Frankfurt University of Applied Sciences
- Fachbereich 2 Informatik - mobile Anwendungen -

Dokumentation Human Machine Interface:

Android Projekt:

Grundlagen, Elementare Bausteine und Spiele

Projekt Dokumentation zur Erlangung des Moduls 23 Human Machine Interfaces (HMI)

vorgelegt von

Marcel Klamm

E-Mail: Marcel.Klamm@googlemail.com

5. Fachsemester

Matrikelnummer: 1055627

Frankfurt University of Applied Sciences, Frankfurt a.M. WiSe 2015/2016

Referent: Prof. Dr. Matthias Deegener

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Kapitel 1

Einleitung

In dieser Dokumentation soll anhand einer Android Applikation die Kommunikation und Interaktion zwischen Mensch und Maschine, auch *"Human Machine Interface"* genannt, aufgezeigt werden. Insbesondere werden die Problematiken der Kommunikation zwischen Mensch und Maschine verdeutlicht und auf die sich daraus ergebenen Konsequenzen eingegangen. Zum Beispiel wird die Frage aufgeworfen, welche Folgen sich aus einer nicht funktionierenden Kommunikation ergeben, wie diese Probleme mittels bestimmten Methoden und Techniken in Android vermieden werden können und letztlich sogar zu einer Optimierung beitragen
Für das Projekt wurde ein YouTube[1] Video erstellt, welches die Inhalte und Codeabschnitte erläutert sowie die App zeigt. Zusätzlich wurde das Projekt auf GitHub[2] veröffentlicht. Das Projekt umfasst insgesamt 9072 Code-Zeilen verteilt auf 135 Quellcode Dateien. Für die Java-Quelltexte fallen darunter 5385 Zeilen Code auf 60 Dateien an und für die Ressourcen sind es 3687 Zeilen Code auf 75 Dateien verteilt, siehe Abbildung 1.1 und 1.2. Programmiert wurde mit Googles Android Studio[3] in der aktuellen stabilen Version 1.5 mit dem Software Development Kit in Version 23.1.0. Während der Entwicklung wurde die App auf einem Virtuellen Google Nexus 5 mit API 23 Android 6.0 sowie einem Samsung Galaxy Note 3 API 21 Android 5.0 getestet. Die Dokumentation wurde mit LaTeX erstellt, welches ich mit diesem Projekt zusätzlich erlernen wollte.

[1] Fussnote

[2] `https://github.com/fujay/MyApplication`

[3] `http://developer.android.com/tools/studio/index.html`

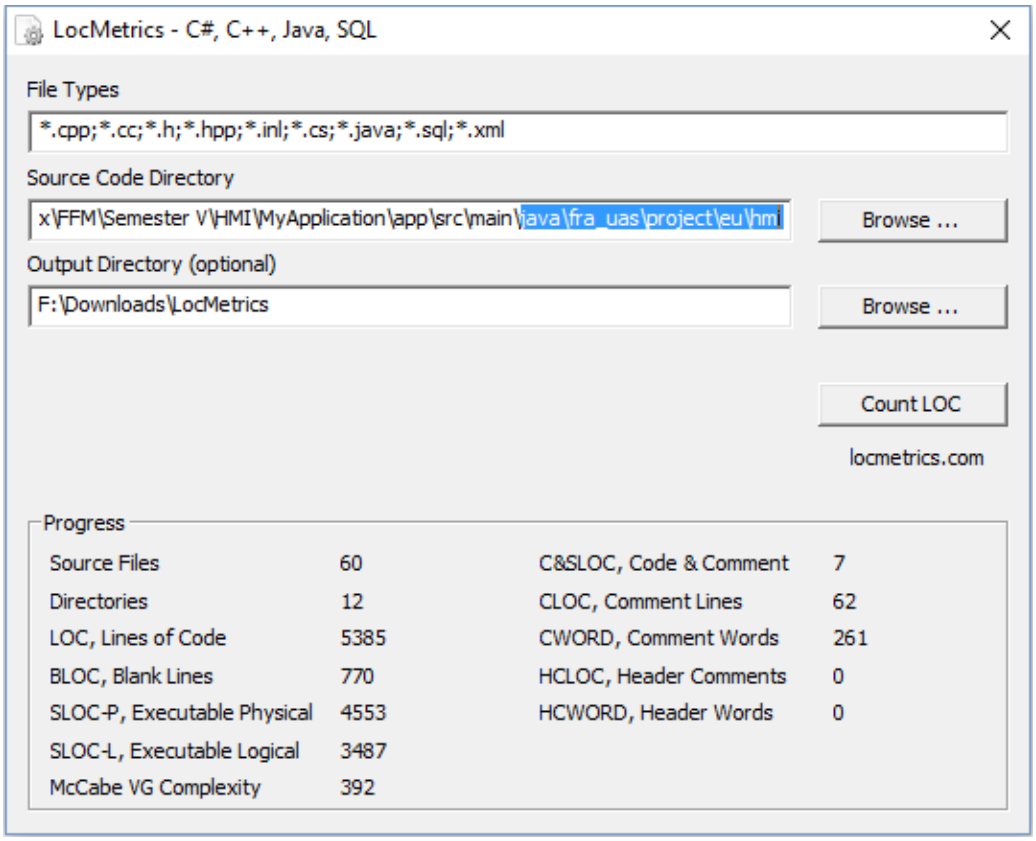

Abbildung 1.1: Statistik der Java-Quelltexte; Programm: LocMetrics, locmetrics.com

