

-  Fachhochschule Köln
Cologne University of Applied Sciences

Diplomarbeit

vorgelegt an der Fachhochschule Köln
Campus Gummersbach
im Studiengang Allgemeine Informatik

Modellgetriebene Entwicklung von Smartphone-Applikationen zur Kommunikation mit Fertigungsmaschinen

erstellt von: Georg Wolf
Matrikelnummer 11053589
Erster Prüfer: Professor Friedbert Jochum
Zweiter Prüfer: Professor Heiner Klocke
Gummersbach, 11. April 2011

Inhaltsverzeichnis

1 Einleitung und Aufgabenstellung	12
1.1 Zielsetzung und Beschreibung der Aufgabenstellung	12
1.2 Das Unternehmen KHS	13
1.3 Grundlagen	14
1.3.1 Model Driven Architecture	14
1.3.2 Arbeitsweise	15
2 Applikationen für Smartphones	17
2.1 Handykultur im heutigen Zeitalter	17
2.2 Begriffsklärung Applikation	19
2.3 Sicherheit in mobilen Netzen	20
2.3.1 Sicherheitsprobleme in mobilen LANs und WANs	20
2.3.1.1 Bluetooth	20
2.3.1.2 Global System for Mobile Communications	21
2.3.1.3 General Packet Radio Service	21
2.3.1.4 Universal Mobile Telecommunications System	22
2.3.1.5 High Speed Downlink Packet Access	22
2.3.1.6 Long Term Evolution	22
2.3.1.7 Wireless LAN	22
2.3.1.7.1 Wired Equivalent Privacy	22
2.3.1.7.2 Wireless Protected Access	23
2.3.1.7.3 WPA2	24
2.3.2 Denkbare Angriffsformen auf Mobiltelefone in einer Firmeninfrastruktur	24
2.3.3 Sicherheitsmechanismen	25
2.3.3.1 WLAN sichern mit Radius	25
2.3.3.2 Mobile Device Management	25
2.3.4 Fallunterscheidung Übertragungstechnik	26
2.4 Gefahren beim Einsatz von Applikationen in der Industrie	27
3 Systemspezifikation	29
3.1 Funktionsumfang und Nutzen für KHS	29
3.1.1 Wirtschaftliche und marketingtechnische Erwägungen	29
3.1.2 Funktionsumfang	30
3.2 Geschäftsanwendungsfall für eine umzusetzende Applikation	32
3.2.1 Vorgaben	32
3.2.2 Beschreibung des Geschäftsanwendungsfalls	32
3.3 Anforderungsdefinition	33
3.3.1 Erstellen eines Computation Independent Model	34
3.3.2 Erstellen eines Platform Independent Model	36

4 Mobile Betriebssysteme und Entwicklungsvoraussetzungen	42
4.1 Mobile Betriebssysteme	42
4.1.1 iOS	43
4.1.1.1 Architektur	43
4.1.1.2 iOS-Komponenten	43
4.1.1.3 Sicherheit	44
4.1.1.4 Entwicklung	44
4.1.2 Android	46
4.1.2.1 Architektur	46
4.1.2.2 Android-Komponenten	47
4.1.2.3 Sicherheit	48
4.1.2.4 Entwicklung	48
4.1.3 Windows Phone 7	49
4.1.4 Symbian	49
4.1.5 BlackBerry OS	50
4.1.6 HP webOS	50
4.2 Plattformübergreifende Ansätze	51
4.2.1 WebApp	51
4.2.2 Cross-Platform-Development-Tools	51
5 Implementierung und Test	54
5.1 Implementierung	54
5.1.1 Vorhandene Tools auf dem Markt	54
5.1.1.1 UML-Tools	54
5.1.1.1.1 Visual Paradigm	55
5.1.1.1.2 MagicDraw	55
5.1.1.1.3 Schlussfolgerung	55
5.1.1.2 Generator-Tools	56
5.1.1.2.1 openArchitectureWare	56
5.1.1.2.2 Acceleo	56
5.1.1.2.3 Auswahl des Generatortools	56
5.1.2 Zielplattformen für den Prototypen	57
5.1.3 Vorbereitung der Entwicklung und Generierung von Testcode	58
5.1.4 Basisstruktur der Anwendung	58
5.1.5 Kommunikation mit der Fertigungsmaschine	60
5.1.6 Anpassung der UML-Diagramme an die zu entwerfende Architektur	60
5.1.7 Codegenerierung	61
5.1.8 Durchgeführte Implementationsarbeiten zur Erstellung des Prototypen	65
5.1.9 Hindernisse bei der Codegenerierung	67
5.2 Testing	68
5.2.1 Einstufung verschiedener Testverfahren	68
5.2.2 Durchführung des Testings	69
5.2.2.1 Test des Generators	69
5.2.2.1.1 Test der Generierung von Views	69
5.2.2.1.2 Testfallentwurf	69
5.2.2.1.3 Implementierung in JUnit	70
5.2.2.1.4 Testergebnisse	71
5.2.2.1.5 Ausblick	71

5.2.2.2	Test des generierten Codes	71
5.2.3	Modellbasiertes Testen	71
6	Zusammenfassung und Ausblick	73
	Literaturverzeichnis	77
	Quellenverzeichnis für Abbildungen	81
A	Anhang	82
A.1	Einrichtung der Entwicklungsumgebungen	82
A.1.1	Android	82
A.1.2	iOS	82
A.1.3	Acceleo	83
A.2	Jailbreak	83
A.3	Hackintosh	84
A.4	Model-View-Controller-Prinzip	85
A.5	Einrichtung Server als Debian Etch-VM	86
A.6	Quellcodebeispiel View-Generierung	86
A.7	JUnit Tests – Quellcode	88
B	beigelegte DVD	91

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit dem Thema „Model Driven Architecture“ (MDA). Es soll ein Konzept erstellt werden, welches es ermöglicht, mittels MDA Smartphone-Applikationen für verschiedene Endgeräte zu erzeugen. Grundlagen der technischen Gegebenheiten von auf dem Markt verfügbaren Smartphones und deren Betriebssystemen werden aufgezeigt. Anschließend wird die prototypenhafte Umsetzung eines Geschäftsanwendungsfalles beschrieben und durchgeführt. Zum Schluss werden die Ergebnisse der Arbeit aufgezeigt. Dabei werden Lösungsansätze erstellt und Probleme während der Entwicklung erläutert. Ebenfalls wird der Einsatz von MDA diskutiert.

Abstract

This paper you are about to read is focused on the topic of „Model Driven Architecture“ (MDA). A concept will be created, giving the possibility to deploy smartphone applications on different mobile operating systems using MDA. Fundamentals of available smartphones on the market and their technical details will be given. In addition the creation of a prototype implementation depending on a business use case will be described and realized. Finally the results of the work done will be presented, especially showing the problems while developing the prototype giving conclusions to solve them. Additionally the benefits of the use of MDA will be discussed.

Vorwort

Smartphones waren in der Vergangenheit nicht für die breite Masse gedacht. Als Firmengeräte wurden sie für den geschäftlichen Einsatz entwickelt. Mit Einführung des iPhones von Apple hat sich dies jedoch verändert. Smartphones sind massentauglich geworden und die Nachfrage nach diesen Geräten steigt immer noch an. Neben Apple haben andere Hersteller diesen neuen Markt auch für sich entdeckt. Die Möglichkeiten moderner Endgeräte sind vielfältig und auch nach Jahren noch immer nicht ausgeschöpft. Deswegen ist es interessant und wichtig, sich dem Gebiet der Applikationsentwicklung für mobile Endgeräte zu widmen.

Danksagung

Zuerst möchte ich mich bei meinen Eltern bedanken, die mich während meines ganzen Studiums unterstützt haben. Ohne sie wäre ich nicht so weit gekommen.

Ferner danke ich meinem Mentor Prof. Friedbert Jochum, der mich über den Zeitraum meiner Diplomarbeit betreute und mir immer wieder neue Anregungen und Aspekte aufzeigte, um meine Abschlussarbeit in die richtige Richtung zu lenken.

Nicht zu vergessen natürlich die Firma KHS GmbH, die mir die Möglichkeit gegeben hat, meine Diplomarbeit in ihrem Hause zu schreiben. Besonderer Dank gilt Herrn Förster, meinem direkten Ansprechpartner.

Dank gilt auch meinen geduldigen Korrekturlesern Jan Van Havelt, Stefan Winopal, Maik P. Schäpperts und Benedikt Schmitz für die investierte Mühe, Zeit und Rotstifte.

Abbildungsverzeichnis

2.1 iPhone 4	18
2.2 HTC Desire HD	18
2.3 Concept Phone 1	18
2.4 Concept Phone 2	18
2.5 Finger Whisper	19
3.1 Use-Case-Diagramm Ersatzteilkatalog	33
3.2 Aktivitätsdiagramm Ersatzteilkatalog durchsuchen	36
3.3 Architektur von YASA-Anwendungen	37
3.4 Definition von Pageflow Profile	38
3.5 Definition des Metamodells View	39
3.6 Dialogfluss Ersatzteilkatalog durchsuchen	40
3.7 Anwendung des View-Metamodells	41
4.1 Architektur des iOS	43
4.2 Die Android-Systemarchitektur	46
5.1 Client-Server-Architektur	59
5.2 Kommunikation Fertigungsmaschine – Mobiles Endgerät	60
5.3 Metamodel View	61
5.4 Start Activity	62
5.5 Login View	63
5.6 Workflow eines Generierungsvorgangs	66
5.7 Baumstruktur der UML-Diagramme; Testfall 1	70
5.8 Baumstruktur der UML-Diagramme; Testfall 2	70

Tabellenverzeichnis

4.1	Smartphone – Marktanteile [Gartner10]	42
4.2	Apple Developer Programs [Applepro10]	45
4.3	JavaScript-Frameworks für mobile Applikationen [CT10], S. 101	53

Abkürzungsverzeichnis

3GL	Third Generation Languages
ADT	Android Development Tools
AES	Advanced Encryption Standard
Apps	ist die Abkürzung von „Applications“. Gemeint sind zusätzliche Anwendungen für Smartphones
BTS	Base Transceiver Station
CI	Corporate Identity
DSL	Domain Specific Language
DVM	Dalvik Virtual Machine
EAP-TLS	Extensible Authentication Protocol - Transport Layer Security
EAP-TTLS	EAP - Tunneled Transport Layer Security
GAF	Geschäftsanwendungsfall
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HMI	Human Machine Interaction
HSDPA	High Speed Downlink Packet Access
HSPA	Highspeed Paket Access
IDE	Integrierte Entwicklungsumgebung
IRDA	Infrared Data Association
ITU	International Telecommunication Union
LEAP	Lightweight Extensible Authentication Protocol
LTE	Long Term Evolution
MAC-Adresse	Media-Access-Control-Adresse
MDA	Model Driven Architecture
MOF	Meta Object Facility
MVC	Model-View-Controller
OMG	Object Management Group
PEAP	Protected Extensible Authentication Protocol
RFC	Request for Comments
SDK	Software Developers/Development Kit
SIM	Subscriber Identity Module
SSH	Secure Shell

TKIP	Temporal Key Integrity Protocol
Visu-Station	Visualisierungs-Station
WEP	Wired Equivalent Privacy
WLAN	Wireless Local Area Network
WPA	Wi-Fi Protected Access
XMI	XML Meta-Data Interchange
XOR	eXclusive OR

1 Einleitung und Aufgabenstellung

Software wird immer komplexer und der Konkurrenzdruck für gewinnbringendes Wirtschaften in einem Unternehmen immer größer. Deswegen ist es für ein Unternehmen wichtig, neue Technologien zu verfolgen und ihren Einsatz zu erwägen. Der Boom von Smartphone-Applikationen in den letzten Jahren hat Einfluss auf viele Bereiche genommen und soll deshalb genauer untersucht werden.

1.1 Zielsetzung und Beschreibung der Aufgabenstellung

Ziel dieser Arbeit ist es, Möglichkeiten aufzuzeigen, mit Hilfe des modellgetriebenen Ansatzes zur Softwareentwicklung Applikationen für Smartphones zu erstellen. Vor- und Nachteile einer solchen Herangehensweise sollen aufgezeigt werden. Dabei wird außerdem geprüft, inwiefern die Kommunikation zwischen einer Fertigungsmaschine und einem Smartphone sinnvoll ist und umgesetzt werden kann.

Diese Diplomarbeit wird als Studie für die KHS GmbH erstellt.

Als Lösungsanbieter für Abfüll- und Verpackungsanlagen stellt KHS Fertigungsmaschinen für die Industrie her. Die Bedienung dieser Maschinen beim Kunden erfolgte in der Vergangenheit direkt an der Maschine ausschließlich über ein Human Machine Interface (später als HMI bezeichnet).

Aufgrund der steigenden Verbreitung von Smartphones und den damit verbundenen Möglichkeiten soll im Rahmen dieser Diplomarbeit überprüft werden, inwiefern die Kommunikation zwischen einer Smartphone-Applikation und einer von KHS hergestellten Fertigungsmaschine sinnvoll sein kann. Dabei werden nicht nur die technischen Rahmenbedingungen geklärt, sondern auch wirtschaftliche Aspekte im Hinblick auf Verbreitung und Marketing aufgezeigt.

Verschiedene Betriebssysteme von derzeit gebräuchlichen Smartphones werden untersucht. Dies erfolgt im Hinblick auf Einstiegsbeschränkungen zur Entwicklung (Komplexität der Hard- und Software). Auch finanzielle Erwägungen (beispielsweise anfallende Kosten für die Entwicklung und den Betrieb der Applikation) werden berücksichtigt.

lende Lizenzgebühren) spielen hier eine Rolle. Für eine fundierte Beurteilung sollen möglichst alle Betriebssysteme einbezogen werden. Dies bedeutet in der Entwicklung einen erheblichen Aufwand, da sich die einzelnen mobilen Endgeräte sehr unterscheiden. Über den Ansatz der „Model Driven Architecture“ wird ein generisches Konzept für das Erstellen einer Applikation entwickelt. Dieses Konzept beinhaltet die Umsetzung für verschiedene Betriebssysteme/Endgeräte. Die dabei entstehenden Probleme werden analysiert, mögliche Lösungen aufgezeigt und wenn möglich behoben.

Weiterhin relevant sind die Rahmenbedingungen für jedes System und die Vorgaben der Smartphonehersteller. In dieser Arbeit wird dargelegt, welche speziellen Restriktionen und Möglichkeiten sich bieten. Dies bezieht sich nicht nur auf die Entwicklung, sondern auch auf die Hardware der Endgeräte.

Es werden verschiedene Anforderungen im Hinblick auf die unterschiedlichen Eigenschaften der Endgeräte formuliert. Diese Einschränkungen sind bei der Softwareerstellung besonders wichtig. Ein kleines Display und diverse andere Eigenschaften (vor allem die Sensorik) der Smartphones müssen berücksichtigt und durch geeignetes Design abgedeckt werden.

Ein Programm in Form eines Prototypen wird spezifiziert und mit Hilfe der MDA auf mobile Endgeräte umgesetzt.

Diese Diplomarbeit soll auch als Grundgerüst für zukünftige Entscheidungen und Entwicklungen im Bereich Applikationen für Smartphones dienen und Aufschluss darüber geben, ob und inwiefern Applikationen für den industriellen Einsatz sinnvoll sind.

KHS möchte im Rahmen der Erforschung neuer Technologien Anschluss an den Markt finden, um moderne Konzepte in der Firma einzuführen und diese für die Produktion/Weiterentwicklung von Fertigungsmaschinen zu nutzen. Der Erfolg von modernen Mobiltelefonen und deren Verbreitung hat großes Potenzial, was die Wachstumsraten im Verkauf bestätigen. Dieser Markt soll im Rahmen der Möglichkeiten genutzt werden, um die KHS Linie entsprechend zu erweitern.

1.2 Das Unternehmen KHS

„KHS¹ ist Lösungsanbieter technologisch innovativer und hochwertiger Abfüll- und Verpackungsanlagen für die Getränke- und Nahrungsmittelindustrie. Mit mehr als 5.500 Mitarbeitern weltweit realisiert der zentral von Dortmund aus geführte Konzern heute einen Jahresumsatz von fast einer Milliarde Euro.“

¹Die KHS AG wurde 1993 als Zusammenschluss der Firmen Holstein & Kappert GmbH und SEN unter dem Dach der Klöckner-Werke AG gegründet. Die Abkürzung „KHS“ steht für die ursprünglichen Firmen Klöckner, Holstein und Seitz.

Neben Produktionsstätten in den USA, Mexiko, Brasilien, Indien und China ist KHS mit mehreren Werken in Deutschland vertreten.

Das Unternehmen ist eine 100-prozentige Tochtergesellschaft der Salzgitter AG, einem der führenden Stahl- und Technologie-Konzerne Europas.

Seit 2008 sind die Unternehmen KHS Corpoplast, KHS Plamax, KHS Moldtec und KHS Asbofill in den Konzern integriert. Damit ist die KHS insbesondere im Bereich ganzheitlicher PET-Lösungen bestens aufgestellt. Durch neue Entwicklungen wie Trockenteil-Lösungen in Modulbauweise, universelle Füllsysteme, Hochleistungs-Etikettiertechnik oder moderne Kommunikations- und Diagnosetechniken festigt die KHS fortlaufend ihre Position als Innovator im Markt.“ [KHS10]

1.3 Grundlagen

Für das Verständnis der nachfolgenden Kapitel werden kurz wichtige Verfahren benannt und Techniken beschrieben, die hier Anwendung finden.

1.3.1 Model Driven Architecture

Von modellgetriebener Software-Entwicklung spricht man, wenn Modelle nicht lediglich zu Dokumentationszwecken eingesetzt werden, sondern zentrale Elemente eines lauffähigen Systems bilden. Dabei ist die Anwendungslogik nicht in einer 3GL-Programmiersprache² definiert, sondern in Modellen spezifiziert. Ausgangspunkt eines Vorgehens nach MDA ist ein spezialisiertes, nicht programmiersprachenbasiertes Wissen, welches als Domäne/Anwendungsdomäne bezeichnet wird. Diese Domäne wird mit einer ausgewählten Sprache (Domain Specific Language) beschrieben. Die Definition einer Domain Specific Language (DSL) ergibt nur dann Sinn, wenn Modellelemente den Problemraum genauer repräsentieren können als eine 3GL-Sprache. Die DSL definiert die Bedeutung des Modells. Dabei kann die Syntax textuell oder grafisch sein, denn die Kernidee der MDA ist, dass die Spezifikation der Softwarekomponente unabhängig von ihrer technischen Umsetzung beschrieben wird.

Basis der MDA bildet die Meta Object Facility (MOF). Sie beschreibt das zugrundeliegende Metamodell für die MDA-konformen Tools, die zur Modellierung verwendet werden. Die MOF wird benötigt, um die DSL anzuwenden. [Stahl07], S. 377ff

²Third Generation Language – Höhere Programmiersprache

In der Vergangenheit hat sich gezeigt, dass Software-Systeme schnell unübersichtlich werden und auf ein nicht mehr zu beherrschendes Maß anwachsen. Abhilfe bot ein Wechsel auf eine höhere semantische Ebene (Abstraktionssprung der Programmiersprache). Maschinensprachen wurden durch Assembler ersetzt, es folgten prozedurale Sprachen, die wiederum durch objektorientierte Sprachen abgelöst wurden. Ausgehend von dieser Entwicklung lässt sich schlussfolgern, dass die Zukunft des Software-Engineerings in den Modellierungssprachen liegt. [Gruhn06], S. 14ff

1.3.2 Arbeitsweise

MDA arbeitet mit verschiedenen Ebenen, die vom abstrakten Entwurf eines Systems zur konkreten Umsetzung führen sollen. Grundlage dieser Schritte bilden die DSL.

Computation Independent Model (CIM)

Das CIM liefert eine Sicht auf das Gesamtsystem, unabhängig davon wie es implementiert werden soll. Es enthält die Anforderungen des Systems an die Umwelt.



Platform Independent Model (PIM)

Das PIM beschreibt die formale Struktur und die Funktionalität des Systems.



Platform Specific Model (PSM)

Durch Anreicherung des PIM mit plattformabhängigen Informationen entsteht das PSM.



Code

Aus dem PSM wird der Quellcode für die Zielplattform generiert. Meist entsteht dabei noch kein ausführbarer Code, sondern lediglich ein Grundgerüst, welches für die weitere händische Implementierung genutzt wird.

Mit MDA gelangt man vom unabhängigen Modell über Transformationen bis hin zum eigentlichen Programmcode. Transformationen bilden die Grundlage für die Überführung von einer Entwicklungsebene in die nächste. Der Prozess der Transformation soll bei der MDA automatisiert durchgeführt werden. Dazu dienen vorher festgelegte Transformationsvorschriften.

Ziele der MDA sind:

- Konservierung der Fachlichkeit
- Portierbarkeit
- Systemintegration und Interoperabilität
- Effiziente Softwareentwicklung
- Domänen-Orientierung

[Gruhn06], S. 21 ff

Mehr Informationen zur Model Driven Architecture findet man auf der Webseite der OMG³ oder in der Literatur ([Gruhn06]).

³<http://www.omg.org>

2 Applikationen für Smartphones

In diesem Kapitel wird der Begriff „App“ erläutert und es wird darauf eingegangen, warum dieses Thema momentan so interessant ist. Ein weiterer wichtiger Bestandteil dieser Arbeit ist der Sicherheitsaspekt einer solchen Anwendung, der im weiteren Verlauf konkretisiert wird. Abschließend wird erläutert, welche Übertragungstechniken für verschiedene Applikationen sinnvoll sind.

2.1 Handykultur im heutigen Zeitalter

Entwickelt wurde das Mobiltelefon, um auch unterwegs unabhängig vom Kabelnetz Telefonate führen zu können. Doch bald war die Telefonfunktion nicht mehr die einzige bzw. relevanzteste Funktion am „Handy“.

„Zur Erfindung der SMS als „unbeabsichtigtes Nebenprodukt“ des Mobiltelefonierens kam es im Jahr 1992 eher zufällig, als einige Entwickler der Firma Vodafone sich einige noch überschüssige Übertragungskapazität des Handy-Signalisierungskanals zunutze machten und über diesen eine Textnachricht mit dem Wortlaut „Merry Christmas“ an das Handy eines Kollegen verschickten.“ [Schmalenbach05], S. 3

Die International Telecommunication Union (ITU) gibt an, dass im Jahr 2010 weltweit 6,1 Billionen SMS verschickt wurden. Von 200.000 Kurzmitteilungen pro Sekunde ist die Rede. Die Verbreitung des Handys nimmt weltweit zu. Die ITU schätzt die Zahl der am Jahresende 2010 vorhandenen Handyverträge auf über 5 Milliarden. Dabei sind die Industrienationen führend in der Nutzung von Mobiltelefonen. [ITU10]

Mobiltelefone haben bereits einen riesigen Markt abgedeckt und trotzdem steigt die Nachfrage nach ihnen weiter an. Sie sind relativ kurzlebig und werden meist mit Vertragsverlängerung durch ein neues Gerät abgelöst. Damit ergibt sich eine Nutzung des Geräts über 12 - 24 Monate (reguläre Vertragslaufzeiten). In kürzester Zeit wechseln Handygenerationen und Hardware. Mobiltelefone können immer mehr. Sie sind Telefon, Organizer sowie Spielekonsole in einem und bieten dem Nutzer viele verschiedene Funktionen an, die vor noch nicht allzu langer Zeit nur auf Desktoprechnern ausführbar

waren. Die Generation der Mobiltelefone mit diesem Leistungsumfang wird als Smartphone bezeichnet.

Aktuelle Vertreter auf dem Markt sind das Apple iPhone 4 (Abbildung 2.1) mit dem Betriebssystem iOS und das HTC Desire HD, welches mit Android betrieben wird (Abbildung 2.2).



Abbildung 2.1: iPhone 4



Abbildung 2.2: HTC Desire HD

Es gibt viele Konzepte für Smartphones und wie diese in der Zukunft aussehen könnten. Entwürfe mit verschiedensten Materialien sollen Handys flexibler machen oder neue Formen von Design aufzeigen (Abbildungen 2.3 und 2.4).

