FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Mobiles Lifelogging auf der Android-Plattform

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Visual Computing

eingereicht von

Roman Hochstöger
Matrikelnummer 0627154

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Univ.Prof. Dipl.-Ing. Dr. Christian Breiteneder
Mitwirkung: Dipl.-Ing. Dr. Matthias Zeppelzauer

Wien, 20.03.2013

(Roman Hochstöger) (Betreuer)

Erklärung

Roman Hochstöger
Mayerlinger Straße 276
A-2534 Alland

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

Wien, 20.03.2013 _____

(Roman Hochstöger)

Abstract

This thesis deals with the design and development of a lifelogging architecture based on the Android platform. The aim is to realize a prototype called Android Lifelogging (ALL), which enables the autonomous recording of personal multimedia data. By a dynamic configuration, ALL is adaptable to many application scenarios. The collected data such as photos, SMS messages or phone calls can be directly exchanged into common formats. A visualization of the collected data is also supported.

The basic ideas and procedures of lifelogging systems are explained. Technologies and practices that are relevant for indexing of Lifelogging data are explained. Multimodal approaches, a comparison of existing approaches and commercial Lifelogging systems are shown. The requirements of ALL, whose architecture and basic concepts are explained in detail. The extensive configuration, called Lifelogging Description Language (LLDL) is presented. Finally, the findings are interpreted and visualized. Experiments show the accuracy of data collection.

Kurzfassung

Diese Diplomarbeit beschäftigt sich mit dem Entwurf und der Entwicklung einer auf Android basierten Lifeloggingarchitektur. Das Ziel ist die Realisierung eines Prototyps, genannt Android Lifelogging (ALL), der eine automatische Erfassung multimedialer Benutzerdaten ermöglicht. Durch eine dynamische Konfiguration soll der Prototyp an möglichst viele Anwendungsszenarien anpassbar sein. Mithilfe einer geeigneten Visualisierung können die erfassten Daten dargestellt werden. Der Austausch sämtlicher Daten wie Fotos, SMS-Nachrichten oder Telefongespräche in gängige Formate wird unterstützt.

Die Entstehung und die grundlegenden Ideen von mobilen Lifeloggingsystemen vorgestellt. Technologien und Vorgehensweisen, die für die Indizierung von Lifeloggingdaten relevant sind, werden erklärt. Multimodale Ansätze, ein Vergleich existierender Ansätze und einige kommerzielle Lifeloggingsysteme werden erörtert. Die Anforderungen von ALL, sowie dessen Architektur und grundlegende Konzepte werden näher erläutert. Die umfassende Konfiguration, realisierbar mithilfe der Lifelogging Description Language (LLDL), wird vorgestellt. Sämtliche Ergebnisse von ALL werden betrachtet. Experimente ermitteln die Genauigkeit der Datenerfassung.

Danksagung

Menschen empfinden das Verstreichen der Zeit als subjektiv. Diese Arbeit untersucht Methoden und Werkzeuge, die einer Erfassung von Momenten ermöglichen. Schöne Momente vergehen meist zu schnell, hingegen verstreichen unangenehme Erlebnisse oft zu langsam. In diesem Sinne möchte ich mich für all die schönen bisherigen Momente meines Lebens bedanken. Besonders danke ich meiner Freundin Stefanie, die immer an mich glaubte, mir viel Kraft gegeben hat, und mir half, meine Träume zu erfüllen. Meiner Mutter Angelika und meinem Vater Michael danke ich für die gute Erziehung und breite Unterstützung. Bei meinen Großeltern möchte ich mich ebenfalls für die Inspiration und Förderung meiner Ideen bedanken. Mein Großvater Simon förderte schon in meiner frühen Kindheit meine mechanische Geschicklichkeit durch die Konstruktion verschiedener mechanischer Apparate. Heinrich führte mich in meiner Kindheit in das Entwerfen und Reparieren elektrischer Schaltungen ein. Meinen Tanten und Onkeln danke ich ebenfalls vielmals für ihr Engagement. Besonders Tante Silvia möchte ich für die Förderung meiner Genauigkeit und Onkel Thomas für die Inspiration und meinen ersten PC danken. Ebenfalls bedanke ich mich vielmals bei Univ. Prof. Dipl.-Ing. Dr. Christian Breiteneder und Dipl.-Ing. Dr. Matthias Zeppelzauer, die mir die Möglichkeit zur Erforschung von Lifeloggingsystemen und den Besuch der ACM-Multimedia 2012 in Nara ermöglichten. Es gibt noch eine Vielzahl anderer Menschen, denen ich danken möchte, zusammenfassend bedanke ich mich somit bei all jenen Menschen, die mein Sein beeinflusst haben.

Abkürzungsverzeichnis

ADT	Android Development Toolkit
AR	Augmented Reality
GLOG	Lifelong Cyborglog
GLOGGING	Cyborg Logging
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
HMM	Hidden Markov Model
JSON	JavaScript Object Notation
LDR	Light Dependent Resistor
LLDL	Lifelogging Description Language
MEMEX	Memory Extender
MVC	Model View Controller
PDA	Personal Digital Assistant
SIFT	Scale-Invariant Feature Transform
SQL	Structured Query Language
SURF	Speeded Up Robust Features
SDK	Software Development Kit
SSL	Secure Sockets Layer
WMMS	Wearable Mobility Monitoring System
WWW	Wearable Wireless Webcam
XML	Extensible Markup Language

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Prinzip . . . 4
- 1.2 Anwendungsgebiete . . . 5
- 1.3 Aufgabenstellung und Ziele . . . 8
- 1.4 Herausforderungen . . . 8
- 1.5 Organisation der Arbeit . . . 9

2 Hintergrund **11**
- 2.1 Kriterien für Lifeloggingsysteme . . . 12
- 2.2 Entstehung von Lifelogging . . . 12
  - 2.2.1 Steve Mann . . . 13
  - 2.2.2 Forget Me Not . . . 18
  - 2.2.3 MyLifeBits . . . 20
- 2.3 Ortsbestimmung . . . 21
  - 2.3.1 Global . . . 21
  - 2.3.2 Lokal . . . 21
- 2.4 Nachverarbeitung und Visualisierung . . . 23
  - 2.4.1 Indizierung durch inhaltliche Analyse . . . 23
  - 2.4.2 Visualisierung . . . 27
  - 2.4.3 Anonymisierung . . . 27

3 Lifelogging Ansätze **31**
- 3.1 Auditiver Ansatz . . . 32
  - 3.1.1 Memory Prosthesis . . . 32
- 3.2 Visuelle Ansätze . . . 32
  - 3.2.1 StartleCam . . . 32
  - 3.2.2 SenseCam . . . 33
  - 3.2.3 Project Glass . . . 34
- 3.3 Audiovisuelle Ansätze . . . 35
  - 3.3.1 Aizawa . . . 35
  - 3.3.2 iGlogger . . . 36
  - 3.3.3 BlackBerry WMMS . . . 36
- 3.4 Biofeedback Ansätze . . . 37

3.4.1 Gesundheitsüberwachung . . . 37
3.4.2 Berührungsbasiertes Lifelogging . . . 38
3.5 Lifelogging Frameworks . . . 39
3.5.1 My Experience . . . 39
3.5.2 UbiqLog . . . 40
3.5.3 Zusammenfassung / Vergleich existierende Ansätze . . . 42

4 Kommerzielle Systeme **45**
4.1 Hardware . . . 46
4.1.1 Vicon Revue . . . 46
4.1.2 Autographer . . . 46
4.1.3 Memoto . . . 46
4.2 Software . . . 47
4.2.1 Lifelapse . . . 47
4.2.2 Replay My Day . . . 47
4.2.3 Funf . . . 47

5 Android Lifelogging (ALL) **49**
5.1 Anforderungsanalyse . . . 50
5.1.1 Charakterisierung . . . 51
5.2 Anwendungsfälle . . . 52
5.3 Architektur und grundlegende Konzepte . . . 53
5.3.1 Klassendiagramm . . . 54
5.4 Datenbank . . . 55
5.4.1 Schema . . . 55
5.5 Webservice . . . 57
5.6 Lifelogging Description Language (LLDL) . . . 58
5.6.1 Locations . . . 59
5.6.2 Profiles . . . 60
5.6.3 Events . . . 64
5.6.4 Annotations . . . 68
5.6.5 Event, Src und Subscriber Übersicht . . . 70
5.7 Synchroner Trigger . . . 71
5.8 Asynchroner Trigger . . . 72
5.8.1 ButtonSensor . . . 72
5.8.2 CallSensor . . . 73
5.8.3 FileObserver . . . 74
5.8.4 SmsSensor . . . 75
5.9 Sensoren . . . 76
5.9.1 AccelerationSensor . . . 77
5.9.2 AudioSensor . . . 78
5.9.3 BarometerSensor . . . 79
5.9.4 BrightnessSensor . . . 81
5.9.5 CameraSensor . . . 81

5.9.6 GpsSensor/NetSensor . . . 81
5.9.7 GyroscopeSensor . . . 82
5.9.8 RotationSensor . . . 83
5.10 Ablauf synchroner und asynchroner Datenerfassung . . . 85
5.11 Entwicklungsumgebung . . . 85
5.11.1 Smartphoneanforderungen . . . 87

6 Ergebnisse **89**
6.1 ALL Benutzeroberfläche am Smartphone . . . 90
6.2 Webinterface . . . 93
6.3 Datenbankschnittstelle . . . 97
6.4 Visualisierung von Lifelogging Daten . . . 100
6.4.1 Landkarte . . . 100
6.4.2 Zeitraffer . . . 105
6.5 Datenfilterung . . . 105
6.5.1 Filterung redundanter Bilder . . . 106
6.5.2 Anonymisierung . . . 106
6.6 Auswertung der Datenqualität . . . 107
6.6.1 Genauigkeit der Lokalisierung . . . 107
6.6.2 Luftdruck . . . 113

7 Zusammenfassung **115**

A Anhang **117**
A.1 Klassendiagramme . . . 117
A.2 Erstellung des Secure Sockets Layer (SSL)-Zertifikates . . . 126

Abbildungsverzeichnis **135**

Quellen- und Literaturverzeichnis **139**

KAPITEL 1

Einleitung

Der Mensch erfasst die Umwelt durch seine Sinnes. Dabei interpretiert und bewertet er ständig neue Informationen. Abhängig von persönlichem Interesse und Wichtigkeit werden diese entweder im Gedächtnis gespeichert oder verworfen. Beim Vergessen werden Assoziationen eliminiert, die das Gehirn nicht mehr für notwendig erachtet. Bedeutsame Ereignisse, deren immerwährende Erinnerung erwünscht ist, werden meist visuell auf einem Trägermedium festgehalten. Diese langlebigen Erinnerungen konnten sich früher nur wohlhabende Personen in Form von Denkmälern, Gemälden oder Büchern leisten. Im heutigen Informationszeitalter ist die digitale Archivierung von Daten wie Sound, Videosequenzen oder Bildern für die breite Masse möglich. Der Einsatz solcher Systeme erfordert allerdings die Verfügbarkeit (das Mittragen) spezieller Hardware, die zusätzliche im Bedarfsfall vom Benutzer aktiviert werden müssen. Somit werden viele Ereignisse nicht erfasst und geraten in Vergessenheit.

Lifelogging bezeichnet die automatische Dokumentation von benutzerrelevanten Daten über längere Zeiträume. Lifeloggingsysteme assistieren Nutzern, indem sie Informationen aufzeichnen und später wiedergeben können. In den letzten Jahren wurden verschiedenen Lifeloggingsysteme entwickelt, diese sind jedoch meist nur für spezielle Einsatzgebiete konzipiert und bieten eine geringe Konfigurationsmöglichkeit. Smartphones sind weit verbreitet, bieten zahlreiche Sensoren und werden meist in Körpernähe getragen. Zudem entfallen dann die Anschaffungskosten eines zusätzlichen Gerätes. In dieser Arbeit wird untersucht, wie Lifelogging mithilfe von Smartphones auf der Android-Plattform realisiert werden kann. Es wird ermittelt welche Daten mithilfe welcher Sensoren aufgenommen werden können.

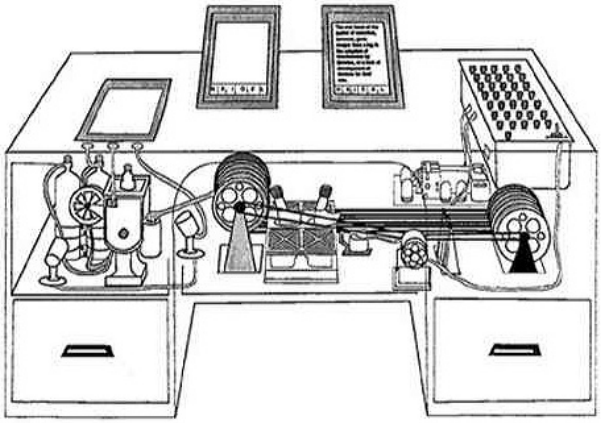
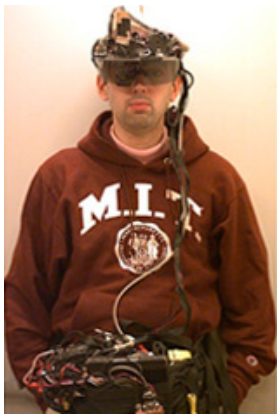


Abbildung 1.1: Vannevar Bush Idee von MEMEX (Quelle: [2]).

Es existiert keine einheitliche Definition für Lifelogging. Im weitesten Sinne versteht man darunter die personenbezogene Datenerfassung in einer Zeitspanne. Für manche Menschen bedeutet dies, täglich Informationen wie Aufenthaltsort, Wohlbefinden oder Schlafdauer schriftlich festzuhalten. Die Erfassung von persönlichen Daten ist nicht neu und wurde schon vor Jahrtausenden in Form von Tagebüchern und religiösen Schriftsammlungen praktiziert. Der Gedanke, diese mithilfe eines mechanischen Apparats durchzuführen war 1945, in einer Zeit, als es noch keinen PC gab, visionär [1]. Vannevar Bush präsentierte damals seine Idee (siehe Abbildung 1.1) einer fiktiven Maschine namens Memory Extender (MEMEX), die jedoch niemals realisiert wurde. Sie sollte dem Einzelnen als geistige Unterstützung dienen, die Form eines Schreibtisches besitzen und üblicherweise an der Arbeitsstätte aufgestellt werden. Dabei sollten Bücher, Kommunikationsprotokolle und sämtliche Unterlagen einer Person erfasst und auf einem Mikrofilm abgelichtet werden. Durch eine schnelle und flexible Abfrage sollten die abgespeicherten Daten auf eine Anzeigetafel projiziert werden. Eine zweite Anzeigetafel würde es ermöglichen, verschiedene Informationen nebeneinander zu vergleichen und Informationen zu verknüpfen. Zum Einbringen von geschäftlichem Schriftverkehr, Notizen und Bildern sollte diese Maschine eine Vorrichtung besitzen, die das augenblickliche Ablichten und Entwickeln erlaubt. Dies sollte über eine transparente Glasscheibe erfolgen. Als Speichermedium würde ein weiterentwickelter Mikrofilm dienen, der eine Kapazität von mehreren hundert Jahren haben sollte. Umfassende Dokumente sollten zudem direkt per Mikrofilm hinzugefügt werden können. Der Aufruf eines bestimmten Buches sollte durch eine eindeutige Codeeingabe erfolgen. Neu waren auch das Verlinken von Texten, die Erstellung einer logischen Abfolge, und das Anbringen von Notizen mithilfe eines Stiftes. Dabei sollten nach menschlichem Vorbild Verknüpfungen von Gedanken, beziehungsweise von Textpassagen mithilfe von Assoziation ermöglicht werden.
Die Grundidee des Lifeloggingparadigmas ist, menschliche Eindrücke festzuhalten. Wobei die Aufzeichnung nicht wie früher mithilfe von Stift und Papier, sondern auf digitalen Speichermedien erfolgt. Ein wichtiger Faktor ist das Design und die Benutzbarkeit des Systems. Bushs

(a) Tragbares System bestehend aus Fernseher, Lampe und Antennen. (Quelle: [3])



(b) System mit Brille und Laptop. (Quelle: [3])

(c) Unauffälligeres Brillensystem. (Quelle: [4])

Abbildung 1.2: Entwicklung Manns tragbarer Computersysteme.

(a) Eyetap-Prototyp um das Jahr 2010. (Quelle: [5])

(b) Konzept der Google Glasses Brille. (Quelle: [6])

Abbildung 1.3: Entwicklung tragbarer Brillensysteme.

MEMEX Konzept wäre nur stationär zu betreiben. Durch die Entwicklung tragbarer Computersysteme können mobile Lifeloggingsysteme entwickelt werden. Wichtig ist, dass sie unauffällig sind und über eine ausreichende Betriebszeit verfügen [7]. Als einer der ersten Menschen experimentierte Steve Mann, ein kanadischer Forscher, mit tragbaren Lifeloggingsystemen. Anfangs waren diese alles andere als unauffällig (siehe Abbildung 1.2a) und erinnerten mit ihren großen Antennen und wuchtigen Ausmaßen an Requisiten eines Science-Fiction-Films. Die sperrigen Computersysteme (siehe Abbildung 1.2b) wurden damals in Form eines Laptops separat getragen. In den späten neunziger Jahren konstruierte Mann die ersten unauffälligeren Prototypen, welche die Form einer Brille (siehe Abbildung 1.2c) besaßen. Sein neuestes Modell besteht nur noch aus einer unscheinbaren Brille. Die zunehmende Miniaturisierung ermöglicht immer kleinere Geräte, wie neue Brillensysteme. 2012 kündigte Google ebenfalls eine Datenbrille mit Lifeloggingfunktionalität an [6].

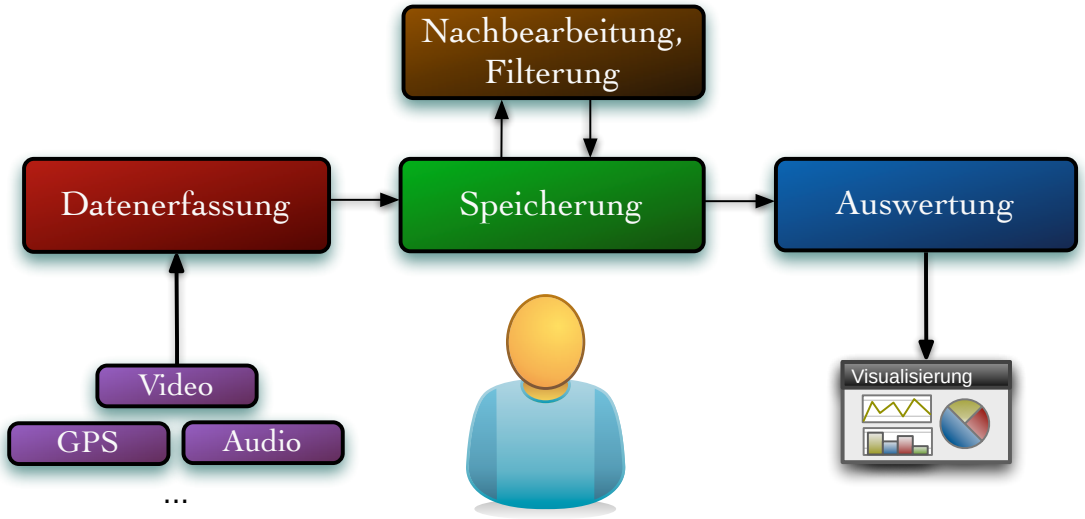


Abbildung 1.4: Grundliegender Ablauf von Lifeloggingsystemen.

1.1 Prinzip

Der grundlegende Ablauf von Lifeloggingsystemen ist in Abbildung 1.4 dargestellt. Die wesentlichen Komponenten sind Datenerfassung, Speicherung, eine optionale Nachbearbeitung beziehungsweise Filterung und Datenauswertung.

Es gibt verschiedenen Sensoren, die systematisch erfasst werden. Beispielsweise erfolgt eine regelmäßige *Datenerfassung* und *Speicherung* der Position mithilfe des Global Positioning System (GPS). Optional erfolgt eine *Nachbearbeitung* der Daten, wie die Berechnung des Mittelwertes. Manchmal ist aber nicht kalkulierbar, welche Informationen später wichtig sind. Ein großer Vorteil bei Lifelogging ist die Möglichkeit der ungefilterten Aufzeichnung von Daten, vorausgesetzt genügend Speicher steht zur Verfügung. Somit müssen Daten nicht vorzeitig verworfen werden und Informationen, die erst später an Relevanz gewinnen, gehen nicht verloren. Jedoch kann je nach System auch eine *Filterung* von unwichtig erscheinenden Daten erfolgen. Beispielsweise, weil nur wenig Speicher vorhanden ist. Schlussendlich erfolgt die *Auswertung* der Daten, beispielsweise indem die aufgezeichneten GPS-Positionen auf einer Karte visualisiert werden. Informationen wie der zurückgelegte Weg, die durchschnittliche Geschwindigkeit oder bereits besuchte Orte können ebenfalls extrahiert und visualisiert werden.

Soziale Netzwerke wie Facebook, Gooogle+ und Twitter bieten auch Lifeloggingfunktionalität; beispielsweise durch eine chronologische Auflistung von Standorten. Viele Menschen teilen regelmäßig persönliche Informationen mithilfe sozialer Netzwerke. Wie in Abbildung 1.5 ersichtlich, verspricht Lifelogging gegenüber den Konzepten Blogging und Microblogging die Erfassung von vielen Informationen mit wenig Aufwand.

Dabei können bewusste, unbewusste oder auch nicht wahrgenommene Informationen aufgezeichnet werden. Aber nicht nur vom Menschen erfassbare Informationen wie Bilder, sondern auch kontextuelle Daten wie Position oder Luftdruck können erfasst werden. Anschließend ste-

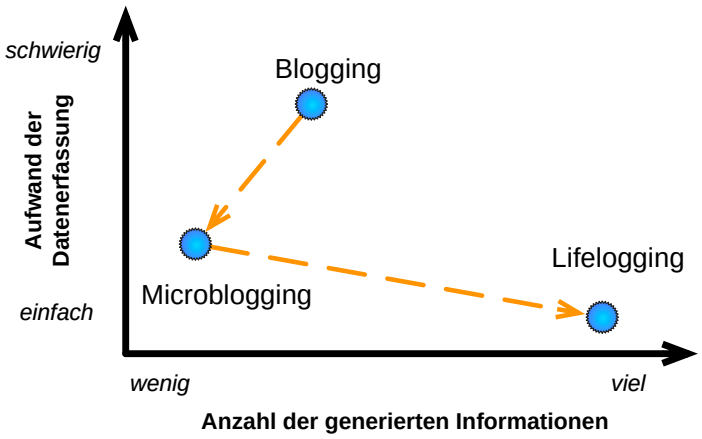


Abbildung 1.5: Zusammenhang zwischen Aufwand und Anzahl, um Daten zu erfassen. (In Anlehnung an: [8])

hen diese Daten dem Individuum selbst zur Verfügung oder werden mit anderen geteilt. Die Anwendungsgebiete sind dabei sehr vielfältig und reichen von einfacher Lokalisierung bis zur systematischen Überwachung und Auswertung.

1.2 Anwendungsgebiete

Die Aufgabe von computergestützten Lifeloggingsystemen besteht in der digitalen Erfassung, Speicherung und Visualisierung sämtlicher Informationen und Erlebnisse, die ein Individuum erfährt. Für Lifelogging können stationäre oder portable Computersysteme eingesetzt werden. Ein Verbund mehrerer Systeme kann ebenfalls erfolgen. Bei stationären Lifeloggingsystemen können Sensoren immer nur einen begrenzten räumlichen Bereich erfassen. Deswegen eignen sich besonders tragbare Computersysteme für die persönliche Datenerfassung. Ein Vorteil von stationären Systemen ist hingegen die einfachere Erkennung wichtiger Ereignisse, da die Umgebung sich nicht stark verändert. Bei mobilen Systemen ist die Bestimmung relevanter Ereignisse meist schwieriger und abhängig vom Einsatzgebiet. Der Sinn und Zweck, wie auch die Anwendungsgebiete sind sehr vielfältig und teilweise noch nicht absehbar. Durch eine digitale Archivierung aller erlebten Ereignisse kann das menschliche Gehirn unterstützt, oder der Mensch überwacht werden. Beispielsweise ist eine Messung des Gewichtes für viele selbstverständlich. Sicherheitstechnische Aspekte wie automatische Beweisfotos sind ebenfalls denkbar. Die Kombination von Augmented Reality (AR)-Anwendungen oder die Übertragung des Sichtfeldes an eine andere Person sind mögliche Erweiterungen. Die folgende Einteilung gibt einen groben Überblick über die möglichen Einsatzgebiete.

Gedächtniserweiterung Ein Anwendungsgebiet ist die Erweiterung des menschlichen Gedächtnisses [9]. Durch die kontinuierliche Erfassung von Daten wie Bildern und Ton werden Erlebnisse eines Menschen aufgezeichnet. Mehrere Menschen besitzen meist unterschiedliche

Erinnerungen an dasselbe Ereignis, die unter Umständen stark divergieren können. Zur Klärung können beispielsweise Lifeloggingsysteme herangezogen werden, da diese objektive Daten aufzeichnen. Das spätere Betrachten der Daten kann vergessene Details hervorbringen [10]. Die Erstellung von Erinnerungen an verstorbene Menschen ist ebenfalls ein Anwendungsgebiet [11].

Selbstkontrolle Lifelogging ermöglicht es, Aufschlüsse über das eigene Leben zu erlangen. Beispielsweise versprechen Weckeranwendungen durch Analyse des Schlafrhythmus, frisches und ausgeruhtes Aufwachen [12]. Durch Überwachung von sportlichen Aktivitäten wird die persönliche Fitness ermittelt. Ebenso können die Daten über Sportportale geteilt und verglichen werden [13]. Dabei wird der Austausch von Laufrouten oder bevorzugten Wanderstrecken unterstützt. Der Tagesablauf kann durch Protokollierung personenbezogener Daten optimiert werden, beispielsweise indem Muster im Bewegungsprofil erkannt und optimierte Routen vorgeschlagen werden [14]. Gezielte Entspannungsübungen mit kontinuierlicher Überprüfung der Vitalfunktionen versprechen eine tiefere Entspannung [15]. Durch eine Analyse des gewohnten Aufenthalts kann eine optimierte Heizungsteuerung erfolgen [16]. Die allgemeine Dokumentation des täglichen Lebens könnte auch in Form eines Alibis nützlich sein. Für ältere Personen kann die frühe Erkennung eines Sturzes, beispielsweise durch Erkennung korrespondierender Events, lebenswichtig sein. Jiangpeng Dai und andere entwickelten einen ersten Prototyp eines Smartphonesystems zum Erkennen eines Sturzes [17]. Vinay Vishwakarma und andere erforschten ein Ansatz zur Sturzerkennung von Personen anhand von fixen Kameras [18]. Das Netzwerk *Quantified Self*, eine Gruppe bestehend aus Entwicklern, Konsumenten und Verkäufern, beschäftigt sich mit Produkten und Anwendungen zur Selbstüberwachung und Optimierung [19].

