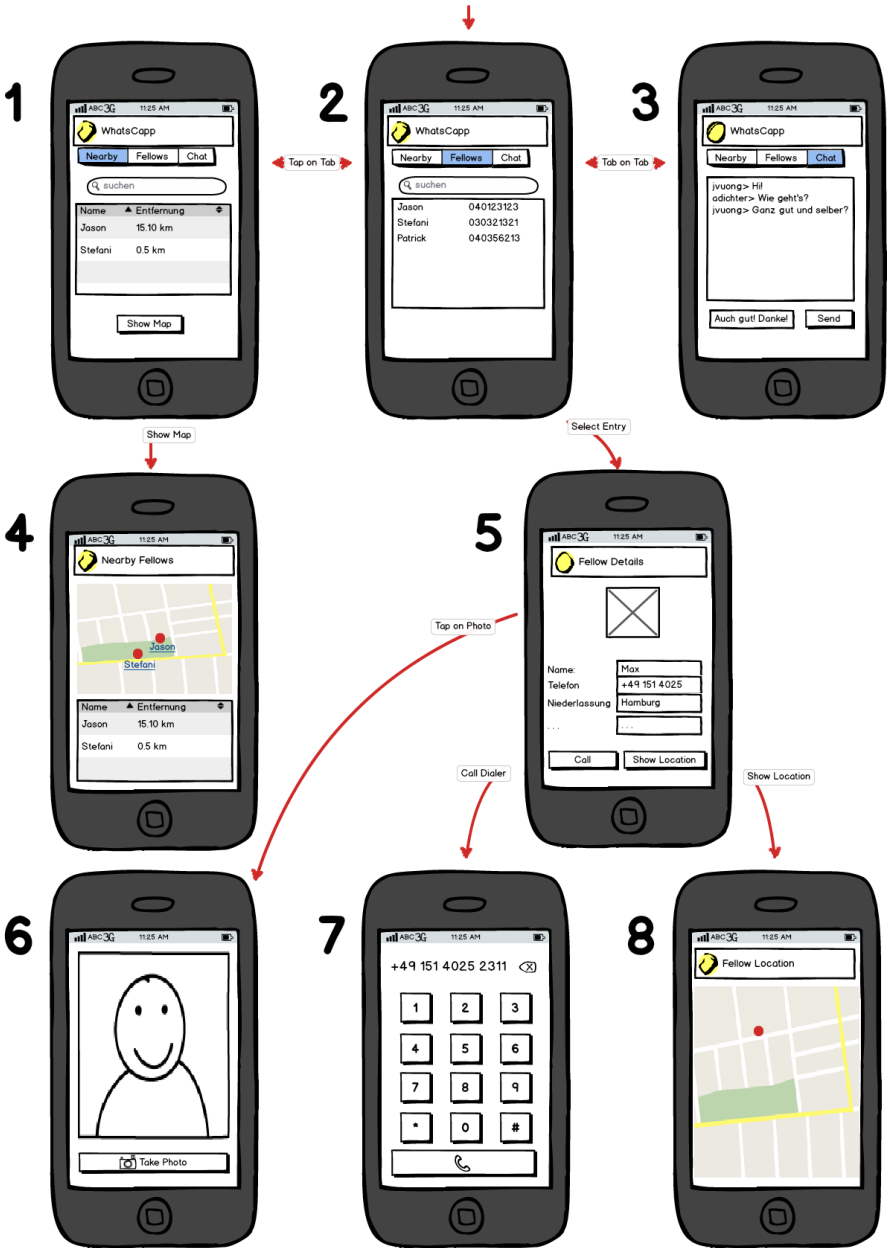

Abbildung 28 Mock-Up zur Entwicklung der Applikation

Zur Visualisierung der Anwendungsfälle und der daraus resultierenden App wurden für die Schulungs-App WhatsCapp Mock-Ups erstellt. Diese dienten als Basis für die Abbildung 28.

Die App soll in drei Tabs unterteilt sein und zeigt beim Start den Fellow-Tab an, in dem alle Kontakte in einer Liste angezeigt werden. Wird einer der Kontakte per Berührung ausgewählt, erscheint die Detailansicht des Kontaktes. In dieser Detailansicht sind drei Aktionen möglich. Bei Berühren des Kontaktbildes wird die Kamera des Gerätes aufgerufen, woraufhin ein neues Bild aufgenommen werden kann. Das aufgenommene Bild ersetzt in der Detailansicht das vorherige Bild. Der „Call“-Button überträgt die gespeicherte Telefonnummer des Kontaktes an die Anruf-App des Gerätes, wohingegen der „Show Location“-Button eine Karte mit dem Standort des Kontaktes aufruft.

Der Chat-Tab ermöglicht es in einem zentralen, gemeinsamen Chat Nachrichten zu lesen, einzugeben und zu versenden.

Der letzte Tab „Nearby“ zeigt alle Kontakte bzw. Fellows, die sich in der Nähe des eigenen Standortes befinden in einer Liste mit der Luftlinienentfernung an. Bei dem Berühren des „Show Map“-Buttons werden die Standorte in einer Karte visualisiert.

Mit dem Ziel die App zu optimieren wurde für die neu entwickelten Apps, mit PhoneGap & SenchaTouch und Titanium, die Benutzeroberflächen 1 und 4 aus Abbildung 28 zu einer neuen Oberfläche, wie in Abbildung 29 zu sehen ist, zusammengefasst. Die Optimierung wurde aus zwei Gründen vorgenommen. Anwender sollten zum einen schneller auf die Map-Ansicht zugreifen können und zum anderen sollte eine Reduktion der Masken vorgenommen werden.

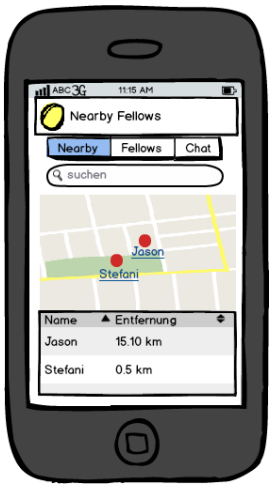

Abbildung 29 Mock-Up des zusammengefassten neuen Nearby-Tabs

7.2 WhatsCapp mit Android

Die nativ mit Android entwickelte WhatsCapp-App war bereits in einer sehr fortgeschrittenen Version vorhanden. Aufbauend auf dieser Version sind kleinere Änderungen vorgenommen worden. Bevor allerdings Änderungen durchgeführt wurden, fanden drei Arten von Analysen statt. Zum einen findet eine formale Kurzanalyse zur Realisierung der Dialogkomponenten statt und zum anderen eine Untersuchung der bestehenden App-Architektur nach den Quasar-Client-Integrationsempfehlungen aus dem Kapitel 6 „Analyse der Integration der Quasar Client Dialogkomponenten-Architektur“. Anschließend findet eine Untersuchung bezüglich der umgesetzten funktionalen Anforderungen statt.

7.2.1 App-Einblick

Der aktuelle Stand der App ist umfangreich und geht über die sechs Screenshots, die in der Abbildung 30 gezeigt werden hinaus. Dialoge wie die Menus, die Kamera- und Anrufansicht sind beispielsweise nicht dargestellt. Die beiden Ansichten werden aber in den Kapitel 7.3 und 7.3.4 gezeigt. Sie unterscheiden sich prinzipiell nicht, weil bei allen ein nativer Aufruf stattfindet und somit die gleiche App aufgerufen wird. Einen kleinen Unterschied gibt es aber beim Anrufen von Kontakten. Die Android App ruft direkt den Kontakt an, wobei bei den anderen Apps aus den folgenden Kapiteln 7.3 und 7.4 zuerst die Wählmaske aufgerufen wird, Das liegt daran, dass bei den anderen beiden Apps der Anruf durch einen Telefon-Link umgesetzt sind.

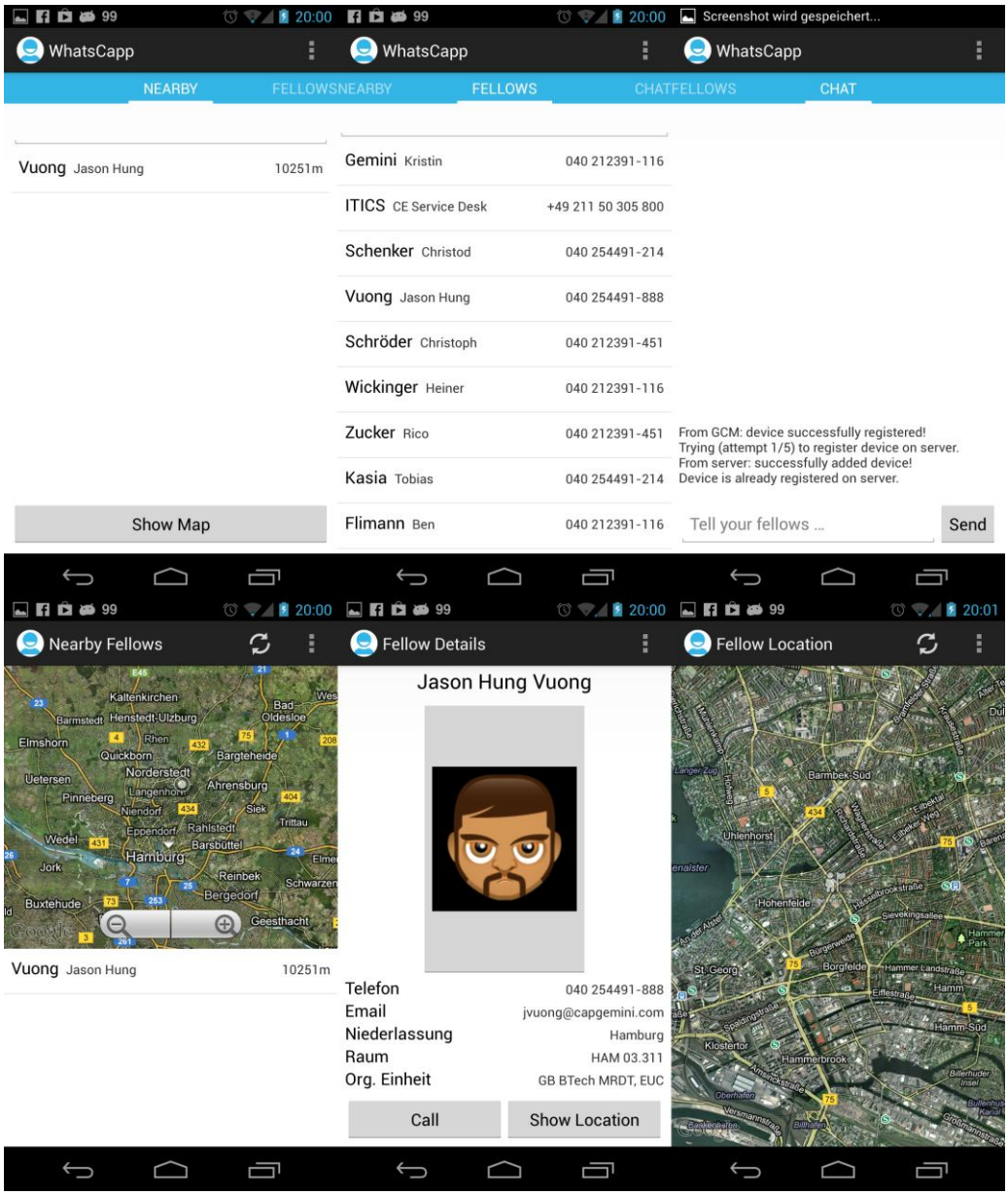

Abbildung 30 Screenshots aus der nativ entwickelten Android App

7.2.2 Realisierungsanalyse

In Kapitel 6.2.2.3 wird beschrieben, wie ein Projekt formal strukturiert werden kann und Konventionen zur Benennung beschrieben. Diese Strukturierung spiegelt im Idealfall die Architektur des Projektes wieder, was die Übersichtlichkeit verbessert.

7.2.2.1. **Projektstruktur**

Abbildung 31 zeigt einen Ausschnitt aus der Projektstruktur der nativ entwickelten Android App. Sie zeigt die Organisation der Dateien. Anders als im Kapitel 6.2.2.3 beschrieben, fand hier keine Unterteilung nach dem Dialogkern und der Präsentation statt. Dies ist darauf zurückzuführen, dass auch keine explizite Trennung von Dialogkern und Präsentation durchgeführt wurde. Beide Dialog-Subkomponenten sind somit in diesem Projekt in jeweils einer Activity miteinander verschmolzen. Es wird beispielsweise in einer Activity direkt eine Aktion verarbeitet und eine weitere Activity aufgerufen.

Es ist aber eine Strukturierung nach Dialogen erkennbar, was in diesem Fall auch für eine ausreichend gute Übersicht sorgt und auch die vorherrschende A-Architektur wiederspiegelt.

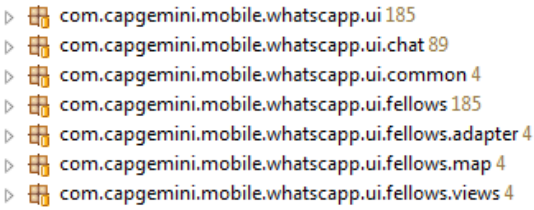

Abbildung 31 Organisation von Dialog Java Dateien der bestehenden WhatsCapp App

7.2.2.2. **Ressourcen**

Aus der Abbildung 32 ist ersichtlich, dass die vorgeschlagenen Konventionen aus Kapitel 6 bei den ausgewählten Ressourcen nur teilweise umgesetzt sind. Bei den Bildern und Werten wurden gar keine fachlichen Präfixe verwendet.

Bei den vorhandenen Bildern handelt es sich allerdings um dialogübergreifende Bilder, wie beispielsweise dem Suche-Icon. Daher sind sie nicht einem Dialog eindeutig zuzuordnen. Die Werte-XML-Dateien enthalten bis auf die strings.xml weniger als 3 Werte, so dass hier auf eine fachliche Aufteilung nach Dialogen verzichtet werden kann. Bei den genannten String Werten ist zwar innerhalb der Datei eine optische Unterteilung durch Kommentare vorhanden, doch eine räumliche Trennung wäre übersichtlicher.

Die Layout-Dateien sind nach der A-Architektur genau richtig unterteilt. Es ist sofort ersichtlich welche Layouts fachlich zusammengehören. Nur fellow_item.xml sollte vollständigkeitshalber zu fellow_list_item.xml umbenannt werden. Die Menus haben zusätzlich zum Dialognamen, „menu“ als Präfix. Dieses Präfix ist zwar nicht notwendig, da Menus sofort erkennbar sind, aber auch nicht schädlich.

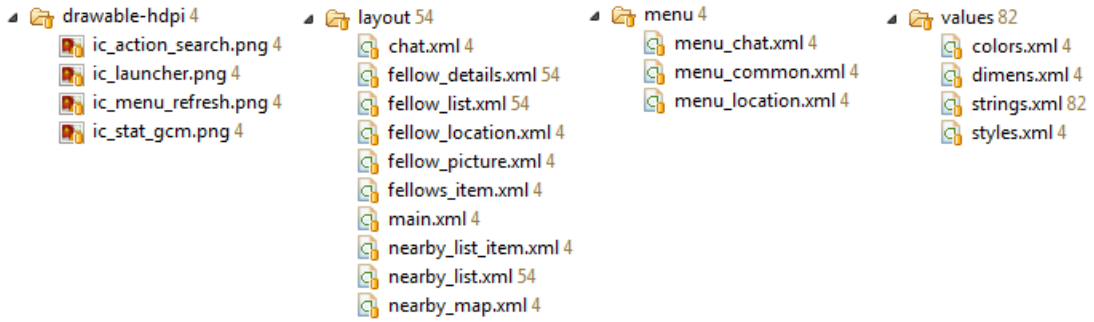

Abbildung 32 Ressourcen der nativ entwickelten Android App von links nach rechts Bilder, Layouts, Menus, Werte

7.2.3 Architekturanalyse

7.2.3.1. **A-Architektur**

Die Abbildung 33 visualisiert anhand eines Komponentendiagramms die A-Architektur der gegebenen nativen Android App von WhatsCapp. Die native Android App von WhatsCapp setzt alle Anwendungsfälle in Gänze um. Die Abbildung 33 zeigt den Quasar Client mit einem grünen Hintergrund. Darin sind mehrere Dialogkomponenten enthalten, welche die Anwendungsfälle umsetzen. Jeder der Dialoge greift dabei auf den Anwendungskern zu, um Lokations-, Kontakt- oder Chatdaten abzurufen.