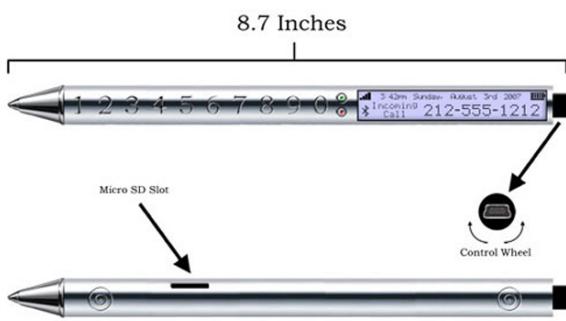


Abbildung 2.3: Concept Phone 1

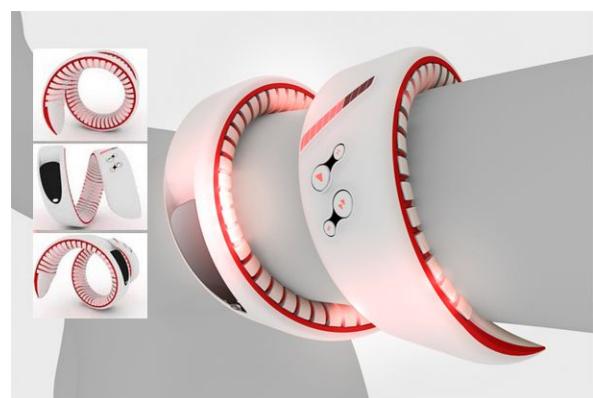


Abbildung 2.4: Concept Phone 2

Ein innovativer Ansatz ist es, wie beim „Finger Whisper“, den Menschen als „Hardware“ in das Mobiltelefon zu integrieren (Abbildung 2.5). Dabei erzeugt ein Terminal am Handgelenk aus Sprachinformationen Schallwellen, die über das Handgelenk weitergegeben werden. Diese Schallwellen gelangen über den Zeigefinger ans Ohr und werden dort wieder zu verständlicher Sprache. [CHI99]



Abbildung 2.5: Finger Whisper

2.2 Begriffsklärung Applikation

Diese Arbeit beschäftigt sich mit Anwendungen für Smartphones. Im allgemeinen Sprachgebrauch werden diese Anwendungen als „App“ bezeichnet. Auch wenn das iPhone sehr viel zur Popularität von Apps beigetragen hat, hat Apple diese nicht erfunden. Allerdings verzeichnet der Apple App Store, die zentrale Vermarktungsplattform für Anwendungen des iPhone, iPod touch, sowie iPad bereits über sieben Milliarden Programm-Downloads. [Apple10]

„App“ steht für Application (Englisch für Anwendung). Bei Kombination von mobilem Endgerät (Smartphone) und einer darauf laufenden Anwendung spricht man daher von einer „App“.

Ihr Nutzen lässt sich wie folgt kurz beschreiben: Eine App soll dem Anwender neue Funktionen in Form von Programmen, wie z.B. Spielen oder Anwendungen, auf seinem mobilen Endgerät zur Verfügung stellen. Dabei gibt es Apps mit ausschließlichem Unterhaltungscharakter. Andere wiederum ermöglichen dem Nutzer den Abruf aktueller Informationen (Ausgehtipps mit Restaurant- und Veranstaltungsinformationen, Wetteraussichten, aktuelle Wechselkurse oder die günstigste Tankstelle).

Die Bedeutung von Apps als Marketinginstrument wird von immer mehr Unternehmen gezielt eingesetzt. Über die Smartphone-Applikation ist die Marke jederzeit beim Anwender präsent. Kundenbindung erfolgt unmittelbar.

2.3 Sicherheit in mobilen Netzen

Besonders wichtig ist der Sicherheitsaspekt bei der Entwicklung von Smartphone Applikationen für Unternehmen. Firmeninterne Daten sollen weder von Dritten eingesehen werden noch in irgendeiner Form manipulierbar sein.

2.3.1 Sicherheitsprobleme in mobilen LANs und WANs

„Funknetze haben im Gegensatz zu leitungsgebundenen Netzen zusätzliche Gefährdungspunkte, die sich zumeist aus den verwendeten Übertragungsprotokollen und der nur begrenzt kontrollierbaren Ausbreitung der Funkwellen ergeben.“ [Eren06], S. 259

2.3.1.1 Bluetooth

Bluetooth ermöglicht eine drahtlose Datenübertragung zwischen zwei Endgeräten. Bei der Übertragung wird kein direkter Sichtkontakt zwischen den beteiligten Systemen benötigt (vgl. Infrared Data Association).

In der Spezifikation von Bluetooth wird kein bestimmter Verschlüsselungsalgorithmus vorgeschrieben, deswegen ist in der Standardkonfiguration vieler Hersteller die Verschlüsselung abgeschaltet. Aber auch wenn diese aktiviert sein sollte, heißt dies nicht unbedingt, dass die Verbindung sicher ist. Der gängige Bluetooth-Verschlüsselungsalgorithmus baut auf einer XOR-Verknüpfung von Klartext und Schlüsselstrom auf. Ein Angreifer kann Teile der versendeten Nachricht herausfiltern, da der zur Kommunikation benutzte TCP/IP-Header eine bekannte Form hat (hinsichtlich Größe und Aufbau). Als Folge des bekannten Headers reduzieren sich mögliche Schlüsselkombinationen (Schlüssellänge von 128 Bit auf 84 Bit). „Brute-Force“-Angriffe werden somit erleichtert. [Eren06]

Ein anderer Angriffspunkt ist der Zufallszahlengenerator. In einigen Sicherheitsfunktionen werden Zufallszahlen verwendet. An die Implementierung dieser werden seitens der Bluetooth-Spezifikation keine expliziten Anforderungen gestellt. Da verschiedene Hersteller bekannte Algorithmen wählen, können die Ergebnisse dieser durch die allgemeine Bekanntheit teilweise vorhergesagt werden.

Der bei einer Authentifizierung notwendige vierstellige PIN-Code stellt auch bei einer 128-Bit-Verschlüsselung immer noch keine ausreichende Sicherheit zur Verfügung. Im Auslieferungszustand eines Geräts ist dieser meist auf „0000“ gesetzt. Wenn der PIN-

Code vom Angreifer richtig ermittelt werden kann, hat er die Möglichkeit, die Kommunikation von gepairten Geräten ungestört zu belauschen.

Alle Sicherheitsdienste bei Bluetooth sind in der Datenübertragungsschicht (vgl. ISO-OSI-Schichtenmodell) angesiedelt. Somit gibt es keine Ende-zu-Ende-Sicherheit. Die Verbindung wird zwar verschlüsselt, aber es fehlt eine nahtlose und durchgehende Datenverschlüsselung zwischen den Endgeräten.

Bei Bluetooth-Push-Diensten wird vor allem das OBEX-Protokoll eingesetzt. Auch dieses hat Schwächen. Der Stack wird vom Zulieferer und nicht vom Hersteller implementiert. Somit kann es passieren, dass Bluetooth-Geräte, die den gleichen Stack und das gleiche virtuelle Dateisystem besitzen, unauthorisierte Zugriffe auf das Dateisystem (z.B. Telefonbuch) unbeabsichtigt zulassen.

2.3.1.2 Global System for Mobile Communications

„Global System for Mobile Communications“ (GSM) ist ein Standard für volldigitale Mobilfunknetze. GSM wurde auf Basis der bereits bestehenden Festnetztechnik entwickelt. Die Verschlüsselung der Daten wurde nur als Option definiert, da in einigen Ländern eine Verschlüsselung von staatlicher Seite untersagt ist. Auch wird auf den meisten Endgeräten nicht angezeigt, ob eine Verbindung verschlüsselt ist oder nicht. Ein weiterer Schwachpunkt ist, dass Daten nur zwischen der BTS (Base Transceiver Station bzw. GSM Basisstation) und dem Teilnehmer verschlüsselt werden. Das Abhören dieser Verbindungen ist an den diversen Schnittstellen der BTS zu anderen Netzkomponenten ohne großen Aufwand möglich. Der potentielle Angreifer verschafft sich physischen Zugang zu einem BTS und kann von dort die Kommunikation belauschen bzw. manipulieren. [Eren06]

2.3.1.3 General Packet Radio Service

„General Packet Radio Service“ (GPRS) ist eine Erweiterung des GSM-Netzes um paketorientierte Datenübertragung. Der Einstieg über das GPRS-Netz ins Internet wird als G_i -Schnittstelle bezeichnet. Während der Datenkommunikation über GPRS ist der Teilnehmer anfällig für im Netz verbreitete Gefahren und Angriffe. Falls der Datenverkehr unverschlüsselt stattfindet, besteht die Gefahr, dass Daten durch einen Angreifer abgefangen und missbraucht werden. Häufig rechnen Mobilfunkbetreiber mit solchen Gefahren und platzieren eine Firewall an der G_i -Schnittstelle.

2.3.1.4 Universal Mobile Telecommunications System

„Universal Mobile Telecommunications System“ (UMTS) ist eine Erweiterung auf Basis der verbreiteten GSM Technik. Es ist abwärtskompatibel zu GPRS/GSM. Prinzipiell ist diese Technologie anfällig für Denial-of-Service- und Impersonationsangriffe. Unter Impersonationsangriffen versteht man die Verwendung eines kompromittierten Authentifizierungsvektors, um sich unter Vorspiegelung der Identität des wahren Teilnehmers an das Netz anzumelden.

2.3.1.5 High Speed Downlink Packet Access

„High Speed Downlink Packet Access“ (HSDPA) ist ein Teil von High Speed Packet Access (HSPA) und ein Protokoll-Zusatz für UMTS-Mobilfunknetze. Für eine höhere Datenrate setzt HSDPA effizientere Modulationsverfahren ein. Mit Hilfe der Modulationsverfahren QPSK (Quadrature Phase Shift Keying / Vierphasenmodulation) und 16-QAM (16 Level Quadratur Amplituden Modulation) erhöht HSDPA die Übertragungsrate im UMTS-Netz. Analog zu UMTS gelten bei HSDPA die gleichen Sicherheitseinschränkungen.

2.3.1.6 Long Term Evolution

„Long Term Evolution“ (LTE) ist eine Weiterentwicklung von UMTS und HSPA. Der damit einhergehende Entwicklungssprung ist mit dem von GSM auf UMTS vergleichbar. LTE erhöht die Kapazität gegenüber HSPA um den Faktor 3,5 und gegenüber HSPA+ um den Faktor 2,5 (im gleichen Frequenzspektrum).⁴

2.3.1.7 Wireless LAN

Die mit der technischen Spezifikation verbundene Systemoffenheit von Wireless LAN (WLAN) bietet zuverlässige und bequeme Einstiegspunkte in ein Netzwerk. Diese Zugriffspunkte eröffnen jedoch abgeschottete und private Netzwerke für Dritte. Im Folgenden werden die verschiedenen Methoden der Verschlüsselung für WLAN erläutert.

2.3.1.7.1 Wired Equivalent Privacy

⁴LTE findet gerade erst Einzug in die weltweiten Netze der Betreiber. Mit Bekanntgabe von Sicherheitslücken hält man sich dementsprechend zurück. Zum Zeitpunkt der Bearbeitung dieser Diplomarbeit standen keine weiteren Informationen zur Verfügung.

Das „Wired Equivalent Privacy Protocol“ (WEP-Protokoll) verwendet zur Absicherung einer Verbindung den symmetrischen Algorithmus RC4.

„Dieser ist ein Stromverschlüsselungsalgorithmus, der den Klartext über eine XOR-Operation mit einer Folge von Pseudozufallszahlen verknüpft.“ [Eren06], S. 287

Der WEP-Key wird entweder mit 40 Bit (WEP 64) oder 104 Bit (WEP 128) erzeugt.

Unzureichende Implementierungen des Initialisierungsvektors (der zusammen mit dem WEP-Key dazu benutzt wird, die Übertragung zu verschlüsseln) führen in den übertragenen Daten zu Kollisionen. Angreifer können diese Kollisionen erkennen und dadurch auf die unverschlüsselte Nachricht schließen.

Bei WEP fehlt eine gegenseitige Authentisierung. Weil man Netzwerkkomponenten (insbesondere MAC-Adresse) leicht fälschen kann, entsteht eine wesentliche Sicherheitslücke.

Nach RFC 1024 müssen alle IP- und ARP-Pakete mit einem „0xAA“ beginnen. Dies kann von einem Angreifer ausgenutzt werden. Nachdem mit relativ wenig Aufwand genug Chiffretext gesammelt wird, kann der Schlüssel ermittelt werden, da die ersten Bytes des Klartextes gemäß Anforderung RFC bekannt sind.

2.3.1.7.2 Wireless Protected Access

„Wireless Protected Access“ (WPA) ist eine Weiterentwicklung von WEP. Es verwendet das „Temporal Key Integrity Protocol“ (TKIP) als Verschlüsselungsprotokoll. Prinzipiell lässt sich WEP mittels Software- und Firmware-Updates auf WPA upgraden.

Dynamische Schlüssel für jedes versandte Paket erhöhen die Sicherheit bei WPA. Dabei besitzt jeder Benutzer einen eigenen Schlüssel.

WPA ist anfällig für Wörterbuchattacken, da der Preshared Master Key direkt aus der Passphrase und der SSID abgeleitet wird. Für einen solchen Angriff reicht ein aufgezeichneter TKIP-Handshake aus. Allerdings benötigen die anschließenden Berechnungen einen aktuellen High-End-PC, da nur rund 70 Passwörter pro Sekunde geprüft werden können. Falls also ein geeignet starkes Passwort gewählt wurde, geht von dieser Attacke eine nur minimale Gefahr aus.⁵

⁵Einzig der Zusammenschluss von vielen Rechnern zu einer Cloud (Cloud Computing) hat in der Hackerszene gezeigt, dass schlecht gewählte Passwörter sehr schnell entschlüsselt werden können. Durch Wahl eines Passwortes mit Sonderzeichen und Vermischung von Groß- und Kleinbuchstaben kann man dieser Attacke vorbeugen. [Spiegel11]

2.3.1.7.3 WPA2

Der bei WEP und WPA zugrundeliegende Algorithmus RC4 gilt als gebrochen. Deshalb wurde bei WPA2 die AES-Verschlüsselung eingesetzt. Dabei bleibt WPA2 noch zu WPA abwärtskompatibel.

Eine „Brute-Force“-Attacke ist zwar rein theoretisch denkbar, aber sehr unwahrscheinlich. Die Wahrscheinlichkeit für eine identische Prüfsumme eines Message Integrity Code liegt bei ca. 1:1 Million. [Dudek08]

Ausserdem wird bei einer Attacke ein 60 Sekunden andauernder Blackout (Unterbrechung der Verbindung) des Access Points ausgelöst, um etwaige Attacken abzublocken. Hat es der Angreifer nicht auf den Schlüssel abgesehen, kann er auf diese Weise erfolgreich eine „Denial-of-Service“-Reaktion auslösen.

2.3.2 Denkbare Angriffsformen auf Mobiltelefone in einer Firmeninfrastruktur

- Ausspähen von Daten:

Der Angreifer verschafft sich Zugang zu relevanten Daten, die auf dem jeweiligen Gerät gespeichert sind. Dazu gehören Kontaktlisten, E-Mails, SMS und andere vertrauliche Dokumente.

- Fremdnutzung von Diensten und Zugängen:

Wird eine Authentifizierung lediglich beim Einschalten verlangt, kann das Gerät im Verlustfall von Kriminellen als Schlüssel für Dienste und Zugänge genutzt werden. Dadurch können Sicherheitsmechanismen für Firmen-, Service- oder Datenstrukturen ausgehebelt werden.

- Manipulation der Software-Komponenten:

In diesem Fall können Angriffe auf Netzwerke in Verbindung mit der Synchronisation zwischen mobilem Gerät und dem Firmennetz erfolgen. Informationen oder Daten werden dabei abgefangen. Darüber hinaus bietet die Konfiguration von Proxies die Möglichkeit des Abhörens und Aufzeichnens sowie Manipulation (Tracing, Capturing, Logging) der an das Gerät zurückgesendeten Informationen.

- Überwachung:

Moderne Smartphones bieten neben Kommunikationsschnittstellen wie GPRS, 3G, WLAN und Bluetooth auch GPS-Module. Damit lassen sich raumbezogene Referenzinformationen erfassen, um genaue Bewegungsprofile zu erstellen. Dazu ist lediglich eine Manipulation des Geräts notwendig.

2.3.3 Sicherheitsmechanismen

Folgende Mechanismen sollen dazu beitragen, dass firmenrelevante Daten nicht von Dritten ausgespäht werden können.

2.3.3.1 WLAN sichern mit Radius

Beim WLAN-Einsatz in Unternehmen reicht die Authentifizierung über ein gemeinsames Passwort (Shared Secret) mit WPA-PSK nicht. Bei großer Verbreitung wird das Passwort schnell öffentlich bekannt. Spätestens wenn ein temporärer Benutzer einen Zugang bekommen hat, tritt dieses Problem auf. Mit serverseitig zugeteilten Passwörtern für den einzelnen Benutzer lässt sich das Problem lösen.

Für einen solchen Fall ist die Lösung WPA Enterprise gedacht, bei der die WLAN-Basisstation Verbindungsanfragen von ihren Clients über das Protokoll IEEE 802.1x mit einem nachgelagerten Radius-Server aushandelt. Auf Linux-Systemen ist dazu das Open-Source-Paket FreeRADIUS gängig. [Radius10]

Es unterstützt mit dem Extensible Authentication Protocol (EAP) verschiedene kryptografisch gesicherte Methoden (EAP-TLS/-TTLS, PEAP, LEAP), One-Time-Passworte und SIMs. Für die Authentifizierung sind Username/Passwort-Kombinationen oder Zertifikate gebräuchlich.

FreeRADIUS verschlüsselt den Datenverkehr zwischen Basisstation und Radius-Server, damit Daten nicht im Klartext übertragen werden. Voraussetzung dafür ist, dass auf dem Client mindestens ein Stammzertifikat (Root CA Certificate) installiert ist, von dem das Zertifikat des Radius-Servers abgeleitet ist. Über das Stammzertifikat prüft der Client auch, dass er sich beim richtigen Radius-Server authentifiziert.

2.3.3.2 Mobile Device Management

Der Diebstahl von Endgeräten führt am häufigsten zu Sicherheitseinbrüchen in Systemen. [Microsoft10]

Mobile Device Management (MDM) bezeichnet die Möglichkeit, von der Ferne aus auf ein Mobiltelefon zuzugreifen und dieses gegebenenfalls zu sperren bzw. Daten zu löschen oder andere Sicherheitsmechanismen auszulösen.

Die derzeitige Funktionspalette umfasst:

- auf dem Gerät enthaltene Daten sichern und wieder aufspielen („Backup & Restore“)

- Software-Updates zentralisiert und drahtlos aufspielen, um Sicherheitslücken zu schließen („Update Over The Air“)
- ein gestohlenes oder verlorenes Gerät aus der Ferne sperren und seine Daten löschen („Remote Lock & Wipe“) sowie per GPS verfolgen („Mobile Tracking“)
- einzelnen Nutzern differenzierte Rechte zuteilen – vom Internet-Zugang bis zur Installation von Programmen („Policy & Provisioning“)
- Statistiken erstellen, wie ein Smartphone genutzt wird und welche Kosten anfallen („Logging & Accounting“)

Bei den MDM-Lösungen bekommen die Geräte eine Identität zugewiesen, die mit den entsprechenden Zugangsberechtigungen und Funktionseinschränkungen verknüpft ist. Der Nutzer bekommt für sein Smartphone ein Passwort, mit dem er sich einmalig für die Erstkonfiguration anmeldet. Während der Standardnutzung im Alltag können die Firmen-Administratoren mittels MDM zentral Software-Updates verteilen, Geräte sperren oder durch Konfigurationsänderungen schnell auf Sicherheitslücken reagieren. Wird das Smartphone schließlich ausgemustert oder verlässt der Nutzer das Unternehmen, werden mittels MDM die Konfiguration auf ein neues Endgerät übertragen oder Berechtigungen und Unternehmensdaten gelöscht.

Der Markt für MDM-Lösungen ist unübersichtlich. Statistiken über Marktanteile existieren nicht. Zu den bekannteren Unternehmen gehören Mobile Iron, Good Technology und Sybase. Ihre Produkte unterscheiden sich dabei vor allem im Detail – während Good Technology großes Know-how in Sales-Umgebungen verspricht, hebt man bei Mobile Iron das Handling unterschiedlicher Smartphone-Plattformen hervor. Sybase wirbt mit einer guten Anbindung an die hausinterne IT. Zumeist bezahlen Unternehmen einen Sockelbetrag plus eine Lizenzgebühr pro Nutzer. Jährliche Aufwendungen können dabei im fünstelligen Bereich liegen.

Seit Apples iPhone den Einzug in Unternehmen angetreten hat, wird MDM nicht nur für reine Businessgeräte wie die BlackBerrys eingesetzt. Kritisch zu betrachten sind private Anwendungen auf Firmen-Smartphones. Gerade Geräte wie das iPhone laden dazu ein, neben geschäftlichen Apps auch Spiele zu installieren oder Multimedia-Content zu konsumieren. Schließen lässt sich diese Sicherheitslücke derzeit nur, indem Firmen das Aufspielen privater Inhalte verbieten oder per MDM kontrollieren und unterbinden.

2.3.4 Fallunterscheidung Übertragungstechnik

Die zu wählende Übertragungstechnik ist abhängig von der Anforderung an die Applikation. Dazu gehören Standort, auszuführende Funktionen und Sicherheitsaspekte.

Bei Aufgaben, die nahe an der Maschine ausgeführt werden, kann eine der Reichweite nach begrenzte Technik gewählt werden, um das Smartphone mit der Fertigungsmaschine zu koppeln (Bluetooth). Ein Abhören der auftretenden Funkverbindung ist relativ unwahrscheinlich. Dies kann nur geschehen, wenn Unbefugte räumlichen Zutritt zur Fertigungsmaschine haben. Durch entsprechende Sicherheitsbefugnisse von Mitarbeitern und Zutrittskontrolle in verschiedenen Bereichen des Unternehmens kann dies ausgeschlossen werden.

Applikationen, die unabhängig vom Standort funktionieren sollen, müssen mit einer WAN-fähigen Verbindungstechnik ausgestattet werden. Diese birgt verschiedene Sicherheitsrisiken. Da sich der Datenverkehr über viele verschiedene Punkte im Web bewegt, können Datenpakete standortunabhängig abgefangen werden. Nur durch Verschlüsselung des Datenstroms mit hinreichenden Mechanismen kann gewährleistet werden, dass Dritte diese Daten weder abhören noch manipulieren.

2.4 Gefahren beim Einsatz von Applikationen in der Industrie

Bei den Vorteilen, die Smartphone-Applikationen für den firmeninternen Einsatz bieten, muss jedoch bedacht werden, welche Probleme entstehen können.

Prinzipiell besteht bei drahtloser Übertragung das Risiko, dass der Datenverkehr abgehört wird. Durch Sicherheitsmaßnahmen kann gewährleistet werden, dass es sehr schwierig für Angreifer ist, sich Zugang zu diesen Daten zu verschaffen. Allerdings hat sich in der Vergangenheit gezeigt, dass auch als sicher definierte Systeme undefinierte Sicherheitslücken aufweisen können.

In der Industrie müssen Maschinen und Bedienelemente so entworfen werden, dass sie robust den täglichen Beanspruchungen genügen und auch bei unsachgemäßer Benutzung nicht ausfallen. Smartphones dagegen werden für den Privatgebrauch gefertigt. Das Augenmerk liegt dabei eher auf Technik, Lifestyle und Bedienkomfort. Entsprechende Geräte sind nicht für einen professionellen Gebrauch über Jahrzehnte ausgelegt (Einzelne Handymodelle bieten hier eine seltene Ausnahme). Fällt ein modernes Gerät beispielsweise auf den Boden, muss man damit rechnen, dass es danach funktionsunfähig wird.

Gegen den Einsatz von Smartphone-Applikationen spricht auch die Gefahr gravierender Bedienfehler. Beispielsweise die Bedienung der Tastatur erfordert genaues, zielgerichtetes Drücken auf ein relativ kleines Display. Dabei können Eingabefehler entstehen,

die bei der Steuerung einer Fertigungsmaschine Maschinenfehler verursachen, welche zu Produktionsausfällen führen können.

