FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Einsatz von mobilen Plattformen als Visualisierungslösung in der Automatisierungstechnik

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Thomas Chrenko
Matrikelnummer 0728121

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dr. Wolfgang Kastner

Wien, 29.09.2014

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Thomas Chrenko
Speckbachergasse 28/21, 1160 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Danksagung

In erster Linie möchte ich mich bei meinem Betreuer Wolfgang Kastner bedanken. Sein Einsatz hat geholfen, diese Arbeit zu einem guten Ende zu bringen.

Weiter gilt mein Dank all meinen Freunden, die viel Abwechslung in mein Studentenleben gebracht haben.

Bei meinen Eltern Marianne und Norbert möchte ich mich für die Ermöglichung des Studiums und die vorhergehende Ausbildung bedanken. Ebenso konnte ich jederzeit auf die familiäre Unterstützung von ihnen und meinen Brüdern, Andreas und Bernhard, vertrauen.

Zu guter Letzt gilt mein besonderer Dank meiner Freundin Diana, die mich fast mein gesamtes Studium hindurch begleitet hat und auch in lern- und arbeitsintensiven Zeiten zu mir gestanden ist.

Abstract

In this thesis, different development approaches for creating mobile applications are presented and their possible applications for interfacing automation systems are analyzed. Native development, developing web applications, hybrid approaches and model-driven software development are discussed. They will be compared on basis of introduced categories, such as support for native mobile device features, usability or available tool suites. It is shown that native approaches are the best choice for the development of such applications when only one platform has to be supported. However, if the application is dedicated to multiple platforms, developing a web application or using hybrid approaches are better suited for this task, even if they do not reach the usability of native applications. Model-driven approaches are a promising alternative but are not mature enough.

Based on the analysis, the porting of an existing visualization solution of a supervisory control and data acquisition (SCADA) system to mobile platforms is presented. For this effort, the development of a web application is chosen, because no native device features are required for the implementation and as a higher order requirement a variety of platforms should be supported. The developed application is then evaluated in a case study. To illustrate its functionality, an existing visualization solution is compared to the one implemented within the context of this thesis.

Kurzfassung

In dieser Arbeit werden verschiedene Entwicklungsansätze für das Erstellen von mobilen Applikationen vorgestellt und ihre Einsatzmöglichkeiten in der Automatisierungstechnik analysiert. Native Entwicklungsansätze, der Ansatz der Entwicklung von Webapplikationen, hybride Entwicklungsansätze und der Ansatz der modellgetriebenen Softwareentwicklung werden diskutiert. Es werden verschiedene Kategorien, wie die Unterstützung von nativen Gerätefunktionen, die Benutzerfreundlichkeit oder das Entwicklerteam unterstützende Tools, eingeführt und die Ansätze anhand dieser verglichen. Es wird gezeigt, dass native Ansätze die beste Wahl für die Entwicklung solcher Anwendungen sind, wenn nur eine Plattform unterstützt werden soll. Wenn die Applikation jedoch auf mehreren Plattformen lauffähig sein soll, eignen sich Webapplikationen oder hybride Entwicklungsansätze besser für diese Aufgabe, auch wenn sie nicht die Benutzerfreundlichkeit von nativen Applikationen erreichen. Modellgetriebene Ansätze sind sehr vielversprechend für die Entwicklung, es existiert jedoch noch kein produktiv einsetzbarer Ansatz.

Auf der Analyse basierend wird die Portierung einer bereits bestehenden Visualisierungslösung für Stationsleittechnik, Fernwirktechnik und Schutztechnik auf mobile Plattformen vorgestellt. Für diese Portierung wird die Entwicklung einer Webapplikation gewählt, weil keine nativ mobilen Gerätefunktionen für die Implementierung benötigt werden und eine Vielzahl von Plattformen unterstützt werden soll. Die entwickelte Applikation wird anschließend in einer Fallstudie evaluiert und mit der bestehenden Referenzimplementierung verglichen. Die Evaluierung zeigt, dass der entstandene Prototyp funktionsfähig ist und plattformunabhängige Visualisierungslösungen für die Automatisierungstechnik realisiert werden können.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

1.1 Motivation

Automatisierungstechnik unterstützt den Menschen seit Jahrzehnten dabei, Prozesse effektiver zu gestalten. Unter Automatisierungstechnik versteht man alle, in mehrere Ebenen unterteilte, Maßnahmen, die benötigt werden, um eine Maschine oder Anlage zu automatisieren, das heißt, sie ohne menschliches Zutun zu betreiben. Es wird Arbeit vom Menschen zur Maschine verlagert, mit dem Grundgedanken, die Maschine die Arbeit schneller, billiger, genauer und zuverlässiger durchführen zu lassen. Die Einsatzmöglichkeiten der Automatisierungstechnik reichen vom Einsatz in der Produktion oder der Logistik bis hin zur Gebäudeautomation und noch vielen mehr. Um Automatisierung in so vielen unterschiedlichen Gebieten zu ermöglichen, ist ein Zusammenwirken vieler Gebiete, wie bspw. Maschinenbau, Informatik oder Steuer- und Regelungstechnik, vonnöten. [76]

Alle Automatisierungssysteme arbeiten prinzipiell nach demselben Schema. Sie beinhalten für den Prozess essentielle Informationen und die Verarbeitung dieser ist ein grundlegender Bestandteil der Automatisierung. Um diese Informationen zu verwalten, wurden bereits in den siebziger Jahren Anstrengungen unternommen, eine einheitliche Automatisierungsarchitektur zu entwickeln, die alle Abteilungen eines Unternehmens einbezieht. Aus diesen Bestrebungen ging das heute unter dem Namen Automatisierungspyramide bekannte Konzept hervor. Es beschreibt einen hierarchischen Informationsfluss, wobei das Gesamtsystem in mehrere Ebenen unterteilt ist. Das Konzept wurde als solches jedoch nie standardisiert, was dazu führte, dass im Laufe der Zeit unterschiedliche Implementierungen derselben Idee entstanden sind. Diese unterscheiden sich beispielsweise in der Anzahl der Ebenen und der Namensgebung. [76]

Die oberste Schicht der in Abbildung 1.1 dargestellten Automatisierungspyramide ist die Enterprise-Resource-Planning (ERP)-Ebene. Sie beschäftigt sich mit der Integration aller Geschäftsprozesse, wie beispielsweise Einkauf, Produktion, strategische Planung, Finanzen, Verkauf u.v.m. Unterhalb der ERP-Ebene befindet sich die Manufacturing Execution System (MES)-Ebene. Sie beinhaltet, im Gegensatz zur ERP-Ebene, kurzfristige Planung und bildet die Schnittstelle zwischen Büro- und Produktionsbereich. Zu den Aufgaben zählen unter anderem Qua-

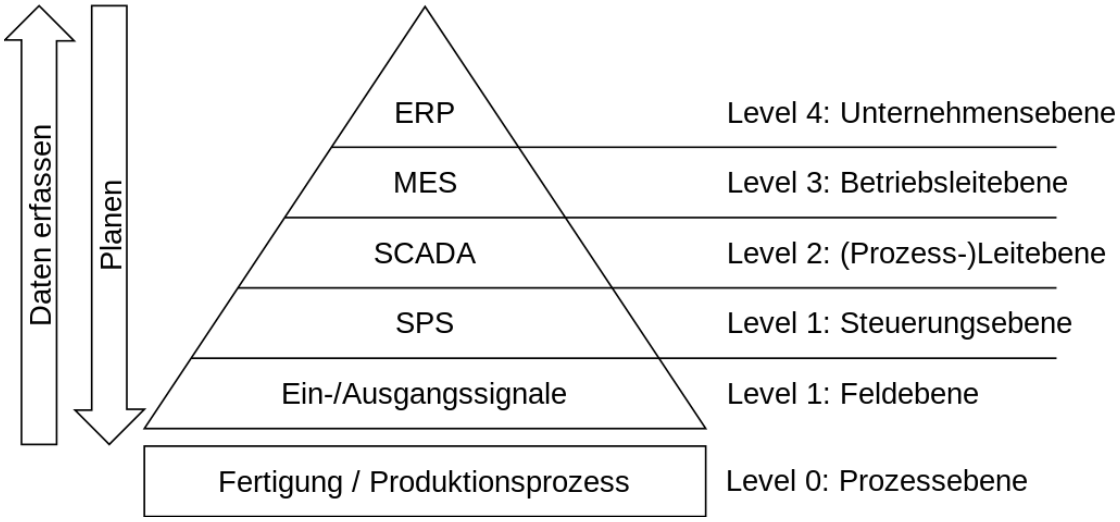

Abbildung 1.1: Automatisierungspyramide mit mehreren Ebenen [84].

litätsmanagement, Prozessmanagement oder Erfolgsanalysen. Die darunter liegenden Ebenen sind stark miteinander verknüpft. Die Feldebene stellt über Ein- und Ausgabesignale die Schnittstelle zum technischen Prozess dar, der mit Hilfe von Sensoren und Aktoren überwacht bzw. geregelt wird. Die gesammelten Daten der Eingänge werden über ein Bussystem an die Steuerungsebene weitergeleitet, wo anhand von einem oder mehreren Datenpunkten (Ist-Werte) und den gewünschten Daten (Soll-Werte), die von der Leitebene vorgegeben werden, ermittelt wird, welche Ausgänge, wie geschaltet werden sollen. Durch dieses Schalten der Ausgänge wird der Prozess gesteuert bzw. geregelt. [76]

Unter dem Begriff Industrie 4.0, wird kürzlich ein Konzept diskutiert, das große Auswirkungen auf den zuvor beschriebenen Automatisierungsvorgang haben wird [34]. Der Begriff Industrie 4.0 steht für die vierte industrielle Revolution und folgt somit den drei vorangegangen industriellen Revolutionen (siehe Abbildung 1.2). Als erste industrielle Revolution wird der, seit dem Ende des 18. Jahrhunderts betriebene, großflächige Einsatz von mechanischen Produktionsanlagen bezeichnet, die mithilfe von Wasser- und Dampfkraft betrieben wurden. Die zweite industrielle Revolution wurde Anfang des 20. Jahrhunderts durch weitverbreiteten Gebrauch elektrischer Energie eingeläutet, durch die Massenproduktion ermöglicht wurde. Als dritte industrielle Revolution gilt der, seit dem Anfang der Siebzigerjahre vollzogene Einsatz von speicherprogrammierbaren Steuerungen in der Automatisierung der Produktion. [3, 60]

Das Ziel der vierten industriellen Revolution ist es nun, Produkte und Dienstleistungen mithilfe von Software und Elektronik miteinander und mit ihrer Umgebung zu verknüpfen. Durch diese Verknüpfung sollen neue, intelligente Produkte und Dienstleistungen entstehen, die autonom Informationen austauschen und sich gegenseitig steuern können. Die intelligenten Produkte können jederzeit eindeutig identifiziert werden und kennen ihren aktuellen Zustand, sowie alternative Wege, um ihren Zielzustand zu erreichen. Sie sind mit betriebswirtschaftlichen Prozessen

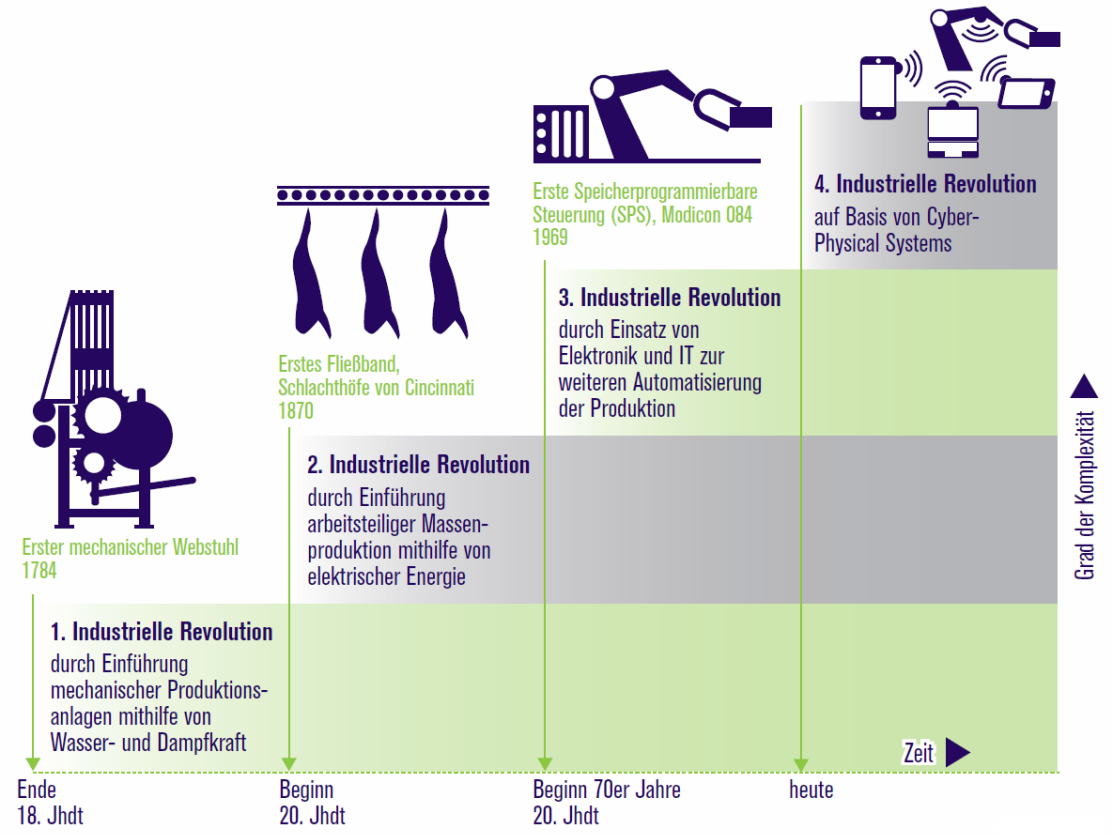

Abbildung 1.2: Die vier industriellen Revolutionen [16].

verknüpft und an verteilte, in Echtzeit steuerbare Wertschöpfungsnetzwerke angebunden. Somit können Geschäfts- und Entwicklungsprozesse dynamisch gestaltet werden, um kurzfristig auf Veränderungen zu reagieren. [60]

Diese Dezentralisierung und Autonomie soll mithilfe von Cyber-Physical Systemss (CPSs) erreicht werden. Als CPS werden vernetzte Komponenten bezeichnet, die eingebettete Systeme enthalten, um kommunikationsfähig zu werden. Sie überwachen und steuern bzw. regeln ihre Umwelt und andere CPS mit Sensoren und Aktoren. Durch den Zugriff auf das Internet können sie auf weltweit verfügbare Daten und Dienste zugreifen, um gesammelte Daten auszuwerten, zu speichern und anhand gewonnener Informationen effektiver zu arbeiten. [3, 4]

Die vierte industrielle Revolution ist mit ihren CPSs stark an das Konzept des Internet of Things (IoT) gekoppelt. Unter IoT wird nicht mehr nur die Vernetzung von Menschen mithilfe von Computern verstanden, sondern auch die Vernetzung von eindeutig identifizierbaren physischen Objekten. Menschen und Objekte sind somit in einem gemeinsamen Netzwerk verbunden, wodurch Menschen bei ihren Tätigkeiten von den Objekten unterstützt werden sollen [4, 60]. Mobile Plattformen bieten hier eine Möglichkeit, das Vernetzen von Menschen und Objekten voranzutreiben. Sie sind schon jetzt in Form von Smartphones und Tablets aus unse-

rem Alltag nicht mehr wegzudenken und erleichtern uns vielzählige Aufgaben. Ausgestattet mit unterschiedlichen Kommunikationstechnologien lassen sie sich mit vergleichsweise geringem Aufwand in bestehende IT-Infrastruktur einbinden und sind bestens für den Zugriff auf Dienste im Internet gerüstet. Sie bilden somit eine solide Ausgangsbasis, um einige der Bestrebungen der vierten industriellen Revolution umzusetzen.

1.2 Problemstellung und Ziel

Mobile Plattformen werden aktuell kaum in der Automatisierungstechnik eingesetzt, obwohl sie allgegenwärtig sind. Viele Automatisierungssysteme basieren auf alten, stellenweise überholten oder proprietären Technologien, die einen Einsatz von mobilen Plattformen zusätzlich erschweren. Desweiteren werden oft die von mobilen Plattformen zur Verfügung gestellten Kommunikationsmöglichkeit nicht bzw. nicht ausreichend unterstützt. Um das zu ändern, müssen Ansätze aufgezeigt werden, um Applikationen für mobile Plattformen zu entwickeln, die in der Automatisierungstechnik eingesetzt werden können.

Üblicherweise werden in der Automatisierungstechnik Bedieneinheiten eingesetzt, um vor Ort Informationen über den überwachten/gesteuerten Prozess bzw. die Automatisierungseinheit selbst zu visualisieren, Parametereinstellungen vorzunehmen und per manueller Steuerung in den Prozess einzugreifen. Zu diesen Informationen zählen unter anderem die Darstellung des Prozesses, das Visualisieren von Messwerten oder das Anzeigen von Ereignis- und Fehlerlisten. Das Ziel dieser Arbeit ist es, zu evaluieren, welche Entwicklungsansätze für das Erstellen von mobilen Applikationen für die Automatisierungstechnik eingesetzt werden können und welche Vor- bzw. Nachteile diese Ansätze mit sich bringen. Dabei gilt es, Anforderungen, wie die Anzeige im Webbrowser, die Verwendung von Touch-Paneelen für die Benutzerinteraktion oder die meist knappen Ressourcen auf Seiten der Automatisierungseinheit zu beachten und die in dieser Branche üblichen langen Lebenszyklen solcher Systeme nicht zu vernachlässigen. Als Nachweis der Machbarkeit soll eine bestehende Visualisierungslösung für Stationsleittechnik, Fernwirktechnik und Schutztechnik anhand der theoretischen Erkenntnisse auf mobile Plattformen portiert werden. Dabei sollen allgemein anwendbare Erkenntnisse gewonnen werden, die als Grundlage bzw. Orientierungshilfe für das Entwickeln von mobilen Applikationen für die Automatisierungstechnik verwendet werden können. Diese Erkenntnisse sollen dadurch gewonnen werden, dass die Ergebnisse des theoretischen und praktischen Teils dieser Arbeit grundsätzliche Designentscheidungen, wie den zu wählenden Entwicklungsansatz, erleichtern und somit verkürzen sollen.

1.3 Methodik

Zu Beginn der Arbeit werden die theoretischen Grundlagen für die Entwicklung von mobilen Applikationen für die Automatisierungstechnik in Literatur und Internet recherchiert. Dabei sollen im ersten Schritt bereits existierende oder gerade entstehende Ansätze für die Entwicklung von mobilen Applikationen gefunden werden.

Im nächsten Schritt müssen gefundene Ansätze auf ihre Einsatzmöglichkeiten in der Automatisierungstechnik analysiert und ihre Vor- bzw. Nachteile aufgezeigt werden. Bei diesen

Analysen müssen unter anderem die in der Problemstellung erwähnten Anforderungen an solche Systeme stets beachtet werden.

Anschließend muss anhand der zuvor erarbeiteten Ergebnisse ein Ansatz gewählt werden, auf dessen Basis eine bestehende Visualisierungslösung auf mobile Plattformen portiert wird. Dieses Vorhaben soll mit Hilfe von Techniken wie Unified Modeling Language (UML) unterstützt werden.

Abschließend muss die entwickelte Applikation anhand einer Fallstudie evaluiert und somit die Einsetzbarkeit von mobilen Plattformen in der Automatisierungstechnik gezeigt werden.

1.4 Aufbau der Arbeit

In Kapitel 2 wird der Stand der Technik zur Entwicklung von mobilen Applikationen beschrieben. Es werden zuerst native Entwicklungsansätze, die auf eine Plattform zugeschnitten sind, diskutiert, gefolgt vom Ansatz der Entwicklung von Webapplikationen. Anschließend werden hybride Ansätze, die native Eigenschaften mit plattformunabhänigen kombinieren, vorgestellt und das Kapitel mit der modellgetriebenen Entwicklung von Applikationen abgeschlossen. Es werden für jeden der Ansätze die eingesetzten Technologien und unterstützenden Werkzeuge vorgestellt.

In Kapitel 3 werden Probleme, die bei der Entwicklung von mobilen Applikationen auftreten können, und Anforderungen an Entwicklungsansätze diskutiert. In diesem Kontext werden anschließend die vorher vorgestellten Entwicklungsansätze analysiert und abschließend verglichen.

Kapitel 4 ist, aufbauend auf den zuvor erlangten Erkenntnissen, der Implementierung einer bestehenden Visualisierungslösung auf einer mobilen Plattform gewidmet. Es wird das bestehende Visualisierungssystem im Detail vorgestellt und anschließend der Entwurf und die Implementierung der im Zuge dieser Arbeit entwickelten Applikation beschrieben.

Die entwickelte Applikation wird in Kapitel 5 anhand einer Fallstudie evaluiert und die dabei erlangten Ergebnisse präsentiert.

Die Arbeit wird in Kapitel 6 mit einer Zusammenfassung, dem Aufzeigen von verwandten Arbeiten und einem Ausblick, auf im Zuge dieser Arbeit nicht behandelte Themen, abgeschlossen.

KAPITEL 2

State of the Art

2.1 Übersicht

Dieses Kapitel umfasst die Grundlagen und den theoretischen Hintergrund, die benötigt werden, um mobile Applikationen zu erstellen. Dazu werden im Unterkapitel 2.2 zuerst die mobilen Plattformen Android, iOS und Windows Phone im Detail behandelt, aber auch andere Plattformen, wie beispielsweise Blackberry 10 oder Tizen, vorgestellt.

Mobile Applikationen können auch auf Basis von Web-Technologien umgesetzt werden, weshalb in Kapitel 2.3 diese Art der Entwicklung vorgestellt wird. Dabei werden die wichtigsten technologischen Grundlagen für die Entwicklung von Webapplikationen behandelt. Zu diesen zählen unter anderem HyperText Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript.

Kapitel 2.4 beschäftigt sich mit Entwicklungsansätzen, die es sich zum Ziel gemacht haben, mit Hilfe von Web-Technologien und nativen Ansätzen eine Codebasis zu schaffen, die auf mehreren Plattformen lauffähig ist.

Abschließend behandelt Kapitel 2.5 die Applikationsentwicklung auf Basis von Modellen. Dabei werden signifikante Teile der Applikation als Modell realisiert, aus welchem per Transformation Quellcode erzeugt wird.

2.2 Native Applikationen

Native Applikationen sind Applikationen, die nur für eine einzige Zielplattform entwickelt wurden und somit nur auf dieser einen Zielplattform ausführbar sind.

Im Folgenden wird im Detail auf verschiedene mobile Plattformen eingegangen. Dabei wird für alle der vorgestellten Plattformen unter anderem geklärt, mit welcher Programmiersprache sich Anwendungen für diese Plattform entwickeln lassen und welche Tools den Entwickler dabei zur Verfügung stehen.

2.2.1 Android

Android ist ein Betriebssystem für mobile Geräte, das von der Open Handset Alliance (OHA) unter der Leitung von Google im Rahmen des Android Open Source Project (AOSP) entwickelt wird.

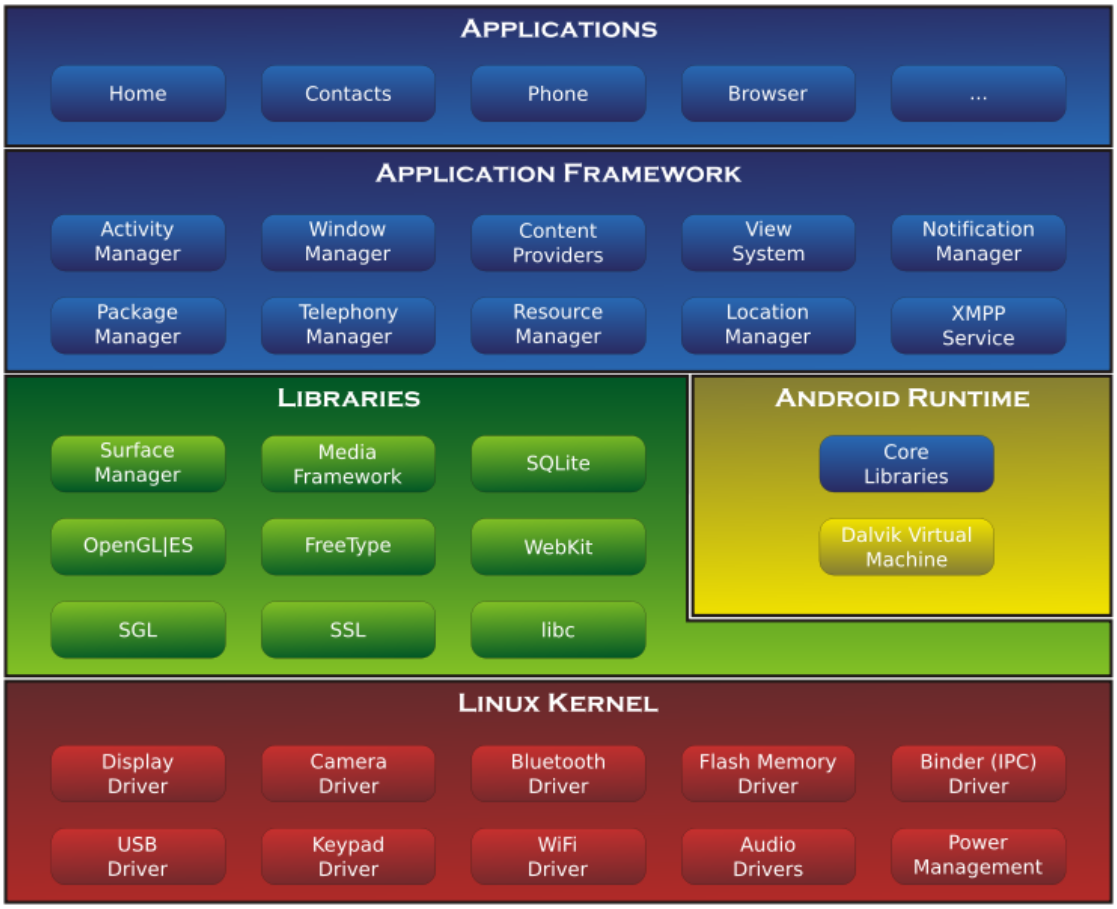

Abbildung 2.1: Android Architektur [52].

Architektur

Android basiert auf einem für den Einsatz auf mobilen Geräten optimierten Linux-Kernel, der die Treiber für die Kommunikation mit der Hardware enthält. Neben dem Linux Kernel setzt Android zu einem großen Teil auf quelloffene Bibliotheken. Gleichzeitig ist der Quelltext von Android frei verfügbar und fast vollständig unter der Apache Software License 2.0 (ASL2.0) lizenziert [51]. Android kann somit von verschiedenen Herstellern, ohne Lizenzgebühren bezahlen zu müssen, auf ihren Geräten eingesetzt werden, wodurch die Anzahl der unterschiedlichen Android-Geräte sehr hoch ist. Eine der wichtigsten Komponenten unter Android stellt die

Android Runtime dar, die aus Kernbibliotheken, basierend auf der Java Application Programming Interface (API), und der Dalvik Virtual Machine besteht. Die Dalvik Virtual Machine setzt auf Just-in-time-Compilierung und benutzt ein optimiertes Dateiformat (.dex), das aus Java-Bytecode erzeugt und für die Portabilität der Anwendungen zwischen den unterschiedlichen Geräten zuständig ist. Auf Basis der bisher genannten Komponenten läuft das Application Framework, das aus Plattform- und Hardware-Services besteht. Die Plattform-Services sind unter anderem für die Steuerung des Lebenszyklus von Applikationen oder die Ressourcenverwaltung zuständig. Die Hardware-Services regeln den Zugriff der Applikationen auf die Hardware (z.B. GPS, Telefon). Abbildung 2.1 stellt die beschriebene Architektur dar. [52]

An dieser Stelle sei noch erwähnt, dass Google mit der Veröffentlichung von Android 4.4 die experimentelle Android Runtime Android RunTime (ART) vorgestellt hat, die in naher Zukunft die Dalvik Virtual Machine ersetzen soll. Eine der größten Änderung in ART ist der Wechsel zu einer Ahead-of-time-Compilierung. [53]

Sicherheitsaspekte im Zusammenhang mit der Architektur von Android werden von Klement in [62] behandelt. Dabei wurde das Sicherheitskonzept von Android von Schwachstellen innerhalb des Kernels, über Open-Source Software (OSS)-Bibliotheken bis hin zu den Anwendungsrechten untersucht.