Der Quasar Anwendungskern ist mit einem blauen Hintergrund dargestellt. Er ist in zwei Bereiche aufgeteilt. Der erste Bereich befindet sich zusammen mit dem Quasar Client, womit die App an sich gemeint ist, auf der Seite des Clients. Der zweite Teil des Quasar Anwendungskerns befindet sich auf dem räumlich getrennten Server. Die Bereichsaufteilung ist dabei durch die Farben Gelb und Lila visualisiert. Gelb steht dabei für den Bereich Client und die Farbe Lila für den Bereich Server.

Lokationsdaten der Kontakte (Fellows) werden vom Server bereitgestellt und aktualisiert. Diese werden von den Dialogen Nearby, NearbyList und FellowLocation benötigt. Die anderen Kontaktdaten liegen lokal im Bereich des Clients und werden von dem Anwendungskern für die Dialoge FellowPicture, FellowDetails und FellowList bereitgestellt. Der Chat-Dialog nutzt einen ChatService, der sich um die Registrierung beim Server und das Senden und Empfangen von Chat-Nachrichten kümmert. Für den Chat Service wird der Google Cloud Messaging Service genutzt.

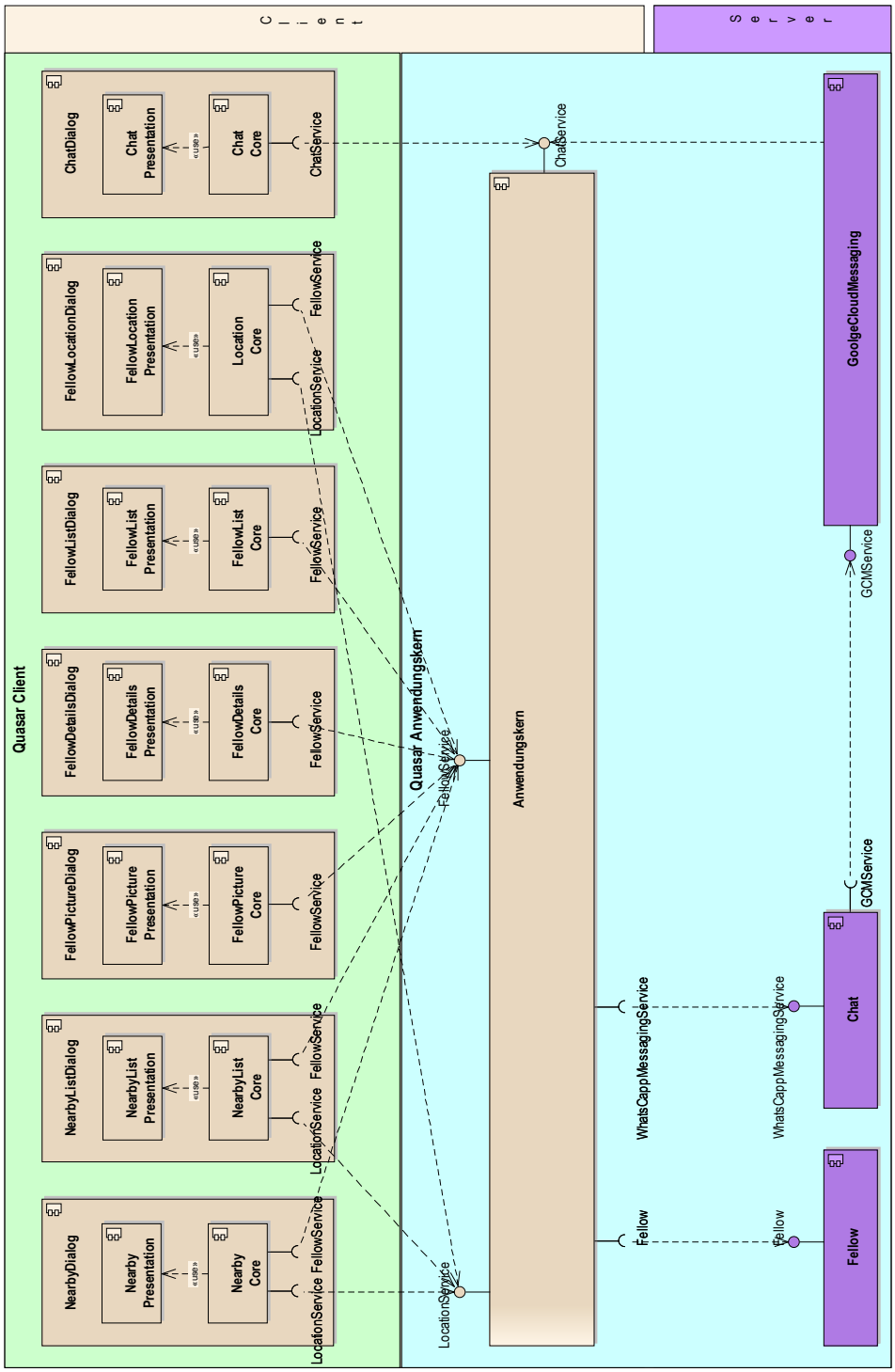

Abbildung 33 A-Architektur der nativen Android WhatsCapp App

7.2.3.2. **Dialogarchitektur**

Kapitel 6.2 beschrieb ein Konzept zur Umsetzung der Quasar Client Dialogarchitektur mit Android. Dabei geht es insbesondere um die Aufteilung nach den Sub-Komponenten Dialogkern und Präsentation.

In dem Konzept wird beschrieben, dass ein zusätzlicher Controller dabei helfen würde eine klare Trennung zwischen fachlichem und technischem Code zu schaffen. In dem bestehenden Android Projekt von Capgemini wurde auf zusätzliche Frameworks verzichtet. Außerdem wurde die App ohne die Kenntnis von der referenzierten Arbeit entwickelt. Aus diesen Tatsachen lässt sich die Vermutung ableiten, dass kein zusätzlicher Controller verwendet wurde. Der Code bestätigt diese Vermutung.

7.2.4 Anforderungsumsetzbarkeitsanalyse

In der folgenden Tabelle sind die 18 funktionalen Anforderungen aus Kapitel 4.3.3 mit einer Beschreibung bezüglich ihrer Bedeutung und ihrer Relevanz in der nativ entwickelten Android App beschrieben. Alle 18 funktionalen Anforderungen sind umsetzbar und benötigen keine zusätzlichen Frameworks. Es wird allerdings nur ein Teil der Anforderungen von WhatsCapp genutzt, da der Einsatz einzelner Anforderungen keinen Sinn macht.

An-forderungs-ID	Berechtigungen / Anforderungen	Um-setzbar?	Zusätze be-nötigt?	Von Whats Capp genutzt?	Beschreibung
1	Uneingeschränkter Internetzugriff	Ja	Nein	Ja	Den uneingeschränkten Internetzugriff erhält eine durch einen Eintrag der Berechtigung "INTERNET" im Androidmanifest. Ein Beispiel für die Verwendung wäre der Abruf von Lokationsdaten der Kontakte in WhatsCapp.
2	Inhalt des USB-Speichers und der SD-Karte ändern/löschen	Ja	Nein	Ja	Für das Lesen und Ändern von Inhalten eines externen Speichers sind zwei Berechtigungen nötig, die im Androidmanifest deklariert werden müssen. Zum Ändern wird WRITE_EXTERNAL_STORAGE und zum Lesen READ_EXTERNAL_STORAGE benötigt. Diese Berechtigungen sind beispielsweise für das Einlesen der Kontaktdatendatei notwendig oder aber auch für das Speichern von Kontaktbildern.
3	Netzwerkstatus anzeigen	Ja	Nein	Ja	ACCESS_NETWORK_STATE lautet die Berechtigung in einem Androidmanifest, um einen Zugriff auf den Netzwerkstatus zu erhalten. Damit kann geprüft werden ob eine Netzwerkverbindung besteht.

4	Vibrationsalarm steuern	Ja	Nein	Ja	Um Vibrationen zu steuern ist die Berechtigung VIBRATE notwendig. Es kann beispielsweise bei eingehenden Nachrichten Vibrationen erzeugt werden.
5	Ungefährer (netzwerkbasierter) Standort	Ja	Nein	Ja	Die Berechtigung ACCESS_COARSE_LOCATION ermöglicht auf den über das Netzwerk ermittelten Standort zuzugreifen, wodurch die aktuelle Position des WhatsCapp Benutzer angezeigt werden kann.
6	Genauer (GPS-) Standort	Ja	Nein	Ja	Die Berechtigung ACCESS_FINE_LOCATION ermöglicht dagegen auf den über GPS ermittelten genaueren Standort zuzugreifen, wodurch auch eine genauere Position des WhatsCapp Benutzer angezeigt werden kann.
7	Automatisch nach dem Booten starten	Ja	Nein	Ja	Um eine App automatisch nach dem Boot zu starten ist die Berechtigung RECEIVE_BOOT_COMPLETED notwendig, um die Broadcastnachricht ACTION_BOOT_COMPLETED zu erhalten. So wird die App benachrichtigt wenn der Bootvorgang des Geräts abgeschlossen ist und kann dementsprechend Aktionen ausführen. Für WhatsCapp ist diese Berechtigung eigentlich nicht nötig. Dennoch wurde diese Berechtigung testweise hinzu implementiert.
8	Bekannte Konten suchen	Ja	Nein	Ja	Um auf die registrierten Konten des Gerätes zuzugreifen wird die GET_ACCOUNTS Berechtigung im Androidmanifest benötigt. WhatsCapp nutzt zum Austausch von Chatnachrichten den Google Cloud Messaging (GCM) Dienst. Um den Dienst zu nutzen, muss auf dem Gerät ein Google Konto registriert sein und eine Zugriffberechtigung für die WhatsCapp App vorliegen.
9	Telefonstatus lesen und identifizieren	Ja	Nein	Nein	Die Berechtigung READ_PHONE_STATE im Androidmanifest erlaubt es auf die Telefonfunktionen des Geräts zuzugreifen. Es können Telefonnummer und die Geräte-ID erfasst werden. Außerdem kann geprüft werden, ob ein Telefonat geführt wird und die Rufnummer des Gesprächspartners gelesen werden. Diese Berechtigung benötigt WhatsCapp nicht.
10	WLAN-Status anzeigen	Ja	Nein	Nein	Die Berechtigung ACCESS_WIFI_STATE kann gegenüber dem dritten Platz dieser Tabelle ACCESS_NETWORK_STATE zur Verwirrung führen. (Keine Vorschläge) erlaubt einen Zugriff auf die Klasse WifiManager, wohingegen die verbleibende Berechtigung Zugriff auf den ConnectivityManager erlaubt. WifiManager ermöglicht es alle Aspekte bezüglich der WLAN Verbindung zu steuern. Es ist beispielsweise möglich auf die Ergebnisse von Accesspoint-Scans zuzugreifen oder auch eine Verbindung herzustellen oder zu trennen. WhatsCapp nutzt auch diese Berechtigung nicht.

11	Standby-Modus des Tablets deaktivieren Standby-Modus des Telefons deaktivieren	Ja	Nein	Ja	Die Berechtigung WAKE_LOCK ermöglicht es den Ruhezustand des Geräts zu deaktivieren und auch das Dimmen des Bildschirms zu verhindern.
12	Bilder und Videos aufnehmen	Ja	Nein	Ja	Um Bilder und Videos aufzunehmen, müssen zum einen die Berechtigung CAMERA gesetzt sein und zum anderen Features deklariert werden, die genutzt werden sollen. Beispiele für Features wären die Kamera selbst und der Autofokus. Bei WhatsCapp wird die Bildaufnahme zum Austausch der Kontaktbilder benötigt.
13	Kontaktdaten lesen	Ja	Nein	Nein	Um Kontaktdaten zu lesen ist die Berechtigung READ_CONTACTS erforderlich. Da WhatsCapp nicht auf die Kontaktdaten des Benutzers zugreift und eigene Kontaktdaten mit ausliefert, ist dieses funktionale Kriterium irrelevant für diese App.
14	Zugriff auf das Benachrichtigungs-System unabhängig vom App-Typ (Kategorie)	Ja	Nein	Ja	Um Benachrichtigungen zu erzeugen und anzuzeigen sind keine spezifischen Berechtigungen erforderlich. Es können Benachrichtigungen auf verschiedene Weisen erfolgen. WhatsCapp nutzt sowohl Benachrichtigungen über kleine Popups per Toasts, Alert Dialoge als auch Benachrichtigungen, die direkt in dem Benachrichtigungsbereich des Gerätes angezeigt werden. Empfange Nachrichten über den Google Cloud Messaging Dienst werden im Benachrichtigungsbereich angezeigt. Ein Beispiel für Alert Dialoge bei WhatsCapp wäre der Hinweis auf eine fehlende Internetverbindung, wohingegen ein Beispiel für einen Toast-Pop-Up der Hinweis auf einen fehlenden Kontakt-Standort wäre.
15	Hintergrundabläufe	Ja	Nein	Ja	Hintergrundabläufe werden bei der nativen Androidentwicklung durch Services realisiert. Ein Beispiel für einen Service in WhatsCapp ist der LocationUpdaterService, der den Standort des Benutzers kontinuierlich aktualisiert.
16	Multitouchscreen-Eingabe	Ja	Nein	Ja	Multitouchscreeneingaben müssen sowohl von der Hardware als auch der Software unterstützt werden. WhatsCapp nutzt GoogleMaps um Standorte anzuzeigen. Dabei ist das Ran- und Rauszoomen per Multitouch möglich.
17	Zugriff auf den Beschleunigungs-sensor	Ja	Nein	Nein	Einen Zugriff auf den Beschleunigungssensor ist über die Sensor-Klassen in Android möglich. Dafür werden keine besonderen Berechtigungen benötigt. Für die WhatsCapp App ist kein Einsatz des Sensors vorgesehen.

18	Möglichkeit andere Apps aufzurufen	Ja	Nein	Nein	Android Apps können andere Apps mit Intents indirekt aufrufen. Dazu sendet die erste App einen Intent per Broadcast, welcher daraufhin von einer zweiten App, empfangen wird. Dazu muss die zweite App einen entsprechenden Intent-Empfänger registriert haben. Sobald der entsprechende Intent empfangen wurde, wird die zweite Applikation automatisch ausgeführt. Die nativ entwickelte WhatsCapp App ruft keine anderen Apps auf.

Tabelle 11 Umsetzbarkeit der funktionalen Anforderungen mit Android und Einsatz in der nativ entwickelten Android App.

7.3 WhatsCapp mit PhoneGap und Sencha

7.3.1 App-Einblick

In Abbildung 34 sind Screenshots der mit PhoneGap und Sencha Touch entwickelten App zu sehen. Die Screenshots stammen von einem Galaxy Nexus mit der derzeit aktuellsten Betriebssystemversion Jelly Bean 4.2.2.
Wie im Kapitel 7.1 angekündigt wurde, wurde bei dieser App und der folgenden Titanium App, der Nearby-Tab optimiert. Da Sencha Touch stark an das Design der IPhone-Apps angelehnt ist, wurde beschlossen die Tab-Leiste an dem unteren Ende der sichtbaren Fläche zu platzieren statt oben. Die App besitzt nicht den vollen Funktionsumfang, den die nativ entwickelte Android-App hat.