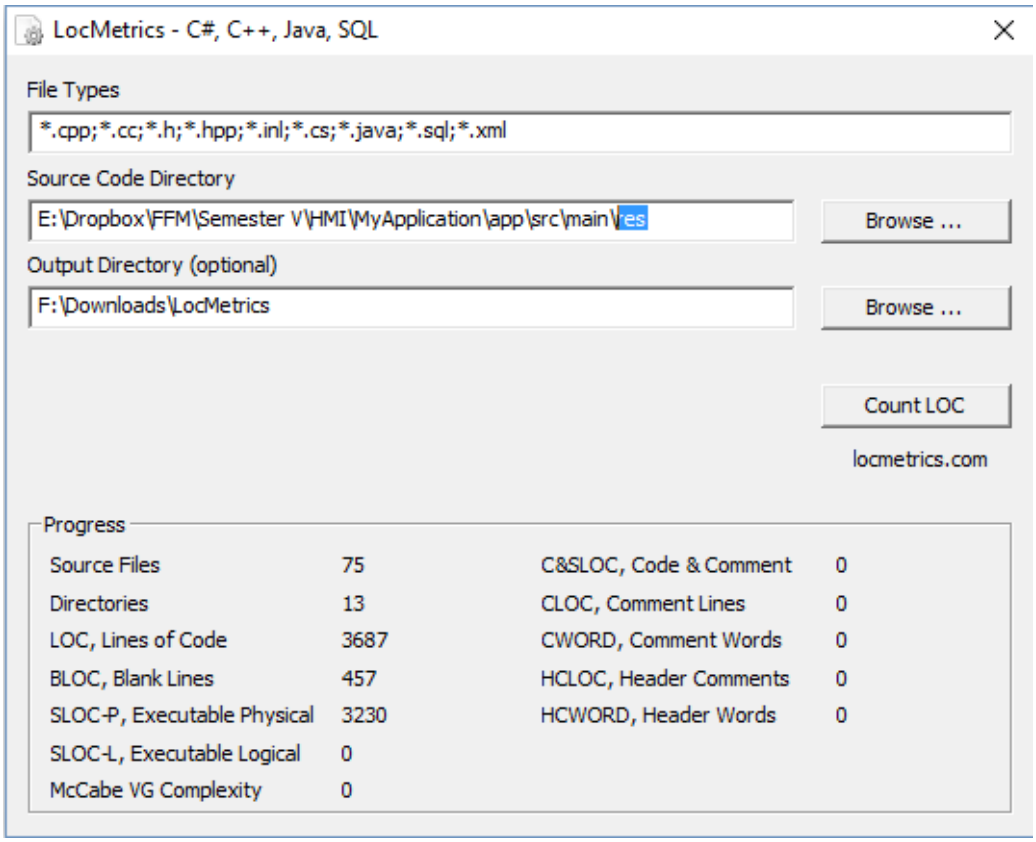

Abbildung 1.2: Statistik der Ressourcen-Quelltexte; Programm: LocMetrics, locmetrics.com

Kapitel 2

Einführung

Das Android System boomt und das liegt nicht zuletzt daran, dass Google es den Entwicklern leichtmacht: Programmiert wird mit der weitverbreiteten Sprache Java, Projektstruktur und Ressourcen werden in menschlich lesbaren XML-Formaten geschrieben, es stehen kostenlose und leistungsfähige Entwicklungswerkzeuge zur Verfügung, die Zugangshürde ist somit niedrig. Dank der mobilen und offenen Plattform, wird sich das System noch weiterverbreiten und auf weitere Technische mobile bzw. Embedded Geräte finden lassen.

Kapitel 3

Das Human Machine Interface

Die Benutzerschnittstelle von Android erfolgt dort, wo ein Dialog zwischen Mensch und Maschine erfolgt, der eine Interaktion bewirkt. Eine solche Schnittstelle bestimmt die Art und Weise wie Mensch und Maschine unmittelbar miteinander kommunizieren, wie der Mensch seine Anweisungen an die Maschine übermittelt und in welcher Form diese Anweisung ausgeführt und die Ergebnisse ausgegeben werden.
In den Anfangsjahren der Computertechnik waren Benutzereingaben textorientiert. Der Benutzer musste spezielle Kommandos eingeben die dann vom Betriebssystem interpretiert wurden. Die Schnittstelle nannte sich *Character User Interface* (CUI).
Solche Eingaben waren sehr aufwendig, fehleranfällig und nicht intuitiv. Es entwickelte sich daraus die visualisierte Schnittstelle, bei der die Eingaben manuell über Maus und Tastatur oder Touch- und Multitouchscreens ausgeführt werden. Eine solch grafische Benutzerschnittstelle wurde *Graphical User Interface* (GUI) genannt.
Eine Weiterentwicklung, die *Voice User Interface* (VUI) arbeitet mit Spracherkennung und Sprachsteuerung sowie einer Sprachausgabe.
Der GUI und VUI kombiniert mit Gestenerkennung entwickelten sich zum *Natural User Interface* (NUI). Geboten wird eine Zweifingerbewegung für die Ausführung von Befehlen auf einem Touchscreen.
Eine Weiterentwicklung dreht sich um Barrierefreiheit für Eingeschränkte Benutzer, *Bain Computer Interface* (BCI). Eine solche Benutzerschnittstelle soll Sprach-, Gesten- und auch Gedankensteuerung besitzen. Seit Version 4.0 besitzt das Betriebssystem einen hohen Grad an Barrierefreiheit[1]. Dazu gehören eine Sprachausgabe (Screenreader) für Bildschirminhalte, Bildschirmvergrößerung sowie Unterstützung für Braille Zeilen[2] und vieles Mehr. Eine Übersicht über Bedienungshilfen für Android gibt folgendes Dokument[3].
Android bietet eine Vielzahl an Benutzerschnittstellen zur Eingabe und Kommunikatun mit dem System. Dazu bietet es Unterstützung für unterschiedliche Bildschirmgrößen, beispielweise die einer Smartwatch bis hin zu großen Tablets und Bedienungshilfen für eingeschränkte

[1] `http://www.marlem-software.de/marlemblog/tag/android/`
[2] `https://support.google.com/accessibility/android/answer/3535226?hl=de`
[3] `https://support.google.com/accessibility/android/answer/6006564?hl=de&ref_topic=6007234`

Nutzer. Dank diesen Schnittstellen ist es Möglich, viele Benutzer und Geräte zusammenzuführen.