Programmiersprachen

Android Applikationen werden mit der Programmiersprache Java entwickelt. Durch die bereits erwähnte Tatsache, dass Kernbibliotheken von Android auf der Java API basieren, können Entwickler, die bereits mit der Programmiersprache Java vertraut sind, in gewohnter Weise ihre Applikationen entwickeln. Der in Java geschriebene Programmcode wird anschließend in Java-Bytecode übersetzt, der wiederum in das für die Dalvik Virtual Machine lesbare Format übersetzt wird.

Android bietet jedoch auch die Möglichkeit nativen C bzw. C++ Code für die Entwicklung von Anwendungen zu verwenden. Google stellt für diesen Zweck das Android Native Development Kit (NDK) zur Verfügung. Es wird jedoch explizit darauf hingewiesen, dass die Komplexität der Anwendung durch den Einsatz des NDKs erhöht wird. Desweiteren wird empfohlen, das NDK nur für sehr Central Processing Unit (CPU)-lastige Aufgaben einzusetzen. Alle Informationen zum Einsatz von nativen Code unter Zuhilfenahme des Android NDK sind unter [50] zu finden.

Entwicklungsumgebung und Tools

Die Grundlage für die Entwicklung von Android-Anwendungen ist das Android Software Development Kit (SDK). Es liefert dem Entwickler die besagten Kernbibliotheken und beinhaltet außerdem alle benötigten Werkzeuge für das Erstellen, Testen und Debuggen von Android-Applikationen. Zu diesen zählen unter anderem ein Emulator, der es ermöglicht, unterschiedliche Gerätetypen zu simulieren und geschriebene Applikationen auf diesen simulierten Geräten auszuführen. [45] [10]

Als Entwicklungsumgebung für Android hat Google das Android Developer Tools (ADT)-Plug-In für Eclipse entwickelt. Es erweitert die Eclipse-Integrated Development Environment

(IDE) dahingehend, dass Android-Projekte erstellt und verwaltet werden können, indem es die Android-SDK Komponenten integriert. Außerdem bietet es neben einem Designer zum Gestalten der Benutzeroberfläche auch die Möglichkeit, die gerade in Entwicklung befindliche Applikationen auf dem Emulator auszuführen. Es kann jedoch auch direkt jedes Android-fähige Gerät für die Entwicklung verwendet werden. [49]

Auch an dieser Stelle sei erwähnt, dass Google mit Android Studio, das auf IntelliJ IDEA basiert, bereits eine neue IDE vorgestellt hat, die in Zukunft Eclipse mit dem ADT-Plug-In ersetzen soll [48].

In [10] beschreiben Chang et al. die Entwicklung einer nativen Android Applikation im Detail. Dabei decken sie Themen wie das Erstellen von Projekten in der IDE bis hin zur Ereignisbehandlung innerhalb der Applikation ab.

Verbreitung

Laut [44] ist Android die mobile Plattform mit dem größten Marktanteil im Smartphone- und Tablet-Markt, weshalb es unabdingbar ist, für Android zu entwickeln, wenn eine Applikation für eine Vielzahl von Personen verfügbar sein soll. Dabei sollte jedoch die Tatsache nicht außer Acht lassen, dass es innerhalb der Android-Plattform zu einer Fragmentierung gekommen ist und somit beachtet werden muss, für welche Android-Version eine Applikation angeboten werden soll [47]. Abbildung 2.2 zeigt diese Fragmentierung der Android Versionen, die dadurch entsteht, dass Gerätehersteller Android mit eigenen Anpassungen erweitern und dann selbst entscheiden, welche Modelle sie mit einem Update versorgen möchten. Zusätzlich können auch Anpassungen der Mobilfunkanbieter und Softwaretests ein Update hinauszögern [36].

Mit Android 4 hat Google das Android SDK um die Support-Library erweitert. Diese ermöglicht es Applikationen, die für eine bestimmte API-Version entwickelt wurden, auch auf niedrigeren API-Versionen auszuführen [46].

Version	Codename	API	Distribution
2.2	Froyo	8	0.8%
2.3.3 - 2.3.7	Gingerbread	10	14.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	12.3%
4.1.x	Jelly Bean	16	29.0%
4.2.x		17	19.1%
4.3		18	10.3%
4.4	KitKat	19	13.6%

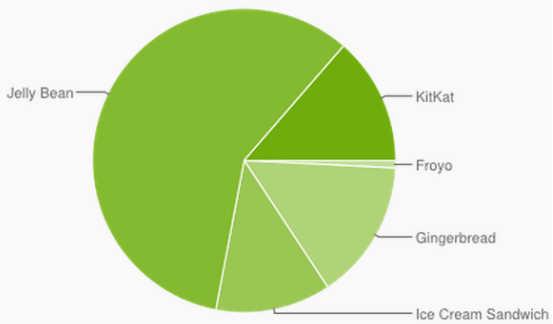

Abbildung 2.2: Zugriffe auf den Google Play Store, aufgeschlüsselt nach Android Version [47].

2.2.2 iOS

iOS ist ein von Apple entwickeltes Betriebssystem für die Produktlinien iPhone, iPad, iPod Touch und Apple TV und ist nur auf Geräten aus diesen Produktlinien lauffähig. Es wurde 2007 zusammen mit dem ersten iPhone vorgestellt.

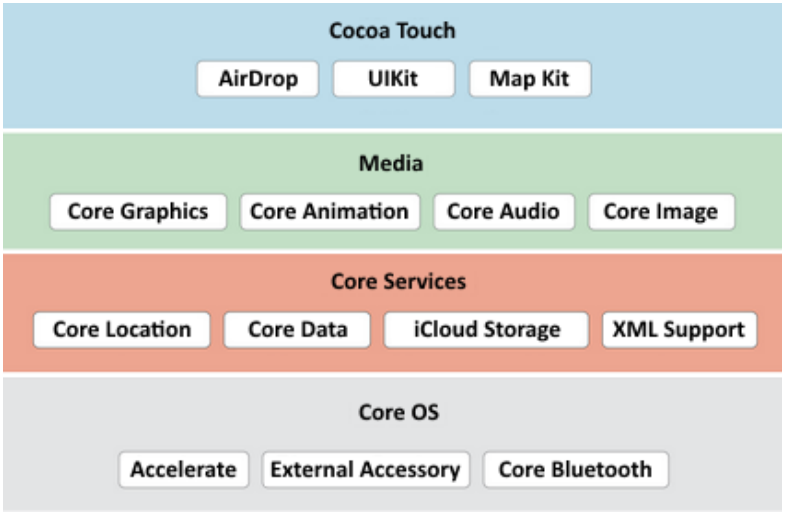

Abbildung 2.3: iOS Architektur [43].

Architektur

Die Architektur von iOS setzt sich aus vier Schichten zusammen (siehe Abbildung 2.3).

Core OS ist die unterste Schicht in der Architektur von iOS. Sie ist der Hardware am nächstgelegen und bildet somit die niedrigste Abstraktionsebene. Die Core OS-Schicht beinhaltet folgende Komponenten:

- System Framework, das die Kernel-Ebene, Treiber und UNIX-Schnittstellen umgibt, und somit Zugriffe auf unterster Ebene auf das Betriebssystem ermöglicht.
- Accelerate Framework für die digitale Signalverarbeitung und um Berechnungen der linearen Algebra oder für die Bildverarbeitung durchzuführen. Dabei sind diese Funktionen stark für die darunterliegende Hardware optimiert.
- Core Bluetooth Framework zur Interaktion mit Bluetooth Low Energy Geräten.
- Security Framework, mit dem die Sicherheit der von der Applikation verwalteten Daten sichergestellt wird. Es stellt Schnittstellen für die Verwaltung von Zertifikaten, öffentlichen und privaten Schlüsseln und Sicherheitsrichtlinien zur Verfügung und ist für die Generierung von Pseudozufallszahlen zuständig.

Core Services ist die über Core OS liegende Schicht und beinhaltet essentielle Dienste, die für die Entwicklung von Anwendungen notwendig sind. Zu diesen zählen:

- Core Location Framework für den Zugriff auf alle standortrelevanten Daten.
- SQLite, um Datenbanken in eine Applikation einzubetten.
- XML Support für die Manipulation von XML-Dateien.
- iCloud Storage für Zugriffe auf Apples iCloud Lösung.
- Automatic Reference Counting (ARC), um dem Entwickler das Speichermanagement zu erleichtern.

Media ist die dritte Schicht und unterteilt sich in verschiedene Komponenten, um die Entwicklung grafischer, akustischer und visueller Inhalte zu vereinfachen. Zu diesen gehören unter anderem:

- Core Graphics Framework für die dynamische Darstellung von zweidimensionalen Formen und Bildern.
- Core Animation Framework zum Steuern des Animationsverhaltens.
- Image I/O für das Lesen und Schreiben der meisten Grafikformate.
- Assets Library für den Zugriff auf Benutzerinhalte.
- Core Audio für das Aufnehmen und Abspielen von Audioinhalten.
- Media Library Framework, das den Zugriff auf die iTunes Bibliothek eines Benutzers steuert.
- Media Player für das Abspielen von Videoinhalten.
- Air Play, um Multimediainhalte zu Apple TV oder Geräten von Dritten zu streamen.

Cocoa Touch ist die höchste Schicht in der iOS Architektur und beinhaltet zentrale Frameworks, die das Erscheinungsbild der Applikation und somit die grundlegende Anwendungsinfrastruktur beeinflussen. Cocoa Touch untergliedert sich insbesondere aber nicht ausschließlich in folgende Komponenten:

- AirDrop, um Daten mit anderen Geräten auszutauschen.
- Multitasking, das für die Steuerung des Multitaskings verantwortlich ist, indem beispielsweise Hintergrundprozesse um Rechenzeit für bestimmte Tätigkeiten anfragen.
- Local Notifications, um den Benutzer über Neuigkeiten innerhalb der eigenen Applikationen zu informieren.
- Apple Push Notification Service, um serverseitig generierte Benachrichtigungen an den Benutzer zu senden.
- Gesture Recognizers, die es dem Entwickler auf einfachem Weg ermöglichen, Gesten zu erkennen, die auf Touch-Events basieren.
- Standard System View Controllers, um vordefinierte Standardansichten in die eigene Applikation einzubinden.

- Map Kit Framework, über das eine interaktive Karte in der Applikation verwendet werden kann.
- UIKit Framework, das die wesentliche Infrastruktur zur Verfügung stellt, um grafische, ereignisgesteuerte Anwendungen zu entwickeln.

Miller et al. bieten in [72] einen tief greifenden Einblick in sicherheitsrelevante Themen innerhalb der iOS Plattform. Dabei werden wichtige Aspekte in der iOS-Sicherheitsarchitektur, wie beispielsweise Speicherschutz, Verschlüsselung oder Code-Signierung, behandelt.

Programmiersprachen

iOS-Applikationen werden in den Programmiersprachen Objective-C und Swift[1] geschrieben. Objective-C erweitert die Programmiersprache C um objektorientierte Sprachkonzepte und eine dynamische Laufzeitumgebung. Swift wurde mit dem Gedanken entwickelt, moderne Sprachkonzepte in eine für Objective-C Entwickler, aber auch für Einsteiger, leicht zu verwendende Programmiersprache zu integrieren [40]. Swift kann ab iOS-Version 7 für die Entwicklung von Applikationen eingesetzt werden [28]. Der Programmcode wird, unabhängig von der Programmiersprache, in Binärcode übersetzt.

Auch iOS bietet die Möglichkeit Programmcode in C/C++ zu schreiben. Dieser wird ebenfalls direkt in Binärcode übersetzt.

Entwicklungsumgebung und Tools

Um Applikationen für iOS zu entwickeln, wird das iOS SDK benötigt. Es beinhaltet die auf Seite 11 erwähnten Frameworks, die Schnittstellen für grundlegende Funktionen zur Verfügung stellen. Außerdem beinhaltet das iOS SDK alle benötigten Werkzeuge für das Erstellen, Testen und Debuggen von iOS-Applikationen. Gesondert sei hier der iOS Simulator erwähnt, der, wie der Emulator im Android-SDK, Geräte simulieren kann, auf denen Applikationen ausgeführt werden. [43]

Apple stellt als Entwicklungsumgebung für iOS die Xcode IDE zur Verfügung. Um Xcode zu verwenden, wird zwingend ein Intel-basierter Mac mit mindestens OS X 10.6 als Betriebssystem und eine Registrierung als Apple Developer benötigt. [42]. Xcode bietet neben typischen IDE-Funktionen, wie Code-Vervollständigung oder dem Setzen von Haltepunkten für das Debuggen, auch für iOS spezifische Funktionalität. Dazu zählt unter anderem der Interface Builder, mit dem grafische Benutzeroberflächen für iOS-Anwendungen erstellt werden können.

Einen Vergleich zwischen der Applikationsentwicklung unter Android und iOS geben Goadrich und Rogers in [26]. Damit wollen sie die Frage, ob Android- oder iOS-Entwicklung unterrichtet werden sollte, beantworten.

[1] Swift wurde erstmals auf der Apple Worldwide Developer Conference 2014 vorgestellt.

Verbreitung

iOS ist nach Android die mobile Plattform mit dem zweitgrößten Marktanteil im Smartphone- und Tablet-Markt [44] und sollte, wie Android, bei der Auswahl der Zielplattform für eine Applikation auf jeden Fall berücksichtigt werden.

Wie bereits erwähnt, ist iOS nur auf Apples eigenen Geräten lauffähig, wodurch sich die Gerätevielfalt in Grenzen hält. Deshalb läuft auf den meisten Geräten eine aktuelle iOS-Version (siehe Abbildung 2.4), wodurch sich die Auswahl erleichtert, für welche iOS-Version eine Applikation entwickelt wird. [41].

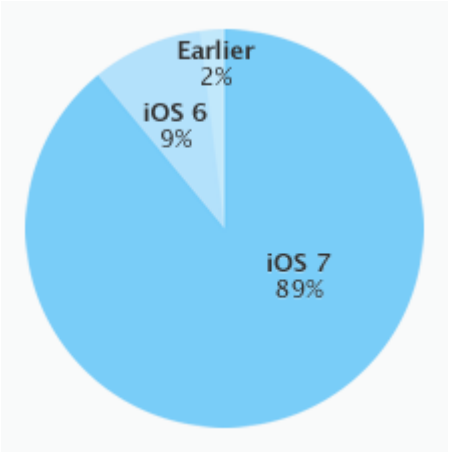

Abbildung 2.4: Zugriffe auf den App Store, aufgeschlüsselt nach iOS Version [41].

2.2.3 Windows Phone

Windows Phone ist der von Microsoft entwickelte Nachfolger des mobilen Betriebssystems Windows Mobile und wurde 2010 vorgestellt.

Architektur

Die Architektur von Windows Phone setzt sich aus verschiedenen Komponenten zusammen (siehe Abbildung 2.5):

Base OS Service kann als eine Ansammlung von Treibern und Bibliotheken verstanden werden, die für die Kommunikation mit der Hardware zuständig sind und Schnittstellen für höhere liegende Komponenten liefern.

Platform Services sind Kernkomponenten, die bestimmte Aufgaben des Betriebssystems ausführen:

- Package Manager ist neben dem Installieren und Deinstallieren von Applikationen für die Verwaltung der Metadaten von Applikationen zuständig.

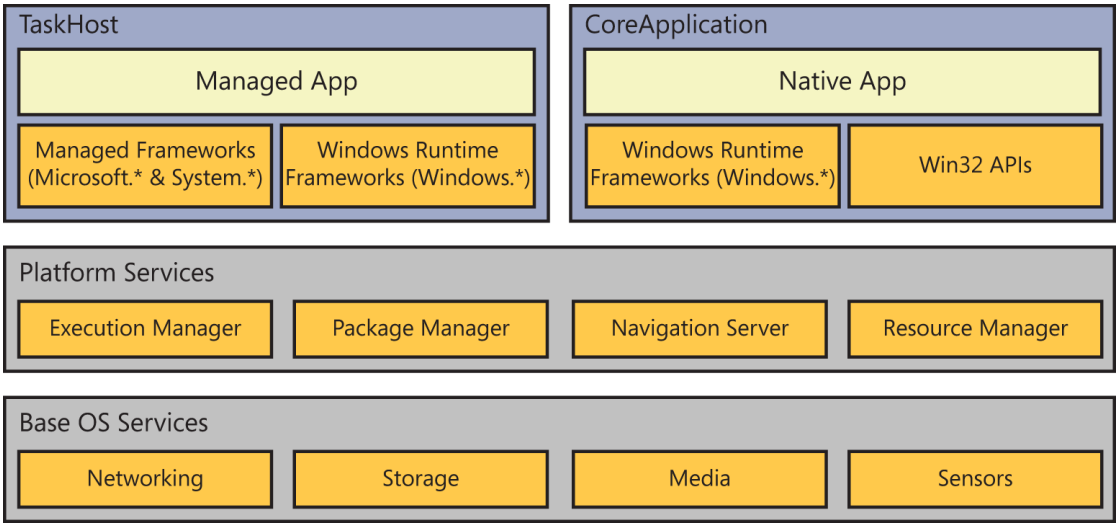

Abbildung 2.5: Windows Phone Architektur [54].

- Execution Manager ist für den Lebenszyklus einer Anwendung und den damit verbunden Ereignissen zuständig.
- Navigation Server steuert den Wechsel zwischen unterschiedlichen Anwendungen.
- Resource Manager verwaltet die Systemressourcen (hauptsächlich CPU und Arbeitsspeicher).

Task-Host ist für die Ausführung von Managed Code[2] verantwortlich und beinhaltet die Managed Frameworks, die dem Entwickler Zugriff auf unterschiedliche Funktionen bieten.

CoreApplication steuert die Ausführung von nativem Code und basiert auf den Frameworks der Windows Phone Runtime.

Programmiersprachen

Anwendung für Windows Phone können in den Programmiersprachen C#, Visual Basic .NET, C++/CX[3] oder einer Kombination von HTML und JavaScript geschrieben werden. Somit besteht für Entwickler die Möglichkeit, die zu verwendende Programmiersprache nach eigenem Kenntnisstand und Anforderungen auszuwählen, um beispielsweise grafisch intensive Anwendungen in C++ zu realisieren [68].

Entwicklungsumgebung und Tools

Das Windows Phone SDK bildet, wie auch die SDKs von Android und iOS, die Grundlage für die Entwicklung von Anwendungen für die Windows Phone Plattform. Es beinhaltet viele der

[2] Als Managed Code wird jener Code bezeichnet, der über die Common Language Runtime ausgeführt wird [27].
[3] C++/CX ist eine Erweiterung von C++, die die Erstellung von Windows Store Applikationen und Windows Runtime-Komponenten ermöglicht [70].

auf Seite 14 erwähnten Schnittstellen, die unterschiedliche Dienste für Entwickler zur Verfügung stellen. Um das SDK zu installieren, wird ein PC mit Windows 8 in der 64-Bit Variante benötigt.

Als Entwicklungsumgebung dient das von Microsoft entwickelte Visual Studio in der Version 2012 oder höher. Wenn bereits eine Visual Studio Version vorhanden ist, wird diese durch die Installation des Windows Phone SDKs erweitert, um Windows Phone Projekte erstellen und verwalten zu können. Sollte noch keine Installation vorhanden sein, wird durch die Installation des Windows Phone SDKs eine im Funktionsumfang begrenzte Express Version von Visual Studio mitgeliefert.

Ebenfalls im Windows Phone SDK bzw. Visual Studio enthalten ist Blend, eine Sammlung von Designtools, die für die Erstellung grafischer Benutzeroberflächen verwendet wird [71].

Auch für Windows Phone gibt es einen Emulator, der Windows Phone-Geräte simulieren kann und auf dem geschriebene Applikationen getestet werden können [69].

Verbreitung

Windows Phone war im vergangenen Jahr die mobile Plattform mit dem drittgrößten Marktanteil [44], wobei der stärkste Markt in Europa lag [61]. Aufgrund dieser Verbreitung sollte Windows Phone als Zielplattform für eine Applikation auf jeden Fall in Betracht gezogen werden.

Durch die strikten Vorgaben, welche Ausstattung Geräte mit Windows Phone mitbringen müssen, können Updates für Geräte schnell ausgeliefert werden, wodurch wiederum die Versionsverteilung von Windows Phone gering ist (siehe Abbildung 2.6) [1].

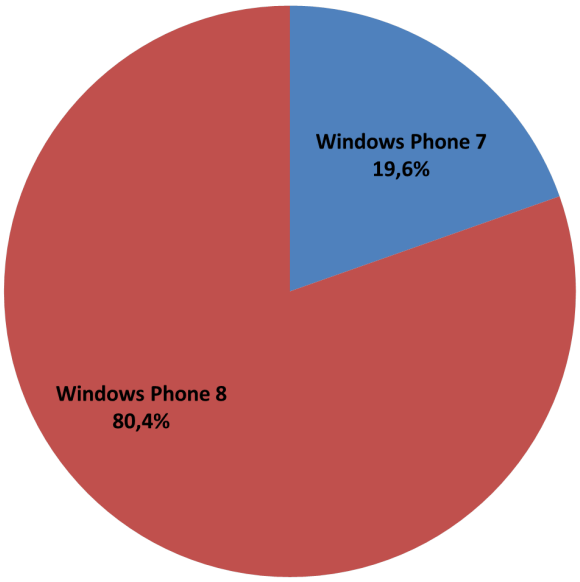

Abbildung 2.6: Windows Phone-Versionsverteilung, ermittelt mithilfe von Applikationen, welche das AdDuplex SDK v.2 verwenden. [1].

2.2.4 Sonstige Plattformen

In diesem Kapitel werden kurz andere mobile Plattformen vorgestellt, für die sich das Entwickeln jedoch aus ökonomischer Sicht, mit Ausnahme von Blackberry 10, nicht rentiert [44].

BlackBerry 10

BlackBerry 10 ist ein von BlackBerry, vormals bekannt als Research In Motion, entwickeltes Betriebssystem, das als Basis einen QNX Neutrino Real-time Operating System (RTOS) Microkernel besitzt (siehe Abbildung 2.7) [67]. Es bietet dem Entwickler standardmäßig die Möglichkeit aus mehreren Entwicklungsansätzen für die Applikationsentwicklung zu wählen:

Native SDK ermöglicht es, native Anwendungen für BlackBerry 10 mit den Programmiersprachen C und C++ zu programmieren. Dafür stellt BlackBerry die Momentics IDE zur Verfügung, die auf Eclipse basiert [66].

WebWorks basiert auf Apache Cordova, das in Kapitel 2.4.1 näher behandelt wird. Damit ist es möglich, Anwendungen mithilfe von Webtechnologien, wie HTML und JavaScript, zu erstellen [65].

BlackBerry SDK for Adobe Integrated Runtime (AIR) erlaubt es, Anwendungen mit Adobe ActionScript und den Adobe Flex APIs zu erstellen [5].

Runtime for Android besteht aus mehreren Plug-Ins und unterstützt den Entwickler, bestehende Android-Applikationen auf BlackBerry 10 Geräten lauffähig zu machen.

Tizen

Tizen ist eine auf dem Linux Kernel basierende mobile Plattform für eine Vielzahl unterschiedlicher Geräte und liegt als Tizen Project innerhalb der Linux Foundation, wo es von einer Technical Steering Group (TSG) geleitet wird, zu der unter anderem Intel und Samsung gehören [83].

Das Tizen SDK liefert die für die Entwicklung notwendige Tizen IDE, die auf den Eclipse JavaScript Development Tools (JSDT) und C/C++ Development Tools (CDT) basiert. Die Architektur von Tizen unterstützt standardmäßig zwei Arten von Applikationen:

Über das **Web Framework** lassen sich Webanwendungen ausführen, die mit den in Kapitel 2.3 beschriebenen Technologien entwickelt wurden [82].

Das **Native Framework** ist für das Ausführen von in C und C++ geschriebenen nativen Anwendungen zuständig. [81].

In der Architektur (siehe Abbildung 2.8) unterhalb der beiden Frameworks liegen der Core, der für beide Frameworks benötigte Dienste über Schnittstellen zur Verfügung stellt, und der Kernel zusammen mit den Gerätetreibern [80].

Jaygarl et al. geben in [56] tiefere Einblicke in die Applikationsentwicklung für Tizen.

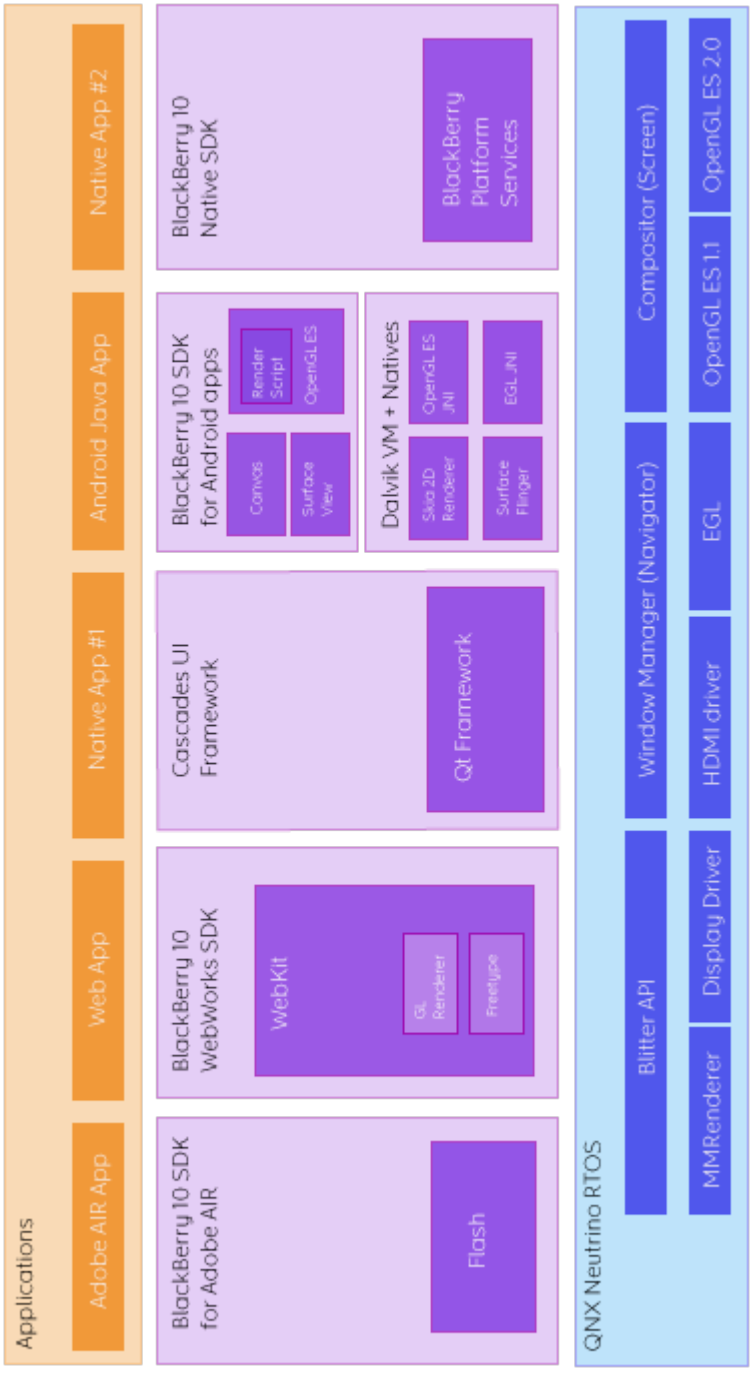

Abbildung 2.7: Blackberry 10 Architektur [67].

Abbildung 2.8: Tizen Architektur [80].

2.3 Webapplikationen

Webapplikationen sind Anwendungen, die innerhalb eines Webbrowsers ausgeführt werden und somit auf einer Vielzahl unterschiedlicher Geräte und Plattformen lauffähig sind. Aufgrund dieser Tatsache bieten Webapplikationen ein einheitliches Aussehen der Anwendung über verschiedene Plattformen hinweg.

Die folgenden Kapitel beschäftigen sich mit den Technologien, die für die Entwicklung von Webanwendungen benötigt werden.