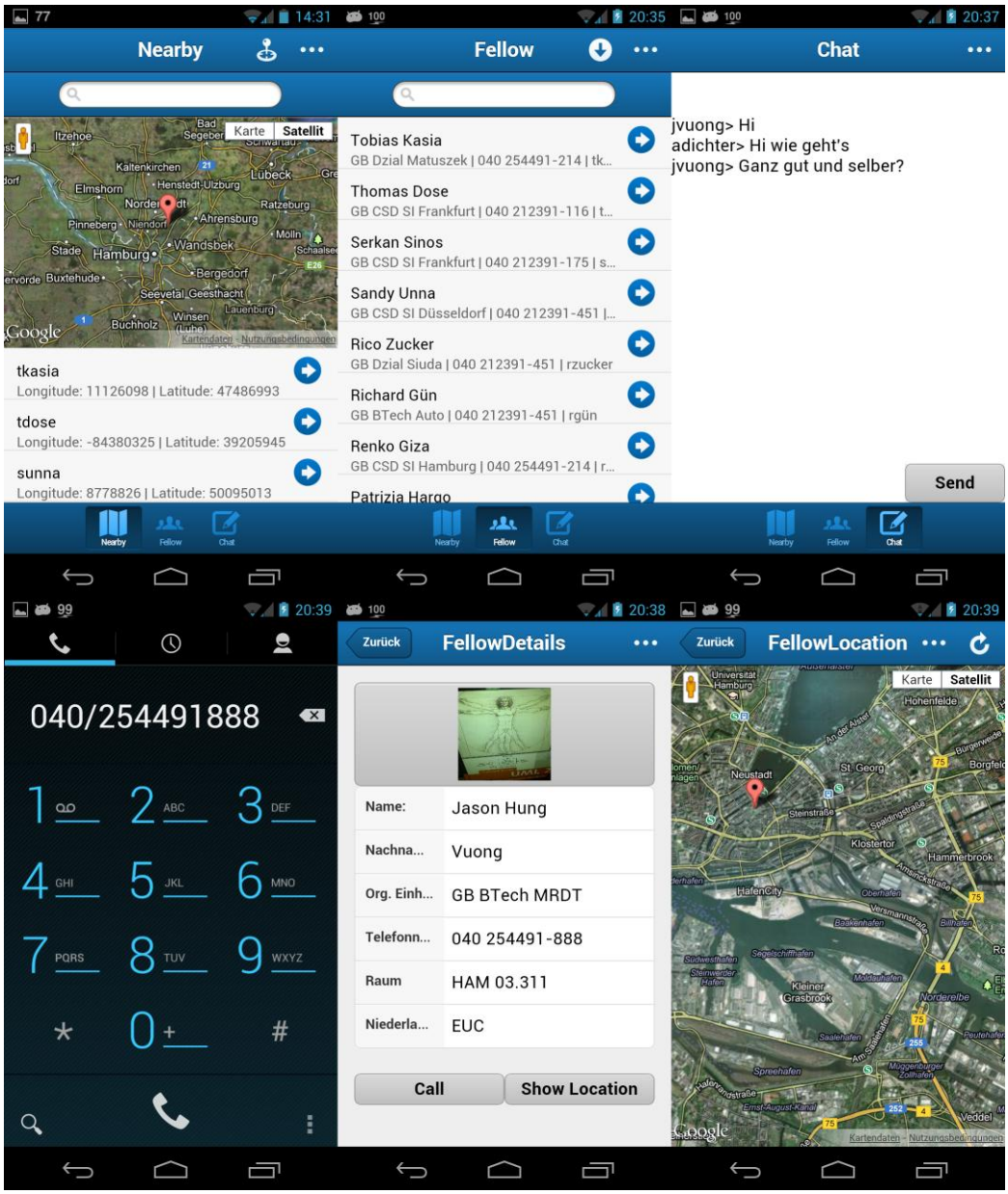

Abbildung 34 Screenshots aus der mit PhoneGap und Sencha Touch entwickelten App

7.3.2 Realisierung

Dieses Unterkapitel beschreibt die Realisierung bezüglich der formalen Organisation der App. Gemeint ist damit wie die Projekte strukturiert wurden und wie mit Ressourcen gearbeitet wurde.

7.3.2.1. **Projektstruktur**

In der Abbildung 35 sieht man wie JavaScript Dateien im WhatsCapp-Projekt mit PhoneGap und Sencha Touch strukturiert wurden. Models werden in Camelcase-Notation mit dem Postfix „Datamanagement“ versehen um die Zugehörigkeit zur Datenverwaltungskomponente zu verdeutlichen. Views bekommen dagegen den Postfix „Mask“ und Controller „Actionprocessing“. Dadurch ist am Namen erkennbar, welche Aufgabe das jeweilige Artefakt hat. Der Begriff Maske wird gleichbedeutend mit dem Begriff der visuellen Darstellung verwendet. Stores, die auch ein Bestandteil der Datenverwaltung sind werden mit dem einfachen Postfix „s“ versehen, da es bei Sencha Touch der Standard ist.

Betrachtet man die View „FellowlistMask“ ist erkennbar, dass nicht durchgehend eine Camelcase-Notation verwendet wurde. Dies soll verdeutlichen, dass der Dialogteil „List“ zu der übergeordneten Maske „FellowMask“ gehört. Die Fellowlist-Maske ist in FellowMask integriert. Da die „Fellowlist“ keine eigenständige Logik besitzt, ist sie ein fester Bestandteil der Fellow-Maske. Man spricht in so einem Fall daher nicht von einer visuellen Einbettung wie es im Kapitel 2.4.5.3 beschrieben wird.

Ein Beispiel für eine visuelle Einbettung findet man in dem Main Dialog. Der Main Dialog bettet die eigenständigen Dialoge Nearby, Fellow und Chat ein. Jeder dieser Dialoge ist unabhängig von den anderen und besitzt seine eigene Logik.

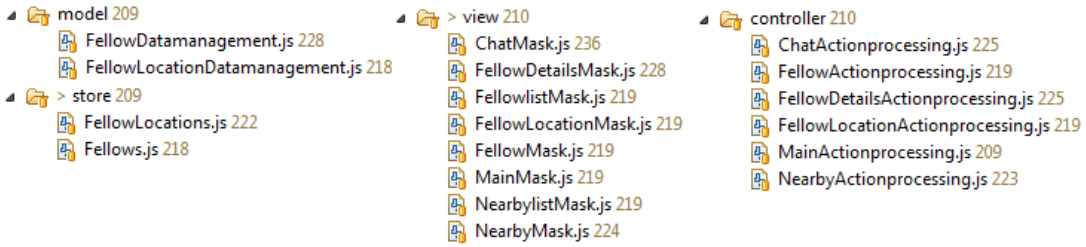

Abbildung 35 Organisation der JavaScript Dateien im Projekt WhatsCappPhoneGap-Projekt

7.3.2.2. **Ressourcen**

Ressourcen sind kaum benötigt worden. Bis auf die Kontaktdaten, die Lokationsdaten, dem Standard-Avatarbild und dem default Stylesheet wurden keine weiteren Ressourcen benötigt.

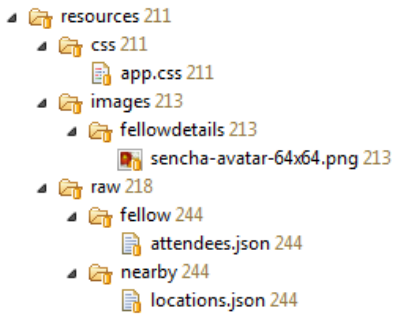

Abbildung 36 Ressourcen des WhatsCappPhoneGap-Projekts

7.3.3 Architekturumsetzung

Dieses Unterkapitel geht näher auf die Umsetzung der in Kapitel 6.4.2 beschriebenen Dialogarchitektur ein und zeigt anhand vom Beispielcode aus der WhatsCappPhoneGap-App, wie die einzelnen Bestandteile der Präsentation und des Dialogkerns umgesetzt wurden.

7.3.3.1. **Präsentation**

Datenanbindung

Das WhatsCappPhoneGap-Projekt nutzt SenchaTouch Stores zur Datenverwaltung. Ein Beispiel einer Datenanbindung wie sie im Kapitel 6.4.2.2 beschrieben wurde ist im Code-Beispiel 1 und Code-Beispiel 3 zu sehen.

```
'</pre><div class="list-item-title">{name} {lastname}</div><div class="list-item-
narrative">{orgunit} | {phone} | {id}</div><pre>'
```

Code-Beispiel 1 Datenanbindung von Feldern per HTML am Beispiel der Fellowlist-View

Im Code-Beispiel 1 ist die HTML Definition, mit der Anbindung diverser Felder, für die Listenansicht der Kontakte zu sehen. Das Referenzieren geschieht hier durch die im Model (Code-Beispiel 7) definierten Attributnamen. Die Aktion des Datenladens wird dabei in der „init“ Methode des Follow-Controllers verarbeitet, wie es im Code-Beispiel 9 zu sehen ist.
Die Fellowlist-View wird in der Fellow-View als Liste angezeigt. Beim Klick auf einen Listeneintrag wird ein Benutzerereignis gefeuert und dadurch eine Aktion im Fellow-Controller ausgelöst, welche daraufhin die Daten des ausgewählten Datensatzes in der FellowDetails-View setzt. Das Setzten des Datensatzes ist im Code-Beispiel 2 zu sehen.

```
activateFellowDetailsMaskCard: function (record) {
    var fellowDetailsMask = this.getFellowDetailsMaskCard();
    fellowDetailsMask.setRecord(record);
    Ext.Viewport.animateActiveItem(fellowDetailsMask, this.slideLeftTransition);
}
```

Code-Beispiel 2 Setzen eines einzelnen Datensatzes für die FellowDeatils-View

Durch das Setzen des Datensatzes werden die UI-Elementen automatisch mit Werten beladen, sofern die entsprechenden Felder als Namen gesetzt wurden.

```
var fellowDetailsNameField = {
        xtype: 'textfield',
        label: 'Name:',
        name: 'name',
        readOnly: true
    };
```

Code-Beispiel 3 Anbindung des Namenfelds an das Textfeld mit dem Namen „Name“

Das Code-Beispiel 3 stammt aus der FellowDetails-View und zeigt einer der UI-Elemente. Durch den Attributnamen „name“ kann dieses Textfeld wie andere auch automatisch gesetzt werden.

Aktionsanbindung

Wie die Aktionsanbindung durchgeführt werden kann wurde im 6. Kapitel beschrieben. Im Folgenden wird am Beispiel des Call-Buttons der FellowDetails-View exemplarisch gezeigt wie die Aktionsanbindungen realisiert worden sind.

Die Erstellung des Call-Buttons des WhatsCappPhoneGap-Projektes in der FellowDetails-View sieht folgendermaßen aus:

```
var fellowDetailsCallButton = {
        xtype: 'button',
        text: 'Call',
        handler: this.onFellowDetailsCallFellowButtonTap,
        scope: this
};
```

Code-Beispiel 4 Erzeugung eines Button mit Handler am Beispiel des Call-Buttons

Es wird beim Drücken des Buttons der lokale Handler `onFellowDetailsCallFellowButtonTap` ausgeführt. Dabei handelt es sich um eine lokale Methode, die Aufgaben verarbeiten kann, welche ausschließlich die Präsentationsschicht betreffen. Es können zwar auch andere Aufgaben verarbeitet werden, aber dies entspricht nicht dem Quasar-Gedanken.

```
onFellowDetailsCallFellowButtonTap : function(){
    this.fireEvent('callFellowPhonenumberCommand', this.getRecord().get('phone'));
},
```

Code-Beispiel 5 Feuern eines Events am Beispielevent „callFellowPhonenumberCommand“

Stattdessen wird ein Event gefeuert, welches vom entsprechenden Controller abgefangen werden kann um eine entsprechende Aktion auszulösen. Die Aktionsanbindung ist somit in der WhatsCapp App durch den Austausch von Events aus View an den Controller realisiert. Statt dem Feuern eines Events wäre auch ein direkter Aufruf einer Controller Methode möglich. Die Nutzung von Events sind in der Community ein häufig empfohlener Ansatz, da der Controller stärker von der View entkoppelt wird und die View dadurch auch übersichtlicher wird.

Präsentationszustandssteuerung
Im WhatsCappPhoneGap-Projekt werden kaum Präsentationszustände gesteuert. Ein Beispiel wo Eigenschaftsvariablen genutzt werden ist in der FellowDetails-View.
Im Code-Beispiel 6 wird ein Textfeld erzeugt, welches über die Konfigurationsvariable „readOnly“ zum Editieren gesperrt wird. Es wäre denkbar bei Erweiterungen das Editieren von Kontaktdaten zu ermöglichen. In so einem Fall könnte zum Beispiel ein Button-Klick das Textfeld wieder freigeben und nach erfolgreicher Speicherung wieder sperren.

```
var fellowDetailsNameField = {
        xtype: 'textfield',
        label: 'Name:',
        name: 'name',
        readOnly: true
    };
```

Code-Beispiel 6 Namen-Textfeld mit Schreibschutz

7.3.3.2. Dialogkern

Datenhaltung
Die Datenhaltung der Dialogkomponenten bei dem WhatsCappPhoneGap-Projekt setzt sich jeweils aus einem Model und einem Store zusammen. Das Model spezifiziert die Struktur der Datenobjekte und der Store stellt die Daten bereit. Dies wird im Folgenden am Beispiel der „Fellows“ gezeigt.

Das Model „FellowDatamanagement“:

```
Ext.define('WhatsCappX.model.FellowDatamanagement', {
    extend: 'Ext.data.Model',
    config: {
        idProperty: 'id',
        fields: [
            { name: 'id', type: 'string' },
            { name: 'lastname', type: 'string' },
            { name: 'name', type: 'string' },
            { name: 'room', type: 'string' },
            { name: 'fax', type: 'string' },
            { name: 'phone', type: 'string' },
            { name: 'orgunit', type: 'string' },
            { name: 'branchoffice', type: 'string' },
            { name: 'email', type: 'string' },
            { name: 'additionaloffice', type: 'string' },
```

```
            { name: 'phoneadditional', type: 'string' },
            { name: 'mobile', type: 'string' },
            { name: 'title', type: 'string' },
            { name: 'rank', type: 'string' },
        ]
    }
});
```

Code-Beispiel 7 Model Konfiguration am Beispiel FellowDatamanagement

In dem Model werden die Felder spezifiziert und die „id“ als eindeutiger Bezeichner festgelegt.

Der Fellow-Store “Fellows”:

```
Ext.define('WhatsCappX.store.Fellows', {
    extend: 'Ext.data.JsonStore',
    config: {
        model: 'WhatsCappX.model.FellowDatamanagement',
        proxy: {
                type: 'ajax',
                url: 'app/store/attendees.json',
                reader : {
                        type : 'json',
                        rootProperty : 'fellows'
                }
        },
        sorters: [{property: 'name', direction: 'DESC'}]
    }
});
```

Code-Beispiel 8 Sencha Touch Store am Beispiel des Fellow-Store „Fellows“

Die Initialisierungsfunktion „init“, die vor Applikationsstart geladen wird lädt die Storedaten aus dem Store mit der Methode:

```
Ext.getStore('Fellows').load();
```

Code-Beispiel 9 Laden der Daten aus einem Store am Beispiel vom Fellows-Store

Da keine Daten gelöscht oder geändert werden die für die Anzeige der Liste relevant sind, ist auch kein Update der Liste notwendig.

Aktionsverarbeitung

Aktionen werden in dem jeweiligen Controller des Dialoges verarbeitet. In Kapitel 7.3.3.1 wurde beschrieben wie die Benutzerereignisse an den Dialogkern angebunden sind. In dieser App geschieht dies über Events. Diese Events können im Controller gefangen werden. Dazu muss in den Konfigurationen des Controllers zuerst eine Referenz auf die View eingetragen werden. Dies geschieht über sogenannte „xtypes“, die man für View oder View-Elemente spezifizieren kann. In diesem Beispiel lautet der xtype der FellowDetails-Maske „fellowdetailsmaskcard“.

```
    config: {
        refs: {
            fellowDetailsMaskCard : 'fellowdetailsmaskcard'
        },
        control: {
        fellowDetailsMaskCard:{
                callFellowPhonenumberCommand: 'onCallFellowPhonenumberCommand'
            }
        }
    },
```

Code-Beispiel 10 Konfigurationen im Controller zum Abfangen von Events zur Aktionsverarbeitung am Beispiel von FellowDetails

Um ein Event abzufangen muss eine Zuordnung von empfangenen Events mit den Aktionsverarbeitenden Methoden erfolgen. Diese Zuordnung definiert man, wie im Code-Beispiel 10 zu sehen ist, unter den Controls.

```
onCallFellowPhonenumberCommand: function (phonenumber) {
        callNumber(phonenumber);
},
```

Code-Beispiel 11 Aktionsverarbeitung am Beispiel des Anrufs

Im Code-Beispiel 11 ist die zugeordnete Methode zu sehen, die für die Aktionsverarbeitung verantwortlich ist. Sie führt dazu, dass die Telefonnummer des Kontaktes in die Wählmaske übernommen wird. Die Wählmaske ist in Abbildung 34 links unten zu sehen.