Der Gebrauch von Smartphones ist allgegenwärtig. Bei ständiger Nutzung des Geräts kommt der Mitarbeiter auch in Versuchung, andere firmenfremde Applikationen zu verwenden. Soziale Netzwerke wie Facebook, Twitter und Co., sowie Spiele, brauchen (Arbeits)Zeit und können dazu führen, dass die Produktivität sinkt. Nicht bei allen Geräten ist es möglich, den Funktionsumfang und die Nutzung der Software auf dem Smartphone zu regulieren.

3 Systemspezifikation

Als Vorbereitung zur Implementierung wird bei der Systemspezifikation die zu entwickelnde Smartphone-Applikation genau beschrieben. Dabei erfolgt diese Beschreibung im Hinblick auf Funktionsumfang und Nutzen einer solchen Applikation. Die Art der Beschreibung erfolgt dabei erst textuell und darauffolgend fachspezifisch in UML.

Im Zuge der Konzeption benennen wir unseren Prototypen wie folgt:

YASA – Yet Another Smartphone Application

3.1 Funktionsumfang und Nutzen für KHS

An Smartphone-Applikationen hat die KHS GmbH verschiedene Erwartungen. Im folgenden werden sowohl finanzielle Erwägungen sowie Aspekte des Marketings und der Funktionalität erläutert.

3.1.1 Wirtschaftliche und marketingtechnische Erwägungen

Entgegen herkömmlicher Softwarelösungen fallen bei Applikationen für Smartphones verhältnismäßig geringe Schulungsaufwände (weder Zeit noch Geld) an. Dank der intuitiven Bedienkonzepte von Smartphones ist es möglich, sehr benutzerfreundliche Programme zu schaffen. Jeder, der bereits ein Smartphone bedient hat, kennt die auf Touchscreens eingesetzten Techniken, um Elemente zu manipulieren. Somit werden intuitiv Oberflächen, die sich an gängige Richtlinien der Smartphones halten, vom Nutzer richtig erfasst und angenommen. Eine längere Schulungsmaßnahme, um das Gerät bedienen zu können, entfällt. [Clark10], S. 11ff

Für mobile Endgeräte spricht ebenfalls, dass Smartphones mittlerweile eine ähnliche Verbreitung finden wie normale Handys (Tendenz steigend). Es ist anzunehmen, dass die benötigte Hardware beim Endkunden bereits vorhanden ist. Dadurch entstehen ihm keine zusätzlichen Kosten, was die Arbeit mit KHS-Maschinen attraktiver macht. Der Vorteil für KHS besteht darin, keine mobilen Bedienterminals entwickeln, produzieren und vertreiben zu müssen.

Durch eine standortunabhängig einsetzbare Applikation wird die Flexibilität im Umgang mit der Maschine erhöht. Beispielsweise sollen verschiedene Basisfunktionen auch über die Smartphone-Applikation zur Verfügung gestellt werden. Der Endkunde kann die Maschine räumlich unabhängig bedienen oder warten. Dies ermöglicht eine höhere Reaktionsgeschwindigkeit. Durch den Zeitvorteil ergibt sich Kosteneinsparpotential.

Smartphone-Applikationen können auch die interne Kommunikation beim Endkunden verbessern. Vorstellbar ist es, dass Nachrichten bzw. Protokolle über die Applikation ausgetauscht werden. Firmeninterne Prozesse werden optimiert und führen dadurch zu einer Zeit- und Kostenreduktion. Denkbar ist es auch, bei einem speziellen Maschinenproblem direkt mit der jeweiligen Hotline Kontakt herzustellen. Die mitunter zeitraubende Suche nach dem richtigen Anprechpartner entfällt.

Durch die Entwicklung von Smartphone-Applikationen entsteht bei KHS eine Erweiterung der Produktpalette. Diese horizontale Diversifikation löst möglicherweise beim Kunden eine erhöhte Nachfrage nach KHS-Maschinen aus, weil beim Bedienen von Maschinen Komfort und Funktionsumfang erhöht werden. Steigende Akzeptanz und daraus resultierende erhöhte Nachfrage könnten Grundlage für einen neuen Unternehmensbereich sein. Dieser wäre ausschließlich zuständig für Smartphone-Applikationen und kann beispielsweise auch Software für andere Firmen herstellen, die eine ähnliche Erweiterung der Produktpalette planen. So könnte ein lukrativer Einstieg in einen neuen Markt entstehen.

Mit dem Einstieg in den Applikationsmarkt für Smartphones reagiert KHS auf aktuell boomende Technologien. Entsprechend ihrer Corporate Identity (CI) ist KHS als moderne, zukunftsorientierte Firma immer auf der Höhe der aktuellen technischen Entwicklung.⁶ Mit der Unterstützung von neuen Technologien beweist sich KHS wieder einmal als Innovator. Dort hergestellte Maschinen bieten dem Endkunden durch effektiven Bedienkomfort Zeit- bzw. Kostenvorteile und führen auf diese Weise zu einem deutlichen Wettbewerbsvorteil gegenüber Mitbewerbern.

3.1.2 Funktionsumfang

Folgende Funktionen einer Applikation sind für KHS denkbar:

- Messaging:

Dem Nutzer der Applikation soll es ermöglicht werden, einfache Meldungen und Notizen an das HMI zu schicken bzw. Meldungen zu empfangen.

⁶CI ist der abgestimmte Einsatz von Verhalten, Kommunikation und Erscheinungsbild nach innen und außen. Basis ist das Unternehmensleitbild, welches mit Leben gefüllt wird. Die CI ist also die Persönlichkeit einer Organisation.

- Push Notification:

Verschiedene Zustände der Maschine sollen dem Nutzer als Nachricht übermittelt werden können. Diese müssen allerdings nicht vom Nutzer abgerufen werden, sondern werden von der Maschine auf das Gerät „gepusht“⁷. Dabei sind Fehlermeldungen genauso denkbar wie andere Nachrichten, z.B. wenn eine Maschine einen vorher festgelegten Richtwert überschreitet.

- Monitoring:

Die Produktion und der Durchlauf der Maschine soll am Smartphone in Echtzeit angezeigt werden. Mögliche Datensätze sind hier Auslastung der Maschine, Umbauzeiten, Dauer von Wartungsarbeiten, etc.

- Reporting:

Aktuelle Statistiken der Produktionsdaten in Form von Reports sollen vom Nutzer abrufbar sein. So können Echtzeitdaten beispielsweise in Präsentationen oder bei Besprechungen verwendet werden. (Die Anzeige auf dem Smartphone oder ein Versand per Email auf Anforderung sind dabei denkbar)

- Control:

Es soll durch den Benutzer möglich sein, unkritische Funktionen der Maschine zu steuern. (Mögliche Fehler durch den Benutzer in der Bedienung sollen ausgeschlossen werden)

- Bug-Reporting:

Maschinenfehler sollen in Form von Fehlermeldungen an den Server versandt werden. Diese Meldungen werden dem Support über eine direkte Verbindung zur Verfügung gestellt. Zusätzlich denkbar ist es, dass ein mit der Kamera am Smartphone erstelltes Foto des Defekts an den Bericht angehängt wird.

- View:

Einzelaggregate mit Ihren verschiedenen Status sollen abfragbar sein. Dabei handelt es sich um Aggregate der Maschine, die für spezielle Operationen benötigt werden und normalerweise nicht am Maschinendisplay auftauchen. (Beispielsweise bei Inbetriebnahme oder Störungsbeseitigung)

- Spare Parts Catalogue:

In einem über das Smartphone aufrufbaren Ersatzteilkatalog soll es möglich sein, Teile der verwendeten Maschine nachzubestellen. Bei Fehlfunktion wird von der Maschine überprüft, ob ein Maschinenteil defekt ist. Gegebenenfalls wird dem Benutzer vorgeschlagen, dieses nachzubestellen.

⁷Pushmeldungen sind Nachrichten, die vom Endgerät unabhängig von laufenden Anwendungen empfangen werden und dem Nutzer Informationen über aufgetretene Sachverhalte mitteilen.

3.2 Geschäftsanwendungsfall für eine umzusetzende Applikation

Ein realistischer Geschäftsanwendungsfall (GAF) soll als Grundlage für die Umsetzung von YASA dienen.

3.2.1 Vorgaben

Es soll geprüft werden, ob die Umsetzung des GAF mit MDA technisch möglich ist und wo die Grenzen dieser Arbeitsweise liegen. Dabei wird gezeigt, dass man folgende Elemente aus einem Modell generieren kann.

- Persistenz
- Fachliche Logik
- Benutzerfrontend

3.2.2 Beschreibung des Geschäftsanwendungsfalls

Als Geschäftsanwendungsfall für den Prototyp wurde in Absprache mit KHS folgende Funktion aus Kapitel 3.1.2 ausgewählt:

„Spare Parts Catalogue:

In einem über das Smartphone aufrufbaren Ersatzteilkatalog soll es möglich sein, Teile der verwendeten Maschine nachzubestellen. Bei Fehlfunktion wird von der Maschine überprüft, ob ein Maschinenteil defekt ist. Gegebenenfalls wird dem Benutzer vorgeschlagen, dieses nachzubestellen.“

3.3 Anforderungsdefinition

Die genauere Spezifizierung des Anwendungsfalls erfolgt in textueller Form und dient als Grundlage zur Erstellung des CIM.

„Spare Parts Catalogue“ soll folgende Funktionen aufweisen:

- Ersatzteilkatalog durchsuchen
- Ersatzteil bestellen
- Bestellung stornieren
- Bestellstatus abfragen
- Maschinenfehler melden

Anhand dieser Beschreibung ergibt sich ein Use-Case-Diagramm mit den unterstützten Funktionen für den Nutzer und das System (Abbildung 3.1).

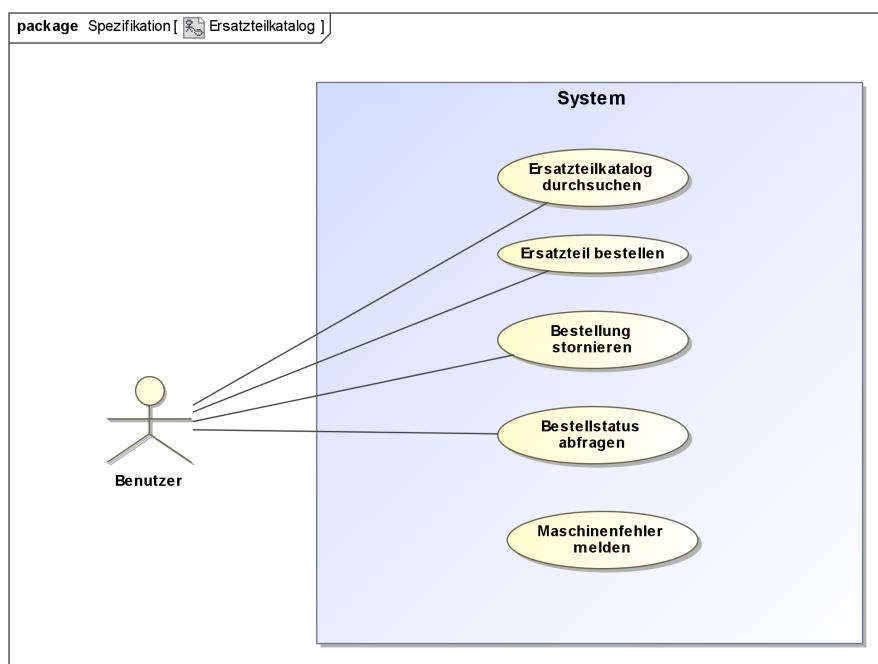


Abbildung 3.1: Use-Case-Diagramm Ersatzteilkatalog⁸

⁸Für die Modellierung der Spezifikation wurde ein Package angelegt, um die Diagramme von der späteren Implementierung im Dateibaum der UML-Diagramme zu trennen. Eine spezielle Bedeutung wird dem Packagename deshalb nicht zugewiesen.

3.3.1 Erstellen eines Computation Independent Model

Die Funktionen von YASA lassen sich im Detail wie folgt beschreiben:

Ersatzteilkatalog durchsuchen	
Kurzbeschreibung	Ein Kunde möchte den Ersatzteilkatalog durchblättern bzw. durchsuchen
Akteur	Benutzer
Involvierte Dinge	Ersatzteilkatalog
Vorbedingungen	Benutzerdaten bekannt
Teilhandlungen	Suchbegriff eingeben Ersatzteile anzeigen Ersatzteil auswählen (Detail ansehen)
Nachbedingungen bei Erfolg	Ersatzteil ausgewählt
Nachbedingungen bei Misserfolg	Kein Ersatzteil gefunden Fehlermeldung ausgeben

Ersatzteil bestellen	
Kurzbeschreibung	Ein Kunde möchte ein Ersatzteil bestellen, dessen Bezeichnung und Funktion ihm bereits bekannt ist
Akteur	Benutzer
Involvierte Dinge	Benutzerdaten, Ersatzteilkatalog
Vorbedingungen	Ersatzteilkatalog durchsuchen / Maschinenfehler melden
Teilhandlungen	Bestellvorgang einleiten Bestelldetails anzeigen Bestellung abschicken
Nachbedingungen bei Erfolg	Bestellung wurde erfolgreich übermittelt
Nachbedingungen bei Misserfolg	Bestellung konnte nicht übermittelt werden

Bestellung stornieren	
Kurzbeschreibung	Ein Kunde möchte eine bereits ausgeführte Bestellung stornieren
Akteur	Benutzer
Involvierte Dinge	Benutzerdaten, Ersatzteilkatalog
Vorbedingungen	Ersatzteil wurde bestellt
Teilhandlungen	Bestelldetails anzeigen Bestellung stornieren

Nachbedingungen bei Erfolg	Bestellstatus wurde geändert auf storniert
Nachbedingungen bei Misserfolg	Bestellstatus wurde nicht verändert (evtl. wurde Bestellung schon versandt)

Bestellstatus abfragen	
Kurzbeschreibung	Ein Kunde möchte den Status seiner Bestellung abfragen (in Bearbeitung, versandt, nicht lieferbar, etc.)
Akteur	Benutzer
Involvierte Dinge	Benutzerdaten
Vorbedingungen	Ersatzteil wurde bestellt
Teilhandlungen	Bestelldetails anzeigen Bestellstatus anzeigen
Nachbedingungen bei Erfolg	Bestellung gefunden
Nachbedingungen bei Misserfolg	Keine Bestellung gefunden Fehlermeldung ausgeben

Maschinenfehler melden	
Kurzbeschreibung	Die Fertigungsmaschine hat einen Fehler festgestellt und kann nicht fortfahren. Eine Systemanalyse stellt fest, dass ein Teil der Maschine defekt ist und sendet dem Benutzer eine Fehlermeldung
Akteur	System
Involvierte Dinge	Fertigungsmaschine
Vorbedingungen	Fehler im System der Maschine Fehler kann ermittelt werden
Teilhandlungen	Systemmeldung mit Fehler erstellen passendes Ersatzteil ermitteln Systemmeldung an Benutzer senden
Nachbedingungen bei Erfolg	Nutzer wurde über Systemfehler informiert und kann Ersatzteil bestellen
Nachbedingungen bei Misserfolg	Meldung konnte nicht übermittelt werden

Als nächstes erfolgt die Umwandlung dieser textuellen Beschreibungen in möglichst genaue Diagramme. Exemplarisch betrachten wir hierfür die Teilfunktion Ersatzteilkatalog durchsuchen.⁹ Zur Erfassung auf grober Detailstufe wird als Darstellung das Aktivitätsdiagramm verwendet (vgl. Abbildung 3.2).

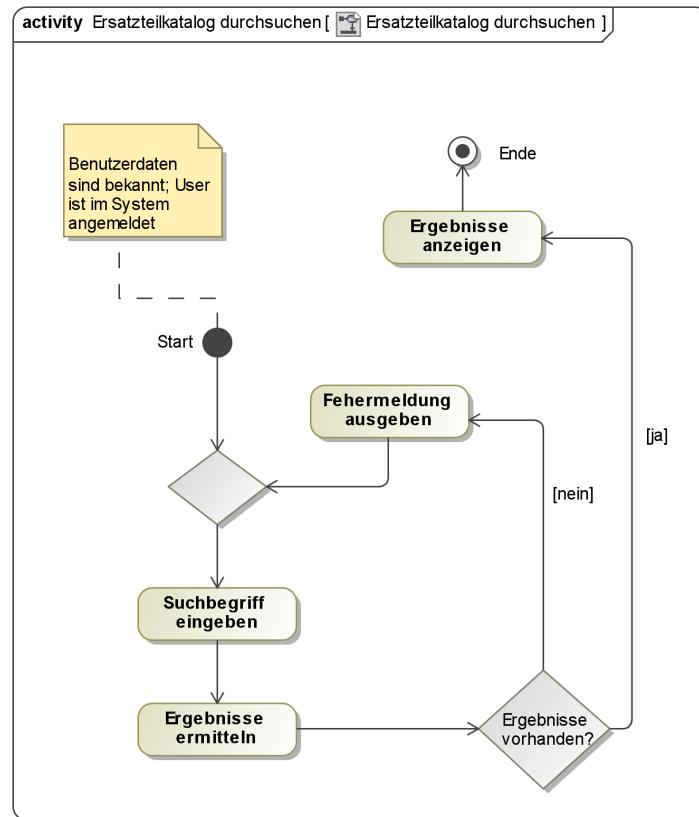


Abbildung 3.2: Aktivitätsdiagramm Ersatzteilkatalog durchsuchen

3.3.2 Erstellen eines Platform Independent Model

Um die dargestellten Funktionen weiter umzusetzen und ein Platform Independent Model (PIM) zu generieren, wird nun die plattformunabhängige Basisarchitektur festgelegt.

⁹Aufgrund der gestiegenen Komplexität wurde während der Implementierungsphase beschlossen, sich zur Umsetzung auf diese Teilfunktion festzulegen. Eine vollständige Auflistung der Diagramme erscheint hier überflüssig und erschwert die Lesbarkeit.

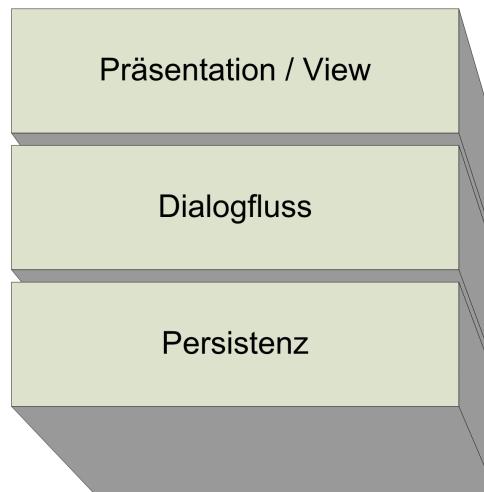


Abbildung 3.3: Architektur von YASA-Anwendungen

Als oberste Schicht in Abbildung 3.3 finden wir die Präsentationsschicht. Diese ist zuständig für die Darstellung des Inhalts auf dem jeweiligen Smartphone in einer für den Benutzer verständlichen Art und Weise.

Darunter befindet sich der Dialogfluss. Hier werden Kontrollflüsse der Anwendung definiert.

Sollen Daten im Rahmen der Geschäftslogik dauerhaft gespeichert und abrufbar sein, wird dies von der Persistenzschicht erledigt.

Die festgelegten Schichten können durch folgende UML-Profile modelliert werden:

- View: Metamodell zur Modellierung der auf dem Smartphone darzustellenden Oberfläche in Form von Dialogmasken. Interaktionsmöglichkeiten des Nutzers mit der Oberfläche und den enthaltenen Schaltflächen werden hier festgelegt.
- Pageflow Profile: Profil zur Modellierung von Dialogflüssen
- Persistence Profile: Modellierung des persistenten Charakters der Anwendung¹⁰

Als Basisdiagramm für den Dialogfluss dient das Aktivitätsdiagramm, welches durch Elemente erweitert wird. Unter anderem mit dem Metamodell View, das die Konstrukte zur Modellierung der Dialogmasken enthält (siehe Abbildung 3.4). Weiterhin werden mit ViewCall und SystemCall die Stereotypen definiert, die zur Modellierung notwendig sind. Der ViewCall ruft anzugezeigende Views auf. Diese sollen auf der Zielplattform grafisch ausgegeben werden. SystemCall ist für Systemaufrufe zuständig. Systemaufrufe sind Methoden, die während der Laufzeit ausgeführt werden.

¹⁰Zur Vollständigkeit der Spezifikation soll das Persistenzprofil zumindest textuell erwähnt werden. Für den GAF findet eine alternative Lösung Anwendung (siehe Kapitel 5.1.4.) Daten werden nicht Clientseitig, sondern auf dem Server verwaltet.

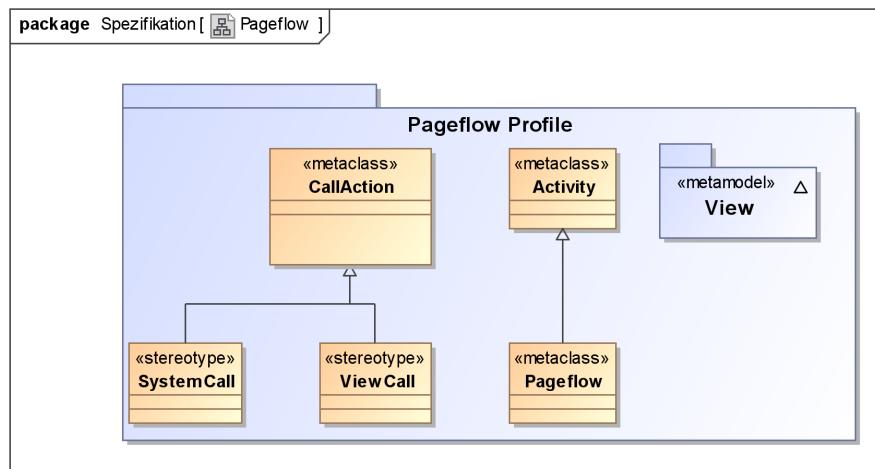


Abbildung 3.4: Definition von Pageflow Profile

Als nächstes zu definieren ist das **View** Metamodell. Hier wird festgelegt, welche Elemente zur Darstellung verwendet werden und wie der Nutzer die dargestellten Elemente manipulieren darf (Abbildung 3.5). Basisbaustein des Metamodells ist der **View**, dem beliebig viele **View**-Elemente zugeordnet werden können. Weiterhin im Modell enthalten sind **OutputElemente** und **InputElemente**. Dabei werden **OutputElemente** in grafischer Form ausgegeben. **InputElemente** sind Eingabefelder und **OutputElemente** zugleich, da diese nicht nur angezeigt werden, sondern auch innerhalb des Pageflow-Kontextes durch den Nutzer manipuliert werden können. Daneben gibt es **CommandElemente**, die Aktionen auslösen (Kontrollflusssteuerung). Durch ein Element des Typs **ContextContainer** werden Nutzdaten den jeweils definierten Klassen zugeordnet.

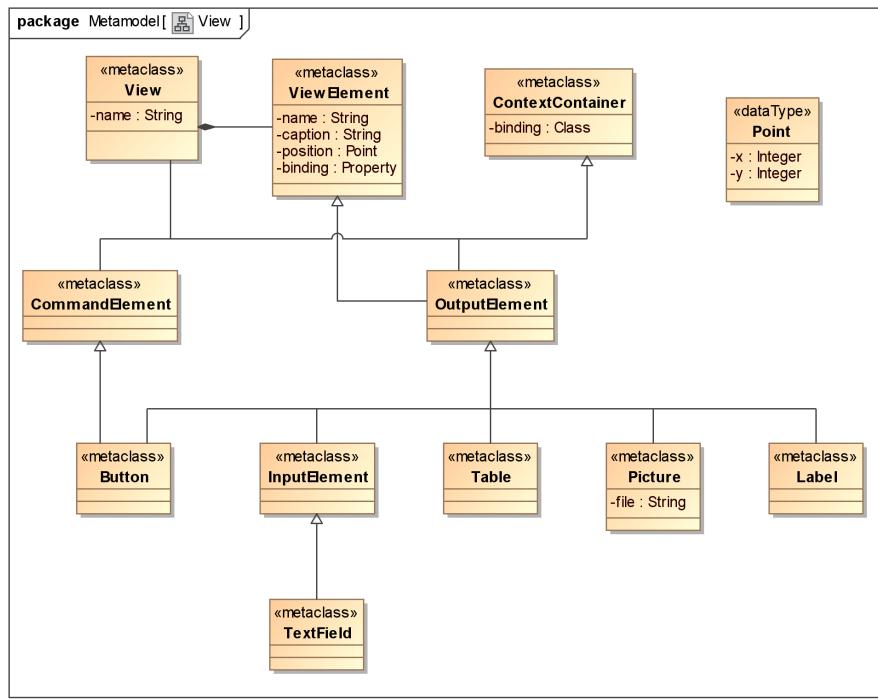


Abbildung 3.5: Definition des Metamodells View¹¹

Besonders zu beachten ist, dass einzelne Klassen keine Methoden besitzen (siehe Abbildung 3.5).

