Fachhochschul-Bachelorstudiengang
SOFTWARE ENGINEERING
A-4232 Hagenberg, Austria

Effizientes Autofahren mit der ECO.Drive App für Android-Mobiltelefone

Bachelorarbeit
Teil 1

zur Erlangung des akademischen Grades
Bachelor of Science in Engineering

Eingereicht von

Philipp Pendelin

Begutachter: DI Dr. Werner Kurschl

Hagenberg, Juni 2012

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 18. Juni 2012

Philipp Pendelin

Inhalt

Kurzfassung

Die Verkehrslage in Mitteleuropa spitzt sich aufgrund des steigenden Verkehrsaufkommens immer weiter zu. Durch das Steigen des Verkehrsaufkommens steigt auch die Umweltbelastung, welche zu gesundheitlichen Schäden der Bevölkerung führen kann.

Ein Beitrag zur Lösung dieses Problems ist es, die Fahrzeuglenker dahingehend zu sensibilisieren, einen möglichst energieeffizienten Fahrstil zu entwickeln. Dies kann beispielsweise mit sanften Beschleunigungsvorgängen und nicht extrem überhöhten Maximalgeschwindigkeiten erreicht werden. Selbst das Einsetzten der aufgebauten Energie durch gezieltes Ausrollen lassen des Fahrzeugs sollte nicht vernachlässigt werden.

Ziel dieser Arbeit ist es, die Android-Plattform zu analysieren und auf Basis dieser Software-Plattform eine Prototypenanwendung namens ECO.Drive zu entwickeln. Die Anwendung soll zwei grundsätzliche Funktionen bieten. Erstens soll der Lenker während der Autofahrt ständig visuelles Feedback (in Form von Strafpunkten) zur Energieeffizienz seines Fahrstils bekommen. Zweitens soll es möglich sein, sämtliche Daten, welche während einer Fahrt aufgezeichnet werden strukturiert und entsprechend aufbereitet (gefiltert) in geeigneter Form exportieren zu können um diese für eventuelle andere Anwendungsfelder (z.B.: Verkehrsdatenberechnung oder Erstellung von Simulationsmodellen für gewisse Regionen) verfügbar zu machen.

Abstract

The situation on the roads of Central Europe due to the increasing traffic density is getting more and more critical. An effect of the increasing traffic density is the increasing environmental pollution which has a negative impact on the health of human beings.

A contribution for solving this problem is to sensitize drivers in that way that they drive their cars as economical as possible. This could be reached by smooth acceleration trajectories and lower maximal speeds. Also the skillful usage of the residual energy should not be neglected.

The main goal of this bachelor thesis is to analyze the Android-software-platform and to develop a prototype application called ECO.Drive. This application should offer two main functionalities. The first functionality is to give feedback (via penalty points) to the driver during driving the car. The second functionality is to provide the recorded data in a structured and appropriate (filtered) format for exporting and further using for other fields of application (e.g. calculating traffic models or creating simulation models for certain regions).

1 Einleitung

1.1 Motivation

Die Verkehrslage in Mitteleuropa spitzt sich immer weiter zu. Nach der letzten Studie der Statistik Austria bezüglich Pendler in Österreich geht der Trend nach wie vor zum PKW (vgl. [7]). Auch in Deutschland und der Schweiz ist eine stetige Zunahme des Personenverkehrs zu beobachten(vgl. [9], [10]). Aufgrund dieser Tatsache kann man mehrere Probleme unmittelbar schlussfolgern(vgl. [8]):

- Längere Arbeitswege für Einpendler durch Staus und Unfälle
- Höhere Umweltbelastung durch Emissionen
- Höhere Lärmbelastung
- Spätfolgen wie z.B. Atemwegs- und Herz-/Kreislauferkrankungen

Wobei Themen wie die Dauer des Weges zur Arbeit als „Einzelschicksal“ gewertet werden können, können Punkte wie Emissionen, Lärmbelastungen und sogar die Gesundheit der Bevölkerung nicht einfach ignoriert werden. Auch eine technische Weiterentwicklung (vgl. [11]) der herkömmlichen Verbrennungsmotoren können das Problem der Umweltbelastung nicht lösen sondern lediglich verbessern. Andere Strategien wie beispielsweise die Gewichtsreduktion von Autos stehen im Konflikt mit dem zunehmenden Maß an Technik und Sicherheitssysteme an Board (vgl. [6]).

1.2 Problemstellung

Ein gänzlich anderer Zugang anstatt der stetigen technischen Weiterentwicklung von Antriebssystemen und Fortbewegungstechnologien ist die Lenker bzw. die Fahrer der Kraftfahrzeuge dahingehend zu sensibilisieren, dass der Verbrauch an Kraftstoff maßgeblich von deren Fahrverhalten, der Benützung von Extras und der Wahl der Route abhängig ist (vgl. [6]).

Eine Möglichkeit zur entsprechenden Konditionierung der Lenker stellt die Entwicklung von so genannten *Realtime-Feedback-Systemen* dar, welche darauf abzielen, dem Fahrer eine unmittelbare Reaktion auf seine Aktion zu geben, da eine „Belehrung“ des Lenkers nach der Fahrt nicht die gleiche Wirkung wie während der Fahrt bzw. des „Vergehens“ nach sich zieht (vgl. [6], No Time Like The Present).

Ein Beispiel für ein solches System wäre ein unmittelbares Feedback zum gewählten Gang und zum aktuellen Beschleunigungsverlauf. Mit dieser simplen Methode ist es dem *University of California, Riverside's Center for Environmental Research and Technology* (kurz CERT) gelungen eine Kraftstoffersparnis von durchschnittlich 6% zu erreichen (vgl. [6], The Nut Holding The Wheel).

Die Gefahr beim unmittelbaren Feedback ist jedoch die Tatsache, dass es unter Umständen zu Gefahrensituationen aufgrund mangelnder Konzentration für den Straßenverkehr kommen kann, falls gleichzeitig diverse Systeme auf verschiedene Sinne des Lenkers einwirken. Andererseits ist darauf zu achten, dass die Systeme nicht als nervend oder gar störend empfunden werden, da dies zu einer kompletten Ablehnung bzw. einer Nichtverwendung des Systems führen kann (vgl. [6], No Time Like The Present).

1.3 Ziele der Arbeit

1.3.1 Analyse der Android Plattform

Ziel ist es, das weltweit am häufigsten verwendete Betriebssystem für Mobiltelefone (vgl. [12]) näher zu beleuchten und die unterschiedlichen Aspekte und Ansätze zu analysieren. Aufbauend auf dem ersten Abschnitt, welcher sich mit den verfügbaren Versionen und deren Keyfeatures auseinandersetzt wird auch auf die Systemarchitektur und die Aspekte des eigens für die Plattform entwickelte Java-Laufzeitsystems und die des Applikationsframeworks näher eingegangen. Einige der wichtigsten Punkte sind dabei:

- Android Architketur
- Basiskonzepte (Activities, Intents, Services, etc.)
- Deklaratives UI-design
- Unterschiede der Android-Klassenbibliothek zur Java-Klassenbibliothek

1.3.2 Arbeiten mit Beschleunigungs- und Positionssensordaten

In modernen Applikationen spielen Sensordaten eine immer wichtigere Rolle. Parkplatzfinder, Augmented Reality Applications und diverse Spiele bedienen sich an Sensordaten wie der aktuellen Position des Endgerätes, dem Neigungswinkel bezüglich des Bodens und vieles mehr. Da der Prototyp auf Beschleunigungs- und Positionswerten basiert, wird auf den Beschleunigungssensor und den GPS-Sensor näher eingegangen und deren Funktionsweise erläutert.

1.3.3 Prototyp ECO.Drive

Das Ziel des Prototyps ist es eine Ratingmöglichkeit für Fahrten mit dem Kraftfahrzeug zu realisieren. Dabei wird mittels der gemessenen Daten des Beschleunigungs- und GPS-Sensors ein Rating erstellt, welches dem Fahrer unmittelbares Feedback zum momentanen Fahrstil und auch zum Gesamtfahrstil einer aufgezeichneten Fahrt gibt. Die wesentlichen Punkte sind daher:

- Sensordaten aufzeichnen
- Sensordaten aufbereiten
- Auswertung der gewonnenen Daten (Rating erstellen)
- Visualisierung der Daten

1.4 Überblick

Kapitel 2 erläutert die Grundlagen der Android-Plattform und die wichtigsten API-Elemente aus der Sicht des Softwareentwicklers. Ebenfalls werden die grundlegenden Begriffe von Beschleunigungs- und GPS-Sensoren eingeführt, sowie auf deren Funktionsweisen und Konzepte näher eingegangen.

Kapitel 3 beschreibt die notwendigen Anforderungen zur Entwicklung des Prototyps ECO.Drive. Auch die Use-Cases für die Applikation für den Betrieb als Datenlogger bzw. als Ratinginstrument werden konkretisiert.

Kapitel 4 schildert das Design des Prototyps ECO.Drive und erläutert die Implementierungstechnischen Details im Bezug auf Architektur, Sensordatenverarbeitung und Visualisierung.

Kapitel 5 geht näher auf konkrete Testfälle des Prototyps ECO.Drive ein und stellt die Resultate entsprechend gegenüber. Dieses Kapitel enthält weiters eine Evaluierung der Android-Plattform in Bezug auf die Applikationsentwicklung mit dem Android-SDK.

Kapitel 6 fasst die Ergebnisse der Arbeit zusammen und gibt Auskunft über mögliche künftige Erweiterungen des Prototyps ECO.Drive.

2 Grundlagen

2.1 Die Android Plattform

Android ist eine OpenSource-Softwareplattform welche in mobilen Endgeräten wie beispielsweise Mobiltelefonen und Tablets eingesetzt wird. Android wurde von der *Open Handset Alliance* entwickelt. Die Open Handset Alliance besteht mittlerweile aus 84 Unternehmen, wobei Unternehmen aus verschiedensten Märkten wie beispielsweise Chiphersteller, Mobiltelefonhersteller und Mobilnetzbetreiber beteiligt sind (vgl. [25], [4]). Gründer dieses Konsortiums war die Firma Google. Android ist seit dem 21. Oktober 2008 offiziell als Open Source Plattform verfügbar (vgl. [24]) und mit einem Marktanteil von 48% der Weltmarktführer bei mobilen Betriebssystemen (vgl. [26]).

Abbildung 2.1: Android Logo mit Android-Roboter (siehe [27])

2.1.1 Versionen

Aufgrund der schnellen Releasezyklen für neue Versionen gibt es bereits eine Vielzahl an verschiedenen Versionen (vgl. [29]). Jede Version besitzt einen so genannten API-Level, bei dem es sich um einen ganzzahligen Wert, welcher die Frameworks-Revision eindeutig kennzeichnet handelt. Aufgrund dieses API-Levels ist eine Abwärtskompatibilität zwischen den einzelnen Versionen gegeben, sodass eine Applikation lediglich einen minimalen API-Level verlangt bzw. verwendet und damit sichergestellt ist, dass diese lauffähig bleiben. In den seltensten Fällen werden API-Funktionen tatsächlich aus dem Framework herausgenommen bzw. dahingehend verändert, dass diese nicht mehr verwendbar sind. Die weitaus gebräuchlichere Methode ist Funktionen zu kennzeichnen, dass diese nicht mehr verwendet werden sollen (deprecated-

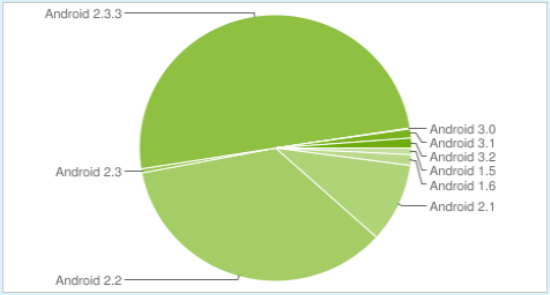
Kennzeichnung). Weiters besitzt jede Version außer den ersten beiden Basisversionen einen eigenen Codenamen, welcher sich an Süßspeisen orientiert.

Übersicht der Versionen (siehe. [30]):

Plattformversion	API-Level	Codename	Erscheinungsdatum
Android 4.0.3	15	Ice Cream Sandwich MR1	
Android 4.0, 4.0.1, 4.0.2	14	Ice Cream Sandwich	Oktober 2011
Android 3.2	13	Honeycomb MR2	Juni 2011
Android 3.1.x	12	Honeycomb MR1	Mai 2011
Android 3.0.x	11	Honeycomb	Februar 2011
Android 2.3.4, 2.3.3	10	Gingerbread MR1	Februar 2011
Android 2.3.2, 2.3.1, 2.3	9	Gingerbread	November 2010
Android 2.2.x	8	Froyo	Juni 2010
Android 2.1.x	7	Eclair MR1	Januar 2010
Android 2.0.1	6	Eclair 0 1	Dezember 2009
Android 2.0	5	Eclair	November 2009
Android 1.6	4	Donut	September 2009
Android 1.5	3	Cupcake	Mai 2009
Android 1.1	2	Base 1 1	Februar 2009
Android 1.0	1	Base	Oktober 2008

Verbreitung der Versionen (Stand 1. Dezember 2011):

Durch ein ständiges Monitoring der Endgeräte, welche auf den Android-Market zugreifen, lässt sich eine repräsentative Verteilung der Versionen festmachen (siehe Abbildung 2.2).



Platform	Codename	API Level	Distribution
Android 1.5	Cupcake	3	0.8%
Android 1.6	Donut	4	1.3%
Android 2.1	Eclair	7	9.6%
Android 2.2	Froyo	8	35.3%
Android 2.3 - Android 2.3.2	Gingerbread	9	0.5%
Android 2.3.3 - Android 2.3.7		10	50.1%
Android 3.0	Honeycomb	11	0.1%
Android 3.1		12	1.1%
Android 3.2		13	1.2%

Data collected during a 14-day period ending on December 1, 2011

Abbildung 2.2: Verteilung der Android-Versionen (siehe [28])

2.1.2 Architektur

Die Android-Architektur (vgl. [2], Chapter 2 – Key Concepts) baut auf einem mehrschichtigen Modell auf, wobei jede Schicht auf den Services und Funktionen der darunterliegenden Schichten aufbaut (siehe Abbildung 2.3).