Kapitel 4

Android - eine offene und mobile Plattform

Android ist sowohl ein Betriebssystem als auch eine Software-Plattform, welche 2007 von Google veröffentlicht wurde. Die Interessen und Neugier der Entwickler und Hersteller an einer offenen Plattform für Embedded Systems war sehr groß. Android läuft mittlerweile auf einer Vielzahl verschiedener mobile Gerätetypen wie Smartphones, Auto-Infotainment, Mediaplayer, Netbooks, Tablet-Computer und Smartwatches. Dank zahlloser Apps für Android lässt sich die Funktionsvielfalt der Geräte zusätzlich erweitern und an die eigenen Bedürfnisse anpassen. Dank der engen Integration von Googles Diensten, den zahlreichen Sensoren und der Offenen Plattform sowie der leichte Einstieg in die Programmierung dank Java als Programmiersprache und Android Studio als Entwicklungsumgebung, machten Android so populär. Entwickelt wird Android von der Open Handset Alliance, ein Konsortium von Netzbetreiber, Softwareentwickler, Halbleiter- und Mobiltelefonhersteller welche mittlerweile 84 Unternehmen umfasst und von Google gegründet wurde. Die aus der Pressemitteilung vom 5.November 2007[1] formulierten Ziele waren unter anderem:

- Eine deutliche Verbesserung der Benutzerfreundlichkeit und des Benutzererlebnisses von mobilen Geräten und Diensten
- Die kostengünstigere Entwicklung und Verteilung innovativer Produkte
- Eine schnellere Markteinführung

Dies ließ sich am besten durch eine auf Offenheit und Flexibilität gegründete Zusammenarbeit erreichen. Deshalb wurden praktisch alle Teile des Android-Softwaresystems als Open Source veröffentlicht. Als Smartphone-Betriebssystem besaß Android im zweiten Quartal 2014 einen weltweiten Marktanteil von 84,6 Prozent[2]. Wenn man bedenkt, dass Android im dritten Quartal

[1] `http://www.openhandsetalliance.com/press_110507.html`

[2] `http://www.golem.de/news/mobile-betriebssysteme-android-laeuft-auf-fast-85-prozent-aller-smartphone`
`html`

2010 einen weltweiten Marktanteil von 25,5 Prozent[3] betrug, ist das eine beachtliche Steigerung. Führt man den Gedankensprung weiter, lässt sich erahnen, dass die Android Plattform auch weiter auf anderen Plattformen ihren Platz finden wird.

4.1 Systemarchitektur

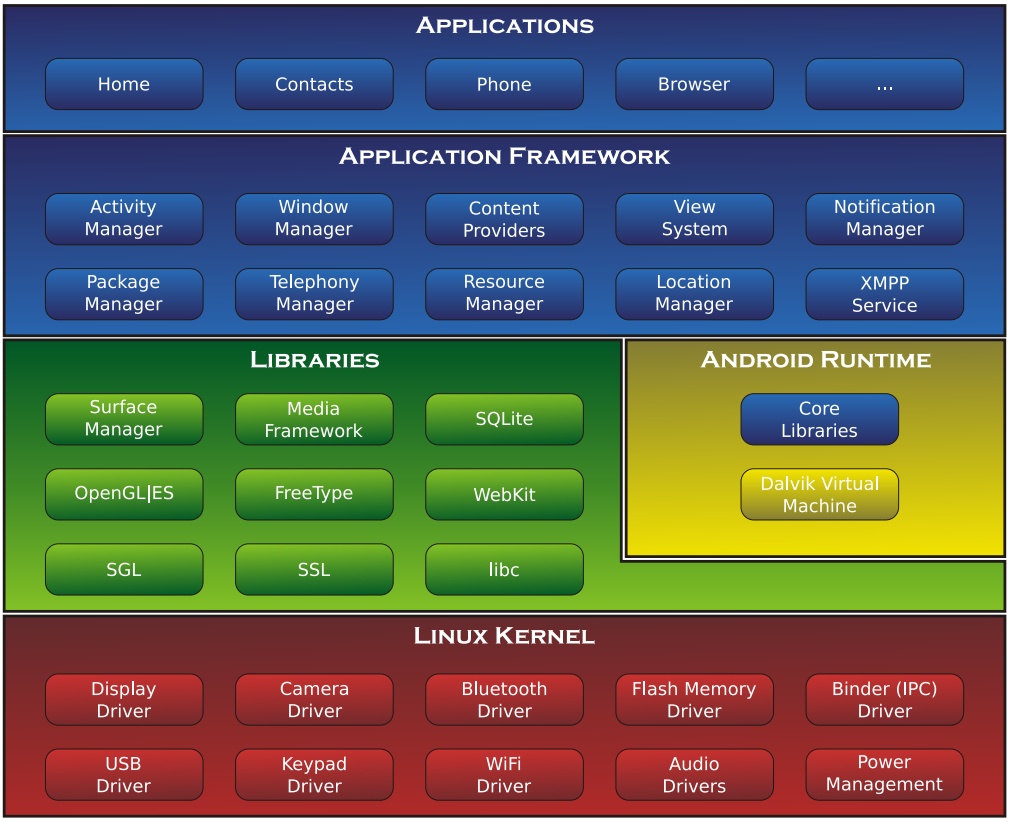

Abbildung 4.1: Schematische Aufbau der Android-Plattform; https://de.wikipedia.org/wiki/Android_(Betriebssystem)#/media/File:Android-System-Architecture.svg

4.2 Linux-Kernel

Android nutzt als Basis den Linux-Kernel, dass Systemdienste wie Sicherheit, Speicher- und Prozessverwaltung, Treibermodell und Netzwerkstapel zur Verfügung stellt, die in Abbildung 1 rot zu sehen sind.