Bei der Implementierung werden die jeweiligen Elemente (TextField, Table, etc.) im Quellcode der Zielplattform generiert. In der entsprechenden API der Zielplattform sind für jedes Element eigene Methodenaufrufe vorgesehen. Die Spezifikation soll unabhängig davon entwickelt werden. Deshalb werden in diesem Kapitel die entsprechenden Methodenaufrufe weder modelliert noch thematisiert.

Nach ihrer Definition werden die verschiedenen Modelle auf den GAF angewendet. Für den Fall Ersatzteilkatalog durchsuchen ergibt sich dadurch der in Abbildung 3.6 dargestellte Dialogfluss.

¹¹ Im Diagramm sind keine Multiplizitäten zu finden, da es sich bei jeder Relation um eine 1:1 Beziehung handelt.

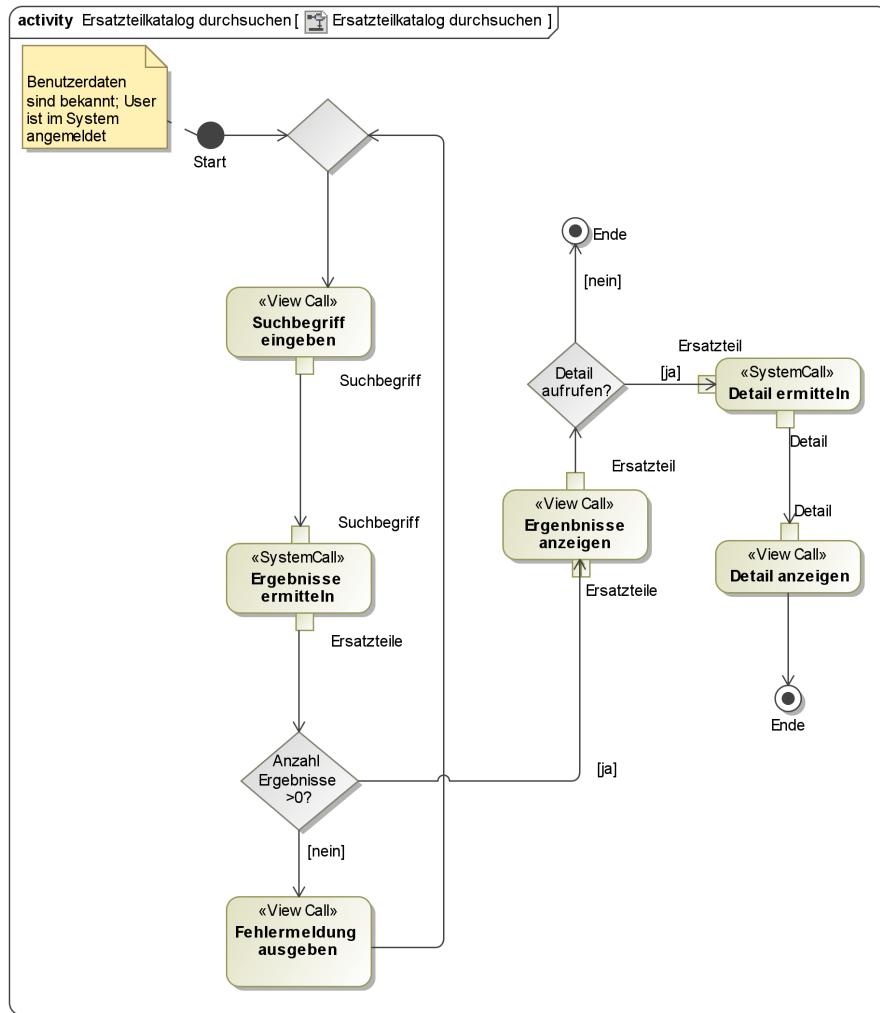


Abbildung 3.6: Dialogfluss Ersatzteilkatalog durchsuchen

Um die gestellten Anforderungen vollständig abzubilden gehören, zur Beschreibung des Dialogflusses weitere Details. So muss jeder von einem ViewCall aufgerufene View weiter detailliert und mit dem entsprechenden Datenmodell verbunden werden (Abbildung 3.7).

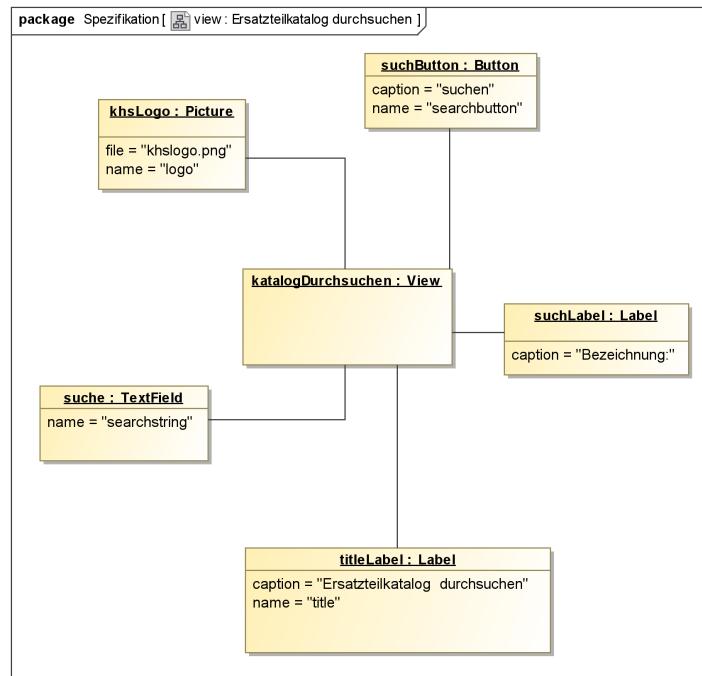


Abbildung 3.7: Anwendung des View-Metamodells

Im nächsten Schritt erstellt man ein auf dem PIM aufbauendes PSM (Platform Specific Model). Da der Entwurf plattformunabhängig bleiben soll und erst im weiteren Verlauf für verschiedene Zielplattformen umgesetzt wird, kann an dieser Stelle auf ein PSM verzichtet werden. Dieses Vorgehen ist in der Softwareentwicklung auch unter dem Namen MDA-light bekannt (vgl. [Gruhn06], S. 49). Der Systementwurf für die umzusetzende Applikation ist hiermit fertiggestellt.

4 Mobile Betriebssysteme und Entwicklungsvoraussetzungen

Durch den modellgetriebenen Ansatz sollen Smartphone-Applikationen für möglichst viele unterschiedliche Endgeräte umgesetzt werden. Dabei trifft man auf verschiedene Betriebssysteme. Für jedes Betriebssystem gibt es spezifische Entwicklungsvoraussetzungen. Das folgende Kapitel zeigt Möglichkeiten und Einschränkungen auf.

4.1 Mobile Betriebssysteme

Der Markt für mobile Betriebssysteme ist relativ groß und unübersichtlich. Im Nachfolgenden werden Eigenschaften von ausgewählten Betriebssystemen analysiert und verglichen, um eine Ausgangsbasis zur Bewertung zu schaffen. Dabei wird verstärkt auf die derzeitigen Marktführer¹² eingegangen (Tabelle 4.1).

Tabelle 4.1: Smartphone – Marktanteile [Gartner10]

Company	3Q10		3Q09	
	Units	Market Share (%)	Units	Market Share (%)
Symbian	29,480.1	36.6	18,314.8	44.6
Android	20,500.0	25.5	1,424.5	3.5
iOS	13,484.4	16.7	7,040.4	17.1
Research In Motion	11,908.3	14.8	8,522.7	20.7
Windows Mobile	2,247.9	2.8	3,259.9	7.9
Linux	1,697.1	2.1	1,918.5	4.7
Other OS	1,214.8	1.5	612.5	1.5
Total	80,532.6	100.0	41,093.3	100.0

¹²Derzeit ist ein Trend zu beobachten, dass Symbian als Marktführer von Android abgelöst wird.

4.1.1 iOS

Bei iOS handelt es sich um ein von Apple entwickeltes Betriebssystem für Geräte wie das iPhone, iPod touch oder iPad. Anwendung finden hier Technologien, die weitgehend dem von Apple entwickelten Desktop-Betriebssystem Mac OS X entsprechen. [Koller10], S. 19ff

4.1.1.1 Architektur

Verschiedene Layer (Abbildung 4.1) beschreiben das iOS. Diese Layer vermitteln zwischen Applikation und Hardware. [Koller10], S. 19

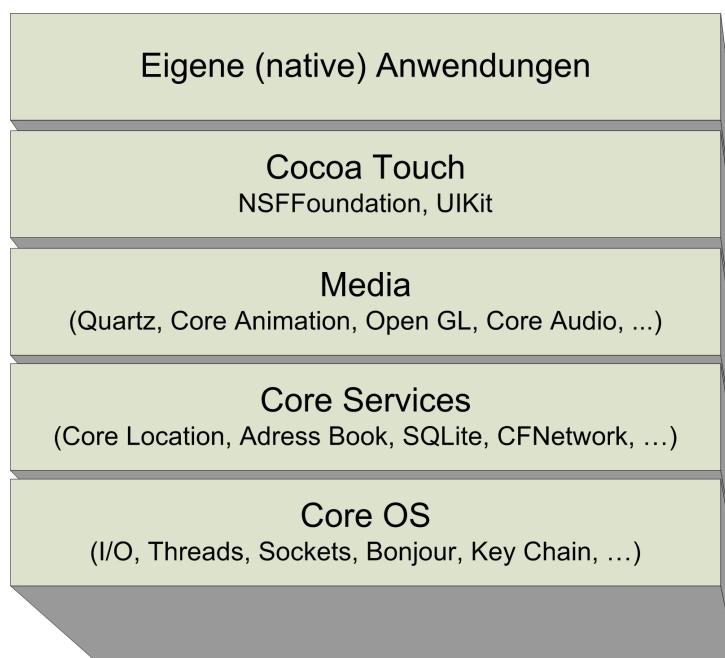


Abbildung 4.1: Architektur des iOS

4.1.1.2 iOS-Komponenten

Der „Cocoa Touch Layer“ beinhaltet die am häufigsten von Entwicklern genutzten Frameworks. Dieser Layer basiert auf der Standard Mac OS X Cocoa API, der an die Bedürfnisse von iOS-Geräten angepasst wurde. Die wichtigsten Komponenten sind das „UIKit Framework“ (zuständig z.B. für das Erstellen des Userinterfaces), der „Push Notification Service“ (Nachrichten werden eventgesteuert an den Benutzer weitergegeben), das „Message UI Framework“ und das „Address Book UI Framework“.

Eine Schicht unter Cocoa befindet sich der „Media Layer“. Dieser ist dafür zuständig, dass das iOS-Gerät mit Audio- und Videodaten sowie (3D-)Grafik umgehen kann.

Mit dem „Core Service Layer“ hat man die Möglichkeit, auf die im System integrierte SQLite Datenbank zuzugreifen. Mit dem „Data Framework“ lässt sich das MVC Prinzip (vgl. Anhang A.4) für Applikationen umsetzen.

Der „Core OS Layer“ ist die Schnittstelle zur Hardware. Hier werden einige Dienste zur Verfügung gestellt, wie Netzwerkzugriff (low level), der Zugriff auf externe Geräte oder betriebssystemspezifische Vorgänge wie Speichermanagement oder Umgang mit Threads.

4.1.1.3 Sicherheit

Entwickelt wird für das iOS-Gerät in der Programmiersprache Objective-C. Der Quellcode wird in Maschinensprache übersetzt und läuft ohne zusätzliche Laufzeitumgebung. Apples Sicherheitskonzept sieht eine Isolierung der Applikationen und Prozesse mittels einer Mandatory Access Control vor. Zugriffsberechtigungen werden nicht nur auf der Basis der Identität des Nutzers vergeben, sondern aufgrund zusätzlicher Regeln und Eigenschaften.

In der Vergangenheit stellte sich immer wieder heraus, dass trotz Sandboxing (vgl. Anhang A.2) zumindest lesend auf Konfigurationsdateien zugegriffen werden kann. Das liegt unter anderem auch an den Mac OS X ähnlichen Zugriffsregeln, die auf regulären Ausdrücken basieren. Dabei werden Zugriffsrechte generisch definiert und nicht auf einzelne Applikationen abgestimmt.

Allerdings existiert neben dem softwareseitigen Sandboxing und dem Code Signing ein hardwareseitiger Schutz. Der ARM-Prozessor des iOS-Geräts unterstützt eine Datenausführungsverhinderung. Diese soll die Auswirkungen von Buffer Overflows und Heap Overflows limitieren. [CT10_2], Seite 80ff

4.1.1.4 Entwicklung

Zur Entwicklung für das iOS muss Apple-Hardware verwendet werden. Die Entwicklungsumgebung Xcode Tools ist nicht für Windows oder Linux erhältlich. Somit scheiden alle anderen Betriebssysteme aus. Allerdings kann man sich nach der Registrierung als „Apple Developer“ auf der Apple Entwicklerhomepage das iPhone SDK kostenlos herunterladen. [AppleDev10]

Dafür erhält man mit dem iPhone SDK für die Xcode Tools eine komfortable Entwicklungsumgebung und einen iPhone Simulator zum Testen der selbstgeschriebenen Applikation.

Tabelle 4.2: Apple Developer Programs [Applepro10]

Apple Programmname	Kosten/Jahr
iOS Developer Program - Individual	99 USD
iOS Developer Program - Company	99 USD
iOS Enterprise Program	299 USD
iOS Developer University Program	kostenlos

Um eine Eigenentwicklung allerdings tatsächlich auf dem Zielgerät laufen zu lassen, muss man sich bei Apple als Entwickler einkaufen. Dies ist in verschiedenen Variationen möglich (vgl. Tabelle 4.2). Nur auf diese Weise kann die Applikation auf das Zielgerät überspielt werden.

Die für das iOS eingesetzte Programmiersprache ist Objective-C. Diese entspricht in Grundzügen der Sprache C, erweitert mit objektorientierten Programmierkonstrukten. Allerdings wurde dabei die Syntax wenig komfortabel auf Objektorientierung angepasst, was den Einstieg in diese Sprache erschwert.

Jailbreak/Hackint0sh

Da Apple sehr restriktiv mit den Entwicklungsvoraussetzungen ist, haben sich „Hacker“ zusammengetan und einen Weg gefunden, diese Voraussetzungen zu umgehen.

Der erste Schritt ist, ein iOS-Gerät zu „jailbreaken“ (siehe Anhang A.2).

Mittels eines SSH-Zugangs lassen sich nun auch Programme auf das Gerät laden, die nicht von Apple signiert sind.

Nicht nur das Zielgerät muss für die Applikation freigeschaltet werden. Prinzipiell ist vorgesehen, dass man nur mit Apple Hardware (z.B. MacBook) und dem Betriebssystem Mac OS X 10.6 (Snow Leopard) entwickeln kann. Da die neuesten MacBooks mittlerweile mit Intel Prozessoren bestückt werden, besteht die (rechtlich bedenkliche) Möglichkeit, das Betriebssystem auch auf herkömmlichen PCs mit Intel und sogar AMD Prozessoren lauffähig zu machen. (siehe Anhang A.3).

Alternativ lässt sich unter Linux bzw. Cygwin mit der Toolchain arbeiten, die den C Compiler um die iOS Frameworkkomponenten erweitert (siehe Anhang A.1.2).

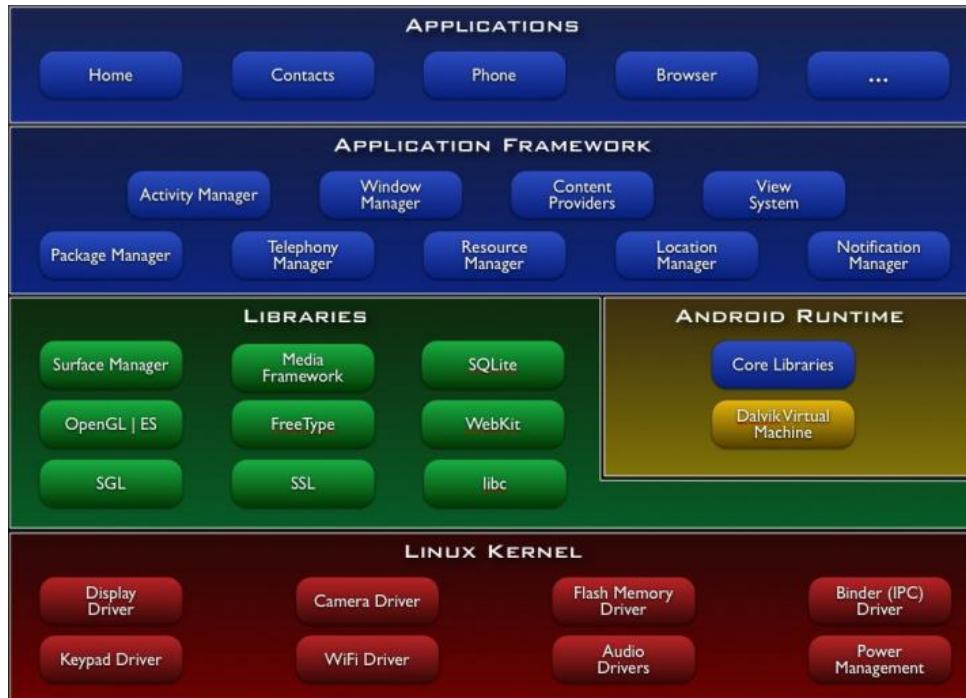


Abbildung 4.2: Die Android-Systemarchitektur

4.1.2 Android

Seit dem 12. November 2007 ist eine Vorabversion des Android-SDK von Google verfügbar. Diese wurde sehr positiv von Entwicklern angenommen. Seitdem wird Android kontinuierlich erweitert und aktualisiert.

Geräte mit Android sind vor allem im Mobilfunkbereich sehr gefragt. Allerdings findet man es auch im Home Entertainment, bei Netbooks, Tablet-PCs und Festnetztelefonen.

4.1.2.1 Architektur

Kern von Android ist ein Linux-Kernel, der speziell auf geringen Energieverbrauch und effizientes Speichermanagement ausgelegt wurde.

Die Android-Laufzeitumgebung ist die Dalvik Virtual Machine (DVM). Sie bildet das Herzstück der Plattform. Android lässt sich komfortabel und ohne Einschränkung in Java programmieren. Allerdings handelt es sich bei der DVM nicht um die reguläre Java Virtual Machine.

„Die DVM und JVM sind grob gesehen sehr ähnlich, wobei sie sich aber in zwei Hinsichten grundlegend unterscheiden: Die DVM ist nicht als Stack-Maschine, sondern als Register-Maschine realisiert, und die Längen der Opercodes betragen bei der DVM zwei anstatt nur einem Byte. Eine auf Register basierte virtuelle Maschine holt ihre Bytecodes und Operanden aus (virtu-

ellen) Registern. Dazu ist es natürlich erforderlich, dass die Operanden in Abhängigkeit des Opcodes in bestimmte Register geschrieben und von dort gelesen werden und nicht generell vom Stack geholt werden, wie es in einer Standard JVM geschieht.“ [Fokus09]

Der Vorteil ist schnell ersichtlich: Bei den Registern handelt es sich um Zwischenspeicher im Mikroprozessor. Dadurch lassen sich Berechnungen über mehrere Zwischenergebnisse stark beschleunigen.

Da die Java-VM und der original Java-Bytecode von Oracle lizenzrechtlich geschützt sind, bediente sich Google eines Tricks. Die DVM wird nicht explizit als Java-VM dargestellt und verarbeitet keinen Java-Bytecode, sondern androideigenen Dex-Bytecode, der nicht unter die Oracle-Lizenz fällt. Es wird also in Java programmiert und mit Hilfe des Java-Compilers von Oracle Bytecode in Form von .class-Dateien erzeugt. Das Android eigene dx-Tool liefert aus dieser Quelle den Dex-Bytecode, der in eine für Android-Geräte ausführbare .apk-Datei (Android Package – fertig ausführbare Anwendung) gepackt wird. Da die Programmierschnittstelle der Java Standard Edition von Oracle bisher noch nicht patentrechtlich geschützt ist, werden hier keine Rechte verletzt.

4.1.2.2 Android-Komponenten

Android stellt eine moderne Plattform für komponentenbasierte Anwendungsentwicklung dar. Wiederverwendbarkeit spielte bei der Entwicklung eine grosse Rolle. Anwendungen oder Teile davon sollen auch in neu entwickelten Anwendungen nutzbar sein. Erklären lässt sich dies an einem Beispiel:

Vorinstalliert auf einem handelsüblichen Android-Gerät findet man Anwendungen wie „Kontakte“ oder „Telefon“. In diesen Anwendungen sind Telefonnummern und Kontakt-daten gespeichert. Möchte man nun die Daten aus den Anwendungen benutzen, kann man über die Datenbank der entsprechenden Applikation (vorhandene Berechtigung vorausgesetzt) direkt auf die benötigten Werte zugreifen.

Die einzelnen Android-Komponenten gliedern sich gemäß des in der Programmierung als Standard geltenden Model-View-Controller-Prinzips (siehe Anhang A.4).

Die Androidkomponente „Activity“ übernimmt die Steuerung des Views. Oberflächen-elemente anzeigen, auslesen von Eingabefeldern und Benutzereingaben werden in der „Activity“ behandelt. Vergleichbar ist dieses Verhalten mit dem Controller im MVC-Prinzip.

Für die Hintergrundprozesse ist im Android-Gerät der „Service“ zuständig. Wichtig sind diese Services, wenn Prozesse weiterlaufen sollen, obwohl die Anwendungen vom Benutzer schon geschlossen wurden (Beispiel: Music Player).

Zuständig für Datenpersistenz in Android ist der „Content Provider“. Analog zum Model beim MVC-Prinzip ist dieser im Allgemeinen für die Verwaltung der darzustellenden Daten zuständig. Diese können über Berechtigungen bestimmten Anwendungen zur Verfügung gestellt werden.

Zusätzlich zu den Standardkomponenten im MVC-Prinzip gibt es bei Android den „Broadcast Receiver“. Dieser empfängt Systemmeldungen z.B. über Störungen des Netzwerks. Applikationen haben so die Möglichkeit, auf einen geänderten Systemzustand zu reagieren.

4.1.2.3 Sicherheit

„Android führt Anwendungen in der Sandbox aus. Eine Sandbox ist eine eingeschränkte Laufzeitumgebung, in der bestimmte Funktionen verboten sind.“ [BeckerPant10], S. 27

Eine Android-Anwendung besitzt aus der Sandbox heraus folgende Eigenschaften:

- eigener Prozess
- für einen eigenen Betriebssystem-User
- eigene DVM
- eigener Bereich im Hauptspeicher (Heap)
- eigener Bereich im Dateisystem

Wird eine Anwendung gestartet, läuft sie unter dem Userkonto in einem eigenen Prozess. Programme sind in Linux gegen den Zugriff von außen geschützt. Dadurch kann kein unerlaubter Zugriff auf den Speicher der DVM erfolgen, die in diesem Prozess läuft.

4.1.2.4 Entwicklung

Prinzipiell ist man bei Android in der Entwicklung weder an Hardware noch Software gebunden. Empfohlen wird allerdings die Eclipse IDE mit Android Development Tools Plugin. [ANDEV10]

Das Android SDK kann entweder von Hand über die Konsole zum Erzeugen eines Projekts benutzt werden oder grafisch mittels der IDE. Anwendungen für die Android-plattform werden ausnahmslos in Java geschrieben.

4.1.3 Windows Phone 7

Relativ neu auf dem Markt ist Windows Phone 7 (Verkaufsstart in Deutschland: 3. November 2010). [Heise10]

Mit Windows Phone 7 will Microsoft sein leicht angestaubtes Mobilbetriebssystemersetzen und zu iOS und Android aufschließen.