2.3.1 Model-View-Controller-Entwurfsmuster

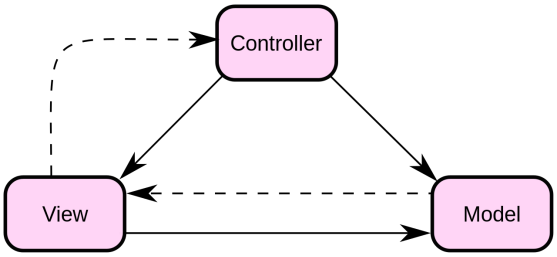

Abbildung 2.9: „Model-View-Controller-Konzept. Die durchgezogene Linie symbolisiert eine direkte Assoziation, die gestrichelte eine indirekte Assoziation (zum Beispiel über einen Beobachter).“ [15]

Mit dem Model-View-Controller (MVC)-Entwurfsmuster wird Software in Models, Views und Controller aufgeteilt (siehe Abbildung 2.9) [8, 63]. Alle diese Komponenten erfüllen eine bestimmte Aufgabe:

Das **Model** repräsentiert die Anwendungsdomäne unabhängig von der Art der Darstellung und der Steuerung und enthält, abhängig von der Implementierung, neben den anwendungsrelevanten Daten auch die Logik, um diese zu manipulieren. Änderungen am Modell werden häufig mithilfe des Beobachter-Entwurfsmusters erkannt, wodurch die View aktualisiert werden kann und der Controller die möglichen Interaktionen mit dem Model anpasst.

Die **View** ist für die Präsentation der Daten des Modells und die Entgegennahme von Benutzerinteraktionen zuständig, die sie an ihren Controller weiterreicht. Es können mehrere Views für das selbe Modell vorhanden sein (z.B. einfache und detaillierte Ansicht).

Der **Controller** bildet das Bindeglied zwischen View und Model und reagiert auf Ereignisse und Benutzerinteraktionen, um die View bzw. das Model zu aktualisieren. Ein Controller kann dabei für die Verwaltung mehrerer Views verantwortlich sein.

Im Zusammenhang mit Webapplikationen sollte jedem Entwickler das MVC-Entwurfsmuster bekannt sein, da es seit Ende der 90er Jahre die Grundlage für viele Web-Frameworks bildet [64]. Dabei wird oft eine HTML-Seite als View, ein serverseitiges Backend als Controller und eine Datenbank als Model verwendet.

2.3.2 HTML5

HTML ist die Standardsprache für die Strukturierung von Dokumenten im World Wide Web. HTML5 ist die Weiterentwicklung des im Jahr 1999 verabschiedeten Standards HTML4 und soll laut [87] Ende 2014 als finale Version standardisiert werden. Es werden jedoch bereits jetzt große Teile von HTML5 von der Mehrheit der Webbrowser unterstützt [18].

HTML5 erweitert HTML4 um eine Vielzahl neuer Elemente und Attribute und soll dabei vorallem die Erstellung dynamischer Inhalte unterstützen bzw. erleichtern [73]. Dies geschieht hauptsächlich über eine engere Bindung von HTML an JavaScript, das in Kapitel 2.3.3 behandelt wird.

Aufbau eines HTML Dokuments

HTML besteht aus verschiedenen Elementen, die mit einem Start-Tag eingeleitet und einem End-Tag geschlossen werden. Elemente können Attribute und weitere Elemente beinhalten, wodurch sich ein HTML-Dokument als Baum darstellen lässt. Die genaue Syntax von HTML-Dokumenten wird in [88] beschrieben. Listing 2.1 zeigt ein sehr einfaches HTML-Dokument:

- Mit dem **Html**-Element wird der Beginn eines HTML-Dokuments markiert.
- Das in Zeile drei startende **head**-Element beinhaltet neben dem Titel auch ein **link**-Element, mit dem eine CSS-Datei verlinkt wird. Diese CSS-Datei ist für die Formatierung, aber auch für Transformationen und Übergänge innerhalb des Dokuments verantwortlich (siehe Kapitel 2.3.4). Mit dem **script**-Element wird eine JavaScript Datei geladen und somit dynamisches Verhalten in das Dokument integriert.
- Das **body**-Element markiert den eigentlichen Beginn des Dokumenteninhalts. Es beinhaltet Text, Bilder, Verweise auf andere Dokumente, Tabellen und vieles mehr.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Meine Seite</title>
5      <link rel="stylesheet" href="meinCSS.css" />
6          <script type="text/javascript" src="meinJavaScript.js"></script>
7    </head>
8  <body>
9    <div id="seite">
10       <p>Das ist der Inhalt meiner HTML-Seite.</p>
11   </div>
12 </body>
13 </html>
```

Listing 2.1: Aufbau eines einfachen HTML-Dokuments

Aufgrund des großen Umfangs der HTML5 Spezifikation werden im folgenden nur die für die Implementierung des im Rahmen dieser Arbeit erstellten Prototypen relevanten Elemente im Detail betrachtet.

Canvas Element

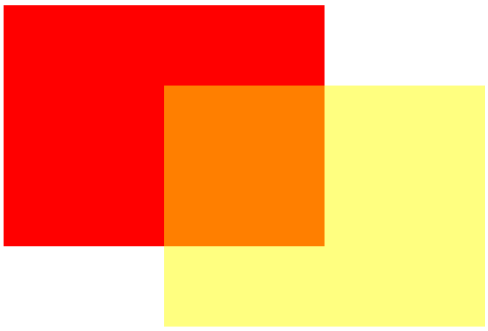

Abbildung 2.10: Zwei überlappende Rechtecke [23]

Bereits ein Monat nach der Gründung der Web Hypertext Application Technology Working Group (WHATWG) wurde von Apple eine Erweiterung für HTML mit dem Namen Canvas vorgestellt und aufgrund positiver Reaktionen innerhalb kürzester Zeit in die HTML5-Spezifikation aufgenommen [23]. Es wird von allen aktuellen Browsern unterstützt [17].

Das Canvas-Element ist ein Container, in dem mittels einer JavaScript-API gezeichnet werden kann. Durch die Attribute *height* und *width* wird die Höhe und Breite dieses Containers in Pixeln auf der Seite festgelegt. Abbildung 2.10 zeigt die mit Listing 2.2 erstellte HTML-Seite, die zwei gezeichnete Rechtecke innerhalb eines Canvas-Elements darstellt.

```
1  <canvas height="800" width="1200">
2          alternativer Inhalt
3  </canvas>
4
5  <script>
6      var canvas = document.querySelector("canvas");
7      var context = canvas.getContext('2d');
8      context.fillStyle = 'red';
9      context.fillRect(0,0,800,600);
10     context.fillStyle = 'rgba(255,255,0,0.5)';
11     context.fillRect(400,200,800,600);
12 </script>
```

Listing 2.2: Code zur Erstellung zwei überlappender Rechtecke

In Zeile sechs wird eine Referenz auf das Canvas-Element erstellt und anschließend der zweidimensionale Zeichenkontext in Zeile sieben definiert. An dieser Stelle sei erwähnt, dass für das Canvas-Element auch ein dreidimensionaler Zeichenkontext erstellt werden kann, indem der Funktion *getContext* der Parameter *webgl* übergeben wird. Innerhalb des Zeichenkontexts werden die eigentlichen Operationen für das Zeichnen ausgeführt und der Inhalt des Canvas-Elements manipuliert. In diesem Beispiel wird für beide Rechtecke eine Füllung definiert, bevor sie tatsächlich gezeichnet werden.

Der Zeichenkontext bietet eine Vielzahl von verschiedenen Funktionen, um das Canvas-Element zu befüllen. Zu diesen gehören unter anderem Funktionen, wie beispielsweise *fillRect* oder *arc*, um geometrische Figuren, *fillText*, um Text und *dramImage*, um Bilder auf das Canvas-Element zu zeichnen. Dabei kann zuvor mit den Funktionen *fillStyle* oder *font* festgelegt werden, wie die Füllung der Figur aussehen soll oder welche Schriftart für den Text verwendet wird. [75]

Der Inhalt eines Canvas-Elements kann auch direkt auf Pixelebene über das Objekt *ImageData* manipuliert werden. Um eine Referenz auf ein ImageData-Objekt zu erhalten, können die Funktionen *createImageData* und *getImageData* verwendet werden. Mit der Funktion *getImageData* wird eine Referenz auf ein ImageData-Objekt erstellt, das einen über Parameter definierten Bildausschnitt des Canvas-Elements repräsentiert. Der eigentliche Bildinhalt ist innerhalb des ImageData-Objekts als eindimensionales Array gespeichert, wobei jedes Pixel durch vier Werte, den Red Green Blue Alpha (RGBA) Werten, repräsentiert wird. Dabei wird jeder der Werte als eine Zahl aus dem Bereich 0 bis 255 dargestellt. Die ersten drei Pixel stehen für die Farben Rot, Grün und Blau. Das vierte Pixel steht für den Alpha Wert, der die Durchsichtigkeit des Pixels definiert. Durch den Alpha Wert entsteht auch der orangefarbene Bereich im Beispiel, in dem die beiden Rechtecke überlappen. [74]

WebSockets

Das WebSocket-Protokoll ist ein auf dem Transmission Control Protocol (TCP) basierendes Protokoll, das es erlaubt, eine bidirektionale Verbindung zwischen einem WebSocket-Server und einem WebSocket-Client herzustellen. Es wurde entwickelt, um einen von beiden Seiten jederzeit beschreibbaren Kommunikationskanal zu erstellen und somit nicht auf das Anfrage-Antwort-Prinzip des Hypertext Transfer Protocols beschränkt zu sein [21]. Laut [19] wird das Protokoll von allen aktuellen Browsern unterstützt.

Die WebSocket-API definiert mehrere Statuscodes im Umgang mit WebSockets:

CONNECTING gibt an, dass die Verbindung noch nicht hergestellt wurde.

OPEN bedeutet, dass die Verbindung erfolgreich hergestellt wurde und Daten gesendet werden können.

CLOSING gibt an, dass die Verbindung gerade getrennt wird.

CLOSED bedeutet, dass die Verbindung getrennt wurde oder nicht hergestellt werden konnte. [35]

Ein weiterer wichtiger Bestandteil der API sind die verschiedenen Ereignisse, die im Zusammenhang mit WebSockets auftreten können:

Das **Open**-Ereignis wird ausgelöst, wenn eine Verbindung erfolgreich hergestellt wurde.

Ein **Close**-Ereignis wird gefeuert, wenn eine Verbindung getrennt wurde.

Das **Error**-Ereignis wird bei allen Arten von Fehlern im Zusammenhang mit WebSockets gefeuert.

Das **Message**-Ereignis wird jedes Mal ausgelöst, wenn Daten über den WebSocket empfangen wurden. [35]

Listing 2.3 zeigt, wie eine WebSocket-Verbindung hergestellt wird. In diesem Beispiel wird versucht, eine Verbindung zu dem WebSocket-Server unter der Adresse *ws://localhost/websocket* herzustellen. Anschließend werden Ereignisbehandler für alle WebSocket-Ereignisse registriert.

```
1  var socket = new WebSocket('ws://localhost/websocket');
2
3  socket.onopen = function() {
4  // Code zum behandeln eines open-Events
5  };
6
7  socket.onclose = function() {
8  // Code zum behandeln eines close-Events
9  };
10
11 socket.onerror = function(error) {
12 // Code zum behandeln eines error-Events
13 };
14
15 socket.onmessage = function(message) {
16 // Code zum behandeln eines message-Events
17 };
```

Listing 2.3: Code zur Erstellung einer WebSocket-Verbindung und Registrierung von Ereignisbehandlern

2.3.3 JavaScript

Netscape entwickelte JavaScript Mitte der neunziger Jahre und reichte die Sprache bei der European Computer Manufacturers Association (ECMA) zur Standardisierung ein. Dort wurde die Sprache aufgrund von namensrechtlichen Problemen unter dem Namen ECMAScript standardisiert. [22]

JavaScript ist eine dynamische, schwach typisierte Programmiersprache, die hauptsächlich innerhalb von Webseiten benutzt wird, um statische HTML-Dokumente zu erweitern, um asynchrone Kommunikation, clientseitige Interaktionen mit dem Benutzer und Kontrolle des Webbrowsers, um Dokumenteninhalte zu verändern. JavaScript kann jedoch auch außerhalb des Webbrowsers für die Programmierung von, oft serverseitigen, Anwendungen eingesetzt werden. [22]

Mit JavaScript ist es möglich, objektorientiert, prozedural oder funktional zu programmieren. Eine Vererbungshierarchie auf Basis von Klassen ist jedoch in JavaScript nicht vorhanden. Um trotzdem den Anforderungen der objektorientierten Programmierung gerecht zu werden, wird in JavaScript ein prototypisches Modell zur Erweiterung von Objekten eingesetzt. Ein Objekt ist bei der Erstellung an ein zweites Objekt, den Prototypen, gebunden. Das Objekt erbt alle im Prototyp vorhandenen Eigenschaften und Funktionen und durch die Veränderung des Proto-

typen können zur Laufzeit daraus erstellte Objekte mit neuer Funktionalität ausgestattet werden. Dieses Vorgehen hat zusätzlich den Vorteil, dass im Prototypen vorhandene Eigenschaften und Funktionen trotz mehrerer daraus erstellter Objekte nur einmal vorhanden sind. Funktionale Programmierung wird aufgrund der Tatsache unterstützt, dass Funktionen in JavaScript first-class-Funktionen sind. Das bedeutet, dass Funktionen in JavaScript bestimmte Bedingungen erfüllen. Funktionen in JavaScript können

- als Parameter an andere Funktionen übergeben werden,
- von einer Funktion als Rückgabewert verwendet werden,
- einer Variable zugewiesen werden,
- zur Laufzeit erzeugt werden und
- in einer Datenstruktur gespeichert werden. [22]

Im Zusammenhang mit JavaScript sei auch das aus JavaScript abgeleitete Dateiformat JavaScript Object Notation (JSON) erwähnt. Es wird hauptsächlich als Alternative zur Extensible Markup Language (XML) verwendet, Daten zwischen einem Server und einer Webapplikation auszutauschen. JSON wird jedoch nicht nur von JavaScript unterstützt, sondern Interpreter stehen für eine Vielzahl von Programmiersprachen zur Verfügung. [22]

Für JavaScript gibt es eine Vielzahl von Frameworks, die das Erstellen von Anwendungen erleichtern, weshalb im Folgenden das Framework jQuery vorgestellt wird, das sich als de-facto Standard beim Einsatz von JavaScript durchgesetzt hat.

jQuery

jQuery ist ein JavaScript Framework, wurde 2006 von John Resig entwickelt und erstmals auf der BarCamp in New York veröffentlicht. Heute wird es als OpenSource-Projekt der jQuery Foundation weiterentwickelt und laut [55] auf über 60% der 10.000 meistbesuchten Webseiten eingesetzt. Zusätzlich wird die Entwicklung von jQuery von einigen namhaften Firmen, wie Microsoft, Wikipedia oder Intel, unterstützt und bereits von zahlreichen Entwicklungsumgebungen und Webentwicklungstools direkt bereitgestellt. Zu den Funktionen von jQuery zählen:

- Document Object Model (DOM)-Manipulation und -Navigation
- Asynchronous JavaScript and XML (AJAX) Unterstützung
- Unterstützung von CSS-Manipulationen
- Erweitertes Ereignissystem
- Erstellung von Effekten und Animationen
- Erweiterbarkeit durch Plug-Ins

- Sonstige Hilfsfunktionen, um beispielsweise Abwärtskompatibilität zu älteren Browsern zu gewährleisten [79]

Diese Funktionalität, die zahlreiche Unterstützung aus der Industrie und die Tatsache, dass um jQuery zu verwenden, nur eine einzige Datei in das Dokument inkludiert werden muss, hat jQuery in den letzten Jahren zu dem wichtigsten Framework im Zusammenhang mit JavaScript gemacht. Hier sei noch erwähnt, dass mit jQuery UI und jQuery Mobile zwei weitere auf jQuery basierende Frameworks existieren, die sich auf das Entwickeln von Benutzeroberflächen konzentrieren.

2.3.4 Cascading Style Sheets

CSS ist eine deklarative Sprache für Stilvorlagen von Dokumenten. Während mit HTML der Inhalt eines Dokuments festgelegt wird, wird CSS benutzt, um das Aussehen von Dokumenten zu verändern. Dabei können neben Schriftarten, Farben und Formatierungen auch Übergänge zwischen Dokumenten in Form von Animationen und mehr definiert werden. Um das zu erreichen, benutzt CSS Regeln, die aus einem Selektor und einer oder mehreren Eigenschafts-Deklarationen innerhalb von geschwungenen Klammern besteht. Dabei werden über den Selektor jene Elemente ausgewählt, deren Eigenschaften die nachfolgenden Werte bekommen sollen. Zu den Selektoren zählen unter anderem:

***** selektiert jedes Element innerhalb des Dokuments.

E selektiert alle Elemente des Typs *E*

E[test] selektiert alle Elemente des Typs *E*, die ein Attribut mit dem Namen *test* enthalten.

.test selektiert alle Elemente, deren class-Attribut *test* enthält.

#test selektiert das Element mit dem Identifikator *test*.

E F selektiert alle Elemente des Typs *F*, die sich hierarchisch unterhalb eines Elements des Typs *E* befinden. [20]