[3] http://www.heise.de/open/meldung/Smartphones-Android-ueberholt-Symbian-Apple-verliert-Marktanteile-html

4.3 Libraries

Android enthält eine Reihe von C/C++-Bibliotheken, die von verschiedenen Komponenten der Plattform genutzt werden, siehe grüner Teil der Abbildung 1. Entwickler greifen indirekt über das Anwendungsframework auf sie zu. Zu diesen Bibliotheken gehören beispielsweise der *Surface Manager*, welcher Bildschirmzugriffe koordiniert und 2D- und 3D-Ausgaben von Anwendungen zu einem Gesamtbild zusammenfügt. *Media Bibliotheken* für die Aufnahme und Wiedergabe von zahlreichen Audio-, Video- und Grafikformaten. *SQLite* ist eine leichtgewichtige relationale Datenbank. Eine 3D-Grafikbibliothek bei der es sich um *OpenGL ES* handelt. *WebKit* ist eine Rendering Engine für Webinhalte.

4.4 Android Runtime

Android-Programme werden durch die virtuelle Maschine *Dalvik* ausgeführt, welche als Aufgabe hat, eine in Java programmierte App aus Bytecode in maschinenlesbaren Code zu übersetzen. Dieser Code wird anschließend vom Prozessor ausgeführt. Mit Android 5.0 wurde *Dalvik*[4] durch den Ahead-of-time-Compiler[5] *Android Runtime* (ART) ersetzt. Siehe dazu den gelben Teil der Abbildung 1. Der zweite Baustein der Android Runtime bilden die sogenannten Core Libraries. Sie enthalten die Android-Java-Klassenbibliotheken. Die Inhalte orientieren sich an der Funktionalität der Java-Standard-Edition (Pakete wie java.math, java.util uvm.).

4.5 Applications

Android ist nicht nur ein Betriebssystem. Zu Android gehören auch eine Reihe von Standardanwendungen, beispielweise Kontakte, Browser und Telefonfunktion.

4.6 Application Framework

Das *Application Framework* erlaubt es äußerst komfortable, ästhetische und leicht bedienbare mobile Anwendungen mit großem Funktionsumfang zu erstellen. Es bietet Zugriff auf die Gerätehardware, zum Beispiel der Kamera, dem Netzwerk oder Sensoren, aber auch das Lesen und Schreiben von Kontaktdaten oder Terminen ist möglich. Ein Rechtesystem steuert hierbei, was ein Programm tun darf, wie auf dem Abbild 2 zu entnehmen ist. Das Rechtesystem erlaubt es auch, dass Anwendungen ihre Funktionen veröffentlichen und anderen Programmen so verfügbar machen können. Programme können also auch Funktionen anderer Anwendungen anfordern. Es lassen sich auch Programmfunktionen ersetzen, so kann der Benutzer sehr leicht den Webbrowser, den E-Mail-Client oder anderes austauschen.

[4] Just-in-time-Compiler, führt Konvertierung in eigenen Bytecode-Format aus für die Java Virtual Machine
[5] Compiler, der Programmcode vor der Ausführung in native Maschinensprache übersetzt