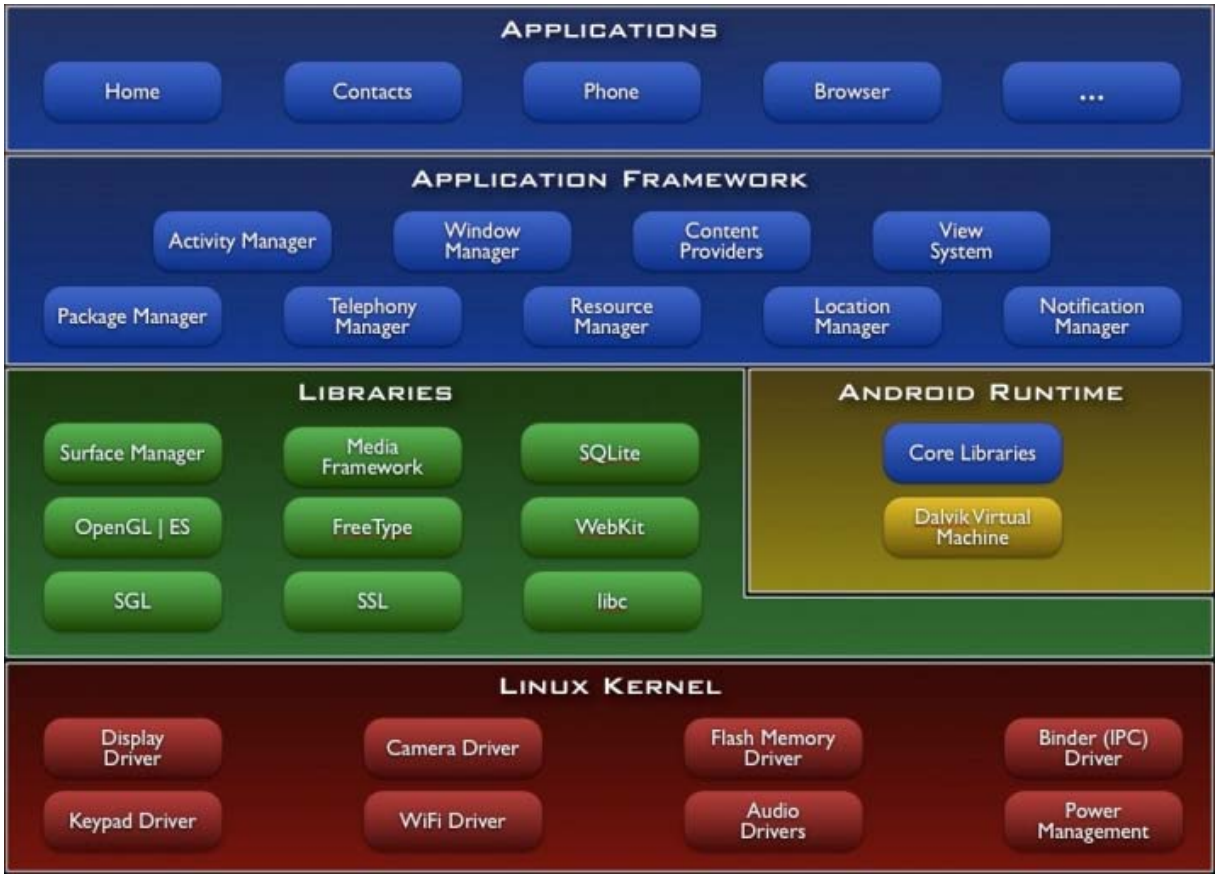


Abbildung 2.3: Android Systemarchitektur (siehe [31])

Im Folgenden wird jede der Schichten näher erläutert:

- *Linux-Kernel:* Die unterste Schicht der Android-Architektur baut auf einem Linux-Kernel auf. Dieser Kernel ist für Speicherverwaltung, Prozessverwaltung und diverse andere Betriebssystemaufgaben zuständig. Anzumerken ist hierbei, dass der Benutzer nichts von dieser Schicht zu Gesicht bekommt, da diese völlig von den darüber liegenden Schichten gekapselt wird.
- *Libraries:* Die auf den Linux-Kernel aufsetzende Schicht besteht im Wesentlichen aus nativen Basisbibliotheken, welche in C und C++ implementiert sind und auf die jeweilige Zielplattform kompiliert sind. Seit Android 1.5 ist es möglich, diese Basisbibliotheken durch eigene bzw. andere zu ersetzen. Ermöglicht wird dies durch den so genannten *Native Devlopment Toolkit* (kurz NDK).

- *Android Runtime:* Auf die Laufzeitumgebung und die virtuelle Maschine wird im Kapitel 2.3.1 näher eingegangen.
- *Application Framework:* Aufbauend auf den Basisbibliotheken und dem Laufzeitsystem befindet sich das so genannte Applikationsframework. Diese Schicht stellt die Basisbausteine zum Entwickeln von Android-Programmen, welche in Java geschrieben sind zur Verfügung. Wie bei der Basisbibliotheks-Schicht ist es auch hier möglich die vorgefertigten Bausteine bzw. Funktionalitäten durch eigene bzw. andere Komponenten zu ersetzten.
- *Applications:* Die oberste Schicht der Android-Architektur wird durch die Applikationsschicht gebildet. Benutzer von Android-Geräten haben ausschließlich mit dieser Schicht zu tun. Sie bietet all jene Applikationen, Widgets und Funktionen, welche nötig sind um das System zu bedienen bzw. mit dem System zu arbeiten. Diese Schicht ist wie die Applikationsframework-Schicht in Java geschrieben. Die wichtigsten Standardapplikationen, welche in jedem Android-Mobiltelefon enthalten sind, sind beispielsweise die Telefonanwendung (Wählen, zuletzt gewählte Rufnummern, etc.), der Emailklient (Senden und Empfangen von E-Mails, Verwaltung der E-Mail-Konten), das Kontaktverzeichnis (Verwaltung von Kontakten bzw. Telefonnummern) oder auch der Webbrowser, der Kalender und viele mehr.

2.1.3 Laufzeitsystem

Ebenfalls auf dem Linux-Kernel aufsetzend befindet sich das so genannte Laufzeitsystem. Das Android-Laufzeitsystem besteht aus der *Dalvik-Virtual-Machine* und den Java-Kernbibliotheken (vgl. [2], Chapter 2 – Key Concepts; [3], Chapter 3 - Introducing the Android computing platform).

Wie bei der herkömmlichen Java-Implementierung wird auch bei Android der geschriebene Java-Programmcode durch den Compiler in Bytecode umgewandelt, welcher später in der *Dalvik-Virtual-Machine* ausgeführt bzw. interpretiert wird. Im Gegensatz zu Standard-Java ist der Bytecode leicht unterschiedlich und die virtuelle Maschine ist für den Einsatz in mobilen Endgeräten optimiert. In der Dalvik-VM laufen so genannte „.dex“-Files, welche zur Kompilezeit aus den herkommlichen „.class“- und „.jar“-Files erzeugt werden. Das „.dex“-Format ist kompakter und effizienter als herkömmliche „.class“-Format und ist wie die virtuelle Maschine selbst gezielt für Geräte mit wenig Hauptspeicher und limitierten Batterielaufzeiten optimiert.

Der zweite Teil des Laufzeitsystems wird durch die nötigen Java-Kernbibliotheken gebildet, welche im Wesentlichen aus einem Subset der Jave-SE Bibliotheken mit einigen Ergänzungen für den mobilen Einsatz besteht.

2.1.4 Activities

Anders als bei herkömmlichen Desktopsystemen gibt es bei Android immer nur eine Applikation, welche sich im Vordergrund befindet. Diese benötigt im Regelfall die gesamte zur Verfügung stehende Bildschirmgröße sowie auch die Statusleiste. Die Standardapplikation, welche läuft nachdem das Gerät eingeschaltet wurde ist die so genannte *Home Application* (vgl. [2], Chapter 2 – Key Concepts).

Intern wird jedoch jede Benutzermaske durch eine so genannte *Activity* abgebildet. Eine Applikation bzw. ein Programm kann aus einer oder mehreren Activities bestehen, wobei immer nur eine Activity zu einem Zeitpunkt aktiv sein kann.

Navigiert der Benutzer durch eine Applikation, welche aus mehreren Ansichten (Activities) besteht, wird der so genannte *ApplicationStack* aufgebaut, welcher durch das Drücken der „Back"-Taste wieder abgebaut wird. Natürlich kann auf diesem AplicationStack auch programmatisch zugegriffen werden.

Im Folgenden wird der Lifecycle einer Android-Activity näher erläutert (vgl. [2], Chapter 2 – Key Concepts; [1], Handling Activity Lifecycle Events). Der Activity Lifecycle hat nicht zwingend etwas mit dem Applikations-Lifecycle zu tun, da wie bereits erwähnt eine Applikation aus mehreren Activities bestehen kann. Die Ausgangsbasis dafür bildet Abbildung 2.4.

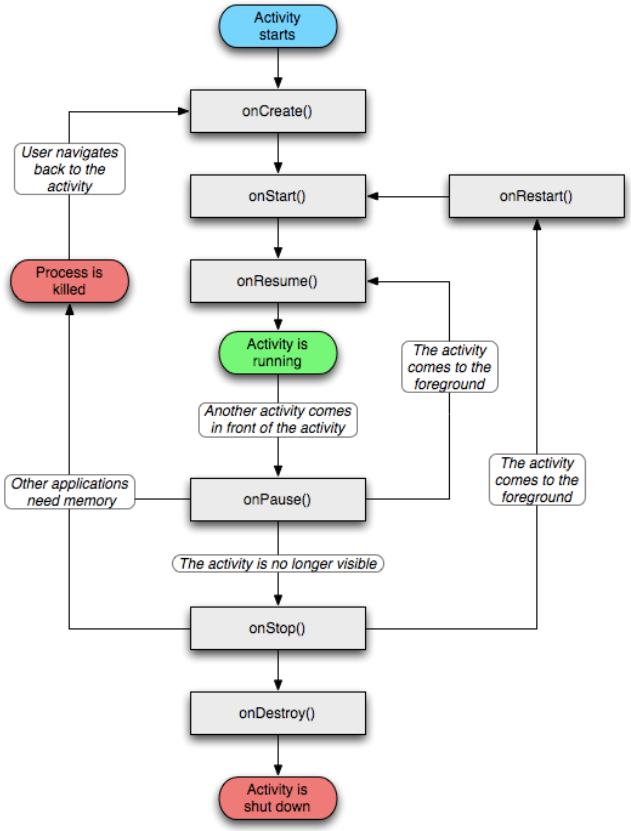


Abbildung 2.4 Activity Lifecycle (siehe [32])

Der Status, in welchem sich eine Activity befindet wird alleine vom System festgelegt und kann vom Programmierer nicht direkt beeinflusst werden (Indirekt durch Mechanismen wie z.B. Intents auf welche im Kapitel 2.1.5 näher eingegangen wird).

Da jede Benutzersicht von der Basisklasse Activity abgeleitet ist, ist es möglich, auf alle Zustände im Activity-Lifecycle zu reagieren, indem man die entsprechenden Methoden überschreibt.

Die einzelnen Zustände im Detail:

- *onCreate:* Wird aufgerufen, wenn die *Activity* das erste Mal geladen bzw. gestartet wird. Hier werden im Normalfall die Initialisierungen der aktuellen Activity (z.B.: Verbindung zur XML-View und deren Steuerelemente) hergestellt.
- *onStart:* Wird aufgerufen, sobald die Activity für den Benutzer sichtbar wird.
- *onResume:* Wird aufgerufen, sobald der Benutzer mir der Activity interagieren kann.
- *onPause:* Wird aufgerufen, wenn die Activity in den Hintergrund versetzt wird. Dies passiert in den häufigsten Fällen, wenn die Activity von einer anderen Activity abgelöst wird. Je schneller diese Methode abgearbeitet wird, desto schneller kann die nächste Activity angezeigt werden, da diese auf die Fertigmeldung dieser Methode wartet. Da diese Methode die letzte aufgerufene ist, falls Speicherknappheit besteht, werden in dieser oftmals die eventuell nötigen Persistierungsaktionen durchgeführt.
- *onStop:* Wird aufgerufen, sobald die Activity nicht mehr angezeigt wird, da bereits eine andere Activity aktiv ist. Falls Speicherknappheit beim System besteht, wird diese Methode unter Umständen gar nicht mehr aufgerufen sondern einfach gestoppt.
- *onDestroy:* Wird aufgerufen, wenn die Activity geschlossen wird. Es handelt sich hierbei um die letzte Mehtode, welche aufgerufen wird, bevor die Activity endgültig stoppt. Wie auch bei der Methode onStop kann es sein, dass diese Methode gar nicht mehr aufgerufen wird falls Speicherknappheit besteht.
- *onRestart:* Wird aufgerufen wenn die Activity wieder zu laufen beginnt, nachdem sie sich zuvor im Stopp-Zustand befunden hat.

2.1.5 Intents

Intents werden genutzt um bestimmte Aktionen zu beschreiben bzw. auszulösen. Beispielsweise das Navigieren durch eine Applikation bzw. das Anzeigen verschiedener Activities passiert durch Intents (vgl. [33]). Auch Services (siehe Kapitel 2.1.6) werden duch Intents gestartet.

Ein Intent-Objekt ist im Grunde lediglich eine passive Datenstruktur, welcher man eine auszuführende Aktion und die entsprechenden Daten bekanntgibt.

Es gibt zwei verschiedene Arten von Intents:

- *Explizite Intents:* Bei dieser Art von Intents kann man bereits beim Anlegen eines neuen Intent-Objekts festlegen, welche Klasse sich um die Anfrage kümmern soll. Diese Art von Intent wird häufig für die Navigation bzw. für die Kommunikation mit Services in der eigenen Applikation verwendet, da dort die Klassennamen oftmals bekannt sind.

- *Implizite Intents:* Bei dieser Art von Intents gibt man keinen Klassenname bzw. eine Komponente an, welche den Intent verarbeitet. Es wird lediglich der „Wunsch" nach einer Aktion welche durchgeführt werden muss abgesetzt und das System bzw. der Benutzer entscheidet, welche Anwendung den Intent verarbeitet. Ein Beispiel dafür wäre, dass man beispielsweise eine E-Mail versenden möchte. Dafür generiert man ein neues Intent-Objekt mit den entsprechenden Daten. Ist nur eine E-Mail-Applikation am System registriert, wird diese umgehend gestartet und verarbeitet die Anforderung. Falls mehrere Applikationen in Frage kommen welche den Intent verarbeiten können, fragt das System beim Benutzer nach, welche E-Mail-Applikation verwendet werden soll. Falls keine Applikation den Intent entgegennimmt, kommt es zu einer Fehlersituation. Über entsprechende Intent-Filter im Android-Manifest (siehe Kapitel 2.1.10) kann man die eigene Applikation als Empfänger für Implizite Intents registrieren.

Intents, welche über die Applikations- bzw. Prozesslebenszeit hinausgehen werden als *Pending Intents* bezeichnet. Es ist somit möglich einen Intent abzusetzen, welcher erst viel später verarbeitet wird. Ein gutes Beispiel hierfür wäre eine Benachrichtigung zu einer bestimmten Zeit. Hierfür kann man einen Pending Intent absetzen, welcher eine gewisse Aktion (z.B.: die eigene Applikation) benachrichtigt, sobald 5 Stunden ab Applikationsstart vergangen sind.

2.1.6 Services

Services sind Applikationskomponenten, welche im Gegensatz zu Activities im Hintergrund laufen und oftmals langandauernde Berechnungen durchführen (vgl. [34]). Services besitzen kein eigenes User Interface und werden durch Intents oder durch das so genannte *Bindings* gesteuert. Services, welche von einer Applikation gestartet wurden können auch nach Beendigung dieser weiterlaufen und von anderen Applikationen weiter verwendet werden.