Medizin Durch das Überwachen von biometrischen Daten können Erkrankungen besser therapiert und frühzeitig erkannt werden. Yinpeng Chen und andere forschten an einem multimodalen Biofeedbacksystem, das Schlaganfallpatienten bei der Rehabilitation unterstützen soll [20]. Dabei wird mithilfe von Motioncapturetechnik die Bewegung des Patienten erfasst. Schlussendlich werden Daten abgespeichert, ausgewertet und als Unterstützung visuell wie auch auditiv Wiedergegeben. Pang Wu und andere entwickelten ein mobiles Überwachungssystem für ältere Menschen [21]. Durch das Monitoring der verschiedenen Aktivitäten soll auf den aktuellen Gesundheitszustand geschlossen werden. Durch Protokollierung der Nahrungsaufnahme können gesundheitsschädigende Auswirkungen frühzeitig ermittelt werden. Keigo Kitamura und andere realisierten bereits ein System, das die relativen Mengenverhältnisse von Lebensmittelgruppen anhand visueller Merkmale ermittelt [22]. Visuelle Aufzeichnungen können auch das Erinnerungsvermögen unterstützen. Eine Studie von Microsoft Research (siehe Abbildung 1.6) ergab, dass durch Rekapitulieren erlebter Situationen das Erinnerungsvermögen von Personen mit Gedächtnisstörung gestärkt wird [23]. Basel Kikhia und andere entwickelten das System Memory Lane, das ebenfalls Personen mit leichter Demenz im Alltag unterstützen kann [24]. Das System besteht aus einem Verbund mehrerer einzelner Systeme wie Türsensor, GPS-Tracker, Audio Recorder und Kamera.

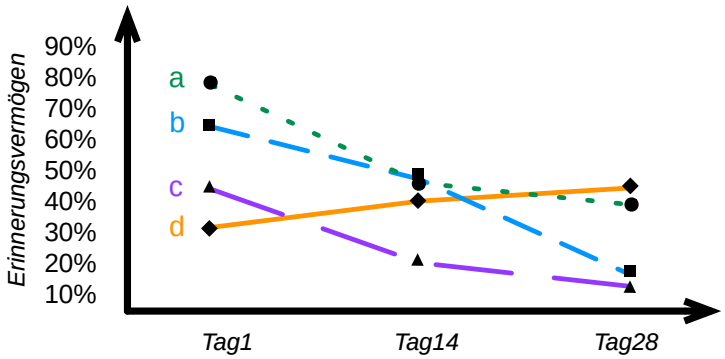


Abbildung 1.6: Ergebnisse einer Studie zum Erinnerungsvermögen. (Quelle: [23])
a: Kontrollpersonen ohne Gedächtnisstörung, b: Pflegepersonal, c: Personen mit Gedächtnisstörung, die vom Pflegepersonal Bilder gezeigt bekamen, d: Personen mit Gedächtnisstörung unter Verwendung einer speziellen Software, die auf einem Tablet-Computer zur Verfügung stand.

Sicherheit Gazmend Bajrami und andere realisierten ein neues Sicherheitskonzept für mobile Systeme [25]. Basierend auf einer Ganganalyse des Trägers wird der Zugriff auf das System legitimiert. Dabei werden die Beschleunigungsdaten mit einem Modell verglichen, und bei Übereinstimmung der Zugriff gewährt. Ein weiteres Szenario ist die Erfassung von Straftaten. Durch die permanente Aufzeichnung können Beweisfotos eines Verbrechens erstellt werden. Steve Mann definierte anhand seiner Forschungsprojekte die sogenannte Unterwachung, dies bedeutet die Umkehrung des normalen Überwachungsweges [7]. Normalerweise blicken Kameras von oben (Eye in the sky) auf das zu überwachende Objekt, hingegen versteht man bei der Unterwachung die personenbezogene Überwachung anhand von tragbaren Kameras. Der Vorteil bei einer solchen Aufzeichnung, die der persönlichen Kontrolle unterliegt, besteht darin dass Daten nicht von fremden Personen oder Institutionen kontrolliert werden können. Spionage oder Erstellung von Benutzerprofilen anhand zeitbezogener Aufenthaltsorte wird damit erschwert. Die Aufnahme erfolgt ebenfalls jeweils aus der Sicht des Benutzers und kann somit das relevante Sichtfeld besser abdecken als stationäre Kameras von oben. Kunstprofessor Hasan Elahi der Rutgers Universität New Jersey wurde 2002 unschuldig vom FBI wegen krimineller und terroristischer Aktivität verhaftet [26]. Zur Vermeidung zukünftiger Festnahmen beschloss Elahi sämtliche Aktivitäten im Vorhinein als pre-alibi aufzunehmen. Auf seiner Webseite `http://elahi.umd.edu/track` veröffentlicht Elahi seitdem ständig seinen aktuellen Standort. Zur Nachvollziehbarkeit und Überprüfung von medizinischen Experimenten forschten Daragh Byrne und andere bereits mit der automatischen visuellen Aufzeichnung [27]. Dabei stießen sie auf Einschränkungen verfügbarer Systeme wie die fehlende zeitliche und räumliche Auflösung.

Wissensvermittlung Der Mensch besitzt die Fähigkeit, durch Beobachtung seiner Mitmenschen zu lernen. Mithilfe von Lifeloggingsystemen können seltene Fertigkeiten und Erfahrungen an eine breite Menschenmenge weitergegeben werden. Steve Mann projiziert wäh-

rend einiger seiner Vorträge zusätzlich einen Live-Videostream aus der Sicht seiner eigenen Perspektive auf eine Leinwand [7]. Dabei können die Teilnehmer zusätzliche Informationen aufnehmen und sich vermutlich besser in die Situation des Vortragenden hineinversetzen. Zusätzlich visualisiert Mann Informationen über Post-its und Skizzen seines Notizblockes. Erfahrungen, die selten, gefährlich oder kostspielig sind, können einer breiten Masse zugänglich gemacht werden. Beispielsweise wurde bei der Google I/O 2012 eine Live-Übertragung eines Fallschirmsprunges in das Internet übertragen [28]. Zudem kann durch die Ansammlung von Informationen vieler Menschen ein kollektives Gedächtnis entstehen [29]. Beispielsweise können durch soziale Netzwerke wie Facebook, Twitter und Google+ verschiedene Interpretationen oder Meinungen einer Sachlage erörtert werden.

1.3 Aufgabenstellung und Ziele

Das Ziel dieser Arbeit ist die Entwicklung einer dynamisch anpassbaren Lifeloggingarchitektur für moderne Smartphones, basierend auf dem Android-Betriebssystem. Sie soll für zukünftige Forschungsprojekte und der Allgemeinheit zur Datenerfassung und Auswertung dienen. Dazu soll die Architektur eine möglichst große Kompatibilität zu verschiedenen Smartphones bieten. Eine klare Struktur der Datenerfassung, Auswertung, Datenspeicherung und Visualisierung soll eine einfache Erweiterung der Architektur ermöglichen. Zur dynamischen Konfiguration soll eine auf Extensible Markup Language (XML)-basierende Beschreibungssprache (fortlaufend als LLDL bezeichnet) dienen. Diese Sprache ermöglicht eine flexibel steuerbare Datenerfassung. Sämtliche Daten sollen in einer Datenbank abgespeichert, exportiert und importiert werden. Über SSL soll die Übertragung verschlüsselt werden, und mithilfe von Basic Access Authentication eine zusätzliche Passwortabfrage erfolgen. Damit sollen die Übertragung und der Zugriff abgesichert werden. Eine Visualisierung soll dabei eine orts- und zeitabhängige Filterung unterstützen. Ebenso soll ein Datenbankmanager benutzerdefinierte Structured Query Language (SQL)-Abfragen ermöglichen.

1.4 Herausforderungen

Android bietet ein umfangreiches API zur Erstellung von Anwendungen, die auf unterschiedlichsten Geräten ausführbar sind. Es gilt zu klären, ob und wie eine Lifeloggінganwendung auf einem Android-Smartphone realisiert werden kann. Weiters ist zu evaluieren, welche Daten ermittelt, und welche Informationen daraus extrahiert werden können. Eine wichtige Anforderung ist, dass die Lifeloggінganwendung stabil und permanent im Hintergrund abläuft. Da es sich um sensible, personenbezogene Daten handelt, ist ein Sicherheitskonzept zum Schutz vor unbefugtem Zugriff zu erarbeiten. Die Datenerfassung soll anhand der verfügbaren Sensoren möglichst oft erfolgen. Dies verursacht einen hohen Energieverbrauch, der die Betriebszeit des Smartphones einschränkt. Eine entsprechende Strategie zur Lösung dieses Problems ist zu entwickeln. Zudem besteht die Schwierigkeit, dass die erfassten Daten entsprechend verarbeitet und persistent abgespeichert werden müssen. Ebenfalls ist zu erarbeiten, wie eine geeignete Visualisierung erfolgen kann, sodass Informationen aus den erfassten Daten gewonnen werden können.

1.5 Organisation der Arbeit

Das erste Kapitel dieser Arbeit beschäftigt sich mit den Ursprüngen, möglichen Anwendungsgebieten und den Herausforderungen von Lifeloggingsystemen. Im zweiten Kapitel wird auf die grundlegenden Abläufe und auf mögliche Einsatzgebiete der Systeme eingegangen. Anschließend werden verschiedene Lokalisierungsverfahren vorgestellt. Schlussendlich wird auf die Indizierungs- und Visualisierungsmöglichkeiten eingegangen. Das dritte Kapitel zeigt Einblicke in den aktuellen Stand der Forschung, in die dazu notwendigen wissenschaftlichen Methoden und entsprechenden Verfahrensweisen. Strukturiert nach Sensormodalität werden einzelne Systeme näher vorgestellt. Das vierte Kapitel stellt am Markt verfügbare Systeme vor und zeigt deren Konfigurationsmöglichkeiten. Im fünften Kapitel wird die Modellierung und Realisierung des praktischen Teils der Arbeit beschrieben. Der Schwerpunkt liegt dabei auf der LLDL Konfiguration, den Möglichkeiten und Einschränkungen. Das sechste Kapitel zeigt die zentralen Ergebnisse der Arbeit. Zuerst werden sämtliche Funktionalitäten wie Konfiguration und Visualisierung von ALL vorgestellt. Danach erfolgt eine Analyse der Datenerfassung bezüglich ihrer Genauigkeit und möglichen neuen Anwendungsgebieten. Diagramme beschreiben dabei den Zusammenhang der tatsächlichen und der gemessenen Daten. Im siebten Kapitel erfolgt ein finaler Überblick als Zusammenfassung. Dabei wird auf die gesammelte Nutzererfahrung sowie Chancen und Risiken der implementierten Anwendung eingegangen. Der Anhang liefert einen näheren Einblick der realisierten Klassen und der Erstellung des SSL-Zertifikates.

KAPITEL 2

Hintergrund

Der Bereich von Lifelogging ist sehr breit, und deckt sowohl Hardware als auch Software ab. Dieses Kapitel führt Kriterien ein mit deren Hilfe eine Einteilung von Lifeloggingsysteme getroffen werden kann. Darauf folgend werden die ersten tragbaren Computersysteme, ihre Einsatzgebiete und möglichen Lifelogging-Funktionalitäten vorgestellt.

2.1 Kriterien für Lifeloggingsysteme

Lifelogging ermöglicht eine persönlichen Datensammlung. Diese Datensammlung erfolgt auf unterschiedlichste Art und Weise. Grundsätzlich kann man die Datenerfassung, Speicherung, Nachbearbeitung, Filterung und Auswertung von Lifelogging anhand unterschiedlicher Aspekte charakterisieren. Für die Entwicklung eines Lifeloggingsystems sollten folgende Aspekte bedacht werden:

Verwendung Welches Ziel wird mit dem Sammeln der Daten verfolgt? Sollen Daten nur privat oder auch öffentlich zugänglich sein? Wie und wann sollen Informationen aus den Daten extrahiert werden?

Bandbreite Wie viele Sensoren sollen benutzt werden? Welche Sensoren sollen welche Daten erfassen?

Perspektive An welchem Ort erfolgt die Erfassung? Sollen Daten über ein fix montiertes oder portables System erfasst werden? Worauf sollen die Sensoren ausgerichtet werden?

Abtastung Wie oft sollen Daten erfasst werden? Sollen redundante Daten gespeichert werden?

Steuerung Wie soll die Aufnahme erfolgen? Bewusst oder manuell durch Betätigung einer Taste. Unbewusst und automatisch, periodisch oder diskriminierend durch Erkennung relevanter Ereignisse.

Organisation Gibt es ein einzelnes, zentrales System oder mehrere zusammenhängende Systeme? Erfolgt die Speicherung zentralen oder auf einem dezentral Server?

2.2 Entstehung von Lifelogging

Einer der ersten tragbaren Computer wurde 1961 von Ed Thorp und Claude Shannon am MIT konstruiert, um die Gewinnchance bei Roulette Glücksspielen zu erhöhen [31]. Das System besteht aus zwölf Transistoren und einer eigenen Stromversorgung. Es wurde in ein Paar Schuhe eingebettet. Die Steuerung erfolgt über einen Schalter, der mit der großen Zehe betätigt wird. Beim Einwurf der Kugel ist der Schalter zu betätigen. Anschließend berechnet das System die wahrscheinlichste Endpostion der Roulettekugel. Die Information über die Endposition wird mithilfe eines kleinen Lautsprechers, der im Ohr versteckt wird, an die spielende Person akustisch übermittelt. Die Studentengruppe *Eudaemonic Pie* entwickelte das System Ende der sieb-

(a) *Eudaemon's shoe* Roulette-Computer. (b) Im Schuh integriert.

Abbildung 2.1: *Eudaemons' shoe* mobiler Roulette-Computer. (Quelle: [30])

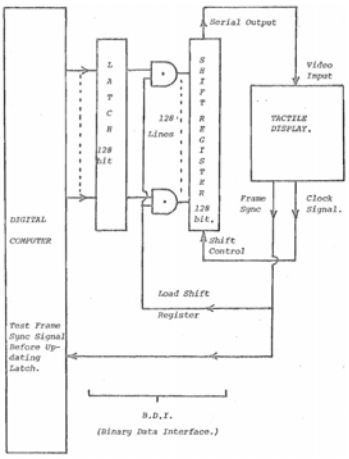
ziger Jahre weiter, wobei dieses mithilfe neuer Technik verbessert wurde. Dabei erfolgt die Ausgabe nicht mehr über einen Lautsprecher, sondern über Vibrationen [7]. In Abbildung 2.1 ist die finale Version des *Eudaemons' shoe* Computers zu sehen.

Eine frühe Studie, die sich mit tragbaren Computersystemen beschäftigt, wurde 1978 von J. A. Brabyn durchgeführt und befasste sich mit Hilfsmitteln für blinde Menschen [32]. Dabei wurde das Potenzial verschiedener mechanischer, optischer und akustischer Sensoren für die Erfassung der Umwelt untersucht. Beispielsweise wird eine elektromechanische Vorrichtung verwendet um Kopfbewegungen zu verfolgen (siehe Abbildung 2.2a) oder ein optomechanischer Abstandsmelder, der akustische Warnmeldungen aussendet. Brabyn entwickelte einen Prototyp eins mobilen Computersystems, das verschiedene Daten auswertet und über ein taktiles Display (siehe Abbildung 2.2b) übermittelt. Diese Ausgabeform wurde ebenfalls für blinde Menschen konzipiert und besteht aus einer Tafel mit vibrierenden Feldern.

2.2.1 Steve Mann

Steve Mann ist ein Wissenschaftler, der einen entscheidenden Beitrag zur Erforschung von Lifeloggingsystemen erbrachte. Mann forscht seit Ende der siebziger Jahre an tragbaren Computersystemen. Anfangs beschäftigte sich Mann mit der experimentellen visuellen Erfassung [4]. Dabei entwickelte Mann ein tragbares System, das am Kopf getragen wird und aus einem tragbaren Fernseher, einer Helmlampe und zwei Antennen (siehe Abbildung 1.2a) besteht. Die zwei Antennen ermöglichen über unterschiedliche Frequenzen das simultane Senden und Empfangen von Computerdaten, Audio- oder Videosignalen. Die Recheneinheit des Systems besteht aus Computerkomponenten, die in einem Stahlrahmen verbaut und am Rücken getragen werden. Anschließend experimentierte Mann mit der sogenannten *lightpainting*-Methode, einer visuellen Aufnahme, in der Licht als künstlerisches Werkzeug verwendet wird. Dabei benutzt Mann ein zusätzliches Beleuchtungssystem, das aus einer Reihe von Glühlampen und Schaltern besteht. In Abbildung 2.3a ist das komplette System ersichtlich. Zur Bedienung des Beleuchtungssystems befinden sich sechs Mikroschalter am Rahmen, (siehe Abbildung 2.3b) deren Signale an

(a) Sensor zur Bestimmung der Kopfrotation. (Quelle: [32] S. 40)



(b) Interface eines taktilen Displays. (Quelle: [32] S. 182)

Abbildung 2.2: Studie zur Erforschung tragbarer Computersysteme und neuer Sensoren.

den tragbaren Computer übermittelt werden. Im *lightpainting*-Programm wird berechnet, welche Lampen des Beleuchtungssystems aktiviert werden sollen, und die entsprechenden Steuerbefehle gesendet. Die Energieversorgung des Systems erfolgt entweder über einen Akku, welcher aber nur wenige Minuten das System versorgen kann oder über einen umhängbaren Transformator. Zu Beginn der achtziger Jahre überarbeitete Mann das System und konnte mithilfe neuer Mikrochips und eigens entwickelten Teilen das Gewicht und den Energieverbrauch reduzieren (siehe Abbildung 2.4a). Abbildung 1.2b zeigt ein Setup, bei dem die Kamera mit diversen anderen Sensoren auf einer Brille montiert sind.

Im folgenden Zeitraum wurde Mann bewusst, dass die kompakte, unauffällige Art und die lange Betriebszeit seiner Systeme (siehe Abbildung 2.4c) neue Möglichkeiten der Dokumentation bieten. Mann startete 1994 ein Experiment namens Wearable Wireless Webcam (WWW). Dies bestand aus einem tragbaren Computersystem, das Bilder in Echtzeit an das Internet übertrug. Dabei ließ Mann die Öffentlichkeit an seinem Leben teilnehmen. Über einen Zeitraum von zwei Jahren sendete Mann ununterbrochen Bilder an seine Webseite, und über $30,000$ Zuschauer besuchten diese täglich. Ein Screenshot der Webseite aus dem Jahre 1995 ist in Abbildung 2.5a zu sehen. Dabei prägte Mann den Begriff *lifecasting*. Darunter versteht man einen kontinuierlichen Broadcast von digitalen Medien, wie Bildern, der das menschliche Leben dokumentiert. In Abbildung 2.5b ist sein tragbares Erfassungssystem (WWW) mit Funkanbindung zu sehen. Als HMD verwendete Mann ein VR4 Stereo System, eine am Kopf getragen Videobrille der Firma Visual Research Systems mit einer Auflösung von je 240 x 120 Pixel pro Auge [33]. Zusätzlich waren noch zwei Kameras und das entsprechende Computersystem mit Funkübertragung integriert. In Abbildung 2.5c ist die Komplettausstattung eines ähnlichen, von Mann entworfenen Systems zu sehen.

In den Jahren 1996 bis 2003 stellte Jennifer Ringley ebenfalls ihren lifecast öffentlich im Internet bereit [34]. Die Bilder auf ihrer Webseite *JenniCAM* zeigten Aufnahmen aus ihrer Woh-

(a) Lightpainting Ausrüstung. (Quelle: [4])

(b) Steuereinheit des Beleuchtungssystems. (Quelle: [4])

Abbildung 2.3: Experimentelles Beleuchtungssystem zur künstlerischen Gestaltung.

(a)

Abbildung 2.4: Manns energiesparendes erneuertes System mithilfe selbst entwickelter Technologie. (Quelle: [3])

(a) Screenshot der lifecasting Webseite, mit Bericht eines Brandes des östlichen Universitätskomplexes (Quelle: [7] S. 130)

(b) Steuereinheit des Beleuchtungssystems. (Quelle: [35])

(c) Tragbarer Fernseher mit Lampe und Antennen. (Quelle: [36])

Abbildung 2.5: Life Übertragungssystem und Internetplattform aus dem Jahr 1994.

nung und wurden alle zwanzig Minuten aktualisiert. Für ein Abonnement um $15 erhielt man eine schnelleres Update im Abstand von zwei Minuten.

Mann forschte in den folgenden Jahren an einer verbesserten Technik, um das tatsächliche Sichtfeld eines Menschen erfassen zu können. Die Erfahrung mit Wearable Wireless Webcam (WWW) zeigte Mann die Schwächen verfügbarer Systeme, wie beispielsweise das eingeschränkte Sichtfeld oder die Verzerrungen der optischen Linsen. Dabei vertiefte Mann sich gezielt in die Forschung verbesserter Videoerfassung und neuartiger veränderbare Realität (Mediated Reality)-Anwendungen. Bisherige Systeme erfassten visuelle Informationen meist in dieselbe Richtung, in die der Oberkörper oder Kopf des Trägers zeigte. 1999 präsentierte Mann seine neue Erfindung namens Eyetap, einer Technologie, die Display und Kamera vereint [37]. Eyetap zeichnet sich durch die Fähigkeit aus, das menschliche Sichtfeld exakt zu erfassen, und es ebenso exakt überlagern zu können [38] [39]. In Abbildung 2.6 ist die schematische Darstellung der Eyetap-Technologie visualisiert. Mithilfe eines Spiegelsystems werden die Lichtstrahlen der realen Welt, die normalerweise in das menschliche Auge eintreffen, abgelenkt und zu einer Kamera weitergeleitet [40]. Die Videoaufnahme der Kamera wird auf der gegenüberliegenden Seite von dem Ausgabesystem AREMAC[1] wieder ausgesendet, wobei in der Zwischenzeit das Bild eventuell von einem tragbaren Computersystem verändert wurde. Die ausgesendeten Lichtstrahlen des AREMAC werden dann über den Spiegel direkt in das Auge projiziert. Der Vorteil des Systems ist, dass nicht nur wie bei AR-Anwendungen Informationen hinzugefügt, sondern auch entfernt oder ersetzt werden können. Mit dem Austausch von visuellen Informationen prägte Mann den Begriff *Mediated Reality*. Dabei wird zum Beispiel unerwünschte Werbung, ähnlich wie bei einem Webbrowser-Werbeblocker, entfernt. Zwei Prototypen von Manns Eyetap-Technologie sind in Abbildung 2.7 zu sehen.

Durch die Verwendung der Eyetap-Technologie betrachtet sich Mann selbst als Cyborg und definiert damit den Begriff Cyborg Logging (GLOGGING). Eine lebenslange Cyborg-

[1] Das AREMAC (*camera* rückwärts buchstabiert) System ermöglicht eine dreidimensionale Ablenkung von Licht und verfügt optional über eine eigene Lichtquelle.

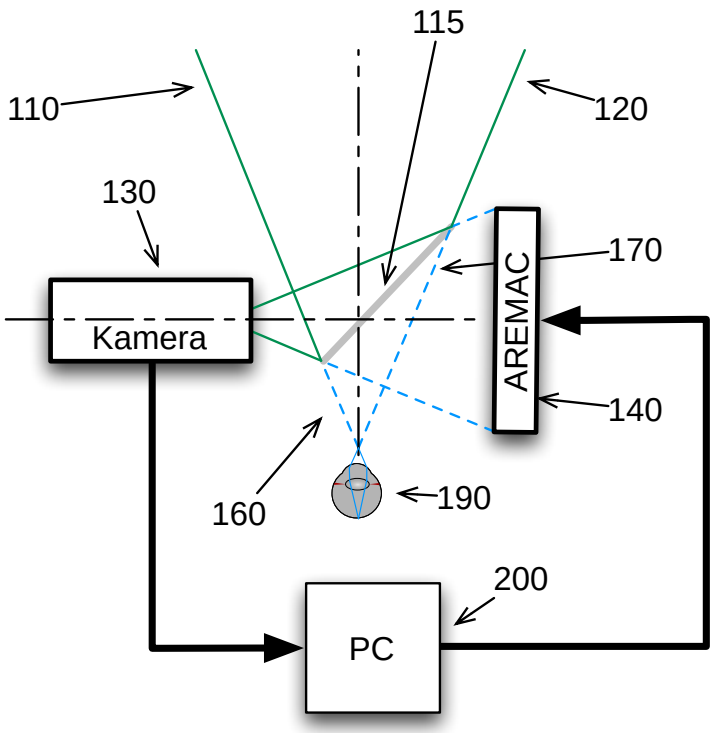
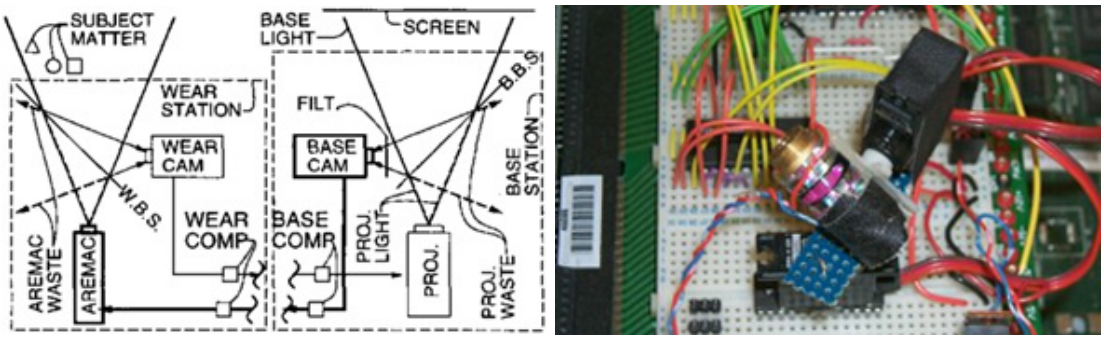


Abbildung 2.6: Schematische Darstellung der Eyetap Technologie. 110,120: Lichtstrahlen des Sichtfeldes; 115: zweiseitiger undurchsichtiger Spiegel; 130: Kamera; 140: AREMAC-System; 160: Eintreffen der synthetisch generierten Lichtstrahlen; 170: Synthetische Lichtstrahlen; 190: Auge; 200: PC der die Bilder verarbeitet; (In Anlehnung an: [40] S. 3)

(a) Eyetap-Prototyp aus dem Jahr 1993. (Quelle: [41])

(b) Eyetap-Prototyp aus dem Jahr 2012. (Quelle: [42])

Abbildung 2.7: Entwicklung der Eyetap-Prototypen.