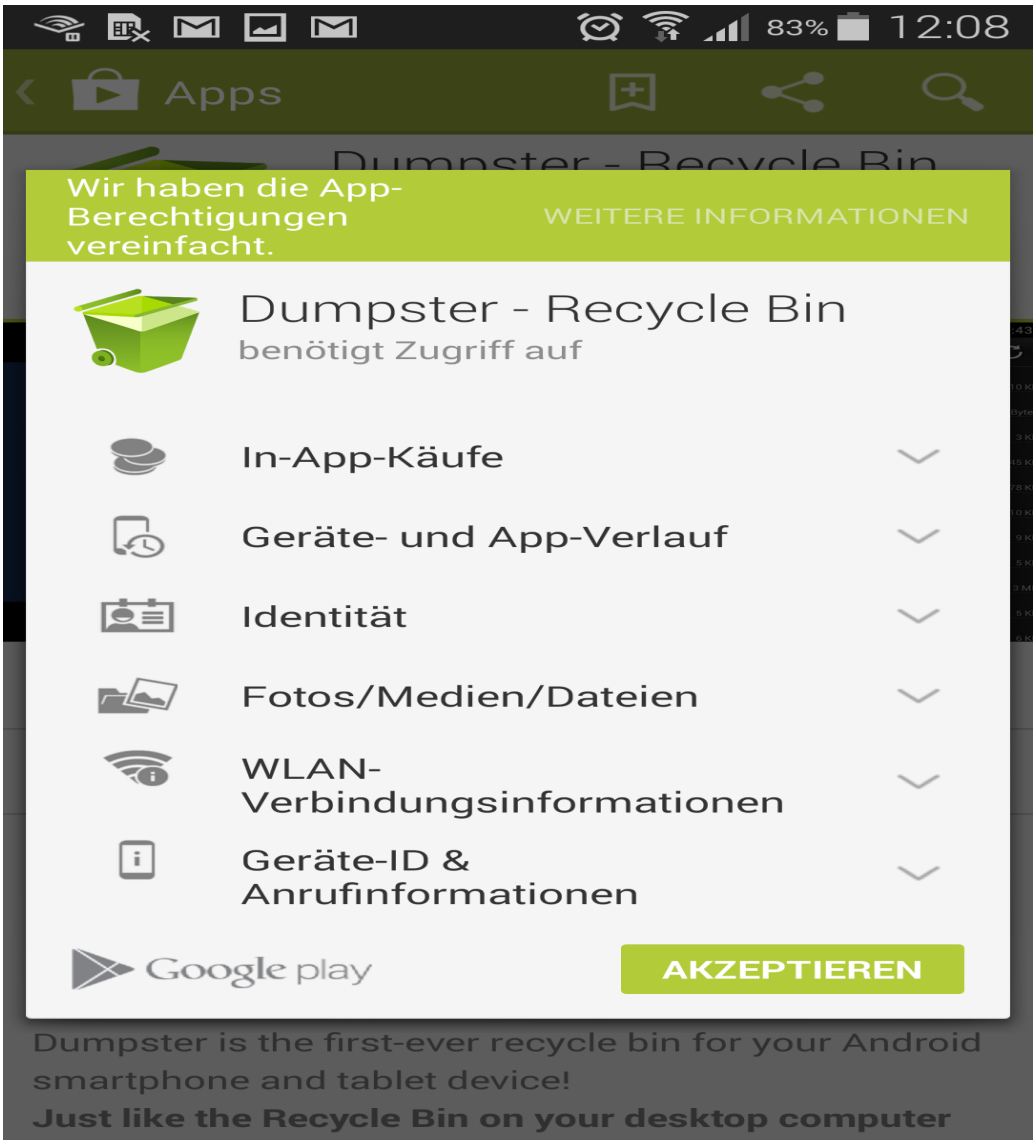

Abbildung 4.2: Schematische Aufbau der Android-Plattform; http://www.giga.de/apps/android/specials/android-berechtigungen-app-rechte-im-detail-erklaert/

Kernbestandteile des *Application Frameworks* (blauer Teil im Abbild 1) sind unter anderem *Views*, welche die Basis für alle Benutzeroberflächen bildet. Dazu gehören auch zahlreiche Standardbedienelemente wie etwa Textfelder oder Schaltflächen. Solche *Views* können durch sogenannte Themes nahezu beliebig angepasst werden. Einen *Activity Manager*, der den Lebenszyklus einer Anwendung steuert. Ein *Content Provider* gestattet Anwendungen den Zugriff auf Daten anderer Programme. Auch das Bereitstellen der eigenen Anwendungsdaten ist auf diese Weise möglich. Der *Resource Manager* gewährt Zugriff auf lokalisierte Zeichenketten, Grafiken und Layoutdateien.

Kapitel 5

Grundlagen

5.1 Android-Projekte

Android-Apps bestehen aus einer ganzen Reihe von Artefakten, die zu einer baumartigen Struktur zusammengefasst werden. Dazu gehören unter anderem Quelltexte, Konfigurationsdateien, Testfälle, aber auch Grafiken, Sounds und Animations. Quelltexte werden in den Ordner src abgelegt. Der Ordner gen der für generated steht, beinhaltet die Klasse R, welche für Ressourcen steht. Sie hat große Bedeutung beim Zugriff auf unterschiedlichste Elemente einer Android-App. Die Ressource wird aber durch den Erstellprozess oder auch Build-tool genannt, Gradle verwaltet und soll deshalb nicht vom Entwickler verändert werden. Der Inhalt ergibt sich aus Dateien des Verzeichnisses res welches für Ressourcen steht. Die Klasse R wird entsprechend angepasst und übersetzt durch Gradle[1] [2], wenn Änderungen an den Unterordnern vorgenommen werden. Der Unterordner values enthält beispielsweise die Datei strings.xml. Sie nimmt Texte auf, die später im Quelltext mithilfe der Klasse R referenziert werden. Genauso wird für die Ordner drawable-[density[3]]dpi verfahren, wobei das density für eine der Größen[4] steht. Dort werden Grafiken beispielsweise im png- oder xml-Format gespeichert. Die Unterschiedlichen drawable-Ordnern stehen für unterschiedliche Bildschirme zur Verfügung. Android läuft auf eine Vielzahl von Geräten mit unterschiedlichen Bildschirmgrößen und Auflösungen. Dadurch resultieren unterschiedliche Pixel Dichten, abhängig von beiden Faktoren. Je nach Pixel Dichte wird eine entsprechende Grafik geladen und skaliert. Eine Grafik wirkt auf einen 5.0 Zoll Smartphone mit einer Bildschirmauflösung von 1920*1080 z.B. sehr groß, auf einem Tablet mit 10.0 Zoll mit identischer Auflösung sehr klein, siehe als Vergleich Abbildung 5.1.

Android führt selbst auch eine Skalierung und Größenänderung durch um verschiedene Bildschirme zu unterstützen. Um ein gleiches Bild einer Anwendung auf unterschiedlichen Geräten zu ermöglichen, sollten Grafiken in mehreren Auflösungen in den Drawable-Ordnern vorliegen,

[1] Auf Java basierendes Build-Management-Automatisierungs-Tool

[2] `https://gradle.org`

[3] `http://developer.android.com/guide/practices/screens_support.html`

[4] ldpi (low): 120dpi, mdpi (medium): 160dpi, hdpi (high): 240dpi, xhdpi (extra-high): 320dpi, xxhdpi (extra-extra-high): 480dpi, xxxhdpi (extra-extra-extra-high): 640dpi.

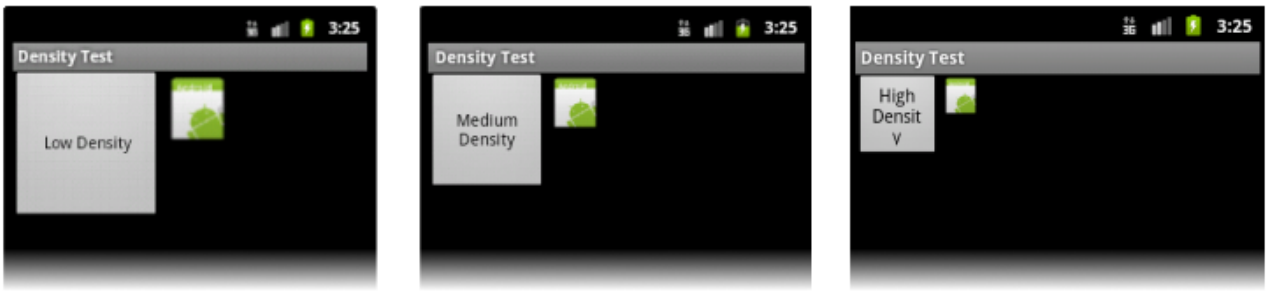

Abbildung 5.1: Beispielanwendung ohne Unterstützung für unterschiedliche Dichten, wie sie auf niedrig, mittel und High-Density-Bildschirme angezeigt wird; http://developer.android.com/guide/practices/screens_support.html

siehe Abbildung 5.2 für eine gute Umsetzung.