Dialogzustandsverwaltung

Wie im Kapitel 6.4.2.1 beschreiben ist wird keine explizite Dialogzustandsverwaltung direkt unterstützt. Es ist auch im Rahmen der aktuellen App nicht nötig. Daher wurden auch nicht implizit Zustände verwaltet.

7.3.4 Anforderungsumsetzbarkeitsanalyse

Die aufgestellten Anforderungen aus Kapitel 4.3.3 konnten bis auf eine, in Gänze umgesetzt werden. Dabei konnten drei der Anforderungen und zwei weitere teilweise nur durch zusätzliche Plugins umgesetzt werden, die dazu auch noch plattformspezifisch sind. Die detaillierten Beschreibungen sind der Tabelle 12 zu entnehmen.

An-forderungs-ID	Berechtigungen / Anforderungen	Um-setzbar?	Zusätze be-nötigt?	Von Whats CappPG genutzt?	Beschreibung
1	Uneingeschränkter Internetzugriff	Ja	Nein	Ja	Der Zugriff auf das Internet ist sowohl mit PhoneGap und Sencha Touch mit dem Setzen der Berechtigung INTERNET ohne weiteres möglich.

2	Inhalt des USB-Speichers und der SD-Karte ändern/löschen	Ja	Nein	Ja	Bestimmte Hardwarezugriffe über PhoneGap wie zum Beispiel die Nutzung der Kamera benötigen Rechte zum Speicherzugriff auf den externen Speicher zur Speicherung von Bildern. Die APIs File und Kamera stellen nähere Informationen bereit. Das setzen der entsprechenden Rechte ist vorausgesetzt.
3	Netzwerkstatus anzeigen	Ja	Nein	Nein	Über die Connection API ist sowohl über PhoneGap als auch SenchaTouch der Netzwerkstatus prüfbar. Das funktioniert über PhoneGap navigator.connection.type oder über Senchas Connection Klasse per Ext.device.Connection.isOnline(). Neben ACCESS_NETWORK_STATE benötigt PhoneGap noch die READ_PHONE_STATE Berechtigung.
4	Vibrationsalarm steuern	Ja	Nein	Ja	Sowohl die PhoneGap als auch Sencha Touch können über die Notification API Vibrationen auslösen.
5	Ungefährer (netzwerkbasierter) Standort	Ja	Nein	Ja	Auch der aktuelle Standort kann über die PhoneGap oder die Sencha Touch API erfolgen. Unter der API Geolocation sind auf den API Seiten detaillierte Informationen
6	Genauer (GPS-) Standort	Ja	Nein	Ja	Hier gilt das gleiche wie bei der Anforderung 5
7	Automatisch nach dem Booten starten	Ja	Ja	Nein	Die PhoneGap API bietet in der aktuellsten Version 2.5.0 keine Möglichkeit automatisch eine App nach dem Booten zu starten. Es gibt allerdings über Github.com von PhoneGap bereitgestellte Plugins für diverse Zwecke. "LocalNotification" nennt sich einer der Plugins mit dem diese Anforderung umsetzbar ist. Dieses Plugin ist androidspezifisch.
8	Bekannte Konten suchen	Ja	Nein	Nein	Die Contacs API von PhoneGap ermöglicht den Zugriff auf bekannte Konten um so Kontaktdaten vom Gerät zu lesen und zu ändern. Auch Sencha Touch bietet eine Contacts API an und kann somit das gleiche. Diese Anforderung geht über die Kontaktsuche hinaus und zielt auch auf andere Konten des Geräts ab wie beispielsweise das Google Konto zur Nutzung des Google Cloud Messaging Services. Für die spezifischen Anwendundungsfälle gibt es offizielle und inoffizielle Lösungen von Privatanbietern.
9	Telefonstatus lesen und identifizieren	Ja	(Ja)	Nein	Um den Telefonstatus zu lesen und somit beispielsweise die Geräte ID zu lesen, bietet die PhoneGap die API "Device". Sencha Touch bietet eine gleiche API. Um festzustellen ob ein Telefonat geführt wird, bietet PhoneGap ein androidspezifisches Plugin mit dem Namen "PhoneListener" an.

10	WLAN-Status anzeigen	Nein	Nein	Nein	Die Verbindung zu prüfen wird über die Anforderung 3 geregelt und ist möglich. Um die Funktionalitäten die sich hinter "WLAN-Staus" anzeigen befinden zu realisieren wurde keine Lösung gefunden. Es kann also über PhoneGap oder Sencha Touch beispielsweise die WLAN Verbindung nicht aktiviert bzw. deaktiviert werden. Eigene Entwicklungen wären denkbar.
11	Standby-Modus des Tablets deaktivieren Standby-Modus des Telefons deaktivieren	Ja	Ja	Nein	Apps ohne Benutzerintaktionen laufen zu lassen kann weder die PhoneGap in der aktuellen Version noch Sencha. In den bereits in Anforderung 7 erwähnten Plugin-Verzeichnis von PhoneGap unter Github.com gibt es ein Plugin, welches sich PowerManagement nennt und dies ermöglicht. Dieses Plugin ist androidspezifisch. Es gibt aber andere Plattformen mit ähnlichen Plugins.
12	Bilder und Videos aufnehmen	Ja	Nein	Ja	Beide APIs (Sencha Touch und PhoneGap) bieten die Aufnahme von Fotos an.
13	Kontaktdaten lesen	Ja	Nein	Nein	Die Contacts API von PhoneGap ermöglicht den Zugriff Kontaktdaten. Auch Sencha Touch bietet eine Contacts API an und kann somit das gleiche.
14	Zugriff auf das Benachrichtigungs-System unabhängig vom App-Typ (Kategorie)	Ja	Ja / Nein	Nein	PhoneGap bietet über die bereits in Anforderung 4 erwähnte Notification API die Möglichkeiten Benachrichtigungen bemerkbar zu machen. Dazu zählt neben der Vibration auch das Anzeigen von Alert Dialogen und einige weitere Benachrichtigungsmöglichkeiten. Sencha Touch setzt die Alert Dialoge mit ihren MessageBoxen um. Benachrichtigungen in der Staus-Leiste sind mit dem zusätzlichen Plugin "StatusBarNotification" von PhoneGap möglich. Dieses Plugin ist androidspezifisch. Es gibt aber andere Plattformen ähnlichen Plugins.
15	Hintergrundabläufe	Ja	Ja	Nein	Das Problem bei JavaScript ist, dass JavaScript nicht im Hintergrund ausgeführt werden. Es gibt plattformspezifische Lösungen dafür. PhoneGap bietet auf Github.com ein Plugin mit dem Namen "BackgroundService" an und stellt dazu androidspezifische Java Klassen bereit.
16	Multitouchscreen-Eingabe	Ja	Nein	Ja	Multitouchscreeneingaben müssen sowohl von der Hardware als auch der Software unterstützt werden. WhatsCappPG nutzt GoogleMaps um Standorte anzuzeigen. Dabei ist das Ran- und Rauszoomen per Multitouch möglich.
17	Zugriff auf den Beschleunigungs-sensor	Ja	Nein	Nein	Einen Zugriff auf den Beschleunigungssensor ist über die PhoneGap API "Accelerometer" möglich.

18	Möglichkeit andere Apps aufzurufen	(Ja)	Nein	Nein	Eine einfache Möglichkeit direkt andere Apps aufzurufen gibt es offiziell nicht von PhoneGap oder SenchaTouch. Man müsste ein Plugin schreiben. Eine indirekte Möglichkeit wäre es über URLs die Ziel-App aufzurufen. YouTube ist einer der Beispiele bei dem es funktioniert.

Tabelle 12 Umsetzbarkeit der funktionalen Anforderungen mit PhoneGap und Sencha Touch

7.4 WhatsCapp mit Appcelerator (Titanium Alloy)

7.4.1 App-Einblick

Die Abbildung 37 zeigt Screenshots der mit Titanium entwickelten App. Auch diese Screenshots stammen von einem Galaxy Nexus Testgerät mit dem derzeit aktuellsten Betriebssystemversion Jelly Bean 4.2.2.
Die Entwicklung dieser App fand in der abschließenden Implementierungsphase statt, so dass die Funktionalitäten, aus zeitlichen Gründen, nicht den gleichen Umfang wie die des WhatsCappPhongeGap-Projektes erreichen konnten. Es konnte aber trotz allem ein guter Eindruck für die App-Entwicklung mit Titanium und dem Framework Alloy gewonnen werden, worum es auch primär ging.
Wie in der PhoneGap-App ist die Chat-View nur gemockt. Funktionalitäten wie die Nutzung von GPS, die Aufnahme von Bildern oder die Anruf Funktion wurden größtenteils umgesetzt. Der Split-Screen der Nearby-View konnte nicht auf Anhieb umgesetzt werden, so dass eine alternative Umsetzung durch eine separate Table-View mit den Lokationen implementiert wurde. Diese Table-View kann per Wisch-Bewegung nach links aufgerufen werden. Da durch die Wisch-Bewegung später auch ein Tab-Wechsel stattfinden soll, wurde diese Alternative zunächst aus der App entfernt. In der Abbildung 37 ist die Kamera-Ansicht rechts unten zu sehen. Sie sieht in allen drei Apps gleich aus, da immer die gleiche native Kamera genutzt wird.

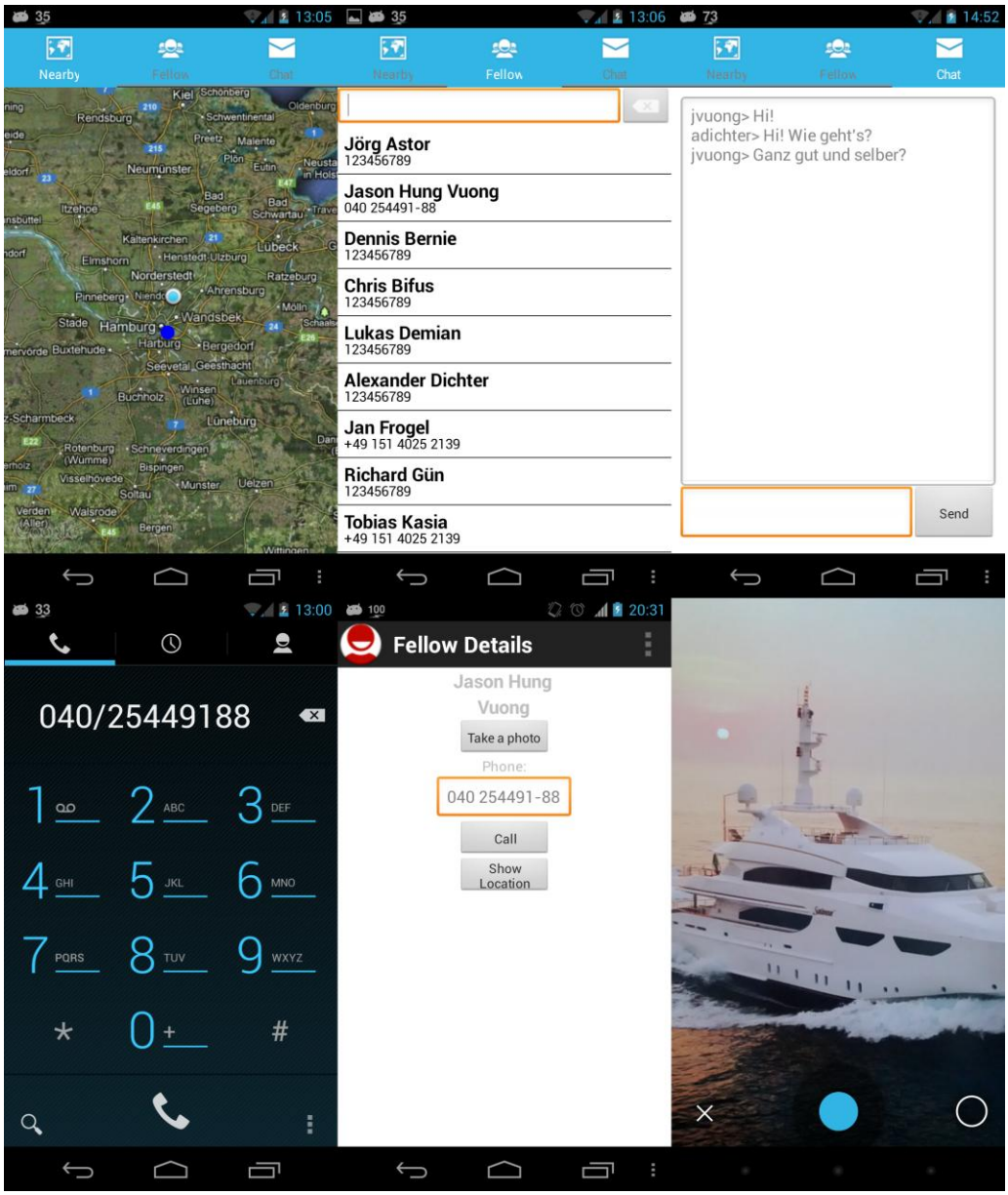

Abbildung 37 Screenshots aus der mit Titanium entwickelten App

7.4.2 Realisierung

Dieses Unterkapitel beschreibt die Realisierung bezüglich der formalen Organisation der Titanium App. Es soll die Strukturierung der App und die Verwaltung der Ressourcen vermittelt werden.

7.4.2.1. **Projektstruktur**

Da die Projektstruktur mit Alloy, wie in Kapitel 0 beschrieben, etwas unflexibel ist heißen die Dateien immer gleich. Es gibt am Beispiel der Kontakte in Abbildung 38 ein fellow-Model, View, Style und Controller mit dem gleichen Namen. Bis auf das Model und den Controller unterscheiden sich alle im Datentyp. Dadurch sind die Dateien optisch besser auseinanderzuhalten.

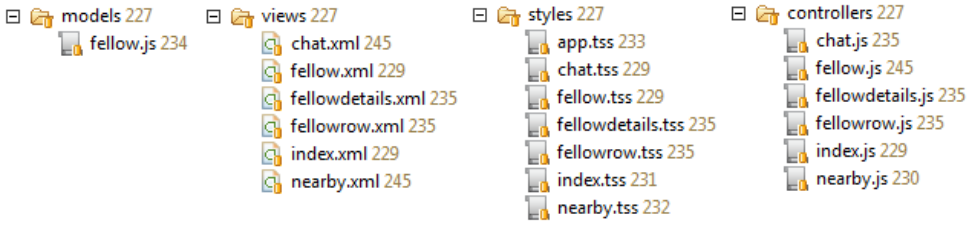

Abbildung 38 Dateiorganisation des WhatsCappTitanium-Projektes

7.4.2.2. **Ressourcen**

In der aktuellen Version des WhatsCappTitanium-Projektes wurden Ressourcen in einem zusätzlichen „common“-Ordner organisiert. Dieser befindet sich in dem „assets“-Ordner auf der gleichen Ebene wie die übrigen plattformspezifischen Ordner, die in Kapitel 0 beschrieben wurden.

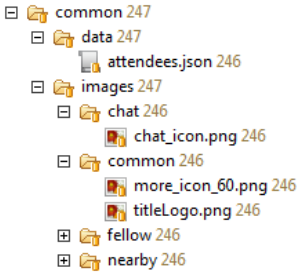

Abbildung 39 Organisation der Ressourcen im WhatsCappTitanium-Projekt

7.4.3 Architekturumsetzung

Dieses Unterkapitel geht näher auf die Umsetzung der in Kapitel 0 beschriebenen Dialogarchitektur ein und zeigt anhand von Beispielcode aus der App wie mit Titanium und dem Alloy Framework einzelne Bestandteile der Präsentation und des Dialogkerns umgesetzt wurden.