(a) Telepointer-System für gemeinschaftliche visuelle Interaktion. (Quelle: [43])

(b) Prototyp eines Laser-AREMAC. (Quelle: [44])

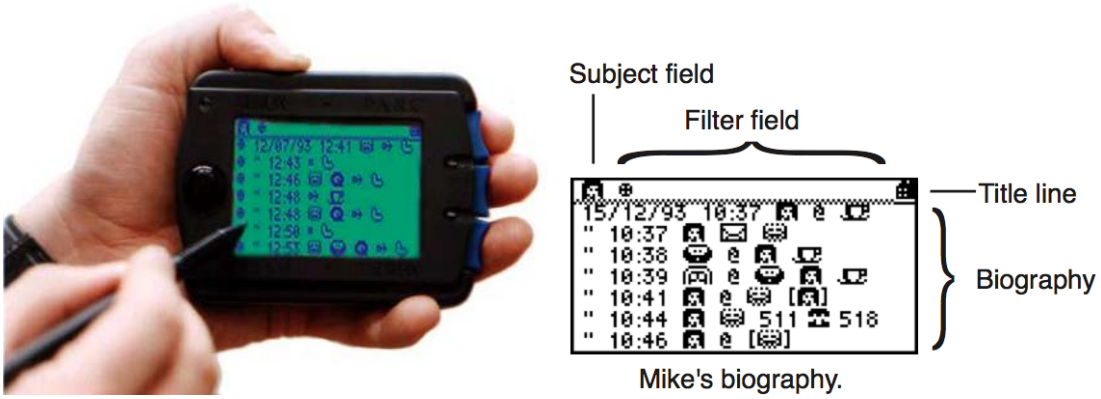
Abbildung 2.8: AREMAC-Anwendung und Prototyp.

Aufzeichnung bezeichnet Mann als Lifelong Cyborglog (GLOG). Mann entwickelte verschiedene Typen von AREMAC, welche 1999 patentiert wurden [40]. Der sogenannte Laser AREMAC projiziert einen Laser im dreidimensionalen Raum [43]. Damit realisierte Mann das sogenannte Telepointer-Konzept. Dabei wird ein Videosignal aus der Sicht einer Person an einen anderen Ort beziehungsweise in das Sichtfeld einer anderen Person übertragen. Zum Beispiel projiziert ein Projektor das Videosignal auf eine Leinwand. Mit einem Laserpointer kann eine zweite Person auf die Leinwand zeigen. Dann wird die Position des Laserstrahls ermittelt und an das Telepointer System (siehe Abbildung 2.8a) zurück übertragen. Mithilfe des Laser-AREMAC-Systems, einem Laserpointer oder Spiegel, der über zwei Servomotoren (siehe Abbildung 2.8b) abgelenkt wird, kann auf ein tatsächliches, physisches Objekt gezeigt werden. Somit kann eine Person aus der Ferne auf ein tatsächliches 3D-Objekt zeigen. Eine direkte Visualisierung in das menschliche Auge ist mithilfe der Eyetap-Technologie ebenfalls möglich.

2.2.2 Forget Me Not

Am Rank Xerox Research Center wurden 1993 über einen Zeitraum von mehreren Monaten die Risiken und Möglichkeiten zukünftiger mobiler Computerassistenten untersucht [45]. Dabei entstand ein Prototyp, genannt *Forget Me Not*, der als mobile Gedächtnisstütze fungieren sollte. Als Grundlage diente ein tragbarer Computer, namens ParcTab, welcher in Abbildung 2.9a dargestellt ist. Dieser wurde von Roy Want und seinen Kollegen entwickelt [46]. Bestehend aus einem monochromen LC-Display (128 x 64 Pixel) mit resistiver Touch-Eingabe, 3 Knöpfen, einem Piezo-Tongenerator, einem mit 12 MHz getakteten Mikrocontroller und einer bidirektionalen Infrarot Schnittstelle.

Beim Entwurf flossen bekannte Eigenschaften des menschlichen episodischen Gedächtnisses ein. Durch alternative Suchfunktionen sollte das Abrufen von Informationen ermöglicht werden, die einst bekannt waren, aber in der Zwischenzeit in Vergessenheit geraten sind. Dabei sollte die Suche kontextbezogen erfolgen. Beispielsweise ist von einem gesuchten Dokument



(a) ParcTab Prototyp. (Quelle: [45] S. 4)

(b) ParcTab UI mit Biografie der erfassten Ereignisse. (Quelle: [44] S. 6)

Abbildung 2.9: ParcTab *Forget Me Not* Prototyp.

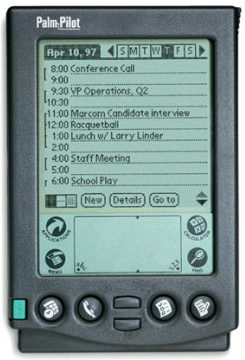
nicht der Inhalt, aber der Aufenthaltsort bekannt, an dem es empfangen wurde. Alltägliche Gedächtnisprobleme wie die Erinnerung an Namen von Personen, ein verlorenes Dokument oder die Bedienung einer Maschine sollten damit gelöst werden. *ParcTab* sollte ausgewählte Aspekte von Benutzer Aktivitäten wie Telefongespräche, Büroarbeit, Reisen, Fernsehen und Einkaufen aufzeichnen. Die Position sollte mithilfe von Funktechnik ermittelt werden, indem der nächstgelegene Sender ermittelt wird (wurde nicht realisiert). Dabei sollte das System leicht zu verstehen, zu erlernen und zu bedienen sein, sowie verlässlich und vertrauenswürdig arbeiten.

Wegen der geringen Displayauflösung entschied man sich für eindeutige Icons, die Ereignisse und Personen repräsentieren. In Abbildung 2.9b ist die grafische Benutzeroberfläche zu sehen, wobei in der Titelzeile die aktuelle Person und optionale Filtereinstellungen angezeigt werden. Im Biografie-Bereich werden zeilenweise einzelne Einträge mit Zeitangabe und eindeutigen Symbolen, die Personen oder Ereignisse repräsentieren, visualisiert. Beispielsweise zeigen die ersten zwei Zeilen an, dass Mike um 10:37 in die Küche gegangen ist, und in derselben Minute ein E-Mail von Grouch eingetroffen ist. Zur Ermittlung der Position wurde eine infrarotbasierte Lokalisierung getestet. Die Erkennung von Treffpunkten mit anderen Personen geschieht durch einen Datentransfer von einem zum anderen *ParcTab*. Zum Schutz der Privatsphäre können auch nur jene Daten betrachtet werden, die auch von der Person selbst aufgezeichnet werden. Sämtliche Daten sollten anfangs exportiert werden, wurden dann aber einfach auf dem Gerät selbst gespeichert.

Der *Forget Me Not*-Prototyp erfasst folgende Daten:

- Position, wie auch Treffpunkt mit andern.
- E-Mail-Verkehr.
- Datenaustausch zwischen zwei ParcTabs.
- Telefongespräche: über die Logdatei der PCX-Telefonanlage

(a) Apple Newton PDA. (Quelle: [53])



(b) Palm Pilot 1000. (Quelle: [54])

Abbildung 2.10: Erste tragbare PDA-Systeme.

Nach mehrmonatigem Test wurde das Projekt beendet. Die Ergebnisse zeigen, dass noch weitere Forschungsarbeit, besonders im Bereich der kontextbezogenen Suche, erforderlich war. Die technischen Realisierungsmöglichkeiten schränkten ebenfalls die Realisierung ein.

Ebenfalls im Jahr 1993 stellte Apple den ersten tragbaren Personal Digital Assistant (PDA), namens Apple Newton, vor. Im Jahr 1996 folgte die Markteinführung der erfolgreichen Palm Pilot-Serie (siehe Abbildung 2.10) von 3Com [47]. Nachfolger dieser PDA-Systeme bilden die Grundcomputereinheit von mehreren Lifeloggingsystemen ([48], [49], [50], [51], [52]).

2.2.3 MyLifeBits

Gordon Bell, ein Mitarbeiter von Microsoft Research erfuhr 1998 von Raj Reddy über das bevorstehende *one million book*-Projekt der Carnegie Mellon University [55]. In diesem Projekt sollte ein digitales Bücherarchiv erstellt werden, das über eine Million Bücher umfasst. Durch das OCR-Verfahren sollte eine Volltextsuche ermöglicht werden. Motiviert durch das *one million book*-Projekt und die sinkenden Speicherpreise, begann Bell 1998 seine Unterlagen zu digitalisieren [55]. Hierfür wurde eine Assistentin beauftragt, sämtliche Bücher und Dokumente zu scannen. Ab sofort erfasste Bell sein Leben mit digitalen Aufzeichnungsgeräten. Doch konnte Bell mit den Tausenden digitalisierten Dokumenten anfangs nichts anfangen [56]. Inspiriert von Bushs Vision und der Notwendigkeit, Daten zu verwalten, entstand das Forschungsprojekt MyLifeBits. Dabei wurden viele Funktionen von MEMEX, wie zum Beispiel das Hinzufügen von Anmerkungen realisiert. Durch die Integration von Audio und Video erweiterte Bell das Konzept von MEMEX. Im Laufe des Projektes entstanden neue Ideen und weitere Daten wie E-Mail-Kommunikation, Tasks und Notizen wurden hinzugefügt. Zusätzlich zu Text-Anmerkungen waren auch Audio Anmerkungen möglich. Damit die Audio-Daten durchsuchbar sind, wurden diese mithilfe einer Spracherkennung in Text umgewandelt. Auch wurden später die Aufnahmen einer eigens entwickelten Technologie namens *SenseCam* (siehe Kapitel 3.2.2, Seite 33), einer um den Hals getragenen Kamera, integriert.

2.3 Ortsbestimmung

Die wichtigsten Informationen für Lifelogging sind Zeit und Ort. Im Gegensatz zur Bestimmung der Zeit ist die Ortsbestimmung aufwendig. Die Lokalisierung ist ein wichtiger Bestandteil der vorliegenden Arbeit. Daher wird im Folgenden auf mögliche Lokalisierungsmethoden eingegangen.

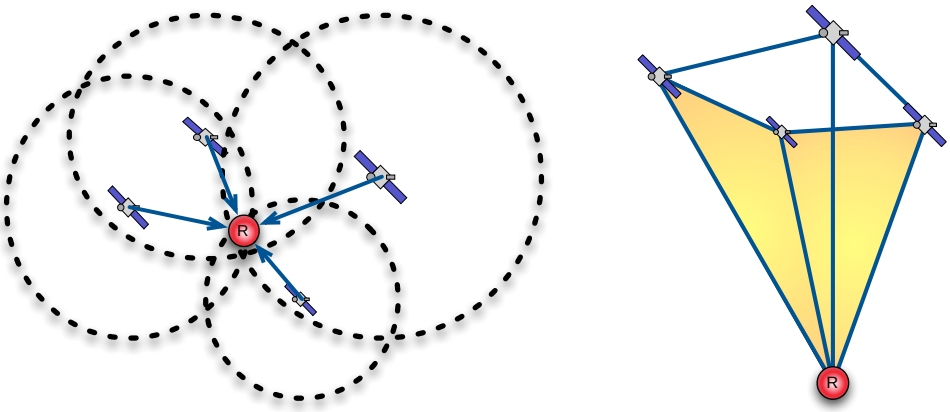
2.3.1 Global

Für eine großflächige Ortung auf der Erdoberfläche werden Satelliten verwendet. Durch Ermittlung von Laufzeit und Lokalisierung der einzelnen Satelliten-Signale kann auf den Ort geschlossen werden [57]. GPS erreicht im zivilen Bereich mit dem C/A Code[2] eine Genauigkeit von $\pm 15\ m$. Bei einer Ausbreitungsgeschwindigkeit von circa $300.000.000\ m/s$ entspricht dies einer Laufzeitabweichung von circa einer milliardstel Sekunde. Die GPS-Satelliten sind mit Atomuhren ausgerüstet, Quarzuhren in den GPS-Empfangsgeräten bieten diese Genauigkeit nicht. Deswegen sind für eine Lokalisierung mindestens vier Satelliten notwendig; drei damit die 2D-Ortung auf der Erdoberfläche ermittelt werden kann, und der vierte zur Ermittlung der exakten Uhrzeit oder 3D-Ortung (siehe Abbildung 2.11a). Der Sendezeitpunkt wird durch Lösung eines Gleichungssystems mit vier Gleichungen und vier Unbekannten ermittelt. Die Position der Satelliten spielt eine wichtige Rolle. Grundsätzlich gilt, dass die Genauigkeit proportional zum aufgespannten Volumen zwischen den Satelliten und dem Empfänger ist (siehe Abbildung 2.11b). Für Smartphones werden mithilfe einer zusätzlichen Datenverbindung Hilfsinformationen (Assisted Global Positioning System, AGPS) wie die ungefähre Aufenthaltsort basierend auf dem Funknetz gesendet. Mit diesen Informationen ist es möglich, die Zeit bis zur ersten Lokalisierung drastisch zu verkürzen. Abhängig von der letzten Benutzung und Standortänderung kann diese über zehn Minuten betragen. Ein Nachteil bei satellitengestützten Systemen ist, dass meist eine direkte Sichtverbindung zum Himmel erforderlich ist.

2.3.2 Lokal

Im Gegensatz zu globalen Navigations-Satellitensystemen verwenden lokale Lokalisierungsverfahren Information wie Signalstärke und Standort-IDs von Funksendern. Der Vorteil dabei ist, dass ebenfalls eine Innenraum-Lokalisierung ermöglicht wird. Dazu werden vom Smartphone die verfügbaren IDs, Signalstärken und weitere Informationen der Handy-Sendemasten an einen Server geschickt und ausgewertet. Der Verbund von mehreren Sendetechnologien wie Handy Sendemasten und WLAN-Access-Points ist ebenfalls gängig [58]. Standardmäßig werden für die Bestimmung in einem dreidimensionalen Raum immer vier paarweise verschiedene Referenzpunkte benötigt [59]. Die Genauigkeit der Lokalisierung ist hauptsächlich von der Anzahl der verfügbaren Sender und deren Abschattung abhängig. WLAN-Lokalisierungsverfahren bieten gegenüber den Handy-Sendemasten Verfahren eine genauere Standortbestimmung, sind aber nicht flächendeckend verfügbar. Gängige iPhone, Android (siehe Abbildung 2.12) und Windows-Smartphones wie auch neuere tragbare GPS-Tracker verwenden bereits eine Kombination aus GPS, Funkortung und WLAN-Lokalisierung.

[2] Allgemein nutzbare Codierung *Coarse/Acquisition*, ermöglicht eine eingeschränkt genaue Ortung.



(a) Dreidimensionale Ortung mithilfe von vier Satelliten. Der Schnittpunkt R ergibt die Lokalisierung des GPS-Empfängers. (In Anlehnung an: [57] S. 29)

(b) Resultierender Tetrahedron zwischen Satelliten- und GPS-Empfänger R. (In Anlehnung an: [57] S. 122)

Abbildung 2.11: Ortung mithilfe von vier Satelliten und Genauigkeit bezüglich aufgespanntem Volumen.

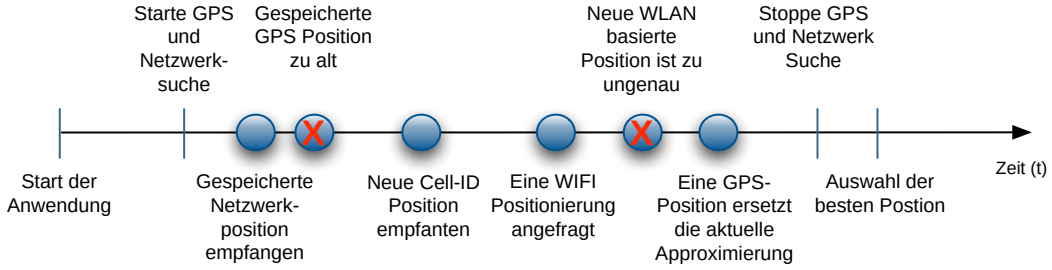


Abbildung 2.12: Strategie zur Lokalisierung in Android. (In Anlehnung an: [60])

Indoor Bluetooth-Navigation bietet die genaueste Lokalisierung in Räumen, wird aber wegen der notwendigen Bluetooth Infrastruktur meist nur für spezielle Einsatzzwecke verwendet [48]. Google bietet ab *Google Maps 6.0* für Android bereits eine WLAN-basierte Innenraumnavigation an. Nokia Research beschäftigt sich mit der auf Bluetooth 4.0 Technologie basierten Innen-Navigation und mit der Forcierung eines internationalen Standards [61]. Dabei wird mit einer Genauigkeit von bis zu $30cm$ geworben [62].

2.4 Nachverarbeitung und Visualisierung

Große Datenansammlungen, wie Bild- oder Videoaufzeichnungen können vom Menschen nur schwer verarbeitet werden. Um beispielsweise Fotos aus fünfzig Jahren Aufzeichnung durchzusehen, würde dies selbst bei schnellem Abspielen geraume Zeit beanspruchen. Lifelogging erzeugt große Datenmengen, daher ist eine Indizierung und Filterung der Daten notwendig um die Datenmenge zu reduzieren. Geeignete Visualisierungsmethoden sind notwendig, um einen schnellen und effizienten Zugriff auf die Daten zu erhalten. In weiterer Folge ist der Aspekt der Sicherheit und Anonymisierung wichtig. Im folgenden werden die Indizierung durch Analyse der Inhalte, eine geeignete Visualisierung und Anonymisierung besprochen.

2.4.1 Indizierung durch inhaltliche Analyse

Die Klassifizierung und resultierende Indizierung der Daten kann auf verschiedene Arten erfolgen. Beispielsweise könnten Audio-Daten ohne zeitlichen Zusammenhang in Szenarien wie *kein Gespräch* und *Gespräch* mithilfe eines Schwellwerts eingeteilt werden. Einfache portable Audio-Aufzeichnungsgeräte nutzen diese Vorgehensweise, um die Aufnahme zu steuern. Für eine zeitliche Segmentierung ist hingegen meist der Informationsunterschied aufeinanderfolgender Daten relevant, beispielsweise bei der Eventdetektion von Überwachungsbildern. Durch semantische Interpretation von Szenen können die Daten ebenfalls klassifiziert werden. Ein visuelles Beispiel ist die Indizierung der Szenarien *Stadt* und *Land* anhand von Visual-Words [63]. Ein auditives Beispiel ist die Indizierung von Szenen anhand eines Zuhörermodelles [64].

Für Bilderfolgen ist einer der einfachsten Ansätze, aufeinanderfolgende Bilder zu vergleichen. Anhand eines Ähnlichkeitsmaßes kann entschieden werden, welche Bilder ähnlich sind und vermutlich denselben Inhalt repräsentieren (siehe Abbildung 2.13). Es gibt verschiedene Ähnlichkeitsmaße, ein simpler Ansatz betrachtet die absolute Pixeldifferenz von zwei Bildern. Dies funktioniert jedoch nur bei stationären Systemen, denn durch eine leichte Bewegung der Kamera können sämtliche Pixel verändert werden [65]. Ein anderer Ansatz schließt auf ähnliche Bilder, indem die einzelnen RGB-Histogramme verglichen werden (Vergleich zwischen Pixel- und Histogrammdifferenz siehe Abbildung 2.14). Der Vorteil dabei ist, dass eine geringe perspektivische Veränderung kaum Auswirkungen auf das Histogramm hat. Jedoch besteht der Nachteil, dass komplett verschiedene Bilder die gleiche Farbverteilung haben können [66]. Ein weiterer Makel ist, dass unterschiedliche Belichtungen sich stark auf die Farbverteilung auswirken können. Durch den Vergleich aufeinanderfolgender Bilder ist nicht immer eine korrekte Entscheidung möglich. Beispielsweise können, wie in Abbildung 2.15 dargestellt, kurzzeitige, starke Veränderungen fehlerhafte Entscheidungen verursachen.

Aiden R. Doherty und andere beschäftigten sich mit dem automatischen Eventclustering von SenseCam-Bildern [67]. Das Ziel war eine effizientere Berechnung der Eventgrenzen. Dabei verwendet Doherty eine Kombination aus MPEG-7-Merkmalen und SenseCam-Datenvektoren. Jedes einzelne Bild wird durch eine Kombination dieser Daten repräsentiert. Damit die einzelnen Merkmale gleichberechtigten Einfluss erzielen, wird zuerst eine Normalisierung durchgeführt. Anschließend ermittelt Doherty die Ähnlichkeit durch einen Vergleich benachbarter Vektoren. Ein Vorteil dieser Variante ist, dass eine schnellere Berechnung erfolgt, und nur gering schlechtere Ergebnisse erzielt werden. In einem weiteren Ansatz beschäftigt sich Doherty ebenfalls

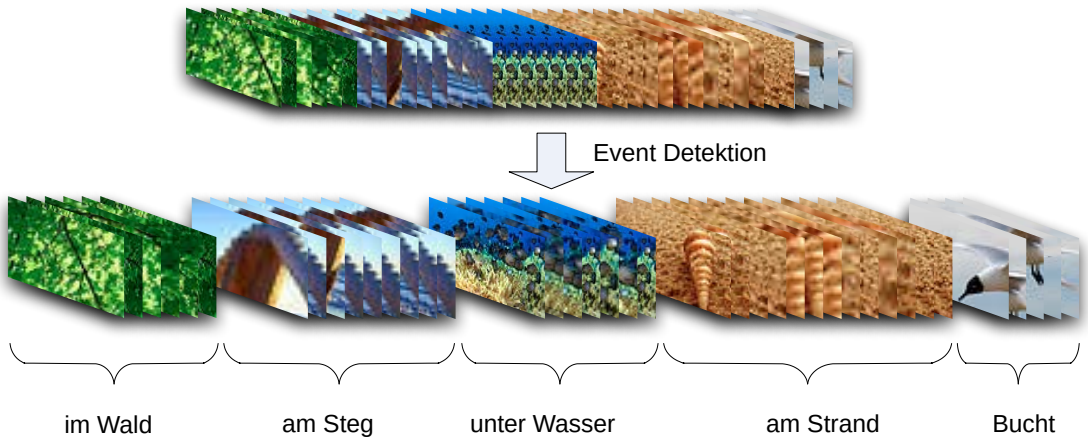


Abbildung 2.13: Eventdetektion und Gruppierung zu Blöcken.

mit der Eventdetektion von SenseCam-Bildern, diesmal aber in Kombination mit iRiver MP3-Audioaufnahmen [68]. Dabei testet Doherty zuerst das Potenzial jedes einzelnen verfügbaren Merkmals und anschießend Kombinationen der Merkmale. Doherty erkannte, dass die Betrachtung aufeinanderfolgender Bilder zu Fehlern führen kann, zum Beispiel wie in Abbildung 2.15 ersichtlich, wo die Person für einen Augenblick zum Himmel aufblickte und eine Möwe sah. Zur Vermeidung dieses Problems verfolgt Doherty den Ansatz, Bildblöcke miteinander zu vergleichen. Wie in Abbildung 2.16 zu sehen, werden beispielsweise Blöcke von je drei Bildern miteinander verglichen. Weiter untersucht Doherty die Möglichkeit, anhand der Audiodaten Events zu detektieren. Um eine kompakte Repräsentation zu erhalten und kurzzeitige Events zu glätten, erfolgt eine Einteilung der Audioaufnahme in eine Minute lange Blöcke. Zur Bestimmung von Eventgrenzen benutzt Doherty das *Bayesian Information Criteria*, welches basierend auf einer Wahrscheinlichkeitsfunktion Entscheidungen trifft. Ebenfalls versucht Doherty, anhand der Temperatur Eventgrenzen zu detektieren. Dazu wird die Varianz der Temperatur berechnet und mithilfe eines Schwellwerts entschieden, ob es sich um einen neuen Event handelt. Doherty testet auch die Möglichkeit, anhand der gemessenen Lichtintensität Events zu detektieren. Beispielsweise ändert sich die Helligkeit, wenn eine Person einen helleren Raum betritt. Kleinere Bewegungen innerhalb desselben Raumes können ebenfalls zu einer Abschattung des Sensors und somit zu einer fehlerhaften Interpretation führen. Mithilfe von Beschleunigungsdaten testet Doherty ebenfalls die Eventerkennung. Dazu werden zuerst die einzelnen Achsen der Beschleunigungswerte abgeleitet und mittels euklidischer Norm zu einem eindimensionalen Vektor kombiniert. Danach eintscheidet einen Schwellwert, ob es sich um eine Eventgrenze handelt. Beispielsweise der Gang vom Büro zum Mittagessen. Abschließend testet Doherty eine Fusion der einzelnen Merkmale mithilfe eines *CombMNZ* Boosting-Ansatzes. Dabei werden alle möglichen Kombinationen der Merkmale getestet.

Michael Blighe und andere beschäftigen sich mit der automatischen visuellen Erkennung verschiedener Lokalitäten, wie Esszimmer, Büro oder Küche [69]. Mithilfe des Scale-Invariant Feature Transform (SIFT)-Algorithmus extrahieren sie dazu beleuchtungs-, rotations-,

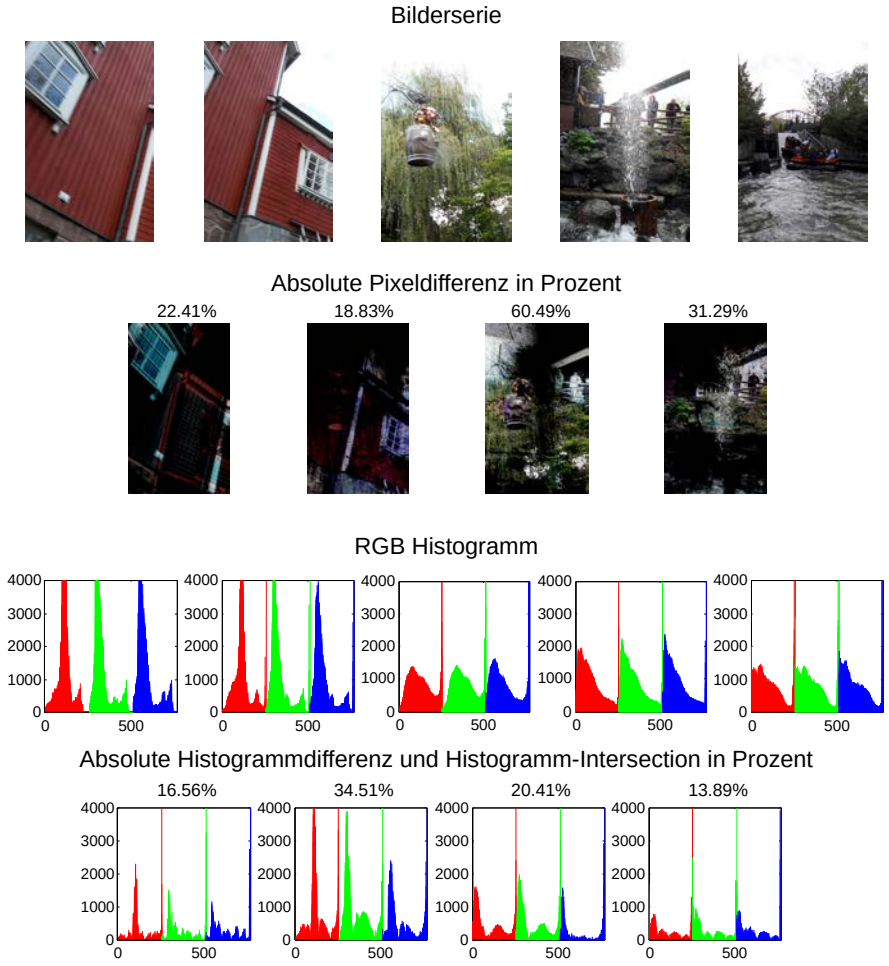


Abbildung 2.14: Bildfolge mit Histogramm und Differenzen.

skalierungs- und translationsunabhängige Merkmale [70]. Durch eine X-Means-Clusteranalyse werden die einzelnen Räumlichkeiten anhand ihrer Merkmale unterteilt. In einem darauffolgenden Ansatz wird die Keyframe-Extraktion von SenseCam-Bildern erforscht. Zuerst erfolgt mittels SIFT-Merkmalen eine Gruppierung ähnlicher Bilder. Danach werden innerhalb dieser Gruppen jene Frames ausgewählt, welche die repräsentativsten SIFT-Merkmale beinhalten [71]. Somit erfolgt eine Eventdetektion wobei die einzelnen Events mit einem repräsentativen Bild (Keyframe) repräsentiert werden. In einem weiteren Ansatz zeigt Blighe dass im Falle von SenseCam-Bildern Speeded Up Robust Features (SURF)-Merkmale (vereinfachte SIFT-Merkmale) gegenüber SIFT-Merkmale vorteilhafter sind [69]. Beispielsweise sind sie robuster gegenüber Bildrauschen. Weiters zeigt Blighe auch, dass mit SURF-Merkmalen eine genauere bildbasierte Lokalisierung erfolgen kann.