Microsoft ist es gelungen, eine einfach zu bedienende Oberfläche zu konzipieren, die sich von iOS und Android abhebt. Der signifikante Unterschied ist die Einbindung von „Live Tiles“. Dabei werden Inhalte von Applikationen bereits als Vorschau im Icon angezeigt.

Microsoft stellt klare Anforderungen an die Hardware für sein Betriebssystem. Windows Phone 7 benötigt einen Prozessor mit mindestens 1 GHz Taktfrequenz und einen internen Speicher von wenigstens 8 GB. Entsprechend der hohen Systemanforderungen laufen die meisten Anwendungen flüssig und ohne Verzögerung.

Applikationen werden in Silverlight bzw. XNA entwickelt. Silverlight ist aus dem Windows Presentation Framework (WPF) heraus entstanden und auf verschiedenen Plattformen und Browsern lauffähig. Das XNA wurde in der Vergangenheit für die Entwicklung von Spielen für den PC bzw. die XBOX 360 benutzt.

XNA ist ein kostenloses Framework und wird als Erweiterung für das Visual Studio 2010 Express angeboten. Microsoft lässt die Entwicklung für seine Geräte nur mit dem eigenen Betriebssystem zu. Damit verfolgt Microsoft eine ähnliche Produktpolitik wie Apple.

4.1.4 Symbian

Entwickelt wird für Symbian mit dem Qt SDK, welches für Windows, Linux und Mac OS X erhältlich ist. Alternativ lässt sich auch der Eclipse-Ableger Carbide verwenden. Programmiert wird in C++. Neu entwickelte Applikationen lassen sich in Nokias Ovi Store vertreiben. Entwickler können sich in diesem für einmalig 50 Euro anmelden.

Die Verwendung von Symbian als mobiles Betriebssystem ist rückläufig. Der Ansatz von Nokia, das Betriebssystem seit dem 24. Juni 2008 unter der Open-Source-Lizenz zu veröffentlichen, führte nicht dazu, dass es wesentlich populärer wurde.

Am 17. Dezember 2010 wurden die Internetseiten von Symbian abgeschaltet. Damit beendete Nokia seine Web-Präsenz für die Open Source-Entwicklung. Die Daten sollen, so die Symbian Foundation, aber auch weiterhin erhältlich sein. Nokia gab bekannt,

dass „aufgrund der Marktentwicklung die Weiterführung des Projekts nicht mehr sinnvoll“ sei. [Symbian10]

In Zukunft sollen Mobiltelefone von Nokia auch mit MeeGo betrieben werden¹³, einem offenen Linux-System, das seine Wurzeln einerseits in Nokias Maemo und andererseits in dem von Intel initiierten Moblin hat. [CT10], Seite 93

4.1.5 BlackBerry OS

Für BlackBerrys gibt es vergleichsweise wenige Applikationen. Dies liegt vermutlich daran, dass ihr Einsatzbereich fast ausschließlich im geschäftlichen Bereich liegt und viele Firmen die Installation von Fremdanwendungen einschränken und/oder verbieten.

Die Herstellerfirma Research In Motion (RIM) setzt auf das im Vergleich zu aktuellen Java-Versionen reduzierte Java ME. Es entspricht ungefähr dem Stand von Java SE 1.3, allerdings mit zusätzlichen Bibliotheken zur Entwicklung von Software für Mobiltelefone.

Empfohlen wird die Entwicklung mit Eclipse und einem entsprechenden Plug-in. Testen lassen sich Applikation mit einem Simulator für fast jedes Gerät von RIM. Allerdings ist der Zugriff auf viele APIs nur mit einer digitalen Signatur möglich. Diese lässt sich RIM einmalig mit 20 USD bezahlen. Möchte man seine selbstgeschriebene Applikation in der BlackBerry App World vertreiben, entstehen einmalig Kosten in Höhe von 200 USD.

4.1.6 HP webOS

HP setzt mit seinem Betriebssystem auf Webstandards. Applikationen werden mit JavaScript, HTML5 und CSS geschrieben. Dabei findet das MVC-Prinzip Anwendung, wobei Views als normale HTML-Dateien erstellt (mit HP-proprietären Attributen und Styles) und Controller mit Hilfe von JavaScript-Klassen dargestellt werden. Diese erhalten definierte Einstiegspunkte und Callbacks. Für die Persistenzschicht zum dauerhaften Speichern der Daten stehen HTML5-konforme APIs und SQLite-Datenbanken zur Verfügung.

Das webOS-SDK ist für Windows, Mac OS X und Linux verfügbar. Zusätzlich bietet HP ein Eclipse-Plugin zum einfachen Packen und Starten der Anwendung aus der IDE. Ein Emulator steht in Form einer virtuellen Maschine für die Virtualisierungssoftware VirtualBox zur Verfügung.

Eine alternative Entwicklungsumgebung für den Browser ist „Ares“. Diese Software

¹³Wie in aktuellen Medienberichten zu verfolgen ist, wird Nokia in Zukunft mit Microsoft zusammenarbeiten und Windows Phone 7 als Betriebssystem einsetzen. Die Weiterentwicklung des eigenen Betriebssystems wird derzeit in den Hintergrund gestellt.

besteht aus einer Projekt-Verwaltung, einem JavaScript-Editor und einem WYSIWYG-Editor.

Applikationen werden nicht signiert und liegen im Quellcode vor. Dies ist kritisch zu betrachten. Leicht lassen sich durch das Auskommentieren einer JavaScript-Zeile Sicherheitsmechanismen aushebeln.

Bei webOS entstehen für die Entwicklung bzw. das Publishing keine Kosten.¹⁴

4.2 Plattformübergreifende Ansätze

Zusätzlich zur Idee, eine Applikation für unterschiedliche mobile Betriebssysteme zu entwickeln, gibt es auch plattformübergreifende Ansätze.

4.2.1 WebApp

Als WebApps bezeichnet man Webseiten mit dem Look and Feel einer Applikation. Sie sind für mobile Geräte im Hinblick auf Darstellung und Benutzbarkeit optimiert worden.

Webapplikationen lassen sich schnell umsetzen, da die benutzte Technik weit verbreitet ist und gängige Browser im Mobilbereich die Standards der Webprogrammierung unterstützen.

Leider ist es dabei nicht möglich, auf die gerätespezifischen Eigenschaften zuzugreifen. Es gibt zwar Ansätze wie beispielsweise Sencha Touch, in der sich das Userinterface den entsprechenden Geräten anpassen lässt, aber Kamera, Accelerometer und andere Smartphonefunktionen sind nicht über ein Webinterface ansprechbar (einzig Lokalisierungsdienste bieten hier eine Ausnahme). So ist die Entwicklung einer WebApp relativ beschränkt und nur für Anwendungen zu empfehlen, die nicht auf die Hardware des Geräts zugreifen.

4.2.2 Cross-Platform-Development-Tools

Cross-Platform-Development-Tools sind Frameworks, mit denen man Applikationen für unterschiedliche mobile Betriebssysteme erstellen kann. Im Vordergrund steht nicht die Performance der Anwendung, sondern die Kompatibilität mit verschiedenen Betriebs-

¹⁴Die ursprünglich von HP festgelegte Gebühr in Höhe von 99 USD zur Teilnahme am Entwicklerprogramm wurde ausgesetzt.

systemen. Dabei will man deren „Look and Feel“ so nahe wie möglich kommen. [CT10], S. 96

Aktuelle Vertreter von Cross-Platform-Tools sind PhoneGap¹⁵, Titanium¹⁶ und Rhodes¹⁷

Sie basieren auf Webtechniken und zielen speziell auf Webentwickler ab, die ihre Fähigkeiten zur Entwicklung von Smartphone-Applikationen einsetzen möchten, ohne die dem Endgerät zu grundeliegende Programmiersprache erlernen zu müssen (vgl. Tabelle 4.3).

PhoneGap und Titanium werden mit Hilfe von HTML, CSS und JavaScript programmiert. Sie greifen über bestimmte JavaScript APIs auf die Funktionen der Smartphones zu. Die Logik (HTML, CSS, JavaScript) wird allerdings in einem Browserfenster angezeigt. Durch die JavaScript APIs hat die Applikation Zugriff auf Funktionen des Handys, wie zum Beispiel GPS, Accelerometer, Kamera, Kontakte, Datenbank oder Dateisystem.

Im Gegensatz dazu wird in Rhodes mit der Programmiersprache Ruby gearbeitet. Hier wird eine komplette Serverumgebung auf dem Gerät und dadurch Zugriff auf die nativen Funktionen geboten.

Prinzipiell kann jede Funktion, die die Zielplattform anbietet, in Javascript überführt werden. Eine normale Web App, die im mobilen Browser des Handys läuft, könnte zum Beispiel nicht auf diese Funktionen zugreifen (Sicherheitsaspekt).

Laut Dokumentation können mit Titanium erstellte Applikationen in native Anwendungen für das Mobiltelefon umgewandelt werden. Dies ist missverständlich formuliert, denn Titanium kann kein HTML, CSS oder JavaScript in eine native Applikation verpacken. Der für Titanium erstellte Code wird als Ressource zu der ausführbaren Applikation hinzugefügt.

Analog dazu ist die Funktionsweise von PhoneGap. Hinsichtlich der Architektur sind sich PhoneGap und Titanium sehr ähnlich. Ein wesentlicher Unterschied besteht darin, dass PhoneGap nicht die nativen Userinterfacekomponenten zur Verfügung stellt. Das „Look and Feel“ muss mittels CSS aufbereitet werden. Dagegen bietet Titanium eine starke UI API, die mittels JavaScript dazu benutzt werden kann, verschiedenste Arten von nativen UI Komponenten zu erzeugen. Im direkten Vergleich wirken Titanium Applikationen eher wie direkt für das jeweilige Smartphone geschrieben. Dabei konzentriert sich Titanium auf iOS und Android¹⁸ und unterstützt damit weniger Zielbetriebssysteme als PhoneGap. Die PhoneGap APIs sind allgemeiner gefasst und können so auf vielen verschiedenen Systemen wie z.B. iOS, Android, Blackberry OS oder Symbian laufen.

¹⁵<http://www.phonegap.com>

¹⁶<http://www.appcelerator.com/>

¹⁷<http://rhomobile.com/>

¹⁸Aktuellen Meldungen der Webseite nach lässt sich Titanium auch bald für Windows Phone 7 verwenden.

Tabelle 4.3: JavaScript-Frameworks für mobile Applikationen [CT10], S. 101

	PhoneGap	Titanium Developer	Rhodes
App-Typ	quasi-nativ	quasi-nativ	(quasi-)nativ
Hersteller	Nitobi	Appcelerator	Rhomobile
Betriebssysteme	alle (Eclipse)	Windows, Linux, Mac OS X	Windows, Mac OS X
Lizenz	MIT	Apache	MIT
Programmiersprache	JavaScript	JavaScript	Ruby
Android	✓	✓	✓
iPhone	✓	✓	✓
WebOS	✓	–	–
Symbian	✓	–	✓
BlackBerry	✓	✓ (nur Pro-Version)	✓
Windows Mobile	✓	–	✓

Das Rhodes Framework gestattet Entwicklern native Applikationen zu erstellen. Es verwendet das MVC-Prinzip, bei dem Logik und Anzeige strikt getrennt sind, wobei die Anzeige in HTML verfasst wird. Das Rhodes Framework hat Grenzen. Audio oder Video können nicht in Applikationen integriert werden. Allerdings bietet Rhodes an, dass man Funktionen aus den nativen APIs des Systems aufrufen kann, wenn diese nicht vom Framework unterstützt werden. Jedoch kann es dabei zu Performance-Einbußen kommen.

Große Unterschiede zeigen sich bei den Lizenzgebühren. PhoneGap ist Open-Source und kann frei genutzt werden. Titanium befindet sich noch im Betastadium, soll aber in Zukunft ein kommerzielles Produkt werden. Rhodes ist bereits kostenpflichtig. Vor Beginn eines jeden Projektes fallen 500 USD Lizenzgebühren an.

5 Implementierung und Test

Vorrangig soll bei der Implementierung gezeigt werden, wie aus den zuvor spezifizierten UML-Diagrammen eine ausführbare Smartphone-Applikation generiert wird. Dafür muss ein Generator so modifiziert werden, dass er in der Lage ist, Diagramme in lauffähige Programme umzuwandeln. Sowohl die erzeugten Programme als auch der Generator müssen getestet werden.

5.1 Implementierung

Zur Vorbereitung auf die Implementierung werden auf dem Markt verfügbare Tools im Hinblick auf Funktionsumfang und Nutzen bewertet. Im Anschluss werden die den Anforderungen am ehesten entsprechenden für die Implementierung ausgewählt.

5.1.1 Vorhandene Tools auf dem Markt

Zugunsten offener Standards und frei verfügbarer Open-Source Technologien soll so weit möglich auf die Verwendung kommerziell-, proprietärer Software verzichtet werden.

Im Folgenden werden erst grafische Tools zur Modellierung der Diagramme untersucht und anschließend Transformationstools, die für die Überführung der einzelnen Modelle in Quellcode verantwortlich sind.

5.1.1.1 UML-Tools

Auf dem Markt verfügbar sind viele verschiedene UML-Tools. Für diese Arbeit wurden die Tools mit der größten Verbreitung genauer untersucht.

5.1.1.1.1 Visual Paradigm

Visual Paradigm ist sehr leicht und intuitiv zu erlernen. Die meisten Funktionen findet man auf Anhieb, ohne auch nur einen Blick in die Dokumentation werfen zu müssen. Allerdings hat sich gezeigt, dass das Arbeiten mit Visual Paradigm in Kombination mit einem Generator nicht viel Sinn macht, da beim Export der Diagramme in XMI-Files Informationen verloren gehen. So fehlen beispielsweise die Namen bzw. Bezeichnungen der Diagramme sowie Zusammenhänge der einzelnen Diagrammarten. Folglich ist Visual Paradigm für den Export von UML-Diagrammen in das XMI-Format nicht geeignet.

5.1.1.1.2 MagicDraw

Tadellos funktionierte der erste Test, Klassendiagramme und Aktivitätsdiagramme zu exportieren. Allerdings ist die Bedienung von MagicDraw wesentlich komplexer und unübersichtlicher als bei Visual Paradigm, was sich negativ auf die Einarbeitungszeit auswirkt. Daher verzögerte sich die Erstellung der Diagramme sehr. Schlechte Navigation, unübersichtliche Menüs und unzureichende Dokumentation zur Software sind die Ursache für einen erhöhten Zeitaufwand. Positiv ist zu bemerken, dass das Programm volle Kompatibilität zum XMI-Standard bietet.

5.1.1.1.3 Schlussfolgerung

Der XMI-Standard wird benutzt, um die Interoperabilität zwischen Modellierungs- und Generierungswerkzeug zu gewährleisten. Mit den selben Modellen erzeugen verschiedene Tools wider erwarten unterschiedliche Ergebnisse.

Momentan sind leider noch zu viele verschiedene XMI-Versionen am Markt vertreten. Dies fällt besonders bei den Modellierungswerkzeugen ins Gewicht. Allem Anschein nach ist beim Export der Diagramme MagicDraw genauer. Deshalb wird im Folgenden dieses Werkzeug verwendet, obwohl wegen seiner Schwächen ein erhöhter Zeitaufwand zu erwarten ist.

5.1.1.2 Generator-Tools

Seit längerem wird versucht mit Generatorprogrammen aus Modellen Code zu generieren. Im folgenden werden zwei dieser Tools näher untersucht.¹⁹

5.1.1.2.1 openArchitectureWare

openArchitectureWare (oAW) bezeichnet eine Plattform für modellgetriebene Softwareentwicklung und modellgetriebenes Testen. Im Wesentlichen bietet openArchitectureWare die Möglichkeit, beliebige Modelle zu verarbeiten. Zu diesen Modellen gehören EMF-Modelle (Eclipse Modeling Framework), fast alle mit UML-Werkzeugen erstellten Modelle, aber auch Microsoft-Visio-Modelle oder textuelle Spezifikationen. Aus den Ausgangsmodellen kann beliebiger Quellcode generiert werden. Dabei bietet oAW im Gegensatz zu vielen anderen Generatoren nicht nur Modell-zu-Text-Transformation, sondern auch die Möglichkeit, eine Transformation von Modell zu Modell durchzuführen.

5.1.1.2.2 Acceleo

Acceleo ist ein Tool zur Modell-zu-Text-Transformation, welches den MDA-Ansatz der OMG verfolgt. Unter Verwendung einer Skriptsprache kann das als XMI hinterlegte Modell in den gewünschten Code transformiert werden.

Einfache Diagramme in Code umzuwandeln ist mit dem Tool schnell zu erreichen. Allerdings ist die Dokumentation unzureichend. Das Durchlaufen von Aktivitätsdiagrammen ist prinzipiell anhand der Aktionen möglich. Allerdings werden diese nicht in der richtigen Reihenfolge darstellt. Acceleo sortiert Objekte so, wie sie im importierten XMI-File des UML-Diagramms gelistet sind. Aus einem Kontrollfluss heraus einen folgerichtigen Ablauf zu erzeugen, erfordert eine komplexe Modifikation des Generators.

5.1.1.2.3 Auswahl des Generatortools

Durch ihren großen Funktionsumfang erfordert die Software oAW eine intensive Einarbeitung mit erhöhtem Lernaufwand. Ihre speziellen Funktionen wie beispielsweise die Modell-zu-Modell-Transformation werden jedoch nicht für die Entwicklung von YASA benötigt.

¹⁹Die Auswahl erfolgte, weil nur oAW in der Lage ist, Modell-zu-Modell-Transformationen durchzuführen und Acceleo ein relativ junges, populäres Projekt ist, dass bei Entwicklern bevorzugt genutzt wird.

Acceleo lässt sich intuitiver bedienen und erfüllt die Anforderungen an das umzusetzende Projekt. Deshalb wird mit Acceleo als Generatortool gearbeitet.²⁰

5.1.2 Zielplattformen für den Prototypen

In der Entwicklung sollen möglichst viele Betriebssysteme für YASA berücksichtigt werden.

Daher wäre es durchaus sinnvoll, Cross-Platform-Development-Tools zu verwenden. Diese haben allerdings den Nachteil, dass die integrierten Bibliotheken den benötigten Speicherplatz einer Smartphone-Applikation unverhältnismäßig vergrößern. Im Hinblick auf Speicherverbrauch sind diese Applikationen für ältere Smartphonemodelle mit weniger Arbeitsspeicher nicht geeignet. Außerdem ergeben sich nicht einzuschätzende Sicherheitsrisiken durch proprietäre Bibliotheken.

Deswegen ist es ratsamer, in der jeweiligen Programmiersprache direkt für die Betriebssysteme zu entwickeln.

Android und iOS finden Einsatz bei Geräten der innovativsten Hersteller. Symbian ist zwar nach wie vor weit verbreitet, wird aber als veraltete Technik nicht weiterentwickelt. Windows Phone 7 befindet sich im Moment noch in der Bewährungsphase. Außerdem am Markt zu finden sind BlackBerry OS und HP webOS sowie weitere Nischenprodukte²¹.

Jedes Betriebssystem erfordert eine eigens für den Generator zu definierende Transformationsvorschrift, die für Syntax, Programmiersprache und Konzepte der Zielplattform speziell entwickelt werden muss. Die Ergebnisse des Generators müssen in der jeweiligen Entwicklungsumgebung des Zielbetriebssystems weiter bearbeitet werden. Dies bedeutet einen immensen Aufwand und würde den Rahmen dieser Arbeit sprengen.

Ziel dieser Arbeit ist es, zu zeigen, ob der modellgetriebene Ansatz für die Art des Projekts geeignet und umsetzbar ist. Dafür ist die exemplarische Umsetzung für ein Betriebssystem hinreichend. Zur besseren Veranschaulichung wurden jedoch zwei Zielplattformen ausgewählt. Plausibel ist dabei die Wahl der aktuell fortschrittlichsten Systeme mit hohem Marktanteil und großem Entwicklungspotential – Android und iOS.

²⁰ Ziel dieser Arbeit soll es sein, ein geeignetes Tool speziell für die Generierung von YASA zu finden – nicht alle auf dem Markt etablierten Systeme zu prüfen. Eine vollständige Evaluation aller auf dem Markt verfügbaren Systeme wäre vom Umfang her als Thema einer anderen Diplomarbeit denkbar.

²¹ Beispielsweise MeeGo, Moblin, bada oder Maemo

5.1.3 Vorbereitung der Entwicklung und Generierung von Testcode

Wichtig für die Umsetzung des Generators für die verschiedenen Zielbetriebssysteme ist die Struktur des Quellcodes.

Dieser wird zunächst geprüft auf Komplexität, Reihenfolge und Struktur von Programmierkonstrukten. Dazu werden händisch Codefragmente erstellt. Dies ermöglicht auch eine Einschätzung des zu erwartenden Aufwandes. Anschließend wird der erstellte Quellcode für die Zielgeräte kompiliert und testweise ausgeführt.

Da für Android sowohl ein Simulator als auch eine sehr komfortable Entwicklungsumgebung frei verfügbar sind, ließ sich in kürzester Zeit ein ausführbares Testprogramm mit verschiedenen grafischen Elementen generieren, die beim Prototypen zum Einsatz kommen sollen. Besonders elegant und einfach ist hier die Trennung zwischen View und Controller, was sich für die Entwicklung durch automatisierte Codegenerierung als wesentlicher Vorteil herausstellt.

Bei iOS lassen sich View und Controller nur dann trennen, wenn man die offizielle Software von Apple benutzt und über die entsprechende Hardware verfügt. Für die Entwicklung während dieser Diplomarbeit stand beides nicht zur Verfügung. Deswegen mussten Umwege gegangen werden (Ausführungen dazu finden sich in Anhang A.1.2 bzw. A.2). Durch Nutzung der inoffiziellen Entwicklungsumgebung der Toolchain war es nicht möglich, Views unabhängig vom Controller zu erzeugen.²² Deshalb mussten die Viewelemente direkt im Controller zur Laufzeit erstellt werden.

5.1.4 Basisstruktur der Anwendung

Im Folgenden wird die Basisstruktur der Client-Server-Architektur beschrieben.

Dafür wird ein schematisch sehr vereinfachter Plan der Infrastruktur erstellt. YASA benötigt mehrere Komponenten (Abbildung 5.1). Die Kommunikation zwischen Client und Server kann drahtlos erfolgen, da die meisten mobilen Endgeräte über eine erforderliche WLAN-Schnittstelle bereits verfügen. Die Verbindung zum Server wird per WLAN über einen Access Point hergestellt. Die Anbindung des Access Points an das firmeninterne Netzwerk findet über einen Router statt. Somit befinden sich beide Geräte im selben IP-Adressbereich und können einfach miteinander kommunizieren. Nach außen hin wird der Zugriff auf das interne Netz durch eine Firewall abgesichert, um unerlaubte Zugriffe aus dem WWW zu vermeiden.

²²Textuell erstellte Views müssen zur Nutzung in iOS von Apple Software kompiliert werden.

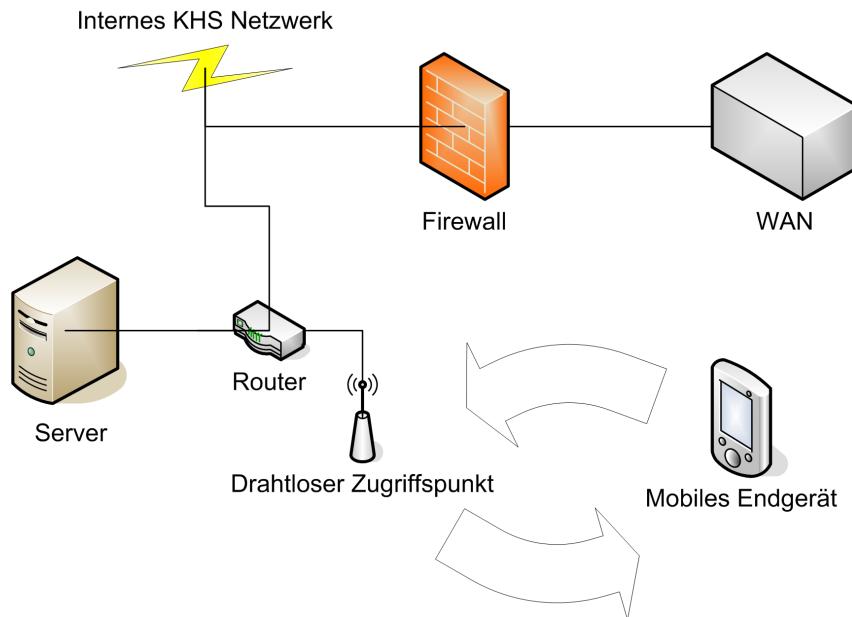


Abbildung 5.1: Client-Server-Architektur

Die automatisierte Codegenerierung soll nur die Client-Anwendungen für die spezifischen Betriebssysteme betreffen. Der für den Zugriff der Clients benötigte Server wird händisch vorbereitet. Auf einem zuvor eingerichteten Debian Linux (Anhang A.5) werden dazu die für Webzugriffe benötigten Komponenten installiert. Dies beinhaltet den Apache Webserver (zur Ausspielung von Webseiten), sowie für die serverseitige Verarbeitung von der Anwendung benötigte Datenbank (MySQL) und eine Skriptsprache zur formatierten Ausgabe der Daten (PHP5).