```
1 p {
2   font-weight: bold;
3 }
```

Listing 2.4: CSS um den Text des HTML-Dokuments fett darzustellen

Listing 2.4 zeigt eine einfache Regel, um den Text innerhalb von <*p*>-Elementen fett darzustellen. Abbildung 2.11 zeigt das in Listing 2.1 verwendete HTML-Dokument einmal mit und einmal ohne diese Regel.

Das ist der Inhalt meiner HTML-Seite.

Das ist der Inhalt meiner HTML-Seite.

Abbildung 2.11: Darstellung eines einfachen HTML-Dokuments. Oben mit und unten ohne eingebundener CSS-Regel.

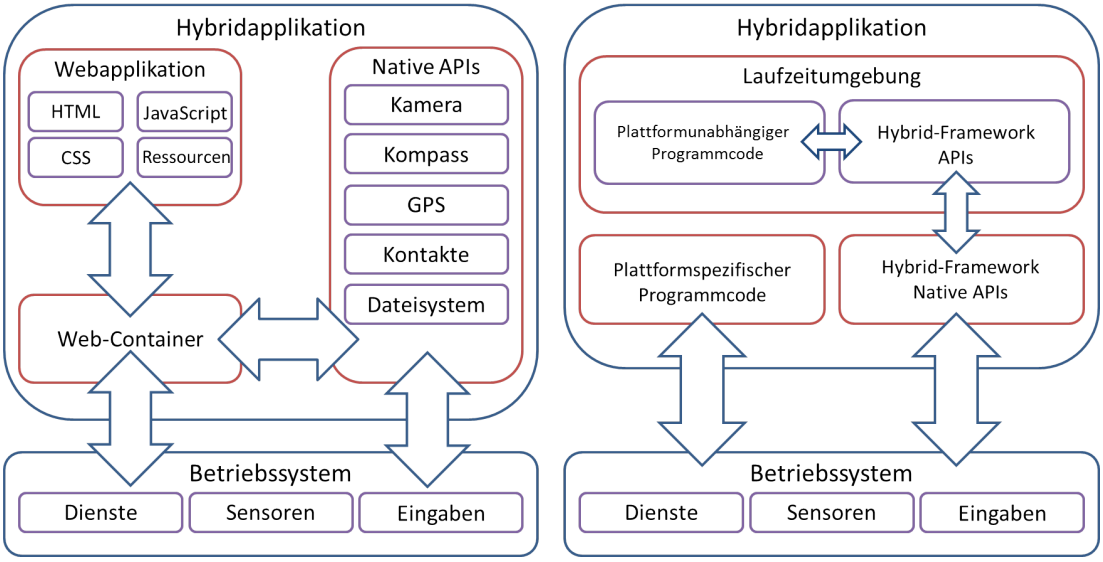

Abbildung 2.12: Architektur von Hybridapplikation basierend auf Webtechnologien bzw. eigenständigen Laufzeitumgebungen.

2.4 Hybridapplikationen

Hybridapplikationen können als eine Kombination von nativer Funktionalität und plattformunabhängigen Ansätzen gesehen werden. Dabei wird zwischen zwei Arten von Hybridapplikationen unterschieden (siehe Abbildung 2.12):

- Hybridapplikationen auf Basis von Webtechnologien betten Webapplikationen in eine native Hülle ein. Dabei liefert die native Hülle eine API, um über Webtechnologien auf Gerätefunktionen zuzugreifen. Gleichzeitig ist die Hülle für die Ausführung der Webapplikationen in einem vom darunterliegenden Betriebssystem zur Verfügung gestellten Web-Container zuständig. Somit können, unabhängig von der Plattform, native Applikationen mit den in Kapitel 2.3 vorgestellten Technologien erstellt werden. In Kapitel 2.4.1 wird mit Apache Cordova einer der bekanntesten Vertreter dieser Art von Hybridapplikationen vorgestellt. [31]
- Hybridapplikationen auf Basis einer eigenständigen Laufzeitumgebung kombinieren die nativen Benutzerschnittstellenelemente mit plattformunabhängigen Code, für dessen Aus-

führung die Laufzeitumgebung verantwortlich ist. Die Benutzerschnittstelle wird dabei meist ebenfalls mit einer eigenen Sprache erstellt und von der Laufzeitumgebung in native Elemente übersetzt. Der Code für die Programmlogik kann mit diesem Ansatz in den von der Laufzeitumgebung vorgegebenen Sprachen erstellt werden und ist somit nicht an eine Plattform gebunden. Einer der bekanntesten Vertreter dieser Art von Hybridapplikationen ist Appcelerator Titanium, das in Kapitel 2.4.2 näher betrachtet wird. [31]

2.4.1 Apache Cordova

Apache Cordova wurde ursprünglich von Nitobi Software unter dem Namen PhoneGap entwickelt. Nachdem Adobe Nitobi Software im Jahr 2011 aufgekauft hat, wurde der Code von PhoneGap der Apache Software Foundation übergeben. Dort wird das Framework unter dem Namen Apache Cordova gepflegt und ist als OSS verfügbar. PhoneGap existiert jedoch weiterhin bei Adobe und ist eine Distribution von Apache Cordova. [31]

Apache Cordova Applikationen werden komplett mit den Webtechnologien HTML5, CSS und JavaScript erstellt. Somit können Applikationen ohne den für die Plattformen sonst gebräuchlichen Programmiersprachen entwickelt werden. Apache Cordova Anwendungen werden innerhalb einer Hülle ausgeführt, die für unterschiedliche Plattformen zur Verfügung steht und eine einheitliche API bereitstellt. Über diese API kann mit JavaScript auf native Gerätefunktionen, wie beispielsweise die unterschiedlichen Sensoren der mobilen Geräte oder die Kontaktdaten des Benutzers, zugegriffen werden. Zusätzlich können Cordova Applikationen über ein Plug-In System erweitert werden. Plug-Ins bestehen dabei aus einer JavaScript-Schnittstelle und einer nativen Komponente, die für jede zu unterstützende Plattformen entwickelt werden muss. Somit können mit vergleichsweise geringem Aufwand alle nativ möglichen Aufgaben mit JavaScript erledigt werden. [24]

2.4.2 Appcelerator Titanium

Appcelerator Titanium wird von Appcelerator als OSS entwickelt und kann kostenlos für die Entwicklung verwendet werden. Über die Appcelerator Plattform wird Kunden, zusätzlich zur freien Titanium Version, Support, Unterstützung beim kompletten Lebenszyklus von mobilen Applikationen und Trainings verkauft. [38]

Die Entwicklung von Titanium Applikationen wird mit der auf Eclipse basierenden IDE Titanium Studio unterstützt. Außerdem muss das Titanium SDK und die jeweiligen SDKs für die zu unterstützenden Plattformen installiert werden. Das SDK stellt die wichtigsten Funktionen über eine API zur Verfügung. Die gesamte Programmlogik von Titanium Anwendungen wird mit JavaScript realisiert [39]. Dabei wird die komplette Entwicklung mit Hilfe des Appcelerator Alloy Frameworks durchgeführt. Alloy ist ein MVC-Framework, in welchem die Benutzerschnittstelle mit XML und CSS erstellt wird [37]. Zusätzlich kann auch auf wiederverwendbare Widgets zurückgegriffen werden. Der gesamte geschriebene JavaScript-Code einer Titanium Anwendung wird zur Laufzeit von der JavaScript-Engine der jeweiligen Plattform interpretiert. Es werden alle Benutzerschnittstellenelemente mit nativen Komponenten umgesetzt, wodurch Applikationen großteils dem nativen Erscheinungsbild entsprechen. [31]

2.5 Modellgetriebene Softwareentwicklung

In der modellgetriebenen Softwareentwicklung liegt die Basis, im Gegensatz zu anderen Entwicklungsansätzen, in Modellen, die eine bestimmte Domäne abbilden und somit versuchen, die Komplexität der Softwareentwicklung durch Einführung einer Abstraktionsebene besser handhabbar zu machen. [86] Die folgenden Unterkapitel beschäftigen sich mit den Grundprinzipien hinter der modellgetriebenen Entwicklung und sollen so die Ideen dahinter näher bringen (siehe Abbildung 2.13).

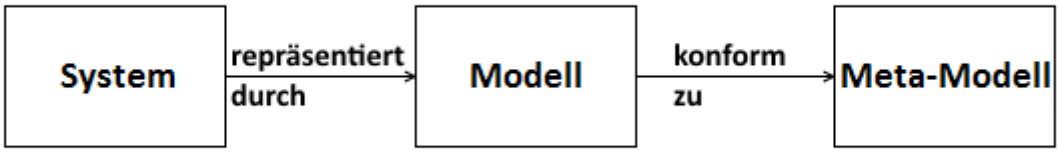

Abbildung 2.13: Grundlegende Idee von MDE basierend auf [9].

2.5.1 Meta-Modelle

Meta-Modelle bilden das Grundgerüst der modellgetriebenen Entwicklung, weil sie als die Basis für die Definition von Modellierungssprachen gelten. Sie beschreiben die für die Erstellung von Modellen zur Verfügung stehenden Sprachkonstrukte [6]. Dabei müssen Meta-Modelle nicht unbedingt in einer standardisierten Sprache spezifiziert werden, sondern können auch in natürlichen Sprachen geschrieben sein. Die computergestützte Arbeit mit Modellen wird jedoch durch formalisierte Meta-Modelle erleichtert oder überhaupt erst ermöglicht. [25]

Laut [6] wird zwischen vier Hauptkomponenten von Meta-Modellen unterschieden:

Abstrakte Syntax: Definiert die Sprachkonzepte und wie diese kombiniert werden können.

Konkrete Syntax: Die eigentliche, textuelle oder grafische, Notation der Sprachkonzepte.

Semantik: Definiert die Bedeutung der Sprachkonzepte.

Serialisierungs Syntax: Um Daten dauerhaft zu speichern und einen Datenaustausch zwischen unterschiedlichen Tools zu gewährleisten.

Aufgrund der Tatsache, dass Meta-Modelle die Grundlage von Modellierungssprachen bilden, kann zwischen zwei Arten von Modellierungssprachen unterschieden werden:

Domain Specific Languages (DSLs) werden verwendet, um Modelle für eine bestimmte Domäne zu erstellen.

General Purpose Languages (GPLs) sind nicht an eine Domäne gebunden und können daher für die Erstellung von Modellen in jedem Aufgabenbereich eingesetzt werden. An dieser Stelle sei die UML als bekanntester Vertreter von GPLs erwähnt. [6]

2.5.2 Modell-zu-Modell-Transformation

In der modellgetriebenen Softwareentwicklung wird Modell-Transformation dazu verwendet, um neue Modelle zu erzeugen [59]. Dabei wird aus einem oder mehreren Ausgangsmodellen mit Hilfe spezieller Transformationsregeln ein oder mehrere Zielmodelle generiert [25]. Somit können beispielsweise mehrere Modelle zusammengefasst oder nur kleinere Anpassungen an einem Modell vorgenommen werden. Transformationen können die folgenden Eigenschaften zugeschrieben werden, wobei sich einige gegenseitig ausschließen:

Zweiseitig gerichtete-Transformation: Die Transformation kann in beide Richtungen ausgeführt werden.

Einseitig gerichtete-Transformation: Die Transformation kann nur in eine Richtung ausgeführt werden.

Vertikale-Transformation: Ein Modell wird in ein spezifischeres, also weniger abstraktes Modell, transformiert.

Horizontale-Transformation: Ein Modell wird in ein Modell derselben Abstraktionsebene transformiert.

1-zu-1-Transformation: Aus einem einzelnen Ausgangsmodell wird ein einzelnes Zielmodell generiert.

1-zu-N-Transformation: Aus einem einzelnen Ausgangsmodell werden mehrere Zielmodelle generiert.

N-zu-1-Transformation: Aus mehreren Ausgangsmodellen wird ein einzelnes Zielmodell generiert.

M-zu-N-Transformation: Aus mehreren Ausgangsmodellen werden mehrere Zielmodelle generiert. [14, 25, 77]

Die Transformationsregeln basieren auf einem Meta-Modell, das wiederum auf den Meta-Modellen der Ausgangs- und Zielmodelle aufbaut. Somit ist die Transformation selbst konform zu ihrem eigenen Meta-Modell, das die Transformationssprache definiert, mit der die Transformationsregeln geschrieben sind [6]. Transformationsregeln bestehen dabei aus einer linken und einer rechten Seite. Das Ausgangsmodell wird durch die linke Seite und das Zielmodell durch die rechte Seite repräsentiert. Beide Seiten bestehen wiederum aus Variablen, die unter anderem Elemente aus den Modellen abbilden, Mustern, die textuell oder grafisch Modell-Fragmente darstellen, oder logischen Operationen, wie beispielsweise Object Constraint Language (OCL)-Abfragen [14].

Für die Umsetzung von Modell-zu-Modell-Transformationen gibt es unterschiedliche Ansätze. Beim Ansatz der direkten Manipulation werden Modelle mithilfe einer API manipuliert (z.B. Java Metadata Interface (JMI)). Der relationale Ansatz basiert auf mathematischen Relationen. Dabei werden Einschränkungen benutzt, um die Beziehung zwischen Ausgangselementen und Zielelementen zu spezifizieren (z.B. Query/View/Transformation (QVT)-Sprachen). Ein weiterer Ansatz ist der auf Graphen-Transformation basierende (z.B. VIsual Automated model TRAnsformations (VIATRA)). Strukturbasierte Ansätze bestehen aus zwei Phasen. In der ersten Phase wird die hierarchische Struktur des Zielmodells erstellt. Die zweite Phase beschäftigt sich mit dem Setzen der Attribute und Verweise innerhalb des Modells. Hybride Ansätze kombinieren unterschiedliche Methoden der bereits genannten Ansätze. An dieser Stelle sei die ATLAS Transformation Language (ATL) als prominenter Vertreter von hybriden Ansätzen erwähnt [14].

2.5.3 Modell-zu-Text-Transformation

Ein weiterer Schwerpunkt der modellgetriebenen Softwareentwicklung ist die Modell-zu-Text-Transformation. Darunter wird die Generierung von Dokumentation, Testfällen und vor allem ausführbaren Programmcode, aber auch die Serialisierung von Modellen, verstanden. Im Gegensatz zu Compilern, die aus vorhandenem Quellcode Maschinencode generieren, ist die Codegenerierung in der modellgetriebenen Entwicklung eine Transformation von einem Modell zu Quellcode [6].

Um Modell-zu-Text-Transformationen durchzuführen, müssen drei wichtige Punkte geklärt werden:

- Kann der gesamte Code oder nur Teile davon generiert werden?
- Die Art des Quellcodes muss definiert werden, wobei höhere Programmiersprachen bevorzugt werden sollten.
- Es muss definiert werden, wie der Code generiert wird. Das heißt, dass eine passende Transformationssprache (DSLs oder GPLs) gewählt oder entwickelt werden muss. [6]

Es gibt zwei Ansätze für die Umsetzung von Modell-zu-Text-Transformationen. Beim besucherbasierten Ansatz wird die Modellrepräsentationen durchlaufen und währenddessen aus den Informationen der Quellcode erzeugt und in einen textbasierten Datenstrom geschrieben. Der zweite Ansatz basiert auf Templates. Er kombiniert den Zielquellcode mit Meta-Code. Dabei wird der Zielquellcode direkt in die Ausgabedatei geschrieben und über den Meta-Code der Zugriff auf Daten des Ausgangsmodells ermöglicht [14].

KAPITEL 3

Analyse der Entwicklungsansätze

3.1 Übersicht

In Kapitel 2 wurde eine Vielzahl unterschiedlicher mobiler Plattformen und mehrere Ansätze vorgestellt, um Applikationen für diese Plattformen zu entwickeln. In diesem Kapitel werden die verschiedene Entwicklungsansätze in Hinblick auf den Einsatz als Visualisierungslösung in der Automatisierungstechnik analysiert. Dafür werden zuerst unterschiedliche Anforderungen an mobile Applikationen erarbeitet, um einen Einsatz als Visualisierungslösung in der Automatisierungstechnik zu ermöglichen. Anschließend werden die Probleme beim Entwickeln von mobilen Applikationen aufgezeigt, die durch die hohe Anzahl mobiler Plattformen entstehen. Danach werden in diesem Kontext die Ideen und verwendeten Technologien hinter den unterschiedlichen Ansätzen diskutiert. Abschließend werden die Ansätze verglichen.

3.2 Schwierigkeiten bei der Entwicklung mobiler Applikationen

Durch die große Anzahl der in Kapitel 2 vorgestellten mobilen Plattformen kommt es zu mehreren Problemen bei der Entwicklung mobiler Applikationen. In Kapitel 3.2.1 werden Probleme im Zusammenhang mit den unterschiedlichen Entwicklungsumgebungen, Programmiersprachen und Tools behandelt. Kapitel 3.2.2 beschäftigt sich mit Schwierigkeiten, die durch die zusätzliche Versions- und Gerätevielfalt innerhalb der Plattformen entstehen. Zum Abschluss dieses Unterkapitels wird in Kapitel 3.2.3 die Problematik diskutiert, die aufgrund der unterschiedlichen Design-Richtlinien für die Plattformen entsteht.

3.2.1 Entwicklungsumgebungen und Programmiersprachen

Tabelle 3.1 zeigt die unterschiedlichen Programmiersprachen und Entwicklungsumgebungen, die für die verschiedenen Entwicklungsansätze verwendet werden. Als Vertreter von Hybridapplikationen wurde Apache Cordova und Appcelerator Titanium gewählt. Als Vertreter der modellgetriebenen Softwareentwicklung wurde für Vergleichszwecke das Eclipse Modeling Frame-

work (EMF) herangezogen. Wie man der Tabelle entnehmen kann, ergibt sich eine Vielzahl an unterschiedlichen Technologien, die beherrscht werden müssen, um für alle großen Plattformen zu entwickeln. Dadurch muss natürlich das Entwicklerteam entsprechende Ausmaße annehmen bzw. die Entwicklungszeit entsprechend verlängert werden.

3.2.2 Versions- und Gerätevielfalt

Durch die in Kapitel 2.2 bereits erwähnte Versions- und Gerätevielfalt kommt es bei der Entwicklung von mobilen Applikationen zu weiteren Problemen [58]:

Displayauflösung - Durch die Vielzahl an Geräten muss eine mobile Applikation an eine beträchtliche Anzahl von Displayauflösungen angepasst werden. Auf der iOS-Plattform ist die Geräte- und somit ebenfalls die Auflösungsvielfalt zwar beschränkt, jedoch müssen auch auf dieser Plattform schon mehrere Auflösungen bedient werden. Auf der Android-Plattform, wo von Einstiegsgeräten bis hin zu Top-Modellen alle Geräteklassen bedient werden, muss dementsprechend eine größere Anzahl an Displayauflösungen bedacht werden.

Softwaretests - Ebenfalls mit der Anzahl der Geräte verbunden ist der Aufwand, der in Softwaretests investiert werden muss. Dieser erhöht sich zusätzlich durch die in Kapitel 2.2 stellenweise angesprochene Versionsverteilung der Plattformen. Durch den gezielten Einsatz der Emulatoren, die für alle großen Plattformen zur Verfügung stehen, kann der Aufwand zwar reduziert werden, doch durch die große Anzahl an verschiedenen Hardwarekombinationen muss auch eine Vielzahl von Geräten für direkte Tests angeschafft werden. Hier sollten die Verkaufszahlen der verschiedenen Gerätehersteller als Orientierung dienen, um selbst auf weit verbreiteten Geräten direkt testen zu können. Des weiteren sollte der Einsatz von Tools in Betracht gezogen werden, die ein automatisiertes Testen der Anwendung erleichtern bzw. ermöglichen. Für native Applikationen stellen die Hersteller der jeweiligen Plattform Unittest-Frameworks zur Verfügung, wie z.B das Android testing framework[1]. JavaScript-Code oder die Benutzeroberfläche von Web- und Hybridapplikationen kann z.B. mit JsUnit[2] oder Selenium[3] getestet werden. Titanium-Applikationen können mit Appcelerator Test[4] getestet werden.

Wartung und Fehlerbehebung - Auch die nach Veröffentlichung der Applikation anfallenden Aufgaben werden durch die Vielzahl an Geräten und Versionen erschwert, weil trotz intensiven Tests nicht verhindert werden kann, dass Fehler nur unter bestimmten Hardwarekonstellationen auftreten. Hier sollte auf die von den Plattformen zur Verfügung gestellten Feedback-Mechanismen zurückgegriffen werden, ob die Bewertung in den entsprechenden App-Stores oder Fehlerberichte bei einem Absturz zur Verfügung stehen. Dadurch kann der Aufwand der Fehlersuche minimiert werden und effektiv auf Kundenwünsche reagiert werden.

[1] https://developer.android.com/tools/testing/index.html
[2] http://jsunit.net/
[3] http://www.seleniumhq.org/
[4] http://www.appcelerator.com/functionaltest/

	Android	**iOS**	**Windows Phone**	**Webapplikationen**
Entwicklungsumgebung	Eclipse mit ADT-Plugin, Android Studio	Apple Xcode	Microsoft Visual Studio	z.B. WebStorm oder Eclipse mit den entsprechenden Plug-Ins
Programmiersprache(n)	Java	Objective-C, Swift	C#, VB.Net oder HTML mit JavaScript	HTML, JavaScript, CSS

	Apache Cordova	**Appcelerator Titanium**	**Modellgetrieben**
Entwicklungsumgebung	Apache Cordova Command-Line Interface (CLI) und zusätzlich die Entwicklungsumgebung für die jeweilige Plattform	Titanium Studio	z.B. Eclipse mit dem EMF
Programmiersprache(n)	HTML, JavaScript, CSS	JavaScript	DSLs und GPLs; Im Falle von EMF: Java, UML, Ecore

Tabelle 3.1: Programmiersprachen und Entwicklungsumgebungen, die zum Erstellen von mobilen Applikationen benutzt werden.

3.2.3 Design-Richtlinien

Für jede mobile Plattform werden von den Herstellern Designrichtlinien für die Erstellung der Benutzeroberflächen angeboten. Dadurch ist es mit viel Aufwand verbunden, eine Applikationen für mehrere Plattformen anzubieten und trotzdem darauf zu achten, dass die jeweiligen Designrichtlinien eingehalten werden. Durch Nichteinhaltung der Designrichtlinien wird riskiert, dass die Bedienung der Anwendung erschwert wird, weil sie sich möglicherweise stark von der gewohnten Bedienung auf der Plattform unterscheidet. Wird jedoch auf ein komplett einheitliches Bedienkonzept auf mehreren Plattformen Wert gelegt, besteht natürlich die Möglichkeit, die Designrichtlinien teilweise zu ignorieren.

3.3 Anforderungsanalyse für mobile Applikationen

Die Anforderungen an eine mobile Applikation sind stark von unterschiedlichen Faktoren abhängig (zum Großteil basierend auf [31, 89]):

Plattformen Welche Plattformen müssen unterstützt werden? Die Antwort auf diese Frage beeinflusst maßgeblich die Auswahl des Entwicklungsansatzes. Wenn beispielsweise nur eine einzige Plattform exklusiv unterstützt werden soll, ist der Einsatz einer nativen Applikation am sinnvollsten, weil so einige der in Kapitel 3.2 genannten Probleme vermieden werden.

Native Funktionen Wird Zugriff auf plattformspezifische Funktionalität benötigt? Zu diesen Funktionen zählen neben dem Zugriff auf unterschiedliche Sensordaten, wie beispielsweise Daten aus dem Global Positioning System (GPS), auch der Zugriff auf die Kamera oder Benachrichtigungen.

Benutzerfreundlichkeit Soll die Anwendung die native Optik und Haptik haben? Native Applikationen sind sich aufgrund der Design-Richtlinien der unterschiedlichen Plattformen in der Bedienung und im Aussehen ähnlich. Webapplikationen können das native Aussehen nur über bestimmte Frameworks, wie beispielsweise jQuery Mobile, simulieren oder verzichten ganz darauf. Hybridapplikationen können, wie Webapplikationen, das native Aussehen emulieren, darauf verzichten oder erlauben fallweise sogar die Generierung einer komplett nativen Benutzeroberfläche.

Geschwindigkeit Muss die Ausführungsgeschwindigkeit einer nativen Applikation entsprechen? Durch die in Hybridapplikationen und Webapplikationen zusätzlich eingeführten Abstraktionsebenen erreichen diese Applikationen nicht die gleiche Ausführungs- und Startgeschwindigkeit wie native Applikationen.

Kommunikation Woher kommen die zu visualisierenden Daten und wie wird darauf zugegriffen? In der Automatisierungstechnik kommen häufig sehr langlebige Komponenten zum Einsatz und somit ist oft eine gewisse Infrastruktur vorgegeben. Nicht selten wird serielle Kommunikation eingesetzt, aber auch Webservices oder WebSockets können für die Datenaggregation verwendet werden, wofür meist eine Wireless Local Area Network (WLAN)-Verbindung benötigt wird.

Verteilung Muss die Anwendung über den jeweiligen App Store der Plattform installierbar sein? Die Frage beschäftigt sich mit der Verteilung der Anwendung nach erfolgter Entwicklung. Jeder native Ansatz beinhaltet eine Möglichkeit die Applikation in dem entsprechenden App Store zu veröffentlichen und auch plattformunabhängige Ansätze bieten dafür eine Lösung, die jedoch meist mit etwas mehr Aufwand verbunden ist. Zusätzlich ist das Aktualisieren einer Applikationen einfacher, die über den App Store vertrieben wird.

Auch aus Entwicklersicht sollten einige Punkte berücksichtigt werden (basierend auf [31]):

IDE Welche Entwicklungsumgebungen und Tools unterstützen den Entwickler? Neben Debugging-Tools sollten hier auch die in Kapitel 2.2 mehrfach erwähnten Emulatoren und Tools in Betracht gezogen werden, die das automatisierte Testen unterstützten oder erst ermöglichen.

Benutzeroberfläche Gibt es Editoren, die das Erstellen von Benutzeroberflächen nach dem What You See Is What You Get (WYSIWYG) Prinzip unterstützen? WYSIWYG-Editoren vereinfachen das Entwickeln von Benutzeroberflächen, weil Zwischenergebnisse nicht immer auf einem Gerät installiert werden müssen. Zusätzlich können Tools vorhanden sein, die das Testen der Benutzeroberfläche erlauben.

Dokumentation und Support Ist eine ausreichende Dokumentation des Entwicklungsansatzes und der mitgelieferten APIs vorhanden? Bei diesem Punkt sollte auch evaluiert werden, ob Codebeispiele in der Dokumentation enthalten sind und Probleme durch Nachfrage bei einer einschlägigen Community gelöst werden können. Hier sollte auch in Betracht gezogen werden, ob der Entwicklungsansatz auf bereits bekannten und etablierten Programmierparadigmen aufbaut.

Skalierbarkeit Ist der Entwicklungsansatz skalierbar? Hier gilt es zu evaluieren, ob der Ansatz Modularisierung unterstützt und es größeren Entwicklerteams somit einfacher erlaubt, parallel an der Applikation bzw. Teilen davon zu arbeiten.

Wiederverwendbarkeit Ist es möglich, bereits geschriebenen Code wiederzuverwenden? Unter diesem Punkt wird bewertet, ob bereits geschriebener Code für einen anderen Ansatz wiederverwendet werden kann, um das Risiko zu minimieren, sich auf einen einzigen Ansatz einzuschränken.

Lerneinsatz Ist mit einer größeren Einstiegshürde im Zusammenhang mit dem Entwicklungsansatz zu erwarten? Werden viele unterschiedliche Paradigmen in einem Ansatz vereint und steigt damit der Aufwand mit diesem Ansatz eine Applikation zu entwickeln? Genauso wirkt sich die Anzahl an Programmiersprachen und zusätzlich benötigten Tools auf den Gesamtaufwand aus, wenn diese für die Entwickler noch nicht bekannt sind.

In den folgenden Unterkapitel werden die in Kapitel 2 vorgestellten Ansätze im Kontext der in diesem Kapitel behandelten Fragen untersucht und abschließend miteinander verglichen.

3.4 Native Entwicklungsansätze

Die Aussagen in diesem Kapitel basieren zu einem großen Teil auf eigenen praktischen Erfahrungen, gestützt durch die Bewertungen und Beschreibungen in [2, 13, 30, 31].

Plattformen

Native Entwicklungsansätze unterstützen nur eine einzige Plattform. Das bedeutet, dass, wenn eine mobile Applikation auf mehreren Zielplattformen lauffähig sein soll, diese für jede Plattform explizit entwickelt werden muss. Dadurch ist es sehr aufwändig, mit nativen Entwicklungsansätzen mehrere Plattformen zu unterstützen.

Native Funktionen

Mit jedem native Entwicklungsansatz kann auf alle nativen Funktionen, wie beispielsweise GPS-Daten oder die Kamera, zugegriffen werden. Dies wird bei allen in Kapitel 2.2 vorgestellten Plattformen über APIs ermöglicht.

Benutzerfreundlichkeit

Wie bereits bei der Auflistung der Anforderung erwähnt, dient das Aussehen von nativen Applikationen als Standard für diese Kategorie. Durch den Einsatz der nativen Benutzeroberflächenelemente ähneln sich native Applikationen auf einer Plattform und erleichtern dem Benutzer somit die Bedienung. Auch die Art und Weise, wie auf bestimmte Ereignisse reagiert werden soll, ist von Plattform zu Plattform verschieden, bietet jedoch innerhalb einer Plattform wieder den Vorteil der einfacheren Bedienung für den Benutzer.

Geschwindigkeit

Auch in dieser Kategorie stellen native Applikationen den zu erreichenden Vergleichsstandard dar. Native Applikationen sind exakt auf die verwendete Plattform abgestimmt und optimiert, wodurch sich eine schnelle Start- und Ausführungsgeschwindigkeit ergibt.

Kommunikation

Die Kommunikationsmöglichkeiten der nativen Applikationen ist von der jeweiligen Plattform abhängig. Alle in Kapitel 2.2 angeführten Plattformen unterstützen die Kommunikation über gängige Kommunikationsstandards wie Bluetooth oder WLAN. Darüber hinaus existieren für Linux basierte Plattformen auch Geräte, die mit seriellen Kommunikationsschnittstellen oder Ethernet Schnittstellen ausgestattet sind.

Verteilung

Beim Einsatz eines nativen Entwicklungseinsatzes ist es sehr einfach, die entwickelte Applikation im App-Store der entsprechenden Plattform anzubieten. Die SDKs enthalten alle nötigen Tools, um die Applikation im richtigen Format zu erstellen und für den App-Store zu signieren.

IDE

Wie bereits ebenfalls in Kapitel 2.2 beschrieben, bieten alle nativen Entwicklungsansätze eine eigene IDE an. Diese ist in allen Fällen mit einem mächtigen Debugger und Emulatoren ausgestattet. Desweiteren kann über die IDEs auf eine Vielzahl an zusätzlichen Informationen, wie beispielsweise Gerätelogs, zugegriffen werden und es werden Methoden angeboten, die das Testen der Applikationen auf der jeweiligen Plattform ermöglichen.

Benutzeroberfläche

Das Erstellen von Benutzeroberflächen wird bei allen nativen Entwicklungsumgebungen durch spezielle Tools unterstützt. Diese Tools erlauben es, die Benutzeroberfläche der Applikation innerhalb der IDE nach dem WYSIWYG-Prinzip zu erstellen. Dadurch können Veränderungen an der Benutzeroberfläche ohne das Installieren der Anwendung auf einem Gerät angezeigt werden, wodurch wiederum Zeit eingespart wird.

Dokumentation und Support

Für alle nativen Entwicklungsansätze steht im Internet eine umfangreiche Dokumentation auf den Webseiten der Hersteller zur Verfügung. Diese reichen von der Installation des SDKs, der IDE und zusätzlich benötigten Tools, bis hin zu Codebeispielen und detaillierten Beschreibungen der APIs. Desweiteren gibt es für alle in Kapitel 2.2 näher beschriebenen Plattformen eine große Community, die bei der Entwicklung auftretende Fragen zügig und meist kompetent beantwortet.

Skalierbarkeit

Auch hier gilt für alle Plattformen, dass sie Konzepte für die Modularisierung unterstützen und somit das parallele Arbeiten mehrerer Entwickler erleichtern.

Wiederverwendbarkeit

Bereits geschriebener Code für eine Plattform kann nicht direkt, sondern nur das dahinterstehende Konzept für eine andere Plattform wiederverwendet werden. Dies ist den unterschiedlichen Technologien geschuldet, die für die Entwicklung eingesetzt werden.

Lerneinsatz

Mit Ausnahme der Windows Phone Plattform stehen für die Entwicklung nativer Applikationen nur eine sehr begrenzte Anzahl an Technologien zur Verfügung, wodurch für den Einstieg in die Entwicklung nur kleinere Hürden im Umgang mit noch nicht Bekanntem im Weg stehen. Zusätzlich wird die Einstiegshürde durch die bereits erwähnte, sehr umfangreiche Dokumentationen verringert.

3.5 Webapplikationen

Die Bewertungen in diesem Unterkapitel basieren zum Großteil auf den Arbeiten von Heitkötter et al. [30, 31] und Corral et al. [13].

Plattformen

Auf jeder der genannten Plattformen ist ein Webbrowser enthalten, wodurch sich Webapplikationen auf allen Plattformen verwenden lassen. Außerdem besteht auf allen Plattformen die Möglichkeit, dass alternative Webbrowser, wie beispielsweise Mozilla Firefox, installiert werden können. Darauf muss zurückgegriffen werden, falls einige der zu verwendenden Funktionen von einem Webbrowser nicht oder nur bedingt unterstützt werden[5].

Native Funktionen

Mit Webapplikationen kann zwar auf manche nativen Gerätefunktionen zugegriffen werden, wie beispielsweise die Positionsbestimmung, es besteht jedoch nicht die Möglichkeit, wie bei nativen Applikationen, auf alle plattformspezifischen Sensordaten zuzugreifen. Hier sei jedoch erwähnt, dass das Abspielen von Multimediainhalten in Webapplikationen kein Problem darstellt. Desweiteren kann in modernen Browsern die “klassische“ Touchbedienung problemlos verwendet werden.

Benutzerfreundlichkeit

Es ist mit Webapplikationen nicht möglich, native Benutzeroberflächenelemente zu verwenden. Um trotzdem nicht ganz auf natives Aussehen verzichten zu müssen, haben es sich einige Projekte, wie beispielsweise jQuery Mobile, zur Aufgabe gemacht, eine nativ aussehende Benutzeroberfläche mit HTML5 und CSS zu erstellen. Dies wird durch die Verwendung von unterschiedlichen Themes ermöglicht. Zusätzlich besteht mit HTML5, CSS und JavaScript die Möglichkeit, das Verhalten von Webapplikationen an jenes nativer Applikationen anzugleichen. Als Beispiel sei hier ein Formular in einer Webapplikation genannt, das nach dem Schließen und erneutem Öffnen die bereits eingegeben Daten wieder anzeigen kann.

Geschwindigkeit

Mit einer Webapplikation kann die Startgeschwindigkeit einer nativen Applikation, aufgrund des erforderlichen Verbindungsaufbaus, nicht erreicht werden. Während eine native Applikation bereits die Benutzeroberfläche anzeigen kann, muss bei der Webapplikationen zuerst der Webbrowser gestartet und anschließend die Webseite aufgerufen werden. Bei der Ausführungsgeschwindigkeit sind moderne Webbrowser durch zahlreiche Optimierungen erst bei einer komplexeren Benutzeroberfläche, die beispielsweise viele Animationen enthält, langsamer als native Applikationen.