Hazem Wannous und andere erarbeiteten einen Ansatz, der Orte anhand von vorhandenen 3D-Modellen erkennt [72]. Dazu wird mithilfe der *Structure from Motion*-Technologie aus den

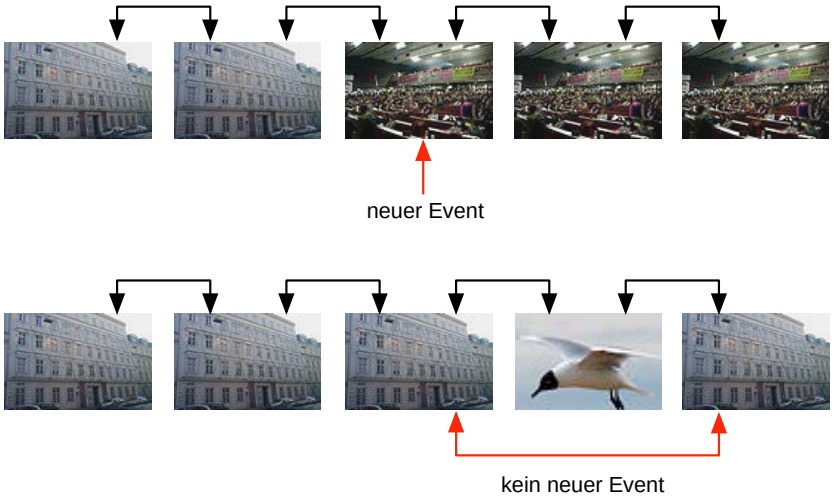
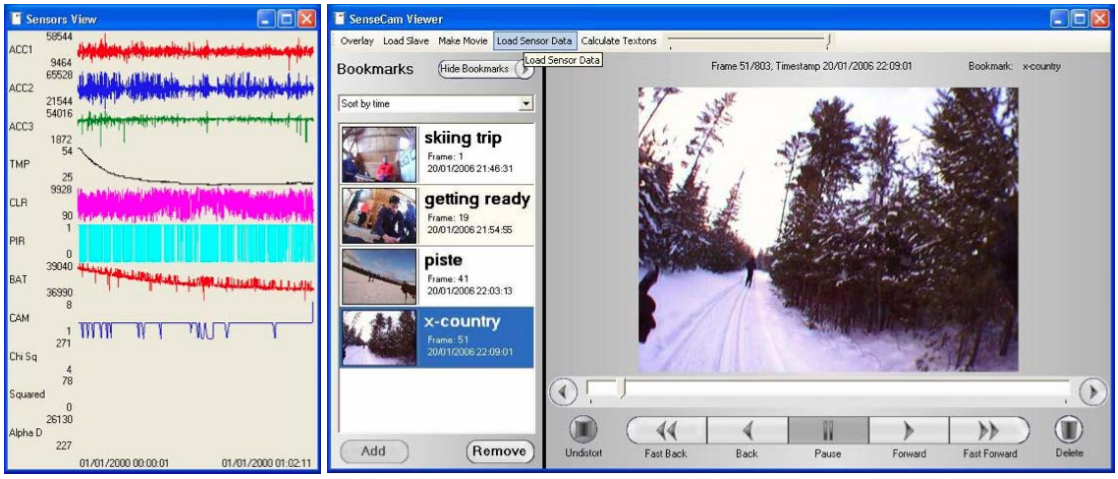


Abbildung 2.15: Fehlerhafte Eventdetektion (In Anlehnung an: [68]).

Abbildung 2.16: Vergleich mithilfe von Bildblöcken (In Anlehnung an: [68]).

Bildern eines 2D-Videos ein 3D-Modell erzeugt. Folgend werden SIFT-Merkmale extrahiert und korrelierende Punkte mithilfe des RANSAC-Algorithmus ermittelt. Aus der Summe der zusammengehörigen Bilder werden dann die internen und externen Kameraparameter geschätzt. Aus der resultierenden Fundamentalmatrix wird die relative Lage der einzelnen korrespondierenden Punkte berechnet, ein 3D-Modell erstellt und mit den vorhandenen Modellen verglichen.

Projekte wie *MyLifeBits* oder *Memory Prosthesis* beschäftigen sich bereits mit der Textextraktion von Audiodaten. Basierend auf der Extraktion von menschlicher Sprache kann ein Text extrahiert und für eine spätere Suche verwendet werden. Gängige Spracherkennungsalgorithmen extrahieren dabei zuerst MFCC-Merkmale und klassifizieren einzelne Wörter anhand von Hidden Markov Model (HMM)-Klassifikatoren. MFCC-Merkmale bieten eine kompakte Darstellung des Frequenzspektrums und spiegeln die Wahrnehmung des menschlichen Gehörs wieder. Schließlich werden Wörter anhand von HMM-Zustandsautomaten mit Übergangswahrscheinlichkeiten ermittelt [73].



(a) Zeitliche Visualisierung der Sensordaten (Quelle: [74] S. 9).

(b) Aktuelle aktiver Sensor (Quelle: [74] S. 8).

Abbildung 2.17: Zeitrafferansicht von *SenseCam*-Bildern.

2.4.2 Visualisierung

Eine Visualisierung der Daten erleichtert den Überblick und ermöglicht einen schnellen und effizienten Zugriff. Ebenso kann eine geeignete Darstellung der Daten neue Informationen liefern. Beispielsweise kann der durchschnittlich zurückgelegte Weg eine Aussage über die Fitness liefern. Eine einfache Methode ist es, Rohdaten auf einer Zeitachse zu visualisieren.

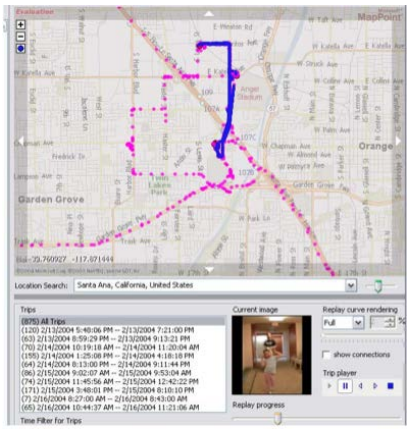
Im Projekt *MyLifeBits* wurden bereits mehrere Visualisierungsmethoden realisiert. Die *SenseCam* bietet (siehe Abbildung 2.17) eine zeitliche Visualisierung der Sensordaten und das Abspielen eines Zeitraffers. *MyLifeBits* bietet zudem eine Kartenansicht, auf der GPS-Positionen eingetragen, und zeitlich sortiert werden.

Alan F. Smeaton realisierte eine Visualisierung von *SenseCam*-Bildern bei der die Bildergröße der Eventlänge entspricht [77]. Das Besondere dabei ist, dass eine Gruppierung in einzelne Events erfolgt. Folgend werden die einzelnen Events durch Bilder repräsentiert (siehe Abbildung 2.19). Die Größe der Bilder spiegelt die Eventdauer wieder, größere Bilder repräsentieren längere Events.

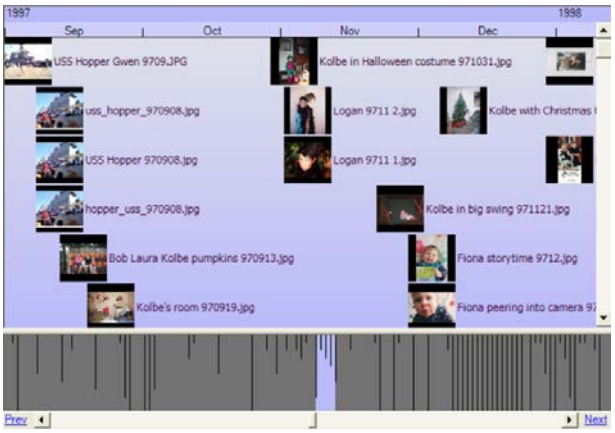
Aaron Parecki realisierte eine orts- und zeitspezifische Visualisierung aus zweieinhalb Millionen GPS-Lokalisierungen [78]. Dabei entsprechen die Farben dem Jahr und die Helligkeit der Anzahl der Aufenthalte. Die Visualisierung zeigt hauptsächlich Straßen und Aufenthaltsorte, die oft besucht werden.

2.4.3 Anonymisierung

Falls eine Anonymisierung der Daten erwünscht ist, können beispielsweise die Daten verzerrt oder überlagert werden. Nijhuis und andere befassen sich mit der Erkennung von Auto Nummerntafeln [79]. Zuerst werden Störungen entfernt, und der Kontrast angepasst. Danach folgt



(a) Visualisierung von Positionen auf einer Karte. Rosa Punkte zeigen GPS-Lokalisierungen. Rote Punkte markieren Fotos und blaue Line visualisiert eine ausgewählte Route. (Quelle: [75] S. 7).



(b) Ansicht als Zeitachse. Die Unterteilung kann frei in Stunden, Tage, Wochen oder Monate unterteilt werden. Zur Navigation dient die Untere Vorschauleiste (Quelle: [76] S. 3).

Abbildung 2.18: Weitere Visualisierungsmethoden von *MyLifeBits*.

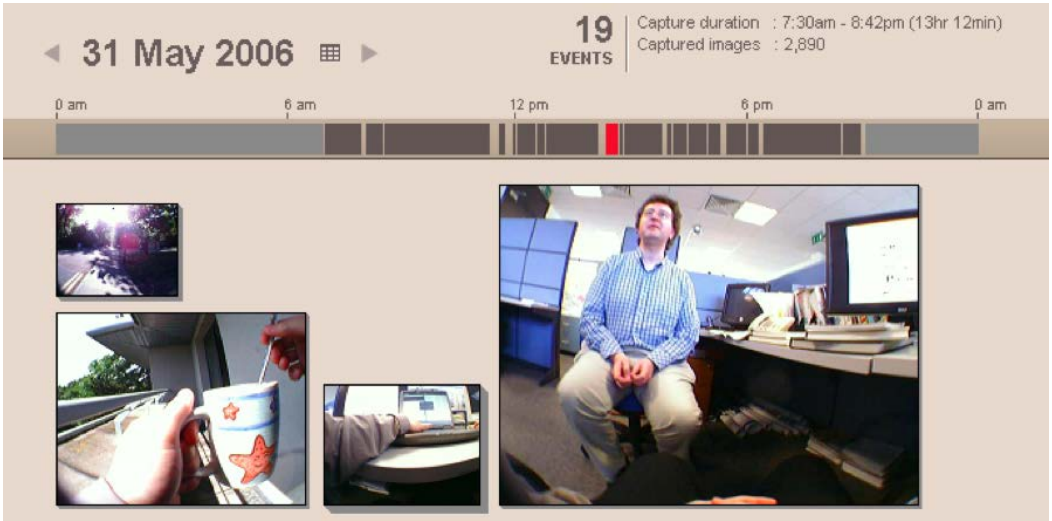


Abbildung 2.19: Visualisierung von Bildern anhand ihrer Eventlänge (Quelle: [77] S. 8).

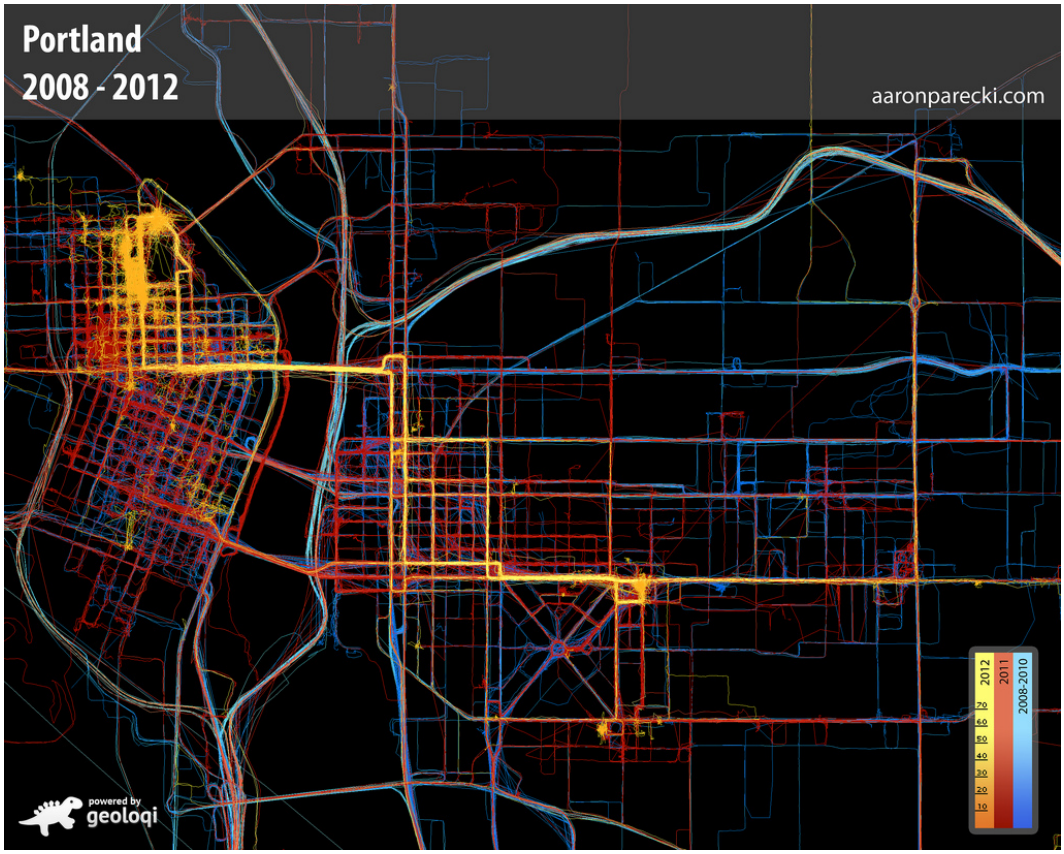


Abbildung 2.20: Orts- und zeitspezifische Visualisierung von GPS-Positionen (Quelle: [78]).

eine Segmentierung durch fuzzy C-Means-Clustering in zwei Segmente: Nummerntafel und Sonstiges. Durch die Lokalisierung können die Bilder durch Überblenden oder Verzerren der Nummerntafeln anonymisiert werden indem.

Jayashri Chaudhari und andere beschäftigen sich mit der automatischen Anonymisierung von Lifeloggingdaten [80]. Für die Verschleierung von Gesichtern bedient sich Chaudhari des Viola-Jones Gesichtsdetektors [81]. Als Resultat überdeckten sie ähnlich wie in Abbildung 2.21 die Gesichter mit Vierecken.

Chaudhari und andere erforschten der Verschleierung von Audio-Daten [80]. Dazu wird zuerst erkannt, wann Personen sprechen. Dies funktioniert anhand unabhängig vom Textverständnis basierenden Spracherkennungssystem. Danach werden die entsprechenden Passagen mithilfe eines Pitch Scale SOLA Sprach-Verzerrungsalgorithmus unkenntlich gemacht.

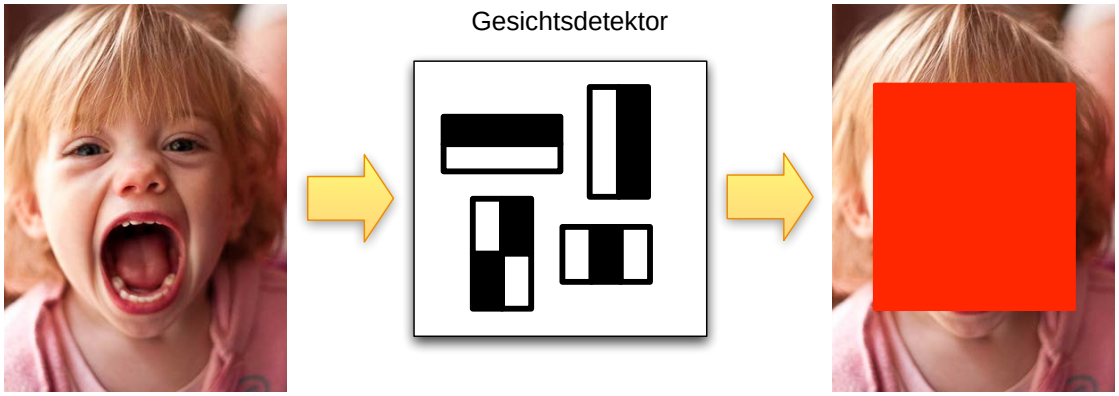


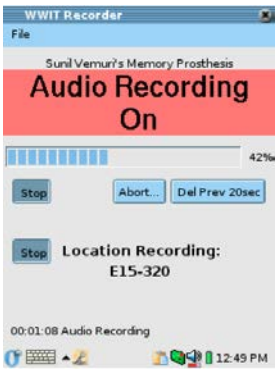
Abbildung 2.21: Gesichtserkennung und Verschleierung mithilfe eines Gesichtsdetektors.

KAPITEL 3

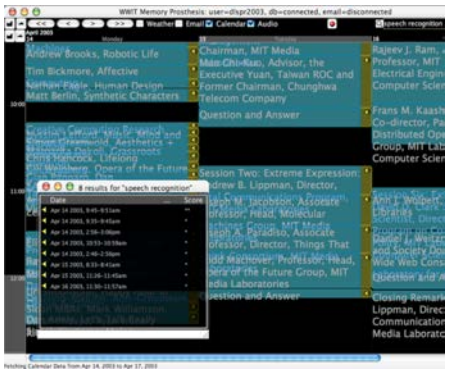
Lifelogging Ansätze

Bei den zuvor genannten Lifeloggingsystemen handelt es sich um erste Prototypen, die teilweise noch ohne permanente Aufzeichnung oder nur textuell funktionierten. In diesem Kapitel werden multimediale Lifeloggingprojekte, strukturiert nach Sensormodalität, vorgestellt.

(a) PDA mit Memory Prosthesis Software . (Quelle: [82] S. 5)



(b) Grafische Benutzeroberfläche von Memory Prosthesis PDA. (Quelle: [82] S. 5)



(c) Memory Prosthesis MACOS GUI. (Quelle: [82] S. 6)

Abbildung 3.1: Memory Prosthesis Audio Lifeloggingsystems.

3.1 Auditiver Ansatz

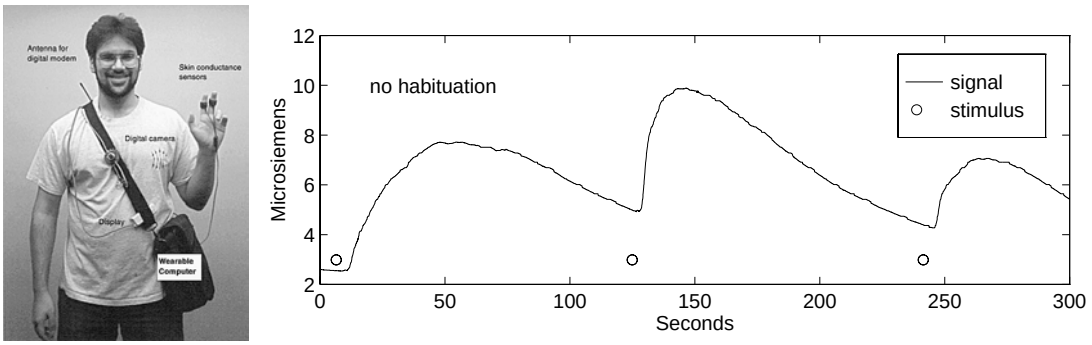
3.1.1 Memory Prosthesis

Sunil Vemuri und andere realisierten 2004 *Memory Prosthesis*, ein mobiles audiobasiertes Lifeloggingsystem [82]. Das Ziel war ein audiobasierter persönlicher Assistent, der Gespräche aufnimmt, durchsuchbar macht und wiedergibt. Die Erfassung der Audiodaten erfolgt mithilfe eines PDAs. Dabei wird die Position anhand von WLAN-Signalen ermittelt. In den Abbildungen 3.1 a und 3.1 b sind der PDA und die grafische Benutzeroberfläche von Memory Prosthesis dargestellt. Aus den aufgenommenen Audiodateien wird mithilfe einer Spracherkennung der Text zugänglich gemacht. Dies wird durchgeführt, damit eine spätere Volltextsuche sämtlicher Aufzeichnungen möglich ist. Anschließend werden die Audiodaten mit Zeit, Position und extrahiertem Text an einen Server übermittelt. Über den PDA und eine Desktopanwendung (siehe Abbildung 3.1c) kann eine Suche nach dem extrahierten Text, Ort und Zeit erfolgen. Das Suchresultat bietet dann die Möglichkeit, die Audioaufnahme abzuspielen.

3.2 Visuelle Ansätze

3.2.1 StartleCam

J. Healey und R. Picard präsentierten 1998 ein tragbares Kamerasystem (siehe Abbildung 3.2a), bestehend aus einem umhängbaren Computer, einer Videokamera, einem Funkmodem und einem ProComp+ Biofeedback Messgerät [83]. Bisherige Kamerasysteme erfassen Bilder entweder durch den Benutzer oder durch ein Computerprogramm. Die Idee und der Name der StartleCam stammte von *startle response*, der Schreckreaktion. Durch die Detektion von Stressfaktoren sollen wichtige Situationen erkannt und mithilfe eines Kamerasystems festgehalten werden. Dazu wird die physiologische Erregung gemessen, welche durch Emotionen oder Stress des Trägers



(a) Ausstattung des Systems. (Quelle: [83] S. 2)

(b) Zeitlicher Verlauf und der Gewöhnungseffekt eines wiederholenden Stimulus. (Quelle: [83] S. 3)

Abbildung 3.2: StartleCam Prototyp mit Verlauf des Hautwiderstandes.

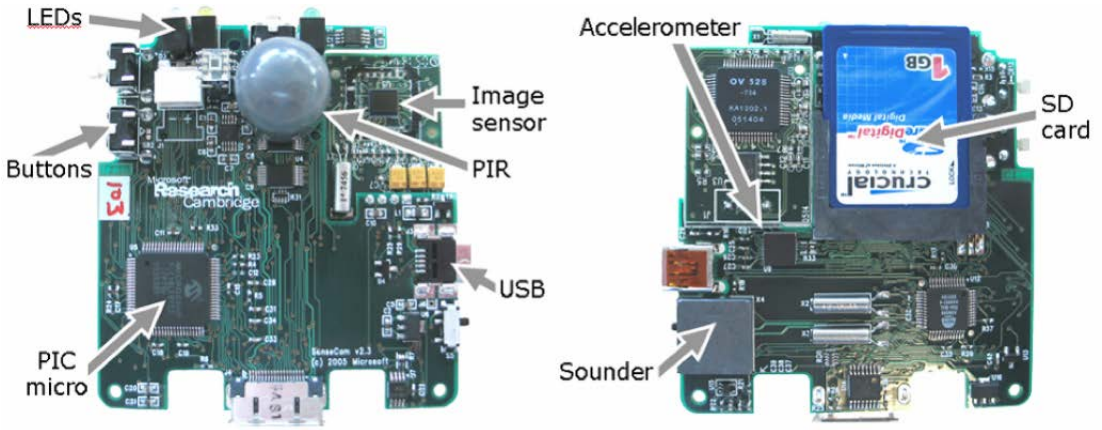


Abbildung 3.3: Der interne Aufbau der SenseCam. (Quelle: [9] S. 183)

auslöst wird. Ein Stimulus verursacht einen kurzzeitigen Anstieg des Hautwiderstands, welcher mithilfe das Messgeräts ProComp+ gemessen wird. Zur Detektion einer Erregung wird das Signal (siehe Abbildung 3.2b) mit einem typischen Erregungssignal verglichen. Bei genügend hoher Korrelation löst die Kamera aus. Über das Funkmodem können dann die Bilder direkt in das Internet übertragen werden.

3.2.2 SenseCam

Lyndsay Williams entwickelte 2004 den ersten Prototypen namens *SenseCam*. Dabei handelt es sich um eine tragbare Kamera (siehe Abbildung 3.4a), die an einer Kordel befestigt um den Hals getragen wird (siehe Abbildung 3.4b) und automatisch Bilder aufnimmt. Durch die kompakte Bauform und ein Gewicht von nur circa 94 g ist das Gerät bequem um den Hals tragbar (siehe Abbildung 3.4a und Abbildung 3.4b). Auf der Vorderseite des Gerätes befinden sich, wie

(a) Prototyp der SenseCam. (Quelle: [9] S. 180)

(b) Ein um den Hals getragener SenseCam-Prototyp. (Quelle: [9] S. 180)

(c) Weitwinkel Aufnahme der SenseCam. (Quelle: [9] S. 180)

Abbildung 3.4: Aussehen, Trageweise und Bilder von SenseCam.

in Abbildung 3.3, eine Kamera, ein passiver Infrarotsensor (PIR), ein RGB-sensitiver Helligkeitssensor und ein Temperatursensor. Da die *SenseCam* keinen Sucher zum Anvisieren auf ein Objekt besitzt, und die Aufnahme meist ohne Interaktion des Benutzers geschieht, wurde eine Kamera mit Weitwinkelobjektiv verbaut. In Abbildung 3.4c ist eine Aufnahme der *SenseCam* dargestellt. Der passive Infrarotsensor misst die Wärmestrahlung, und bei einer signifikanten Änderung, wie zum Beispiel der Begegnung mit einem anderen Menschen, wird ein Bild aufgenommen. Der Helligkeitssensor kann bei einer starken Änderung ebenfalls eine Aufnahme auslösen. Aus der Sicht des Trägers befinden sich auf der rechten Seite der *SenseCam* zwei Tasten. Die obere *privacy*-Taste pausiert die automatische Aufnahme, die untere *manual*-Taste löst manuell eine Aufnahme aus. Auf der oberen Seite befinden sich eine Taste um die *SenseCam* ein-beziehungsweise auszuschalten und drei LEDs, die den aktuellen Status visualisieren. Die SenseCam-Technologie wurde 2009 von Vicon lizenziert, und ist unter dem Namen Vicon Revue erhältlich.