Zuvor erstellte UML-Diagramme enthalten sogenannte SystemCalls. Ausgehend vom Client rufen SystemCalls Funktionen mit verschiedenen Eingabeparametern auf dem Server auf und sollen eine Antwort erhalten. Diese kann entweder keinen Inhalt haben, wenn die Datenbank keine Ergebnisse zur Abfrage liefert oder die gewünschten Informationen senden. Reagiert der Server nicht, erfolgt keine Antwort.

Dem Client sollen Informationen aus der Datenbank über ein JSON-Objekt zur Verfügung gestellt werden. JSON steht für JavaScript Object Notation und ist ein Datenformat, welches in verschiedenen Programmiersprachen sehr einfach geparsed werden kann. JSON-Objekte haben weit weniger Datenvolumen als das alternativ nutzbare XML Format. Dies hat den Vorteil, dass auch bei sehr niedrigen Übertragungsraten Daten schnell beim Client ankommen. Bei Android ist die Implementierung des JSON Client besonders einfach, da in der zugehörigen API bereits eine Bibliothek dafür vorhanden ist. Beim iOS dagegen muss erst eine passende Bibliothek gefunden und in das Betriebssystem eingebunden werden.

5.1.5 Kommunikation mit der Fertigungsmaschine

Thema dieser Diplomarbeit ist vorrangig die modellgetriebene Softwareerstellung von Smartphone-Applikationen. Allerdings darf nicht vergessen werden, dass die generierte Zielanwendung mit einer Fertigungsmaschine kommunizieren soll. Die Frage ist nun, auf welche Weise eine solche Kommunikation mit dem oben genannten Basiskonstrukt stattfindet.

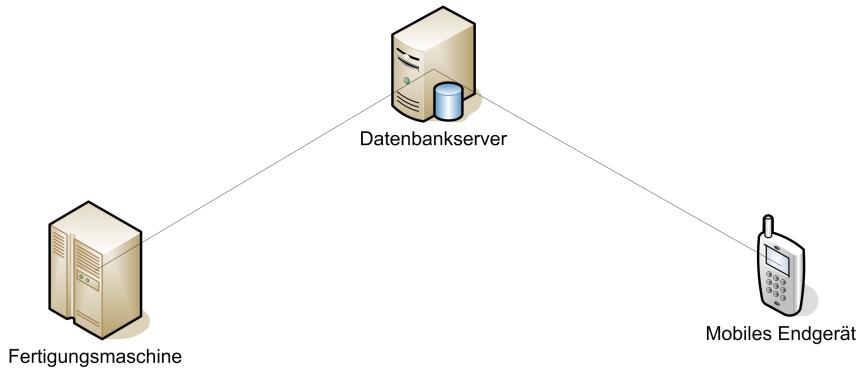


Abbildung 5.2: Kommunikation Fertigungsmaschine – Mobiles Endgerät

Eine direkte Kommunikation zwischen Client und Maschinenframework ist aus sicherheitsrelevanten Erwägungen abzulehnen. Das Framework soll als abgeschottetes Black-box-System unabhängig von der Anwendung funktionieren. Dies wird erreicht, indem das Framework nach geringen Modifikationen direkt auf die Serverdatenbank zugreifen kann. Somit erfolgt eine indirekte Kommunikation zwischen Client und Fertigungsmaschine (Abbildung 5.2). Bei einem entsprechenden Event schreibt das Framework in die Datenbank. Die neuen Daten werden an den Client übertragen und ausgewertet.

Vorteil dieser Kommunikationsart ist, dass so nicht direkt auf das Framework zugegriffen wird. Auf diese Weise werden Sicherheitsrisiken vermieden.

Ein Nutzerkonflikt²³ wird vermieden, indem er durch das Transaktionskonzept der Datenbank verhindert wird.

5.1.6 Anpassung der UML-Diagramme an die zu entwerfende Architektur

Aufgrund der festgelegten Anwendungsarchitektur müssen die zuvor definierten UML-Diagramme an die Architektur angepasst werden.

²³Zeitgleicher Zugriff auf die Maschine über HMI und Smartphone-Applikation

Nach Definition der Anwendungsarchitektur ist es sinnvoll, die in Kapitel 3 definierten UML-Diagramme entsprechend abzuändern. Zuvor wurde der Ablauf der einzelnen Systemfunktionen schematisch aufgezeichnet und textuell beschrieben. Für die Codegenerierung ist das eher unpraktisch, da an dieser Stelle der Implementierung Funktionsnamen benötigt werden. Dadurch werden die Modelle inhaltlich komprimiert, um Overhead im Generator zu vermeiden. Auf diese Weise optimiert man die Diagramme für die folgenden Arbeitsschritte.

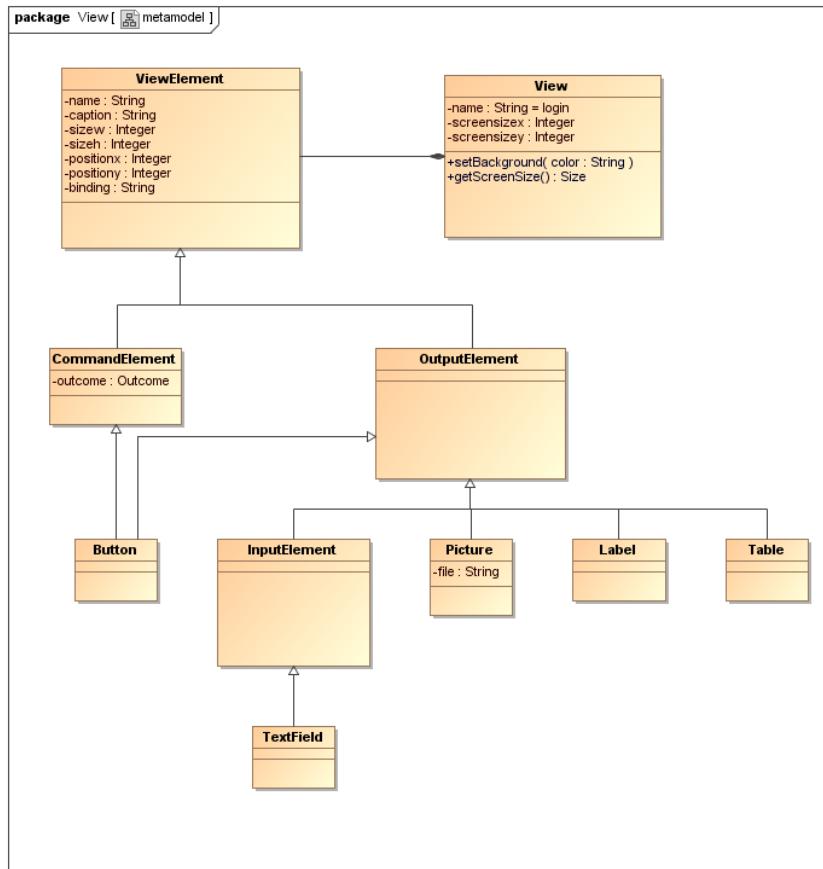


Abbildung 5.3: Metamodell View

Zusätzlich werden die Metamodelle aus Kapitel 3 überarbeitet, da während der Implementierung des Testcodes festgestellt wurde, dass nicht alle zuvor definierten Konstrukte nötig sind. Abbildung 5.3 zeigt das neue Metamodell für die View-Darstellung.

5.1.7 Codegenerierung

Nachdem alle Grundvoraussetzungen für den zu implementierenden Prototypen YASA geklärt sind, wird nun das Konzept umgesetzt.

Im Folgenden wird anhand ausgewählter UML-Diagramme erläutert, wie man lauffähigen Code für Android bzw. iOS generiert.

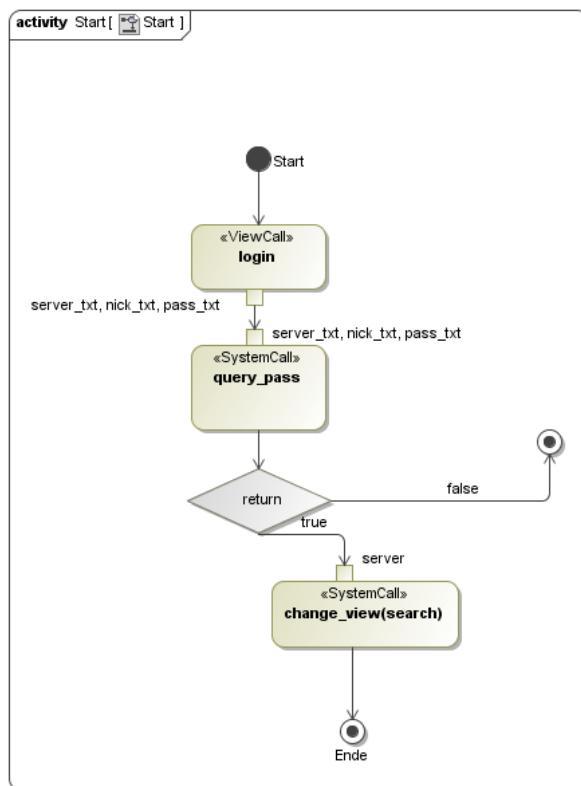


Abbildung 5.4: Start Activity

Das UML-Diagramm in Abbildung 5.4 ist der Einstiegspunkt für den Generator. Ausgehend vom Startknoten sollen die verschiedenen Operationen auf dem mobilen Endgerät der Reihe nach aufgerufen werden.

Als erstes muss also innerhalb des Generators ein Weg gefunden werden, wie der reihenfolgerichtige Ablauf des Diagramms gewährleistet werden kann. Lösen lässt sich dieses Problem mit Objektreferenzen. Acceleo verfügt über die Möglichkeit, einfache Schleifenkonstrukte zu verwenden. Dabei kann eine Schleife beginnend beim Startknoten, der aus dem Aktivitätsdiagramm ausgelesen wird, über die jeweilige Referenz zum nächsten bzw. vorherigen Element bis zum Endknoten durchlaufen werden.

Weiterhin stellte sich bei der Implementierung heraus, dass es nicht einfach ist, über die Objektreferenz von einer Aktion zur nächsten zu springen, wenn OutputPins, InputPins oder andere Elemente definiert wurden. Diese werden in Acceleo genauso wie Aktionen behandelt und führen bei der Generierung zu einer Endlosschleife, die das Programm zum Absturz bringt. Deshalb ist es erforderlich, alle Elemente herauszufiltern, die während des Schleifendurchlaufs erfasst werden und keine Aktion sind.

Am ehesten kann man den Ablauf der Schleife mit einer doppelt verketteten Liste vergleichen. Eine Aktion hat einen Vorgänger und einen Nachfolger. Diese werden über Referenzen erreicht. Beginnend beim Startknoten wird das aktuelle Element so lange auf den Nachfolger gesetzt, bis man sich im Endknoten befindet. Mit seinem Erreichen

ist der Schleifendurchlauf beendet und der Code für dieses Aktivitätsdiagramm wurde erfolgreich generiert.

Nach Definition des Elementdurchlaufs müssen die einzelnen Aktivitäten und deren Unterelemente näher betrachtet werden.

Das Konzept von YASA sieht vor, dass aus möglichst einfachen Kontrollstrukturen komplexe Sachverhalte generiert werden. Am Beispiel des Diagramms 5.4 werden nun die einzelnen Elemente untersucht. Im Detail wird erläutert, wie der Generator diese Elemente behandelt.

Nachdem der Generator den Startknoten verlassen hat, befindet er sich im ersten Aktivitätsknoten mit Handlungscharakter. Dieser besitzt den im Metamodell definierten Stereotypen **ViewCall**. Das bedeutet für den Generator, dass er aus diesem Aktivitätsknoten einen View für die Anwendung anlegen soll (vgl. Model-View-Controller-Prinzip A.4). Views befinden sich in einem separaten Unterverzeichnis. Über den Namen des Knoten wird nun im UML-Baum nach einem analogen Element im View-Verzeichnis gesucht. Wird ein entsprechendes Diagrammelement gefunden, erhält der Generator die Anweisung dieses auszuwerten.

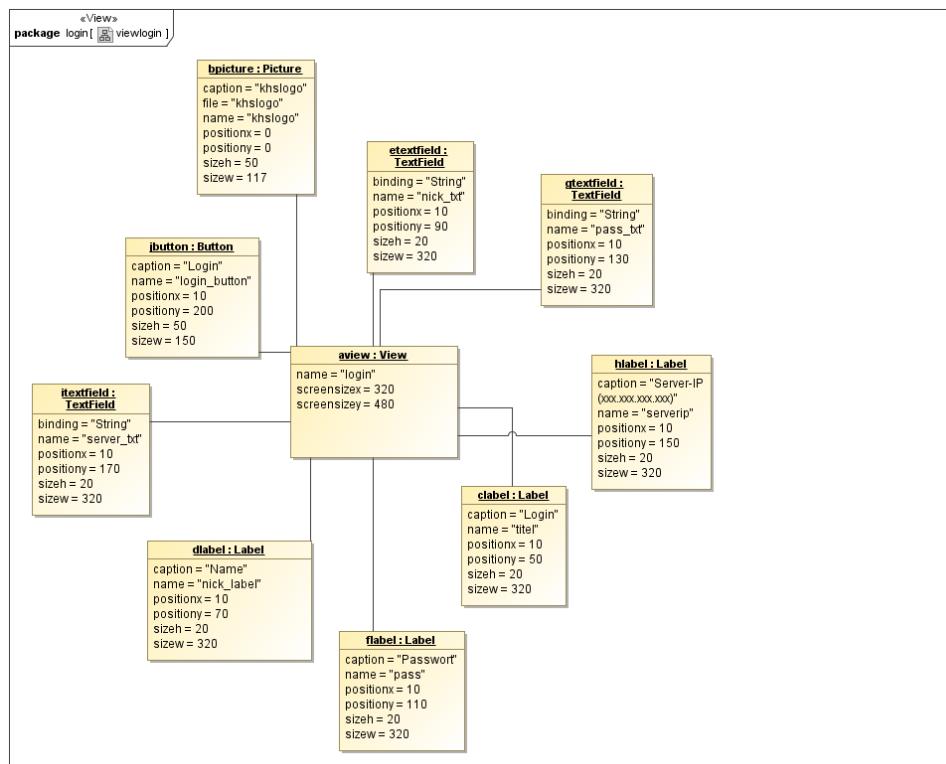


Abbildung 5.5: Login View

Ein View kann aus beliebig vielen Elementen der Metaklasse **View** bestehen. Diese Elemente müssen nun für die Darstellung auf dem Endgerät aufbereitet werden. Dafür wird die Reihenfolge der Darstellungselemente benötigt. Bei einem Klassendiagramm

ist jedoch keine Elementreihenfolge vorgesehen. Elemente werden beliebig angeordnet und besitzen Beziehungen zu anderen Elementen. Für die richtige Ausgabe müssen die Elemente allerdings eine Reihenfolge (Android Layout) oder eine absolute Positionierung (iOS) besitzen. Die Schwierigkeit besteht darin, Objektinstanzen im Klassendiagramm einer Reihenfolge nach zu ordnen. Nach fachlicher Diskussion kam man zu dem Schluss, dass das auftretende Problem für Android über eine Erweiterung des Objektnamens der instanzierten Klasse gelöst werden kann (siehe Abbildung 5.5). Eine dem Alphabet folgende Buchstabenkombination vor dem zur eindeutigen Identifizierung notwendigen Namen ordnet die Elemente der Reihe nach.²⁴

Der Generator durchläuft nun die nach dem Alphabet sortierten Objekte und erzeugt gleichnamige Elemente im View-Layout. Im Beispiel von Android handelt es sich dabei um eine XML-Datei, die unabhängig von der Implementierung des Controllers generiert wird. Jedes Objekt erhält einen eigenen Eintrag (ID) über die vom Controller aus auf das jeweilige Element zugegriffen werden kann. Weiterhin werden für jedes Objekt die spezifischen Eigenschaften der Attributdefinitionen aus dem Modell (Textueller Name, etc.) extrahiert und in den View eingebettet.

Nach Fertigstellung des Views gelangt der Generator zum nächsten Aktivitätsknoten. Dieser ist ein System Call. System Calls sind im Controller der Anwendung befindliche Systemaufrufe, die entweder direkt auf dem Client ausgeführt werden (beispielsweise den View zu wechseln) oder mit dem Server interagieren.

Systemaufrufe können Eingabeparameter enthalten. Diese werden durch Input Pins modelliert. Dadurch erfolgt eine Verknüpfung zwischen View und Controller. Der Generator erstellt für die jeweiligen Pins Methoden zum Bearbeiten der Viewelemente. Diese Methoden dienen zur Überwachung von Benutzereingaben, z.B. können Texteingaben über die ID des Elements ausgelesen werden. Die jeweilige Implementierung der Methoden hängt vom Zielsystem ab und benutzt die in der API des Systems zur Verfügung gestellten Vorgaben.

Im Generator wird zwischen Elementen unterschieden, denen Eigenschaften entnommen werden sowie Elementen, die Ereignisse auslösen. Im Beispiel werden drei Texteingaben aus dem View ausgelesen und an den Server geschickt. Dabei wird auf dem Server nach einer ausführbaren Skriptdatei gesucht, die den Namen des Systemaufrufs trägt. Auf das Aktivitätsdiagramm bezogen heißt dies, dass die Datei „query_pass.php“ auf dem Server gesucht wird und ihm die Texteingabefelder als Parameter übergeben werden. Voraussetzung dafür ist, dass die entsprechende Datei vorher auf dem Server manuell angelegt wird.

²⁴Naheliegender wäre es sicher gewesen, Zahlen zu benutzen. Allerdings ist es zum Beispiel in Java nicht möglich, Objekte mit einer führenden Zahl zu benennen. Eine andere Idee wäre es, Vorgänger- und/oder Nachfolgerattribute zu verwenden. Das würde den Generator-Code allerdings zu stark verkomplizieren.

Auslösendes Ereignis des Systemaufrufs ist die Betätigung des Buttons im View.

Im jeweiligen Controller wird von Acceleo innerhalb des ActionsListeners des Buttons ein Systemaufruf der JSON-Klasse dynamisch für die Anzahl der Übergabeparameter generiert. So kann gewährleistet werden, dass für variable Eingabeparameter in Form von Input- bzw. OutputPins dynamischer Code erzeugt wird.

System Calls, die auf Funktionen des Server zugreifen, liefern entweder true zurück, wenn die Abfrage ausgeführt werden kann oder false, wenn die Abfrage nicht verarbeitet werden kann oder der Server nicht erreichbar ist.

Ergebnisse aus einer Abfrage können über InputPins als Variable in einen anderen Controller übernommen werden. Diese InputPins werden vor einem SystemCall modelliert, der die Funktion change_view aufruft. Wichtig dabei ist, dass die Serverantwort ein Ergebnis enthält, das wie die Variable benannt ist. Andernfalls wird zwar versucht, einen Wert zu übergeben, dieser bleibt jedoch im Programm uninitialisiert und kann das generierte Programm zum Absturz bringen.

5.1.8 Durchgeführte Implementationsarbeiten zur Erstellung des Prototypen

Um einen besseren Überblick über die zahlreichen Implementationsschritte zu erhalten, werden im Folgenden alle Vorgänge aufgelistet, die für die Erstellung von YASA notwendig waren.

Zunächst wurden exemplarisch Zielplattformen für diese Arbeit ausgewählt. Dies sind Android und iOS. Um einen Generator für diese Programmiersprachen zu erstellen, mussten die Grundstrukturen der Betriebssysteme und die Programmiersprachen Objective-C und JAVA speziell in Bezug auf Smartphoneprogrammierung erlernt werden.

Darauffolgend wurde die virtuelle Maschine für den Server erstellt. Diese beinhaltet den Apache Webserver, PHP und MySQL, die eingerichtet und auf Lauffähigkeit sowie Erreichbarkeit im Netzwerk getestet wurden. Im Anschluss mussten händisch Funktionen in PHP erstellt werden, die der Client auf dem Server aufrufen kann. Auch diese wurden auf Funktionalität getestet. Dies geschah mittels eines Browsers, da sich auf diese Weise PHP-Dateien als serverseitige Skriptsprache aufrufen lassen.

Parallel dazu wurde der Generator entwickelt. Acceleo bietet zwar die Möglichkeit auf UML-Diagramme zuzugreifen, allerdings musste festgelegt werden, wie diese Diagramme benutzt werden. Für die verschiedenen Diagramme wurden Template-Systeme entwickelt, die unterschiedliche Funktionen übernehmen. Ihre Hauptaufgabe ist es, die Diagramme strukturiert einzeln abzuarbeiten und alle Informationen zu extrahieren. Mit

den erhaltenen Informationen wird der Quellcode für das Zielgerät erstellt. Weiterhin sind die Templates dafür verantwortlich, wichtige Manifestdateien (vgl. Android) zu generieren. Diese enthalten Informationen wie beispielsweise Systemeinstiegspunkt oder Beziehungen der einzeln erstellten Quellcodedateien untereinander.

Im Anschluss wurden die generierten Dateien auf Funktion getestet. Android ließ sich im Simulator ausführen. Zum hardwarenahen Funktionstest wurde ein firmeneigenes Gerät der KHS benutzt. Es handelt sich hierbei um ein Samsung Galaxy Tab. Für iOS wurde der Umweg über Toolchain gewählt. Deshalb konnte nur auf dem Endgerät auf Funktionalität getestet werden. Zur Verfügung standen hier ein iPhone sowie ein von der KHS gestelltes iPad.

Schematisch nachvollziehen lässt sich der Ablauf eines vollständigen Generiervorgangs in Abbildung 5.6.