Im Folgenden wird der Lifecycle eines Android-Service näher erläutert:

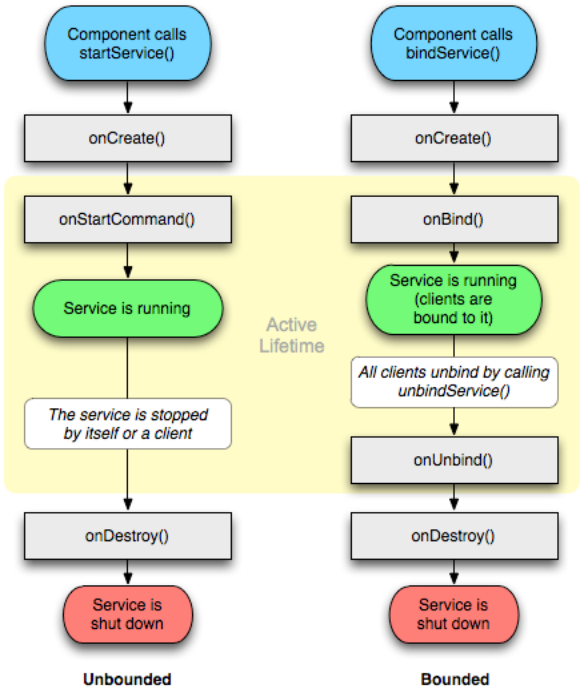


Abbildung 2.5 Service-Lifecycle (siehe [35])

Wie aus Abbildung 2.5 hervorgeht, gibt es 2 Arten von Services. Um welche Art von Service es sich handelt wird beim Starten angegeben.

- *Ungebundene Services* werden oftmals für einmalige Berechnungen bzw. für einmalige Aufgaben verwendet.
- *Gebundene Services* werden beispielsweise verwendet um die angebotene Funktionalität mehrfach zu nutzen bzw. auf diese Einfluss zu nehmen. Ein Beispiel hierfür wäre ein Musikplayer-Service, welcher von mehreren Applikationen gesteuert wird.

Da jeder Service von der Basisklasse Service abgeleitet ist, ist es möglich, auf alle Zustände im Service-Lifecycle zu reagieren, indem man die entsprechenden Methoden überschreibt.

Die einzelnen Zustände im Detail:

- *onCreate:* Wird aufgerufen sobald der Service erstellt wird.
- *onStartCommand:* Wird aufgerufen sobald der Service gestartet wurde.
- *onDestroy:* Wird aufgerufen sobald der Service nicht länger gebraucht wird und folglich zerstört wird.
- *onBind:* Wird aufgerufen sobald sich eine Applikation durch das Kommando *bindService* an den Service bindet.

- *onUnbind:* Wird aufgerufen sobald eine Applikation die bestehende Bindung durch das Kommando *unbindService* freigibt.
- *onRebind:* Wird aufgerufen sobald sich eine Applikation an den Service bindet und schon einmal an den Service gebunden war.

2.1.7 Broadcast Receivers

Broadcast Receiver werden verwendet um beim Auftreten von bestimmten Intents bestimmte Aktionen zu triggern (vgl. [36]). Im Unterschied zu Services sind Broadcast Receiver-Objekte nur so lange gültig, solange die entsprechende Intent-Behandlungsmethode (Methode *onReceive*) läuft. Somit ist es nun möglich auch über Applikationsgrenzen hinweg zu kommunizieren. Um bestimmte Intents zu empfangen ist es nötig sich am System für diese zu registrieren. Dies kann entweder über das Manifest-File (siehe Kapitel 2.1.10) oder programmatisch erfolgen.

Eines der wichtigsten Anwendungsgebiete von Broadcast Reveivern ist das Empfangen von Systemmeldungen. Damit ist es beispielsweise möglich auf Änderungen des Akkuzustandes oder auf die des Telefonstatus zu hören.

Wenn ein vom System generierter Intent abgearbeitet wird, ist es nötig diesen sehr schnell zu bearbeiten, da es ansonsten zu einer „Application Not Responding"-Exception kommen kann. Grund dafür ist, dass das System stets eingabefähig bleiben muss und deshalb nicht „einfrieren" darf. Falls dennoch längere Operationen durchgeführt werden müssen, muss eine entsprechende Threadentkoppelung oder ähnliches implementiert werden.

Es gibt zwei Broadcasttypen, welche empfangen werden können:

- *Normale Broadcasts*, welche asynchron verarbeitet werden und folglich auch in undefinierter Reihenfolge bzw. parallel zur Ausführung kommen.
- *Geordnete Broadcasts,* welche nur von einem Broadcast Receiver zu einer Zeit verarbeitet werden. Somit ist es möglich, dass die einzelnen Broadcast Receiver eigene Ergebnisse an folgende Receiver weitergeben können, welche mit diesen dann weiterarbeiten. Die Reihenfolge der Abarbeitung kann über die Vergabe einer Priorität für jeden Broadcast Receiver angegeben werden.

2.1.8 Content Providers

Content Providers können Daten sammeln bzw. speichern und diese anderen Applikationen zur Verfügung stellen. Es ist also möglich auch über Prozess- bzw. Applikationsgrenzen hinweg Daten auszutauschen (vgl. [37]). Da es derzeit für Android-Applikationen nicht möglich ist auf einen gemeinsamen Speicherbereich zuzugreifen ist diese Funktionalität von großer Bedeutung.

Alle Content Provider sind wiederum von derselben Basisklasse abgeleitet und stellen daher fest vorgegebene Methoden zur Verfügung:

- *onCreate:* Wird aufgerufen, sobald das Content Provider-Objekt erzeugt wurde und lauffähig ist.

  Bei allen folgenden Methoden sollte darauf geachtet werden, dass diese Methoden von mehreren Clients und folglich auch von mehreren Threads aufgerufen werden können und daher unter Umständen eine entsprechende Synchronisierung notwendig ist.

- *query:* Wird verwendet um Daten vom Content Provider abzufragen.
- *insert:* Wird verwendet um neue Daten im Content Provider zu speichern.
- *update:* Wird verwendet um Daten vom Content Provider zu aktualisieren bzw. zu verändern.
- *delete:* Wird verwendet um Daten vom Content Provider zu löschen.
- *getType:* Wird verwendet um den MIME-Typen der im Content Provider gespeicherten Daten abzufragen.

Verschiedene Anwendungsgebiete für die entsprechende Content Provider zur Verfügung gestellt werden sind beispielsweise Telefonkontaktdaten, Medienbibliotheksdaten und Telefonverlaufsdaten.

Um Daten von einem Content Provider abzufragen sind drei Dinge notwendig:

- Ein URI, welcher die Ressource identifiziert
- Die Namen der Datenspalten welche abgefragt werden sollen
- Die Datentypen der abgefragten Datenspalten

Falls ein ganz bestimmter Datensatz abgefragt werden soll, wird ebenfalls noch die ID des Datensatzes benötigt.

2.1.9 Abstrahierte Sensorschnittstelle

Die Android-API stellt eine weitestgehend abstrahierte Sensorschnittstelle zu Verfügung, welche sehr komfortabel verwendbar ist (vgl. [38]). Da in modernen Mobiltelefonen mehrere Sensoren eingebaut bzw. in Verwendung sind, werden folgende Sensoren unterstützt (siehe [39]):

- Beschleunigungssensoren
- Temperatursensoren
- Gravitationssensoren
- Gyroskope
- Lichtsensoren

- Linearbeschleunigungssensoren
- Magnetfeldsensoren
- Luftdrucksensoren
- Näherungssensoren
- Feuchtigkeitssensoren
- Drehrichtungssensor

Die oben erwähnte Abstrahierung wird durch die Klasse *SensorManager* realisiert. Ein Objekt dieser Klasse wird durch das Muster *FactoryMethod* erzeugt, welchem man mit Hilfe einer Enumeration den gewünschten Sensortyp bekanntgibt. An diesem gelieferten *SensorManager*-Objekt kann man nun die entsprechenden Ereignisbehandlungsmethoden für eine Werteänderung des Sensors oder auch für die Genauigkeitsänderung des Sensors registrieren.

Funktionen zur Lokalisierung werden von der Android-API nicht als Sensorfunktionen im engeren Sinne gesehen, da zur Positionsbestimmung nicht nur GPS sondern auch Wifi bzw. Zellenortung herangezogen werden kann. Lokalisierungsfunktionalitäten sind daher in die Klasse *LocationManager* ausgelagert.

2.1.10 Manifest

Das Manifest ist fester Bestandteil jeder Android-Applikation (vgl. [40]). Es stellt die nötigen Metainformationen zur Verfügung um eine Applikation zur Ausführung bringen zu können bzw. stellt wichtige Information für die Ausführung selbst zur Verfügung.

Das Manifest erfüllt folgende Aufgaben:

- Identifizierung des Packagenamens für die Applikation, welcher als „Unique Identifier“ dient.
- Beschreibung der verwendeten Komponenten (Activities, Services, Intents, etc.) und deren „Fähigkeiten“ wie z.B. welche Intents diese verarbeiten können.
- Beschreibung welche Rechte von der Applikation benötigt werden.
- Auflistung der verwendeten Profilingklassen, welche zur Entwicklung benötigt werden.
- Festlegung welcher Prozess die Applikationskomponenten hostet.
- Festlegung des minimalen API-Levels, welcher zur Ausführung benötigt wird.
- Festlegung der benötigten externen Bibliotheken für die Ausführung.
- Festlegung des Einstiegspunkts für die Applikation.

Das Manifest ist eine XML-Datei, welche folgendermaßen aufgebaut ist:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
```

```
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>

        <uses-library />
    </application>
</manifest>
```

Am Beginn des Manifests werden wichtige Metadaten wie z.B. die minimale SDK-Version welche zur Ausführung benötigt wird oder auch die benötigten Rechte festgelegt.

Danach folgen die verwendeten Komponenten bzw. deren Eigenschaften wie Intents, welche sie verarbeiten können und sonstige Metadaten für die jeweilige Komponente. Jede in der Applikation verwendete Activity, jeder verwendete Service, Broadcast Receiver oder auch Content Provider muss in dieser Aufzählung enthalten sein, da die Applikation ansonsten nicht lauffähig ist.

Am Ende des Manifests folgt eine Aufzählung der benötigten externen Bibliotheken.

2.1.11 Sicherheitskonzept

Android verfolgt ein strenges Sicherheitskonzept, welches entsprechende Transparenz für den Benutzer schafft (vgl. [40]). Somit werden bei der Installation sämtliche von der Applikation benötigten Rechte vom *Package Manager* visualisiert und der Benutzer muss diesen zustimmen um die Applikation installieren zu können.

Die von der Applikation benötigten Rechte werden in der Sektion *<uses-permission>* im Manifest (siehe Kapitel 2.1.10) angegeben.

Ist eine bestimmte Berechtigung wie beispielsweise der Zugriff zum Internet nicht gegeben, so kommt es zum Laufzeitfehler der Applikation wenn diese auf das Internet zugreifen möchte.

Im Folgenden folgen einige der wichtigsten Berechtigungen näher erläutert:

- `android.permission.INTERNET:` Wird benötigt um Zugriff auf das Internet zu bekommen. Es ist hierbei nicht näher spezifiziert ob dieser Zugriff per Netzbetreiber oder per Wifi erfolgt.
- `android.permission.ACCESS_COARSE_LOCATION:` Wird benötigt um auf die aktuelle Position, welche sich aus der im Moment eingewählten Zelle des Mobiltelefons bzw. aus umliegenden Wifi-Netzwerken ableiten lässt. Diese Art der Positionsbestimmung ist sehr ungenau.
- `android.permission.ACCESS_FINE_LOCATION:` Wird benötigt um auf die aktuelle GPS-Position zugreifen zu können und erlaubt somit eine sehr genaue Ortung.
- `android.permission.WRITE_EXTERNAL_STORAGE:` Wird benötigt um Daten auf die externe SD-Karte schreiben zu können.
- `android.permission.CAMERA:` Wird benötigt um auf die eingebaute Kamera zugreifen zu können.
- `android.permission.READ_PHONE_STATE:` Wird benötigt um den aktuellen Telefonstatus auslesen zu können.

Eine komplette Auflistung ist in der entsprechenden API-Dokumentation (siehe [41]) zu finden.

2.1.12 Deklaratives UI Design

Das Android-User-Interface kann wie bei anderen modernen Softwareplattformen entweder deklarativ oder prozedural entwickelt werden (vgl. [2], Chapter 3 - Designing the User Interface).

Von prozeduralem Design spricht man, wenn man das User-Interface über entsprechenden Quelltext beschreibt. Deklaratives Design hingegen ist eine Beschreibung des User-Interfaces, bei dem kein Code im engeren Sinne erzeugt wird, sondern die einzelnen Steuerelemente und deren Eigenschaften wie Position und Größe lediglich beschrieben bzw. deklariert werden. Beispiele für deklarative Designtechnologien sind HTML, XAML (Microsoft) oder auch Flex (Adobe).

Der große Vorteil von einem deklarativen Ansatz ist, dass das Design und der Programmcode streng voneinander getrennt sind, was eine Zusammenarbeit von Applikationsentwicklern und GUI-Designern verbessert. Weiters ist es auch möglich, ein Design völlig zu verändern, nachdem die Entwicklungsarbeit abgeschlossen ist, da es lediglich nötig ist, dass der Kontrakt zwischen UserInterface und Implementierung (Eingabe und Ausgabeschnittstellen) eingehalten wird.

In Android wird für die Beschreibung von Benutzerschnittstellen eine XML-Datei verwendet. Diese „*View*“ kann im Programmcode referenziert werden und somit kann auch auf die darin enthaltenen Steuerelemente zugegriffen werden.

Das folgende Beispiel zeigt die Deklaration und die tatsächliche Ansicht einer Benutzerschnittstelle einer kleinen Beispielapplikation:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView android:text="Suche" android:id="@+id/textView1"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"/>
    <EditText android:text="Beschreibung, Teilenummer, ..."
              android:layout_height="wrap_content"
              android:id="@+id/partDesc"
              android:layout_width="match_parent"/>
    <Button android:text="Suche starten!" android:id="@+id/searchButton"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
    <ListView android:layout_height="fill_parent"
              android:id="@+id/listView1"
              android:layout_width="match_parent"/>
</LinearLayout>
```

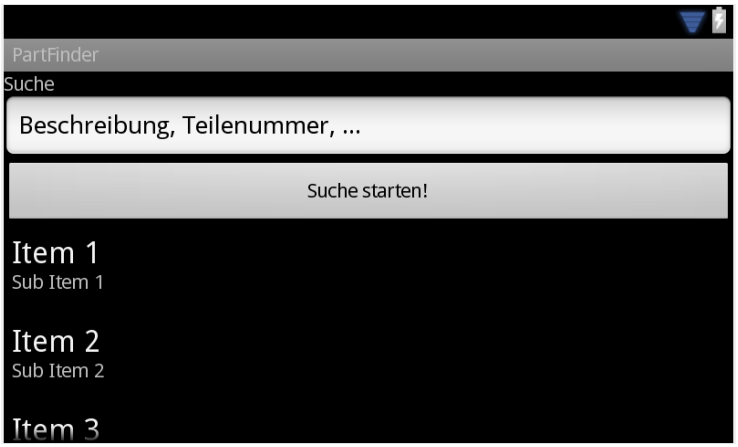


Abbildung 2.6 Deklarativ erzeugte Benutzerschnittstelle

2.1.13 Android-Klassenbibliothek im Vergleich zur Java-Klassenbibliothek

Android Programme werden in Java geschrieben und verwenden den Java 5 SE Sprachumfang (vgl. [2], Appendix A, [5], Kapitel 21). Der geschriebene Quelltext wird mit einem Standard Java-Compiler in Bytecode übersetzt, welcher dann wiederum in Dalvik-Instruktionen bzw. in Dalvik-Bytecode umgesetzt wird (Näheres zum Laufzeitsystem siehe Kapitel 2.1.3).