[5] Für die Überprüfung, ob eine bestimmte Funktion von einem Webbrowser unterstützt wird, kann die Webseite http://www.caniuse.com verwendet werden.

Kommunikation

Die Kommunikation von Webapplikationen begrenzt sich auf die vom Webbrowser unterstützten Protokolle. Jedoch kann mit modernen Webbrowsern zumindest eine WebSocket-Verbindung, wie in Kapitel 2.3.2 beschrieben, verwendet werden.

Verteilung

Der Verteilung von Webapplikationen basiert auf dem einfachen Prinzip, dass nur der Uniform Resource Locator (URL) bekannt sein muss, unter dem die Applikation aufrufbar ist. Das Anbieten von reinen Webapplikationen über den entsprechenden App-Store ist nicht, oder nur über Umwege möglich. Dafür wird bei jedem Besuch der Webseite die aktuellste Version der Applikation ausgeführt.

IDE

Für die Entwicklung von Webapplikationen stehen zahlreiche ausgereifte IDEs zur Verfügung. Als Beispiel sei hier die IDE WebStorm der Firma JetBrains erwähnt. Desweiteren beinhalten alle moderne Browser Debugging- und Profiling-Tools, die die Entwicklung von Webapplikationen erleichtern.

Benutzeroberfläche

Auch für Webapplikationen existieren eine Vielzahl an WYSIWYG-Editoren, die das Erstellen von Benutzeroberflächen vereinfachen. Zusätzlich lässt sich die Benutzeroberfläche einer Webapplikation ohne teilweise aufwendiges Compilieren sehr schnell neu laden. Als weiterer Vorteil sei hier das schnelle Benutzeroberflächen-Prototyping erwähnt, das mit Webapplikationen realisiert werden kann. Dabei werden nur die Übergänge von einer Ansicht zu einer Anderen implementiert.

Dokumentation und Support

Die Dokumentation der Technologien von Webapplikationen (abhängig von den eingesetzten Frameworks) ist als sehr gut zu bewerten. Des Weiteren gilt für bekanntere Frameworks, dass diese getrieben durch eine starke Community auch einen sehr guten Support bieten können.

Skalierbarkeit

Üblicherweise lassen sich Webapplikationen ohne großen Aufwand in mehrere parallel zu bearbeitende Dateien zerlegen. Jedoch ist die Modularisierung sehr stark von etwaig verwendeten Frameworks abhängig.

Wiederverwendbarkeit

Ein bereits als Webapplikationen gestartetes Projekt kann, meist ohne großen Aufwand, in eine Hybridlösung übernommen werden. Beispielsweise könnte eine Applikation, wenn doch Zugriff auf bestimmte native Funktionen benötigt wird, in Apache Cordova portiert und um die fehlende Funktionalität erweitert werden.

Lerneinsatz

Die Anzahl an zu beherrschenden Technologien ist bei der Entwicklung von Webapplikationen überschaubar. Durch die zusätzlich qualitativ hochwertigen Dokumentationen zu eben diesen Technologien, ist der Einstieg in die Webentwicklung als einfach zu betrachten. Dieser Vorteil kann jedoch durch den Einsatz vieler, oft sehr komplexer, Frameworks verloren gehen.

3.6 Hybridansätze

Der Großteil der Aussagen und Bewertungen in diesem Kapitel basieren auf praktischen Erfahrungen, die sich auch in den Arbeiten [12, 13, 29–31, 58, 89] wiederfinden.

Plattformen

Die mit den hybriden Entwicklungsansätze Apache Cordova und Appcelerator Titanium erstellten Anwendungen sind auf allen in Kapitel 2.2 vorgestellten Plattformen lauffähig. Dadurch sind sie eine gute Wahl, wenn eine Vielzahl von Geräten unterstützt werden soll.

Native Funktionen

Auch für die Verwendung von nativen Funktionen gilt, dass mit beiden Ansätzen auf den Großteil der plattformspezifischen Funktionalität über eine einheitliche API zugegriffen werden kann.

Benutzerfreundlichkeit

Während Apache Cordova Applikationen, im Zusammenhang mit der Benutzeroberfläche, mit den gleichen Problemen wie Webapplikationen zu kämpfen haben, haben Applikationen, die mit Appcelerator Titanium entwickelt werden, eine native Benutzeroberfläche. Beide Ansätze unterstützen den jeweiligen Anwendungslebenszyklus der jeweiligen Plattform.

Geschwindigkeit

PhoneGap Anwendungen erreichen aufgrund derselben eingesetzten Technologien eine mit Webapplikationen vergleichbare Ausführungsgeschwindigkeit. Die Startgeschwindigkeit ist wegen der stellenweise lokal vorhandenen Daten meist etwas schneller. Appcelerator Titanium Anwendungen starten in etwa gleich schnell, haben jedoch bei komplexeren Benutzeroberflächen mit einer abnehmenden Ausführungsgeschwindigkeit zu kämpfen.

Kommunikation

Mit beiden Ansätzen können die plattformspezifischen Kommunikationsmöglichkeiten genutzt werden. Die serielle Kommunikation, die selten auf Geräten mit Linux basierender Plattform unterstützt wird, ist in Apache Cordova über den Einsatz von Plug-Ins realisierbar.

Verteilung

Auch die Verteilung der Hybridapplikationen über den jeweilige App-Store stellt kein Problem dar. Beide gewählten hybriden Ansätze bieten sehr gute Anleitungen, die das Erstellen der jeweils plattformspezifischen Installationspakete beschreiben.

IDE

Für die Entwicklung von Apache Cordova Applikationen können die selben IDEs wie für Webapplikationen benutzt werden. Zusätzlich ist ein Verwenden der plattformspezifischen IDEs stellenweise sinnvoll oder sogar notwendig. Titanium Studio ist eine sehr ausgereifte Entwicklungsumgebung, die für die Umsetzung Titanium-spezifischer Aufgaben bestens geeignet ist.

Benutzeroberfläche

Für Apache Cordova Applikationen kann, wie auch für Webapplikationen, eine Vielzahl an WYSIWYG-Editoren verwendet werden. Für die Entwicklung von Appcelerator Titanium Applikationen steht jedoch kein WYSIWYG-Editor zur Verfügung, wodurch das Erstellen der Benutzeroberfläche erschwert wird.

Dokumentation und Support

Für beide Ansätze existiert umfangreiche Dokumentationen, die alle Schritte von der Installation der Entwicklungsumgebungen bis zum Erstellen der einzelnen Installationspakete sehr gut beschreiben. Support für Appcelerator Titanium kann, über die in Kapitel 2.4.2 bereits erwähnte, Appcelerator Plattform gekauft werden. Der Community-Support von Titanium ist nicht sehr ausgeprägt, hingegen ist die unterstützende Community von Apache Cordova, ähnlich wie bei den nativen Ansätzen, sehr aktiv und Probleme können somit schnell gelöst werden.

Skalierbarkeit

Beim Thema Skalierbarkeit gelten für Apache Cordova die selben Bedingungen wie für Webapplikationen. Auch Titanium bietet Konzepte, die die Modularisierung unterstützen und somit ein paralleles Arbeiten beschleunigen.

Wiederverwendbarkeit

Die Wiederverwendbarkeit von für Apache Cordova geschriebene Applikationen verhält sich gegenteilig zu jener von Webapplikationen. Wenn keine nativen Funktionen eingesetzt werden, kann der für Apache Cordova geschriebene Code sehr einfach als Webapplikationen veröffentlicht werden. Für Titanium Anwendungen gilt, dass geschriebener JavaScript-Code auch für andere, auf JavaScript aufbauende Ansätze verwendet werden kann, wenn er nicht auf die API von Titanium zurückgreift.

Lerneinsatz

Die Einstiegshürde für die Entwicklung von Appcelerator Titanium Applikationen ist durch einige Framework-spezifische Aspekte als erhöht zu betrachten. Für Apache Cordova hingegen gelten dieselben Aussagen wie für Webapplikationen.

3.7 Modellgetriebene Entwicklung

Mit dem in [32] vorgestellten Ansatz von Heitkötter et al. existiert zwar ein Forschungsprojekt, das die Erstellung von Business-Applikationen für Android und iOS ermöglicht, doch nach eigenen Aussagen befindet sich dieses erst in einem prototypischen Stand und benötigt somit noch einiges an Entwicklungsaufwand. Die Entwicklung eines eigenen modellgetriebenen Ansatzes würde den Aufwand der Entwicklung einer plattformunabhängigen Applikation bei weitem übertreffen. Nichtsdestotrotz basieren stellenweise die Aussagen in diesem Kapitel auf der Annahme, dass ein ausgereifter modellgetriebener Ansatz existiert, der produktiv eingesetzt werden kann.

Plattformen

Modelle sind unabhängig von der Zielplattform. Somit eignet sich die modellgetriebene Softwareentwicklung auch für die Entwicklung von plattformunabhängigen Applikationen. Wobei Generatoren für alle zu unterstützenden Plattformen vorhanden sein müssen.

Native Funktionen

Weil mit modellgetriebenen Ansätzen native Applikationen erzeugt werden können, kann auch auf native Funktionen zurückgegriffen werden, wenn entsprechende Codegeneratoren zur Verfügung stehen bzw. entwickelt wurden.

Benutzerfreundlichkeit

Mit entsprechendem Aufwand können mit modellgetriebener Entwicklung erstellte Applikationen eine native Optik und Haptik aufweisen.

Geschwindigkeit

Weil es sich bei den erstellen Applikationen um native Applikationen handelt, können mit modellgetriebenen Ansätzen native Geschwindigkeiten erreicht werden.

Kommunikation

Auch hier gelten dieselben Aussagen wie im Bezug auf native Applikationen, wobei wieder der Aufwand der zu entwickelten Codegeneratoren nicht unterschätzt werden darf.

Verteilung

Der für die entsprechende Plattform generierte Code muss, wie bei nativen Ansätzen, mit den in den SDKs enthaltenen Tools in das entsprechende Applikationsformat gebracht werden und kann anschließend als native Applikation in den App-Stores bereitgestellt werden.

IDE

Mit dem EMF-Projekt steht Entwicklern ein ausgereiftes Projekt zur Verfügung, das alle benötigten Technologien für die Entwicklung eines modellgetriebenen Ansatzes beinhaltet. Für das Arbeiten mit dem generierten Code können die IDEs der nativen Ansätze verwendet werden.

Benutzeroberfläche

Um die Vorteile eines WYSIWYG-Editors nutzen zu können, müsste dazu ein Editor entwickelt werden. Die damit erstellte Benutzeroberfläche würde das Basismodell für die plattformspezifische Benutzeroberflächen-Generierung bilden. Dieses Vorhaben scheint jedoch mit sehr viel Aufwand verbunden.

Dokumentation und Support

Die Dokumentation der einzelnen Technologien innerhalb des EMF-Projekts kann als sehr gut bezeichnet werden, auch der Community-Support ist sehr stark ausgeprägt.

Skalierbarkeit

Ein ausgereifter modellgetriebener Ansatz würde auch das parallele Arbeiten von mehreren Entwicklern erlauben. Dabei könnte auf bewährte Konzepte zurückgegriffen werden.

Wiederverwendbarkeit

Dieser Punkt ist im Zusammenhang mit modellgetriebener Entwicklung schwer zu beantworten. Zum einen könnten bereits umgesetzte Modellkonzepte und auch damit erstellter Code wiederverwendet werden, zum anderen kann der Entwicklungsansatz nicht ohne enormen Aufwand gewechselt werden.

Lerneinsatz

Es stehen zwar ausgereifte Tools für die Entwicklung eines modellgetriebenen Ansatzes zur Verfügung, jedoch existiert, wie anfangs in diesem Kapitel erwähnt, noch keine fertige Toolsuite, die für die Entwicklung einer Applikation eingesetzt werden kann. Die Einarbeitung in einen solchen Ansatz würde das Erlernen vieler Technologien mit sich bringen. Dies könnten extra für den Ansatz entwickelte DSLs oder für viele Entwickler weniger bekannte allgemeine Konzepte der modellgetriebenen Softwareentwicklung sein.

3.8 Vergleich der Entwicklungsansätze

Tabelle 3.2 zeigt zusammengefasst die Ergebnisse der in diesem Kapitel beschriebenen Analyse der Entwicklungsansätze. Dabei wurde, aufgrund des aktuell Nichtvorhandenseins eines produktiv einsetzbaren Ansatzes, die modellgetriebene Softwareentwicklung nicht berücksichtigt.

	Nativ	**Web**	**Hybrid**
Plattformen	1 pro Ansatz	Android, iOS, Windows Phone	Android, iOS, Windows Phone
Verteilung	Sehr einfach, SDK-Tools	Sehr einfach, Webserver	Einfach, SDK-Tools
Native Funktionen	Ja	Nein	Großteil
Benutzerfreundlichkeit	Ausgezeichnet	Mäßig	Sehr gut
Geschwindigkeit	Ausgezeichnet	Sehr gut	Sehr gut
Kommunikation	Ausgezeichnet	Mäßig	Sehr Gut
Entwicklungsumgebung	Android: Eclipse mit ADT-PlugIn, iOS: Xcode, Windows Phone: Microsoft Visual Studio	z.B. WebStorm	Apache Cordova: z.B. WebStorm, Titanium: Titanium Studio
Benutzeroberfläche (WYSIWYG)	Ja	Ja	Ja
Dokumentation & Support	Ausgezeichnet	Sehr gut	Sehr gut
Skalierbar	Ja	Ja	Ja
Wiederverwendbar	Nein	Teilweise	Teilweise
Lerneinsatz	Moderat	Gering, abhängig von Frameworks	Moderat bis hoch

Tabelle 3.2: Vergleich der Entwicklungsansätze.

KAPITEL 4

Entwurf und Implementierung

4.1 Übersicht

Dieses Kapitel widmet sich der Portierung eines bestehenden Visualisierungssystems für Stationsleittechnik, Fernwirktechnik und Schutztechnik auf mobile Plattformen. In Unterkapitel 4.2 wird die Funktionsweise des vorhandenen Systems im Detail behandelt. Zuerst werden die Anwendungsfälle beschrieben und anschließend die einzelnen Komponenten vorgestellt, die für die Visualisierung benötigt werden. Desweiteren wird die Interaktion zwischen diesen Komponenten diskutiert. Unterkapitel 4.3 beschäftigt sich mit der Auswahl eines Entwicklungsansatzes. Es werden Annahmen getroffen, die im Zusammenhang mit der Portierung stehen und die grundlegende Architektur vorgestellt. In Unterkapitel 4.4 wird die Implementierung des Systems für mobile Plattformen präsentiert. Es wird die Testhardware und die eingesetzte Entwicklungsumgebung vorgestellt und abschließend die Implementierung beschrieben.

4.2 Bestehendes System

In diesem Kapitel wird das zu portierende, bestehende Visualisierungssystem der Sprecher Automation GmbH, soweit wie für diese Arbeit vonnöten, vorgestellt.

4.2.1 Anwendungsfälle

Abbildung 4.1 zeigt die Anwendungsfälle des bestehenden Systems. Im Folgenden werden diese Anwendungsfälle vorgestellt. Die Vorbedingung für alle Anwendungsfälle ist, dass die Initialisierungsphase des Systems erfolgreich abgeschlossen wurde (siehe Unterkapitel 4.2.3).

Prozessbilder anzeigen

Der aktuelle Zustand des überwachten Prozesses wird mithilfe mehrerer Seiten angezeigt. Der Benutzer kann auf der aktuell angezeigten Seite Objekte auswählen und an diesen Schaltvorgänge durchführen oder Parametereinstellungen ändern.

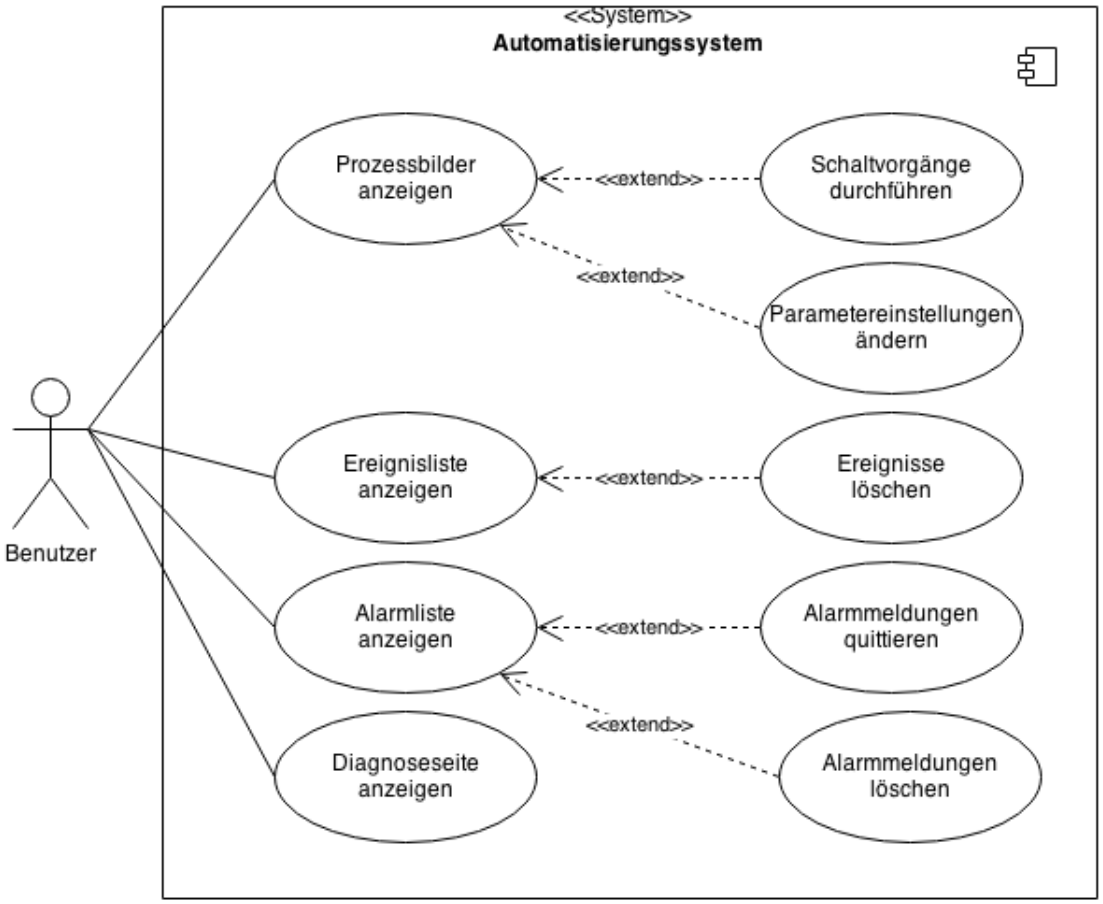

Abbildung 4.1: Anwendungsfälle des Systems.

Schaltvorgänge durchführen

Nachdem der Benutzer ein Objekt ausgewählt hat, das Schaltvorgänge unterstützt, kann der Zustand des Objekts geändert werden, z.B. von Ein zu Aus. Durch diese Änderung des Zustandes wird die Anzeige aktualisiert.

Parametereinstellungen ändern

Nachdem ein Objekt ausgewählt wurde, das Parameteränderungen unterstützt, können Parameter verändert werden, wie beispielsweise Sollwerte. Durch das Ändern von Parametern wird der Zustand des Prozesses verändert und die Anzeige aktualisiert.

Ereignisliste anzeigen

Es werden alle aufgetretenen Ereignisse in Listenform dargestellt und dem Benutzer wird die Möglichkeit geboten, Ereignisse zu löschen.

Ereignisse löschen

Werden Ereignisse gelöscht, werden sie permanent aus der Ereignisliste entfernt.

Alarmliste anzeigen

Es werden alle aufgetretenen Alarmmeldungen in Listenform dargestellt. Es wird anhand der Darstellung zwischen bereits quittierten und zu quittierenden Alarmmeldungen unterschieden. Bereits quittierte Meldungen können gelöscht und zu quittierende Meldungen quittiert werden.

Alarmmeldungen löschen

Werden Alarmmeldungen gelöscht, werden sie permanent aus der Alarmliste entfernt.

Alarmmeldungen quittieren

Werden Alarmmeldungen quittiert, wird ihre Darstellung in der Alarmliste geändert.

Diagnoseseite anzeigen

Aktuelle Informationen über alle in der Automatisierungseinheit enthaltenen Hardwarekomponenten werden angezeigt. Dazu gehören unter anderem die Zustände von Ein- und Ausgängen.

4.2.2 Komponenten

Für die Realsierung der Anwendungsfälle werden mehrere Komponenten eingesetzt. Desweiteren sind zusätzliche Komponenten an der Erstellung eines Visualisierungsprojektes beteiligt. Im Folgendem werden diese Komponenten und ihre Interaktionen beschrieben (siehe Abbildung 4.2).

Display-Logik

Die Logik der Visualisierung befindet sich in der Firmware der Automatisierungseinheit. Sie beinhaltet den semantischen Zustand der Visualisierung und gibt diesen mithilfe von Steuer- bzw. Zeichenbefehlen über Inter-process communication (IPC) an den Display Panel Manager (DPM) weiter.

Display Panel Manager

Der DPM ist für die Kommunikation des Automatisierungsgerätes mit der Bedieneinheit beziehungsweise den Meldeeinheiten verantwortlich. Er gibt die per IPC von der Logik erhaltenen Befehle auf der seriellen Schnittstelle aus und erwartet Antworttelegramme oder die Meldung von Tastenereignissen, die wiederum mittels IPC an die Visualisierungslogik weitergereicht werden. Der DPM dient somit als Vermittler zwischen Automatisierungseinheit und Bedieneinheit. Zusätzlich ist er für die Erkennung von Verbindungsproblemen auf der Schnittstelle verantwortlich.

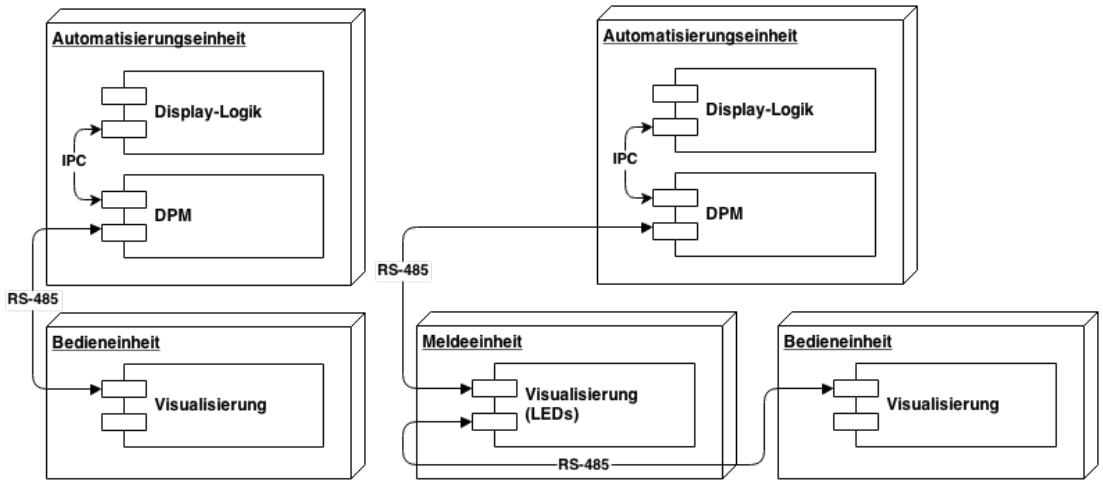

Abbildung 4.2: Aufbau des bestehenden Systems. Links ohne und rechts mit Meldeeinheit.

Bedieneinheit

Die Bedieneinheit ist über eine RS-485 Verbindung an der Automatisierungseinheit angeschlossen. Auf der Bedieneinheit werden die zu visualisierenden Daten auf einem 320x240 Pixel messenden Display (monochrom oder farbig) ausgegeben (siehe Abbildung 4.3). Desweiteren sind auf der Bedieneinheit 25 frei parametrierbare LEDs, ein oder zwei Schlüsselschalter und zwölf Bedien- und Navigationstasten vorhanden, mit denen der Prozess und die Parametereinstellungen gesteuert bzw. überwacht werden können.

Meldeeinheit

Es können bis zu zwei Meldeeinheiten zwischen die Verbindung des Automatisierungsgerätes mit der Bedieneinheit gesteckt werden. Eine Meldeeinheit bietet 100 frei parametrierbare, mit verschiedenen Zuständen belegbare LEDs. Damit kann es als erweiterte Statusanzeige verwendet werden. Zusätzlich bietet es einen frei parametrierbaren Schlüsselschalter. Abbildung 4.4 zeigt eine in Betrieb befindliche Meldeeinheit. Ankommende Datenpakete werden an die danach angeschlossene Melde- oder Bedieneinheit weitergeleitet.

Display Editor

Mit dem Display Editor können Visualisierungsprojekte erstellt werden. Dabei werden mehrere anzuzeigende Seiten definiert, die mit dynamischen und statischen Inhalten gefüllt werden können. Aus allen statischen Inhalten wird ein Hintergrundbild für die Seite gebildet. Zu den statischen Inhalten zählen:

- Texte
- Formatierbare Texte

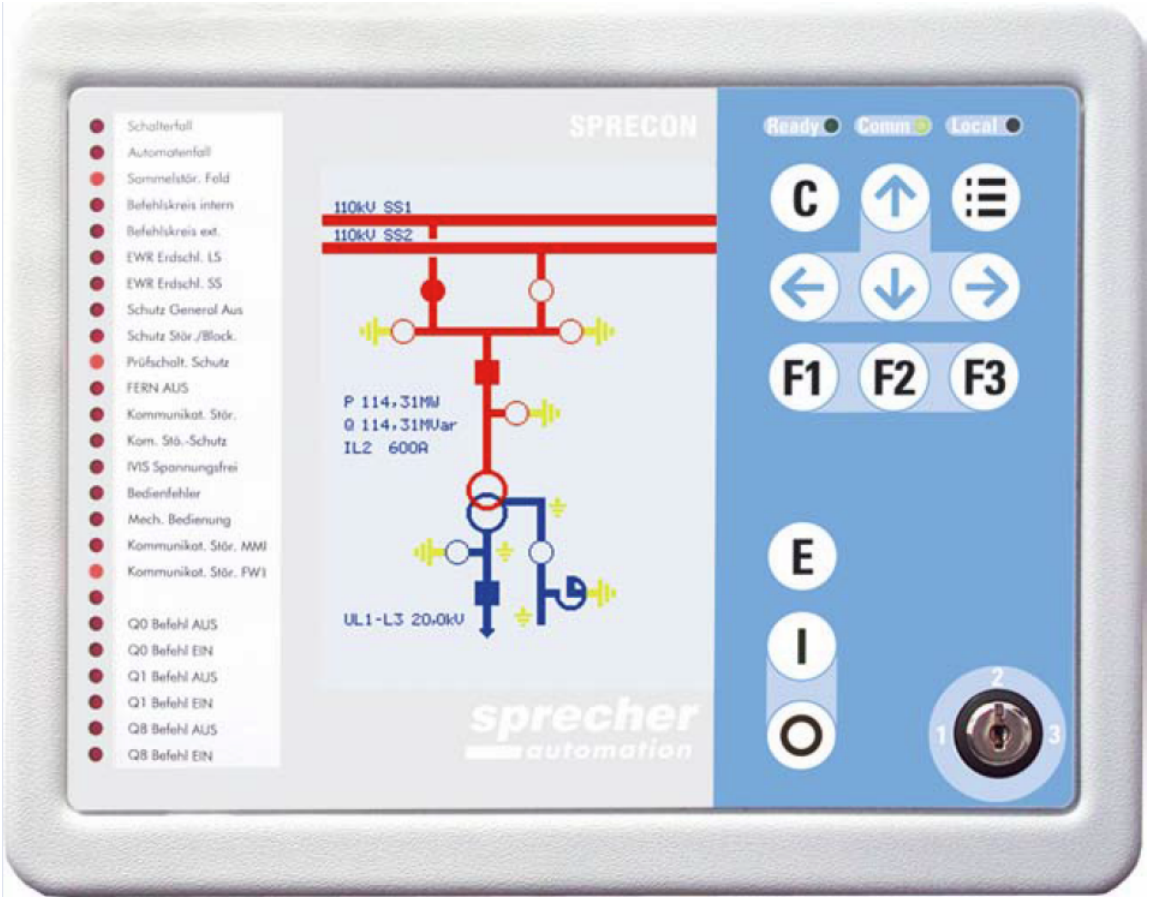

Abbildung 4.3: Eine in Betrieb befindliche Bedieneinheit mit farbigem Display.

- Bilder (mit der Dateiendung .bmp)
- Geometrische Figuren (Rechtecke, Dreiecke, Kreise, etc.)

Dynamischen Inhalten muss die Adresse eines Datenpunktes zugewiesen werden, der innerhalb der Automatisierungseinheit definiert ist. Zu den dynamischen Inhalten gehören:

- Schaltobjekte können durch charakteristische Schaltzustände dargestellt werden, wie beispielsweise ein, aus oder undefiniert. Dazu wird jedem Zustand ein Symbol aus vordefinierten oder selbst erstellten Symbolbibliotheken zugeordnet. Zusätzlich kann einem Schaltobjekt ein Cursor zugewiesen werden, mit dessen Hilfe ein Schaltobjekt ausgewählt werden kann, um Schaltvorgänge durchzuführen.
- Binärobjekte können ebenfalls die Zustände ein, aus und undefiniert darstellen. Zusätzlich können Binärobjekte blinkend dargestellt werden.

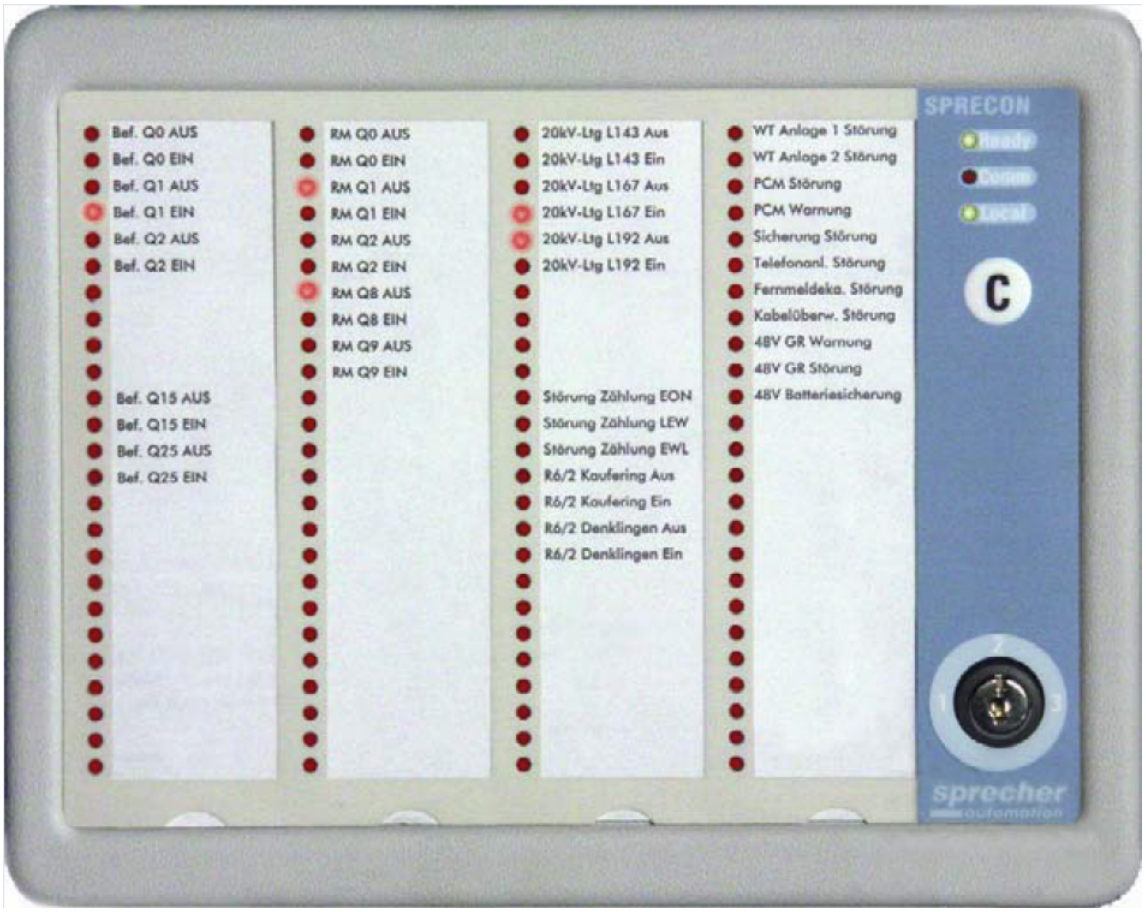

Abbildung 4.4: Eine in Betrieb befindliche Meldeeinheit.

- Dynamische Texte dienen der Anzeige von Texten, die zur Laufzeit aktualisiert werden können.
- Messwertbalken

Abbildung 4.5 zeigt ein Display Editor Projekt, das aus mehreren statischen und dynamischen Inhalten besteht.

4.2.3 Funktionsweise

Initialisierung

Abbildung 4.6 zeigt die Initialisierung der Bedieneinheit. Die Firmware der Automatisierungseinheit beinhaltet auch die Firmware der Bedieneinheit. Nach dem Anschließen einer Bedieneinheit wird über das definierte Kommunikationsprotokoll abgefragt, welche Firmwareversion auf

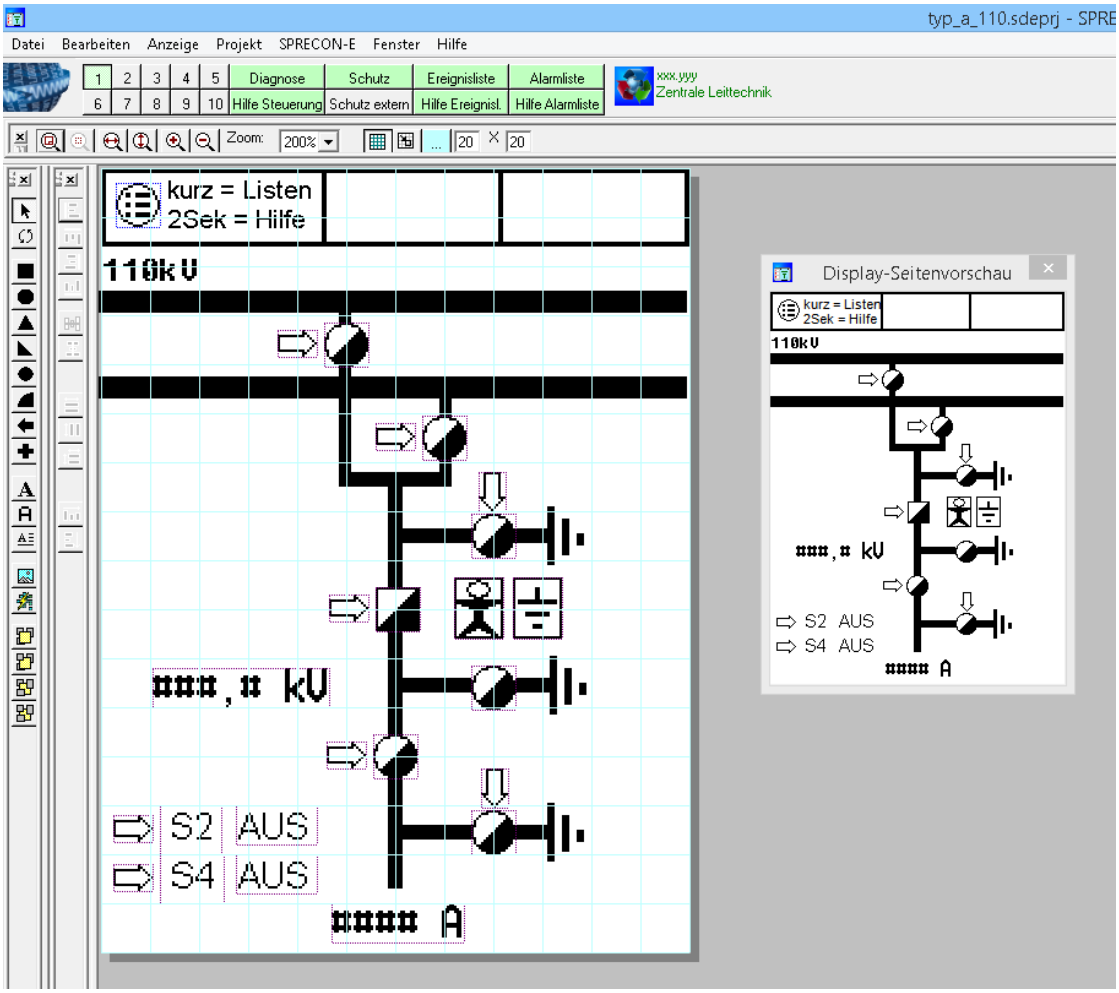

Abbildung 4.5: Display Editor Projekt mit statischem und dynamischem Inhalt auf der aktuellen Seite.

der Bedieneinheit vorhanden ist. Wenn die vorhandene Version nicht mit der in der Automatisierungseinheit hinterlegten kompatibel oder keine Firmware vorhanden ist, wird die Firmware der Automatisierungseinheit zur Bedieneinheit übertragen. Dabei wird zwischen monochromen und farbigen Bedieneinheiten folgendermaßen unterschieden:

- Bei monochromen Bedieneinheiten ist, falls noch keine Firmware vorhanden, im Auslieferungszustand ein Bootprogramm im Flashspeicher hinterlegt, das für die Übertragung der Firmware auf die Bedieneinheit verwendet wird. Sollte bereits eine Firmware vorhanden sein, so ist diese ebenfalls im Stande, eine andere Firmware entgegenzunehmen. Dazu werden die ankommenden Firmwaredaten in vorgegebene Bereiche im Flashspeicher geschrieben. Nach erfolgreicher Übertragung wird die Bedieneinheit mit der soeben übertragenen Firmware neu gestartet.

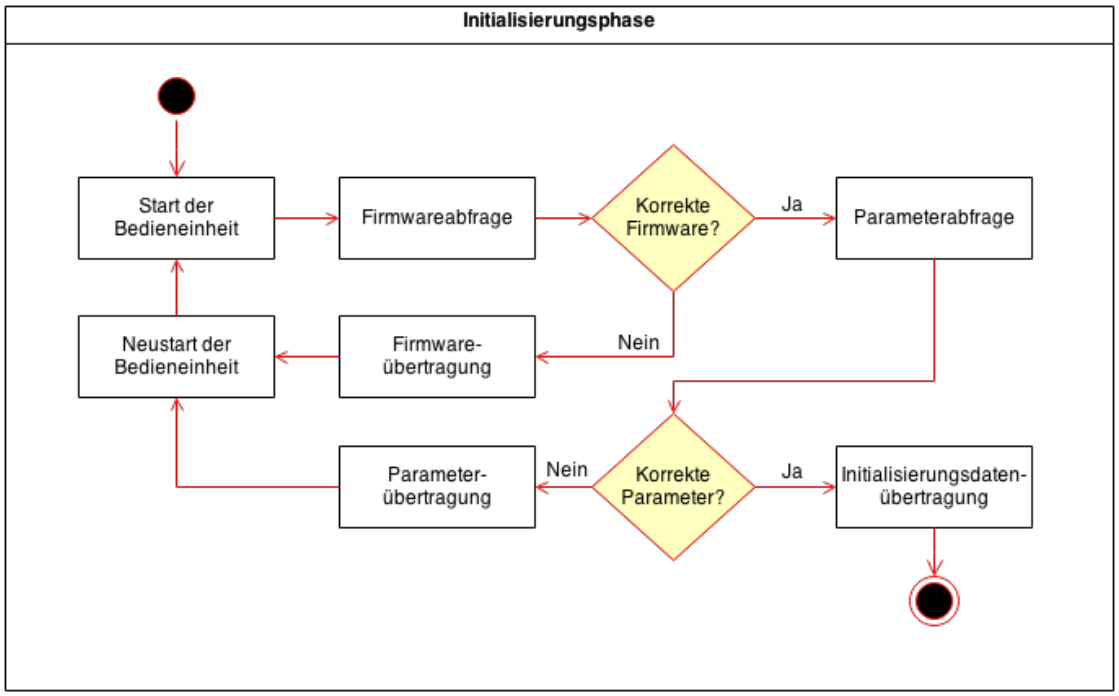

Abbildung 4.6: Initialisierungsphase der Bedieneinheit.

- Auf farbigen Bedieneinheiten läuft im Auslieferungszustand ein Embedded-Linux-System, auf dem die Firmware als ausführbare Datei im Dateisystem abgelegt ist. Auch hier kann die Firmware durch die der Automatisierungseinheit ersetzt werden. Dazu wird die neue Firmware als Zip-Datei übertragen und anschließend wird der aktuelle Firmwareprozess beendet und die neue Firmware entpackt und gestartet.

Nach dem Start der korrekten Firmwareversion werden die Parameter (basierend auf den Daten des Display-Editor-Projekts) von der Automatisierungseinheit zur Bedieneinheit übertragen. Auch hier unterscheiden sich monochrome und farbige Bedieneinheiten. Parameter werden auf der monochromen Bedieneinheit wieder direkt in den Flashspeicher geschrieben. Im Gegensatz dazu, wird bei der farbigen Bedieneinheit wieder eine Zip-Datei übertragen. Nach der erfolgreichen Übertragung der Parameter, wird die Bedieneinheit neu gestartet. Beim Start werden die vorhandenen Parameter von der Firmware eingelesen. Anschließend werden einige Initialisierungsdaten übertragen, wozu auch der aktuelle Stand der dynamischen Schalt- und Binärobjekte zählt. Abschließend wird mit dem Befehl für das Aufschalten der ersten Seite die Initialisierungsphase abgeschlossen.

Betrieb

Im Normalbetrieb können Änderungen der Anzeige auf zwei verschiedene Arten erreicht werden:

Polling Der DPM sendet in einem definierten Intervall Änderungsabfragen zur Bedieneinheit. Auf diese Änderungsabfragen wird nur geantwortet, wenn auf der Bedieneinheit Tasten gedrückt wurden oder sich die Stellung eines Schlüsselschalters verändert hat (siehe Abbildung 4.7). Diese Information wird vom DPM an die Visualisierungslogik weitergereicht, die dann die entsprechenden Aktionen durchführt und somit den internen Zustand der Visualisierung aktualisiert und diese Änderungen wiederum an die Bedieneinheit weiterleitet.

Pushing Die Visualisierungslogik wird von anderen Prozessen (z.B. Leittechnik, Schutztechnik) über Änderungen der visualisierten Daten informiert und gibt diese Änderungen direkt an die Bedieneinheit weiter, um eine aktuelle Präsentation des Prozesses zu gewährleisten.

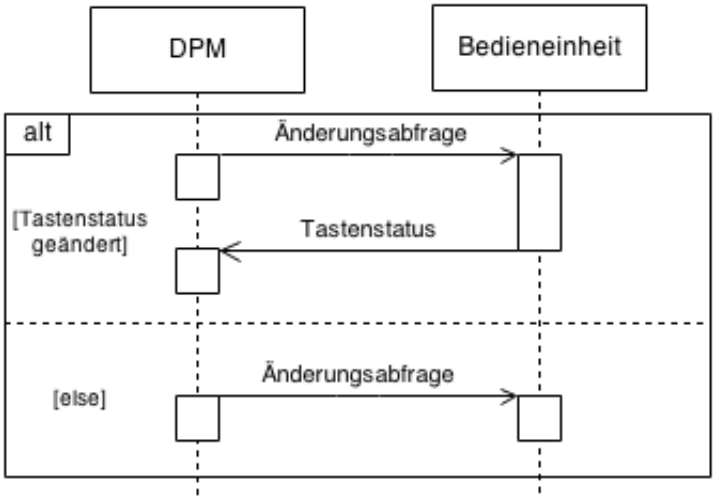

Abbildung 4.7: Empfang einer Änderungsabfrage.

4.2.4 Architektur

Aus der Beschreibung der einzelnen Komponenten und der Funktionsweise des Systems lässt sich herauslesen, dass die Architektur eine starke Ähnlichkeit zu dem in Kapitel 2.3.1 vorgestellten MVC-Entwurfsmuster aufweist (siehe Abbildung 4.2).

Modell

Für die Bedieneinheit sind die angezeigten Daten auf dem Display der Kern des Modells. Wenn jedoch das Gesamtsystem betrachtet wird, ist es nur eine Sicht auf das Gesamtmodell, das in der Display-Logik lokalisiert ist.

View

Die Bedieneinheit bildet die View des Systems, die statische und dynamische Inhalte darstellen kann. Die View wird durch die Darstellungsanweisungen vom View-Programmteil der Display-Logik gesteuert.

Controller

Die Bedieneinheit nimmt Benutzeraktionen entgegen und leitet diese an den DPM weiter. Der setzt wiederum die Impulse, um das Modell zu verändern. Somit bildet der DPM zusammen mit dem für die Kommunikation verantwortlichen Programmteil den Controller des Systems. Es wird jedoch nicht auf das klassische Beobachter-Entwurfsmuster zurückgegriffen, um Änderungen am Modell zu erkennen, sondern der DPM ist aktiver Vermittler zwischen Modell und View.

4.3 Entwurf

Die Implementierung umfasst die Portierung der Bedieneinheit (monochrom und farbig) auf eine mobile Plattform. Ein Eingriff in die Firmware der Automatisierungseinheit hätte den Rahmen dieser Arbeit gesprengt und war von Sprecher Automation GmbH explizit nicht erwünscht, wodurch in Kapitel 4.3.2 einige Annahmen bezüglich der Kommunikation getroffen werden, weil sich diese nicht ohne große Eingriffe in die Firmware hätten ändern lassen.

4.3.1 Entwicklungsansatz

Von Seiten Sprecher Automation GmbH gab es die Anforderung, dass, wenn möglich, die portierte Applikation nicht nur auf mobile Plattformen beschränkt ist. Um diese Anforderung zu erfüllen und gleichzeitig eine Vielzahl von Plattformen zu unterstützen, wurde der Ansatz der Entwicklung einer Webapplikation gewählt. Diese kann auf den mobilen Plattformen in eine Apache Cordova Applikation eingebettet werden, damit der Benutzer auf der mobilen Plattform eine installierte Anwendung starten kann und nicht direkt von einem Webbrowser abhängig ist.

4.3.2 Annahmen

Die Kommunikation über die serielle Schnittstelle kann nicht, wie bereits erwähnt, ohne großen Aufwand und Eingriffe in die Automatisierungsfirmware geändert werden. Desweiteren wird serielle Kommunikation nur selten auf mobilen Geräten unterstützt, die meist auf Linux basieren. Das von allen modernen Webbrowsern unterstützte Kommunikationsprotokoll WebSocket, das bereits in Kapitel 2.3.2 näher beschrieben wurde, wäre für die plattformunabhängige Kommunikation besser geeignet. Daher wird innerhalb der Webapplikation auf WebSocket Kommunikation zurückgegriffen. Da jedoch für die Kommunikation über WebSockets auch ein WebSocket-Server vorhanden sein muss, wird ein Kommunikationsserver implementiert, der die per serieller Kommunikation erhaltenen Befehle unverändert über eine WebSocket-Verbindung an die Webapplikation weiterreicht. Dieser Server und damit verbundene Einschränkungen werden in

Kapitel 4.4.4 im Detail behandelt. Die Entwicklung des Kommunikationsservers basiert auf der Annahme, dass es aufgrund des Tunneling-Prinzips für die Funktionalität der Webapplikationen egal ist, ob die Befehle der Automatisierungseinheit direkt über die serielle Schnittstelle empfangen werden, oder ob sie vom Kommunikationsserver bereitgestellt werden.

4.3.3 Architektur

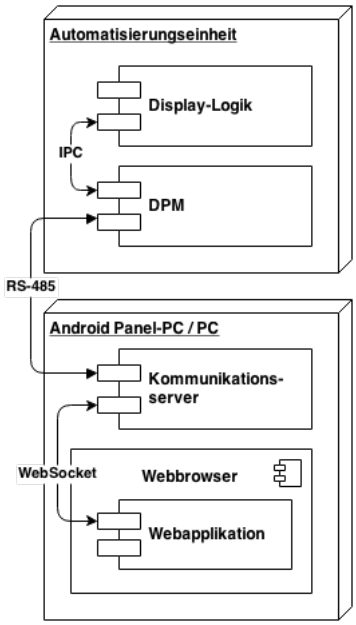

Abbildung 4.8: Aufbau des portierten Systems.

In Kapitel 4.2 wurden die Komponenten und die Architektur des bestehenden Systems beschrieben. Aufbauend auf diesen Informationen, dem ausgewählten Entwicklungsansatz und der Annahmen ergibt sich für die Implementierung eine ebenfalls auf dem MVC-Entwurfsmuster basierende Architektur (siehe Abbildung 4.8).

Modell

Für die Webapplikation sind die auf der HTML-Seite angezeigten Daten das Modell, das jedoch, wie zuvor, nur eine Sicht auf das Gesamtmodel darstellt.

View

Die Webapplikation übernimmt in dieser Architektur den Part der View und ersetzt somit die Bedieneinheit. An der Steuerung der View mithilfe von Darstellungsanweisungen der Display-Logik ändert sich nichts.

Controller

Benutzeraktionen werden von der Webapplikation entgegen genommen und an den Kommunikationsserver weitergeleitet. Dieser leitet sie entsprechend der bestehenden Architektur an den DPM weiter. Somit wird der Controller um den Kommunikationsserver erweitert, agiert jedoch weiterhin als direkter Vermittler zwischen Modell und View.

4.4 Implementierung

4.4.1 Hardware

Sprecher Automation GmbH stellte für die Implementierung ein Vorserienmodell des auf Android basierten Panel-PCs IOVU-07F-AD der Firma IEIMobile zur Verfügung. Der Panel-PC ist mit seriellen Schnittstellen ausgestattet und eignet sich somit für die Portierung der Bedieneinheit (siehe Abbildung 4.9).

Abbildung 4.9: Schnittstellen an der Unterseite des IOVU-07F-AD Panel-PCs [11].

4.4.2 Entwicklungsumgebung

Bevor mit der Implementierung begonnen wurde, musste eine geeignete IDE für die Entwicklung gefunden werden, die gute Unterstützung für die Erstellung von Webapplikationen bietet. Desweiteren sollte es möglich sein, den Kommunikationsserver in der Programmiersprache Java zu realisieren, um ihn direkt in die Android-Applikation einbinden zu können.

Die Wahl fiel aus den folgenden Gründen auf die IDE Eclipse:

- Eclipse ist frei verfügbar und kann somit verwendet werden, ohne Lizenzgebühren entrichten zu müssen
- Java Unterstützung
- Eclipse ist über Plug-Ins erweiterbar (z.B. JSDT-Plug-In für JavaScript Unterstützung)

4.4.3 Frameworks

Im Folgenden werden die für die Implementierung verwendeten JavaScript Frameworks vorgestellt.

jQuery

jQuery wurde bereits in Kapitel 2.3.3 vorgestellt und wurde für den vereinfachten Zugriff auf die HTML-Elemente und den einfacheren Umgang mit Ereignissen verwendet.

RequireJS

RequireJS erlaubt es dem Entwickler, eine JavaScript-Applikation in Module zu unterteilen. Dabei können Abhängigkeiten zwischen Modulen festgelegt werden, wobei sich RequireJS um die Reihenfolge der Instanziierung der einzelnen Module kümmert. RequireJS unterstützt alle modernen Webbrowser und lässt sich mit einer einzigen Codezeile in eine Applikation einbinden (siehe Listing 4.1) [7]. Dafür muss lediglich die *require.js*-Datei über ein HTML Script-Element eingebunden werden. Mit dem Attribut *data-main* wird RequireJS mitgeteilt, welches Script nach dem erfolgreichem Laden von RequireJS ausgeführt werden soll.