7.4.3.1. **Präsentation**

Datenanbindung
Für die Datenanbindung ist bei der WhatsCappTitanium-App die Nutzung des Model-View Binding angedacht. Zurzeit werden Daten aus einer lokal definierten Collection geladen. Daher ist hier stattdessen die geplante Umsetzung skizziert:

```
<Alloy>
        <Tab title="Fellowtab" icon="/common/images/fellow/fellow_icon.png">
                <Window title="Fellow">
                        <TableView dataCollection="fellow">
                                <TableViewRow title="{name}" />
                        </TableView>
                </Window>
        </Tab>
</Alloy>
```

Code-Beispiel 12 Datananbindung über das Alloy Model View Binding der Fellows

Das Code-Beispiel 12 zeigt die zukünftige Fellow-View, welche die Tabelle an das Model „fellow“ bindet und den Namen aus dem jeweiligen Datensatz an eine Tabellenzeile bindet. Der Controller sorgt mit der Methode fetch() im Code-Beispiel 13 …dafür, dass die Daten beim Aufruf des Tabs geladen werden. Bei der Methode fetch() handelt es sich um eine Backbone Methode, die dem Read der CRUD Operationen bzw. einem SELECT eines SQL Statements entspricht.

```
var contactbook = Alloy.Collections.fellow;
contactbook.fetch();
```

Code-Beispiel 13 Lesezugriff auf die Fellow-Daten der Collection Fellow

Aktionsanbindung
Die Aktionsanbindung ist bei dem WhatsCappTitanium-Projekt durch Klickevents realisiert. Dies wird am Beispiel des Call Buttons gezeigt. In dem Code-Beispiel 14 ist der Aufbau einer Alloy View ersichtlich, wo auch die Definition des Buttons dargestellt ist. Unnötige Code-Teile wurden im Beispiel durch Kommentare ersetzt.

Alloy Views beginnen immer mit einem Alloy-Tag. In dem Alloy-Tag ist ein Fenster enthalten, in der eine View definiert ist, die den Button enthält. Ein Button-Klick ruft die Methode „callFellow“ im FellowDetails-Controller auf. Somit ist die Aktionsanbindung durch das Klickevent gesteuert und realisiert.

```
<Alloy>
	<Window id="fellowdetails">
		<!--some code-->
		<View id="fellowdetailscontent">
			<!--some code-->
			<Button id="callfellow" onClick="callFellow">Call</Button>
			<!--some code-->
		</View>
	</Window>
</Alloy>
```

Code-Beispiel 14 Button Definition in der View von FellowDetails in dem WhatsCappTitanium-Projekts

Präsentationszustandssteuerung
Analog zum WhatsCappPhoneGap-Projekt werden auch hier kaum Präsentationszustände gesteuert. Auch hier können in der FellowDetails-View Zustände über Eigenschaftsvariablen gesetzt werden. Hier heißen sie „Properties“ statt „Configurations“.

Im Code-Beispiel 15 wird ein Textfeld erzeugt, welches über die Eigenschaftskonfiguration „editable“ zum Editieren gesperrt wird. Auch hier könnte man bei Erweiterungen die Zustandseigenschaften beim Eintreten verschiedener Ereignisse ändern.

```
<TextField id="fellowdetailsphone" editable=false/>
```

Code-Beispiel 15 Telefonnummern-Textfeld mit Schreibschutz

7.4.3.2. Dialogkern

Datenhaltung
Daten werden in der aktuellen App-Version aus einer lokal definierten Collection geladen. Die Datenhaltung wird für weitere Schritte geplant und im Folgenden skizziert:

```
exports.definition = {
	config: {
		"columns": {
			"id":"text",
			"lastname":"text",
			"name":"text",
			"room":"text",
			"fax":"text",
			"phone":"text",
			"orgunit":"text",
			"branchoffice":"text",
			"additionaloffice":"text",
			"phoneadditional":"text",
			"mobile":"text",
			"title":"text",
			"rank":"text"
		},
		"adapter": {
			"type": "sql",
			"collection_name": "fellow"
```

```
                }
        }
}
```

Code-Beispiel 16 Fellow Model im WhatsCappTitanium-Projekt

Das Code-Beispiel 16 zeigt das Fellow-Model in dem die Spalten und der SQL-Adapter mit dem Namen des Models definiert sind. Dieser Name kann daraufhin in TabellenViews als „dataCollection“ referenziert werden.

Aktionsverarbeitung

Im Kapitel 0 wurde im Unterabschnitt „Aktionsabindung“ das Beispiel des Call-Buttons vorgestellt. Die daraufhin ausgeführte Funktion ist im Code-Beispiel 17 zu sehen. Auch hier wird, wie bei Sencha Touch, die Wählmaske durch das Auslösen eines Telefonlinks aufgerufen. Der technische Aufruf des Telefonlinks ist dabei in die separate Methode „callNumber“ entkapselt.

```
function callFellow() {
        callNumber(phonenumber);
};
```

Code-Beispiel 17 Aktionsverarbeitung des Klick-Ereignis vom Call-Button

Dialogzustandsverwaltung

Wie im Kapitel 6.5.2.1 beschrieben ist, wird keine explizite Dialogzustandsverwaltung unterstützt. Es gilt hier was auch für die Dialogzustandsverwaltung von Sencha gilt.

7.4.4 Anforderungsumsetzbarkeitsanalyse

Auch bei Titanium konnten, mit Ausnahme von zwei Anforderungen, alle aufgestellten Anforderungen aus Kapitel 4.3.3 umgesetzt werden. Statt zusätzlicher plattformspezifischer Plugins, bietet Titanium teilweise integrierte native API’s um „Anforderungslücken“ zu schließen. Die detaillierteren Informationen bzgl. der einzelnen Anforderungen sind der Tabelle 13 zu entnehmen.

An-forderungs-ID	Berechtigungen / Anforderungen	Um-setzbar?	Zusätze be-nötigt?	Von Whats CappTI genutzt?	Beschreibung
1	Uneingeschränkter Internetzugriff	Ja	Nein	Ja	Der Zugriff auf Internet ist mit Titanium problemlos möglich. Es sind keine manuellen Konfigurationen zur Anforderung der Berechtigung notwendig, wie beispielsweise bei PhoneGap/Cordova. Es ist zwar nicht nötig, aber auf Wunsch können auch manuell Berechtigungen gesetzt werden.

2	Inhalt des USB-Speichers und der SD-Karte ändern/löschen	Ja	Nein	Ja	Den Zugriff auf den externen Speicher ist über die Titanium.Filesystem API möglich. Des Weiteren kann beispielsweise bei der Aufnahme von Bildern das Speichern durch Eigenschaftsvariablen gesteuert werden (Beispiel: saveToPhotoGallery: true).
3	Netzwerkstatus anzeigen	Ja	Nein	Ja	Die Titanium.Network API bietet verschiedene Methoden an um beispielsweise den Online Status des Gerätes abzufragen aber auch Anforderung 9 wird hierdurch erfüllt.
4	Vibrationsalarm steuern	Ja	Nein	Ja	Die Titanium.Media API bietet unter Anderem eine Methode an um Vibrationen zu steuern.
5	Ungefährer (netzwerkbasierter) Standort	Ja	Nein	Ja	Auch der aktuelle Standort kann über die Titanium API erfolgen. Zu diesem Zweck steht die Titanium.Gelocation API bereit.
6	Genauer (GPS-) Standort	Ja	Nein	Ja	Hier gilt das gleiche wie bei der Anforderung 5
7	Automatisch nach dem Booten starten	Nein	Nein	Nein	Die Titanium API bietet keine Möglichkeit an, auf das Booten des Gerätes zu reagieren und somit nach dem Booten die App automatisch zu starten. Zwar bietet Titanium im Fall von Android eine Anbindung an Android Elemente wie Activities an aber Broadcastreceiver werden nicht unterstützt, die Boot Events empfangen würden.
8	Bekannte Konten suchen	Ja	(Ja)	Nein	Die Titanium.Contacs API ermöglicht den Zugriff auf bekannte Konten um so Kontaktdaten vom Gerät zu lesen und zu ändern. Diese Anforderung geht über die Kontaktsuche hinaus und zielt auch auf andere Konten des Geräts ab wie beispielsweise das Google Konto zur Nutzung des Google Cloud Messaging Services. Für die spezifischen Anwendungsfälle gibt es einzelne Lösungen von Privatanbietern.
9	Telefonstatus lesen und identifizieren	(Ja)	Nein	Nein	Wie in der Anforderung 3 beschrieben bietet die Titanium.Network API auch Methoden zur Ermittlung von Geräteeigenschaften wie zum Beispiel die ID des Gerätes abzufragen. Diese Möglichkeit funktioniert jedoch nur für iOS Geräte.
10	WLAN-Status anzeigen	Nein	Nein	Nein	Die Verbindung zu prüfen wird über die Anforderung 3 geregelt und ist möglich. Um die Funktionalitäten, die sich hinter "WLAN-Staus anzeigen" befinden zu realisieren, bietet Titanium keine Lösung an. Es ist demnach beispielsweise nicht möglich die WLAN Verbindung zu aktivieren bzw. zu deaktivieren.
11	Standby-Modus des Tablets deaktivieren Standby-Modus des Telefons deaktivieren	(Ja)	Nein	Nein	Titanium bietet keine Möglichkeit eine App vor dem Standby Modus zu bewahren. Für die iOS Geräte bietet die Titanium.App API eine Möglichkeit den Leerlauftimer (Idle-Timer) zu deaktivieren, wodurch die Bildschirmsperre deaktiviert werden kann.
12	Bilder und Videos aufnehmen	Ja	Nein	Ja	Die Titanium.Media API bietet einen Zugriff auf die native Kamera des Gerätes.

13	Kontaktdaten lesen	Ja	Nein	Nein	Der Zugriff auf Kontaktdaten wird über die Titanium.Contacts API angeboten.
14	Zugriff auf das Benachrichtigungs-System unabhängig vom App-Typ (Kategorie)	Ja	Nein	Ja	Titanium bietet über die Titanium.UI.AlertDialog API an Alert-Dialoge zu erstellen. Darüber hinaus kann plattformspezifisch im Fall von Android über die Titanium.Android.NofificationManager API auf die Notification zugegriffen werden, um Nachrichten in der Statusbar anzuzeigen. Für iOS bietet Titanium vergleichbares über die Titanium.App.iOS API.
15	Hintergrundabläufe	Ja	Nein	Nein	Titanium bietet für Hintergrundprozesse nur plattformspezifische Lösungen an. Die APIs Titanium.App.iOS.BackgroundService und Titanium.Android:Service bieten für iOS und Android die spezifischen Lösungen.
16	Multitouchscreen-Eingabe	Ja	Nein	Ja	Multitouchscreeneingaben müssen sowohl von der Hardware als auch der Software unterstützt werden. WhatsCappTI nutzt GoogleMaps um Standorte anzuzeigen. Dabei ist das Ran- und Rauszoomen per Multitouch möglich.
17	Zugriff auf den Beschleunigungs-sensor	Ja	Nein	Nein	Einen Zugriff auf den Beschleunigungssensor ist über die Titanium.Accelerometer API möglich.
18	Möglichkeit andere Apps aufzurufen	Ja	Nein	Nein	Titanium kann durch die Titanium.Android.Intent API androidspezifisch andere Apps aufrufen. Die indirekte Möglichkeit über URLs , die bei Sencha Touch und PhoneGap erwähnt wurde, funktioniert für iOS mit der Titanium.Platform API.

Tabelle 13 Umsetzbarkeit der funktionalen Anforderungen mit Appcelerator (Titanium)

8 Vergleichsanalysen und Bewertung

Dieses Kapitel beschreibt zum Einen in Abschnitt 8.1 durchgeführte und theoretisch betrachtete Vergleichsanalysen und zum Anderen in Abschnitt 8.2 Bewertungen zu den gewählten CPTs bezüglich verschiedener Kriterien und Erfahrungen. Die Erfahrungen basieren dabei auf den in dieser Arbeit gesammelten Erfahrungen.

8.1 Vergleichsanalysen

8.1.1 Paketgrößen

In der Tabelle 14 sind, am Beispiel der Android Plattform, die aktuellen App-Versionen bzgl. ihrer Größe im Vergleich dargestellt. Hierbei wird zum einen die Größe der “Android Application Package”-Datei (.akp-Datei) und zum anderen der eingenommene Speicher nach der Installation auf dem Gerät betrachtet. Die Größenangaben sind in Megabyte (MB).

Die .apk-Datei ist die Datei, die an den Endkunden ausgeliefert wird. Die Größe der Installationsdatei spielt dadurch eine entscheidende Rolle, da sie vom Endkunden zur Installation heruntergeladen werden muss. Sie ist dann von größerem Interesse, wenn die App unter der Nutzung einer mobilen Internetverbindung heruntergeladen wird, da dies zusätzliche Kosten für den Endkunden verursachen kann. Die Größe nach der Installation ist ebenfalls wichtig für Benutzer, die ihren Gerätespeicher nicht erweitern können.

Die .apk-Datei mit PhoneGap und Sencha Touch App (ID 2) ist mehr als sechs Mal so groß wie die .apk-Datei der nativ entwickelten Android App (ID 1). Die mit Titanium entwickelte App (ID 3) überbietet dies und produziert dagegen eine mehr als doppelt so große .apk-Datei wie die mit PhoneGap und Sencha Touch entwickelte. Das ist mehr als die dreizehnfache Größe der nativ entwickelten Android App.

Die Größen nach der Installation ändern sich an dem Verhältnis zwischen der Titanium App und der PhoneGap App nur wenig. Es bleibt bei etwa der doppelten Größe. Der Größenunterschied zur nativ entwickelten App reduziert sich bei der PhoneGap App immer noch von dem sechsfachen auf das knapp vierfache und bei der Titanium App vom dreizehnfachen auf ca. das Achtfache.

ID	Name	Entwickelt mit	.apk Größe in Megabyte (MB)	Größe nach der Installation in Megabyte (MB
1	WhatsCapp	Android	0,66	1,35
2	WhatsCappPhoneGap	Sencha Touch und PhoneGap	4,33	5,14
3	WhatsCappTitanium	Appcelerator (Titanium Alloy)	8,98	10,81

Tabelle 14 Paketgrößen Vergleich der aktuellen WhatsCapp Versionen

Die Größenunterschiede erscheinen extrem groß, vor allem wenn man bedenkt, dass die nativ entwickelte Android App die meisten Anwendungsfälle abdeckt und damit die meisten Funktionen implementiert hat.

Bei der PhoneGap App lässt sich die Paketgröße auf die Sencha Touch Bibliotheken zurückführen. Würde man im Fall von WhatsCapp ganz auf die Bibliotheken verzichten, so würde die .apk-Datei nur noch 0,32 MB groß sein. Dadurch wäre sie nur halb so groß wie die nativ entwickelte App. Sie wäre allerdings nicht mehr lauffähig. Dadurch lässt sich aber ableiten, dass die App bezüglich ihrer Größe noch Optimierungspotential hat. Optimierungen können durch den Verzicht auf nicht genutzte Bibliotheken erzielt werden.

Bei der Untersuchung verschiedener Test-Apps wurde festgestellt, dass die meisten der Apps eine vergleichbare .apk-Dateigröße haben. Selbst bei einer einfachen „Hello World“-App mit dem Alloy Framework beträgt die .apk-Größe bereits 8,94 MB. Verwendet man den klassischen Ansatz mit Titanium reduziert sich die Größe um ca. 0,8 MB.

Es stellt sich die Frage, wieso die App so groß ist und was man tun kann, um die Größe zu reduzieren. In Kapitel 2.3.5.2 wurde beschreiben, dass mit jeder App eine JavaScript Runtime und eine WebKit Rendering-Engine ausgeliefert wird, welche sich auf die App Größe auswirken. Um die .apk-Dateigröße zu reduzieren, werden in einigen Foren empfohlen ältere SDK Versionen zu verwenden, was allerdings keine Lösung ist.

Der zertifizierte Titanium Experte Kevin Whinnery beschrieb in den Q&As, dass es für Entwickler kaum eine Möglichkeit gibt die Größe zu reduziern. Whinnery beschrieb auch, dass die Größe nicht auf unnötige Module zurückzuführen ist, da Titanium Build-Scripts dafür sorgen, dass nur Module von Titanium eingebunden werden die für die jeweilige App benötigt werden [AppceleratorQ&A 2013].

8.1.2 Deployment-Dauer

Dieses Kapitel beschäftigt sich mit dem Vergleich der Dauer eines Deployments auf einem Testgerät. Das verwendete Testgerät war das „Galaxy Nexus“ mit der Android Version 4.2.2. Bei der Entwicklung von Android Apps werden sie in der Regel für Testzwecke entweder auf einem Android Emulator oder auf einem echten Gerät getestet. Die Erfahrung mit dem Android Emulator war bei dem Deployment der Titanium App etwas enttäuschend, da dieser auch häufiger abstürzte. Die bessere Wahl in so einem Fall ist daher ein echtes Gerät. Dauert das Deployment auf einem Gerät sehr lange können sich Wartezeiten summieren und die Produktivität mindern. Aus diesem Grund wurden empirische Untersuchungen bzgl. der Deploymentzeit vorgenommen. Die Ergebnisse hierzu sind der Tabelle 15 zu entnehmen.