Anders als beim Sprachumfang, sind die Android-Kernbibliotheken in manchen Bereichen unterschiedlich zu den Jave SE 5 Bibliotheken (siehe [2], Appendix).

Folgende Java-Bibliotheksfunktionen sind in der Android-API enthalten:

java.awt.font	java.beans
java.io	java.lang (nicht vollständig – siehe Unten)
java.math	java.net
java.nio	java.security
java.sql	java.text
java.util	javax.crypto
javax.microedition.khronos	javax.net
javax.sql	javax.sercurity (nicht vollständig – siehe Unten)
javax.xml.parsers	org.w3c.dom
org.xml.sax	

Folgende Java-Bibliotheksfunktionen sind in der Android-API nicht enthalten:

java.applet	java.awt
java.lang.management	java.rmi
javax.accessibility	javax.activity
javax.imageio	javax.management
javax.naming	javax.print
javax.rmi	javax.security.auth.kerberos

javax.security.auth.spi	javax.security.sasl
javax.sound	javax.swing
javax.transaction	javax.xml (ausgenommen javax.xml.parsers)
org.ietf.*	org.omg.*
org.w3c.dom.*	

Folgende Bibliotheken sind zusätzlich enthalten:

org.apache.http	org.json
org.xml.sax	org.xmlpull.v1

2.2 Beschleunigungssensor

2.2.1 Einführung

Ein Beschleunigungssensor, oftmals auch G-Sensor genannt ist ein elektromechanisches Bauteil, welches Beschleunigungskräfte misst. Die Messung erfolgt dabei durch das Ermitteln der Trägheitskräfte, welche auf eine Testmasse wirken (vgl. [13,14,15]). Man kann die Beschleunigungskräfte in zwei Kategorien einteilen:

- Statische Kräfte
- Dynamische Kräfte

Bei den statischen Kräften handelt es sich um Gravitations- bzw. Erdanziehungskräfte, welche immer und im gleichen Ausmaß auf den Sensor einwirken. Diese Erdanziehungskraft, auch Erdbeschleunigung genannt beträgt ca. 9,81m/s^2. Verwendet man nun einen 3-Achs-Sensor, welcher beispielsweise in allen modernen Mobiltelefonen eingesetzt wird, können die Kippwinkel um alle 3 Achsen sehr einfach ermittelt und somit Rückschlüsse auf die Lage im Raum gezogen werden.

Bei den dynamischen Kräften handelt es sich im Wesentlichen um durch Bewegungen oder Vibrationen hervorgerufene Beschleunigungsveränderungen. Diese dynamischen Beschleunigungsverläufe stellen auch die Grundlage des im Kapitel 4 gezeigten Prototyps dar, bei dem die Beschleunigungs- und Bremsbeschleunigungswerte eine Bewertungsgrundlage über den Fahrstil des Lenkers darstellen.

2.2.2 Typen von Beschleunigungssensoren

Es gibt verschiedenste Typen von Beschleunigungssensoren, welche mit verschiedensten Technologien realisiert sind. Die wichtigsten sind jedoch (vgl. [14, 15]):

- *Piezoelektrische Beschleunigungssensoren,* bei welchen ein piezokeramisches Sensorelement, auf welchem eine Testmasse angebracht ist die einwirkenden Beschleunigungsveränderungen in elektrische Signale umwandelt.

- *Kapazitive Beschleunigungssensoren,* bei welchen sich die Kapazität des Sensors durch den Abstand bzw. die Verformung der Kondensatorplatten ergeben(vgl. [16]).
- *Piezoresistive Beschleunigungssensoren,* bei welchen sich der Widerstandswert des Sensors aufgrund der einwirkenden Kräfte ändert.
- *Hall-Effekt Beschleunigungssensoren,* bei welchen sich die Ausgangsspannung des Sensors durch eine Änderung des magnetischen Feldes um den Sensor ändert.
- *Magnetoresistive Beschleunigungssensoren,* bei welchen das Messprinzip ähnlich wie bei Hall-Effekt-Sensoren ist, jedoch sich die Beschleunigungsveränderungen des Sensors durch eine Veränderung des Widerstandswertes manifestiert.
- *MEMS Beschleunigungssensoren,* bei welchen so genannte „Mikro-Elektro-Mechanische-Systeme“ auf einem Siliziumhalbleiter modelliert werden und beispielsweise durch eine Kapazitätsänderung auf dem Halbleiterbaustein auf die Beschleunigungsveränderung rückgeschlossen werden kann. Diese Technologie ist die modernste zur Beschleunigungsmessung und wird beispielsweise in allen State-of-the-Art Mobiltelefonen verwendet.

2.3 Geschwindigkeitsermittlung mittels GPS

2.3.1 Einführung

Unter GPS (*Global Positioning System*) versteht man ein satellitengestütztes Navigationssystem, welches Positions- und Zeitdaten auf der ganzen Welt zur Verfügung stellt (vgl. [17], S. 252). Zum Empfangen dieser Daten ist ein so genannter GPS-Sensor notwendig, welcher die GPS-Signale empfängt und daraus den aktuellen Standort ermitteln kann. Da GPS-Sensoren in nahezu jedem modernen Mobiltelefon eingebaut sind, ist es daher möglich, die aktuelle Position und folglich auch die aktuelle Geschwindigkeit zu berechnen.

2.3.2 Konzept der Positionsermittlung

Um die Position auf der Erdoberfläche ermitteln zu können, werden zunächst drei Satelliten benötigt (vgl. [17], S. 252). Schneidet man nun die drei resultierenden Kugeloberflächen (siehe Abb. 2.1), kann man auf die Position auf der Erdoberfläche (Schnittpunkt 1) rückschließen, da sich die zweite resultierende Position nicht innerhalb der Erdatmosphäre befindet (Schnittpunkt 2).

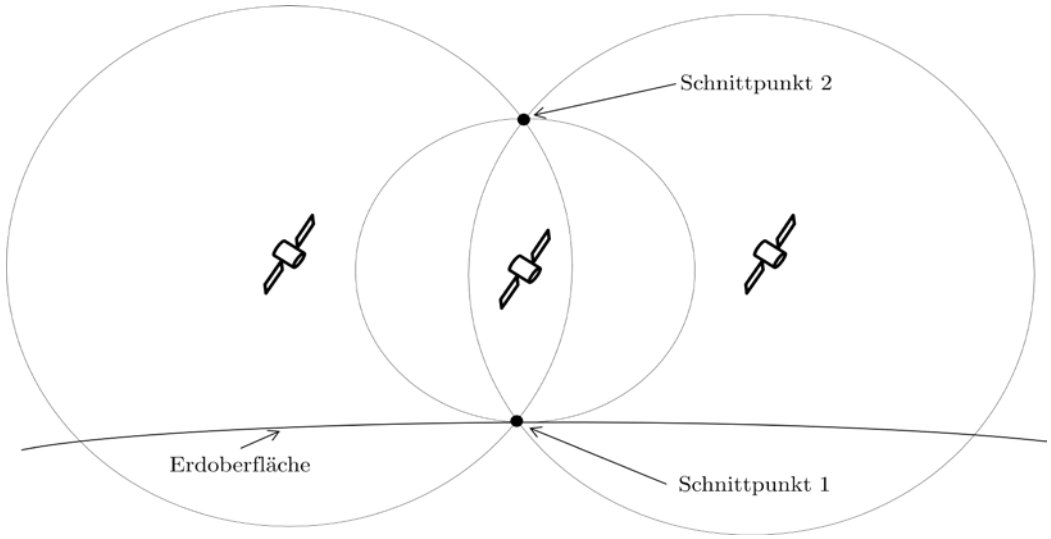


Abbildung 2.7: Prinzip der Positionsbestimmung (siehe [17, S. 253])

Bei diesem Verfahren werden Konzepte wie „Time of Arrival“ (kurz TOA) und „Time Difference of Arrival“ (kurz TDOA) verwendet, welche im Wesentlichen die Laufzeit der Signale zwischen Sender und Empfänger heranziehen, um dann auf die Entfernung schließen zu können(vgl. [17], S. 250ff).

Diese Konzepte beruhen darauf, dass das gesamte System (Sender bzw. Satelliten und Empfänger) die gleiche Systemzeit verwendet, was in der Realität aufgrund der nicht exakten Übereinstimmung der Satellitennavigationssystemzeit mit der Empfängersystemzeit (z.B.: Mobiltelefon) nicht eingesetzt werden kann, da beispielsweise eine Abweichung von nur 1µs einen Unterschied von 300m (Strecke, welche sich aufgrund der Ausbreitung mit Lichtgeschwindigkeit ergibt) machen würde. Um dieses Problem zu lösen, wird ein vierter Satellit hinzugezogen, mit Hilfe dessen eine Ausgleichsberechnung möglich ist und somit auf die Position rückgeschlossen werden kann.

Erweitert man nun dieses Verfahren und zieht zur Genauigkeitsbestimmung noch weitere Satelliten hinzu, so kann bei modernen Mobiltelefonen eine Genauigkeit von bis zu wenigen Metern erreicht werden.

2.3.3 Berechnung der Geschwindigkeit aus Positionsdaten

Da die Geschwindigkeit die erste Ableitung des Weges nach der Zeit ist (vgl. [18, Kap. L2],[17, S. 247]), und man bei einer Updaterate des GPS-Sensors von ca. 1 Sekunde davon ausgehen kann, dass keine extremen Änderungen der aktuellen Fahrtgeschwindigkeit des Fahrzeugs auftreten, ist eine sehr brauchbare Näherung zur Bestimmung der aktuellen Fahrtgeschwindigkeit möglich.

Um nun den Geschwindigkeitswert in Kilometer pro Stunde (kurz km/h) zu erhalten, ist es nötig die empfangenen Positionsdaten in geographischer Längen- und geographischer Breitenangabe in die Einheit Meter umzurechnen. Eine Funktion, welche

diese Umrechnung durchführt wird beispielsweise vom Android-SDK zur Verfügung gestellt und kann folglich direkt verwendet werden(vgl. [19). Bringt man nun die zwischen zwei Samples bestehende Wegdifferenz zur aktuellen Abtastrate bzw. zur Zeitspanne zwischen den letzten beiden Samples ins Verhältnis und korrigiert den Zeitfaktor indem man auf Stunden umrechnet, kann auf die Aktualgeschwindigkeit in der Einheit km/h geschlossen werden.

3 Anforderungen an den Prototypen

Im Rahmen dieser Arbeit gilt es einen Prototypen einer Android-Applikation zu entwickeln, welcher einerseits zum Vermessen der Fahrzeugbeschleunigung- und der Fahrzeuggeschwindigkeitsverläufe dient, andererseits soll der Prototyp im Stande sein, ein im Kapitel 1.2 angeführtes „Realtime-Feedback-System“ zu realisieren, welches dem Lenker unmittelbares Feedback zum aktuellen Fahrstil gibt. Folgende Anforderungen und Use-Cases lassen sich dabei ableiten:

3.1 Funktionale Anforderungen

- *Messung der Beschleunigungsdaten:* Sämtliche Beschleunigungsverläufe, welche während der Fahrt bzw. der Messungsdauer auftreten sollen aufgezeichnet werden.
- *Messung der Geschwindigkeitsdaten:* Sämtliche Geschwindigkeitsverläufe, welche während der Fahrt bzw. der Messungsdauer auftreten sollen aufgezeichnet werden.
- *Filterung der Messdaten:* Die gemessenen Beschleunigungs- und Geschwindigkeitsverläufe sollen entsprechend gefiltert bzw. aufbereitet werden um eine qualitativ hochwertige Weiterverarbeitung zu garantieren. Es sollen also Störsignale, welche beispielsweise durch Schlaglöcher oder Positionssprünge der GPS-Messung auftreten nicht maßgeblich in die Messung mit einfließen und entsprechend ausgeblendet werden.
- *Export der Daten:* Sämtliche Daten (Roh- und Filterdaten) sollen auf die SD-Karte des Gerätes exportierbar sein, um somit eine einfache Weiterverarbeitung zu ermöglichen.
- *Visualisierung der Aktualdaten:* Sämtliche relevanten Aktualdaten sollen bereits während der Messung entsprechend visualisiert werden.
- *Montagewinkelausgleich:* Der gewählte Montagewinkel des Mobiltelefons im Fahrzeug soll entsprechend ermittelt werden und darf auf die Messung keinen Einfluss haben.
- *Rating:* Entsprechend zum aktuellen Fahrstil soll ein Rating erstellt werden, welches es erlaubt eine Aussage über den Fahrstil des Lenker machen zu können.

3.2 Nicht-funktionale Anforderungen

- *Stabilität:* Das Messsystem soll stabil laufen und durch keine Fehler- bzw. Ausnahmesituation im Programm abstürzen.
- *Benutzerfreundlichkeit:* Der Prototyp soll einfach zu bedienen und zu konfigurieren sein. Weiters soll das Starten und das Beenden einer Messung so einfach als möglich sein.
- *Korrektheit:* Mehrere gleiche Messungen sollen auch zu mehreren gleichen Ergebnissen führen.
- *Erweiterbarkeit:* Neue Filtermethoden sollen einfach in das bestehende Programmsystem zu integrieren sein.
- *Portierbarkeit:* Die Applikation soll einfach auf andere Clients zu portieren und zu installieren sein.

3.3 Use-Cases

- *Rating für den Lenker:* Der Lenker soll Informationen zum aktuellen Fahrstil erhalten (farblich bzw. mit Punktesystem), mit Hilfe dessen der Lenker unmittelbar den Fahrstil anpassen und somit auch das Rating beeinflussen kann. Dazu gibt es ein Rating für den Moment und ein Rating für die ganze Fahrt, welches den Durchschnittswert der Einzelratings darstellt.
- *Fahrtdatenmessung für fortgeschrittene Benutzer:* Um die Applikation entsprechend weiterzuentwickeln bzw. um Fahrtanalysen durchführen zu können, ist es nötig entsprechende Schnittstellen und Funktionalitäten für fortgeschrittene Benutzer zu realisieren. Bei diesen Funktionalitäten handelt es sich einerseits um den Zugriff auf die aufgezeichneten Daten und andererseits um die Möglichkeit die aufgezeichneten Daten entsprechend zu filtern.

4 Design und Implementierung

Dieses Kapitel beschreibt die Realisierung des Prototyps ECO.Drive, wessen Anforderungen bereits im Kapitel 3 näher erläutert wurden. Im ersten Unterkapitel wird auf die Softwarearchitektur näher eingegangen. Das zweite Unterkapitel beschäftigt sich mit dem Sensor-Management und mit konkreten Problemlösungsansätzen, welche nötig sind um qualitativ hochwertige Messdaten zu erhalten. Im dritten und letzten Unterkapitel wird näher auf die Visualisierung der ermittelten Daten eingegangen.

4.1 Architektur

4.1.1 Schichtenkonzept

Die Softwarearchitektur des Prototyps baut auf drei Schichten auf, wobei sich die Trennung der einzelnen Schichten an der Verwendung der Daten orientiert (siehe Abb. 4.1).