3.2.3 Project Glass

Google präsentierte 2012 das aktuelle Forschungsprojekt *Project Glass* in Form einer tragbaren Brille (siehe Abbildung 3.5). Diese fungiert als externes Display und kann Bilder aufnehmen. Erste Konzepte wurden bereits in einem Video veröffentlicht und zeigen Anwendungen, die das Wetter, Fußgängernavigation oder die aktuelle Position über das transparente Display einblenden [28]. Die Datenbrille verfügt laut der amerikanischen Zulassungsbehörde *FCC* über WLAN- und Bluetoothdatenanbindung [84]. Die erste Auslieferung der Google Glass-Brillen soll noch 2013 an Entwickler erfolgen.

(a) Google Glasses Prototyp (schwarz). (Quelle: [85])

(b) Visualisierung einer AR Navigation. (Quelle: [86])

Abbildung 3.5: Google Project Glass Prototypen.

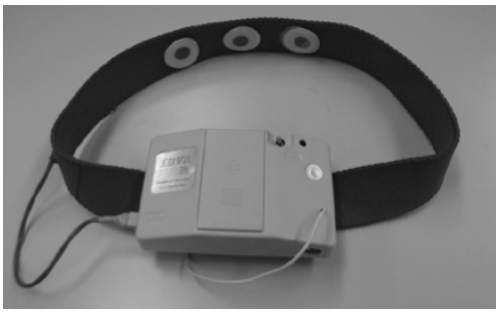


Abbildung 3.6: Tragbares Gehirn-aktivitäts-Messgerät. (Quelle: [87] S. 399)

3.3 Audiovisuelle Ansätze

3.3.1 Aizawa

Kiyoharu Aizawa und andere begannen 2001 mit den ersten Versuchen eines Audio- und Video-Lifelogs [87]. Das Ziel war die Erfassung des menschlichen Lebens mithilfe einer kontinuierlichen Audio- und Videoaufzeichnung. Zusätzlich zeichneten sie die Gehirnaktivität mit einem tragbaren Sensor auf (siehe Kapitel 2.4, Seite 23). 2003 erweiterten sie das System mit GPS-, Beschleunigungs- und Gyroskop-Sensoren [88]. Bisherige Systeme wie die SenseCam konzentrierten sich primär auf die Aufnahme einzelner Standbilder. Im Vergleich zum Projekt MyLife-Bits, das vermehrt Daten des Büroalltages erfasste, sollte das System allgegenwärtige Lebenserfahrung multimodal erfassen. Einzelne Modalitäten wurden mit verschiedenen Geräten aufgezeichnet und später auf einem Computer ausgewertet. Das überarbeitete System bestand, wie in Abbildung 3.7 ersichtlich, aus einem kompakten PC, einer tragbaren Kamera, einem Mikrofon, Beschleunigungssensor, GPS-Empfänger und Bio-Sensoren. Diese Bio-Sensoren messten Hautwiderstand, Hauttemperatur, Umgebungstemperatur und Wärmestromdichte.

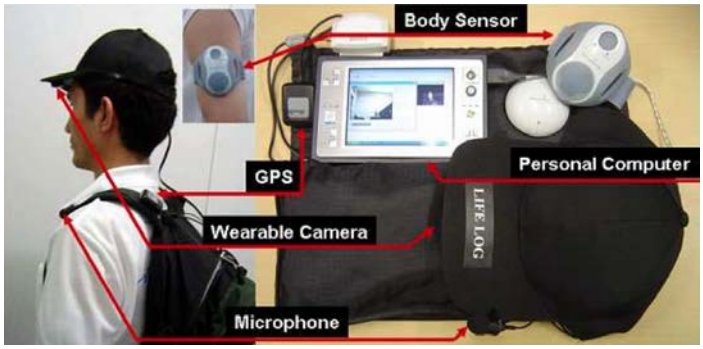


Abbildung 3.7: Aizawa Lifeloggingsystemkomponenten. (Quelle: [88] S. 166)

3.3.2 iGlogger

Raymond Chun Hing Lo wurde 2010 im Rahmen seiner Masterarbeit von Prof. Steve Mann betreut und erstellte ein Framework für Echtzeit-Cyborglogging [89]. Dabei implementierte Hing Lo mehrere Prototypen für die Plattformen Symbian, iOS und Java Micro Edition (siehe Abbildung 3.8). Die Aufnahme der Kamera wird durch drei verschiedene Algorithmen gesteuert. Der erste Algorithmus erkennt eine signifikante Änderung des Beschleunigungssensors wie etwa Schütteln. Der zweite Algorithmus löst bei lauten Umgebungsgeräuschen aus. Der dritte Algorithmus beobachtet die Lage des Mobiltelefons. Sobald die Lageänderung größer ist als das Blickfeld der Kamera, löst diese aus. Um sicherzustellen, dass die Bildaufnahme nicht verschwommen erfolgt, wird der Beschleunigungssensor verwendet. Der Shutter löst erst dann aus, wenn das Smartphone eine stabile Lage einnimmt. Abhängig von der Plattform sind Audio, Video, Kompass, Beschleunigung und GPS-Daten erfassbar und können an einen Webservice übermittelt werden. Eine öffentliche Version des Frameworks ist auf einer Glogger Webseite `http://glogger.mobi` verfügbar. Dort können Bilder und Videos auf einer Landkarte oder in Form eines Storyboards öffentlich geteilt werden.

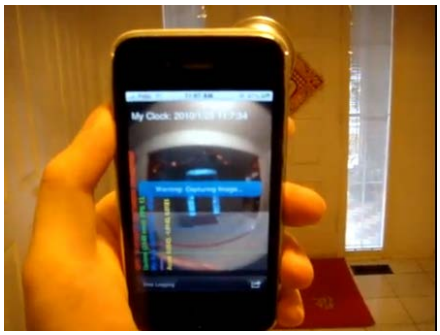
3.3.3 BlackBerry WMMS

Hui Hsien Wu entwickelte im Rahmen seiner Masterarbeit ein einfaches Blackberry basiertes Wearable Mobility Monitoring System (WMMS) [91]. Das System besteht aus einem Blackberry und einem externen Board (siehe Abbildung 3.9). Der Quellcode befindet sich im Anhang seiner Arbeit. Ziel waren die Entwicklung und die Evaluierung eines Lifeloggingprototypen. Dabei sollten anhand der Daten Zustände des täglichen Lebens wie Stehen, Gehen, Liegen, Lift fahren, Autofahrt, Treppensteigen, Gehen auf einer Rampe oder die Zubereitung von Mahlzeiten ermittelt werden.

Die Daten werden dabei jede Sekunde erfasst und im CSV-Dateiformat auf eine SD Card gespeichert. Die Auswertung erfolgt mit Unterstützung von Excel und Matlab. Als Resultat entstand ein hierarchischer Entscheidungsbaum. Durch Aussortierung schlecht ermittelbarer Situationen konnte eine Genauigkeit und Sensitivität von nahezu 100 Prozent erreicht werden. Die Akkulaufzeit des Systems beträgt ungefähr drei Stunden.

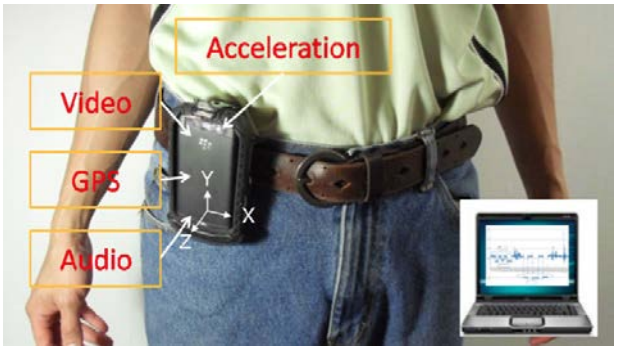
(a) Betrieb von iGlogger auf einem J2ME Smartphone. (Quelle: [90])

(b) J2ME GUI von iGlogger. (Quelle: [90])



(c) Betrieb von iGlogger auf einem iPhone 3GS. (Quelle: [90])

Abbildung 3.8: iGlogger System auf verschiedenen Smartphones.



(a) Blackberry mit Gürtelhalterung. (Quelle: [91] S. 31)

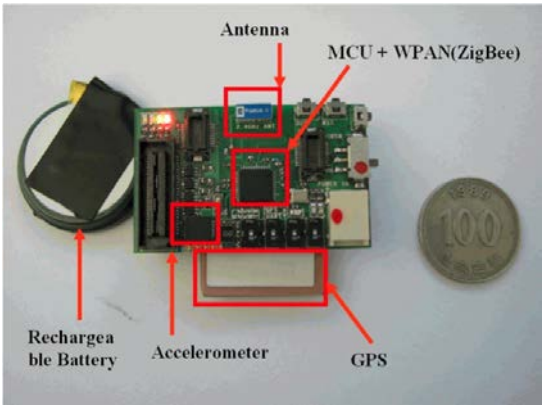
(b) Blackberry inklusiv Board. (Quelle: [91] S. 24)

Abbildung 3.9: Blackberry WMMS.

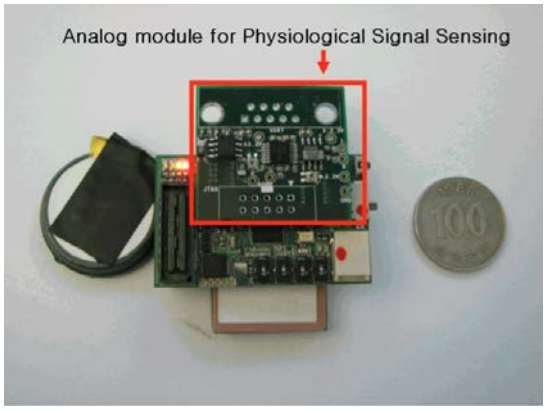
3.4 Biofeedback Ansätze

3.4.1 Gesundheitsüberwachung

Sang-Hyun Kim und andere realisierten ein tragbares Lifeloggingsystem auf Basis eines PDAs. Zusätzlich wird eine Platine zur Messung von physiologischen Signalen verwendet [92]. Es sollte ein tragbares System zur Unterstützung der Gesundheitsüberwachung geschaffen werden. Das System besteht aus einer tragbaren Einheit (siehe Abbildung 3.10) und einem Lifeloggingserver. Die tragbare Einheit erfasst Beschleunigungsdaten, GPS-Position, Stresslevel, EKG, Mean HR und andere Körpersignale. Danach folgt eine Merkmalsextraktion, und die Daten werden mithilfe der energiesparenden ZigBee-Funktechnik an den Lifeloggingserver gesendet. Befugte Ärzte können dann auf die Informationen des Lifeloggingservers zugreifen, und die Daten von Patienten einsehen.

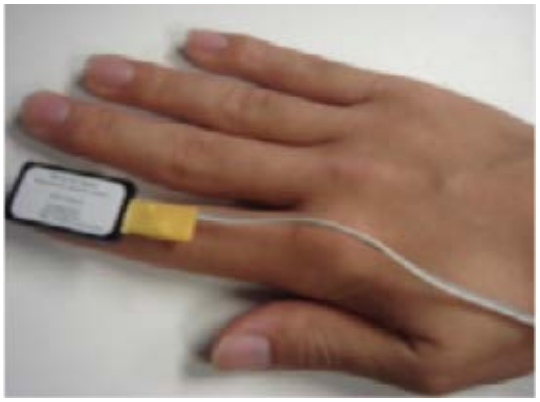


(a) Mobile Einheit mit GPS- und Beschleunigungssensor. (Quelle: [92] S. 843)



(b) Analoges Modul zur Messung von physiologischen Signalen. (Quelle: [92] S. 843)

Abbildung 3.10: Tragbare Gesundheitsüberwachung Lifeloggingeinheit.



(a) Piezo-Sensor am Finger. (Quelle: [93] S. 294)

(b) Zwölf tägliche taktile Aktivitäten. (Quelle: [93] S. 295)

Abbildung 3.11: Prototyp eines Systems, das anhand von Vibrationen Oberflächen erkennt.

3.4.2 Berührungsbasiertes Lifelogging

Yasutoshi Makino und andere entwickelten ein Lifeloggingsystem, das anhand von Berührungen Objekte erkennt [93]. Beim Anfassen eines Gegenstandes entsteht eine charakteristische Vibration, die mit der spezifischen Oberflächenstruktur korreliert. Dazu wandelt ein Piezo-Sensor (siehe Abbildung 3.11a), der auf dem Fingernagel angebracht ist, die bei einer Berührung entstehenden Vibrationen in ein elektrisches Signal um. Aus dem Signal werden Audiomerkmale berechnet, und ein HMM ist in der Lage, zwischen zwölf verschiedenen Berührungen (siehe Abbildung 3.11b) zu unterscheiden.

(a) Interner Zusammenhang von sensor, triggers und actions. (Quelle: [94] S. 4)

(b) Befragung der Sprachqualität. (Quelle: [94] S. 7)

Abbildung 3.12: Struktureller Aufbau und GUI von MyExperience.

3.5 Lifelogging Frameworks

Folgend werden Lifeloggingsysteme vorgestellt, die Frameworkcharakter aufweisen. Beispielsweise ermöglichen sie eine einfache Anpassung oder Erweiterung des Systems.

3.5.1 My Experience

My Experience ist eine von Jon Froehlich und anderen geschaffene BSD-lizenzierte Open Source-Plattform für mobile Datenerfassung auf Windows Mobile 5 Smartphones [94] [95]. Es kombiniert passive und aktive Datenerfassung unterschiedlichster Sensorquellen. Grundlegend gibt es Sensoren, die mithilfe eines Triggers eine sogenannte *action* auslösen (siehe Abbildung 3.12). Sensoren sind beispielsweise Smartphone Interaktionen wie telefonieren, SMS-Nachrichten, die Aufnahme eines Bildes aber auch Veränderungen von Kalender oder Kontaktinformationen gelten als solche. Weiters gibt es Umgebungs-Sensoren, die sämtliche Drahtlosnetzwerke wie GSM, GPS oder WLAN erfassen. Ein Trigger stellt einen logischen Mechanismus bereit, der beispielsweise bedingt durch Abhängigkeiten eine Bildaufnahme an einen Server sendet. Konfiguriert wird MyExperience mithilfe von XML. Die Haupteinträge sind: sensor, trigger, action und question. Im Folgenden werden die einzelnen Einträge näher erklärt:

Sensor : Sensoren aktivieren die Datenerfassung. Dabei werden die Sensoren permanent überwacht, und bei einer Statusänderung ein *trigger* aufgerufen.

$Trigger$: Diese ermöglichen die Deklaration von Abhängigkeiten. Beispielsweise wird ein *trigger* definiert, der nach Beendigung eines Telefongespräches ausgeführt wird, und eine *action* durchführt.

$Action$: Hierbei werden weitere Tätigkeiten abgearbeitet, die von einem *trigger* ausgelöst wurden. Dabei können eine Verzögerung und/oder eine Referenz auf eine Frage angegeben werden.

$Question$: Ermöglicht GUI-basierte Fragen an den Benutzer. Beispielsweise kann, wie in Abbildung 3.12b ersichtlich, die Sprachqualität anhand eines Ratings von *Bad* bis *Excellent* bewertet werden.

Die erfassten Daten werden in einer SQL-Datenbank (SQL Server 2005 Mobile Edition) abgespeichert. In einer vierwöchigen Benutzerstudie mit 16 Personen wurde die Lauffähigkeit der Software getestet. Dabei wurden die Probanden täglich bis zu elf mal nach Daten wie Aufenthaltsort oder sozialer Interaktion gefragt. Der Stromverbrauch der Anwendung wurde ebenfalls evaluiert. Die Betriebsdauer ist abhängig von der Konfiguration und reicht von wenigen Stunden bis zu vier Tagen.

3.5.2 UbiqLog

UbiqLog ist ein von Reza Rawassizadeh geschaffenes Androidbasiertes Framework [96]. Es wurde für keinen speziellen Anwendungszweck entwickelt, und soll einen großen Einsatzbereich ermöglichen. Konfiguriert wird die Datenerfassung durch die GUI am Smartphone. Dabei werden folgende Daten erfasst: laufende Anwendung, Telefon-Log, SMS-Nachrichten, Positionsdaten, Beschleunigungssensor, Bilder, Temperatur, Kompass, Bluetooth und Orientierung. Wie in Abbildung 3.13 ersichtlich, sind bei manchen Sensoren optionale Parameter wie Intervalle spezifizierbar. Die einzelnen Daten werden im JavaScript Object Notation (JSON) Format in dem lokalen Telefonspeicher abgelegt. Über ein Webinterface oder die Smartphone GUI können die Daten visualisiert werden (siehe Abbildung 3.14).

Sensor	Configuration values
Application	Enable = yes, record interval in ms = 10,000
Telephone	Enable = yes, record communication = no
SMS	Enable = yes
Location	Enable = yes, update rate in ms = 10,000
Accelerometer	Enable = no, rate = delay_game, force threshold = 900
Temperature	Enable = no, measurement unit = Celsius
Compass	Enable = no
Bluetooth	Enable = no, scan interval in ms = 60,000
Orientation	Enable = no

Abbildung 3.13: Beispielkonfiguration von UbiqLog. (Quelle: [96] S. 7)

Abbildung 3.14: GUI von UbiqLog. 1: Standort; 2: History der aktiven Anwendungen; 3: Telefongespräche; 4: Abstrahierte Aufenthalts-Visualisierung. (Quelle: [96] S. 15)

3.5.3 Zusammenfassung / Vergleich existierende Ansätze

Tabelle 3.1: Vergleich der einzelnen Systeme.

Jahr	Projektname	Autor	Anwendungsszenario
Entstehung			
1980	LightPainting	Steven Mann	Kunstprojekt
1993	Eyetap	Steven Mann	Lifecasting
1994	WWW	Steven Mann	Lifecasting
1994	Forget me not	M. Lamming und M. Flynn	Büro
1996	JennyCAM	Jennifer Ringley	öffentliches Lifecasting
1998	WearCam	Steven Mann	Lifecasting
2001	MyLifeBits	George Bell	umfassend
Auditiv			
2004	Memory Prosthesis	Sunil Vemuri	Audi-Management
Visuell			
1998	StartleCam	J. Healey und R. Picard	emotionale Erfassung
2004	SenseCam	Lyndsay Williams	visuelle Event Erfassung
2012	Project Glass	Google	AR Anwendungen
Audiovisuell			
2001	Video Lifelog	Kiyohara Aizawa	Video Lifelog
2010	iGlogger	Raymond Chun Hing Lo	mobiles Lifecasting
2011	BlackBerry WMMS	Hui Hsien Wu	Forschungsarbeit
Biofeedback			
2007	Gesundheitsüberwachung	Sang-Hyun Kum	Gesundheit
2010	Berührungs - Lifelogging	Yasutoshi Makino	Oberflächen
Frameworks			
2007	MyExperience	Jon Froehlich	mobile Datenerfassung
2012	UbiqLog	Reza Rawassizadeh	Forschungsarbeit

Erfasste Daten	Bereitstellung	Plattform
Kamera	-	Laptop
Kamera/Audio	-	Laptop/Eyetap
Kamera	öffentlich	Laptop/HMD
Anrufliste, E-Mail, IR	zeitliche Abfolge	PDA, Telefonzentrale
Kamera	öffentlich	Webcam/PC
Kamera	streaming	analoge Kamera
Dokumente, E-Mail, SensCam, Audio, ...	Abfolge Zeit/Ort	PC/Sensecam
Audio, WLAN	Text, Ort, Zeit	PDA
Hautwiderstand, Kamera	-	Laptop
IR, Helligkeit, Kamera, Temperatur, Beschleunigung	-	SenseCam
Audio, Video, WLAN, Bluetooth	-	Glass-Brille
Gehirnaktivität, GPS, Körpersensor, Kamera, Mikrofon	Abfolge Zeit/Ort	PDA
Kamera, Audio, Beschleunigungssensor, Kompass, GPS	öffentlich	iOS/Android/JavaME
Beschleunigungssensor, Audio, Video, GPS, Temperatur, Luftfeuchtigkeit, Helligkeit	WMMS	BlackBerry
Beschleunigung, GPS, Körpersensor	Gesundheitszustand	PDA/Platine
Piezo	-	PC
Location, Beschleunigungssensor, Bilder, Temperatur, Kompass, Bluetooth, Orientierung, GPS, Luftfeuchtigkeit, Helligkeit,.. >50	-	Windows Mobile
Anwendung, Telefon-Log, SMS, Location, Beschleunigungssensor, Bilder, Temperatur, Kompass, Bluetooth, Orientierung	UbiqLog	Android

KAPITEL 4

Kommerzielle Systeme

In diesem Kapitel werden aktuell am Markt verfügbare Lifeloggingsysteme und Smartphone-Lifeloggingsoftware vorgestellt.

(a) Abbildung der Vicon Revue. (Quelle: [97])

(b) Vorderansicht des Autographer. (Quelle: [98])

(c) Memoto in verschiedenen Farb-Varianten. (Quelle: [99])

Abbildung 4.1: Erhältliche Systeme: Vicon Revue, Autographer und Memoto.

4.1 Hardware

4.1.1 Vicon Revue

Vicon lizenzierte die SenseCam-Technologie und vertrieb die sogenannte Vicon Revue (siehe Abbildung 4.1a) [97]. Neben kleinen optischen Veränderungen wurde bei der Vicon Revue gegenüber der SenseCam der interne Speicher, wie auch die Auflösung erhöht

4.1.2 Autographer

Hierbei handelt es sich ebenfalls um lizenzierte SenseCam-Technologie. Gegenüber der SenseCam und Vicon Revue wurden das Gesamtgewicht und die Größe stark reduziert, ein GPS-Empfänger, Bluetooth-Datenaustausch und ein OLED-Display integriert (siehe Abbildung 4.1b) [98]. Durch einen integrierten Clip auf der Rückseite ist es möglich, den *Autographer* wie einen Kugelschreiber an einem Hemd zu befestigen. Das Ziel des Autographers ist es, eine verbraucherfreundliche Version der Vicon Revue zu erschaffen.

4.1.3 Memoto

Die Lifeloggingkamera *Memoto* versucht mithilfe einer besonders einfachen Bedienung zu punkten. Das schwedische Start-up-Unternehmen Memoto wurde über die Internetplattform *Kickstarter.com* dank Crowdfunding finanziert. Das Ziel von Memoto ist eine Kamera, die als fotografisches Gedächtnis dient, und so klein ist, dass sie überall getragen werden kann (siehe Abbildung 4.1c) [99]. Sie verfügt über eine fünf Megapixel-Kamera, GPS und einen Beschleunigungssensor.

(a) Umgehängter Lifepouch mit iPhone und installierter Lifelapse Software. (Quelle: [100])

(b) GUI von Replay My Day. (Quelle: [101])

(c) GUI der Funf Anwendung. (Quelle: [102])

Abbildung 4.2: Strukturell Aufbau und GUI von MyExperience.

4.2 Software

4.2.1 Lifelapse

Hierbei handelt es sich um eine iOS Anwendung (Version 1.1), die in regelmäßigen Intervallen (30 Sekunden) Fotos tätigt (siehe Abbildung 4.2a) [100]. In der *Lifelapse* Ansicht können die getätigten Fotos als Diashow betrachtet und exportiert werden. Eine speziell angefertigte Umhängetasche namens *Lifepouch* wird direkt auf der Herstellerseite vertrieben.

4.2.2 Replay My Day

Das niederländische Start-up *Replay My Day* entwickelt die gleichnamige Lifeloggingsoftware für iOS. Betrachtet wurde die aktuelle Version 1.0.5, die zuletzt am 13. Dezember 2011 überarbeitet wurde. Im Stil eines Tagebuches werden dabei Bilder, Videos, Tweets und Facebook-Meldungen mit zugehöriger Position aufgezeichnet (siehe Abbildung 4.2b) [101]. Am Tagesende spielt man die gesammelten Daten als Video ab.

4.2.3 Funf

Funf ist eine Alphaversion einer Androidanwendung zur Aufzeichnung von Smartphone-Daten (siehe Abbildung 4.2c) [102]. Entwickelt wurde sie am MIT Media Lab zur Erfassung und Visualisierung gängiger Daten. Dabei ist die Anwendung so konfigurierbar, dass sie in regelmäßigen Intervallen Daten erfasst. Über eine separate Desktopanwendung *Funf Analyze* erfolgt eine Visualisierung der aufgezeichneten Daten.

KAPITEL 5

Android Lifelogging (ALL)

Dieses Kapitel befasst sich mit der Beschreibung und Realisierung des praktischen Teils der Arbeit. Zuerst werden die Anforderungen an das Lifeloggingsystem beschrieben und das Anwendungsfallmodell erstellt. Weiters wird auf die Architektur und das Datenbank-Schema eingegangen. Danach werden die Struktur und die Spezifikation der XML-Beschreibungssprache LLDL näher erklärt.

5.1 Anforderungsanalyse

Aus den Projektzielen und den Erkenntnissen vorhandener Lifeloggingsysteme ergeben sich folgende Anforderungen:

- Es soll eine stabile, individuell konfigurierbare und leicht erweiterbare Software Architektur für das Android Betriebssystem geschaffen werden. ALL ist ein Prototyp, der mit nur minimalen Auswirkungen auf die Benutzbarkeit des Smartphones permanent im Hintergrund laufen soll.

- Eine auf XML basierende Beschreibungssprache LLDL soll den internen Programmablauf festlegen. Hierbei sollen der zeitliche Ablauf, Sensoren, Abhängigkeiten und eine Filterung der Daten ermöglicht werden.

- Durch die LLDL soll realisiert werden, dass sämtliche Sensoren der Android-API verwendet werden können. Dabei werden durch regelmäßige Intervalle oder externe Events ausgelöste Daten erfasst und abgespeichert. Abhängig von den gemessenen Daten (Abhängigkeiten) sollen weitere Sensoren abgerufen werden.

- Es sollen Abhängigkeiten definiert werden können; beispielsweise soll nur dann ein Foto gemacht werden, wenn die Umgebungshelligkeit hoch genug ist. Dadurch werden zusätzliche Energie und Speicherplatz gespart.

- Der Benutzer soll die Möglichkeit haben, Profile zu erstellen, die je nach Anwendungsgebiet eine spezifische Aufzeichnung der Daten ermöglichen. Dazu soll es automatische und manuelle Profile geben. Ein automatisches Profil aktiviert sich beispielsweise an einem spezifischen Ort. Im Gegensatz dazu muss ein manuelles Profil explizit gestartet werden.

- Die einzelnen Daten sollen auf dem Smartphone in einer SQLite-Datenbank abgespeichert werden. Sämtlicher Zugriff auf die Daten soll verschlüsselt, beziehungsweise mithilfe eines Passworts abgesichert sein. Ein Export, Import oder Austausch wird immer verschlüsselt durchgeführt.

- Beim Abspeichern der einzelnen Daten soll die Definition eines Filters möglich sein. Durch die Verwendung von Filtern wie einem Median- oder Mittelwertfilter soll das Speichern von redundanten Daten verhindert werden.

- Die Konfiguration soll das manuelle Hinzufügen von textuellen Anmerkungen ermöglichen. Dabei sind die ID und der Typ wie Text, oder ein Zahlenbereich der Anmerkung definierbar.

- Unterschiedliche Visualisierungen und Auswertungen der Daten sollen über ein Webinterface zeitlichen und örtlichen Zusammenhang veranschaulichen. Die Übertragung wird mithilfe von SSL verschlüsselt und durch ein Passwort geschützt.