Das Deployment geschieht bei der WhatsCapp und WhatsCappPhoneGap in der gleichen Eclipse Entwicklungsumgebung und wird über einen Rechtsklick auf das Projekt mit der anschließenden Auswahl von „Run as Android Application“ ausgeführt. Die Titanium App wird über die Eclipse basierte IDE Titanium Studio deployed. Dies geschieht analog zu den anderen beiden Projekten per Rechtsklick auf das Projekt mit der anschließenden Auswahl „Install to Android Device“. Alle Projekte wurden so konfiguriert, dass sie die App nach dem Klick auf „Run“ oder „Install“ auf dem aktuell aktiven Gerät deployen.

Die Messspanne der Zeit beginnt nach dem Klick auf „Run“ bzw. nach „Install“ und endet sobald das App-Icon im App-Launcher erscheint. Da es keine Möglichkeit gibt diese Zeitspanne mit Werkzeugen zu untersuchen, wurden die Zeiten manuell per Stoppuhr gemessen. Aufgrund anderer laufender Prozesse auf dem Gerät und mögliche Messungsungenauigkeiten kann die Zeit bis zum fertigen Deploy variieren, deshalb wurde aus jeweils insgesamt 20 Messwerten ein Durchschnittswert für jede App ermittelt.

ID		1	2	3
Name		WhatsCapp	WhatsCappPhoneGap	WhatsCappTitanium
Entwickelt mit		Android	Sencha Touch und PhoneGap	Appcelerator (Titanium Alloy)
Messwert in Sekunden (s)	1	5,5	9,4	33,2
	2	5,7	8,0	34,2
	3	5,3	8,8	35,7
	4	5,1	9,4	32,8
	5	5,9	9,2	35,5
	6	4,8	8,9	34,7
	7	5,8	10,2	33,3
	8	5,0	9,7	33,2
	9	4,4	8,4	31,2

	10	4,8	10,0	34,6
	11	3,7	7,7	35,3
	12	3,8	7,5	34,2
	13	4,9	8,9	33,9
	14	3,9	9,4	34,5
	15	4,1	10,3	31,2
	16	4,5	10,5	33,9
	17	4,4	7,6	33,0
	18	5,6	7,4	35,3
	19	5,1	8,4	34,0
	20	5,0	7,1	33,1
Durchschnittswert		**4,9**	**8,8**	**33,8**

Tabelle 15 Messreihe der Deploymentzeiten für die WhatsCapp Apps mit den Paketgrößen die in Tabelle 14 beschrieben sind

Die Messwerte stammen von den in Kapitel 8.1.1 beschriebenen Apps mit den entsprechenden Paketgrößen. Dieses sollte man bei der Betrachtung der Deployment-Zeiten mit berücksichtigen.

Die Ergebnisse aus Tabelle 15 zeigen, dass der Deploy von WhatsCappPhoneGap im Durschnitt ca. vier Sekunden länger dauert, was schon doppelt so lange ist. Die Titanium hebt sich auch hier im Deployment wieder stark von den anderen beiden Apps ab. Dort dauert das Deployment fast sieben Mal so lange wie bei der nativ entwickelten Android App. Die App ist zwar nach der Installation wie in Kapitel 8.1.1 beschrieben auch mehr als acht Mal so groß aber der Grund für die lange Deployment-Dauer ist, dass zusätzliche Vorgänge bis zur fertigen .apk-Datei nötig sind. Die Vorgänge gehen aus den Konsolenausgaben hervor. Darunter sind Vorgänge, wie beispielsweise das Hinzufügen zusätzlicher Java Klassen, die Kompilierung der JavaScript Ressourcen, das Schreiben der Androidmanifest.xml und die Kompilierung der Android Ressourcen.

8.1.3 Realisierungsaufwand

Dieses Kapitel beschreibt, wie viel Zeit für die Implementierungen aufgewendet wurde. Die Zahlen sind allerdings mit etwas Distanz zu betrachten, da mehrere zusätzliche Faktoren Einfluss auf das Endergebnis haben, die hier erläutert werden. Bei den Werten handelt es sich um Schätzwerte, die Anhand des Projektplans und SVN Commits rekonstruiert wurden.

Der Tabelle 16 ist zu entnehmen, dass für das WhatsCappPhoneGap-Projekt an etwa 27 Tagen gearbeitet wurde. In dieser Hauptimplementierungsphase wurden die Hauptfunktionalitäten der aktuellen Version entwickelt. Nachträgliche Änderungen und

Optimierungen sind nicht in diesem Wert enthalten. Was in den 27 Tagen bearbeitet wurde, sind Einarbeitungen in Technologien (JavaScript) und CPTs (Sencha Touch und PhoneGap), Problemlösungsfindungen, die mehrere Tage aufhielten und anfallende Nebentätigkeiten. Aus diesem Grund wurden die 27 Tage um den Faktor 0,5 reduziert. Dies bedeutet, dass effektiv etwa 13,5 Tage dafür aufgewendet wurden.

Bei der Titanium App war der Bearbeitungszeitraum etwas kürzer. Da dort weniger Nebentätigkeiten durchgeführt wurden, wurde der Faktor auf 0,6 angehoben, so dass insgesamt 6 Tage daraus resultierten.

Stellt man die Werte 13,5 und 6 in Relation, so muss dabei beachtet werden, dass die PhoneGap App einen etwas mehr als doppelt so weiten Entwicklungsstand hat wie die Titanium App.

ID	Name	Entwickelt mit	Bearbeitungszeitraum in Tage (8h)	Faktor	Endergebnis in Tage (8h)
2	WhatsCappPhoneGap	Sencha Touch und PhoneGap	27	0,5	**13,5**
3	WhatsCappTitanium	Appcelerator (Titanium Alloy)	10	0,6	**6**

Tabelle 16 Geschätzter Realisierungsaufwand für die mit CPTs entwickelten Apps

Des Weiteren sind die Werte einzeln ebenfalls kritisch zu betrachten, da während der Entwicklung der Fokus nicht auf der schnellen Umsetzung der Anforderungen lag, sondern darin die CPTs besser kennenzulernen. Es ist also eine schnellere Umsetzung möglich, wenn der Fokus auf die Anforderungsumsetzung gelegt wird oder die Realisierung mit einem größeren Erfahrungs-Fundament angegangen wird.

8.1.4 Performance

Der Begriff Performance kann je nach Bezug unterschiedlich definiert werden. Der Begriff bezieht sich hier ausschließlich auf die Antwortzeiten der Benutzerinteraktionen. Ein Beispiel hierfür wäre die Dauer eines Tab-Wechsels nach der Auswahl eines Tabs.

Android bietet beispielweise das Traceview Tool an, welches dabei hilft Ausführungszeiten zu messen und zu visualisieren. Dies ist allerdings, wie mit einigen anderen Java Tools, nicht ohne weiteres für die ausgewählten CPTs nutzbar.

Eine Möglichkeit für Messungen, die übergreifend nutzbare wären, sind Konsolenausgaben-Auswertungen. Um eine Auswertung dieser Art vorzunehmen, müssen geeignete Messpunkte gesetzt werden an denen die Logausgaben geschrieben werden. Hier entsteht eine Schwierigkeit, wenn man mit den gewählten CPTs arbeitet. Ein Tab-Wechsel wird von den CPTs selbst ohne manuelle geschriebene Methoden umgesetzt, so dass der Startpunkt

dieser Messung nicht festlegbar ist. Aus diesen Gründen werden im Folgenden subjektive Eindrücke beschrieben.

Im Fall eines Tab-Wechsels sind weder bei WhatsCapp noch bei WhatsCappTitanium Verzögerungen bemerkbar. Bei WhatsCappPhoneGap sind im Gegensatz dazu schon Verzögerungen mit einer manuellen Stoppuhr messbar. Die Werte liegen dabei knapp unter einer Sekunde. Die Ausführung von WhatsCappPhoneGap auf dem Desktop-Browser Chrome unter Windows 7 zeigt dagegen nicht manuell messbare Verzögerungen. Da die Leistungsfähigkeit des Testrechners in Bezug auf den Arbeitsspeicher und die Rechenleistung höher ist als bei dem Test-Smartphone wird die geringere Leistung als naheliegendster Grund vermutet.

Neben den WhatsCapp-Apps sind auf dem Testgerät noch weitere Apps installiert, welche den Arbeitsspeicherverbrauch negativ beeinflussen. Die Einflüsse von anderen Apps können zwar ausgeschaltet werden, aber dies würde nicht mehr der Realität entsprechen, da meistens neben den mitgelieferten Apps ab Werk noch zusätzliche installiert sind, die den Arbeitsspeicherverbrauch zusätzlich beeinflussen.

8.1.5 User Expierience

Unter User Expierience wird hier das subjektive Nutzungserlebnis einzelner Personen beim Verwenden von WhatsCapp verstanden. Zu diesem Zweck können Untersuchungen vorgenommen werden, die an Probanden durchgeführt werden können. In diesem Kapitel wird beschrieben, was man bei einer Untersuchung dieser Art beachten muss. Da die ausgiebige Planung und Ausführung dieser Untersuchungen den Rahmen dieser Arbeit sprengen würden, bleiben die Betrachtungen beim Theoretischen.

Die Ergebnisse einer Untersuchung des Nutzungserlebnisses an Probanden hängen von mehreren Faktoren ab:

- App
    - Aussehen (Look & Feel)
    - Performance (Kapitel 8.1.4)
    - Funktionalität
- Probanden
    - Erfahrungen mit Apps und mobilen Betriebssystemen
    - Altersgruppe
    - Technophob oder technikaffin?
- Test
    - Reihenfolge der App
    - Optik

Eine zentrale Rolle spielen die Apps an sich. Um aussagekräftige Ergebnisse zu erhalten, sollte das Aussehen der Apps sich möglichst gleichen. Betrachtet man die WhatsCapp-Apps sind starke Unterschiede (Abbildung 30, Abbildung 34 und Abbildung 37) zu erkennen. Da es hierbei um das Betriebssystem Android geht, würde man die mit CPTs entwickelten Apps möglichst an die Nativ entwickelte App anpassen, was ein größerer Mehraufwand ist, wenn man es manuell durchführt. In Abbildung 37 wurde unter anderem versucht die Kopfleiste (Bei Android „Action Bar“ genannt) der FellowDetails-View nachzubilden. Zwar bietet Titanium die Möglichkeit die Nutzung einer nativen Actionbar an, aber an diesem Beispiel konnte ein Eindruck gewonnen werden, wie aufwändig und sauber es wäre einzelne UI Elemente nachzubilden. Das Ergebnis dieser ersten Erfahrung ist, dass es Zeit kostet und etwas Experimentieraufwand bedeutet, was sich letztendlich nicht lohnt, da das Ergebnis doch nicht dem Original in Perfektion entspricht.

Ein weiterer Punkt ist die Performance, die im vorangehenden Kapitel beschrieben wurde aber auch die Umsetzung der Funktionalitäten ist ein wichtiger Punkt. Fehlen Funktionalitäten, die bei der Anderen App vorhanden sind, können die Ergebnisse der Untersuchungen an Aussagekraft verlieren. Daher müssten alle Apps erst einmal auf den gleichen Stand gebracht werden.

Ein weiterer Aspekt, der sich auf die Ergebnisse auswirkt, sind die Probanden. Man muss sich bei der Wahl der Probanden Gedanken darüber machen, welchen Hintergrund sie mitbringen. Probanden, die vorher nie mit Smartphone Apps in Berührung kamen, werden vermutlich andere Erwartungen an die Performance, das Aussehen und die Funktionalitäten haben als jüngere, technikaffine Probanden.

Bei der Durchführung sollte die Test-Reihenfolge der Apps bei Probanden variiert werden um festzustellen ob, die Reihenfolge Auswirkungen auf die Ergebnisse hat. Die Optik sollte auch nach Möglichkeit bei allen gleich sein. Im Gegensatz zum Look and Feel sind hiermit Iconfarben der Apps, Ladebildschime usw. gemeint.

Neben all diesen Aspekten existieren noch weitaus mehr Aspekte mit denen man sich auseinandersetzen könnte. Dies wäre ein Ansatzpunkt für aufbauende Arbeiten.

8.2 Bewertung

8.2.1 PhoneGap und Sencha Touch

8.2.1.1. **Integration der Quasar-Client Dialogarchitektur**

Die Verwendung des MVC Architekturmuster in dem WhatsCappPhoneGap-Projekt legt eine gute Grundlage für die Integration der Quasar-Client Dialogarchitektur. In Kapitel

6.4.1.2 und Kapitel 7.3.2 wird gezeigt, dass die Übertragung des Dialogbegriffs auf Projektstruktur mit geeigneter Namensgebung und Organisation in Ordnern abbildbar ist.
Dialoge sind nach dem Standardvorgaben von Sencha Touch zuerst technisch nach dem MVC Architekturmuster und erst auf zweiter Ebene fachlich getrennt. Die Standardvorgaben sind nicht verkehrt und für den Einstieg in Sencha Touch die richtige Wahl. Bei größeren und komplexeren Apps ist allerdings die Unterteilung zuerst nach der Fachlichkeit vorzuziehen, da dadurch eine Dialogkomponente genau einem Ordner zugeordnet werden kann. Dadurch wird die Übersichtlichkeit, die auch die A-Architektur der Abbildung 33 wiederspiegelt erhöht. Die Möglichkeit dies abzubilden bietet Sencha Touch, was sich positiv auf die Integration der Quasar-Client Dialogarchitektur auswirkt und auch die Beherrschbarkeit sehr großer Apps verbessert.

Die Dialogkern-Subkomponenten konnten bis auf die Dialogzustandsverwaltung durch das Sencha Model, dem Store und dem Controller abgebildet werden. Die Präsentations-Subkomponenten dagegen konnte durch die View größtenteils abgedeckt werden, wobei einzelne Subkomponenten mit dem Controller aus dem Dialogkern verschmolzen sind. Dies ist suboptimal und sollte durch strikte Konventionen in der Aufteilung von Methoden in zusätzliche Artefakte neben den MVC-Artefakten optimiert werden.

Auch ohne zusätzliche Artefakte zu erstellen ist es möglich durch Namenskonventionen die Zugehörigkeit einer Methode zu kennzeichnen. Durch die stärkere Entkapselung des technischen Codes in Methoden ist eine saubere Trennung zwischen Fachlichkeit und Technik möglich. Hiermit sind Mehraufwände nötig, aber es ist möglich. Dabei ist zu beachten, dass man durch die Verwendung des Sencha Touch Frameworks durch alle Artefakte hindurch an die Technik gebunden ist. Um die Fachlichkeiten ganz von dem technischen Sencha Touch Framework zu trennen, müssen diese mit purem JavaScript abgebildet werden.

Die Integration ist im Groben bereits gegeben, aber Verschmelzungen verhindern eine saubere Aufteilung. Um Letztere zu erhalten ist ein Mehraufwand nötig, der auf festzulegende Konventionen aufsetzen muss.

8.2.1.2. **Anforderungsumsetzbarkeit**

Das Kapitel 7.3.4 beschreibt, dass die aufgestellten Anforderungen aus Kapitel 4.3.3, bis auf eine, gänzlich umgesetzt werden können. Dabei können einzelne Anforderungen nur durch zusätzliche Plugins umgesetzt werden, die dazu auch noch plattformspezifisch sind.

Gut ist, dass viele PhoneGap Plugins zur Verfügung stehen, um notwendige Anforderungen abzubilden, doch sind diese Plugins nicht immer für jede Zielplattform vorhanden. Die Zahl der verfügbaren Plugins ist für die Android und iOS Plattform groß, aber für alle anderen wesentlich geringer. Die Nutzung eines Plugins bedeutet für jede weitere Plattform die

unterstützt werden soll, einen zusätzlichen Aufwand für die Integration und den Einsatz des Plugins. Dies mindert den Vorteil gegenüber der nativen Entwicklung mit jedem weiteren Plugin. Die Nutzung von Plugins ist der Meinung des Autors nach zwar ein gangbarer Weg aber keine optimale Lösung für die Cross-Plattform Entwicklung.