Abbildung 4.1: Schichtenarchitektur

In der Datenaufzeichnungsschicht werden die Rohdaten des Beschleunigungs- und GPS-Sensors erfasst und entsprechend für die Weiterverwendung in der darüber liegenden Datenverarbeitungsschicht aufbereitet. Die Datenverarbeitungsschicht dient zur Berechnung des eigentlichen Ratings und zur Aufbereitung der Daten für die Datenvisualisierungsschicht. Diese beinhaltet die grafische Benutzerschnittstelle, welche

die Daten nur mehr anzeigt und folglich keine Berechnungen oder Manipulationen an den gelieferten Daten mehr vornimmt.

4.1.2 Domänenmodell

Durch die strikte Trennung in die oben erwähnten drei Schichten kann ein klar strukturiertes Domänenmodell abgeleitet werden (siehe Abb. 4.2), wobei jede Klasse eine klar definierte Aufgabe bzw. Funktion hat.

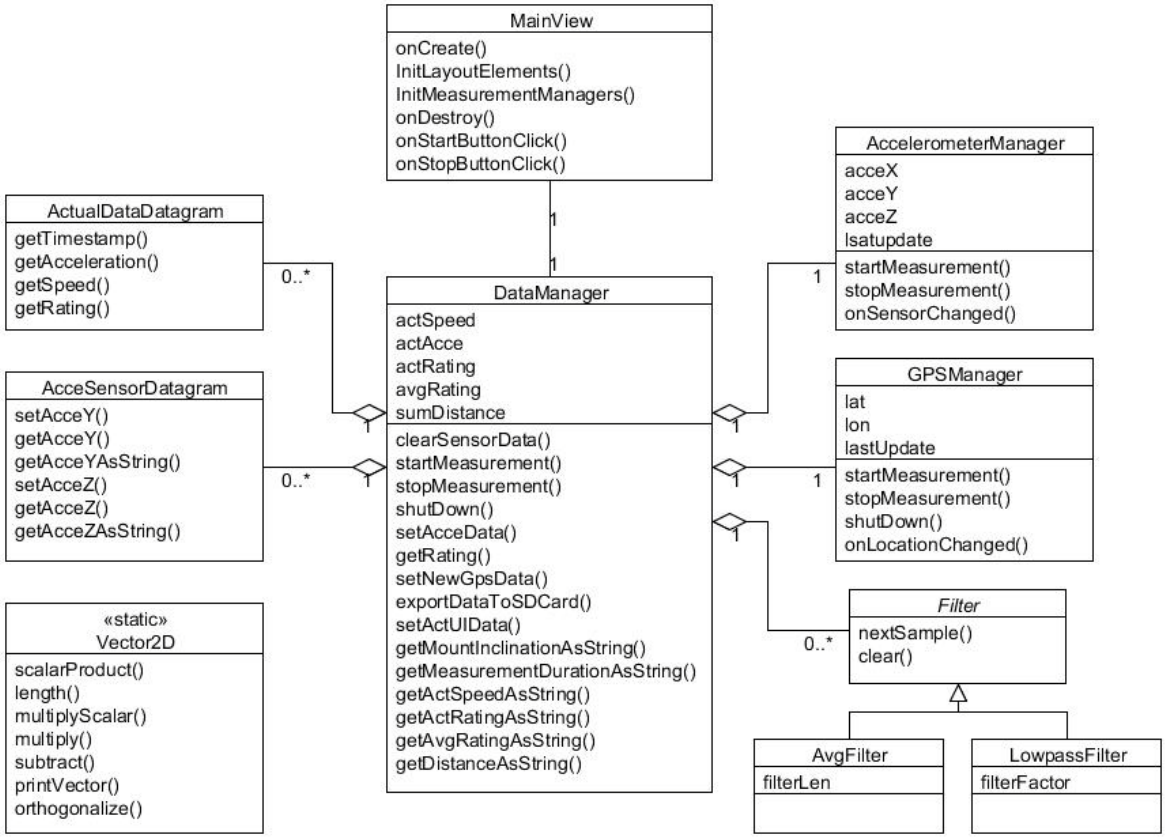


Abbildung 4.2: Domänenmodell

Im Folgenden werden die einzelnen Klassen und Ihre Aufgaben näher erläutert:

- *DataManager:* Diese Klasse stellt eine zentrale Komponente in der Applikation dar. Er stellt die Verbindung zu den Sensormanagern (AccelerometerManager, GPSManager) dar und leitet die gemessenen Rohdaten in die entsprechenden Filter weiter bzw. speichert diese in Rohdatenpuffer, welche zu Diagnosezwecken dienen. Nachdem die gefilterten Daten vorliegen werden diese wiederum in Datenpuffer gespeichert um einen Vergleich zwischen den gemessenen und den bearbeiteten Daten durchführen zu können. Sämtliche Daten können auch exportiert und auf der SD-Karte abgespeichert werden. Das eigentliche Rating, welches dem Benutzer ein Feedback zum Fahrstil gibt wird ebenfalls in dieser Klasse erstellt.

- *AccelerometerManager:* Diese Klasse stellt die Kommunikation mit dem Beschleunigungssensor her und implementiert das *SensorEventListener*-Interface, welches nötig ist um Daten von einem Sensor zu erhalten. Diese Klasse stellt auch die Methoden zum Aktivieren bzw. Deaktivieren der Messung zur Verfügung, was im Grunde bedeutet, dass man sich am *Android-Sensor-Event* an- bzw. abmeldet. Die gemessenen Daten werden ungefiltert an den DataManager weitergereicht, welcher dann die Weiterverarbeitung vornimmt.

- *GPSManager:* Diese Klasse stellt die Kommunikation mit dem GPS-Sensor her und implementiert das *LocationListener*-Interface, welches nötig ist um Positionsupdates bzw. GPS-Sensor-Events zu erhalten. Diese Klasse stellt auch die Methoden zum Aktivieren bzw. Deaktivieren der Messung zur Verfügung, wobei aber im Gegensatz zum *AccelerometerManager* die Events nicht an- und abgemeldet werden, da ansonsten der Neustart der Messung sehr lange dauern würde. Der Grund dafür ist, dass es eine Weile dauert, bis sich der GPS-Sensor initialisiert und bis die Positionsdaten zu Verfügung stehen. Um die Messung dennoch An- bzw. Abschalten zu können, realisiert eine interne Logik das Handling der auftretenden Events und leitet diese erst an den DataManager weiter, sobald die Messung tatsächlich gestartet wurde. Eine Abmeldung vom Eventmechanismus passiert folglich erst beim Beenden der Applikation bzw. nach Aufruf der entsprechenden Methode (siehe Methode *shutDown* im Klassendiagramm). Die aktuelle Geschwindigkeit welche sich aufgrund der Positionsänderung berechnen lässt wird ebenfalls in dieser Klasse ermittelt und an den *DataManager* übertragen.

- *MainView:* Diese Klasse, welche von der Android-Klasse *Activity* abgeleitet ist stellt den Haupteinstiegspunkt in die gesamte Applikation dar und rendert die Benutzerschnittstelle. Nachdem also die Applikation gestartet wurde wird in dieser Klasse eine neue Instanz des DataManagers erstellt, welche sich um die restliche Initialisierung kümmert. Diese Klasse realisiert außerdem das Handling der auftretenden Userevents, welche von zwei Buttons zum Starten und Stoppen der Messung getriggert werden können. Sobald diese Activity gestoppt wird, wird die gesamte Messung gestoppt und die Applikation beendet.

- *AcceSensorDatagram:* Diese Klasse stellt einen Daten-Container für die gemessenen Beschleunigungswerte dar und ermöglicht es somit, Behälter zu bilden, in welche man diese Daten einfach verwalten kann. Diese Klasse kann sowohl Rohdaten als auch gefilterte Datensätze aufnehmen, was dazu führt, dass man eine gute Abstimmung der Filter vornehmen kann.

- *Filter:* Das Interface *Filter* stellt im Wesentlichen die beiden Grundvoraussetzungen für neue Filterimplementierungen zur Verfügung. Diese sind einerseits das Löschen der sich im Filter befindlichen Daten, andererseits das Einfügen eines neuen Samples in den Filter, wobei hier im Gegenzug ein gefiltertes Sam-

ple als Rückgabewert zurückgeliefert werden muss. Somit ist es möglich in jedem Schritt der Verarbeitung (Auftreten von Sensorevents) die Rohdaten in Filterdaten zu konvertieren. Weiters ist ein einfaches Anwenden bzw. Nicht-Anwenden der Filter möglich, da an den entsprechenden Stellen im Programmcode nur der Filteraufruf gemacht oder eben nicht gemacht werden muss. Ein weiterer Vorteil dieser samplebasierten Filterarchitektur stellt das einfache Kaskadieren mehrerer Filterfunktionen, welche beispielsweise mit verschiedenen Grenzfrequenzen agieren dar.

- *AvgFilter:* Diese Klasse implementiert das Interface Filter und ist die konkrete Implementierung eines *Mittelwertfilters*. Der Filter kann beim Instanziieren mit der Anzahl der Filterstufen parametrisiert werden.

- *LowpassFilter:* Diese Klasse implementiert das Interface Filter und ist die konkrete Implementierung eines *Tiefpassfilters* (vgl. [20]). Der Filter kann beim Instanziieren mit einem Filterfaktor, welcher die Glättung des Signals angibt parametrisiert werden.

- *ActualDataDatagram:* Diese Klasse stellt einen Datencontainer für sämtliche berechneten Daten dar und dient dazu, sämtliche Berechnungen einer Fahrt offline nachvollziehen zu können. In diesem Datencontainer befinden sich also Werte wie beispielsweise die Geschwindigkeit, die Gesamtbeschleunigung in Fahrzeugrichtung und auch das aktuell ermittelte Rating.

- *Vector2D:* Diese statische Klasse stellt lediglich mathematische Operationen für zweidimensionale Vektoren zu Verfügung, welche vom DataManager genutzt werden, um beispielsweise die resultierende Beschleunigung in Fahrzeugrichtung zu ermitteln.

4.2 Sensordaten

Im Folgenden wird näher auf die Messung und Aufbereitung der Beschleunigungs- und GPS-Sensordaten eingegangen. Weiters wird die Möglichkeit des Datenlogger-Mechanismus und die Berechnung des Ratings näher erläutert.

4.2.1 Beschleunigungssensor

Da die Beschleunigungssensordaten die Grundlage für die Messung darstellen ist es nötig, diese entsprechend zu verarbeiten. Einerseits ist eine Tiefpassfilterung nötig um nicht fälschlicherweise hochfrequente Störeinflüsse in die Berechnung mit einzuziehen (z.B.: Schlaglöcher oder kleine Messungenauigkeiten). Andererseits müssen die gemessenen Rohdaten in eine zur Fahrtrichtung und zum Montagewinkel angepasste Form gebracht werden um eine korrekte Messung durchführen zu können. Im Folgenden wird auf diese beiden Punkte näher eingegangen:

- *Datenfilterung und Aufbereitung für die Weiterverwendung:* Da hochfrequente Störeinflüsse nicht in die Messung einfließen sollen muss zunächst die Datenqualität der Beschleunigungssensordaten (siehe Abb. 4.4) evaluiert werden und folglich festgestellt werden, welche Frequenzbereiche des Signals für die Messung relevant sind und welche nicht.

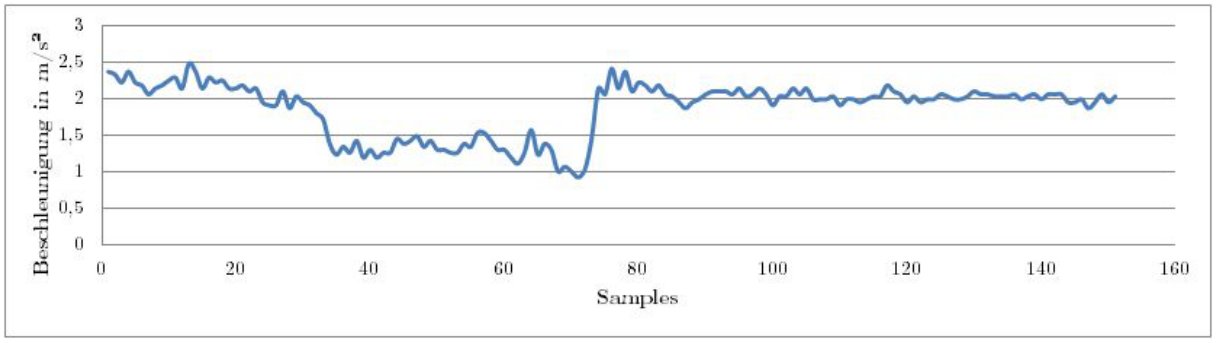


Abbildung 4.3: Rohdatenaufzeichnung in Y-Richtung des Sensors

Dies wird mittels einer *Fast-Fourier-Transformation* (kurz FFT), welche auf die Rohdaten angewendet wird sichtbar gemacht (siehe Abb. 4.3). Man kann nun erkennen, dass es sich bei der Beschleunigung in einem Fahrzeug um einen sehr trägen Vorgang handelt, und daher eine Tiefpassfilterung mit einer Grenzfrequenz von ca. 3-5Hz angemessen ist.

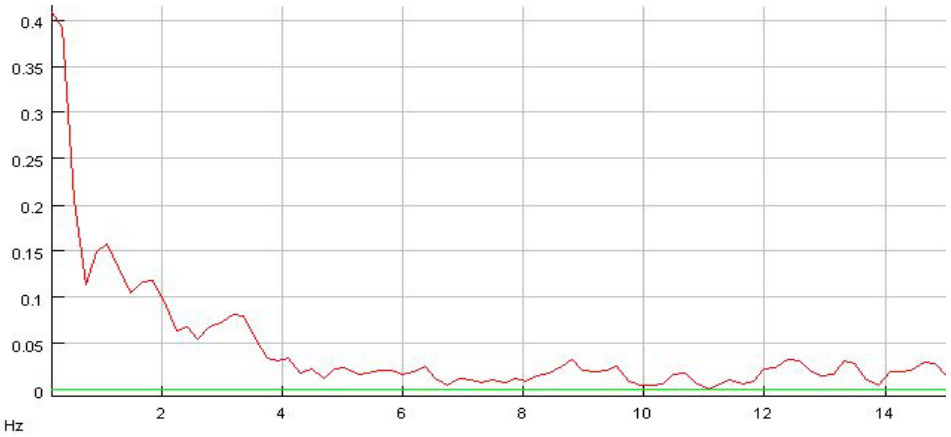


Abbildung 4.4: FFT der Rohdaten

Filtert man die Rohdaten mit einem entsprechenden Tiefpassfilter (siehe Kapitel 4.1.2 – Klasse *LowpassFilter*) kann aus dem ursprünglichen Signal eine qualitativ hochwertige Datenbasis für die weitere Berechnung geschaffen werden (siehe Abb. 4.5).