```
1 <script data-main="app.js" src="require.js"></script>
```

Listing 4.1: Code zum einbinden von RequireJS.

JSZip

JSZip ist eine JavaScript Bibliothek, die für das Erstellen, Lesen und Bearbeiten von Zip-Dateien verwendet werden kann. Dafür stellt JSZip eine einfache, leicht verständliche API zur Verfügung [85]. Listing 4.2 zeigt exemplarisch, wie eine Zip-Datei mit JSZip erstellt werden kann.

```
1 var zip = new JSZip();
2 zip.file("Hallo.txt", "Hallo");
3 zip.folder("Verzeichnis");
4 zip.file("Verzeichnis/Welt.txt", "Welt");
```

Listing 4.2: JavaScript-Code für das Erstellen einer Zip-Datei mit mehreren Einträgen mithilfe von JSZip.

4.4.4 Kommunikationsserver

Der Kommunikationsserver ist ein in Java geschriebenes Programm, das die über serielle Kommunikation erhaltenen Befehle des DPMs über eine WebSocket-Verbindung an die Applikation weiterleitet, ohne sie dabei zu verändern. Dasselbe gilt für die gegengesetzte Richtung. Von der Applikation zum Kommunikationsserver gesendete Antworten werden von diesem unverändert an den DPM weitergereicht (siehe Abbildung 4.10).

Der Kommunikationsserver wurde auf zwei verschiedene Arten implementiert:

- Als Java-Applikation für das Ausführen auf einem herkömmlichen PC, der sich im selben Netzwerk wie das mobile Gerät befindet, das als Bedieneinheit verwendet wird. Somit können alle Geräte als Bedieneinheit verwendet werden, auf denen die Applikation

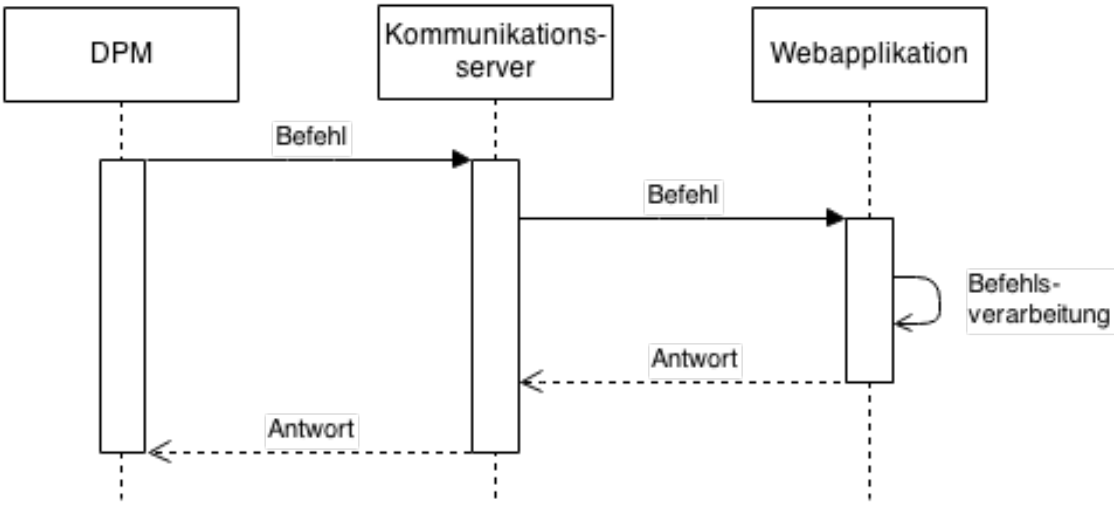

Abbildung 4.10: Kommunikation zwischen DPM und der Applikation.

ausgeführt werden kann. Der PC agiert dabei als WebSocket-Server und ist mit der Automatisierungseinheit über eine serielle Schnittstelle verbunden. Desweiteren kann der PC selbst die Applikation im Webbrowser ausführen und somit als Bedieneinheit verwendet werden.

- Integriert in die Apache Cordova-Applikation, die für den Android Panel-PC erstellt wurde.

Diese Trennung in zwei Implementierungen wurde durchgeführt, damit der Panel-PC direkt über die serielle Schnittstelle mit der Automatisierungseinheit kommunizieren kann.

Für den Zugriff auf die serielle Schnittstelle wird in der PC-Variante des Kommunikationsservers auf die Java-Bibliothek java-simple-serial-connector (jSSC)[1] zurückgegriffen. Listing 4.3 zeigt wie die serielle Schnittstelle mit jSSC initialisiert wird. Die Apache Cordova-Applikation für den Android Panel-PC verwendet die von IEIMobile mitgelieferte Bibliothek für den Zugriff auf die serielle Schnittstelle (siehe Listing 4.4). Der WebSocket-Server wurde auf beiden Plattformen mithilfe der Java-Bibliothek Java-WebSocket[2] realisiert.

[1] https://code.google.com/p/java-simple-serial-connector/
[2] http://java-websocket.org/

```
1  class DisplayServer extends WebSocketServer
2                  implements SerialPortEventListener {
3
4          ...
5
6          private SerialPort serialPort;
7
8          ...
9
10         private void initSerial(String port) {
11                 serialPort = new SerialPort(port);
12                 serialPort.openPort();
13                 serialPort.setParams(
14                         SerialPort.BAUDRATE_57600,
15                         SerialPort.DATABITS_8,
16                         SerialPort.STOPBITS_1,
17                         SerialPort.PARITY_NONE);
18                 serialPort.setEventsMask(SerialPort.MASK_RXCHAR);
19                 serialPort.addEventListener(this);
20         }
21
22         ...
23
24 }
```

Listing 4.3: Java-Code für die Initialisierung der seriellen Schnittstelle mit der Bibliothek jSSC.

4.4.5 Projektdateien

Im Folgenden wird der Aufgabenbereich bzw. die Funktionaliät der einzelnen Projektdateien der Webapplikation im Detail behandelt.

index.html

Die Datei *index.html* stellt den Einstiegspunkt in die Webapplikation dar. Sie beinhaltet die Initialisierung des RequireJS-Frameworks und definiert die Komponenten, die auf der Seite angezeigt werden sollen (siehe Abbildung 4.11). Zu diesen Komponenten gehören Buttons, die als Bedien- und Navigationstasten verwendet werden, und Canvas-Elemente, die für die Visualisierung des Displays und der Status-LEDs eingesetzt werden.

values.js

Im *values.js*-Modul sind von mehreren Modulen benötigte Konstanten definiert. Zu diesen gehören unter anderem die Bezeichner für Messagetypen, die den unterschiedlichen Befehlen im Kommunikationsprotokoll zugeordnet sind, um diese beim Empfang zu unterscheiden.

Abbildung 4.11: Aussehen der Webapplikation. Oben Buttons als Bedien- und Navigationstasten, unten Canvas-Elemente für das Anzeigen von LEDs und des Displays.