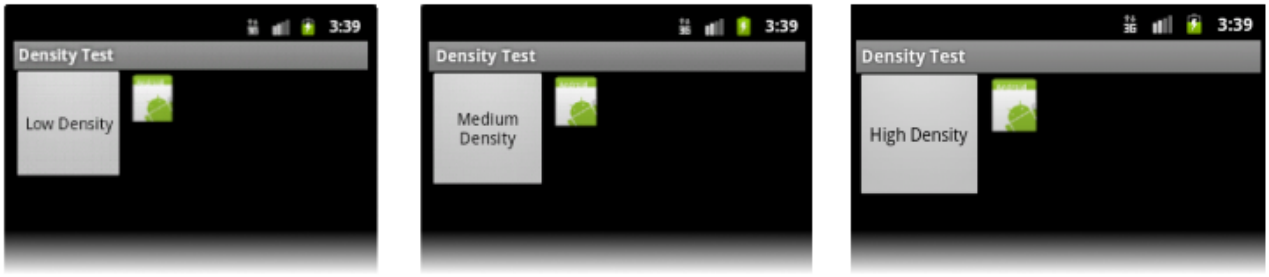

Abbildung 5.2: Beispielanwendung mit guter Unterstützung für unterschiedliche Dichten (Die App ist Bildschirm Dichte unabhängig), gezeigt auf niedrig, mittel und einer hohe Dichte.; http://developer.android.com/guide/practices/screens_support.html

AndroidManifest.xml ist die zentrale Beschreibungsdatei einer Anwendung. In dieser werden unter anderem die Bestandteile der App aufgeführt. Dazu zählen Informationen darüber, welche Activities (Bildschirmseiten) gestartet werden, welche Rechte eine App benötigt, welche Hardware sie erwartet und unter welchen Systemversionen sie lauffähig ist.

5.2 Benutzeroberfläche

Die Benutzeroberfläche ist das Aushängeschild einer Anwendung. Gerade auf mobilen Geräten mit vergleichsweise kleinen Bildschirmen sollte jede Funktion leicht zugänglich und intuitiv erfassbar sein.

5.3 Texte

Neben Bilder und Symbole als Gestaltungsmittel spielen auch Texte eine sehr wichtige Rolle. Sie dienen als Beschriftungen von Bedienelementen. Werden für erläuternde Texte genutzt, die

durch Screenreader vorgelesen werden. Dazu noch für Hinweis- und Statusmeldungen. Da die Apps in der Programmiersprache Java geschrieben werden, kann man sie einfach im Quelltext ablegen. Das sähe folgendermaßen aus:

```
/* Java Text Beispiel */

nachricht.setText("Hallo Android");
```

Listing 5.1: Das Listing zeigt wie auf eine Referenz namens nachricht ein Text im Quellcode gesetzt wird

Das hätte allerdings eine ganze Reihe von Nachteilen. Da jede Klasse in einer eigenen Datei abgelegt wird, merkt man oft nicht, wenn man gleiche Texte mehrfach definiert. Dies vergrößert die App und kostet unnötig Speicher. Zudem wird es auf dieser Weise sehr schwer, mehrsprachige Anwendungen zu programmieren.
Unter Android werden Texte daher zentral in der Datei *strings.xml* abgelegt.

```
/* strings.xml Beispiel */

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HMI</string>
</resources>
```

Listing 5.2: Das Listing zeigt einen Ausschnitt der strings.xml Ressource

Das Attribut *name* des Elements *<string>* wird später im Quelltext als Bezeichner verwendet. Der Name muss also projektweit eindeutig sein. Auf den Text wird dann innerhalb der App mittels Ressource Klasse R zugegriffen. Für das Listing aus 5.2 würde der Zugriff folgend lauten: *nachricht.setText(R.string.app_name);* Neben dem Verzeichnis *values* kann es lokalisierte Ausprägungen[5] geben. Zum Beispiel *values-en* für Englisch oder *values-fr* für Französisch. Die Datei *string.xml* in diesen Ordnern enthält Texte in den korrespondierenden Sprachen[6]. Greift Android auf eine Zeichenkette zu, geht das System vom Speziellen zum Allgemeinen. Ist die Standardsprache beispielsweise Englisch, wird zuerst versucht, den Text in *values-en/strings.xml* zu finden. Gelingt dies nicht, findet *values/strings.xml* Verwendung.

5.4 Oberflächenbeschreibugen

Eine Android-App beschreibt ihre Benutzeroberflächen mittels XML-basierter Layoutdateien. Dieser werden zur Laufzeit der App zu Objektbäumen *aufgeblasen*. Alle Bedienelemente werden in einen Container gepackt.
Oberflächenbeschreibungen werden im Ordner *layout*, einem Unterverzeichnis von *res*, gespeichert. Die XML-Datei bildet die Hierarchie der Benutzeroberfläche ab.