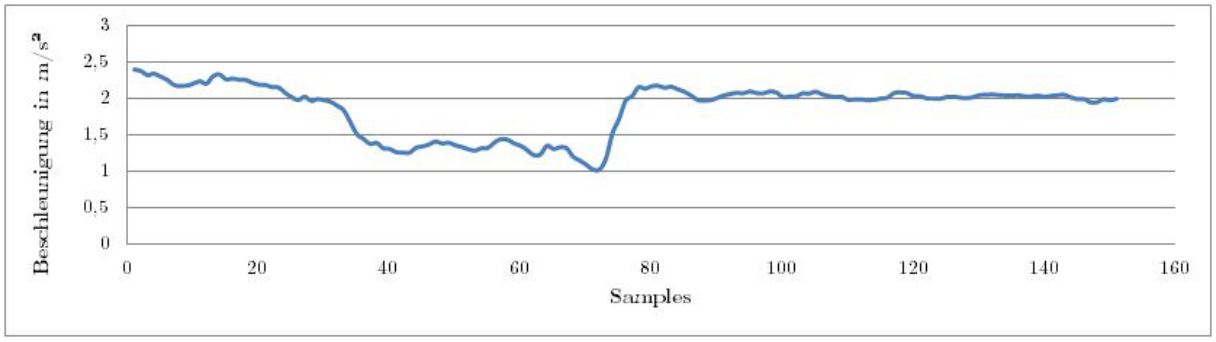


Abbildung 4.5: Filterdatenaufzeichnung in Y-Richtung des Sensors

Nach erneuter FFT der mit einer Grenzfrequenz von 3 Hz gefilterten Daten kann man erkennen, dass die höherfrequenten Störeinflüsse weitestgehend ausgeblendet wurden und die Daten nun bereit für die Weiterverwendung sind.

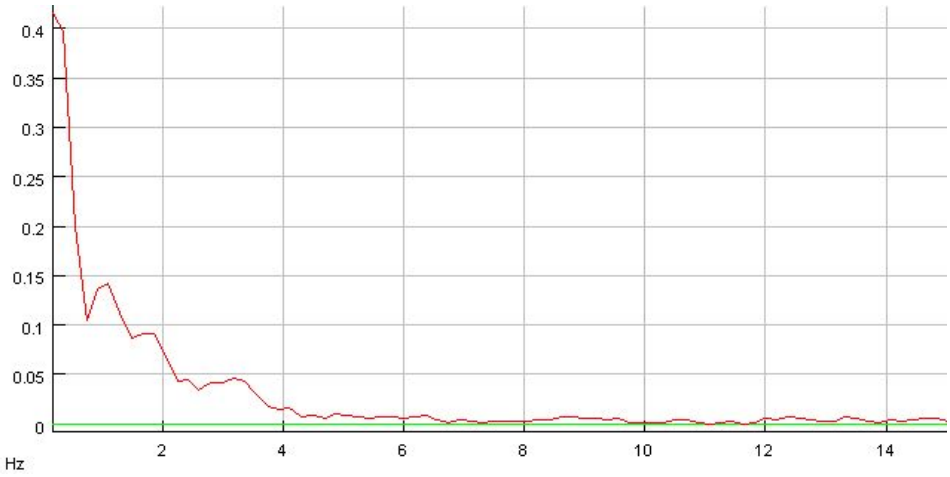


Abbildung 4.6: FFT der gefilterten Daten

- *Auswertung der Beschleunigungssensordaten in Fahrtrichtung:* Da die Daten in X,Y und Z – Koordinaten des Mobiltelefons vorliegen (siehe Abb. 4.6) aber ein Beschleunigungsvektor in Fahrtrichtung benötigt wird, ist es notwendig die Daten in die passende Form zu transformieren.

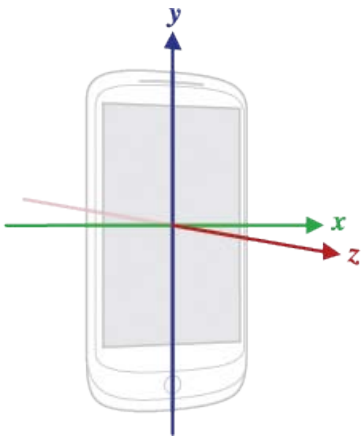


Abbildung 4.7: Koordinatensystem des 3-Achs Beschleunigungssensors (siehe [21])

Um den Beschleunigungsvektor in Fahrzeugrichtung bestimmen zu können und einer Verfälschung der Messergebnisse entgegenzuwirken, müssen folgende Punkte beachtet werden:

- Der Montagewinkel des Mobiltelefons im Fahrzeug darf sich nach dem Starten der Messung nicht mehr verändern.
- Der Start der Messung muss im Stillstand des Fahrzeugs passieren.
- Das Fahrzeug muss beim Start der Messung normal zur Erdoberfläche stehen.
- Die Y-Achse des Mobiltelefons muss parallel zur Hauptfahrtrichtung des Fahrzeugs liegen.

Die Grundlage um schließlich den resultierenden Beschleunigungsvektor in Fahrtrichtung berechnen zu können stellt das Gram-Schmidtsche-Orthogonalisierungsverfahren dar (vgl. [22]), welches verwendet wird um vom aktuellen Messvektor im Kordinatensystem des Beschleunigungssensors auf den Ergebnisvektor im Fahrzeugkoordinatensystem schließen zu können. Für die Messung werden nur die Y- und Z-Komponenten des Beschleunigungsvektors verwendet, da nur diese für die Beschleunigungsverläufe in Fahrtrichtung relevant sind.

Zur Realisierung sind folgende Schritte nötig (siehe Abb. 4.7):

1. Ermitteln des Initialvektors *w1* beim Start der Messung. Dies ist nötig, um auf den Montagewinkel des Mobiltelefons im Fahrzeug schließen zu können.

2. Ermitteln des aktuellen Beschleunigungsvektors *w2* während der Fahrt. Das Problem hierbei ist, dass dieser Vektor nicht am Fahrzeugkoordinatensystem ausgerichtet ist.

3. Projektion von *w2* auf *w1* (mittels Skalarprodukt), was eine Ausrichtung auf das Koordinatensystem des Fahrzeugs bedeutet. Das Resultat ist der Vektor v_p.

4. Subtrahiert man nun den Vektor v_p vom Vektor *w2*, erhält man den Ergebnisvektor *tmpVect*, welcher die resultierende Beschleunigung im Koordinatensystem des Fahrzeuges darstellt.

5. Um das korrekte Vorzeichen des Ergebnisvektors *tmpVect* zu bestimmen, ist es nötig das Skalarprodukt mit einem Vektor in Fahrtrichtung (Vektor *forwardDirHelpVect*) zu bilden.

Der so ermittelte Ergebnisvektor bildet die Grundlage für die weitere Verarbeitung, beispielsweise in der *Ratingfunktion*.

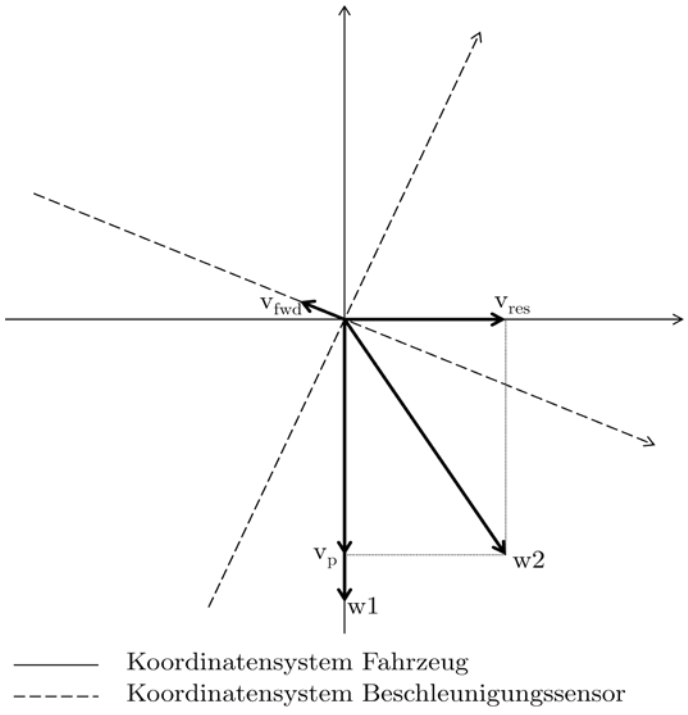


Abbildung 4.8: Koordinatensystemtransformation

4.2.2 GPS-Sensor

Da die aktuelle Geschwindigkeit ebenfalls in das Rating einfließen soll, ist es nötig die GPS-Sensordaten entsprechend zu verarbeiten. Wie ausgehend von Positionsdaten auf die aktuelle Geschwindigkeit geschlossen werden kann wurde bereits im Kapitel 2.3.3 näher erläutert. Um die so ermittelten Geschwindigkeitswerte für die Weiterverarbeitung aufzubereiten wird ein Mittelwertfilter (siehe Klasse *AvgFilter* im Kapitel 4.1.2), welcher das arithmetische Mittel über 5 Stufen berechnet angewandt, da dieser bei einer Updaterate des GPS-Sensors von ca. 1Hz die besten Ergebnisse bezüglich Trägheit und Geschwindigkeitssprüngen erzielt. Diese Geschwindigkeitssprünge treten

auf wenn das Fahrzeug steht bzw. sich sehr langsam bewegt (kleiner 5-10 km/h). Grund für diese Sprünge ist, dass die aktuell ermittelte Position einer gewissen Schwankung unterliegt, welche sich im Stillstand aufgrund der Differenzberechnung viel stärker auswirkt. Während der Fahrt kann diese Ungenauigkeit jedoch vernachlässigt werden.

4.2.3 Logging

Um sämtliche Messdaten auswerten zu können bzw. die Filterdaten mit den Rohdaten vergleichen zu können wurde ein entsprechender Logging-Mechanismus implementiert. Dieser basiert auf der Java-Klasse *LinkedList*, welche eine verkettete Liste darstellt, bei der schnell angefügt werden kann. Da die aufgezeichneten Daten nicht durchsucht werden müssen, sondern lediglich am Ende der Messung exportiert werden ist dies die performanteste Realisierung für diesen Anwendungsfall.

Im Prototyp werden verschiedene Logger angeboten, welche unabhängig voneinander Aktiviert bzw. Deaktiviert werden können. Die zu Verfügung stehenden Logger sind im Folgenden näher erläutert:

- *SensorDataRawBuffer*: Dieser Datenpuffer kann Objekte vom Datentyp *AcceSensorDatagram* (siehe Kapitel 4.1.2) aufnehmen und beinhaltet sämtliche Rohdatensamples, welche während der Messung ermittelt wurden.
- *SensorDataFilteredBuffer*: Dieser Datenpuffer kann Objekte vom Datentyp *AcceSensorDatagram* (siehe Kapitel 4.1.2) aufnehmen und beinhaltet sämtliche gefilterten Samples, welche während der Messung berechnet wurden.
- *ActualDataBuffer*: Dieser Datenpuffer kann Objekte vom Datentyp *ActualDatagram* (siehe Kapitel 4.1.2) aufnehmen und beinhaltet sämtliche errechneten Daten, welche zur Ratingberechnung herangezogen werden.

4.2.4 Rating

Das aktuelle Rating der Messung ergibt sich durch eine Kombination der Beschleunigungs- und Geschwindigkeitswerte, da beispielsweise ein Beschleunigungsvorgang von 0 auf 30 km/h weniger Energie benötigt als ein Beschleunigungsvorgang von 60 auf 90 km/h. Grund dafür ist der zur Geschwindigkeit quadratisch ansteigende Luftwiderstand (vgl. [23]).

Es wird folgende Ratingfunktion verwendet:

$$Rating = \left(\frac{MaxRatingPoints}{MaxAccel} * ActAccel * SpeedPenality\right) + \left(ActSpeed * \frac{MaxRatingPoints}{MaxSpeed}\right)$$

Wobei als maximale Geschwindigkeit (*MaxSpeed*) 150 km/h, als maximale Beschleunigung (*MaxAccel*) 1,3g, als maximale Anzahl von Strafpunkten (*MaxRatingPoints*) 10 und als Geschwindigkeitsstraffaktor (*SpeedPenality*) 10% der aktuellen Geschwindigkeit angenommen werden. Als Aktualwerte für Geschwindigkeit (*ActSpeed*) und

Beschleunigung (*ActAccel*) werden die ermittelten Messwerte herangezogen. Zusätzlich ist das Aktualrating mit der maximalen Anzahl der Ratingpunkte begrenzt. Um eine bessere Vorstellung dieser Zusammenhänge zu bekommen ist diese Funktion im Folgenden graphisch dargestellt (siehe Abb. 4.9). Der Zusammenhang zwischen Geschwindigkeit, Beschleunigung und Rating lässt sich somit sehr einfach ableiten.

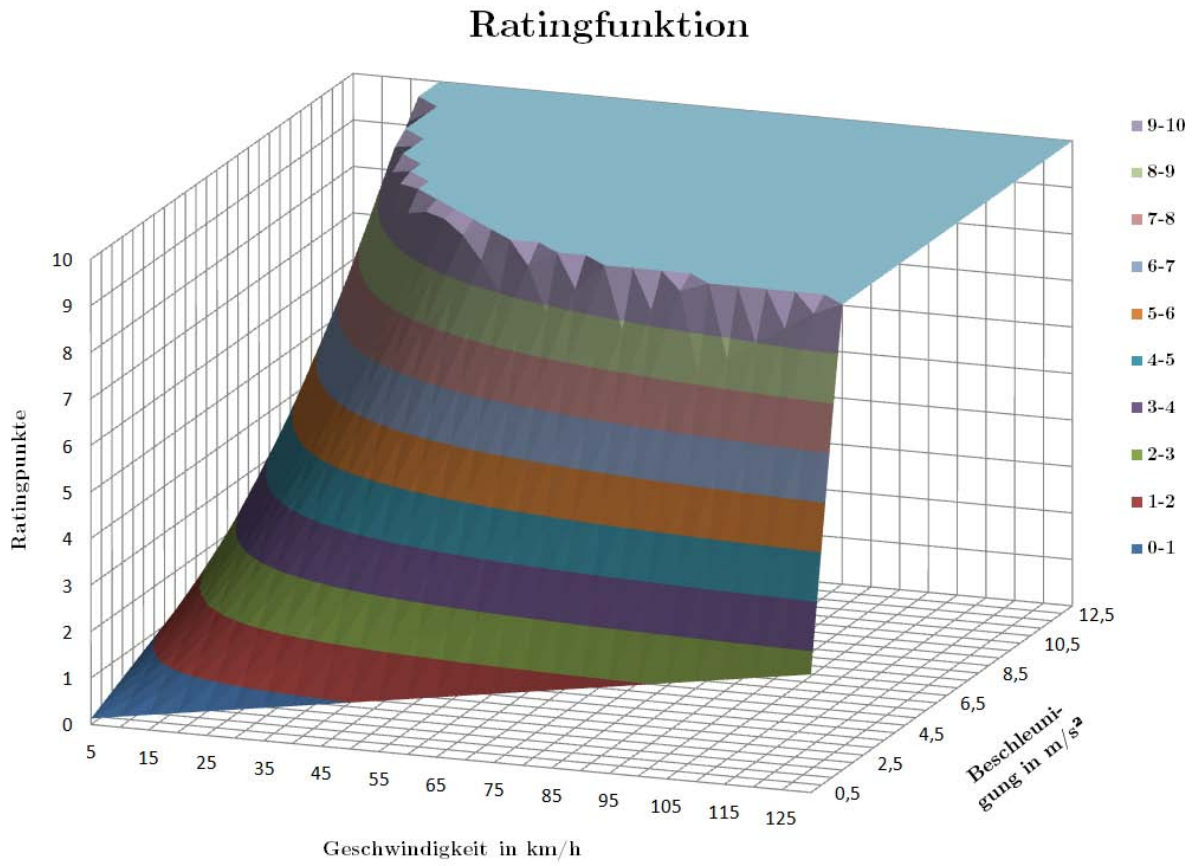


Abbildung 4.9 Graphische Darstellung der Ratingfunktion

Für den Langzeitmesswert bzw. das Langzeitrating wird das arithmetische Mittel der Aktualratings herangezogen.