```
1  class DisplayServer extends WebSocketServer
2                  implements OnPortDataListener {
3
4          private static final String DEVICE = "/dev/ttymxc1";
5          private static final int BAUDRATE = 57600;
6
7          ...
8
9          private SerialLib serial;
10         private Lib lib;
11
12         ...
13
14         private void initSerial(Context context) {
15                 Object service = context.getSystemService("gpio");
16                 if(service != null) {
17                         lib = new Lib(context);
18                         lib.setSerialMode(Lib.MODE_RS485_NAME);
19                         serial = new SerialLib();
20                         serial.setPortDataListener(this);
21                         serial.openDevice(DEVICE, BAUDRATE);
22                 }
23         }
24
25         ...
26
27 }
```

Listing 4.4: Java-Code für die Initialisierung der seriellen Schnittstelle der Apache Cordova-Applikation.

communication.js

Das *communication.js*-Modul ist für die WebSocket-Kommunikation verantwortlich. Es baut beim Start der Applikation eine Verbindung zum Kommunikationsserver auf und ist somit die Schnittstelle der Applikation, um Befehle zu erhalten.

Wenn Daten empfangen werden, wird überprüft, ob es sich um eine Änderungsabfrage handelt. Wenn eine Änderungsabfrage empfangen wurde, wird die Abfrage direkt an das *commandHandler.js*-Modul weitergegeben. Ansonsten werden die Daten in einem Puffer zwischengespeichert. Sobald ein kompletter Paket-Header empfangen wurde, wird anhand des Längenfeldes innerhalb des Headers überprüft, wie groß das gesamte zu empfangende Paket sein soll. Sobald der Puffer die im Header definierte Größe erreicht hat, wird die Checksumme des gesamten Pakets berechnet und mit der Übertragenen verglichen. Bei Übereinstimmung wird das Paket an das *commandHandler.js*-Modul weitergegeben. Sollte dies nicht der Fall sein, wird das Paket verworfen und der Puffer geleert. Das UML-Diagramm in Abbildung 4.12 zeigt die Funktionswiese des *communication.js*-Moduls.

Antwortpakete, die dem *communication.js*-Modul vom *commandHandler.js*-Modul übergeben werden, werden ohne Veränderung an den Kommunikationsserver weitergeleitet.

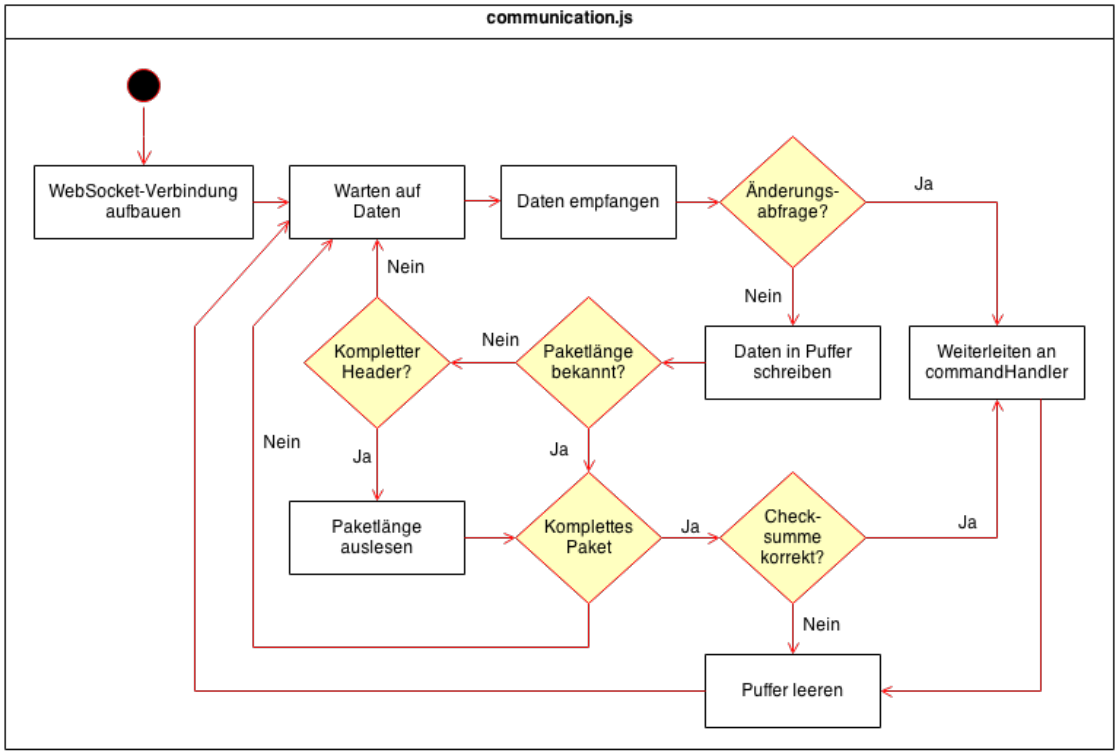

Abbildung 4.12: UML-Diagramm des *communication.js*-Moduls beim Empfang von Daten.

commandHandler.js

Das *commandHandler.js*-Modul extrahiert aus den vom *communication.js*-Modul empfangenen Paketen die einzelnen Bestandteile, die im Kommunikationsprotokoll definiert sind. Danach gibt es die aus den Paketen erhaltenen Informationen an die zuständigen Module weiter und erstellt die entsprechenden Antwortpakete, die dem *communication.js*-Modul übergeben werden (siehe Abbildung 4.13).

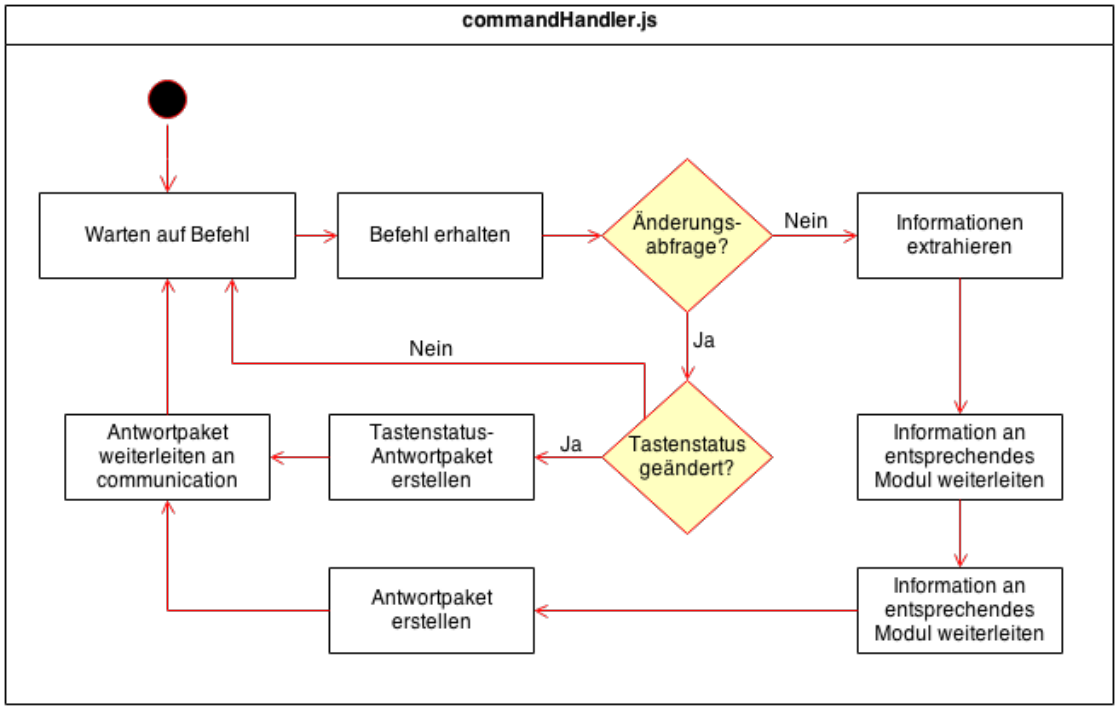

Abbildung 4.13: UML-Diagramm des *commandHandler.js*-Moduls beim Erhalten eines Befehls.

param.js

Das *param.js*-Modul ist für die Verwaltung der Visualisierungsparameter verantwortlich. Beim Start der Applikation wird überprüft, ob bereits Parameter vorhanden sind. Sollten keine Parameter vorhanden sein, wird über einen Aufruf des *page.js*-Moduls eine Fehlermeldung auf dem Display ausgegeben. Das Nichtvorhandensein von Parametern wird jedoch im Zuge der Initialisierungsphase erkannt und die Übertragung der korrekten Parameter eingeleitet. Dazu reicht das *commandHandler.js*-Modul die aus den empfangen Paketen extrahierten Parameterinformationen an das *param.js*-Modul weiter. Nachdem alle Parameter übertragen wurden, werden diese im Web-Storage des Webbrowsers abgelegt, um bei einem erneuten Start der Applikation bereits auf vorhandene Parameter zurückgreifen zu können. Sollten die Parameter beim nächsten Start

nicht zu jenen der Automatisierungseinheit passen, wird das ebenfalls in der Initialisierungsphase erkannt und die Übertragung der korrekten Parameter eingeleitet.

Sind Parameter vorhanden, werden diese eingelesen und die entsprechenden Objekte erzeugt, die für den Betrieb benötigt werden. Zu diesen zählen unter anderem:

- Displayseiten
- Dynamische Objekte, die den entsprechenden Displayseiten zugeordnet werden.
- Hintergrundbilder für Displayseiten
- Bilder, die für die Anzeige von dynamischen Objekten verwendet werden können.

led.js

Das *led.js*-Modul ist für die Anzeige der Status-LEDs zuständig. LEDs können dabei folgende Zustände annehmen:

- Ein
- Aus
- Blinken
- Flackern

Die Zustände werden vom *commandHandler.js*-Modul an das *led.js*-Modul weitergereicht. LEDs werden mithilfe des HTML5 Canvas-Elements (siehe Kapitel 2.3.2) gezeichnet. Listing 4.5 zeigt die Funktion *drawLed*, die für das Zeichnen einer einzelnen LED verwendet wird. Um die Zustände Blinken und Flackern zu realisieren, werden LEDs mit dem entsprechenden Zustand in einem 700 bzw. 125 Millisekunden Intervall neu gezeichnet.

```
1 function drawLed(context, x, y, color) {
2         context.beginPath();
3         context.arc(x,y,10,0,2*Math.PI);
4         context.fillStyle = color.fillStyle();
5         context.fill();
6         context.strokeStyle = colorBorder.fillStyle();
7         context.stroke();
8 }
```

Listing 4.5: JavaScript-Code, für das Zeichnen einer einzelnen LED. Mit der *arc*-Funktion wird ein Kreis an den übergebenen *x*- und *y*-Koordinaten mit einem zehn Pixel großen Radius gezeichnet.

keyHandler.js

Das *keyHandler.js*-Modul ist für die Ereignisbehandlung im Zusammenhang mit den Buttons zuständig. Es beinhaltet alle Funktionen, die aufgerufen werden, wenn ein Button gedrückt wird und ist auch für das Registrieren der Funktionen als Ereignisbehandler zuständig.

Der DPM sendet, wie bereits in Kapitel 4.2.3 beschrieben, in regelmäßigen Intervallen Änderungsabfragen an die Bedieneinheit. Als Antwort auf diese Abfrage wird der Tastenstatus zurückgesendet, wenn zwischen der letzten und der aktuellen Abfrage eine Taste gedrückt wurde. Der Tastenstatus wird ermittelt, indem jeder Taste eine Bitmaske zugeordnet ist und diese für das Antwortpaket verknüpft werden (siehe Listing 4.6).

```
1  define(['jquery', 'values', 'param'], function($, values, param) {
2
3          ...
4
5          var buttons = values.KB_SS1_1;
6
7          function cButtonPressed() {
8                  buttons |= values.KB_QUIT;
9                  handleButton(buttons);
10         }
11
12         function cButtonReleased() {
13                 buttons &= 0xFFFFFFFF-values.KB_QUIT;
14                 handleButton(buttons);
15         }
16
17         ...
18
19         $('#btnClear')
20                 .bind('touchstart', cButtonPressed)
21                 .bind('touchend', cButtonReleased);
22
23         ...
24
25 }
```

Listing 4.6: Auschnitt aus dem *keyHandler.js*-Modul. Die Funktionen *cButtonPressed* und *cButtonReleased* werden als Ereignisbehandler für die Ereignisse *touchstart* und *touchend* definiert.

page.js

Die Anzeige des Displays wurde mithilfe von zwei Canvas-Elementen realisiert, die vom *page.js*-Modul gesteuert werden. Auf das erste Element werden alle permanent anzuzeigenden Inhalte gezeichnet und auf das zweite nur jene, die blinkend dargestellt werden sollen. Die Elemente werden in einem Intervall von einer Sekunde umgeschaltet, um somit das Blinken zu realisieren.

Das *page.js*-Modul beinhaltet alle Funktionen, um auf die Canvas-Elemente zu zeichnen und aktuelle Anzeigeinformationen auszulesen (siehe Listing 4.7). Die Funktionen erfüllen dabei unter anderem folgende Aufgaben:

- Löschen der gesamten Anzeige
- Löschen von Teilbereichen
- Aufschalten eines Hintergrundbildes entsprechend der anzuzeigenden Seite
- Aufschalten von dynamischen Objekten
- Aufschalten von Bildern, die dynamischen Objekten zugeordnet sind
- Schreiben von Texten
- Scrollen von Bildschirminhalten