[5] http://developer.android.com/training/basics/supporting-devices/languages.html
[6] http://developer.android.com/guide/topics/resources/localization.html

```
/* mainlayout.xml Beispiel */

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout>
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/buttonOK"
        android:text="R.string.OK"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Listing 5.3: Das Listing zeigt ein Beispiel zur Oberflächenbeschreibung einer Layout Datei

Um eine Referenz auf die Schaltflächen ermitteln zu können, wird ein Name definiert, beispielweise *buttonOK* aus dem Listing 5.3. Dieser wird wieder als Konstante in der Klasse *R* aufgenommen. Die Angabe *android:id="@+if/* sorgt also dafür, dass ein Bedienelement ein Bezeichner zugewiesen wird, auf den mittels *R.id* zugegriffen werden kann.
Übersicht vorhandener Layouts:

Name	Beschreibeung
FrameLayout	Anzeigen einer einzigen View
LinearLayout	Horizontale oder vertikale Ausrichtung
TableLayout	Jedes Oberflächenelement wird in einem Feld einer Tabelle dargestellt
AbsoluteLayout	Oberflächenelemente werden absolut zum Bildschirmrand ausgerichtet
RelativeLayout	Elemente werden in relation zu sich, anderen Views oder Bildschirmrand ausgerichtet
ListView	Listendarstellung von Oberflächenelementen
Gallery	Stellt eine Liste von Bildern horizontal dar
GridView	Stellt eine Oberfläche in Gitterstruktur dar
TabHost	Layout für Verwendung von Tab- und Reiter-Strukturen
MapView	Stellt eine Google-Maps-Karte dar
WebView	Dienst zur Darstellung von Internetseiten innerhalb der App

5.5 Activities

Jeder Activity ist eine Benutzeroberfläche[7], also ein Baum bestehend aus *Views*[8] und *ViewGroups*[9], zugeordnet. Activities bilden demnach die vom Anwender wahrgenommenen Bau-

[7] http://developer.android.com/guide/topics/ui/overview.html
[8] http://developer.android.com/reference/android/view/View.html
[9] http://developer.android.com/reference/android/view/ViewGroup.html

steine einer App. Sie können sich gegenseitig aufrufen und somit innerhalb einer App eine Vorwärtsnavigation realisieren. Da das System Activities auf einem Stack, also einem Stapel, ablegt, müss man sich als Entwickler nicht darum kümmern, von wem eine Activity aufgerufen wurde. Drückt der Benutzer den *Zurück-Knopf*, wird automatich die zuvor angezeigte Activity reaktiviert. Android fördert die saubere Strukturierung einer App durch Activities. Jeder Activity steht ein Fenster für ihre Bedienelemente zur Verfügung.
Jede Activity besitzt einen Lebenszyklus, siehe Abbildung 5.3 Die Methode *onCreate()* wird

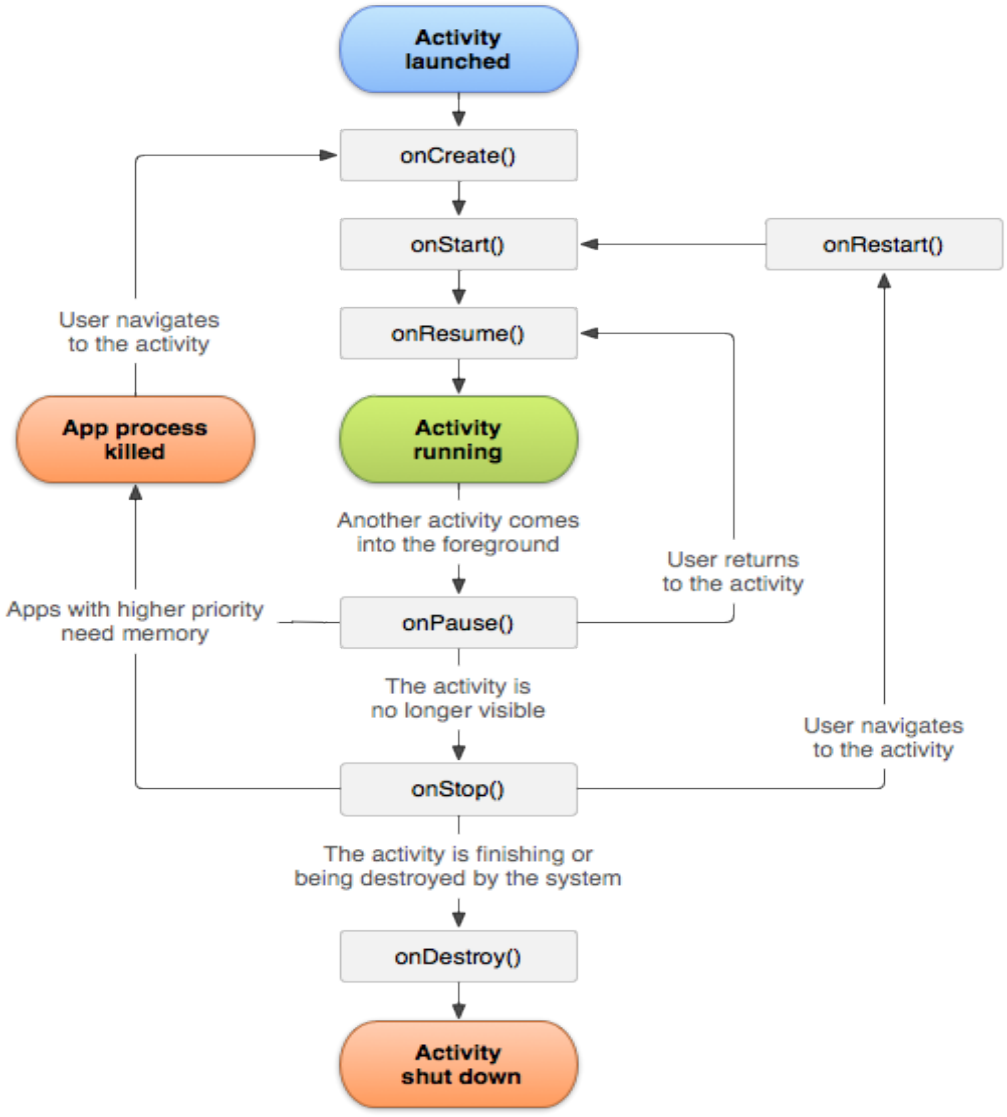

Abbildung 5.3: Lebenszyklus von Activity; http://developer.android.com/guide/components/activities.htm

von praktisch jeder Activity überschrieben. Android ruft sie während der Initialisierungsphase einer Aktivität auf. Sie erledigt normalerweise folgende Aufgaben:

1. Aufrufen der gleichnamigen Elternmethode
2. Initialisieren von Instanzvariablen
3. Setzen der Benutzeroberfläche

4. Wiederherstellen eines gespeicherten Zustands