4.3 Visualisierung

Da es sich um einen Prototyp handelt und der Fokus am Rating bzw. am Datenlogging liegt wurde eine sehr einfache Benutzerschnittstelle realisiert, welche als Interaktion lediglich das Starten und Stoppen der Messung erlaubt. Sämtliche Daten, welche gemessen bzw. berechnet werden, werden ebenfalls entsprechend visualisiert (siehe Abb. 4.9).

Abbildung 4.10 Screenshot - Prototyp ECO.Drive

Das Layout wurde deklarativ über eine entsprechende XML-Datei erzeugt (siehe Kapitel 2.1.11).

5 Evaluierung und Test

Dieses Kapitel beschreibt die am Prototyp ECO.Drive durchgeführten Tests und evaluiert die Android-Plattform aufgrund einiger ausgewählter Kriterien, im Bezug auf die Entwicklung von eigenen Applikationen.

5.1 Test des Protoyps ECO.Drive

Beim Test des Prototyps wurde eine 1 km lange Teststrecke mit einem Mittelklassewagen zwei Mal gefahren und die gemessenen Daten auf die SD-Karte exportiert. Eine dermaßen kurze Teststrecke wurde insofern gewählt, da man aufgrund anderer Verkehrsteilnehmer bei längeren Strecken keine annähernd reproduzierbaren Ergebnisse erhält. Bei der ersten Fahrt wurde darauf geachtet, dass die Fahrt möglichst energieeffiziente Beschleunigungsverläufe aufweist. Bei der zweiten Fahrt wurde auf eine möglichst ineffiziente Fahrweise (starke Beschleunigs- und Bremsmanöver) geachtet. Die Samplerate der Aufzeichnungen beträgt 33 Hz.

5.1.1 Energieeffiziente Testfahrt

Bei dieser Testfahrt (siehe Abbildung 5.1) wurde darauf geachtet, dass die zu fahrende Strecke mit möglichst sanfter Beschleunigung und unter guter Ausnützung der aufgebauten Geschwindigkeit durch Ausrollen lassen des Fahrzeuges gefahren wird. Man erkennt, dass die maximal auftretende Beschleunigung den Wert von ca. 2,5m/s² nicht übersteigt. Das daraus resultierende Summenrating beträgt 5,3.

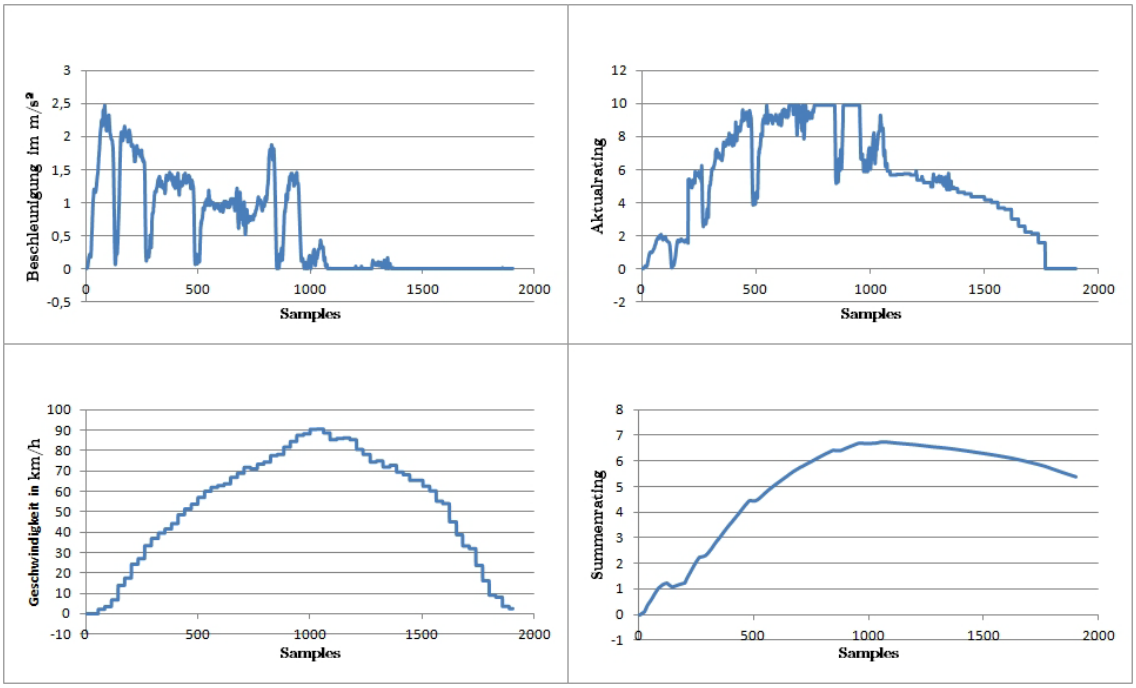


Abbildung 5.1 Testfahrt energieeffizient

5.1.2 Energieineffiziente Testfahrt

Bei dieser Testfahrt (siehe Abbildung 5.2) wurde darauf geachtet, dass die zu fahrende Strecke mit übermäßiger Beschleunigung und unter schlechter Ausnützung der aufgebauten Geschwindigkeit durch Ausrollen lassen des Fahrzeuges gefahren wird. Man erkennt, dass die maximal auftretende Beschleunigung den Wert von knapp 5m/s^2 erreicht. Das daraus resultierende Summenrating beträgt 6,9.

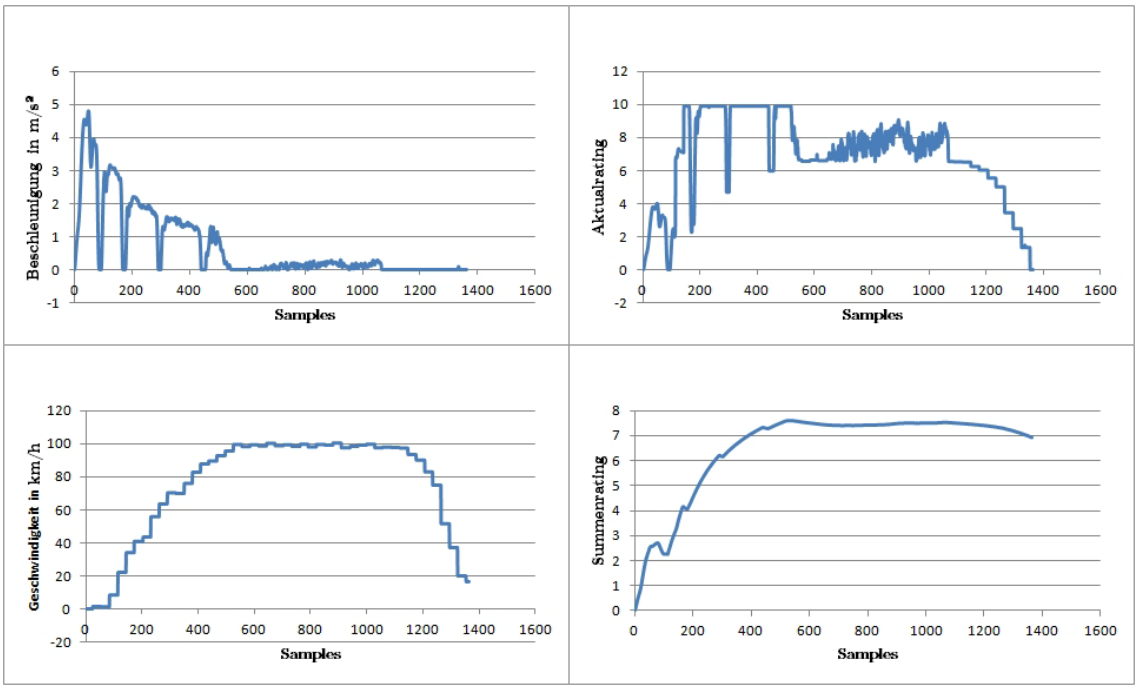


Abbildung 5.2 Testfahrt energieineffizient

5.1.3 Vergleich der Testfahrten

Beim Vergleich der beiden Testfahrten fällt auf, dass die auftretenden Maximalbeschleunigungen fast um einen Faktor zwei differieren. Die kürzere Messungsdauer der energieineffizienten Testfahrt (weniger Samples) kann durch die schnellere Durchschnittsgeschwindigkeit begründet werden. Das Summenrating unterscheidet sich bei dieser kurzen Teststrecke schon um 1,6 Ratingpunkte wobei bei kurzen Strecken mit hohen Geschwindigkeiten die Ratings an sich schlechter sind als bei längeren Strecken mit geringeren Geschwindigkeiten. Grund dafür ist, dass eine hohe Geschwindigkeit an sich ein Indikator für eine energieineffiziente Fahrweise ist.

Falls man eine Testfahrt aufzeichnet, bei der man eine längere Strecke zurücklegt, „festigt“ sich das Summenrating aufgrund der Bildung des arithmetischen Mittels und somit kann eine bessere Aussage über den Fahrstil getroffen werden. Da aber die Messungsergebnisse bei einer kürzeren Strecke anschaulicher sind wurden diese zum Vergleich herangezogen.

5.2 Evaluierung der Android-Plattform

Bei der nachfolgenden Evaluierung werden die wesentlichen Aspekte im Bezug auf die Entwicklung von Applikationen bzw. die Plattformspezifika aus Softwareentwicklersicht bewertet.

5.2.1 Kriterien und Bewertung

- *Design und Erweiterbarkeit:* Da die Android-Plattform modular aufgebaut ist (siehe Kapitel 2.1.2), bietet das Design sehr viele Möglichkeiten. Wie bereits oben erwähnt, ist es möglich, die vorgefertigten Komponenten wie beispielsweise die Telefonanwendung oder auch die Kontaktverwaltung durch eigene Komponenten zu ersetzen. Diese Möglichkeit bietet besonders für Systemintegratoren hohes Potential da sie recht einfach und ohne hohen Aufwand eigene Ideen und Funktionen entsprechend umsetzen können. Diese Tatsache bietet ebenso den Applikationsentwicklern große Möglichkeiten, da diese eigene so genannte *Replacements* oder auch *Replacement Apps (Replacement Applications)* entwickeln können und diese im *Android Market* zum Download anbieten können. Dieses Design ermöglicht es somit zusätzlich zu den „herkömmlichen" *Apps (Applications)* eine ganz andere Variante von *Apps* bereitzustellen, die wesentlich tiefer in das System eingebettet sind.

- *Dokumentation:* Die Dokumentation für sämtliche Funktionalitäten der API ist in Form einer sehr übersichtlichen und gut strukturierten Onlinedokumentation verfügbar. Weiters gibt es auch für die oben erwähnten *API-Levels* entsprechende Anmerkungen, falls diese beispielsweise in neueren *API-Levels* nicht mehr verwendet werden sollen (*deprecated*-Kennzeichnung). Herauszuheben ist auch, dass eine Vielzahl von Demo-Applikationen mit dem entsprechenden Sourcecode und geeigneten Kommentaren verfügbar sind und somit die *Best-Practices* seitens des Herstellers für die Entwicklergemeinde zugänglich gemacht werden. Als Beispiel hierfür könnte man die Verwendung der Bluetooth-, Sensor- oder auch NFC-API nennen, für die sehr gut dokumentierte Demo-Applikationen zur Verfügung stehen.

- *IDE:* Als Entwicklungsumgebung für das Erstellen von Android-Programmen dient die *Eclipse-IDE* (vgl. [42]) mit einem entsprechenden Android-SDK-Plugin (vgl. [43]). Mit dem Plugin wird eine Reihe von Werkzeugen mitgeliefert, welche für den gesamten Entwicklungszyklus benötigt werden (Debugger, Signierungwerkzeug, Deploymentwerkzeug, etc.). Durch die hervorragende Eclipse Integration können alle Vorgänge über die zum Teil über Wizards geführte graphische Benutzeroberfläche durchgeführt werden, was gegenüber kommandozeilenorientierten Werkzeugen einen erheblichen Vorteil bringt, da die Einstiegshürde entsprechend geringer ist. Für automatisierte oder Batch-Vorgänge stehen jedoch entsprechende kommandozeilenorientierten Werkzeuge zur Verfügung.

- *Akzeptanz:* Aufgrund der Programmiersprache Java und der Tatsache, dass die Android-Plattform mit einem Marktanteil von 48% Weltmarktführer (siehe Kapitel 2.1) ist, gibt es eine dementsprechend hohe Akzeptanz der Software-Plattform auf dem Markt. Mittlerweile läuft Android auch schon auf Armbanduhren (vgl. [44]) oder auf *Augmented-Reality* Skibrillen (vgl. [45]). Es ist aufgrund diesen Entwicklungen ein Trend festzustellen, bei dem Android als Plattform für eingebettete Systeme (*Embedded-Systems*) immer attraktiver wird. Dies lässt sich auf die Modularität der Plattform und den Aufbau auf Linux-Systemen zurückführen.

- *Usability:* Aufgrund der oben erwähnten hohen Qualität der Dokumentation und der hohen Anzahl von Demo-Applikationen sinkt die Einarbeitungszeit für das Entwickeln von Android-Applikationen enorm. Auch die Tatsache, dass die weit verbreitete *Eclipse-IDE* mit allen nötigen Werkzeugen zum Einsatz kommt wirkt sich sehr positiv auf den Entwicklungsfortschritt aus, da man sich auf die wesentlichen Teile der Applikationsentwicklung fokussieren kann.

- *Community*: Aufgrund der bereits mehrfach erwähnten geringen Einarbeitungszeit ist auch die Entwicklercommunity entsprechend groß. Ein weiterer Grund für diese Tatsache ist, dass es bei Android möglich ist, eigene *Apps* beispielsweise auf der eigenen Webseite zum Download anzubieten. Es ist ebenfalls möglich *Apps*, welche keinen aktivierten Kopierschutz besitzen direkt auf andere Endgeräte zu kopieren. Zusätzlich zu den genannten Gründen ist auch das bereits erwähnte offene Design der Plattform für viele Entwickler sehr attraktiv, was sich sehr positiv auf eine große und aktive Community auswirkt. Eine Konsequenz daraus ist, dass es viele Supportforen wie beispielsweise *AndroitPIT* (vgl. [46]) oder *Android Forums* (vgl. [47]) für den Informationsaustausch gibt und man oftmals sehr schnelle und kompetente Hilfe von anderen Android-Entwicklern erhält.