5.1.1 Charakterisierung

Der praktische Teil der vorliegenden Arbeit ist anhand der Kriterien für Lifeloggingsysteme (siehe Kapitel 2.1, Seite 12) folgendermaßen einzuordnen:

Verwendung: ALL dient zur Erstellung einer privaten Datensammlung mit dem Ziel, neue Informationen und Anwendungen mittels Lifelogging zu entwickeln.

Bandbreite: Sämtliche Sensoren sind optional zu verwenden. Damit lassen sich ein simpler GPS-Logger wie auch eine multimediale Erfassung realisieren.

Perspektive: ALL ist primär für eine mobile Erfassung konzipiert. Beispielsweise kann eine visuelle Erfassung durch die Unterbringung in einer um den Hals getragenen Tasche erreicht werden. Eine nicht visuelle Datenerfassung ist durch das Einstecken des Smartphones in die Hosentasche möglich.

Abtastung: Durch die Steuerung von Profilen ist eine individuelle Abtastung möglich. Die maximale Abtastfrequenz ist mit zehn Sekunden festgelegt. Daten, die für nicht wichtig erachtet werden, können mittels einer Abhängigkeit aussortiert werden.

Steuerung: Sämtliche Datenerfassung wird anhand von Ort und Zeit automatisch gesteuert. Durch manuelle Profile kann jedoch jederzeit eine spezifische Erfassung realisiert werden.

Organisation: ALL speichert sämtliche Daten zentral auf dem Smartphone. Dabei kann angegeben werden ob der interne Speicher, oder die SD-Karte benutzt werden sollen.

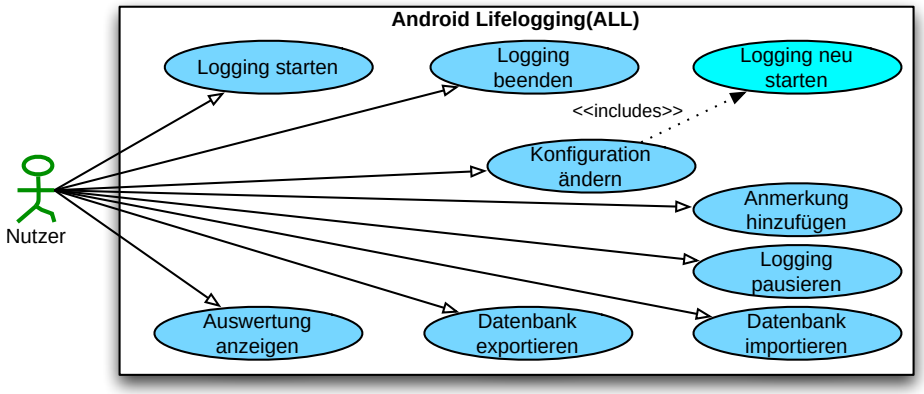


Abbildung 5.1: ALL Anwendungsfalldiagramm zur Veranschaulichung des Systems.

5.2 Anwendungsfälle

Anhand der Anforderungsanalyse erfolgt die Modellierung der Anwendungsfälle (siehe Abbildung 5.1) von ALL.

Logging starten/beenden: Beendet beziehungsweise startet den Hintergrunddienst von ALL. Zu beachten ist, dass beim Beenden sämtliche Abonnements sauber beendet werden.

Konfiguration ändern: Dieser Anwendungsfall verändert den internen Ablauf. Zusätzlich beinhaltet *Konfiguration ändern* den Anwendungsfall *Logging neu starten*.

Anmerkung hinzufügen: Mithilfe des Anwendungsfalles kann je nach Konfiguration eine spezifische Notiz hinzugefügt werden.

Logging pausieren: Hierbei wird die Aufzeichnung pausiert und kann durch *Logging starten* wieder fortgesetzt werden. Wie auch schon bei *Logging beenden* ist zu beachten, dass sämtliche Abonnements sauber beendet werden.

Datenbank importieren: Durch diesen Anwendungsfall wird das Importieren sämtlicher Lifeloggingdaten unterstützt. Nebenläufig soll der Hintergrunddienst von ALL nicht gestört werden.

Datenbank exportieren: Ein Export der Geodaten soll in gängigen Dateiformaten erfolgen. Ein kompletter Export sämtlicher Daten soll ebenfalls möglich sein. Wie auch schon beim Importieren soll ALL davon nicht beeinflusst werden.

Auswertung anzeigen: Im Anwendungsfall sollen verschiedene Visualisierungsarten, wie eine Karte oder ein Zeitraffer unterstützt werden.

5.3 Architektur und grundlegende Konzepte

Aus den zuvor definierten Anwendungsfällen leiten sich die prinzipielle Architektur und ihre Konzepte (siehe Abbildung 5.2) ab. ALL soll abhängig vom Einsatzort und der Zeit verschiedene Erfassungsarten unterstützen, beispielsweise eine energiesparende Aufzeichnung im Alltag, oder eine präzise Erfassung bei sportlicher Aktivität. Grundsätzlich erfasst ALL durch *Trigger* gesteuert *Datenquellen* und speichert diese in eine *Datenbank*. Zur Steuerung der Datenaufnahme werden *Events* definiert. *Events* spezifizieren eine spezifische Datenerfassung und werden *Profilen* zugeteilt. *Profile* dienen der orts- und zeitspezifischen Aktivierung von *Events* und werden vom *Profil Manager* aktiviert. Der *Profil Manager* entscheidet mithilfe der LLDL Deklaration, wann welche *Profile* aktiviert werden sollen. Weiters sollen jederzeit manuelle *Annotationen* hinzugefügt werden können.

Datenbank: Als Datenbank soll das relationale Datenbanksystem SQLite verwendet werden. Sämtliche Anwendungseinstellungen, Sensordaten und Pfade zu Mediendateien sollen damit abspeichern werden.

Datenquelle: Eine Source ermittelt Daten meist mit einem Sensor. Dies können Positionsdaten, Multimediadaten wie Bilder oder kontextuelle Daten sein. Beispielsweise soll die Source *camera* ein Bild machen oder *SmsInAndOut* Daten von ein- und ausgehenden SMS-Nachrichten liefern. Sobald die Daten ermittelt wurden, entscheiden Abhängigkeiten über die weitere Datenerfassung und das Abspeichern der Daten.

Trigger: Bei einem Trigger handelt es sich um einen *synchronen* oder *asynchronen* Auslöser, der das Event startet. Bei einem *synchronen* Trigger handelt es sich um ein periodisches Intervall, das vom System ausgelöst wird. Alle anderen Trigger sind *asynchron*. Diese können vom Benutzer, einem aktiven Anruf, einer SMS-Nachricht oder einer anderen Person ausgelöst werden. Nach dem Start eines Events werden verschiedene Daten ermittelt und optional weiterverarbeitet.

Event: Jedes Event spezifiziert eine Menge von Sources und besitzt einen Trigger, der die Erfassung initiiert. Mehrere Events sollen gleichzeitig und unabhängig unterschiedliche oder gleiche Daten erfassen können. Die Erfassung einzelner Daten erfolgt mithilfe einer Source. Der Start eines Events soll durch verschiedene Arten von Triggern ausgelöst werden. Beispielsweise durch eine empfangene SMS-Nachricht oder ein periodisches Intervall.

Profil: Ein Profil weist einem Einsatzort eine passende Datenerfassung zu. Dazu referenziert ein Profil auf ein oder mehrere Events. Automatische Profile ermöglichen eine orts- und zeitspezifische Erfassung. Durch das Überprüfen von Bedingungen entscheidet dann ein *Profil Manager*, wann und wo welches Profil aktiv ist. Ein Profil benötigt mindestens ein *Event*, damit die Datenerfassung erfolgen kann. Eine Deklaration von manuellen Profilen wird ebenfalls unterstützt. Ein solches wird nur durch ein explizites Starten ausgeführt und bietet keinen automatischen Profilwechsel an.

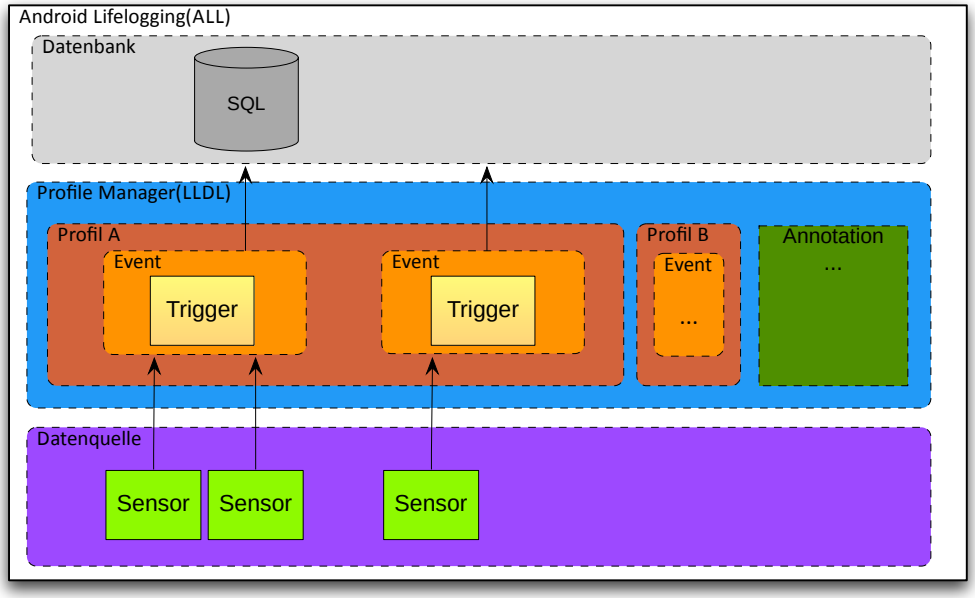


Abbildung 5.2: Architektur und grundlegende Konzepte von ALL.

Profil Manager: Abhängig von der LLDL-Deklaration werden Profile vom Profil Manager aktiviert und die Datenerfassung gestartet. Wenn der Einsatzort oder die gültige Zeit eines Profiles verlassen wurde, wird das nächste gültige Profil aktiviert.

Annotation: Dies sind textuelle Anmerkungen, die manuell zusätzliche Informationen in das Lifeloggingsystem einbringen. Dabei können diese aus mehreren Eingabefeldern bestehen und jederzeit hinzugefügt werden.

5.3.1 Klassendiagramm

Aus der Architektur werden die Klassendiagramme von ALL abgebildet. ALL unterstützt die Konfigurationsbeschreibungssprache LLDL und wurde anhand der Anforderungen realisiert. Die einzelnen Pakete kapseln logische Komponenten (siehe Abbildung A.1) ab, und können in Präsentation (View), Anwendungslogik (Controller) und Datenbank (Model) eingeteilt werden. Diese Model View Controller (MVC) Unterteilung, ein gängiges Entwurfsmuster in der Softwareentwicklung, trennt die eigentlichen Daten von der Anwendungslogik und Präsentationslogik. Die Vorteile dabei sind, dass spätere Änderungen möglichst präzise, einfach und übersichtlich durchgeführt werden. Die Wiederverwendbarkeit von Codefragmenten und die Aufteilung auf mehrere Entwickler werden dadurch ebenfalls erleichtert. Die Visualisierung der über 200 Klassen würde hier den Rahmen sprengen, deshalb sind sämtliche für das System notwendigen Klassendiagramme im Anhang (siehe Anhang A.1, Seite 117) ersichtlich.

5.4 Datenbank

Beim Schreiben in eine SQLite-Datenbank unter Android gibt es die Einschränkung, dass nur jeweils eine Activity zu einem Zeitpunkt Schreibrechte erhält. Deshalb wurden die allgemeinen Einstellungen der Tabelle *settings* in eine separate Tabelle verschoben und mithilfe eines eigenen *Contentproviders* mit einer *Contentprovider-Activity* synchronisiert. Damit können beliebig viele Activities gleichzeitig die Einstellungen lesen und schreiben, wie auch Änderungen über einen *ContentObserver* abonnieren. Die Android-API fordert keine weiteren Beschränkungen des SQLite Standards. Sämtliche anderen Tabellen werden über einen *BroadcastReceiver* (*ScheduleReceiver*) geschrieben und über eine beliebige andere Activity ausgelesen.

5.4.1 Schema

Folgend werden die einzelnen Tabellen der Datenbank, wie auch in Abbildung 5.3 dargestellt, kurz beschrieben.

settings: Diese Tabelle speichert sämtliche Einstellungen von ALL als Schlüssel (*key*) / Wert (*value*)-Paare ab.

calendar: Ein Kalendereintrag beinhaltet eine eindeutige ID und einen Initialisierungs-Unix-Zeitstempel *init*. Die Spalte *name* betitelt den Kalendereintrag, wobei *start*, *end* und *timezone* die Zeitspanne mit zugehöriger Zeitzone spezifiziert. In *notes* wird eine optionale Notiz und in *location* der zugehörige Ort eingetragen.

event: Dies ist das zentrale Element jedes Sensoreintrages. Diese Event-Tabelle ist mit sämtlichen Sensoreinträgen mittels einer 1:n Verbindung (*id_event*) verbunden. Jedes Event hat einen zugehörigen Namen aus der Tabelle *event_name*, sowie Profil Namen aus der Tabelle *event_profile* und Sicherheitseinstufung (*permission*) aus der Tabelle *event_profile*. Die Berechtigung kann beispielsweise genutzt werden, um sensible Daten zu schützen.

media: Hierbei handelt es sich um eine Hilfstabelle, um die Daten auszulagern. Diese Tabelle organisiert verschiedene Medienevents (*type*) wobei der Dateipfad (*path*) einzelne Mediendateien und die Dateigröße (*size*) erfasst werden.

acceleration In dieser Tabelle wird die Standardabweichung (*std*) der relativen Beschleunigung einer Beschleunigungsmessung abgespeichert.

annotation: Eine Anmerkung, die mithilfe eines Schlüssel (*key*)-Parameters in verschiedene Arten eingeteilt, und anhand eines Wert (*value*)-Parameters einen zugehörigen Text spezifiziert.

barometer: Diese Tabelle speichert den gemessenen Luftdruck (*pascal*) in der Einheit $pascal$ einer Luftdruckmessung.

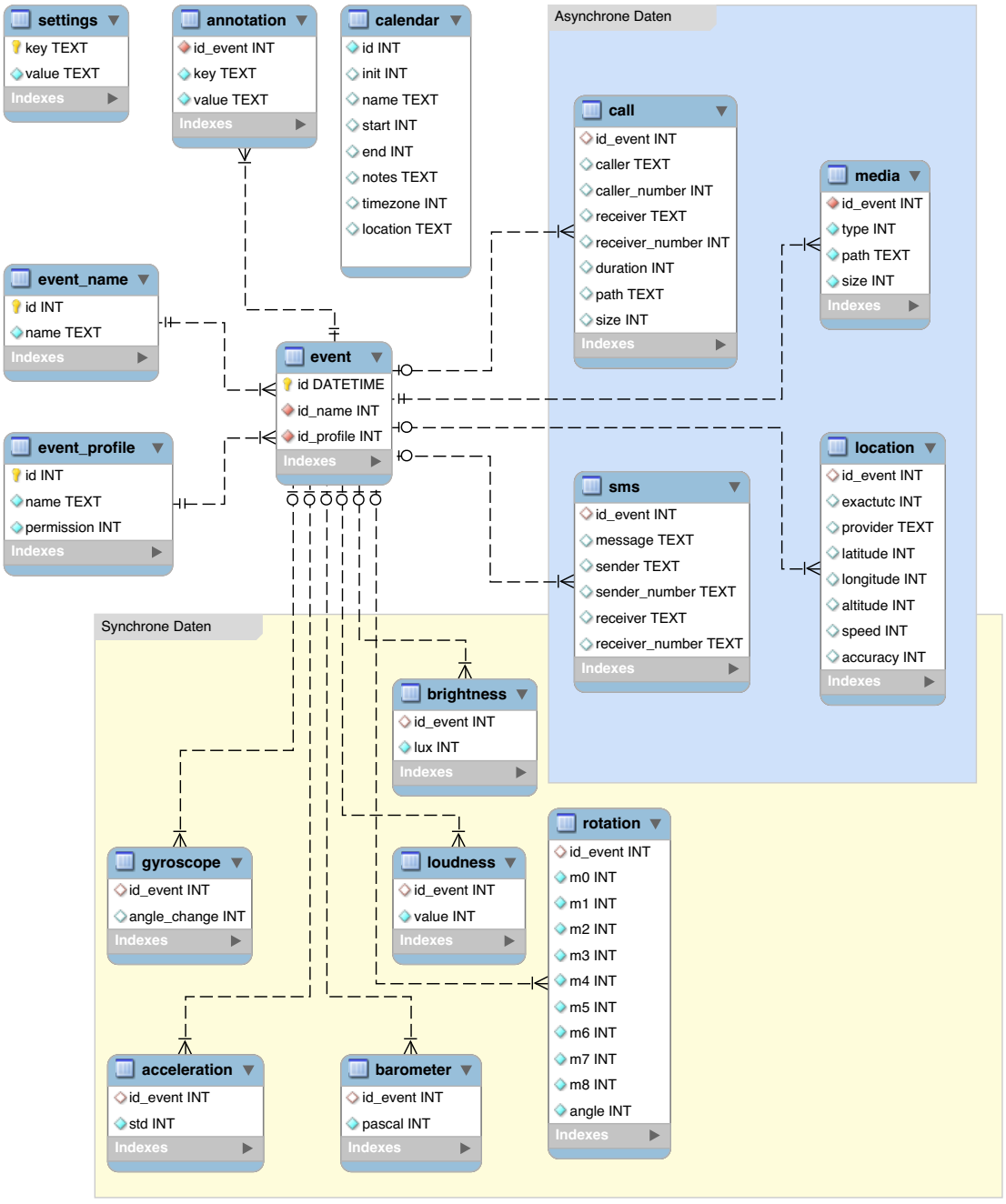


Abbildung 5.3: Datenbank Schema von Android Lifelogging.

brightness: In dieser Tabelle wird die Helligkeit (*lux*) einer Helligkeitsmessung in der Einheit Lux gespeichert.

call: Im Eintrag *caller* steht der Name derjenigen Person, die den Anruf getätigt hat. Der *receiver* Parameter beschreibt denjenigen, der den Anruf angenommen hat, wobei die Felder mit der Endung *_number* jeweils die zugehörige Telefonnummer angeben. Die Dauer des Gespräches wird mit *duration* angegeben. Falls eine Aufnahme stattgefunden hat, wird mithilfe des *path* der Dateipfad und mit *size* die Dateigröße angegeben.

gyroscope: Beinhaltet den *angle_change* Parameter, welcher aus der durchschnittlichen Rotationsänderung des Smartphones ermittelt wird.

loudness: Speichert einen gerätespezifischen Lautstärkewert *value*. Der Wert *rotation* gibt eine OpenGL-kompatible Rotationsmatrix an, welche die Lage des Handys repräsentiert. Der Parameter *angle* gibt den 2D-Winkel bezüglich des magnetischen Nordpoles an.

location: Mithilfe der *location*-Tabelle werden unterschiedliche Lokalisierungen, die mit den Satelliten-, WLAN- oder Sendemastensystemen erfolgten, abgespeichert. Die exakte Erfassungszeit wird in *exactutc* und der Lokalisierungsdienst mithilfe eines *provider* angegeben. Die eigentliche Position spezifizieren *latitude* und *longitude*, wobei falls verfügbar, auch die Höhe *altitude*, Geschwindigkeit *(*m/s*)* und Genauigkeit *accuracy* angegeben werden.

sms: Beinhaltet eine SMS-Nachricht, wobei *message* den Nachrichtentext, *sender* den Absender und *receiver* den Empfänger angeben. Die Felder mit der Endung *_number* verweisen jeweils auf die zugehörige Telefonnummer.

5.5 Webservice

Über den Webservice können sämtliche Daten abgerufen, und Android Lifelogging mithilfe der LLDL konfiguriert werden. Der Webservice wurde als länger laufende und unabhängige Komponente mithilfe eines Android Services realisiert. Ein Android Service kann im Fall von Speicherknappheit jedoch von der Laufzeitumgebung beendet werden. Der Webservice wurde deswegen mithilfe einer Benachrichtigung und eines zusätzlichen *STICKY* Attributs so realisiert, dass bei der Beendigung des Services ein umgehender Neustart des Services erfolgt. Dank Multithreading werden sämtliche HTTPS-Anfragen über SSL-Sockets des Ports 8080 bitweise abgewickelt. Die SSL-Verschlüsselung dient dabei als Sicherheitsvorkehrung, damit sämtliche Kommunikation nicht abgehört werden kann. Die Erstellung des Zertifikates ist im Anhang (siehe Kapitel A.2, Seite 126) ersichtlich. Über einen Keystore wird jede Verbindung zuerst asynchron und danach synchron verschlüsselt. Die Datenverarbeitung und Auswertung erfolgen direkt in der Android Lifeloggingapplikation. Als Zugriffskontrolle wird das HTTP-Authentifizierungsverfahren *basic access authentication* verwendet. Dabei werden vom Browser aus einmalig Benutzername und Passwort abgefragt, und nach erfolgreicher Authentifizierung

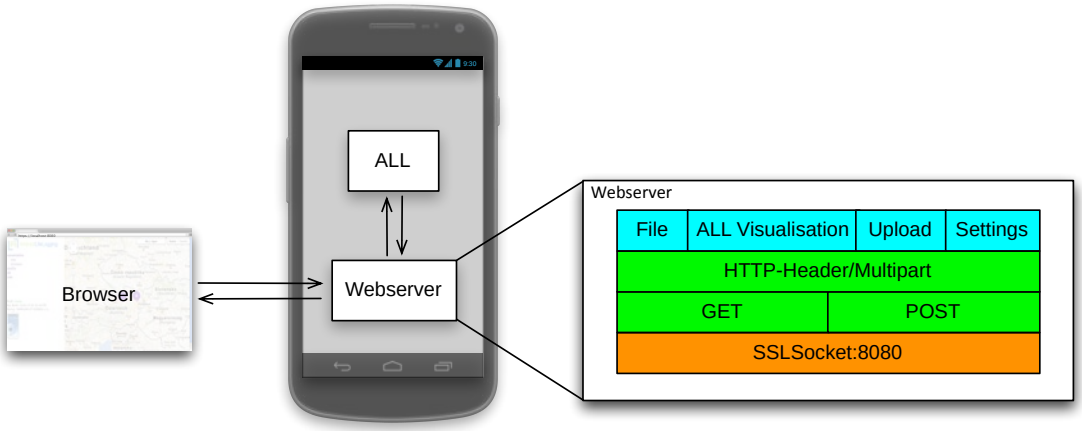


Abbildung 5.4: Allgemeiner Überblick über die ALL Architektur, das Zusammenspiel der Komponenten und den internen Aufbau des Webservers.

jeder weiteren Anfrage mitgegeben. Die HTTP-Request-Methoden GET, POST inklusive HEADER und MIME-Typen wurden implementiert (siehe Abbildung 5.4a) und für die Übertragung verwendet. Zur Abfrage von Dateien des Android Dateisystems wurde ein einfacher Fileserver realisiert. Für den Upload von Dateien wird multipart/form-data unterstützt. Dies wird für Zusatzinformationen beim Dateitransfer benötigt.

Das Zusammenspiel zwischen dem ALL-Hintergrunddienst, dem Webserver und einem Browser, der das Webinterface aufruft, ist in Abbildung 5.4b ersichtlich.

5.6 Lifelogging Description Language (LLDL)

Die LLDL ist eine auf XML basierende Sprache für die Konfiguration der Lifeloggingan-wendung. Sie beschreibt, welche Daten wann und wo erfasst werden. Die Sprache definiert auch Abhängigkeiten der Aktualisierung bedingt durch die erfassten Daten. Damit soll eine energieeffiziente und ressourcenschonende Datenerfassung erzielt werden. Durch Deklaration von Anmerkungen können einfache Notizen bis hin zu komplexen Fragebögen realisiert werden. Anschließend werden einzelne Begriffe oder Buchstaben der XML-Beschreibungssprache groß geschrieben, um eine bessere Lesbarkeit von zusammenhängenden Wörtern zu erzielen.