Das setzen der Benutzeroberfläche erfolgt durch die Anweisung *setContentView()*. Der übergebene Parameter, zum Beispiel R.layout.main, referenziert das entsprechende Layout. Wiederherstellen von gespeicherten Zuständen wird genutzt, wenn sich die Ausrichtung des Handy ändert, wird die *onCreate()* Methode erneut aufgerufen und die App neugestartet. Um nicht gespeicherte Daten zu verlieren, können diese als Zustände gespeichert werden und wiederhergestellt werden. Die *onRestart()* Methode wird aufgerufen, nachdem die Aktivität gestoppt wurde, gerade bevor sie wieder in Betrieb genommen wird. *onStart()* wird aufgerufen, kurz bevor die Aktivität für den Benutzer sichtbar wird. Die *onResume()* Anweisung wird aufgerufen, kurz bevor die Aktivität mit dem Benutzer beginnt zu interagieren. An diesem Punkt ist die Aktivität an der Oberseite des Aktivitätsstapel und bieten Benutzereingaben an. *onPause()* wird aufgerufen, wenn das System kurz vor dem Start eine andere Tätigkeit wieder aufnimmt. Die Methode wird verwendet um nicht gespeicherte Änderungen persistent zu speichern. *onStop()* Wird aufgerufen, wenn die Aktivität nicht mehr sichtbar für den Benutzer ist. Dies kann passieren, weil es zerstört wurde oder weil eine andere Aktivität (entweder eine bestehende oder eine neue) wieder aufgenommen wurde und die Activity bedeckt. Zum Beenden der App, wird die *onDestroy()* Methode aufgerufen, bevor die Aktivität zerstört wird. Dies ist der letzte Aufruf, die die Aktivität erhält. Es kann passieren, weil entweder die Aktivität beenden ist (Finish() aufgerufen wurde), oder weil das System vorübergehend Platz zu sparen möchte.

Kapitel 6

Android-App: HMI

Kapitel 7

Fazit

Insgesamt lässt sich hieraus der Schluss ziehen, dass

Kapitel 8

Beispiele

Dieses Kapitel soll viele alltaegliche Beispiele[1] abdecken um einen LaTeX Dokument zu setzen

8.1 Schriftarten

- Kursiv: *Das ist ein Beispiel*
- Unterstreichen: Das ist ein Beispiel
- Fettschrift: **Das ist ein Beispiel**
  - Kombination aus dreien: ***Das ist ein Beispiel***
- Serifen: Das ist ein Beispiel
- Schreibmaschinen Schrift: `Das ist ein Beispiel`
- Kleine Grossbuchstassen: Das ist ein Beispiel
- Ausfuehrungszeichen: “Das ist ein Beispiel”
- asld

8.1.1 Symbole

1. Deutsche Umlaute: Ä, Ö, Ü, ä, ö, ü, ß
2. Sammlung von Sonderzeichen `http://de.wikibooks.org/wiki/LaTeX-Kompendium:_Sonderzeich`

8.2 Abbildungen

Wie folgt bindet man Abbildungen ein:

[1] Fussnote

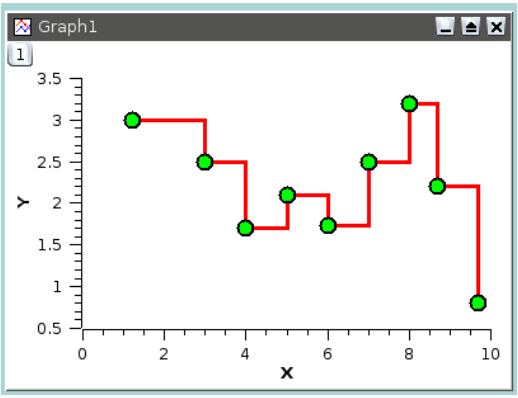

Abbildung 8.1: Beispiel Bild; Quelle ist png

8.3 Tabellen

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren.

Stadium	Substratfreie Kontrolle	Probenansatz	
	Farbe	Farbe	Bewertung
Alpha1	farblos	braun	+++
Beta2	farblos	farblos	-

2. Beispiel

Rang	Name	Rating
1	Garry Kasparov	2817
2	Viswanathan Anand	2774
3	Wladimir Kramnik	2764

Tabelle 8.1: Beispiel Beschriftung einer Tabelle

8.4 Verweise

Hier werden Verweise auf verschiedene Elemente erstellt [LK73]

8.4.1 Pageref und Ref

Diese Textstelle ist sehr interessant.
Hier wird auf die Textstelle 8.4.1 verwiesen,

die sich auf der Seite 4 befindet.

Verweis auf Listing 8.1 auf Seite 5
Verweis auf Abbildung 8.1 auf Seite 4
Verweis auf Tabelle 8.1 auf Seite 4

8.5 Listing

```
/* Java Text Beispiel */

nachricht.setText("Hallo Android");
```

Listing 8.1: Das Listing zeigt Java Quellcode

Literaturverzeichnis

[LK73] Shen Lin and Brian W. Kernighan, *An effective heuristic algorithm for the travelling-salesman problem*, Operations Research **21** (1973), 498–516.

[Wik12] Wikipedia, *Nick clegg — wikipedia, die freie enzyklopadie*, 2012, [Online; Stand 11. September 2012].