Die Funktionalitäten, die PhoneGap bzw. jetzt Cordova und Sencha Touch zusammen bieten reichen für die meisten Anwendungen aus. Muss man auf ein Plugin ausweichen oder sogar eins selber schreiben sind es meistes Nebenanforderungen, die diese benötigen. Hauptanforderungen sind meist nicht betroffen und falls doch ist die Nutzung eines Plugins auch kein großes Hindernis. Werden zu viele Plugins benötigt, sollte man sich Gedanken machen, ob eine native Entwicklung oder die Entwicklung mit anderen CPTs nicht doch eine bessere Lösung bietet. Ob und wie stark ein CPT geeignet ist hängt von den individuellen Anforderungen eines Projektes ab.

8.2.1.3. **Erfahrungen**

Dieses Unterkapitel soll einen kurzen und knappen Überblick über gemachte Erfahrungen mit PhoneGap und Sencha Touch in Bezug auf Tools, Support, Lernkurve, Entwicklung & Debugging, Gerätefunktionalitäten und UI Funktionalitäten geben.

Tools
Für PhoneGap gibt es keine zusätzlichen Tools. Sencha bietet aber kostenpflichtige Tools an. Es wurde ein kurzer Blick auf Sencha Architect und dem Eclipse Plugin geworfen. Mit Sencha Architect können per Drag and Drop die Benutzeroberflächen erstellt werden, was vor allem für Sencha Anfänger eine besondere Erleichterung ist und Zeit sparen kann. Das Suchen der richtigen UI-Elemente in den Dokumentationen und die Strukturierung dieser, werden subjektiv betrachtet für Anfänger erheblich beschleunigt. Bei der Installation muss auf die richtige Eclipse Version geachtet werden. Das Plugin wurde nur kurz betrachtet, wobei auf den ersten Blick nichts besonders auffiel.

Support
Die PhoneGap Dokumentationen erklären kurz und prägnant, was für ein Hardwarezugriff notwendig ist und zeigt kurze Beispiele dazu was als sehr gut empfunden wurde. Bei Problemen finden sich im Internet bereits nach kurzem Suchen Lösungen bzw. Erklärungen.
Bei Sencha Touch stifteten die Versionsunterschiede zwischen Sencha Touch und Sencha Touch 2 Verwirrungen. Die Suchergebnisse waren meist auf die Sencha Touch 1.X Versionen bezogen. Beispielprojekte gibt es subjektiv betrachtet wenige im Internet. Es werden aber mehrere Beispiele im Downloadpaket mitgeliefert. Diese sollten während der Projekteinrichtung nicht in Vergessenheit geraten. Die online Dokumentation von Sencha Touch ist die Hauptbezugsquelle von Informationen und bietet ausreichende Informationen.

Lernkurve
Die Lernkurve für PhoneGap ist subjektiv betrachtet geringer als bei Sencha Touch. Die Beispiele sind aussagekräftig und verständlich, wenn JavaScript Kenntnisse vorhanden sind. Die Schwierigkeit lag am Anfang dabei, dass neben JavaScript auch noch der Umgang mit den beiden CPTs auf einmal erlernt werden mussten. Das Zusammenwirken mehrerer Faktoren, wie die Einarbeitung in verschiedene CPTs und Technologien, den Versionsverwirrungen und anfänglich übersehenen Codebeispielen machen den Einstieg in Sencha Touch nicht einfach. Nach der Überwindung der ersten Hürden steigt die Lernkurve subjektiv betrachtet, wie auch bei anderen Frameworks.

Entwicklung & Debugging
Die Nutzung von PhoneGap macht, wie bereits beschrieben, geringe Probleme und daher lässt sich hier nicht viel kritisieren. Bei Sencha lässt sich auch kaum etwas kritisieren, wenn die ersten Hürden erst einmal genommen wurden. Der Code sieht strukturiert und verständlich aus bei Sencha Touch, so dass die Entwicklung mehr Spaß machte als mit Titanium.

Positiv ist, dass die App mit Sencha Touch bequem auf dem lokalen Browser getestet werden kann, doch bei Hardwarezugriffen wird durch die Nutzung von PhoneGap ein echtes Gerät oder im Fall von Android ein Emulator benötigt. Das Debugging erschien über Logausgaben in der Konsole sehr mühselig. Hinzu kamen noch die fehlende Syntaxprüfung bei der direkten Integration in ein Android Projekt, was allerdings nach kurzer Zeit ein geringes Problem darstellte.

Gerätefunktionalitäten
Die Zugriffe auf Gerätefunktionalitäten gelangen gut. Der Zugriff auf den Standort wurde über Sencha Touch realisiert und der Rest über PhoneGap. So wurden in dieser Hinsicht Erfahrungen von beiden CPTs gewonnen.

UI Funktionalitäten
PhoneGap bietet selber keine UI-Funktionalitäten an. Dies war auch einer der Gründe Sencha Touch in Kombination einzusetzen. Bei der Umsetzung der UI-Elemente von Sencha Touch fehlte es an nichts. Es sei aber dabei anzumerken, dass die Benutzerüberfläche stark an iOS Oberflächen angelehnt ist.

8.2.2 Appcelerator (Titanium Alloy)

8.2.2.1. **Integration der Quasar-Client Dialogarchitektur**

Wie bei Sencha Touch bildet auch bei der Entwicklung mit dem Alloy-Framework das MVC Architekturmuster ein gutes Fundament zur Integration der Quasar-Client Dialogarchitektur. Aber anders als bei Sencha Touch lässt sich, wie in Kapitel 0 und Kapitel 7.4.2 beschrieben, der Dialogbegriff nur visuell begrenzt auf die Projektstruktur abbilden. Eine freie Benennung der einzelnen Dateien einer Dialogkomponente ist nicht möglich und auch die Strukturierung mit einem Ordner pro Dialogkomponente lässt sich nicht ohne weiteres umsetzen. Die Abbildung der Quasar-Client Dialogarchitektur auf die Alloy-Projektstruktur ist eingeschränkter und damit schlechter als bei PhoneGap & Sencha Touch.

Die Dialogkern-Subkomponenten konnten auch hier bis auf die Dialogzustandsverwaltung durch das Model und dem Controller abgebildet werden. Die Präsentations-Subkomponenten werden dagegen in der View und dem Style umgesetzt. Die erstellte Abbildung 26 visualisierte diese Zuordnung anschaulich. Dabei fiel wie bei Sencha Touch auf, dass ein Teil der Präsentationszustandsteuerung mit dem Controller des Dialogkerns verschmolzen ist. Diese Verschmelzung ist, wie bei PhoneGap & Sencha Touch in Kapitel 8.2.1.1 beschrieben, suboptimal und sollte auch hier durch strikte Konventionen in der Aufteilung von Methoden geregelt werden. Namenskonventionen und eine stärkere Entkapselung von Methoden sind hier ebenfalls ein Mittel, um eine saubere Trennungen zwischen Fachlichkeit und Technik zu ermöglichen.

Die Integration der Quasar-Client Dialogarchitektur ist in vielen Ansätzen gegeben, aber nicht vollständig und nicht immer trennscharf. Die Übertragung der Architektur auf die Projektstruktur ist schlechter als bei PhoneGap & Sencha Touch aber die teilweise Verschmelzung von Dialogkern und Präsentation stellt bei beiden ein ähnliches Defizit dar.

8.2.2.2. **Anforderungsumsetzbarkeit**

Das Kapitel 7.4.4 beschreibt, dass die aufgestellten Anforderungen aus Kapitel 4.3.3 bis auf zwei Anforderungen alle umgesetzt werden können und dass durch die Integration von plattformspezifischen APIs die Anforderungserfüllbarkeit verbreitert wurde.

Zum einen ist es gut, dass man nicht auf Plugins zurückgreifen muss bei denen die Funktionsfähigkeit nicht zwangsweise gewährleistet ist. Doch die Nutzung integrierter plattformspezifischer APIs ändert nichts daran, dass für jede Plattform individuelle Anpassungen vorgenommen werden müssen, wenn die Anforderung nicht mit der puren (ohne plattformspezifische Bibliotheken) Titanium API realisiert werden kann.

Auf plattformspezifische APIs wurde bei WhatsCappTitanium nicht zurückgegriffen. Hier gilt prinzipiell die gleiche Aussage wie bei PhoneGap & Sencha Touch. Die plattformspezifischen „Lösungen“ wiedersprechen nach Meinung des Autors dem Grundgedanken der Cross-Plattform-Entwicklung, wenn die Nutzung des Spezifischen überhandnimmt. Für die meisten Anwendungen bietet Titanium aber ausreichend viele Funktionalitäten zur Erfüllung der Anforderungen. Ob und wie stark Titanium geeignet ist hängt wiederum von den individuellen Anforderungen eines Projektes ab.

8.2.2.3. **Erfahrungen**

Analog zu den Erfahrungen zu PhoneGap und Sencha Touch des Kapitels 8.2.1.3 wird auch hier zu den gleichen Kriterien ein kurzer und knapper Überblick über gemachte Erfahrungen mit Titanium beschrieben.

Tools
Für die Entwicklung mit Titanium gibt es die auf Eclipse basierende kostenlose IDE Titanium Studio. Es gab keine Probleme und sie erfüllt ihren Zweck so wie sie soll inkl. Syntax-Highlighting, Syntaxprüfung und Autovervollständigung.

Support
Da das Alloy Framework erst seit kurzem (Kapitel 0) veröffentlicht wurde, ist der Support durch die Community noch nicht sehr groß. Beispiel Code-Schnipsel und kleinere Beispielprojekte sind direkt über die Titanium Studio abrufbar. Es gibt große Unterschiede zwischen der Entwicklung mit dem Alloy-Framework und der klassischen Titanium Entwicklung. In der aktuellen Dokumentation sind teilweise parallele Beispiele für das Alloy Framework vorhanden, aber an vielen fehlen diese noch. Aufgrund der kleinen Community und der nicht durchgängig gepflegten Dokumentation, wird der Support schlechter als bei Sencha Touch eingestuft.

Lernkurve
Die Lernkurve hier war trotz Einarbeitung in das Framework und der Auseinandersetzung mit den XML-Views und Titanium Stylesheets subjektiv betrachtet viel geringer. Dies kann allerdings daran liegen, dass die Erfahrungen mit PhoneGap und Sencha Touch sich positiv auf die Einarbeitung ausgewirkt haben.

Entwicklung & Debugging
Da Titanium Studio wie bereits beschrieben bei der Entwicklung Unterstützung anbietet und die IDE dazu auch noch kostenlos ist, wird dies als klarer Pluspunkt gegenüber der Sencha Touch gesehen. Die Fehlersuche wurde nicht als negativ empfunden, da bei der bisherigen Entwicklung Fehler durch aussagekräftige Beschreibungen und Lokationsangaben schnell gefunden werden konnten. Hierbei sollte beachtet werden, dass im Vergleich zum WhatsCappPhoneGap-Projekt, weniger Zeit und damit auch weniger Erfahrungen mit Alloy gesammelt wurden und das Debugging dadurch auch nur begrenzt

beurteilt werden kann. Was aber negativ auffiel sind die in Kapitel 0 kritisierten langen Deploymentzeiten.

Gerätefunktionalitäten
Hardwarezugriffe, die genutzt wurden, klappten ohne nennenswerte Probleme. Aus diesem Grund ist hier nichts Negatives hervorzuheben.

UI Funktionalitäten
Die UI-Elemente entsprechen tatsächlich dem nativen Look and Feel von Android. Da sich das Look and Feel bzgl. der Farben und der UI-Elemente im Laufe der Weiterentwicklung des Betriebssystems verändert hat, sieht die Benutzeroberfläche von Titanium Apps subjektiv betrachtet nicht mehr zeitgemäß modern aus. Ein Alert Dialog sieht beispielsweise bei der Ausführung auf einem Gerät mit Android 4.X nicht weiß und blau sondern schwarz und grau aus.

9 Zusammenfassung und Ausblick

Dieses abschließende Kapitel fasst die Ergebnisse dieser Arbeit im Unterkapitel 9.1 prägnant zusammen und gibt im Unterkapitel 0 einen Ausblick auf zukünftige Arbeiten.

9.1 Zusammenfassung

Ziel der Arbeit war es die Integration von Standard-Architekturen für Cross-Plattform Entwicklungsansätze in mobile Apps zu analysieren und zu beurteilen. Als Beispiel für eine Standard Architektur wurde die komponentenbasierte Quasar-Client Architektur aus Kapitel 2.4.5 gewählt. Der Fokus wurde dabei im Speziellen auf die Dialogkomponenten-Architektur (Dialogarchitektur) mit den Subkomponenten Präsentation und Dialogkern gelegt. Das konkretisierte Analyse- und Bewertungsziel war es diese Dialogarchitektur für ausgewählte Cross-Plattform-Tools (CPTs) zu untersuchen.

In Kapitel 0 wurden zunächst die Verständnisgrundlagen in Bezug auf Cross-Plattform Entwicklung gelegt. Dazu zählt einerseits die Unterscheidung verschiedener App Typen, das Aufzeigen des Sinns für die Verwendung von Cross-Plattform-Tools (CPTs) und andererseits auch die CPTs selbst mit ihren verschiedenen Technologieansätzen und die Beschreibung einiger dieser Tools. Des Weiteren wurden zusätzlich grundlegende Informationen zur Quasar-Client-Dialogarchitektur vermittelt und das MVC-Architekturmuster als Lösungsbasis für die spätere Entwicklung erläutert.

Vor der Entwicklung und Planung wurden vergleichbare Arbeiten herangezogen und bezüglich ihrer Ergebnisse, Gemeinsamkeiten, Differenzen und möglichen Nutzen analysiert und in Kapitel 3 prägnant zusammengefasst.

Für die Durchführung der Analysen und Bewertungen der Integration waren in erster Linie Erfahrungen mit CPTs notwendig. Diese Erfahrungen sollten an einem sinnvollen Fallbeispiel gesammelt werden. Vor dem Beginn der Implementierungen wurden in Kapitel 4 Anforderungen analysiert, die als Orientierung für die zu entwickelnde App dienen sollten. Durch die exemplarische Analyse von Apps aus dem Google Play Store wurde am Ende des 4. Kapitels eine Liste mit Anforderungen aufgestellt, die das Ergebnis einer Kombination von Rechercheergebnissen und den eigenen Anforderungsanalysen darstellte.

Nach den aufgestellten Anforderungen wurden in Kapitel 0, basierend auf Nutzungsanalyseergebnissen und eigenem Testen, die beiden CPTs PhoneGap (Cordova) und Titanium gewählt. Zur Ergänzung von PhoneGap wurde Sencha Touch gewählt, da hierdurch die mangelnde Benutzeroberflächenfunktionalität von PhoneGap ausgeglichen wurde und das CPT zudem noch die Umsetzung des MVC-Architekturmusters begünstigt.

Aufbauend auf WhatsCapp, einer Schulungs-App der Firma Capgemini, wurden die benötigten Erfahrungen durch Nachbildungen für die ausgewählten CPTs gesammelt. Mit den gesammelten Erfahrungen wurden in Kapitel 6 die Integration der Quasar Prinzipien, von dem konkreten Fallbeispiel losgelöst, theoretisch beschrieben und daraufhin in Kapitel 7 am Beispiel von WhatsCapp praxisbezogener erläutert und aufgezeigt. Das Ergebnis ist, dass durch die Verwendung des MVC-Architekturmusters bei beiden CPTs bzw. CPT-Kombinationen eine gute Grundlage gebildet wird, um die Quasar Dialogarchitektur zu integrieren. Es gibt allerdings Einschränkungen, wie zum Beispiel die teilweise Verschmelzung von Präsentation und Dialogkern wie es auch bei Android, im Kapitel 6.2.1 beschrieben, der Fall ist. Um eine höhere Trennschärfe zu erreichen und damit eine saubere Dialogarchitektur nach Quasar-Client umzusetzen sind allgemein formuliert für die Entwicklung mit JavaScript, Regeln und klare Konventionen zu erstellen, die eine strikte Trennung möglich machen. Ein Ansatz dies teilweise zu verwirklichen ist es, Methoden zum einen visuell durch Post- oder Präfixe bzgl. ihrer Zugehörigkeit zu kennzeichnen, und zum anderen Methoden auf einer ausreichend granularen Ebene zu entkapseln, so dass der Dialogkern frei von Technik sein kann.