Die Konfiguration der Beschreibungssprache erfolgt über das Webinterface oder das GUI der ALL-Applikation. Die XML-Deklaration beschreibt das spezifische Verhalten. Dabei können abhängig vom Ort zugehörige Profile automatisch aktiviert oder deaktiviert werden. Die grundlegende Spezifikation der XML besteht (siehe Abbildung 5.5) aus vier Einträgen: *locations*, *profiles*, *events* und *annotations*. Die Reihenfolge dieser Elemente ist beliebig, wird aber zur Veranschaulichung immer so gewählt. Zur besseren Verständlichkeit wird dort, wo es sinnvoll erscheint, zur XML-Spezifikation eine abstrahierte Visualisierung beigefügt.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
        <locations>     ...
        </locations>
        <profiles>      ...
        </profiles>
        <events>        ...
        </events>
        <annotations>   ...
        </annotations>
</config>
```

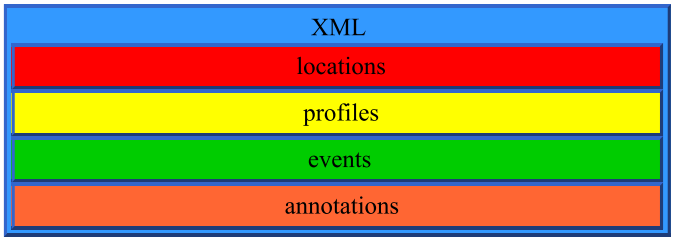


Abbildung 5.5: Grundlegende Struktur der XML-Konfiguration und abstrahierte Darstellung.

Kinderknoten des Wurzelknoten *config*:

locations	In diesem Element können geografische Bereiche deklariert werden, die für Profilabhängigkeiten verwendbar sind.
profiles	Hier wird deklariert, unter welchen zeitlichen und örtlichen Bedingungen die Ausführung eines spezifischen Profils und der zugehörige Event erfolgt.
events	Unter diesem Element wird durch die Angabe eines *triggers* ein asynchrones oder synchrones Event spezifiziert. Dies ist ein synchrones Event, falls es sich um ein zeitliches Intervall handelt.
annotations	Mithilfe dieses Elements ist es möglich, eine Notiz oder einen Fragenkatalog zu spezifizieren.

Die eben aufgelisteten Wurzelknoten beinhalten jeweils einzelne Kinderknoten, deren Namen der Einzahl des englischen Begriffes entsprechen.

5.6.1 Locations

Im Element *locations* werden die einzelnen *location* Einträge deklariert. Ein *location* Element beinhaltet die geografischen Koordinaten und den zugehörigen Radius.

location

Die *location* Einträge dienen als boolesche Variablen, die zum automatischen Wechsel zwischen den Profilen notwendig sind. Damit wird entschieden, welches Profil aktuell aktiv ist.

In Abbildung 5.6 ist ein XML-Beispiel angegeben, das zwei Bereiche deklariert.

Verfügbares Attribut:

$-$ *name*	definiert einen eindeutigen Schlüssel einer *location*
$-$ *latitude*	gibt die geografische Breite an $\pm 90^{\circ}$
$-$ *longitude*	spezifiziert die geografische Länge $\pm 180^{\circ}$
$-$ *range*	definiert einen Radius um den spezifizierten Punkt

Mehrere Vorbedingungen sind zu erfüllen, damit der automatische Wechsel funktioniert. Primär muss gewährleistet sein, dass ein regelmäßiges Update der Position stattfindet. Somit ist eine regelmäßige Lokalisierung in jedem automatischen Profil notwendig. Je nachdem, ob es sich um eine Satelliten-, WLAN- oder Sendemastenlokalisierung handelt, muss ein entsprechend genauer Radius angegeben werden. Zur Aktualisierung werden jeweils die aktuelle letzte Position, beziehungsweise der Median oder Mittelwert der einzelnen Positionsgeber verwendet. Falls keine neue Lokalisierung möglich ist, kann auch kein Wechsel erfolgen. Dabei ist aber zu beachten, dass Messfehler zu einem falschen und vermehrten Wechsel der Profile führen. Ein zu kleiner Radius bei der Deklaration einer *location* kann ebenfalls zu einem fehlerhaften Profilwechsel führen. Deshalb sollte ein dementsprechender Radius gewählt werden, der groß genug ist, um Messfehler zu tolerieren und klein genug, um den Einsatzort zu repräsentieren. Falls ein Kreis den gewünschten Ort nicht optimal abdeckt, kann auch ein Verbund von mehreren Orten mithilfe eines *and*- oder *or*-Knotens erfolgen.

5.6.2 Profiles

Im *profiles*-Abschnitt (siehe Abbildung 5.7) werden die einzelnen *profile*-Elemente deklariert. Es muss mindestens ein Profil angegeben werden.

profile

Verfügbares Attribut:

$-$ *name*	Eindeutiger Name, der in Kleinbuchstaben eindeutig sein muss. Dieser repräsentiert ein Profil.
$-$ *permission*	Spezifiziert die Sicherheitseinstufung eines Profiles, beispielsweise um sensible Daten leichter filtern zu können. Falls nicht angegeben, wird die Zahl 1 verwendet.

Das Element *profile* kann eine oder mehrere *condition*-Bedingungen beinhalten (siehe Abbildung 5.8) und verweist mindestens auf ein *event*-Element. Falls keine Bedingung angegeben ist, handelt es sich um ein manuelles Profil (siehe Abbildung 5.9). Ein manuelles Profil enthält kein *condition*-Element, da auch kein automatischer Wechsel erwünscht ist. Dabei ist auf die Reihenfolge zu achten. Zuerst müssen die automatischen Profile deklariert werden. Erst danach ist die

```
<locations>
        <location name="hauptbibliothek" latitude="48.199938" longitude= "
            16.367697" range="125"/>
        <location name="karlskirche" latitude="48.19892" longitude= "
            16.372101" range="125"/>
</locations>
```

Abbildung 5.6: Zwei *locations*-Elemente mit den IDs *hauptbibliothek* und *karlskirche*, sowie zugehörigem Längengrad, Breitengrad und Radius.

Deklaration von manuellen Profilen möglich. Eine Definition eines automatischen Profiles nach einem manuellen Profil ist nicht möglich und führt zu einer Fehlermeldung.

condition

Eine *condition* ist eine Bedingung und kann aus mehreren Teilbedingungen bestehen. Die gesamte Bedingung muss logisch auswertbar sein, sodass die Bedingung entweder wahr oder

```
<profiles>
        <profile name="default" permission="1">          ...
        </profile>
        <profile name="periodicPicture" permission="1"> ...
        </profile>
        <profile name="loudness" permission="1">         ...
        </profile>
        <profile name="sport" permission="2">            ...
        </profile>
</profiles>
```

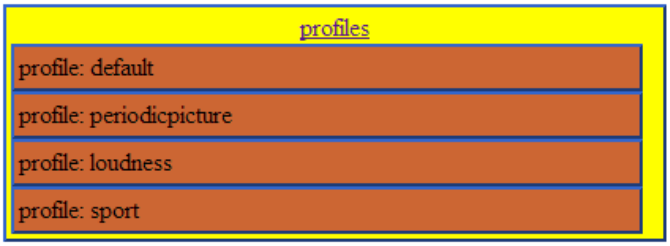


Abbildung 5.7: XML-Deklaration einzelner Profile.

falsch ist. Eine Teilbedingung kann ein Konstrukt bestehend aus *or*, *and*, *not*, *true*, *false* oder *when* sein.

when

Verfügbares Attribut:

- *Location* Der Attributwert ist eine Referenz auf ein *location*-Element aus *locations*. Falls sich das Smartphone innerhalb des deklarierten Bereiches befindet, wird das *when*-Element als wahr gewertet.
- *Time* Eine zeitliche Beschränkung der Form: *HH:MM,HH:MM*, wobei die Zeit vor dem Beistrich, dem Intervallanfang, und die Zeit nach dem Beistrich dem Intervallende entspricht. Beispielsweise gibt *09:00,12:00* einen gültigen Zeitbereich zwischen neun und zwölf Uhr vormittags an.
- *Not* Durch das vorangestellte *Not* wird die *Condition* negiert.

Die Attribute eines *when*-Elementes sind entweder *Location* oder *NotLocation*. Dies hat die gleichen Auswirkungen wie eine zusätzliche Kapselung des *when*-Elements in ein *not*-Element. Falls wie im oberen Beispiel zwei *when*-Teilbedingungen nicht logisch mit *or* oder *and* verknüpft sind, werden sie automatisch mithilfe einer *and*-Verknüpfung verbunden. Es ist darauf zu achten, dass immer nur ein Profil und somit nur ein *condition*-Konstrukt zu einem Zeitpunkt erfüllt ist. Anderenfalls können die Abfolge und somit der gewünschte Erfassungsbereich nicht ermittelt werden. Durch logische Verknüpfungen von *and* und *or* können beispielsweise mehrere *Location* Kreise zur besseren Abdeckung eines Gebietes zusammengefasst werden. Durch *Not* wird aus einer *Location* eine negierte NotLocation Bedingung.

```
<profile name="default" permission="1">
        <condition>
                <and>
                        <when NotLocation="hauptbibliothek"/>
                        <when NotLocation="karlskirche"/>
                </and>
        </condition>
        <event ref="defaultTracking"/>
        <event ref="pictureTaken"/>
</profile>
```

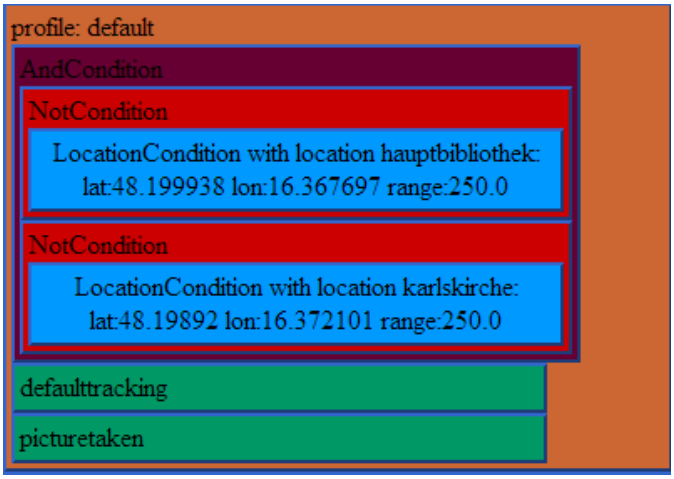


Abbildung 5.8: Deklaration eines automatischen Profils, das aktiv wird, wenn sich das Smartphone außerhalb der Orte *hauptbibliothek* und *karlskirche* befindet. Bei Aktivierung werden die Events *defaultTracking* und *pictureTaken* ausgeführt, um Daten zu sammeln.

```
<profile name="sport" permission="2">
        <event ref="sportTracking"/>
</profile>
```

Abbildung 5.9: Deklaration eines manuellen Profils, das bei manueller Aktivierung den Event *sportTracking* ausführt, beispielsweise um sportliche Aktivität permanent zu erfassen.

event

Verfügbares Attribut:

– ref Verweist auf das Event, das später definiert wird. Der Vorteil einer Referenz besteht darin, dass ein Event nur einmal definiert werden muss, aber mehrmals verwendet werden kann.

5.6.3 Events

Im Abschnitt *events* (siehe Abbildung 5.10) werden die einzelnen Eventeinträge deklariert.

```
<events>
        <event trigger="makePicture" name="pictureTaken"> ...
        </event>
        <event trigger="1min" name="loudnessTracking"> ...
        </event>
        <event trigger="20sec" name="sportTracking"> ...
        </event>
</events>
```

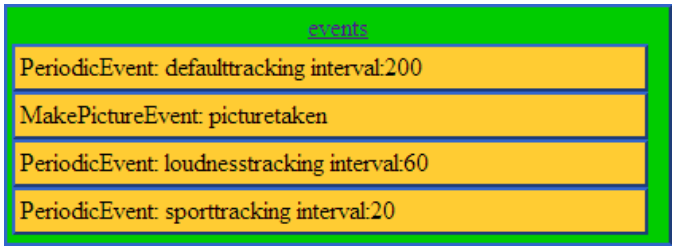


Abbildung 5.10: Allgemeine Struktur des Elements *events*.

event

Verfügbares Attribut:

– $trigger$ Angabe eines synchronen oder asynchronen Auslösers.
– $name$ Eindeutiger Name, der das Event repräsentiert.

Jedes Event hat einen Auslöser *trigger*, der eine Aktualisierung startet. Dabei gibt es synchrone und asynchrone Auslöser. Ein Synchroner *trigger* löst in periodischen Intervallen ein Update aus. Asynchrone Auslöser wie ein Telefonanruf sind Ereignisse, die schwer vorhersehbar sind.

Im synchronen Fall (siehe Abbildung 5.11) besteht der Attributwert von *trigger* immer aus einer ganzen Zahl und der Endung „sec“, „min“ oder „h“. Dieser gibt die Pause zwischen den Updates an.

Asynchrone Auslöser (siehe Abbildung 5.12) sind: *call*, *sms*, *makeVideo*, *makePicture* und *mediaButton*. Zu diesen *trigger* Attributen gibt es zugehörige *src* Typen: *callOutAndIn*, *SmsInAndOut*, *SmsIn*, *SmOut*, *Filesystem* und *Button*.

```
<event trigger="200sec" name="defaultTracking">
        <src type="location" maxSearchTime="20" method="net" precision="
            median[5,11]">
                <dependency locationAccuracy="[0,200]">
                        <writeDb data="location"/>
                </dependency>
        </src>
</event>
```

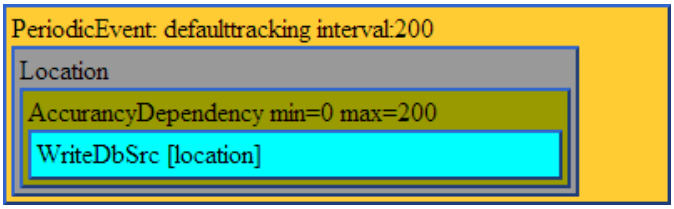


Abbildung 5.11: XML-Deklaration eines synchronen Events, das regelmäßig die Position mittels WLAN und Funkmasten ermittelt. Falls die Messgenauigkeit unter 200 Meter liegt, wird die Messung mithilfe von *writeDb* in die Datenbank geschrieben.

src

Ein *src* Element definiert (siehe Abbildung 5.13) eine Sensordatenquelle. Diese kann beliebig verschachtelt und gereiht werden.

Verfügbares Attribut:

– *type* gibt an, welche Daten gesammelt werden sollen

Bei Verwendung des Attributes *Location* sind die maximale Erfassungszeit *maxSearchTime*, die Erfassungsmethode *method* und ein optimaler Filterparameter *precision* anzugeben.

Für die Datenerfassung *Filesystem* und *Call[Out|In|OutAndIn]* wird ein *path*-Attribut benötigt. Beim *Filesystem* gibt das Attribut den zu überwachenden Pfad an. Hingegen wird bei *Call[Out|In|OutAndIn]* der Pfad angegeben, an jenem die Aufzeichnung des Telefongespräches erfolgt. Bei der automatischen Bilderfassung sind die Attribute *path* und *resolution* anzugeben. Mittels *path* wird der Zielpfad des Dateisystems und mit *resolution* [small|medium|large] die Bildgröße angegeben. Für die Erfassung von *acceleration*, *audio*, *gyroscope*, *loudness* und *rotation* ist ein *time* Attribut anzugeben.

dependency

Ein *dependency*-Element (siehe Abbildung 5.14) repräsentiert eine Abhängigkeit und wird mithilfe eines Sensorattributs und Intervalls deklariert. Kinderelemente werden nur ausgeführt, wenn die Bedingung erfüllt ist.
Verfügbares Attribut:

```
<event trigger="makePicture" name="pictureTaken">
        <src type="Filesystem" path="/sdcard/DCIM/Camera/">
                <writeDb data="filepath"/>
                <src type="location" maxSearchTime="60" method="gpsandnet"
                    precision="median[3,3]">
                        <dependency locationAccuracy="[0,200]">
                                <writeDb data="location"/>
                        </dependency>
                </src>
        </src>
</event>
```

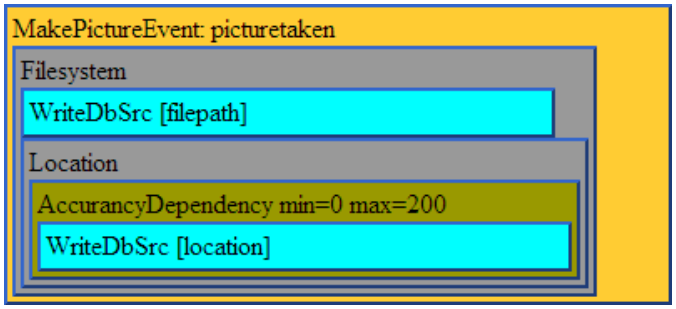


Abbildung 5.12: XML-Deklaration eines asynchronen Events, das die Erstellung neuer Fotos im Dateisystem überwacht. Sobald ein neues Foto im angegebenen *path* erstellt wurde, werden der Pfad mithilfe von *writeDb* in die Datenbank geschrieben und die Position mit einer Kombination aus GPS/WLAN/Funkmasten ermittelt. Bei einer Messgenauigkeit unter 200 Meter wird die Position zum gleichen Event gespeichert.

```
<src type="Filesystem" path="/sdcard/DCIM/Camera/"> ...
<src type="CallOutAndIn" path="/sdcard/all/call/"> ...
```

Abbildung 5.13: Spezifikation der *src* Referenzen.

- $callDuration$ Dauer des Telefongespräches in Sekunden.
- $accelerationStd$ Standardabweichung der Beschleunigung.
- $barometerPascal$ Luftdruck in Pascal.
- $brightnessLux$ Helligkeit in Lux.
- $locationAccuracy$ Genauigkeit der Lokalisierung.
- $gyroscopeAngleChange$ Durchschnittliche Änderung des Rotationswinkels.
- $loudness$ Durchschnittliche gerätespezifische Lautstärke.

Die Bedingung einer Abhängigkeit ist erfüllt, wenn sich der Attributwert im geschlossenen Intervall des Sensorattributes befindet. Ein Intervall wird mithilfe von zwei Zahlen getrennt durch

ein Komma definiert. Bei der ersten Zahl handelt es sich immer um ein Minimum und bei der zweiten um ein Maximum. Die Verwendung eines $\leq$ und $\geq$ Operators wird ebenfalls unterstützt, indem der offene Bereich nicht mit einer Zahl belegt wird. Im Beispiel Abbildung 5.14 ist die Bedingung erfüllt, wenn der Wert ≥ 100 ist.

```
<dependency brightnessLux="[100,]">
        <writeBb data="location"/>
</dependency>
```

Abbildung 5.14: Spezifikation von *dependency* und *writeDb* Knoten. Falls die Bedingung erfüllt ist, (die Helligkeit muss mindestens 100 Lux betragen) wird mittels *writeDb* Knoten die Position in die Datenbank gespeichert.

writeDb

Ein *writeDb* Element definiert mit dem Attribut *data* jene Daten, die in die Datenbank geschrieben werden. Folgende Sensordaten sind verfügbar:

- $acceleration$ $\quad Standardabweichung\ der\ Beschleunigung$
- $audioPath$ $\quad Pfad\ zur\ Audiodatei$
- $barometer$ $\quad Luftdruck\ in\ Pascal$
- $brightness$ $\quad Helligkeit\ in\ Lux$
- $call$ $\quad Daten\ eines\ Telefonanrufes$
- $cameraPath$ $\quad Pfad\ der\ atomatischen\ Bilder$
- $filePath$ $\quad Pfad\ der\ manuellen\ Bilder$
- $gyroscope$ $\quad durchschnittliche\ Winkeländerung$
- $location$ $\quad Positionsdaten$
- $loudness$ $\quad gerätespezifische\ Lautstärke$
- $rotation$ $\quad Daten\ über\ die\ Lage\ des\ Smartphones$
- sms $\quad Daten\ einer\ SMS-Nachricht$

Einige Daten können aus mehreren Teildaten bestehen. Bei den Daten *call*, *sms*, *location* und *rotation* ist das Abspeichern einzelner Teildaten möglich. Dazu können folgende Attributwerte verwendet werden:

- $call$ $\quad caller, caller\_number, receiver, receiver\_number, duration, path$
- $location$ $\quad provider, latitude, longitude, altitude, speed, accuracy$
- $rotation$ $\quad matrix, angle$
- sms $\quad message, sender, sender\_number, receiver, receiver\_number$

5.6.4 Annotations

```
<annotations>
        <annotation name="diary"> ...
        </annotation>
        <annotation name="health"> ...
        </annotation>
</annotations>
```

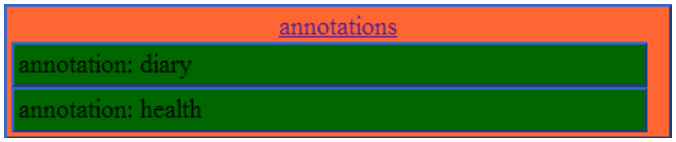


Abbildung 5.15: Grundlegende *annotations*-Spezifikation.

Anmerkungen können im *annotations* Element (siehe Abbildung 5.15) deklariert werden. Die variable Spezifikation hat den Vorteil, dass benutzerdefinierte Anmerkungen erstellt werden können. Damit werden einfache allgemeine Notizen oder umfangreichere Erfassungen unterstützt.

annotation

```
<annotation name="diary">
        <note name="entry" value="text"/>
</annotation>
```

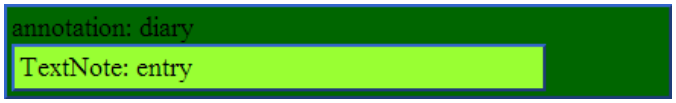


Abbildung 5.16: Beispiel einer annotation Deklaration, die mithilfe eines *note* Knotens eine Textanmerkung definiert.

Verfügbares Attribut:

– *name* gibt den Namen des Eintrages an

In Abbildung 5.16 wird eine einfache Textanmerkung deklariert.

note

Verfügbares Attribut:

– *name* deklariert den Namen eines Attributes