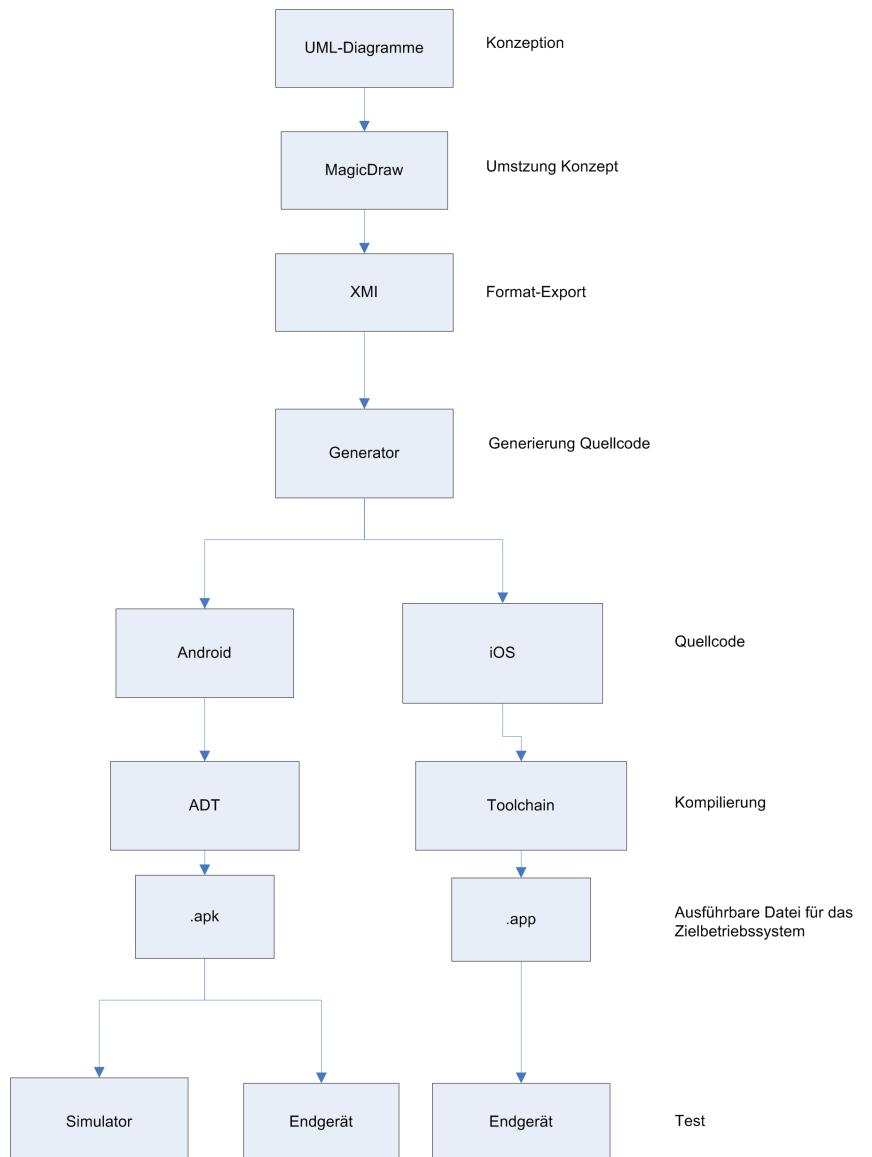


Abbildung 5.6: Workflow eines Generiervorgangs

5.1.9 Hindernisse bei der Codegenerierung

Nachfolgend wird auf erhöhte Anforderungen besonders bei der Erweiterung des Generators hinsichtlich Modell-zu-Text-Transformation eingegangen.

Um einen funktionierenden Generator erstellen zu können, ist fundiertes Wissen über die Zielsysteme notwendig. Im Prozess der Implementierung hat sich gezeigt, dass es schwierig ist, ein Konzept für verschiedene Betriebssysteme umzusetzen. Nur mit Expertenwissen für die Zielplattformen können Probleme in der Umsetzung vermieden werden. Dies ließ sich nur durch umfassende Recherche realisieren.

Als am schwierigsten stellten sich jedoch die Modifikationen des Generators heraus. Teilweise wurden Syntaxfehler der generatoreigenen Sprache erst Minuten später dargestellt, was den Entwicklungsprozess immens verlangsamt. Die ungewöhnliche Klammersyntax der Generatorbefehle stellte sich vor allem bei der Implementierung für das iOS als schwierig heraus. Denn sowohl in Objective-C als auch für den Generator werden die selben Ausdrücke sehr oft verwendet. Deshalb war es notwendig, große Teile des iOS-Quellcodes bzw. der auftretenden Klammern ([,]) zu escapen. Die Lesbarkeit des Generatorquellcodes leidet teilweise sehr darunter.

Weiterhin ist es nicht möglich, den Generator qualifiziert anhand der UML-Diagramme zu debuggen. Eine Kombination des Acceleo Debuggers mit den UML-Diagrammen wäre sehr hilfreich. Fehler müssen umständlich im Quellcode gesucht werden. Bei Syntaxfehlern bricht der Generator den Prozess ab. Entsteht während des Generierens eine Endlosschleife, stürzt die Entwicklungsumgebung ab.

Gerade die Umsetzung für das Apple Betriebssystem iOS verursachte viele Probleme. Durch das Benutzen der Toolchain (Anhang A.1.2) war es möglich, den generierten Quellcode auf dem iOS zu testen. Allerdings stand kein Simulator zur Verfügung. Folglich konnten Fehler bei Ablauf des Programms nicht richtig nachvollzogen werden (fehlendes Debugging und Logging). Während der Ausführung auf dem iPhone bzw. iPad stürzte das Gerät beim Wechsel der Views ab. Ein akribischer Test der einzelnen Views zeigte jedoch, dass diese fehlerfrei waren. Zurückzuführen ist dieser Absturz eventuell auf die veraltete Firmwareversion beim Kompilieren des Quellcodes mit der Toolchain. Allerdings kann dies nicht bewiesen werden.

Deshalb ließ sich die Entwicklung für iOS an dieser Stelle nicht mit den vorhanden Mitteln weiterführen.

5.2 Testing

Im Folgenden werden die für den Prototypen wichtigen Punkte in Bezug auf Software-Qualität erörtert.

5.2.1 Einstufung verschiedener Testverfahren

Essentieller Bestandteil eines jeden Software-Entwicklungsprozesses ist das Testing. Im Allgemeinen unterscheidet man vier verschiedene Teststufen:

- Komponententest
- Integrationstest
- Systemtest
- Abnahmetest

Dieses Kapitel konzentriert sich auf den Komponententest, der zur Verifikation der Systemkomponenten verwendet wird. Er wird auch als Unit-Test oder Modultest bezeichnet.

Basis dieses Tests ist der Komponentenentwurf. Als „Komponenten“ bezeichnet werden Funktionen oder Prozeduren eines Programms. Im vorliegenden Fall sprechen wir von Templates, die in Acceleo dazu benutzt werden, aus UML-Diagrammen Code zu generieren.

Für den Komponententest muss ein geeigneter Testrahmen definiert werden. Ein solcher Rahmen beinhaltet Testtreiber, Platzhalter und die verwendete Entwicklungsumgebung.

Der Testtreiber dient zur Ausführund des Tests und soll das Testing vorzugsweise automatisiert ablaufen lassen.

Platzhalter dienen der Isolierung zu testender Objekte. Durch den Einsatz von Platzhaltern kann das Testobjekt unabhängig von Einflüssen abhängiger Komponenten getestet werden. [Cleff10]

Als Entwicklungsumgebung wird die Eclipse IDE verwendet, da sich sowohl Generator als auch ADT Plugin ebenfalls darin ausführen lassen. Als Testtreiber für unseren Generator bietet sich demnach JUnit an.²⁵ Spezielle Platzhalter werden für die folgenden Tests nicht benötigt.

Ziel des Testens ist es, Fehlerwirkungen aufzudecken.

²⁵JUnit hat sich in der Programmiersprache Java als Standard durchgesetzt.

5.2.2 Durchführung des Testings

Getestet werden muss sowohl der Generator als auch der generierte Quellcode für die Zielplattform. Im folgenden werden die verschiedenen Herangehensweisen an das Testing erläutert.

5.2.2.1 Test des Generators

Angestrebt wird ein Unit-Testing, das die Komponenten des Generators auf Übereinstimmung von vorgegebenen Eingabewerten und zu erwartenden Resultaten prüft.

Zur Zeit befindet sich ein spezielles Testframework für Acceleo in Entwicklung. Zum Zeitpunkt der Fertigstellung dieser Arbeit war das Testframework allerdings noch nicht Verfügbar.²⁶

Folglich musste eine andere Lösung gefunden werden. Acceleo bietet die Möglichkeit, den damit entwickelten Generator als Stand-Alone-Application in Java zu exportieren. Auf dieses Weise konnten mit Hilfe von JUnit und Manipulation der exportierten Anwendung einzelne Templates des Generators getestet werden. Dazu wurde die ausführbare Generatoranwendung um eine eigens für das Testing geschriebene Methode erweitert, die es ermöglicht, manuell ein Testmodell zur Auswertung an den Generator zu schicken und ein Feedback von diesem an JUnit weiterzureichen.

Exemplarisch wird im Folgenden das Testing an einem konkreten Fallbeispiel erläutert.

5.2.2.1.1 Test der Generierung von Views

Der Generator erzeugt View-Dateien für die verschiedenen Benutzeroberflächen, die in der Androidimplementierung Verwendung finden. Ein möglicher Testfall für das Template ist die Überprüfung der generierten Dateien auf Vollzähligkeit. Bei diesem speziellen Testfall muss der innere Aufbau des Templates nicht bekannt sein, deswegen spricht man von einem sogenannten „Black-Box-Test“.

5.2.2.1.2 Testfallentwurf

Beim Testing werden ausgehend von Äquivalenzklassen verschiedene Grenzwertfälle gebildet, die getestet werden können. Äquivalenzklassen werden normalerweise anhand von Wertebereichen gebildet, die an eine zu testende Funktion übergeben werden.

²⁶Erste Ansätze finden sich auf dem Developer-Blog eines Acceleo-Entwicklers. [Begaudeau10]

Hier kommt erschwerend hinzu, dass es sich bei den Eingabewerten für den Generator um UML-Diagramme handelt.

Prinzipiell können wir also zwei verschiedene Arten von Eingabewerten bestimmen. Einerseits sind dies vollständige UML-Dateimodelle gemäß der Systemspezifikation und andererseits unvollständige Modelle, die von der Spezifikation abweichen.

Für einen exemplarischen Test werden UML-Modelle generiert. Testfall 1 (Abbildung 5.7) zeigt das Vorhandensein von drei Elementen im Views-Verzeichnis. Der Generator sollte folglich drei Viewdateien ausgeben.

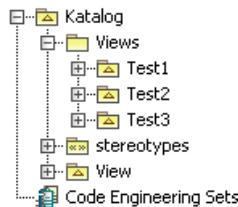


Abbildung 5.7: Baumstruktur der UML-Diagramme; Testfall 1

Testfall 2 (Abbildung 5.8) enthält neben den Metamodellen und Stereotypen keine weiteren Elemente im UML-Baum. Demnach dürfte der Generator keine Viewdateien anlegen.



Abbildung 5.8: Baumstruktur der UML-Diagramme; Testfall 2

5.2.2.1.3 Implementierung in JUnit

In dem erstellten JUnit-Test werden die UML-Modelle der Testfälle als Eingabeparameter benutzt und an den Generator weitergeleitet. Dieser verarbeitet die übergebenen Diagramme. Im Anschluss erhält JUnit Rückmeldung vom Generator, ob er den Generierervorgang vollständig durchführen konnte. Darauffolgend werden im Dateisystem bzw. dem Zielverzeichnis der Codegenerierung die erstellten View-Dateien gesucht. Je nach Ergebnis dieses Suchvorganges wird der Test als bestanden oder nicht bestanden gewertet.²⁷

²⁷Quellcode des JUnit Tests findet sich im Anhang A.7 und auf der beiliegenden DVD.

5.2.2.1.4 Testergebnisse

Bei Ausführung des ersten Testfalls wurde der JUnit-Test erwartungsgemäß bestanden.

Testfall 2 allerdings liefert ein fehlerhaftes Ergebnis. Das Zielverzeichnis weist Viewdateien auf, obwohl im zuvor gezeigten Diagramm 5.8 keine verarbeitungsrelevanten Elemente auftauchen. Da im vorliegenden Fall eine einzelne Komponente des Generators getestet wird, versucht er alle vorhandenen Elemente in der ihm übergebenen Modelldatei auszuwerten. Fälschlicherweise werden Views für das Metamodell View erstellt.²⁸

5.2.2.1.5 Ausblick

Dieser Ansatz kann für viele andere Unit-Tests ausgebaut werden. Vorstellbar ist es zum Beispiel, dass einzelne Elemente im generierten Zielquellcode mittels regulärer Ausdrücke auf Vorhandensein und Vollständigkeit überprüft werden.

5.2.2.2 Test des generierten Codes

Neben dem Generator muss auch der generierte Code aufgetestet werden. Die Generierung eines ausführbaren Programmes impliziert nicht, dass dieses die implementierten Methoden korrekt ausführt.

Erschwerend beim Test der Smartphone-Applikation ist, dass dieser nur erfolgen kann, wenn die Applikation im Simulator parallel zum Testing ausgeführt wird. Eine notwendige Verknüpfung zwischen laufender Applikation und Testsuite ist ein Mehraufwand. Allerdings ist es in Eclipse möglich, einzelne Projekte miteinander zu verknüpfen(Project dependency). So kann man über JUnit einzelne Android Activities laden, ausführen und Tests in Bezug auf einzelne Funktionen bzw. Methodenaufrufe durchführen.

5.2.3 Modellbasiertes Testen

Modellbasiertes Testen (MBT) bezeichnet die Testfallerstellung und Testauswertung mit Hilfe von Modellen. [Liggesmeyer09]

²⁸Ausgehend vom diesen Ergebnis muss im Anschluss eine Fehlerkorrektur erfolgen und ein Regressionstest durchgeführt werden.

Bekannt sind zwei verschiedene Ansätze für modellbasiertes Testen. Zum einen kann das Testen auf Basis von bereits vorhanden Systemmodellen erfolgen. Ein Verhaltens- oder Strukturmodell wird aus den Anforderungen konstruiert.

Zum anderen gibt es die Möglichkeit, Testmodelle für einen modellbasierten Test zu generieren. Hier wird neben dem Systemmodell ein weiteres Modell erzeugt. Ein Vorteil dieser Herangehensweise liegt darin, dass eventuelle Fehler im Systemmodell erkannt werden können.

Aufgrund der Tatsache, dass bei Erstellung des Prototypen YASA der modellgetriebene Ansatz zur Softwareentwicklung benutzt wurde, ist diese Art des Testings ein interessanter Ansatz. Das MBT wäre eine alternative Herangehensweise zum Testing des Prototypen gewesen. Da die komplette Anwendung aus UML-Diagrammen generiert wurde und durch den Generator keine textuelle Implementierung mehr stattfindet, wäre es sinnvoll, auch das Testing auf Basis von UML-Diagrammen zu gestalten. Auf diese Weise könnte der Testerstellungsaufwand mittels MBT erheblich verringert werden. Im Gegensatz dazu müssen mit der Unit-Testmethode für jedes Zielbetriebssystem eigene Tests in der jeweiligen Programmiersprache entwickelt werden.

Allerdings ist das modellbasierte Testen ein mehr als umfassendes Thema und sollte in einer gesonderten Arbeit ausführlich diskutiert werden. Daher findet es hier keine Anwendung.

6 Zusammenfassung und Ausblick

Im Folgenden wird die Arbeit kurz zusammengefasst und auf Probleme und mögliche Ausblicke eingegangen.

Die Erwartungshaltung für diese Diplomarbeit bestand darin, in einer vorgegebenen Zeit zu evaluieren, ob der modellgetriebene Ansatz zur Entwicklung von Smartphone-Applikationen lohnend ist oder nicht. Ausgehend vom Wissensstand zu Beginn dieser Arbeit wurde vermutet, dass die Umsetzung mittels MDA durchaus möglich ist, wenn auch nicht einfach.

Die größte Schwierigkeit im Vorfeld lag vor allem bei der Evaluierung der zu verwendenden Tools und der Spezifikation der UML-Diagramme in Zusammenhang mit der Codegenerierung.

Während der Implementierung stellten sich alle benutzten UML-Tools als ziemlich unzureichend dar. Unzulänglichkeiten im Hinblick auf Benutzbarkeit und die Integration des UML-Standards nach OMG erschweren das Arbeiten mit solchen Tools. Dabei soll UML den Entwicklungsaufwand erwartungsgemäß verkleinern und nicht Abläufe komplizieren.

Bei den verwendeten Tools zur Generierung des Quellcodes stellte sich erst während der Nutzung heraus, wie unkomfortabel sich mit ihnen die Entwicklung gestaltete. Es war nur mit erheblichem Aufwand möglich, die definierten Diagramme korrekt wiederzuverwenden.

Da der modellgetriebene Ansatz zur Softwareentwicklung schon einige Zeit genutzt wird, um Programme zu entwerfen, sollte man meinen, dass die Kinderkrankheiten dieser Generatoren mittlerweile behoben wären. Das ist nicht der Fall.

Konzepte in der Programmierung des Generators sind verbessерungsbedürftig, beispielsweise durch einen grafischen Debugger. Man sollte zur Laufzeit des Generator-tools in dem verwendeten UML-Diagramm sehen können, in welchen Elementen man sich befindet, wie diese Elemente durchlaufen werden und welche Objekteigenschaften zur Laufzeit vorhanden sind. Alleine diese Eigenschaften hätten die Entwicklungszeit des umgesetzten Prototypen drastisch verkürzen können.

Denkbar wäre diese Erweiterung des Generators auch durch ein Plugin für die Entwicklungsumgebung Eclipse, da sowohl Generator als auch Compiliervorgang des generierten Codes darin ablaufen.²⁹

Insgesamt betrachtet sind die auf dem Markt verfügbaren untersuchten Generatortools nur als unvollständig und umständlich zu bewerten. Mit Generatoren aus Diagrammen Quellcode zu generieren, ist sicher der richtige Weg. Allerdings wird vermutlich noch einige Zeit vergehen, bis dies einen größeren Nutzen bietet, als die herkömmliche textuelle Programmierung.³⁰

Im Hinblick auf die exemplarisch verwendeten Betriebssysteme sei ebenfalls ein Fazit gezogen. iOS Implementierungen werden mit Objective-C entworfen. Diese Sprache stammt aus den 1980er Jahren und beruht auf Konzepten, die im Vergleich zu anderen Programmiersprachen nur als veraltet angesehen werden können.³¹ Bei Android kann man aufsetzend auf die Programmiersprache Java effizient und strukturiert Anwendungen entwerfen.

Die fertige Applikation ist jedoch für das jeweilige Betriebssystem ähnlich komfortabel.

In dieser Arbeit galt es zu klären, ob der Einsatz von MDA Sinn macht. Und zwar einerseits in Bezug auf Praktikabilität, andererseits auf programmatische Umsetzung, Anwendung, Implementierung und Entwicklung.

Dies ist jeweils von mehreren Faktoren abhängig. Die Einführung von MDA im Unternehmen ist für alle Beteiligten eine große Umstellung. Der Herstellungsprozess von Software wird signifikant geändert. Es entsteht ein Mehraufwand, der allerdings nur in der Einführungsphase von MDA zu verbuchen ist. Bei wiederkehrenden Projektstrukturen bzw. Anwendungen mit ähnlichem Charakter wird die Implementationsphase stark verkürzt. MDA ist demnach sinnvoll, wenn nicht nur eine Anwendung mit diesem Konzept entwickelt wird, sondern Folgeanwendungen eine ähnliche Struktur und/oder bereits implementierte Konzepte verwenden.

Praktisch bewiesen wurde mit dieser Arbeit, dass es mittlerweile sehr wohl möglich ist, eine vollständig lauffähige Anwendung auf Grundlage von UML-Diagrammen zu generieren. Allerdings ist die händische Erstellung eines Programmes zur Zeit noch schneller umzusetzen, als die automatisierte Generierung (zumindest für ein Pilotprojekt).

²⁹Allerdings ist dieses Thema so umfangreich, dass es in einer gesonderten Arbeit behandelt werden sollte.

³⁰Es hätte nur wenige Tage gedauert, eine Applikation händig für die Zielplattformen fertigzustellen. Dabei wäre bei gleichem Zeitaufwand der Funktionsumfang des durch den Generator erstellten Programms weit übertroffen worden.

³¹Beispielsweise fehlt in Objective-C eine Garbage Collection. Der Programmierer muss entwicklungsirrelevante Elemente wie die Ressourcenverwaltung berücksichtigen.

Als innovatives Unternehmen ist KHS willens und auch bereit, neue Technologien zu verwenden. Die Umstellung auf den modellgetriebenen Ansatz für die Entwicklung von Smartphone-Applikationen zur Kommunikation mit Fertigungsmaschinen ist ein großer Schritt in die Zukunft der (teil)automatisierten Programmierung. Der aktuelle Stand der Technik hat gezeigt, dass es möglich ist, aus Spezifikationen lauffähige Programme zu generieren. Auch, wenn es den modellgetriebenen Ansatz schon länger gibt, kann man hier durchaus von einer Pionierleistung sprechen.

Benötigt werden Mitarbeiter mit Kenntnissen sowohl bei der Modellierung von UML-Diagrammen als auch Spezialisten für mobile Betriebssysteme. Da seit Jahren in hoch-qualifiziertes Fachpersonal investiert wird, steht dieses KHS zur Verfügung. Die (noch) unzureichende Software (Generatortools) könnte anfänglich zu Verzögerungen führen, dürfte jedoch bald den Erfordernissen angepasst werden können.

Ein besonderer Vorteil für KHS ist, dass UML als Basis für Systemspezifikationen bereits verwendet wird. Aus diesen Spezifikationen ist es durchaus möglich, lauffähige Programme zu generieren. Daher ist es zum modellgetriebenen Ansatz nur ein kleiner Schritt.

Weiterhin werden für unterschiedliche Maschinen ähnlich strukturierte Konzepte angewendet. Immer wieder finden sich gleiche Anforderungen mit nur geringen Änderungen – eine ideale Grundlage für die MDA.

Unter genannten Voraussetzungen ist für KHS die Anwendung von MDA durchaus zu empfehlen.

Smartphones werden in einigen Jahren einen ähnlichen Stellenwert haben, wie heute Mobiltelefone ohne erweiterte Funktionen. Funknetze werden leistungsfähiger, genau so wie die Hardware über immer mehr Speicher und Rechenleistung verfügen wird. Nahezu jeder Bürger wird über ein kleines Multifunktionsgerät verfügen. Dieses wird aus dem täglichen Leben nicht mehr wegzudenken sein und für die unterschiedlichsten Bereiche verwendet werden können – schnelle Verfügbarkeit jeder erdenklichen Information, ständige Verknüpfung zu (privaten)sozialen Netzwerken, die Steuerung von Maschinen im beruflichen Bereich und auch im privaten, wie der Zugriff auf die Kaffeemaschine in der heimischen Küche, und das von überall auf der Welt.

Kein Unternehmen der Zukunft wird auf die Nutzung von Smartphone-Applikationen verzichten können. Dabei werden immer leistungsfähigere Geräte immer komplexere Anwendungen ermöglichen. Fast zwingend logisch erscheint in diesem Zusammenhang der Schritt in die nächste semantische Ebene (siehe Kapitel 1.3.1).

MDA, oder allgemein die modellgetriebene Entwicklung, wird sich mit der Zeit immer weiter durchsetzen. Dabei wird sich MDA sicher nicht wie in der reinen Spezifikation durchsetzen lassen, sondern es werden praktische Erfahrungen aus der allgemeinen

modellgetriebenen Entwicklung einfließen. Generativ erstellte Artefakte sind heute nicht mehr wie vor Jahren nur Programmskelette.

Es bleibt zu hoffen, dass die Generatortools der Zukunft übersichtlicher, durchdachter und strukturierter sind. Mit diesen Grundlagen wäre der modellgetriebene Ansatz ein großer Vorteil für die effiziente Entwicklung immer leistungsfähigerer Applikationen.