6 Zusammenfassung

In diesem Kapitel werden die Ergebnisse der Arbeit abschließend zusammengefasst und ein Ausblick auf die möglichen Erweiterungen bzw. Weiterentwicklungen des Prototyps ECO.Drive gegeben.

6.1 Ergebnisse

Ein Ziel dieser Arbeit war, die Android-Plattform näher zu analysieren und die Plattformspezifika aus der Sicht eines Softwareentwicklers zu erörtern. Dazu wurde darauf eingegangen, wie Android entstanden ist und welche Versionen es bisher gibt. Danach wurde näher auf die Architektur und das Laufzeitsystem eingegangen. Außerdem wurden die wichtigsten API-Konzepte, das Sicherheitskonzept, das Visualisierungskonzept und die Unterschiede zur Java-Klassenbibliothek erläutert. Schließlich wurde die Plattform auf die wichtigsten Kriterien aus Sicht eines Softwareentwicklers evaluiert und die Ergebnisse entsprechend zusammengefasst.

Ein weiteres Ziel dieser Arbeit war, näher auf die in einem mobilen Endgerät zur Verfügung stehenden Sensoren einzugehen. Nach einer Übersicht der zur Verfügung stehenden Sensoren wurde näher auf den Beschleunigungssensor und auf den GPS-Sensor eingegangen. Auch die entsprechenden Konzepte der Messung bzw. Ermittlung der Sensordaten und die grundlegenden Funktionsweisen wie beispielsweise die Berechnung der aktuellen Bewegungsgeschwindigkeit ausgehend von den GPS-Positionsdaten wurden angeführt.

Das letzte Ziel dieser Arbeit war, einen funktionierenden Prototyp eines *Realtime-Feedback-Systems* mit dem Namen ECO.Drive zu entwickeln. Nachdem die Anforderungen näher spezifiziert wurden wurde sehr detailliert auf die Architektur bzw. auf das Design des Prototyps eingegangen. Auch der Signalfluss vom Messen der Rohdaten über Filterfunktionen bis hin zum Ergebnis der Ratingfunktion wurde sehr genau beschrieben. Abschließend wurden konkrete Testfälle, bei dem Echtdaten einer Fahrt von der Applikation verarbeitet wurden dargestellt. Schließlich wird kurz auf die Visualisierung der gemessenen Daten bzw. auf die Visualisierung des Ratings eingegangen.

6.2 Ausblick

Da es sich beim Prototyp ECO.Drive tatsächlich um eine sehr prototypenhafte Anwendung im Bezug auf Benutzerfreundlichkeit und Usability handelt gibt es eine Reihe von Verbesserungsmöglichkeiten und Vorschlägen, mit Hilfe dessen die Anwendung reif für die Verwendung von „herkömmlichen“ Android-Benutzern werden könnte.

Mögliche Punkte hierfür sind:

- *Verbesserung der Ratingfunktion:* Die bestehende Ratingfunktion funktioniert zwar gut aber kann beispielsweise dahingehend adaptiert werden, dass die gefahrene Strecke in das Ratingergebnis mit einfließt um die Problematik von kurzen Strecken, bei denen duch kurze aber starke Beschleunigungen das Summenrating sehr stark nach Oben „gedrückt" wird besser auszugleichen.

- *Konfiguration des Fahrzeugtyps:* Da beispielsweise ein LKW nie die Beschleunigungsmaxima eines PKWs erreichen kann würde ein Konfigurationsdatum, welches den maximalen Beschleunigungswert der gewählten Fahrzeugart entsprechend einstellt für eine bessere Skalierung der Ratingfunktion sorgen.

- *Verbessertes User-Interface:* Da das bestehende Userinterface lediglich die gemessenen Daten in Form von Zahlenwerten darstellt, könnte das Userinterface entsprechend verbessert werden, indem beispielsweise Farbtöne während der Fahrt Rückschlüsse auf die momentane Energieeffizienz ziehen lassen. Somit wäre es auch für den Fahrzeuglenker einfacher zu reagieren, da unter Umständen die Konzentration zu sehr auf das Lesen des Ratings anstatt auf den Straßenverkehr gelenkt wird.

- *Pause-Funktion:* Im Moment gibt es keine Möglichkeit die Aufzeichnung zu pausieren und zu einem späteren Zeitpunkt wieder fortzusetzen. Diese Funktionalität wäre aber sehr wichtig, da Pausen gerade bei einer längeren Autofahrt durchaus vorkommen und diese auch berücksichtigt werden sollten.

- *Anbindung an soziale Netzwerke:* Um eine breitere Masse vom energieeffizienten Fahrstil zu überzeugen könnte man eine Möglichkeit realisieren, mit Hilfe deren man das Rating, welches man an einem bestimmten Streckenabschnitt erzielt hat beispielsweise auf *Facebook* oder *Google+* zu veröffentlichen. Dies könnte zu einem Energiespar-Wettbewerb der Nutzer führen, welcher sich sehr positiv auf unsere Umwelt auswirken würde.

- *Kartenansicht:* Zusätzlich zum Rating könnte man seine Autofahrten in Form eines Fahrtenbuchs auch mit Hilfe einer Kartenkomponente visualisieren und auch dort die Ratingergebnisse entsprechend einzeichnen. Eine farbliche Darstellung würde somit am ersten Blick erkennen lassen, auf welchen Streckenabschnitten es noch Verbesserungspotential des Fahrstils gibt.

- *Verkehrsdatenauswertung:* Durch die zur Verfügung stehenden Fahrtdaten könnte man die Verkehrssituation in bestimmten Regionen erfassen und für weitere Verarbeitungen verwenden. Ein Beispiel wäre ein Ampelmodell für eine Stadt zu erstellen, welches mit Hilfe einer eventuellen Weiterverarbeitung entsprechend optimiert werden könnte.

7 Literaturverzeichnis

[1] Mark L. Murphy. (CommonsWare, LLC). (2009). *The Busy Coder's Guide to Android Development*. 2nd Edition.

[2] Ed Burnette. (Pragmatic Programmers, LLC). (2010). *Hello, Android – Introducing Google's Mobile Development Platform.* 3rd Edition.

[3] Sayed Y. Hashimi and Satya Komatineni. (Apress). (2009). *Pro Android.*

[4] Reto Meier. (Wiley Publishing, Inc.). (2009). *Professional Android Application Development.*

[5] Arno Becker, Marcus Pant. (dpunkt.verlag). (2009). *Android - Grundlagen der Programmierung.*

[6] Keith Barry. (2011). *Cars Don't Waste Fuel. Drivers Waste Fuel.* Verfügbar im Internet unter http://www.wired.com/autopia/2011/09/uc-riverside-cert-study-fuel-efficiency/, zuletzt zugegriffen am 19.9.2011

[7] Statistik Austria. (2011). *Pendler und Pendlerinnen.* Verfügbar im Internet unter http://www.statistik.at/web_de/statistiken/bevoelkerung/volkszaehlungen_registerzaehlungen/pendler/index.html, zuletzt zugegriffen am 20.09.2011

[8] Umweltbundesamt Deutschland. (2009). *Daten zum Verkehr.* Verfügbar im Internet unter http://www.umweltdaten.de/publikationen/fpdf-l/3880.pdf, zuletzt zugegriffen am 20.09.2011

[9] Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit Deutschland. (2007). *Verkehr und Umwelt – Herausforderungen.* Verfügbar im Internet unter http://www.bmu.de/verkehr/herausforderung_verkehr_umwelt/doc/40760.php, zuletzt zugegriffen am 20.09.2011

[10] Zeitung SÜDOSTSCHWEIZ.CH. (2011). *Verkehr nimmt weiter zu – vor allem auf der A53.* Verfügbar im Internet unter http://www.suedostschweiz.ch/vermischtes/verkehr-nimmt-weiter-zu-%E2%80%93-vor-allem-auf-a53, zuletzt zugegriffen am 20.09.2011

[11] CO_2Handel.de. (2011). *Klimawandel? – Studie zeigt Trend zu immer mehr PS unter der Haube.* Verfügbar im Internet unter http://www.co2-handel.de/article341_16806.html, zuletzt zugegriffen am 20.09.2011

[12] areamobile.de. (2011). *Android erobert fast 50 Prozent weltweiten Marktanteil.* Verfügbar im Internet unter http://www.areamobile.de/news/19389-canalys-android-erobert-fast-50-prozent-weltweiten-marktanteil, zuletzt zugegriffen am 21.09.2011

[13] DimensionEngineering.com. *A beginner's guide to accelerometers.* Verfügbar im Internet unter http://www.dimensionengineering.com/accelerometers.htm, zuletzt zugegriffen am 8.10.2011

[14] SENSR.com. *Practical guide to Accelerometers.* Verfügbar im Internet unter http://www.sensr.com/pdf/practical-guide-to-accelerometers.pdf, zuletzt zugegriffen am 8.10.2011

[15] Wikipedia. *Beschleunigungssensor.* Verfügbar im Internet unter http://de.wikipedia.org/wiki/Beschleunigungssensor, zuletzt zugegriffen am 8.10.2011

[16] Wikipedia. *Kapazitiver Sensor.* Verfügbar im Internet unter http://de.wikipedia.org/wiki/Kapazitiver_Sensor, zuletzt zugegriffen am 8.10.2011

[17] Jörg Roth. (dpunkt.verlag). (2002). *Mobile Computing – Grundlagen, Technik, Konzwpte.* Erste Auflage.

[18] K. + R. Gieck. (Gieck Verlag). (1995). *Technische Formelsammlung.* 30. Deutsche Auflage 1995. 75. Gesamtauflage

[19] Developer.android.com. *Location. Methode „distanceBetween".* Verfügbar im Internet unter http://developer.android.com/reference/android/location/Location.html, zuletzt zugegriffen am 28.10.2011

[20] eNotes|Study smarter. *Low-pass filter.* Verfügbar im Internet unter http://www.enotes.com/topic/Low-pass_filter, zuletzt zugegriffen am 05.11.2011

[21] Developer.android.com. *SensorEvent.* Verfügbar im Internet unter http://developer.android.com/reference/android/hardware/SensorEvent.html, zuletzt zugegriffen am 05.11.2011

[22] Wurzelzieher Mathepedia. *Gram-Schmidtsches Orthogonalisierungsverfahren.* Verfügbar im Internet unter http://www.mathepedia.de/Gram-Schmidtsches_Orthogonalisierungsverfahren.aspx, zuletzt zugegriffen am 05.11.2011

[23] www.michael-bosch.net. *Berechnung der Fahrwiderstände.* Verfügbar im Internet unter http://home.foni.net/~michaelbosch/auto/economic/calculat.htm, zuletzt zugegriffen am 07.11.2011

[24] web.archive.org. *Android is now available as open source.* Verfügbar im Internet unter http://web.archive.org/web/20090228170042/http://source.android.com/posts/opensource, zuletzt zugegriffen am 29.12.2011

[25] Open Handset Alliance. *Open Handset Alliance FAQ.* Verfügbar im Internet unter http://www.openhandsetalliance.com/oha_faq.html, zuletzt zugegriffen am 29.12.2011

[26] ZDNet. *Android erreicht 48 Prozent Marktanteil im zweiten Quartal*. Verfügbar im Internet unter http://www.zdnet.de/news/41555363/android-erreicht-48-prozent-marktanteil-im-zweiten-quartal.htm, zuletzt zugegriffen am 29.12.2011.

[27] Android.com. Branding Guidelines. Verfügbar im Internet unter http://www.android.com/branding.html, zuletzt zugegriffen am 29.12.2011

[28] Developer.android.com. *Platform Versions*. Verfügbar im Internet unter http://developer.android.com/resources/dashboard/platform-versions.html, zuletzt zugegriffen am 30.12.2011.

[29] Developer.android.com. *Android API Levels*. Verfügbar im Internet unter http://developer.android.com/guide/appendix/api-levels.html, zuletzt zugegriffen am 30.12.2011.

[30] Developer.android.com. *Build.VERSION_CODES*. Verfügbar im Internet unter http://developer.android.com/reference/android/os/Build.VERSION_CODES.html, zuletzt zugegriffen am 30.12.2011.

[31] Developer.android.com. *SystemArchiteture*. Verfügbar im Internet unter http://developer.android.com/images/system-architecture.jpg, zuletzt zugegriffen am 01.01.2012.

[32] Developer.android.com. *Activity Lifecycle*. Verfügbar im Internet unter http://developer.android.com/images/activity_lifecycle.png, zuletzt zugegriffen am 02.01.2012.

[33] Developer.android.com. *Intents and Intent Filters*. Verfügbar im Internet unter http://developer.android.com/guide/topics/intents/intents-filters.html, zuletzt zugegriffen am 02.01.2012.

[34] Developer.android.com. *Services*. Verfügbar im Internet unter http://developer.android.com/guide/topics/fundamentals/services.html, zuletzt zugegriffen am 02.01.2012.

[35] Developer.android.com. *Service Lifecycle*. Verfügbar im Internet unter http://developer.android.com/images/service_lifecycle.png, zuletzt zugegriffen am 02.01.2012.

[36] Developer.android.com. *Broadcast Receiver*. Verfügbar im Internet unter http://developer.android.com/reference/android/content/BroadcastReceiver.html, zuletzt zugegriffen am 02.01.2012.

[37] Developer.android.com. *Content Providers*. Verfügbar im Internet unter http://developer.android.com/guide/topics/providers/content-providers.html, zuletzt zugegriffen am 03.01.2012.

[38] Developer.android.com. *SensorManager*. Verfügbar im Internet unter http://developer.android.com/reference/android/hardware/SensorManager.html, zuletzt zugegriffen am 03.01.2012.

[39] Developer.android.com. *Sensor*. Verfügbar im Internet unter http://developer.android.com/reference/android/hardware/Sensor.html, zuletzt zugegriffen am 03.01.2012.

[40] Developer.android.com. *The AndroidManifest.xml File*. Verfügbar im Internet unter http://developer.android.com/guide/topics/manifest/manifest-intro.html, zuletzt zugegriffen am 03.01.2012.

[41] Developer.android.com. *Manifest.permission*. Verfügbar im Internet unter http://developer.android.com/reference/android/Manifest.permission.html, zuletzt zugegriffen am 04.01.2012.

[42] Eclipse.org. *Eclipse Foundation Website.* Verfügbar im Internet unter http://www.eclipse.org/, zuletzt zugegriffen am 27.02.2012.

[43] Developer.android.com. *Android SDK.* Verfügbar im Internet unter http://developer.android.com/sdk/index.html, zuletzt zugegriffen am 27.02.2012.

[44] imwatch.it. *I'm Watch Website.* Verfügbar im Internet unter http://www.imwatch.it/at-de/, zuletzt zugegriffen am 27.02.2012.

[45] TheGuardian. *Recon Instruments aims to augment reality of skiing with head-up goggles.* Verfügbar im Internet unter http://www.guardian.co.uk/technology/blog/2011/jan/09/ces-2011-recon-gps-augmented-reality-goggles, zuletzt zugegriffen am 27.02.2012.

[46] AndroidPIT. *Android Forum.* Verfügbar im Internet unter http://www.androidpit.de/de/android/forum, zuletzt zugegriffen am 27.02.2012.

[47] androidforums.com. *Android Forums.* Verfügbar im Internet unter http://androidforums.com/, zuletzt zugegriffen am 27.02.2012.

8 Abbildungsverzeichnis