```
1  function getColorAt(x, y) {
2          if(backgroundData) {
3                  var imageData = bgCtx.getImageData(x, y, 1, 1);
4                  return new Color(
5                          imageData.data[0],
6                          imageData.data[1],
7                          imageData.data[2]);
8          } else {
9                  return new Color(255, 255, 255);
10         }
11 }
```

Listing 4.7: Die Funktion *getColorAt* liest aus dem aktuell angezeigten Hintergrundbild die Pixelinformation an den übergebenen *x*- und *y*-Koordinaten aus. Anschließend wird ein *Color*-Objekt mit den ausgelesenen Informationen zurückgegeben.

KAPITEL 5

Evaluierung

5.1 Übersicht

Dieses Kapitel ist der Evaluation der in Kapitel 4 entwickelten Web- bzw. Apache Cordova-Applikation gewidmet. Es soll anhand der entwickelten Applikation gezeigt werden, dass mobile Plattformen als Visualisierungslösung in der Automatisierungstechnik eingesetzt werden können und auch bereits bestehende Systeme vom Einsatz von mobilen Plattformen profitieren können. Dazu wird in Unterkapitel 5.2 eine durchgeführte Fallstudie beschrieben, in der das bestehende Visualsierungssystem der Sprecher Automation GmbH, durch die entwickelte mobile Plattform, ersetzt wird.

5.2 Fallstudie

Das Ziel dieser Evaluation ist es, die Funktionalität der entwickelten Applikation zu demonstrieren, um somit zu zeigen, dass mobile Plattformen für die Visualisierung in der Automatisierungstechnik eingesetzt werden können. Um das zu erreichen, wird eine Fallstudie durchgeführt. Dafür wird zuerst der Ablauf der Fallstudie gezeigt, der auf den in Kapitel 4.2.1 gezeigten Anwendungsfällen basiert. Anschließend wird der für die Fallstudie benötigte Aufbau beschrieben. Dieser beinhaltet alle Hardware- und Software-Komponenten, die für einen reibungslosen Ablauf benötigt werden. Danach werden die Anwendungsfälle anhand eines Testplans evaluiert, der erfolgreich durchgeführt werden muss, damit die entwickelte Applikation das bestehende System ersetzen kann.

5.2.1 Ablauf

Die Fallstudie basiert, wie bereits erwähnt, auf den in Kapitel 4.2.1 gezeigten Anwendungsfällen, die funktionale Anforderungen an das System abdecken. Somit muss im ersten Schritt die Inbetriebnahme inklusive Initialisierungsphase durchgeführt werden, da eine erfolgreich abgeschlossene Initialisierung die Vorbedingung für alle Anwendungsfälle ist. Dazu zählt vor

allem die Parameterübertragung, die aufgrund der Unterschiede für monochrome und farbige Bedieneinheit durchgeführt werden muss. Sobald die Parameterübertragung erfolgreich abgeschlossen ist, kann mit der Ausführung der einzelnen Anwendungsfälle begonnen werden. Jeder Anwendungsfall muss zuerst mit dem bestehenden System durchgeführt werden, um Referenzdaten für die Implementierung zu aggregieren.

Neben den funktionalen Anforderungen werden auch die nicht-funktionale Anforderungen Leistung und Zuverlässigkeit getestet. Bezüglich der Leistung soll die Ausführungsgeschwindigkeit des portierten Systems in Bezug auf das bestehenden Systems evaluiert werden. In Hinblick auf die Zuverlässigkeit soll das Verhalten der Applikation bei Ausfall der Verbindung zur Automatisierungseinheit getestet werden.

5.2.2 Komponenten

Um die Fallstudie durchführen zu können, werden mehrere Komponenten benötigt, deren Zusammenspiel und Konfiguration im Folgenden beschrieben wird. Die Abbildungen 5.1, 5.2 und 5.4 zeigen den verwendeten Testaufbau.

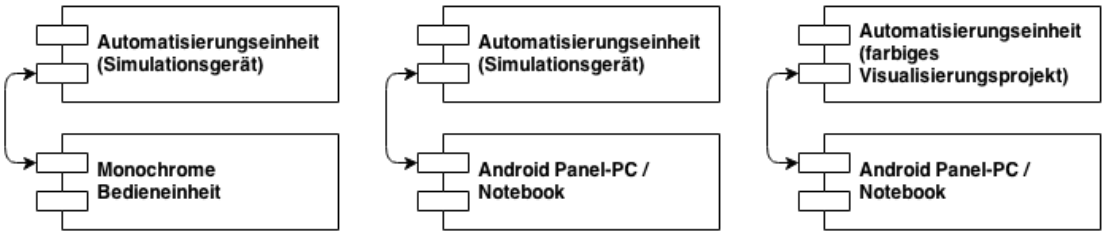

Abbildung 5.1: Testaufbau bestehend aus Simulationsgerät, Automatisierungseinheit mit farbigen Visualisierungsprojekt, monochromer Bedieneinheit, Android Panel-PC und Notebook.

Automatisierungseinheit

Auf der Automatisierungseinheit muss ein geeignetes Visualisierungsprojekt ausgeführt werden, um die Funktionalität der entwickelten Applikation und die der bestehenden Bedieneinheit zu vergleichen. Sprecher Automation GmbH besitzt Simulationsgeräte, die für das Testen und Simulieren von kompletten Automatisierungsanlagen eingesetzt werden. Diese Simulationsgeräte basieren auf der gleichen Automatisierungshardware, die auch für das Realisieren von Automatisierungsprojekten im industriellen Umfeld herangezogen wird. Die Simulationsgeräte verwenden eine für diesen Zweck erstellte Parametrierung, die ein sehr umfangreiches Visualisierungsprojekt beinhaltet. Dieses Visualisierungsprojekt ist aufgrund seiner Größe und Komplexität bestens für die Fallstudie geeignet. Es besteht aus der maximal konfigurierbaren Anzahl an Visualisierungsseiten, wobei jede der Seiten eine Vielzahl von Schalt- bzw. Binärobjekten (siehe Kapitel 4.2.2) und einige statische Elemente beinhaltet. Mit allen Schaltobjekten

können Schaltvorgänge vorgenommen werden. Die dabei entstehenden Änderungen werden auf der Bedieneinheit visualisiert.

Abbildung 5.2: Testaufbau bestehend aus Simulationsgerät mit angeschlossener monochromer Bedieneinheit.

Das Visualisierungsprojekt der Simulationsgeräte ist für monochrome Bedieneinheiten ausgelegt. Deshalb wird parallel ein einfaches Visualisierungsprojekt für farbige Bedieneinheiten erstellt, das auf eine zweite Automatisierungseinheit geladen wird. Mit diesem Visualisierungsprojekt soll der Unterschied in der Parameterübertragung zwischen monochromen und farbigen Displays abgedeckt werden. Abbildung 5.3 zeigt das verwendete Display Editor-Projekt.

Monochrome Bedieneinheit

Die monochrome Bedieneinheit wird an das Simulationsgerät angeschlossen. Nach erfolgreicher Parameterübertragung werden die Anwendungsfälle darauf ausgeführt. Die dabei angezeigten Daten dienen als Referenz für die Implementierung.

Android Panel-PC

Die erstellte Applikation wird auf einem Android Panel-PC installiert und gestartet. Danach wird der Panel-PC mit dem Simulationsgerät verbunden, um somit eine Parameterübertragung einzuleiten. Nach erfolgreicher Parameterübertragung werden dieselben Anwendungsfälle ausgeführt, die zuvor mit der monochromen Bedieneinheit durchgeführt wurden. Um die Parameterübertragung einer farbigen Bedieneinheit zu testen, wird der Panel-PC anschließend mit der zweiten Automatisierungseinheit verbunden, auf der das zuvor erstellte Visualisierungsprojekt geladen ist.

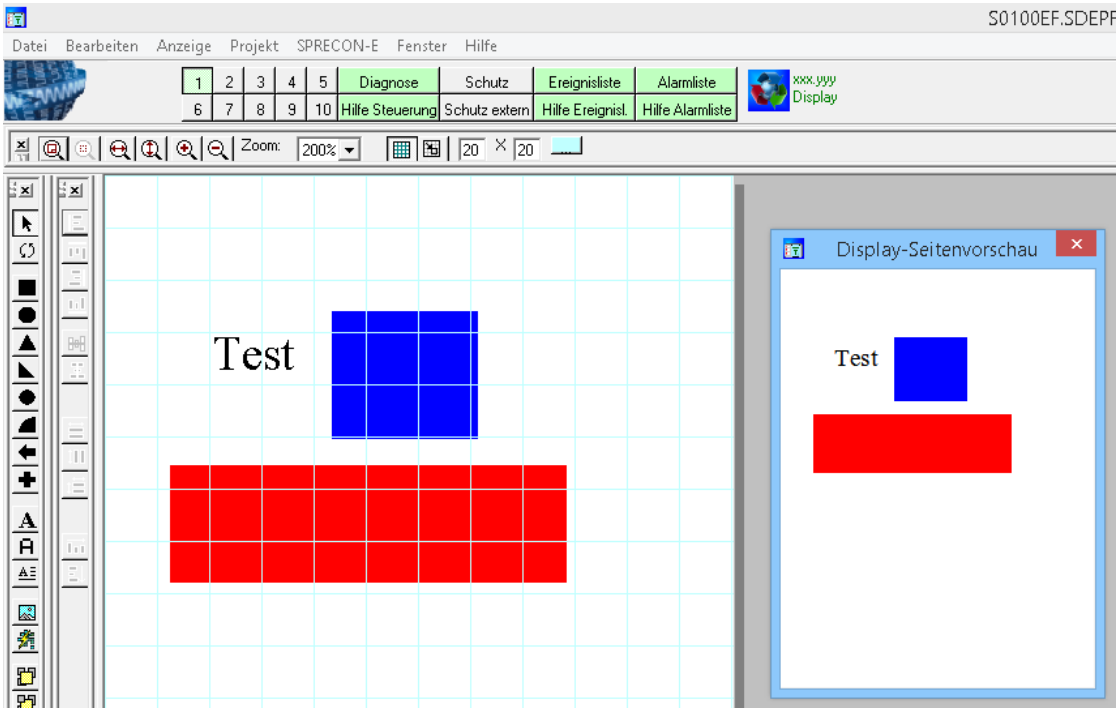

Abbildung 5.3: Display Editor-Projekt, das für das Testen der Parameterübertragung bei farbigen Bedieneinheiten benutzt wird.

Notebook

Auf dem Notebook muss der Kommunikationsserver als eigenständige Applikation gestartet werden. Danach wird die erstellte Applikation im Webbrowser ausgeführt. Anschließend wird das Notebook, wie der Android Panel-PC zuvor, der Reihe nach mit dem Simulationsgerät und der zweiten Automatisierungseinheit verbunden.

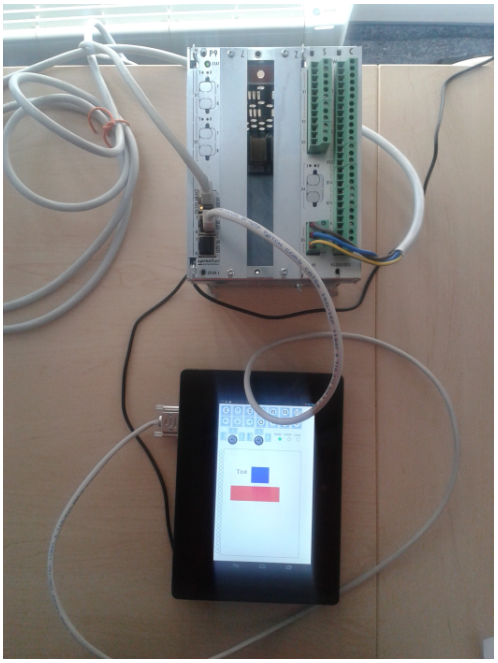

Abbildung 5.4: Testaufbau bestehend aus zweiter Automatisierungseinheit mit angeschlossenem Android Panel-PC.

5.2.3 Evaluierung

Die Evaluierung basiert, wie bereits erwähnt, auf einem zuvor erstellten Testplan, dessen Testfälle manuell durchgeführt werden müssen. Tabelle 5.1 zeigt die auszuführenden Testfälle im Detail.

Name	Beteiligte Komponenten	Beschreibung
Inbetriebnahme & Parameterübertragung - Monochrom - 1/2	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Erkennen des Nichtvorhandenseins korrekter Parameter, um somit eine Parameterübertragung einzuleiten. Messung der Dauer der Parameterübertragung, um die Ausführungsgeschwindigkeit vergleichen zu können.
Inbetriebnahme & Parameterübertragung - Monochrom - 2/2	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Nach erfolgreicher Parameterübertragung sind die Parameter auf der Bedieneinheit gespeichert. In diesem Testfall soll ermittelt werden, ob das Vorhandensein von korrekten Parametern erkannt wird und die Initialisierungsphase mit dem Aufschalten der ersten Visualisierungsseite abgeschlossen wird.

Inbetriebnahme & Parameterübertragung - Farbig - 1/2	Automatisierungseinheit mit farbigen Visualisierungsprojekt, Android Panel-PC, Notebook	Es wird ermittelt, ob die Parameterübertragung auch für farbige Bedieneinheiten funktioniert. Bei diesem Testfall wird ermittelt, ob in der Initialisierungsphase das Nichtvorhandensein korrekter Parameter erkannt wird und somit eine Parameterübertragung eingeleitet wird.
Inbetriebnahme & Parameterübertragung - Farbig - 2/2	Automatisierungseinheit mit farbigen Visualisierungsprojekt, Android Panel-PC, Notebook	In diesem Testfall wird ermittelt, ob das Vorhandensein korrekter Parameter erkannt wird und die Initialisierungsphase mit dem Aufschalten der ersten Visualisierunsseite abgeschlossen wird.
Prozessbilder anzeigen	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der ersten Visualisierungsseite. Es wird mit den Navigationstasten zwischen den Visualisierungsseiten navigiert, bis wieder die Ausgangsseite angezeigt wird. Die dabei angezeigten Daten werden miteinander verglichen. Die Dauer zwischen dem letzten Druck der Navigationstaste und der Anzeige der Ausgangsseite wird gemessen, um die Ausführungsgeschwindigkeit zu vergleichen.
Schaltvorgang durchführen	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der ersten Visualisierungsseite. Es wird mit den Navigationstasten das erste Schaltgerät ausgewählt und ein Schaltvorgang durchgeführt. Die dabei angezeigten Daten werden miteinander verglichen.
Parametereinstellung ändern	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der dritten Visualisierungsseite. Es wird mit den Navigationstasten das erste Objekt ausgewählt, das eine Parameteränderung erlaubt. Mithilfe der Bedientasten wird der vorhandene Wert geändert. Die dabei angezeigten Daten werden miteinander verglichen.
Ereignisliste anzeigen	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der ersten Visualisierungsseite. Durch das Betätigen der Menü-Navigationstaste wird die Ereignisliste angezeigt. Die dabei angezeigten Daten werden miteinander verglichen.

Ereignisse löschen	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der Ereignisliste. Die aktuell angezeigten Ereignisse werden mithilfe der Bedientasten gelöscht. Die dabei angezeigten Daten werden miteinander verglichen.
Alarmliste anzeigen	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der Ereignisliste. Durch das Betätigen der Menü-Navigationstaste wird die Alarmliste angezeigt. Die dabei angezeigten Daten werden miteinander verglichen.
Alarmmeldungen quittieren	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der Alarmliste. Die aktuell nicht quittierten Alarmmeldungen werden mithilfe der Bedientasten quittiert. Die dabei angezeigten Daten werden miteinander verglichen.
Alarmmeldungen löschen	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der Alarmliste. Die aktuell quittierten Alarmmeldungen werden mithilfe der Bedientasten gelöscht. Die dabei angezeigten Daten werden miteinander verglichen.
Diagnoseseite anzeigen	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der ersten Visualisierungsseite. Durch das gleichzeitige Betätigen der Menü- und Rechts-Navigationstaste wird die Diagnoseseite angezeigt. Die dabei angezeigten Daten werden miteinander verglichen.
Verbindungsabbruch	Simulationsgerät, monochrome Bedieneinheit, Android Panel-PC, Notebook	Dieser Testfall startet nach dem erfolgreichen Aufschalten der ersten Visualisierungsseite. Die Verbindung zwischen Simulationsgerät und Bedieneinheit wird getrennt. Die Applikation muss den Verbindungsabbruch erkennen.

Tabelle 5.1: Durchzuführende Testfälle.

Ergebnisse

Alle im Zuge der Evaluierung durchgeführten Aktionen lieferten auf der monochromen Bedieneinheit, dem Android Panel-PC und auf dem Notebook dasselbe Resultat. Auch die Parameterübertragung des farbigen Visualisierungsprojekts auf Panel-PC bzw. Notebook funktionierte fehlerfrei und die zuvor erstellte farbige Visualisierungsseite wurde angezeigt. Die Abbildungen 5.5 und 5.6 zeigen Ausschnitte aus dem Evaluierungsvorgang. Die Zeitmessungen haben

gezeigt, dass die Ausführungsgeschwindigkeit des portierten Systems mit jener des bestehenden Systems vergleichbar ist. Auch das Trennen der Verbindung zwischen Automatisierungseinheit und Android Panel-PC bzw. Notebook wurde korrekt erkannt.

Somit kann abschließend das Fazit gezogen werden, dass die Implementierung der Applikation erfolgreich durchgeführt wurde. Desweiteren wurde anhand der Entwicklung einer Webapplikation gezeigt, dass plattformunabhängige Visualisierungslösungen für die Automatisierungstechnik prinzipiell realisiert werden können.

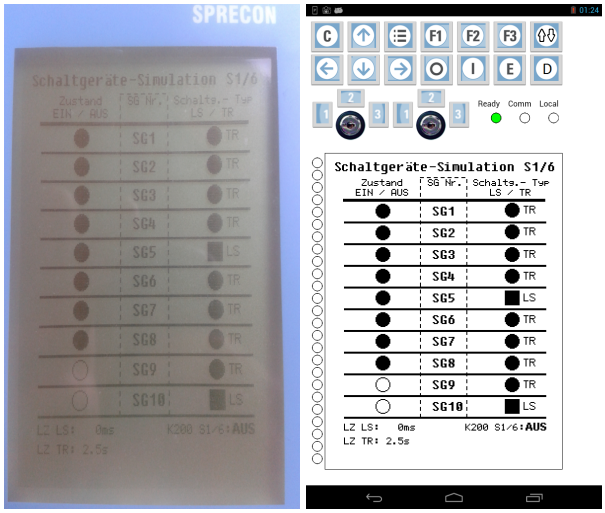

Abbildung 5.5: Bedieneinheit und Android Applikation vor dem Schaltvorgang.

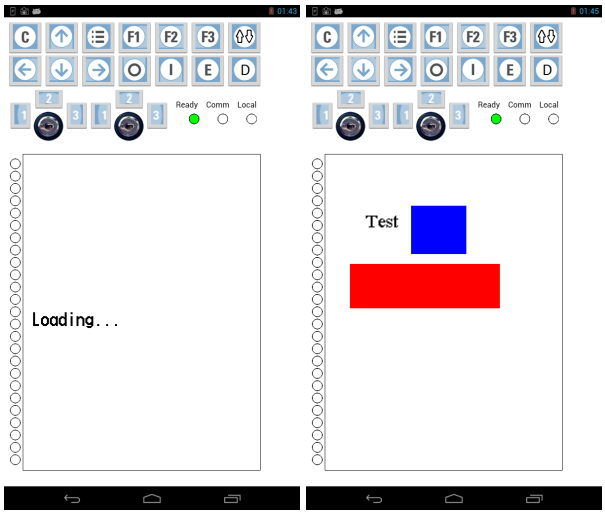

Abbildung 5.6: Android Applikation während der Parameterübertragung und beim Anzeigen der farbigen Startseite.

KAPITEL 6

Fazit

6.1 Zusammenfassung

In dieser Arbeit wurden Entwicklungsansätze für die Entwicklung von mobilen Applikation im Kontext der Visualisierung in der Automatisierungstechnik analysiert. Die Ergebnisse dieser Analyse wurden verwendet, um ein bestehendes Visualisierungssystem für Stationsleittechnik, Fernwirktechnik und Schutztechnik auf mobile Plattformen zu portieren. In den folgenden Absätzen werden die Ergebnisse der Analyse und der Implementierung zusammengefasst.

Zuerst wurden native Entwicklungsansätze für unterschiedliche Plattformen vorgestellt. Native Applikationen bieten auf allen Plattformen die beste Benutzerfreundlichkeit, da sie sich am besten in das Gesamtkonzept der jeweiligen Plattform einfügen und den Benutzern somit die Bedienung der Applikation erleichtern. Zusätzlich besitzen native Applikation die höchste Start- und Ausführungsgeschwindigkeit und bieten Zugriff auf alle nativen Funktionen, die auf der jeweiligen Plattform unterstützt werden, wozu auch unterschiedliche Arten der Kommunikation zählen. Auch für das Entwicklerteam bieten native Entwicklungsansätze eine Vielzahl von Erleichterungen. So sind für jede Plattform eigene IDEs verfügbar, die mit sehr mächtigen Debuggern und Emulatoren ausgestattet sind. Dadurch und durch die Verfügbarkeit von WYSIWYG-Editoren, guten Community-Support und ausgezeichneter Dokumentation eignen sich native Entwicklungsansätze am besten für die Entwicklung von Visualisierungsapplikationen. Sie haben jedoch den großen Nachteil, dass für jede Zielplattform eigens entwickelt werden muss und somit der Gesamtaufwand für die Unterstützung von mehreren Plattformen stark ansteigt.

Als nächstes wurde der Ansatz der Entwicklung von Webapplikationen behandelt. Webapplikationen basieren auf den Webtechnologien HTML, CSS und JavaScript und können auf allen Plattformen ausgeführt werden, auf denen ein Webbrowser zur Verfügung steht. Sie können die native Optik und Haptik nur mithilfe von Frameworks simulieren und bieten somit nicht die Benutzerfreundlichkeit einer nativen Applikation. Auch bei der Geschwindigkeit reichen sie nicht ganz an native Applikationen heran, diese kann jedoch aufgrund von zahlreichen Optimierungen innerhalb moderner Webbrowser als sehr gut bezeichnet werden. Der Zugriff auf

native Funktionen ist mit Webapplikationen nur sehr bedingt (z.B. Positionsbestimmung) oder gar nicht möglich. Desweiteren sind sie auf jene Kommunikationsprotokolle eingeschränkt, die vom ausführenden Webbrowser unterstützt werden (z.B. WebSockets). Für das Entwicklerteam stehen sehr gute IDEs zur Verfügung und auch die Dokumentation der Webtechnologien und der Community-Support kann als sehr gut bezeichnet werden. Zusammengefasst ist die Entwicklung einer Webapplikation eine sehr gute Wahl, wenn nicht auf native Gerätefunktionen zurückgegriffen werden muss und das Bedienkonzept der Applikationen nicht dem Plattformstandard entsprechen muss.

Als drittes wurden hybride Entwicklungsansätze vorgestellt. Hybride Entwicklungsansätze können in zwei Kategorien eingeteilt werden. Die erste Kategorie besteht aus Webapplikationen, die in eine native Hülle eingebettet werden. Die native Hülle bietet dabei einen API Zugriff auf native Gerätefunktionen. Ansonsten gelten für diese Art der hybriden Applikationen dieselben Vor- bzw. Nachteile wie für Webapplikationen. Die zweite Kategorie von hybriden Applikationen basiert auf eigenständigen Laufzeitumgebungen, die native Benutzeroberflächenelemente mit plattformunabhängigen Code kombinieren, für dessen Ausführung die Laufzeitumgebungen zuständig ist. Diese Art von hybriden Applikationen bieten eine native Optik und Haptik, die den Benutzern zugute kommt. Für das Entwicklerteam ist diese Kategorie von hybriden Applikationen meist mit großem Aufwand verbunden. Es stehen zwar gute IDEs zur Verfügung und auch die Dokumentation kann durchwegs als gut bezeichnet werden, doch durch zahlreiche, sehr Framework-spezifische Aspekte wird der Aufwand der Entwicklung erhöht. Wenn jedoch innerhalb des Entwicklerteams bereits Erfahrung mit einem solchen Ansatz gesammelt wurde, können (mit moderatem Zeitaufwand) plattformunabhängige Applikationen mit nativem Verhalten entwickelt werden.

Abschließend wurde die modellgetriebene Entwicklung vorgestellt und analysiert. Meta-Modelle sind das Grundgerüst für die Definition von Modellierungssprachen, die für das Erstellen von Modellen verwendet werden. Modelle sind die Basis dieser Art der Entwicklung und bilden eine bestimmte Domäne ab. Das Ziel dieses Ansatzes ist es, aus plattformunabhängigen Modellen mittels Transformationen plattformspezifischen Code zu erstellen. Die modellgetriebene Entwicklung wäre somit hervorragend für die Entwicklung von mobilen Applikationen geeignet. Es existiert jedoch aktuell kein produktiv einsetzbarer Ansatz, weshalb dieses Thema in Unterkapitel 6.3 noch einmal behandelt wird.

Für die Implementierung im Zuge dieser Arbeit wurde der Ansatz einer Webapplikation, die für Android jedoch als Hybridapplikation eingesetzt wurde, gewählt und damit ein bestehendes Visualisierungssystem portiert. Der Ansatz wurde gewählt, weil nicht auf native Gerätefunktionen zurückgegriffen werden musste und eine Vielzahl von Plattformen (nicht nur mobile) unterstützt werden sollte. Anhand der erfolgreichen Portierung wurde gezeigt, dass mobile Plattformen als Visualisierungslösung in der Automatisierungstechnik prinzipiell eingesetzt werden können.

6.2 Verwandte Arbeiten

In diesem Kapitel werden bereits existierende Arbeiten vorgestellt, deren Themen im Zusammenhang mit dieser Arbeit stehen.

Wie bereits erwähnt, existiert keine ausgereifte Toolkette für die modellgetriebene mobile Applikationsentwicklung. Heitkötter et al. haben in den Arbeiten [32, 33] das von ihnen entwickelte Framework md^2 vorgestellt, auf dessen Basis sie die plattformunabhängige Entwicklung von kommerziellen mobilen Applikationen ermöglichen wollen. Mithilfe des Frameworks sollen Modelle definiert werden können, aus denen per Transformation nativer Code für die mobilen Plattformen Android und iOS generiert wird. Dazu haben die Autoren typische Anwendungsfälle für datengetriebene, kommerzielle Anwendungen analysiert und anhand dieser Analyse eine DSL entwickelt, die auf diese Art der Anwendungen zugeschnitten ist. Das MVC-Entwurfsmuster wurde als Basis für die Entwicklung dieser DSL und den Code-Transformatoren verwendet. Der Controller behandelt im generierten Code Benutzerinteraktionen und applikationsinterne bzw. gerätespezifische Ereignisse. Realisiert wurde das Framework unter Zuhilfenahme des EMF.

Einen ähnlichen Ansatz verfolgen Jia und Jones in [57]. Sie entwickelten ebenfalls eine eigene DSL, die auf der dynamischen Programmiersprache Groovy basiert. Die DSL bildet die Basis ihres AXIOM getauften modellgetriebenen Ansatzes. Die Autoren haben sich bei diesem Ansatz nicht auf ein spezielles Entwurfsmuster festgelegt und die Code-Transformation in zwei Phasen aufgeteilt. In der ersten Phase wird die strukturelle Transformation vorgenommen. Das bedeutet, dass in dieser Phase die getroffenen Entscheidungen bezüglich Plattform, Programmiersprache, Frameworks, Architektur und Entwurfsmustern umgesetzt werden. In der zweiten Phase werden anschließend feinere Details umgesetzt, wie z.B. die Wahl eines Themes oder Algorithmus.

In seiner Masterarbeit untersuchte Sommer plattformunabhängige Frameworks für die Entwicklung von mobilen Business-Applikationen [78]. Er erstellte verschieden gewichtete Kriterien, nach denen er die unterschiedlichen Ansätze anhand eines Punktesystems vergleicht. Er untersuchte neben Titanium und PhoneGap auch andere hybride Ansätze für die Entwicklung von plattforumunabhängigen Applikationen und kam zu dem Ergebnis, dass zwar native Ansätze die beste Benutzerfreundlichkeit bieten, hybride Ansätze diesem Vorteil jedoch die plattforumübergreifende Entwicklung entgegenstellen. Die Ergebnisse decken sich somit zu einem großen Teil mit jenen, die innerhalb dieser Arbeit erarbeitet wurden.

Eine Zusammenfassung über aktuelle Trends der Softwareentwicklung in der Prozessautomation wird von Vyatkin in [86] geboten. Dabei wird unter anderem darauf eingegangen, wie sich Normen, beispielsweise IEC 61850, auf die Softwareentwicklung auswirken und welche Möglichkeiten für das Testen, Warten und Weiterentwickeln von Software bestehen. Ein weiterer Teil der Arbeit behandelt das Thema Softwarequalität, worunter der Autor neben funktionaler Sicherheit auch formale Verifikation diskutiert. Diese Arbeit liefert einen sehr guten Überblick über alte und neue Entwicklungsansätze, die in der Automation eingesetzt werden.

6.3 Ausblick

Wie bereits erwähnt, wurde im Zuge dieser Arbeit festgestellt, dass aktuell kein produktiv einsetzbarer modellgetriebener Ansatz für die Entwicklung von mobilen Applikationen existiert. Diesem Umstand sollte gezielt entgegen gewirkt werden, da in modellgetriebenen Ansätzen sehr viel Potential steckt. Mit ihnen könnten Applikationen auf Basis eines plattformunabhängigen Konzeptes entwickelt werden, die alle Vorteile von nativen Applikationen vereinen.

Auch auf Seiten der Automationseinheiten besteht Handlungsbedarf. Getrieben durch Ansätze im IoT oder die Initiative İndustrie 4.0ßollten Automationsstationen mit mächtigeren Schnittstellen und Protokollen ausgestattet werden, die den Aufbau von Service-orientierenten Architekturen unterstützen.

Im Zusammenhang mit der Implementierung, die im Zuge dieser Arbeit durchgeführt wurde, könnten Erweiterungen Vorteile mit sich bringen. So könnte der DPM um das WebSocket Protokoll erweitert werden, um somit den entwickelten Kommunikationsserver überflüssig zu machen. Alternativ wäre auch eine direkte Integration der Implementierung (oder zumindest eine angepasste Form davon) in den bereits auf den Automatisierungseinheiten vorhandenen Webserver eine Bereicherung. In diesem Fall müsste jedoch den nicht-funktionalen Anforderungen bezüglich Sicherheit Aufmerksamkeit zuteil werden. Diese wurden bei der Implementierung bewusst nicht behandelt, weil keine Änderungen auf Seiten der Automatisierungsfirmware vorgenommen wurden und somit weiterhin nur über die serielle Schnittstelle mit der Automatisierungseinheit kommuniziert werden kann.

Abkürzungsverzeichnis

ADT	Android Developer Tools
AIR	Adobe Integrated Runtime
AJAX	Asynchronous JavaScript and XML
AOSP	Android Open Source Project
API	Application Programming Interface
ARC	Automatic Reference Counting
ART	Android RunTime
ASL2.0	Apache Software License 2.0
ATL	ATLAS Transformation Language
CDT	C/C++ Development Tools
CLI	Command-Line Interface
CPS	Cyber-Physical Systems
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DOM	Document Object Model
DPM	Display Panel Manager
DSL	Domain Specific Language
ECMA	European Computer Manufacturers Association
EMF	Eclipse Modeling Framework
ERP	Enterprise-Resource-Planning

GPL	General Purpose Language
GPS	Global Positioning System
HMI	Human–Machine Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IoT	Internet of Things
IPC	Inter-process communication
JMI	Java Metadata Interface
JSDT	JavaScript Development Tools
JSON	JavaScript Object Notation
jSSC	java-simple-serial-connector
MDE	Model Driven Engineering
MES	Manufacturing Execution System
MVC	Model-View-Controller
NDK	Native Development Kit
OCL	Object Constraint Language
OHA	Open Handset Alliance
OSS	Open-Source Software
QVT	Query/View/Transformation
RGBA	Red Green Blue Alpha
RTOS	Real-time Operating System
SCADA	supervisory control and data acquisition
SDK	Software Development Kit
TCP	Transmission Control Protocol
TSG	Technical Steering Group
UML	Unified Modeling Language
URL	Uniform Resource Locator

VIATRA VIsual Automated model TRAnsformations

W3C World Wide Web Consortium

WHATWG Web Hypertext Application Technology Working Group

WLAN Wireless Local Area Network

WWW World Wide Web

WYSIWYG What You See Is What You Get

XML Extensible Markup Language

Abbildungsverzeichnis

Listingverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] AdDuplex. http://blog.adduplex.com/2014/02/adduplex-windows-phone-statistics.html. Accessed: 2014-04-21.

[2] Sarah Allen, Vidal Graupera und Lee Lundrigan. *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution*. Apress, 2010.

[3] Thomas Bauernhansl, Michael ten Hompel und Birgit Vogel-Heuser. *Industrie 4.0 in Produktion, Automatisierung und Logistik*. Springer, 2014.

[4] Gerhard Baum, Holger Borcherding, Manfred Broy, Martin Eigner, Anton S. Huber, Kohler Herbert K., Siegfried Russwurm und Matthias Stümpfle. *Industrie 4.0: Beherrschung der industriellen Komplexität mit SysLM*. Springer, 2013.

[5] Ltd BlackBerry. http://developer.blackberry.com/air/documentation/overview_ms_2010836_11.html. Accessed: 2014-06-25.

[6] Marco Brambilla, Jordi Cabot und Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.

[7] James Burke. http://requirejs.org/. Accessed: 2014-08-24.

[8] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1996.

[9] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4:171–188, 5 2005.

[10] Guiran Chang, Chunguang Tan, Guanhua Li und Chuan Zhu. Developing Mobile Applications on the Android Platform. In Xiaoyi Jiang, Matthew Y. Ma und Chang Wen Chen, editors, *Mobile Multimedia Processing. Fundamentals, Methods, and Applications*, pages 264–286. Springer Berlin Heidelberg, 2010.

[11] IEI Integration Corp. http://www.ieimobile.com/index.php?option=com_flexicontent&view=items&cid=47:iovu-risc-based-pc&id=244. Accessed: 2014-08-24.

[12] Luis Corral, Andrea Janes und Tadas Remencius. Potential advantages and disadvantages of multiplatform development frameworks-a vision on mobile environments. *Procedia CS*, 10:1202–1207, 2012.

[13] Luis Corral, Alberto Sillitti und Giancarlo Succi. Mobile Multiplatform Development: An Experiment for Performance Analysis. *Procedia CS*, 10:736–743, 2012.

[14] Krzysztof Czarnecki und Simon Helsen. Classification of model transformation approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, pages 1 – 17, 2003.

[15] Davjoh. `http://commons.wikimedia.org/wiki/file:modelviewcontrollerdiagram2.svg`. Accessed: 2014-08-22.

[16] Deutsche Forschungszentrum für Künstliche Intelligenz GmbH, 2011.

[17] Alexis Deveria. `http://caniuse.com/canvas`. Accessed: 2014-07-18.

[18] Alexis Deveria. `http://caniuse.com/#cats=html5`. Accessed: 2014-06-28.

[19] Alexis Deveria. `http://caniuse.com/websockets`. Accessed: 2014-07-18.

[20] Tantek Çelik, Elika J. Etemad, Daniel Glazman, Ian Hickson, Peter Linss und John Williams. Selectors level 3. Technical report, World Wide Web Consortium, 2011.

[21] I. Fette, Google Inc., A. Melnikov und Ltd. Isode. The WebSocket Protocol. RFC 6455, Internet Engineering Task Force, 2011.

[22] David Flanagan. *JavaScript: The Definitive Guide (6th ed.)*. O'Reilly Media, Inc., 2011.

[23] Klaus Förster und Bernd Öggl. *HTML5 - Leitfaden für Webentwickler*. Addison-Wesley, 2011.

[24] The Apache Software Foundation. `http://cordova.apache.org/docs/en/3.5.0/guide_overview_index.md.html#overview`. Accessed: 2014-07-24.

[25] Dragan Gaševic, Dragan Djuric und Vladan Devedžic. *Model Driven Engineering and Ontology Development*. Springer-Verlag Berlin Heidelberg, 2 edition, 2009.

[26] Mark H. Goadrich und Michael P. Rogers. Smart Smartphone Development: iOS versus Android. In *Proceedings of the 42nd ACM technical symposium on Computer science education, März 09-12, Dallas, Texas, USA*, pages 607–612. ACM, Inc., 2011.

[27] Kate Gregory. Managed, Unmanaged, Native: What Kind of Code Is This?, `http://www.developer.com/net/cplus/article.php/2197621/managed-unmanaged-native-what-kind-of-code-is-this.htm`. Accessed: 2014-06-22.

[28] Joe Groff. https://twitter.com/jckarter/status/473539096065736705. Accessed: 2014-06-20.

[29] Gustavo Hartmann, Geoff Stead und Asi DeGani. Cross-platform mobile development, 2011.

[30] Henning Heitkötter, Sebastian Hanschke und Tim A. Majchrzak. Comparing Cross-platform Development Approaches for Mobile Applications. In *Web Information Systems and Technologies (8th International Conference, WEBIST 2012)*, pages 299–311. Springer-Verlag, 2012.

[31] Henning Heitkötter, Sebastian Hanschke und Tim A. Majchrzak. Evaluating Cross-Platform Development Approaches for Mobile Applications. In *Web Information Systems and Technologies (8th International Conference, WEBIST 2012)*, pages 120–138. Springer-Verlag, 2012.

[32] Henning Heitkötter, Tim A. Majchrzak und Herbert Kuchen. Cross-platform model-driven development of mobile applications with md^2. In *SAC*, pages 526–533, 2013.

[33] Henning Heitkötter, Tim A. Majchrzak und Herbert Kuchen. Md^2-dsl — eine domänenspezifische sprache zur beschreibung und generierung mobiler anwendungen. In *Software Engineering 2013 — Workshopband*, volume 215, pages 91–106. Gesellschaft für Informatik, 2013.

[34] Roland Hensel. http://www.vdi-nachrichten.com/technik-wirtschaft/industrie-40-konzepte-ruetteln-an-automatisierungspyramide. Accessed: 2014-09-15.

[35] Ian Hickson und Google Inc. The WebSocket API. Technical report, World Wide Web Consortium, 2012.

[36] HTC Corporation. http://www.htc.com/us/go/htc-software-updates/. Accessed: 2014-06-07.

[37] Appcelerator Inc. http://docs.appcelerator.com/titanium/latest/#!/guide/alloy_concepts. Accessed: 2014-07-28.

[38] Appcelerator Inc. http://www.appcelerator.com/plans-pricing/. Accessed: 2014-07-24.

[39] Appcelerator Inc. http://www.appcelerator.com/titanium/. Accessed: 2014-07-28.

[40] Apple Inc. https://developer.apple.com/library/prerelease/ios/documentation/swift/conceptual/swift_programming_language/index.html. Accessed: 2014-06-20.

[41] Apple Inc. https://developer.apple.com/support/appstore/. Accessed: 2014-06-20.

[42] Apple Inc. https://developer.apple.com/support/ios/ios-dev-center.php. Accessed: 2014-06-20.

[43] Apple Inc. iOS Technology Overview, https://developer.apple.com/library/ios/documentation/miscellaneous/conceptual/iphoneostechoverview/iostechoverview.pdf. Accessed: 2014-06-20.

[44] Gartner Inc. http://www.gartner.com/newsroom/id/2665715. Accessed: 2014-04-20.

[45] Google Inc. http://developer.android.com/sdk/index.html. Accessed: 2014-06-07.

[46] Google Inc. http://developer.android.com/tools/support-library/index.html. Accessed: 2014-06-07.

[47] Google Inc. https://developer.android.com/about/dashboards/index.html. Accessed: 2014-06-07.

[48] Google Inc. https://developer.android.com/sdk/installing/studio.html. Accessed: 2014-06-07.

[49] Google Inc. https://developer.android.com/tools/index.html. Accessed: 2014-06-07.

[50] Google Inc. https://developer.android.com/tools/sdk/ndk/index.html. Accessed: 2014-06-07.

[51] Google Inc. http://source.android.com/source/licenses.html. Accessed: 2014-06-07.

[52] Google Inc. https://sites.google.com/site/io/anatomy--physiology-of-an-android. Accessed: 2014-06-07.

[53] Google Inc. https://source.android.com/devices/tech/dalvik/art.html. Accessed: 2014-06-07.

[54] O'Reilly Media Inc. http://chimera.labs.oreilly.com/books/1234000001853/ch01.html. Accessed: 2014-06-22.

[55] SimilarTech Inc. https://www.similartech.com/categories/javascript. Accessed: 2014-07-22.

[56] HoJun Jaygarl, Cheng Luo, YoonSoo Kim, Eunyoung Choi, Kevin Bradwick und Jon Lansdell. *Professional Tizen Application Development*. John Wiley & Sons, Inc., 2014.

[57] Xiaoping Jia und Chris Jones. Cross-platform application development using axiom as an agile model-driven approach. In *Software and Data Technologies*, volume 411, pages 36–51. Springer Berlin Heidelberg, 2013.

[58] Mona Erfani Joorabchi, Ali Mesbah und Philippe Kruchten. Real Challenges in Mobile App Development. In *ESEM*, pages 15–24. IEEE, 2013.

[59] Frédéric Jouaulta, Freddy Allilairea, Jean Bézivina und Ivan Kurtevb. ATL: A model transformation tool. *Science of Computer Programming*, 72(1–2):31 – 39, 2008.

[60] Henning Kagermann, Wolfgang Wahlster und Johannes Helbig, editors. *Deutschlands Zukunft als Produktionsstandort sichern: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0, Abschlussbericht des Arbeitskreises Industrie 4.0*. Forschungsunion im Stifterverband für die Deutsche Wirtschaft e.V., 2013.

[61] Kantar Group. `http://www.kantarworldpanel.com/global/news/apple-regains-momentum-as-windows-stutters`. Accessed: 2014-04-21.

[62] Stefan Klement. Sicherheitsaspekte der Google Android Plattform. Master's thesis, Universität Bremen, 2011.

[63] G. E. Krasner und S. T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.

[64] Avraham Leff und James T. Rayfield. Web-application development using the model/view-/controller design pattern. In *EDOC*, pages 118–127. IEEE Computer Society, 2001.

[65] BlackBerry Ltd. `http://developer.blackberry.com/html5/documentation/v2_1/what_is_a_webworks_app.html`. Accessed: 2014-06-24.

[66] BlackBerry Ltd. `http://developer.blackberry.com/native/documentation/cascades/getting_started/intro/index.html`. Accessed: 2014-06-24.

[67] BlackBerry Ltd. `http://developer.blackberry.com/native/documentation/core/opengl_es_developer_guide.html`. Accessed: 2014-06-24.

[68] Microsoft Corporation. `http://msdn.microsoft.com/en-us/library/windowsphone/develop/br229577.aspx`. Accessed: 2014-06-21.

[69] Microsoft Corporation. `http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402563.aspx`. Accessed: 2014-06-22.

[70] Microsoft Corporation. `http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh699871.aspx`. Accessed: 2014-06-21.

[71] Microsoft Corporation. http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj129478.aspx. Accessed: 2014-06-22.

[72] Charlie Miller, Dion Blazakis, Dino DaiZovi, Stefan Esser, Vincenzo Iozzo und Ralf-Philip Weinmann. *iOS Hacker's Handbook*. John Wiley & Sons, Inc., 2012.

[73] Refsnes Data. http://www.w3schools.com/html/html5_new_elements.asp. Accessed: 2014-07-11.

[74] Refsnes Data. http://www.w3schools.com/tags/canvas_getimagedata.asp. Accessed: 2014-07-18.

[75] Refsnes Data. http://www.w3schools.com/tags/ref_canvas.asp. Accessed: 2014-07-18.

[76] Thilo Sauter, Stefan Soucek, Wolfgang Kastner und Dietmar Dietrich. The Evolution of Factory and Building Automation. *Industrial Electronics Magazine, IEEE*, 5(3):35–48, 2011.

[77] Shane Sendall und Wojtek Kozaczynski. Model transformation: the heart and soul of model-driven software development. *IEEE Software*, 20(5):42 – 45, 2003.

[78] Andreas Sommer. Comparison and evaluation of cross-platform frameworks for the development of mobile business applications. Master's thesis, Technische Universität Müunchen, 2012.

[79] Ralph Steyer. *jQuery: Das universelle JavaScript-Framework für das interaktive Web und mobile Anwendungen*. Carl Hanser Verlag GmbH & Co. KG, 2014.

[80] Tizen Project. https://developer.tizen.org/dev-guide/2.2.0/org.tizen.gettingstarted/html/tizen_overview/tizen_architecture.htm. Accessed: 2014-06-27.

[81] Tizen Project. https://developer.tizen.org/dev-guide/2.2.1/org.tizen.native.appprogramming/html/cover_page.htm. Accessed: 2014-06-27.

[82] Tizen Project. https://developer.tizen.org/dev-guide/2.2.1/org.tizen.web.appprogramming/html/cover_page.htm. Accessed: 2014-06-27.

[83] Tizen Project. https://www.tizen.org/about. Accessed: 2014-06-27.

[84] UlrichAAB. http://de.wikipedia.org/wiki/datei:automatisierungspyramide2.svg. Accessed: 2014-09-15.

[85] Unbekannt. http://stuk.github.io/jszip/. Accessed: 2014-08-24.

[86] Valeriy Vyatkin. Software Engineering in Industrial Automation: State-of-the-Art Review. *IEEE Transactions on Industrial Informatics*, pages 1234–1249, 8 2013.

[87] World Wide Web Consortium. `http://dev.w3.org/html5/decision-policy/html5-2014-plan.html#plan`. Accessed: 2014-06-28.

[88] World Wide Web Consortium. `http://www.w3.org/tr/html-markup/syntax.html`. Accessed: 2014-06-28.

[89] Spyros Xanthopoulos und Stelios Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, BCI '13, pages 213–220, 2013.