Literaturverzeichnis

[Clark10] Josh Clark: Tapworthy - Designing Great iPhone Apps, 1. Auflage, O'Reilly 2010

[Broy10] Manfred Broy: CYBER-PHYSICAL SYSTEMS - Innovation durch softwareintensive eingebettete Systeme, 1. Auflage, Springer 2010

[KHS10] o.V.: Unternehmen, In: KHS GmbH, Stand: 2010, URL:
<http://www.khs.com/de/unternehmen.html> (letzter Abruf 21.11.2010)

[Gartner10] Gartner: Technology Research & Business Leader Insight, In: Gartner, Inc., Stand: November 2010, URL:
<http://www.gartner.com/it/page.jsp?id=1466313> (letzter Abruf 17.11.2010)

[CHI99] Fukumoto Masaaki: Whisper - A Wristwatch Style Wearable Handset, o.V., USA 1999

[Schmalenbach05] Jenni Schmalenbach: Das Phänomen Short Message Service (SMS) - Einführung in eine neue Kommunikationsform, 1. Auflage, Ludwig-Maximilians-Universität München 2005

[BeckerPant10] Arno Becker, Marcus Pant: Android 2 - Grundlagen und Programmierung, 2. Auflage, dpunkt.verlag 2010

[Petzold10] Charles Petzold: Programming Windows Phone 7, 1. Auflage, Microsoft Press 2010

[ITU10] o.V.: International Telecommunication Union, Stand: 15. Februar 2010, URL:
http://www.itu.int/net/pressoffice/press_releases/2010/06.aspx (letzter Abruf 30.11.2010)

[Apple10] o.V.: Apple Special Event, Stand: 20. Oktober 2010, URL:
<http://events.apple.com.edgesuite.net/1010qwoeiuryfg/event/index.html> (letzter Abruf 30.11.2010)

[AppleDev10] o.V.: Apple Developer, Stand: 2010, URL: <https://developer.apple.com> (letzter Abruf: 30.11.2010)

- [Applepro10] o.V.: Apple Developer Programs, Stand: 2009, URL:
<http://developer.apple.com/programs> (letzter Abruf: 24.11.2009)
- [Fokus09] o.V.: Fokus Report, Fachhochschule Nordwestschweiz, 250. Auflage,
Verlag: Fachhochschule Nordwestschweiz FHNW Institut für Mobile und Verteilte
Systeme
- [Heise10] o.V.: Heise Online, Stand: 20.10.2010, URL:
<http://www.heise.de/newsticker/meldung/Windows-Phone-7-startet-bei-T-Mobile-mit-Verzoegerung-1121952.html>
(letzter Abruf 30.11.2010)
- [CT10] o.V.: c't - magazin für computer technik, Heft 16, Heise Zeitschriften Verlag
GmbH & Co. KG, Stand 19.07.2010
- [CT10_2] o.V.: c't - magazin für computer technik, Heft 20, Heise Zeitschriften Verlag
GmbH & Co. KG, Stand 13.09.2010
- [ANDEV10] o.V.: Android Developers, Stand: 02.11.2010, URL:
<http://developer.android.com/guide/developing/other-ide.html> (letzter Abruf
01.12.2010)
- [AndroidSDK11] o.V.: Android Developers, Stand: 2011, URL:
<http://developer.android.com/sdk/index.html> (letzter Abruf 06.04.2011)
- [Symbian10] o.V.: Symbian Foundation, Stand: 28.11.2010, URL:
http://developer.symbian.org/wiki/Symbian_Foundation_web_sites_to_shut_down
(letzter Abruf 02.12.2010)
- [Koller10] Dr. Dirk Koller: iPhone-Apps entwickeln - Applikationen für iPhone, iPad
und iPod touch programmieren, 1. Auflage, Franzis Verlag, Poing, 2010
- [Eren06] Evren Eren, Kai-Oliver Detken: Mobile Security - Risiken Mobiler
Kommunikation und Lösungen zur Mobilen Sicherheit, 1. Auflage, Carl Hanser
Verlag, München, 2006
- [Radius10] o.V.: freeRADIUS - The world's most popular RADIUS Server, Stand:
28.09.2010, URL: <http://freeradius.org/> (letzter Abruf 07.12.2010)
- [Stahl07] Thomas Stahl, Markus Völter, Sven Efftinge, Arno Haase: Modellgetriebene
Softwareentwicklung - Techniken, Engineering, Management, 2. Auflage,
dpunkt.verlag, Heidelberg, 2007
- [Gruhn06] Volker Gruhn, Daniel Pieper, Carsten Röttgers: Effektives
Software-Engineering Mit UML2 und Eclipse, 1. Auflage, Springer Verlag, Berlin
Heidelberg, 2006

- [Eilebrecht10] Karl Eilebrecht, Gernot Starke: Patterns kompakt, 3. Auflage, Spektrum Akademischer Verlag, Heidelberg, 2010
- [Spiegel11] Holger Dambeck: Amazon-Rechner helfen beim Passwort-Knacken, Stand: 15.01.2011, URL:
<http://www.spiegel.de/netzwelt/web/0,1518,739716,00.html> (letzter Abruf 04.04.2011)
- [Dudek08] Denise Dudek, Christoph Sorge, Stephan Krause, Lars Völker, Martina Zitterbart: Netzsicherheit und Hackerabwehr, TeleMatics Technical Reports, Universität Karlsruhe, Stand: 2008, URL:
<http://www.tm.uka.de/doc/tr/TM-2008-3.pdf> (letzter Abruf 04.04.2011)
- [Microsoft10] o.V.: Microsoft Security Intelligence Report v9, Stand: 30.06.2010, URL:
<http://www.microsoft.com/security/sir/> (letzter Abruf 04.04.2011)
- [Cleff10] Thorsten Cleff: Basiswissen Testen von Software, 1. Auflage, W3L GmbH, 2010
- [Liggesmeyer09] Software-Qualität - Testen, Analysieren und Verifizieren von Software: Peter Liggesmeyer, 2. Auflage, Spektrum Akademischer Verlag, Heidelberg, 2009
- [Begaudeau10] Generating Blog Posts: Stephane Begaudeau, Stand 20.12.2010
URL: <http://stephanebegaudeau.tumblr.com/post/2379262223/having-fun-with-junit-and-acceleo>
(letzter Abruf 07.04.2011)
- [Zdziarski08] Jonathan Zdziarski: iPhone Open Application Development, 1. Auflage, O'REILLY Media, 2008
- [Toolchain09] o.V.: iphonedevonlinux - Building the Toolchain, Stand 21.09.2009,
URL: <http://code.google.com/p/iphonedevonlinux/wiki/Installation> (letzter Abruf 07.04.2011)
- [Blog11] o.V.: DEV-TEAM BLOG, Stand 2011, URL: blog.iphone-dev.org (letzter Abruf 07.04.2011)
- [Safaribreak10] o.V.: Erneut iPhone-Jailbreak via Safari [Update], Stand 02.08.2010
URL: <http://www.heise.de/mobil/meldung/Erneut-iPhone-Jailbreak-via-Safari-Update-1049024.html>
(letzter Abruf 07.04.2011)
- [Saurik11] Jay Freeman: The Realm of Avatar, Stand 2011, URL: www.saurik.com
(letzter Abruf 07.04.2011)

[Gov11] o.V.: Rulemaking on Anticircumvention, Stand 2006, URL:
<http://www.copyright.gov/1201/> (letzter Abruf 07.04.2011)

[ApplePress05] Natalie Kerris, Steve Dowling, Tom Beerman: Apple to Use Intel Microprocessor Beginning in 2006, Stand 06.07.2005, URL:
<http://www.apple.com/pr/library/2005/jun/06intel.html> (letzter Abruf 07.04.2011)

[BGB02] o.V.: Inhaltskontrolle, Stand 01.01.2002, URL:
<http://dejure.org/gesetze/BGB/307.html> (letzter Abruf 07.04.2011)

Quellenverzeichnis für Abbildungen

Alle in dieser Arbeit verwendeten Grafiken, die nicht selbst erstellt wurden, unterliegen der Creative Commons Lizenz.³²

- Abbildung 2.1 iPhone 4, <http://www.3gstore.de/> (letzter Abruf 28.03.2011)
- Abbildung 2.2 HTC Desire HD, <http://www.flickr.com/photos/3bilder/5113955952/> (letzter Abruf 04.04.2011)
- Abbildung 2.3 Concept Pen Phone, <http://www.technology.am/50-concept-mobile-phones-073622.html> (letzter Abruf 07.04.2011)
- Abbildung 2.4 BenQ Snake Phone, <http://www.technology.am/50-concept-mobile-phones-073622.html> (letzter Abruf 07.04.2011)
- Abbildung 2.5 Finger Whisper, <http://www.yourworldtoday.ca/2007/12/21/iphone-vs-bone-phone-which-will-sell-in-japan/> (letzter Abruf 07.04.2011)
- Abbildung 4.2 Android-Systemarchitektur, <http://developer.android.com/images/system-architecture.jpg> (letzter Abruf 11.04.2011)

³²<http://creativecommons.org/licenses/by/3.0/de/>

A Anhang

A.1 Einrichtung der Entwicklungsumgebungen

Im folgenden werden die verwendeten Entwicklungsumgebungen hinsichtlich Installation und Nutzung näher beschrieben. Soweit möglich wurden diese in Eclipse eingebunden.

A.1.1 Android

Um für Android Software entwickeln zu können, wird als erstes das Android SDK benötigt. Dieses lässt sich für verschiedene Betriebssysteme von [AndroidSDK11] herunterladen. Empfehlenswert ist die Verwendung des Eclipse ADT Plugin zur Entwicklung. Dieses lässt sich über das Update-Feature von Eclipse aus komfortabel installieren.

Sind beide Tools eingerichtet, muss noch der Pfad des Android SDK in den Eclipse-Einstellungen eingetragen werden. Im Anschluss lässt sich durch den Eclipse Project-Wizard ein neues Android Projekt erzeugen, welches im mitgelieferten Simulator sofort getestet werden kann.

A.1.2 iOS

Die Installation der Toolchain ist theoretisch unter Windows und Linux möglich. Mehrere Versuche der Installation unter Windows mittels Cygwin schlugen fehl. Deswegen wurde in einer virtuellen Maschine mit Ubuntu gearbeitet.

Zur erfolgreichen Installation sind mehrere Schritte notwendig. Zuerst wird die aktuelle Toolchain-Version heruntergeladen³³. Des Weiteren wird eine iOS Firmware benötigt. Diese lässt sich aus dem iPhone SDK von Apple extrahieren.³⁴ Ein Skript der Toolchain sorgt für das korrekte Einbinden des iOS-Frameworks in das Betriebssystem.³⁵

³³svn checkout <http://iphonedevonlinux.googlecode.com/svn/trunk>

³⁴Im Moment wird nur eine Firmwareversion bis 3.0 des iOS unterstützt. Diese ist nicht mehr auf der offiziellen Apple-Internetpräsenz erhältlich.

³⁵Eine detaillierte Beschreibung dieses Vorgehens findet sich in [Toolchain09].

Um nun eine für das iOS-Gerät geschriebene Applikation zu kompilieren, wird der Quellcode mit der Toolchain in ein Binary umgewandelt. Mit dem mitgelieferten Makefile erfolgt dies automatisiert.

Anschließend muss die Anwendung mittels SSH auf das Zielgerät in das Application-Verzeichnis kopiert werden. Bevor man nun die Anwendung auf dem Gerät testen kann, muss diese noch signiert werden. Mittels des Befehls „ldid“ angewandt auf dem Endgerät, wird eine Apple-Signierung vorgetäuscht. Der letzte Schritt ist das Neustarten der grafischen Anzeige auf dem iOS-Gerät. Danach sollte die selbstgeschriebene Applikation auf dem Homescreen erscheinen und kann ausgeführt werden.

A.1.3 Acceleo

Die Installation der aktuellen Acceleo-Version findet direkt in Eclipse statt. Über das Menü zum Hinzufügen neuer Modellierungskomponenten kann das Acceleo-Plugin für Download und Integration in Eclipse ausgewählt werden. Nach der erfolgten Installation erstellt man über den Project-Wizard ein neues Acceleo-Projekt.

A.2 Jailbreak

iOS, Apples Betriebssystem für mobile Endgeräte, basiert auf einer „change root“ Umgebung. Alle Anwendungen, die nicht Teil des Betriebssystems selbst sind, werden dadurch in einer Sandbox (vom Rest des Systems abgeschirmte Umgebung mit eingeschränkten Rechten) ausgeführt. Diese Art von Sandbox wird in der Unix-Community auch „chroot jail“ genannt. Die Bezeichnung Jailbreak steht daher wortwörtlich für das „Ausbrechen“ aus eben jener Umgebung und der Erlangung eines Vollzugriffs auf das Dateisystem des iOS-Gerätes. [Zdziarski08], S. 1

Abhängig von der Betriebssystemversion des Gerätes gibt es unterschiedliche Wege, einen Jailbreak durchzuführen. Im Regelfall wird ein Jailbreak mit Hilfe eines Computers und einem entsprechendem Jailbreak-Tool ausgeführt. Der bekannteste Jailbreak wird vom „Chronik Dev Team“ zur Verfügung gestellt, nennt sich PwnageTool und ist für das Mac OS erhältlich. Windows-Nutzer verwenden das Programm „redsn0w“ vom selben Entwicklerteam. [Blog11] Einen anderen Weg ist der Jailbreak über die Website Jailbreakme.com. Er sorgte für Aufsehen (vgl. [Safaribreak10]), da er auch einen Eingang für potentiellen Schadcode aufzeigt. Muss man beim Jailbreak über das PwnageTool oder redsn0w das iOS-Gerät über ein Dock-Connector Kabel an den PC anschließen, benötigt der Jailbreakme.com-Jailbreak lediglich den Aufruf der Website und die dortige Betätigung eines Slider-Elements. Hierbei wird eine Sicherheitslücke in der

PDF-Anzeige des mobilen Safari Browsers ausgenutzt, um eigenen Programmcode auszuführen und den Jailbreak zu vollziehen.

Hat man den Jailbreak durchgeführt, so befindet sich im Anschluss die Anwendung Cydia auf dem Homescreen wieder. Cydia ist eine freie Appstore-Alternative, welche von Jay Freeman entwickelt wurde. Cydia bietet eine grafische Benutzeroberfläche für APT, das DPKG Paket Management System. Dieses verwaltet alle installierten Anwendungen, Modifikationen, Themes und deren Abhängigkeiten. Nutzer können sich zwischen drei Modi in Cydia bewegen – Anwender-, Hacker- und Entwicklermodus. Je nach Modus wird das Anwendungsverzeichnis mit einem Filter belegt, sodass nur die für den Anwender geeigneten Anwendungen aufgelistet werden. Die Anwendungen selbst bezieht Cydia aus Repositories – Quellen im Internet. Vorinstalliert sind zunächst die vertrauenswürdigen Quellen der Cydia-Community. Bei Bedarf kann der Nutzer jedoch eigene Quellen hinzufügen.

Seit März 2008 stellt Cydia ein Bezahlinterface zur Verfügung, wodurch es Entwicklern ermöglicht wird, ihre Programme und Themes in Cydia zu verkaufen. [Saurik11]

Für Entwickler ist der Jailbreak besonders dann interessant, wenn sie für Entwicklung und Testing nicht auf Xcode und damit dem iOS-SDK, welches ausschließlich für Mac OS X bereitgestellt wird, zurückgreifen wollen oder können. So wird zum Beispiel die Entwicklung mit der Toolchain ermöglicht.

Ein Jailbreak birgt bei allen Freiheiten jedoch auch Angriffsfläche für Schadsoftware, da Cydia keiner Qualitätskontrolle unterliegt und der Nutzer selbst entscheiden muss, welche Software er auf sein Gerät herunterlädt.

In den Vereinigten Staaten ist der Jailbreak seit Mitte 2010 legal (vgl. [Gov11]), während man sich in Deutschland rechtlich in einer Grauzone bewegt. Strafrechtlich relevant ist ein Jailbreak nicht. Er verstößt jedoch gegen die Endnutzerlizenzenbestimmungen von Apple, wodurch ein manipuliertes iOS-Gerät seine Garantie verliert.

Letztendlich kann das iOS-Gerät jedoch jederzeit in den Ausgangszustand zurückgesetzt werden, indem es mittels iTunes unter Verwendung der offiziellen Firmware wiederhergestellt wird.

A.3 Hackintosh

Unter einem „Hackintosh“ versteht man einen Computer, welcher das Apple Mac OS X Betriebssystem auf einer nicht von Apple stammenden Hardware verwendet.

Im Jahr 2005 kündigte Apple auf der Worldwide Developer Conference (WWDC) in San Francisco an, ab 2006 mit seinen Computern von IBM PowerPC-Prozessoren auf Intel x86-Prozessoren umzusteigen. Bereits im Jahr 2007 wurden alle Apple Computer mit der Intel-Technik betrieben. [ApplePress05]

Eine Anleitung für eine verbreitete Hackintosh-Variante stellt das OSx86-Projekt zur Verfügung. In den Supportforen des Projekts finden sich Tutorials und Know-How für eine Vielzahl von Hardwarezusammenstellungen.

Durch eine Hackintosh-Installation können Entwickler ohne Apple-Hardware die von Apple bereitgestellten Entwickler-Tools verwenden und somit Applikationen für iOS-Geräte entwickeln.

In den USA ist die Installation von Mac OS X auf Nicht-Apple-Hardware laut dem Digital Millennium Copyright Act (DMCA) strafbar. Lizenzen, die dem Käufer vor dem Kauf einer Software nicht bekannt sind, sind in Deutschland nicht bindend. Daher darf in Deutschland eine legal erworbene Mac-OS-X-Version zur Installation eines Hackintosh verwendet werden. [BGB02]

A.4 Model-View-Controller-Prinzip

Das Model-View-Controller-Prinzip beschreibt ein Software-Entwurfsmuster, welches die Programmarchitektur in drei Teillogiken (Model, View & Controller) mit unabhängigen Verantwortlichkeiten aufteilt.

Modelle verantworten die Datenlogiken, lesen oder schreiben Daten und geben diese zurück an einen Controller. Views sind für die visuellen Elemente, die Darstellung der Benutzeroberfläche zuständig. Controller beinhalten die Implementierung der Geschäftslogiken und sind Schnittstelle zwischen Darstellung (Views) und Daten (Model).

Wesentlicher Vorteil des MVC-Prinzips ist, dass Model, View und Controller unabhängig voneinander geändert werden können. Des Weiteren ist es möglich, mehrere verschiedene Benutzeroberflächen für die gleiche Anwendung einzubinden. Anpassungen und Fehlersuche werden erleichtert, da der Entwickler aufgrund der Aufspaltung in die drei Einheiten die entsprechenden Quellcodeabschnitte schneller identifizieren kann. [Eilebrecht10], S. 77ff

A.5 Einrichtung Server als Debian Etch-VM

Ausgehend von einer virtuellen Maschine mit einem vorinstallierten Debian Etch sind folgende Schritte notwendig, um einen Server einzurichten.

Über die Paketverwaltung werden der Reihenfolge nach die Pakete „apache2“, „PHP5“ und „MySQL5“ installiert. Für die Ausspielung der Serverdateien wird ein virtueller Host in der Konfiguration des Apache Webservers angelegt. Weiterhin wird zum Verwalten der Datenbankeinträge phpmyadmin auf dem Server installiert. Mittels „phpMyAdmin“ werden Einträge in der Datenbank erstellt. Im Anschluss werden die zur serverseitigen Verarbeitung wichtigen PHP-Dateien erstellt. Aufbau dieser ist:

- Connector zur Datenbank erzeugen
- HTTP-GET zum Auslesen der Variablen
- Erzeugung einer Datenbank-SELECT-Anweisung mit den Variablen
- Ergebnis aus der Datenbank auslesen
- Ergebnis umwandeln in ein JSON-Array
- Rückgabe des Ergebnisses als String

A.6 Quellcodebeispiel View-Generierung

```

1 [comment encoding = UTF-8 /]
2 [module generate_views_android('http://www.eclipse.org/uml2/3.0.0/UML')/]

4 [template public generate_views_android(a : Model)]
5 [comment @main /]
6 [comment if ((a.name.toString()).strstr('view'))/]
7 [comment query public positionx (instanceSpecification :
    InstanceSpecification) : String = 'positionx' /]
8 [file ('res/layout/').concat(a.getModel().name.concat('.xml')), false, 'UTF-8'
    )]
9 <?xml version="1.0" encoding="utf-8"?>
10 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
11     android:orientation="vertical"
12     android:layout_width="fill_parent"
13     android:layout_height="fill_parent">
14 [for (c: InstanceSpecification | a.eAllContents()>asSequence()>filter(
    InstanceSpecification)>sortedBy(name))]
```

```

15 [comment c.name.toString()/>
16 [for (cf: Classifier | c.classifier)]
17 [if (cf.name.toString().strstr('Picture'))]
18     <ImageView
19     [for (att: Slot | c.slot)]
20     [if att.definingFeature.name.strstr('file')]
21     [comment new picture /]
22         android:layout_width="wrap_content"
23         android:layout_height="wrap_content"
24         android:src="@drawable/[att.value.stringValue()]/"
25     [elseif att.definingFeature.name.strstr('name')]
26         android:id="@+id/[att.value.stringValue()]/"
27     [/if]
28 [/for]
29     />
30 [elseif (cf.name.toString().strstr('Label'))]
31     <TextView
32     [for (att: Slot | c.slot)]
33     [comment att.definingFeature.name/]
34     [if att.definingFeature.name.strstr('caption')]
35     [comment att.value.stringValue()/>
36     [comment new label /]
37         android:layout_width="fill_parent"
38         android:layout_height="wrap_content"
39         android:text="[att.value.stringValue()]/"
40     [elseif att.definingFeature.name.strstr('name')]
41         android:id="@+id/[att.value.stringValue()]/"
42     [/if]
43 [/for]
44     />
45 [elseif (cf.name.toString().strstr('TextField'))]
46     <EditText
47     [for (att: Slot | c.slot)]
48     [if att.definingFeature.name.strstr('binding')]
49     [comment new textfield /]
50         android:layout_width="fill_parent"
51         android:layout_height="wrap_content"
52         android:inputType="textCapWords"
53     [elseif att.definingFeature.name.strstr('name')]
54         android:id="@+id/[att.value.stringValue()]/"
55     [/if]
56 [/for]

```

```

57      />
58      [elseif (cf.name.toString().strstr('Button'))]
59          <Button
60          [for (att: Slot | c.slot)]
61              [if att.definingFeature.name.strstr('caption')]
62                  [comment new button /]
63                      android:layout_width="fill_parent"
64                      android:layout_height="wrap_content"
65                      android:text="[att.value.stringValue()/]"
66                      [elseif att.definingFeature.name.strstr('outcome')]
67                          android:onClick="[att.value.stringValue()/]"
68                      [elseif att.definingFeature.name.strstr('name')]
69                          android:id="@+id/[att.value.stringValue()/]"
70                      [/if]
71          [/for]
72      />
73      [else]
74          [comment // Viewklasse /]
75      [/if]
76  [/for]
77  [/for]
78 </LinearLayout>
79 [/file]
80 [comment if/]
81 [/template]

```

A.7 JUnit Tests – Quellcode

```

1 import java.io.File;
2 import junit.framework.TestCase;
3 import junit.framework.TestSuite;
4 import junit.textui.TestRunner;
5 import org.eclipse.acceleo.module.sample_.files.Generate_views_android;

7 public class Testing extends TestCase {

9     public Testing(String name) {
10         super(name);
11     }

13     public static void main(String[] args) {
14         TestRunner.runAndWait(new TestSuite(Testing.class));

```

```

15 }

17 /**
18  * Test des Uml Diagramms , ob Dateien für den View erzeugt werden */
19 public void testViewsCreatedTest1() throws Exception {
20     Generate_views_android gen = new Generate_views_android();
21     String args[] = new String[2];
22     args[0] = ("c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testuml
23             /Test1/Test1.uml");
24     args[1] = ("c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testing
25             ");
26     Boolean genresult = gen.myTest(args);
27     if (!genresult) {
28         System.out.println("Generatorfehler!");
29     }
30 }

31 File file1 = new File(
32     "c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testing/res/
33             layout/",
34     "Test1.xml");
35 File file2 = new File(
36     "c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testing/res/
37             layout/",
38     "Test2.xml");
39 File file3 = new File(
40     "c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testing/res/
41             layout/",
42     "Test3.xml");

43 assertEquals(true, genresult && file1.exists() && file2.exists()
44             && file3.exists());
45 }

46 /**
47  * Fehlerhaftes UML wird übergeben */
48 public void testViewsCreatedTest2() throws Exception {
49     Generate_views_android gen = new Generate_views_android();
50     String args[] = new String[2];
51     args[0] = ("c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testuml
52             /Test2/Test2.uml");
53     args[1] = ("c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testing
54             ");
55     Boolean genresult = gen.myTest(args);
56     if (!genresult) {
57         System.out.println("Generatorfehler!");
58     }
59 }

60 
```

```
50     System.out.println("Generatorfehler!");
51 }
52 Boolean directoryempty = null;

54 File file = new File(
55     "c:/fh/khsworkspace/org.eclipse.acceleo.module.sample_/testing/res/
56     layout/");
57 if (file.isDirectory()) {
58     String[] files = file.list();
59     if (files.length > 0) {
60         System.out.println("Verzeichnis: " + file.getPath()
61             + " ist nicht leer!");
62         directoryempty = false;
63     } else {
64         directoryempty = true;
65     }
66 }

67 assertEquals(true, genresult && directoryempty);
68 }
69 }
```

B beigelegte DVD

Inhalt:

- Diplomarbeit als PDF
- Prototyp im Quelltext-Format
- JUnit Tests
- UML-Diagramme im MagicDraw-Format inklusive Screenshots
- Prototyp-Server als virtuelle Maschine (root-Passwort: thoughtpolice)
- Screenshots der eingesetzten Werkzeuge

Ehrenwörtliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, den 11. April 2011

Georg Wolf