Zusätzlich wurden die in Kapitel 4 aufgestellten Anforderungen hinsichtlich der generellen Umsetzbarkeit mit den CPTs und der Verwendung in WhatsCapp betrachtet. Die meisten der aufgestellten Anforderungen wurden nicht von WhatsCapp genutzt, da sie teilweise nicht sinnvoll anwendbar waren und lediglich als Orientierung dienen sollten und somit nicht den Fokus bildeten. Die Umsetzbarkeitsanalyse der Anforderungen ergab, dass sowohl für PhoneGap & Sencha Touch als auch für Titanium mit dem Alloy Framework fast alle Anforderungen umgesetzt werden können. Ein Teil der Anforderungen kann allerdings nur plattformspezifisch umgesetzt werden.

Abschließend zu dieser Arbeit wurden in Kapitel 8 Vergleichsanalysen bzgl. verschiedener Kriterien beschrieben und teilweise durchgeführt. Dabei schnitten PhoneGap & Sencha Touch und Titanium in unterschiedlichen Kriterien mal besser und mal schlechter ab. Die anschließenden Bewertungen fassen die Integration der Quasar-Client Dialogarchitektur noch einmal gebündelt zusammen, bewerten die Anforderungsumsetzbarkeit und spiegeln gesammelte Erfahrungen wieder.

9.2 Ausblick

Zukünftige Arbeiten können in verschiedenen Bereichen erfolgen. In der Zusammenfassung und in den Bewertungen der Integrationen wurde angemerkt, dass keine vollständige Integration der Quasar-Client Dialogarchitektur gegeben ist und dass ein vollständiges Konzept mit Regeln und Konventionen für die JavaScript Entwicklung notwendig ist. Dieses Konzept existiert noch nicht. Dazu bedarf es einen Erfahrungsaustausch zwischen mehreren erfahrenen JavaScript-Entwicklern und technischen Architekten. Die Erstellung dieses Konzepts wäre einer der Bereiche in denen weitere Arbeiten laufen könnten.

Im Anschluss an die Erstellung des Konzepts wäre zu betrachten, welche Konzeptanpassungen für spezifische CPTs gemacht werden müssen. Daraufhin wäre eine weitere Umsetzung eines Fallbeispiels nach den Regeln und Konventionen des Konzepts möglich, um so in einer weiteren Iteration das Konzept auf Schwachstellen zu untersuchen und daraufhin zu verbessern. Statt der Umsetzung einer neuen App, ist auch ein Refactoring der bestehenden App sinnvoll.

Ein weiterer Bereich wäre das Testen der erstellten Apps auf anderen Plattformen wie iOS, Blackberry oder Windows Phone. Hierdurch können der Mehraufwand des Deployments anderer Plattformen betrachtet und verglichen, aber auch Unterschiede hinsichtlich verschiedener Kriterien untersucht werden.

In Kapitel 8.1.5 wurde kurz gefasst beschrieben, was es bedeuten würde User Expierience Untersuchungen durchzuführen. Auch hier wird Potential für weitere Arbeiten gesehen.

Glossar

Begriff	Bedeutung
App	Anwendung: Kurzform für das englische Wort Application [Franke u.a. 2012].
Assets	Asstes beschreiben Datenbestände.
B2B	Abkürzung für Business-To-Business [Sommer 2012]. Es beschreibt die Beziehung zwischen zwei Unternehmen.
B2C	Abkürzung für Business-To-Customer [Sommer 2012]. Es beschreibt die Kommunikationsbeziehung zwischen einem Unternehmen und Privatpersonen.
B2E	Abkürzung für Business-To-Employee [Sommer 2012]. Es beschreibt die Kommunikationsbeziehung zwischen einem Unternehmen und seinen Mitarbeitern.
CRUD	CRUD steht für die typischen Datenbankoperationen Create, Read, Update und Delete [Frei 2012].
Enterprise Applikation	Eine in einem Unternehmen genutzte Anwendung, die meist mit anderen Anwendungen interagiert.
Exabyte	Eine Maßeinheit die 10^18 Bytes entspricht
MVC	Abkürzung für Model View Controller. Es ist ein Architekturmuster, welches das Ziel hat eine losere Kopplung zu erreichen [Lahres u.a. 2006] [Veit u.a. 2003].
MVVM	Abkürzung für Model View ViewModel. Es ist eine Variante des MVC Architekturmusters.
Observer Pattern	Ein ist objektbasiertes Verhaltensmuster. Beim Einsatz dieses Musters werden bei Zustandsänderungen eines Objektes alle abhängigen Objekte benachrichtigt und automatisch aktualisiert [Gamma u.a. 2004].
Rich Web Applikation	Äquivalent mit Rich Internet Application. Bezeichnet komplett im Browser laufende Web-Clients [Däschlein u.a. 2010].
Separation of Concerns	Ein Architekturprinzip zur Trennung von Zuständigkeiten in einem System. Jeder Softwareteil beschäftigt sich dabei genau mit einer Aufgabe [Däschlein u.a. 2010].
Theme	Themes sind eine veränderbare grafische Oberflächengestaltung.
Touch UI	Mit Touch User Interfaces sind Benutzeroberflächen gemeint, die über Berührungen gesteuert werden können.
WAP	WAP ist die Abkürzung für Wireless Application Protocol. Es beschreibt einen einheitlichen Standard, der es erlaubt Internetinhalte über das langsame Mobilfunknetz zu übertragen und auf einem mobilen Gerät darzustellen [Frei 2012].
Web App	Eine Webseite, die sich wie ein Programm anfühlt und bedienen lässt [Franke u.a. 2012].

Abbildungsverzeichnis

Tabellenverzeichnis

Code-Beispiel-Verzeichnis

Quellenverzeichnis

[6Wunderkinder 2013] 6Wunderkinder: Wunderlist, URL: http://www.6wunderkinder.com/wunderlist, Zugriffsdatum: 23.02.2013

[Android-Binding 2013] Android-Binding, URL: http://code.google.com/p/android-binding/, Zugriffsdatum: 11.02.2013

[AppceleratorBlog 2013] Appcelerator: Appcelerator Blog, URL: http://developer.appcelerator.com/blog/, Zugriffsdatum: 13.03.2013

[AppceleratorDoc 2013] Appcelerator: Titanium 3.0, URL: http://docs.appcelerator.com/titanium/latest/, Zugriffsdatum: 17.02.2013

[AppceleratorQ&A 2013] Appcelerator: Community Questions & Answers, URL: http://developer.appcelerator.com/questions, Zugriffsdatum: 19.03.2013

[AppceleratorWiki 2013] Appcelerator: Appcelerator Wiki, URL: https://wiki.appcelerator.org/, Zugriffsdatum: 17.02.2013

[BITKOM 2012] BITKOM: Fast eine Milliarde App-Downloads allein in Deutschland, BITKOM - Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., URL: http://www.bitkom.org/files/documents/BITKOM-Presseinfo_Markt_Apps_in_Deutschland_23_02_2012.pdf, Februar 2012

[Charland u.a. 2011] Andre Charland und Brian LeRoux: Mobile Application Development: Web vs. Native, ACM, URL: http://queue.acm.org/detail.cfm?searchterm=ios+android&id=1968203, 2011

[Cisco 2012] Cisco: Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update,2011–2016, URL: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html, Februar 2012

[CordovaDoc 2013] Apache: Apache Cordova Documentation, URL: http://docs.phonegap.com/en/2.4.0/index.html, Zugriffsdatum: 11.02.2013

[Däschlein u.a. 2010] Ralf Däschlein, Jochen Englert, Thilo Hermann, Michael Friedrich, Christian Harms, Andreas Hess, Bernhard Humm, Marc Jäckle, Alexander Ramisch, Joachim

Wenzel, Arno Weiß, Thomas Wolf und Marcus Zander: Architekturleitfaden Anwendungskonstruktion, Capgemini sd&m AG, 2010

[Developer Android 2013] Android Developer, URL: http://developer.android.com/, Zugriffsdatum: 11.02.2013

[Fling 2009] Brian Fling: Mobile Design and Development, O'Reilly Media, ISBN: 978-0-596-15544-5, August 2009

[Franke u.a. 2012] Florian Franke und Johannes Ippen: Apps mit HTML5 und CSS3 für IPhone, IPad und Android. 1. Bonn: Galileo Computing 2012 - ISBN 978-3-8362-1848-1

[Frei 2011] Maximilian Frei: X-Platform mobile apps, Capgemini, Oktober 2011

[Frei 2012] Maximilian Frei: User Expierience in mobilen Cross Platform Anwendungen, Universität Hamburg, Februar 2012

[Gamma u.a. 2004] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides: Entwurfsmuster, Addison-Wesley Verlag, ISBN-13: 978-3-8273-2199-9

[Gartner 2012] Gartner: Gartner Says Worldwide Sales of Mobile Phones Declined 3 Percent in Third Quarter of 2012; Smartphone Sales Increased 47 Percent, URL: http://www.gartner.com/newsroom/id/2237315, Zugriffsdatum: 23.01.2013

[Geisler u.a. 2012] Robert Geisler und Sven Roth: Fit for Mobile; Ein mobiles Fitness-programm für Organisationen, OBJEKTSpektrum, URL: http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/os/2012/Mobile/roth_geisler_OS_Mobile_2012.pdf, 2012

[Golem.de 2013] Golem.de: Wunderkit ist tot, es lebe Wunderlist 2, URL: http://www.golem.de/news/6wunderkinder-wunderkit-ist-tot-es-lebe-wunderlist-2-1209-94392.html, Zugriffsdatum: 23.02.2013

[Haft u.a. 2005] Martin Haft und Bernd Olleck: Quasar Client Architekturen, sd&m AG, 2005

[Haft u.a. 2007] Martin Haft und Bernd Olleck: Komponentenbasierte Client-Architektur, Springer Verlag, 2007, URL: http://www.springerlink.com/content/f52u333756rxhw6g/

[Haft u.a. 2012] Martin Haft, Bernd Olleck, Thomas Rath und Christoph Schönfeld: Architektur der Dialogkomponenten: Die Quasar-Client Architektur für Dialoge Version 4.0, Capgemini, Oktober 2012

[Ho u.a. 2010] Hui-Yi Ho und Ling-Yin Syu: Uses and gratifications of mobile application users, URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5559869&contentType=Conference+Publications&queryText%3Dmobile+applications

[Jasar 2011] Michael Jaser: Evaluation, Bewertung und Implementierung verschiedener Cross-Plattform Development Ansätze für Mobile Internet Devices auf Basis von Web-Technologien, Hochschule Augsburg, März 2011, URL: http://cross-mobile-apps.de/files/bachelorthesis-michael-jaser.pdf - Zugriffsdatum: 23.09.2012

[Joussen 2012] Friedrich Joussen: Mobile Kommunikation – Daten und Trends, BITKOM - Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., URL: http://www.bitkom.org/files/documents/BITKOM-Praesentation_PK_Mobile_World_15_02_2012.pdf, Februar 2012

[Lahres u.a. 2006] Bernhard Lahres und Gregor Rayman: Praxisbuch Objektorientierung, Galileo Computing, ISBN 978-3-89842-624-4, URL: http://openbook.galileocomputing.de/oo/oo_06_moduleundarchitektur_001.htm

[Maier 2011] Heiko Maier: Entwurf einer komponentenbasierten Dialogarchitektur mobiler Endgeräte auf Basis von Quasar Client und Android, Hochschule Karlsruhe Technik und Wirtschaft, November 2011

[Microsoft 2013] Microsoft: Model-View-Controller, Microsoft, URL: http://msdn.microsoft.com/en-us/library/ff649643.aspx, Zugriffsdatum: 24.01.2013

[Sencha 2011] James Pearce: A Sencha Touch MVC application with PhoneGap, Sencha Inc., URL: http://www.sencha.com/learn/a-sencha-touch-mvc-application-with-phonegap/, Zugriffsdatum: 18.10.2012

[SenchaDoc 2013] Sencha: Touch 2.1.1 Sencha Docs, URL: http://docs.sencha.com/touch/2-1/, Zugriffsdatum: 11.02.2013

[Siedersleben 2003] Johannes Siedersleben: Quasar: Die sd&m Standardarchitektur: Teil 1, sd&m Research, 2003, URL: https://www.fbi.h-da.de/fileadmin/personal/b.humm/Publikationen/Siedersleben_-_Quasar_1__sd_m_Brosch_re_.pdf, Zugriffsdatum: 22.10.2012

[Siedersleben 2004] Johannes Siedersleben: Moderne Softwarearchitektur: Umsichtig planen, robust bauen mit Quasar, 1. Auflage, Dpunkt-Verlag, 2004, Heidelberg – ISBN: 3-89864-292-5

[Sommer 2012] Andreas Sommer: Comparison and evaluation of cross-platform frameworks for the development of mobile business applications, Technische Universität München, Oktober 2012

[StatCounter 2012] StatCounter Global Stats: Top 8 Mobile Operation Systems on Oct 2012, URL: http://gs.statcounter.com/#mobile_os-ww-monthly-201210-201210-bar, Zugriffsdatum: 15.10.2012

[StatCounter] StatCounter: Global Stats, 2013, URL: http://gs.statcounter.com/

[Ullrich 2012] Florian Ullrich: Konzeption und Realisierung einer komponentenbasierten Client-Architektur für mobile Anwendungen nach Quasar für die iOS-Plattform, Hochschule für Angewandte Wissenschaften Hamburg, August 2012

[Veit u.a. 2003] Matthias Veit und Stephan Herrmann: Model-View-Controller and Object Teams: A Perfect Match of Paradigms, ACM, URL: http://dl.acm.org/citation.cfm?id=643618, 2003

[VisionMobile 2012] VisionMobile Ltd.: Cross-Platform Developer Tools – Bridging the worlds of mobile apps and the web, Februar 2012, URL: http://www.visionmobile.com/product/cross-platform-developer-tools-2012/, Zugriffsdatum: 28.08.2012

[VisionMobile 2013] VisionMobile Ltd.: Developer Economics – Developer Tools: The Foundations of the App Economy, Januar 2013, URL: http://www.visionmobile.com/product/developer-economics-2013-the-tools-report/, Zugriffsdatum: 28.01.2013

[WhatsApp 2013] WhatsApp Inc: WhatsApp, URL: http://www.whatsapp.com/, Zugriffsdatum: 23.02.2013

[Worklight 2012] IBM: IBM Worklight, URL: http://www-01.ibm.com/software/mobile-solutions/worklight/features/ - Zugriffsdatum: 23.09.2012

[Worklight] Worklight: Worklight and PhoneGap – Comparison, Worklight , URL: http://www.worklight.com/assets/files/Worklight%20and%20PhoneGap.pdf, Zugriffsdatum: 21.08.2012

[Xamarin 2013] Xamarin: Hello, Multiscren Applications, Xamarin, URL: http://docs.xamarin.com/Android/Guides/Getting_Started/Hello,_Multi-Screen_Applications#Anatomy_of_an_Android_Application, Zugriffsdatum: 07.02.2013

Hinweise zur CD

Die Inhalte der CD sind wie in der Abbildung 40 strukturiert. Im Folgenden werden die Inhalte der Ordner kurz erläutert:

- 10 Masterarbeit
  - Masterarbeit als PDF-Datei
- 20 Projekte
  - WhatsCapp-Projekt
  - WhatsCappPhoneGap-Projekt
  - WhatCappTitanium-Projekt
- 30 APKs
  - Android Installationsdateien der Projekte
- 40 Analysedokumente
  - Vollständige Analyseergebnisse, die im Kapitel 4.3.1 erwähnten wurden
  - Weitere analyserelevanten Dateien
- 50 Sonstiges
  - Verschiedene Dateien
    - Enterprise Architect
    - Balsamiq
    - Powerpoint
    - Excel
    - u.v.m.

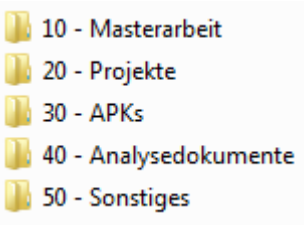

Abbildung 40 Struktur der beiliegenden CD

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 27.03.2013 _____