```
<annotation name="health">
        <note name="headache">
                <note name="kind" value="option"/>
                        <option value="pulsating"/>
                        <option value="partial"/>
                </note>
                <note name="strength" value="[0,10]"/>
                <note name="other" value="text"/>
        </note>
</annotation>
```

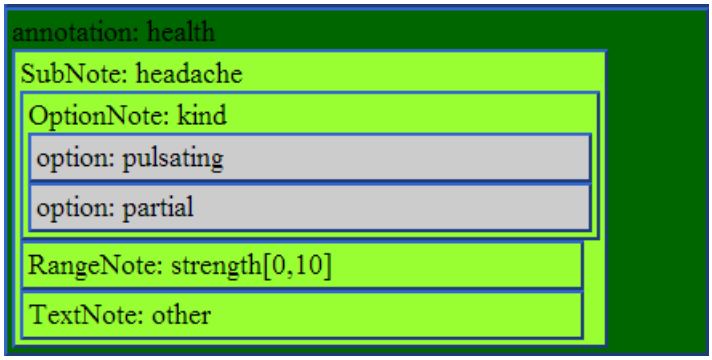


Abbildung 5.17: Erstellung einer komplexeren *annotation*-Spezifikation, die aus den Einträgen *Art der Kopfschmerzen (kind)*, *Stärke der Kopfschmerzen (strength)* und einer *Textanmerkung (other)* bestehen.

– *value* deklariert den Type der Variable

Dabei ist pro Anmerkung mindestens ein *note* Element erforderlich. Der Variablentyp der Notiz wird mit einem *value*-Attributwert spezifiziert. Dabei kann eine Anmerkung mittels Text (*text*), Zahl (*int*,*double*), logische Aussage (*boolean*), Wertebereich (*[0,10]*) oder Auswahl aus einer Liste (*option*) erstellt werden.

Mithilfe von Kinderelementen ist es möglich, ein Konstrukt von mehreren Variablen pro Anmerkung zu deklarieren. Beispielsweise werden mit der in Abbildung 5.17 deklarierten *health*-Anmerkung folgende drei zusammenhängende Einträge erstellt:

1355667143349, health-headache-other, bad
1355667143349, health-headache-kind, pulsating
1355667143349, health-headache-strength, 2

option

Wird dem *value*-Attribut ein *option*-Wert zugewiesen, ist eine Deklaration der Optionen in einem Kinderknoten notwendig. In den *value*-Attributen der Optionen werden dann keine Variablen deklariert, sondern es erfolgt die direkte Textspezifikation.

5.6.5 Event, Src und Subscriber Übersicht

Für die Verteilung der Daten wird in ALL eine Server/Subscriber Hierarchie verwendet. Zum Erlangen der Sensordaten abboniert ein Subscriber einen Server. Solange ein Server Abonnenten besitzt, werden die Sensoren ausgewertet und die Daten an die Subscriber weitergeleitet.

In den nächsten Tabellen werden die einzelnen Datenquellen *Src* mit zugehörigen Parametern angegeben, die zur Deklaration des XMLs notwendig sind. Die Spalte *trigger* gibt bei den asynchronen Events an, welche Event Auslöser für die Erfassung notwendig sind. Synchrone Events können entweder im Verbund mit asynchronen Events oder durch ein regelmäßiges Zeitintervall erfasst werden. Die Spalte *writeDb* gibt die jeweiligen Parameter zur Abspeicherung an. In *dependency* werden optionale Abhängigkeiten angeführt, und die *Subscriber Klassen* verweisen auf die zuständigen Klassen.

Tabelle 5.1: Asynchrone Events, zugehörige *Src*, *writeDb*, *dependency* und *Subscriber-Klassen*.

Src	writeDb	dependency	Subscriber-Klasse	Parameter	trigger
CallOutAndIn	call*	callDuration	CallSensor, AudioCallSensor	path	call
SmsInAndOut, SmsIn, SmOut	sms*	————	SmsSensor	————	sms
Filesystem	filePath	————	FileObserverSensor	path	makeVideo
Filesystem	filePath	————	FileObserverSensor	path	makePicture
Button	————	————	ButtonSensor	————	mediaButton

Tabelle 5.2: Synchrone Events, zugehörige *Src*, *writeDb*, *dependency* und *Subscriber-Klassen*.

Src	writeDb	dependency	Subscriber-Klasse	Parameter
acceleration	acceleration	accelerationStd	AccelerationSensor	time
audio	audioPath	————	AudioSensor	time
barometer	barometer	barometerPascal	BarometerSensor	————
brightness	brightness	brightnessLux	BrightnessSensor	————
CameraFront, CameraBack, FrontAndBack	cameraPath	————	CameraSensor	path, resolution
location	location*	locationAccuracy	GpsSensor, NetSensor	time, method precission
gyroscope	gyroscope	GyroscopeAngleChange	GyroscopeSensor	time
loudness	loudness	loudness	LoudnessSensor	time
rotation	rotation*	————	RotationSensor	time

* bedeutet, die Daten können auch gefiltert (siehe Kapitel 5.6.3, Seite 67) in die Datenbank gespeichert werden

5.7 Synchroner Trigger

ALL verwendet zur regelmäßigen Erfassung der Sensoren einen periodischen Trigger. Beispielsweise, um periodisch die Position zu erfassen. Für das Verständnis der folgenden Trigger und Sensoren, sind die folgenden Konzepte von Android notwendig:

Activity Hierbei handelt es sich um eine zentrale Applikationskomponente, die einen Bildschirm und grafische Benutzeroberfläche bereitstellt; beispielsweise zur Darstellung von Browser, Spielen oder Terminkalender.

Broadcast Receiver Die asynchrone Einwegkommunikation dient zur Nachrichtenübermittlung von einer beliebigen Activity oder von dem System, dabei wird eine Nachricht (*Broadcast*) an eine oder mehrere Anwendungen (*Broadcast Receiver*) gesendet. Hierbei handelt es sich beispielsweise um die Benachrichtigung einer eintreffenden SMS-Nachricht, Wecker (AlarmManager) oder um ein Telefongespräch.

Content Provider Das sind spezielle Anwendungsbausteine, die einen gleichzeitigen Datenaustausch mehrerer Anwendungen ermöglichen. Beispielsweise zur Verwendung eines gemeinsamen Adressbuches, das gleichzeitig von mehreren Anwendungen gelesen und beschrieben wird. Dabei wird eine SQL-ähnliche Abfrage[3] unterstützt, und eine URI dient zur eindeutigen Identifizierung der Datenquelle. Zu beachten ist, dass dabei keine automatische Synchronisierung erfolgt, und somit eigens realisiert werden muss.

Die ALL Anwendung soll regelmäßig Daten aufzeichnen. Dazu wird ein periodischer Zeitgeber zur Realisierung der Programmsteuerung benötigt. Die Klasse *AlarmManager* der Android API bietet Zugriff auf den Alarmdienst des Systems. Dieser Dienst ermöglicht Anwendungen, Tätigkeiten zu einem definierten Zeitpunkt oder durch einen regelmäßigen Zeitintervall zu starten, indem ein Alarm einen Broadcast mit zugehöriger Nachricht auslöst. In ALL wird ein regelmäßiger Alarm von zehn Sekunden registriert (siehe Abbildung 5.18). Sobald der Alarm ausgelöst wurde, wird ein Broadcast gesendet. Schließlich wird der Broadcast von einem *BroadcastReceiver* empfangen.

Die Klasse *ScheduleReceiver* empfängt einen periodischen Alarm (*Broadcast*) in der *onReceive* Methode (Main Loop). Folgend wird der *ProfileManager* aktualisiert und die LLDL-deklarierten periodischen Trigger ausgeführt.

Wie in Abbildung 5.19 dargestellt, folgt danach die Datenerfassung, indem ein *SensorService* einen Hardware-Sensor abruft, und an die entsprechenden *SensorSubscriber* weiterleitet. Einzelne Trigger wie *CallSensor* liefern bereits eigene Daten wie: Telefonnummer und Name des Gesprächspartners.

[3] intern wird oft eine lokale SQL-Datenbank verwendet

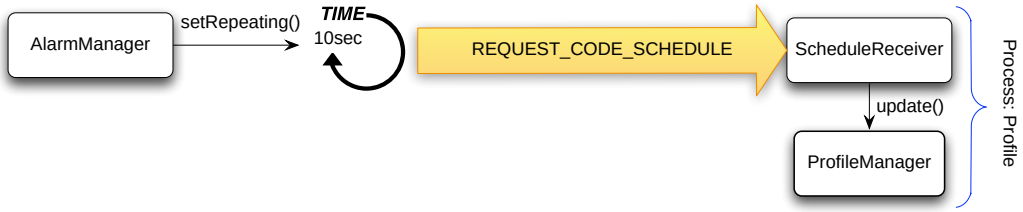


Abbildung 5.18: Regelmäßiger Trigger durch Echtzeituhr.

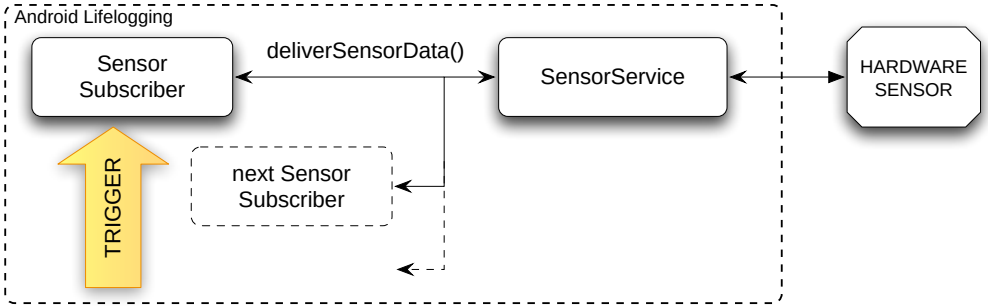


Abbildung 5.19: Abonnement mehrerer SensorSubscriber eines SensorService.

5.8 Asynchroner Trigger

In diesem Abschnitt wird die Funktionsweise der einzelnen Auslöser (Trigger) des Event erklärt. Sämtliche periodischen Events und asynchrone Sensoren sind ebenfalls Trigger. Diese können mithilfe von LLDL als *trigger* eines Events deklariert werden. Die Klassen, die das Interface *SensorService* beinhalten, sind so realisiert, dass sie von beliebig vielen *Subscriber* Instanzen abonniert werden können. Es dürfen beliebig viele periodische zeitliche *trigger* definiert werden. Jeder *SensorService* kann von einem zugehörigen *SensorSubscriber* abonniert werden, und leitet seine Informationen an alle *Subscriber* weiter. Die jeweiligen Abonnenten triggern je nach Konfiguration spezifische Profile, wobei weitere Informationen gesammelt und weiterverarbeitet werden. Sobald die geforderten Daten bei allen *SensorSubscriber* Instanzen eintreffen und ihr Abonnement stornieren, oder eine Zeitüberschreitung eintritt, wird die Datenerfassung beendet.

5.8.1 ButtonSensor

Mit dem *ButtonSensor* ist es möglich, auf ein externes asynchrones Event einzugehen. Es handelt sich dabei um einen Trigger, der selbst keine Daten erfasst. Ein Broadcast wird vom System gesendet, wenn der Play-Knopf auf der Kopfhörersteuerung gedrückt wird (siehe Abbildung 5.20). Dazu bietet die Android API mit der Klasse *AudioManager* die Methode *registerMediaButtonReceiver* an. Dabei ist zu beachten, dass das Drücken und das Loslassen des Media Buttons jeweils einen Broadcast versendet und diese nicht zu unterscheiden sind. Um dem entgegen-

zuwirken, wird die Systemzeit des ersten Triggers gespeichert. Bei jedem weiteren Broadcast wird entschieden, ob es sich um einen neuen Trigger handelt, indem die letzte Systemzeit mit der aktuellen verglichen wird. Falls mindestens fünf Sekunden zwischen den zwei Broadcasts liegen, wird es als ein neues Event gewertet und führt ein Update des entsprechenden Profiles aus. Für den Empfang des *MediaButton* Broadcasts ist die Klasse *ButtonReceiver* zustän-

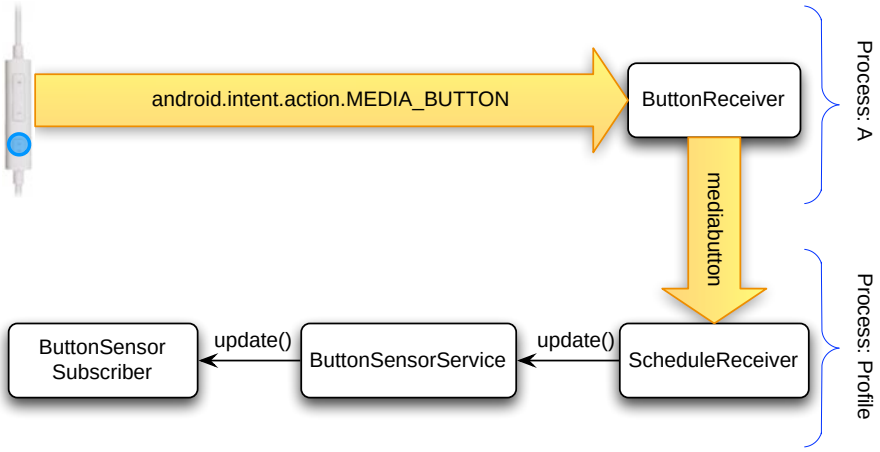


Abbildung 5.20: Asynchroner Trigger des ButtonSensor.

dig. Diese erweitert einen *BroadcastReceiver* und sendet die empfangenen Broadcasts an den *ScheduleReceiver* weiter. Diese Umleitung des Broadcasts ist aus zwei Gründen notwendig. Einerseits wird dadurch verhindert, dass zu viele gleichzeitig schreibende Datenbankzugriffe stattfinden. Andererseits ist das Abspeichern der Systemzeit und das Aufrufen des Profiles durch das ständige Wechseln der Anwendungs-ID des MediaButton Broadcasts nicht möglich. Für das Empfangen des Brodcasts benötigt der *ButtonReceiver* eine Berechtigung, die im Android Manifest definiert ist. Ein *Receiver* verweist dazu auf den *ButtonReceiver* und gibt mithilfe eines Intent-Filters die action *android.intent.action.MEDIA_BUTTON* an.

Einschränkungen: Falls eine andere Anwendung nach der Registrierung ebenfalls den *MediaButton* abonniert, wird der Broadcast an den neuen Abonnenten und nicht mehr an ALL gesendet.

5.8.2 CallSensor

Die Android API ermöglicht es, Broadcasts zu abonnieren, die Statusveränderungen von Telefongesprächen melden. Diese Meldungen beinhalten Informationen über eingehende und ausgehende Anrufe, sowie über die Beendigung eines Gespräches. Der CallSensor wurde so realisiert, dass der Sensor nicht nur als Trigger für Profile genutzt werden kann, sondern mit der Klasse *AudioCallSubscriber* in Kooperation mit dem *AudioSensorService* Gespräche aufnehmen kann.

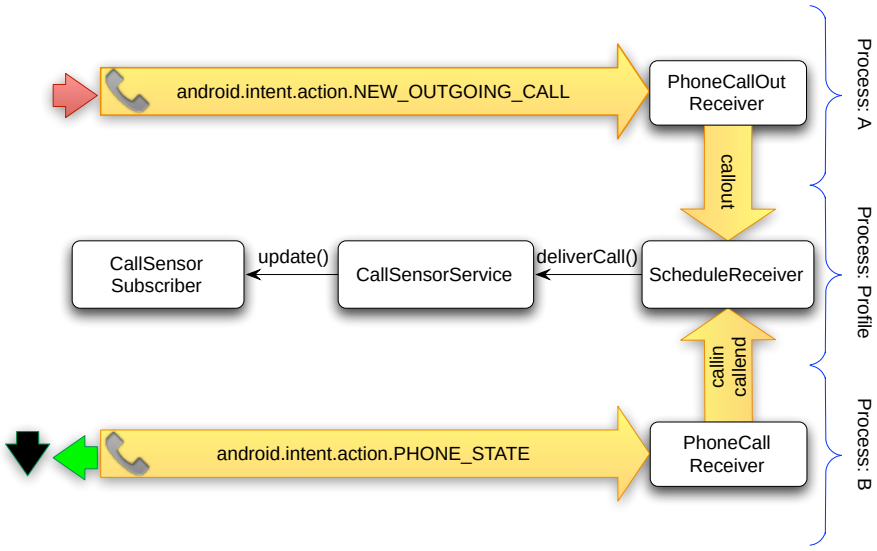


Abbildung 5.21: Asynchrone Abfolge des CallSensor.

Das Empfangen der ausgehenden Telefongespräche wird von der *PhoneCallOutReceiver* Klasse realisiert. Für den Empfang des Broadcasts erweitert auch diese Klasse den *BroadcastReceiver* und sendet einen neuen Broadcast an den *ScheduleReceiver*. Wie in Abbildung 5.21 ersichtlich, wird bei einem ein- oder ausgehenden Anruf jeweils ein Broadcast an den *PhoneCallReceiver* beziehungsweise *PhoneCallOutReceiver* gesendet. Diese Broadcasts des *PhoneCallReceivers* unterscheiden sich anhand des Status. Beim Status *EXTRA_STATE_RINGING* wird ein eintreffender Anruf gemeldet. Hingegen wird beim Status *IDLE* die Beendigung des Gespräches mitgeteilt. Diese Statusmeldungen werden wieder über einen neuen Broadcast an den *ScheduleReceiver* weitergeleitet.

Einschränkungen: Bei einer Konferenzschaltung von drei Personen besteht aber das Problem, dass mit dem *IDLE* Status zwar eine Beendigung eines Telefonpartners mitgeteilt wird, aber nicht mit welchem Teilnehmer. Somit lässt sich nicht ermitteln, mit welchem Gesprächspartner gerade gesprochen wird, und wie lange die einzelnen Gespräche dauern.

5.8.3 FileObserver

Die Android API ermöglicht die Überwachung des Dateisystems. Dazu wird die Linux-Kernel-Funktion *Inotify* verwendet, die Änderungen des Dateisystems meldet (siehe Abbildung 5.22). Ein großer Vorteil dieser Funktion ist, dass das Dateisystem nicht permanent auf Änderungen durchsucht werden muss. Die in Android zur Verfügung gestellte Klasse *FileObserver* kann einzelne Dateien oder ganze Verzeichnisse überwachen. Dazu muss bei der Instanziierung der Klasse lediglich der Pfad angegeben werden. Bei der Überwachung eines Verzeichnisses werden auch Veränderungen aus Unterverzeichnissen gemeldet. Wichtig ist, dass die *FileObserver*

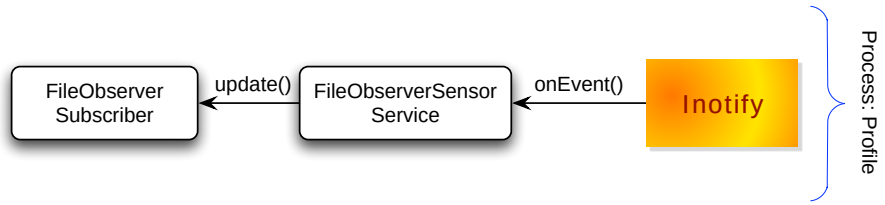


Abbildung 5.22: Aufruf von Inotify an den FileObserver.

Instanz nicht von der automatischen Speicherbereinigung freigegeben wird. Dies erreicht man beispielsweise, indem sie statisch definiert oder nur in einer aktiven Activity benutzt wird. Es gibt elf verschiedene Arten von Dateisystemveränderungen, die von *Inotify* übermittelt werden.

Für Android Lifelogging wichtige Datei-Änderungen sind:

- $CREATE$ Eine Datei wurde erzeugt.
- $MOVED\_TO$ Eine Datei wurde verschoben.

Damit die Klasse *FileObserverSensorService* Aktualisierungen erhält, muss sie den *FileObserver* erweitern. Der *FileObserver* sendet die Aktualisierungen aber nicht mit einem Broadcast, sondern direkt an die Instanz. Von dort aus werden die Events an die *Subscriber* weitergeleitet. Falls ein Verzeichnis mehrfach überwacht werden soll, erkennt dies die *FileObserverSensorService* Klasse. Dabei wird das Verzeichnis dann nur einmal überwacht aber die Veränderung an alle Abonnenten gesendet.

$Einschränkungen$**:** Die Erstellung von Dateien folgt nicht immer nach dem gleichen Schema. Bei den getesteten Geräten erfolgt die Videoaufzeichnung zuerst in eine temporäre Datei, bei Beendigung wird die Datei dann in den Zielpfad verschoben. Dagegen werden Bilder direkt im Dateisystem abgelegt. Weiters ist dies von der Implementierung des Herstellers und der Firmware Version abhängig. Auf dem *Samsung GALAXY S III I9300* werden Veränderungen des Dateisystems doppelt gemeldet. Dieses Problem wird gelöst, indem doppelt vorkommende Meldungen innerhalb einer Sekunde mithilfe einer Liste gefiltert werden.

5.8.4 SmsSensor

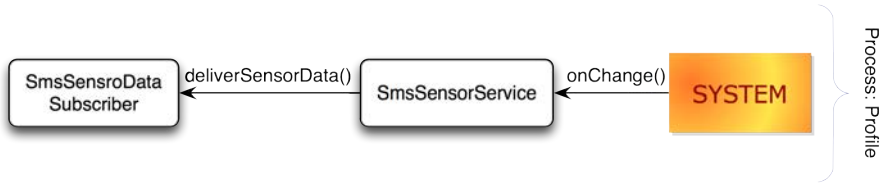


Abbildung 5.23: SmsSensor Benachrichtigung von empfangenen und gesendeten SMS-Nachrichten.

Die Android API ermöglicht das Empfangen und Versenden von SMS-Nachrichten. Eine Möglichkeit ein- und ausgehende SMS zu protokollieren, ist diese mithilfe eines *ContentObserver* zu überwachen. In der Android API gibt es dazu eine vordefinierte URI *„content://sms“*, die über einen ContentResolver abonniert werden kann. Die Klasse *SmsSensorService* erweitert den *ContentObserver* und abonniert ein- und ausgehende SMS-Nachrichten. Beim Versenden oder Empfangen (siehe Abbildung 5.23) einer SMS-Nachricht wird die *onChange* Methode aufgerufen, und die zugehörigen Daten über eine Query ausgelesen. Der Parameter Type einer Nachricht gibt dabei an, ob es sich um eine eintreffende (type=1) oder ausgehende (type=2) SMS-Nachricht handelt.

Dabei ist zu beachten, dass eine eintreffende SMS auch mehrmals die *onChange* Methode aufrufen kann. Der Grund dafür ist, dass komplexere Situationen wie der Lesestatus übermittelt werden. Deswegen ist bei einer eintreffenden SMS-Nachricht darauf zu achten:

- $read \neq 1$ Falls die SMS noch nicht gelesen wurde.
- $deletable = 1$ Falls die SMS nicht löschbar ist.

$Einschränkungen$**:** Falls der Autostart zu spät erfolgt, und vor dem Abonnieren des *ContentObserver* SMS-Nachrichten eintreffen, müssen diese nachträglich importiert werden.

5.9 Sensoren

Folgend wird die Datenerfassung der synchronen Sensoren erklärt. Wie schon bei den asynchronen Sensoren und der Echtzeituhr, gibt es hierbei *SensorSubscriber*, die einen geeigneten *SensorService* abonnieren.

Die Android API verfügt über eine *SensorManager*-Klasse, die ein Abonnement von gewissen Daten erlaubt. Dazu erweitert die jeweilige *SensorService*-Klasse den *SensorEventListener* und empfängt über die Methode *onSensorChanged* die Rohdaten des Sensors. Solange mindestens ein *SensorSubscriber* den *SensorService* abonniert, werden Daten gesammelt. Diese Daten werden gegebenenfalls weiterverarbeitet und weitergegeben. Folgende Sensoren stellt die Android API zur Verfügung:

- $TYPE\_ACCELEROMETER$ Beschleunigungssensor
- $TYPE\_AMBIENT\_TEMPERATURE$ Umgebungstemperatur
- $TYPE\_GRAVITY$ Beschleunigung (Rohdaten)
- $TYPE\_GYROSCOPE$ Gyroskop
- $TYPE\_LIGHT$ Helligkeit
- $TYPE\_LINEAR\_ACCELERATION$ Beschleunigung (fusioniert)
- $TYPE\_MAGNETIC\_FIELD$ Magnetfeld (Rohdaten)
- $TYPE\_ORIENTATION$ Lage (fusioniert)
- $TYPE\_PRESSURE$ Luftdruck

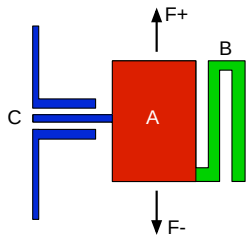


Abbildung 5.24: Schematischer Aufbau eines Beschleunigungssensors Mikrosystemes.
A: Probemasse, B: Federung, C: veränderbare Kondensatoren, F±: einwirkende Kraft.
(In Anlehnung an: [104])

- $TYPE\_PROXIMITY$ Entfernung zum Display
- $TYPE\_RELATIVE\_HUMIDITY$ Luftfeuchtigkeit
- $TYPE\_ROTATION\_VECTOR$ Lage (fusioniert)
- $TYPE\_TEMPERATURE$ veralteter Temperatur Sensor

Für Android Lifelogging wurden zusätzlich folgende Sensoren definiert:

$-AudioSensor$	Nimmt ein Telefongespräch oder Umgebungsgeräusch auf.
$-CameraSensor$	Aufnahme eines Bildes durch verfügbare Kamera.
$-LoudnessSensor$	Ermittlung der Lautstärke aus verfügbaren Quellen.

Kalendereinträge: Für die Erfassung der Kalender Daten werden sämtliche relevanten Kalender Einträge in die SQL-Datenbank kopiert. Als relevant gelten alle Einträge deren Zeitabstand in einem definierten Zeitbereich liegen.

5.9.1 AccelerationSensor

Ein Beschleunigungssensor ist ein Messinstrument, das die relative Beschleunigung im Raum misst. Zur Ermittlung der Beschleunigung wird die Ablenkung einer Masse beobachtet. In moderner Halbleiterform befinden sich dazu drei Mikrosysteme (siehe Abbildung 5.24) einer gefederten Masse, deren Ablenkung über Kapazitätsveränderungen gemessen wird [103].

Diese drei Systeme sind zueinander jeweils orthogonal angeordnet, damit eine Messung aller Achsen möglich ist. Die Einheit der Beschleunigung wird in m/s^2 angegeben. Im iPhone wurde 2007 erstmals ein Beschleunigungssensor in ein Smartphone verbaut, um damit die Displayausrichtung anzupassen. Die verbauten Sensoren wie der BMA150 ermöglichen meist verschiedene Sensorbereiche wie ± 2, ± 4, $\pm 8g$, wobei von der Android API der kleinere und genauere $\pm 2g$ Bereich verwendet wird [105]. Durch einmalige Integration erhält man die Geschwindigkeitsänderung und bei zweifacher Integration den relativen Weg. Eine Anwendung ist die Erkennung von verschiedenen Aktivitäten wie Laufen, Gehen und Ruhen.

In einer ruhenden Position erfährt das Smartphone eine Beschleunigung, die der Erdbeschleunigung $1\ g = 9,807\ m/s^2$ entspricht. Die maximale in eine Dimension gemessene relative Beschleunigung beträgt $2g$. Wobei bei entsprechender Lage eine Aufteilung der Beschleunigung in zwei oder drei Dimensionen erfolgen kann. In diesem Fall ist es möglich Beschleunigungen bis zu $\sqrt{2^2 + 2^2 + 2^2} = 2 * \sqrt{3}$ zu messen.

Durch Integration ist es möglich, ein Trägheitsnavigationssystem zu realisieren. Dieses kann für die Navigation in Innenräumen benutzt werden, und kommt ohne Hilfe von Satelliten oder sonstigen Lokalisierungsverfahren aus. In dem Projekt *gesturedroid* wurde bereits versucht ein solches Android-basiertes Trägheitsnavigationsgerät zu realisieren [106]. Leider ohne Erfolg, denn die Daten des Beschleunigungssensors im Smartphone sind zu ungenau, um dies zu realisieren. Zusätzlich war damals in den Testgeräten noch kein Gyroskop verbaut, das die Bestimmung der Lage im Raum vereinfacht. Aktuelle Sensoren müssten circa um den Faktor zehn genauer sein, damit brauchbare Ergebnisse ermittelt werden können.

In Android Lifelogging wird nach der Datenerfassung in der Methode *processData* die Standardabweichung berechnet. Dazu wird zuerst die euklidische Norm der dreidimensionalen Rohdaten berechnet. Diese normierte Beschleunigung korrespondiert mit der relativen Beschleunigung des Smartphones im Raum und entspricht in Ruhelage der Erdbeschleunigung $1\ g$.

Aus den dreidimensionalen Rohdaten a_1, a_2, a_3 wird die relative Beschleunigung $a = \sqrt{a_1^2 + a_2^2 + a_3^2}$ und anschließend die Standardabweichung berechnet, mit der auf die Aktivität geschlossen werden kann.

$Einschränkungen$**:** Auf dem *Nexus S* Smartphone fällt unter der Betriebssystemversion *4.1.1* gelegentlich die Übermittlung der Beschleunigungsdaten aus. Eine Behebung des Problems ist nur durch einen Neustart möglich.

5.9.2 AudioSensor

Ein Mikrofon ist ein Schallwandler, der Schallwellen in korrespondierende elektrische Signale umwandelt. Es ist essenziell für ein Smartphone, damit Telefonate getätigt werden können. Das erste Android Smartphone mit zwei Mikrofonen war das Nexus One (Jänner 2010). Mithilfe des zweiten Mikrofons werden während eines Telefongespräches aktiv Hintergrundgeräusche unterdrückt. Dieses zusätzliche Mikrofon nimmt durch die Positionierung auf dem Telefon vermehrt Hintergrundgeräusche auf. Durch das Abschwächen dieses Störgeräusches wird das eigentliche Signal, das zum Gesprächspartner gesendet wird, klarer. Seit 2012 gibt es auch Smartphones, die das zweite Mikrofon für eine Stereo Audioaufnahme verwenden. Die Android API erlaubt die Tonaufnahme von acht unterschiedlichen Audio-Quellen:

$-CAMCORDER$	Mikrofon mit der gleichen Orientierung der Kamera
$-DEFAULT$	Standard Audio Quelle
$-MIC$	Standard Mikrofon
$-VOICE\_CALL$	eingehende und ausgehende Sprachleitung
$-VOICE\_COMMUNICATION$	Audio Quelle spezielle für VoIP

$-VOICE\_DOWNLINK$	eingehende Sprachleitung
$-VOICE\_RECOGNITION$	Audio Quelle optimiert für Spracherkennung.
$-VOICE\_UPLINK$	ausgehende Sprachleitung.

Für eine Lifeloggінganwendung sind besonders VOICE_CALL für die Aufzeichnung von Telefonaten und DEFAULT für allgemeine Aufnahmen wichtig. Die API bietet die Möglichkeit die Lautstärke als normalisierte, maximale absolute Amplitude der letzten Millisekunden auszulesen. Dafür steht die Methode *MediaRecorder.gerMaxAmplitude()* zur Verfügung. Wobei diese Lautstärke nicht in einer absoluten Maßeinheit gemessen, sondern hardwarespezifisch ist.

Die Klasse *AudioSensor* ist für die Aufzeichnung von Telefongesprächen, Umgebungsgeräuschen und der Ermittlung der Lautstärke zuständig. Dazu wird die von der Android zur Verfügung gestellte *MediaRecorder* Klasse zur Audio Aufzeichnung aus unterschiedlichen Quellen verwendet. Die Klasse *AudioSensorService* ist so umgesetzt, dass sie gleichzeitig Audio aufnehmen und die Lautstärke ermitteln kann. Dazu verfügt sie über mehrere Zustände (siehe Abbildung 5.25), die vorrangig auch den Aufnahmeort angeben. Beim Starten der Erfassung befindet sich der AudioSensor zuerst im INITIAL-Zustand. Dort werden notwendige Parameter wie der Zeitstempel der Aufnahme gesetzt. Danach geht der Zustand in *START* über und die Aufnahme wird gestartet. Bei einem Telefongespräch wird die Audioquelle *VOICE_CALL* gewählt, die eingehende und ausgehende Telefonleitungen aufnimmt, der Zustand geht in *CALL* über. Bei einer Aufnahme von Umgebungsgeräuschen wird die Audioquelle *DEFAULT* gewählt, und der Zustand ändert sich auf *REC*. Bei den zwei bisherigen Möglichkeiten erfolgt die Aufnahme in das 3gp Containerformat mit dem AAC-Codec. Falls dieser Codec auf dem Smartphone nicht verfügbar ist, erfolgt die Codierung mit dem AMR Wideband Codec. Zusätzlich zu den erwähnten Aufnahmearten kann nebenläufig die Lautstärke erfasst werden. Dazu kann der *AudioSensorService* von einem *LoudnessSensorSubscriber* abonniert werden. Ist nur die Ermittlung der Lautstärke erwünscht, wird von der Datenquelle *DEFAULT* auf das Linux Nulldevice *„/dev/null“* aufgenommen und der Zustand geht in *LOUDNESS* über. Ein Wechsel zwischen den Zuständen CALL, REC und LOUDNESS kann jederzeit ohne Unterbrechung erfolgen.

$Einschränkungen$**:** Eine gleichzeitige Aufnahme von unterschiedlichen Quellen wird von der Android API nicht unterstützt. Es wird auch nur eine Instanz einer Anwendung zu einem Zeitpunkt unterstützt. Zudem unterstützt nicht jedes Smartphone die gleichen Codecs.

5.9.3 BarometerSensor

Ein *BarometerSensor* misst den Luftdruck, genauer gesagt den hydrostatischen Druck. Die Gravitation verursacht innerhalb der Atmosphäre einen Druck auf die Luft. Dieser Druck wird in der Einheit Pascal (Pa) gemessen. Ein Pa entspricht einer Kraft von einem Newton auf einer Fläche von einem Quadratmeter. Dabei wird der Luftdruck meistens in Hektopascal $1\ hPa$ = $100\ Pa$ angegeben. Als Normaldruck ist ein Druck von $1013,25\ hPa$ definiert, dieser ist der durchschnittlich gemessene Luftdruck auf Meeresspiegelhöhe. Wobei dieser Normaldruck, wie in Abbildung 5.26 ersichtlich, natürlichen Schwankungen unterworfen ist.

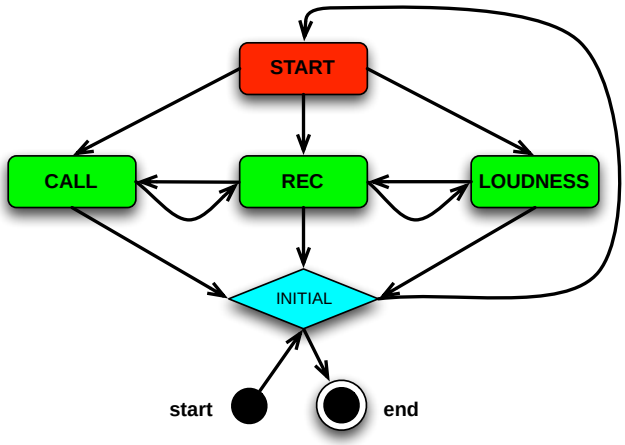


Abbildung 5.25: Zustände der AudioSensor Klasse.

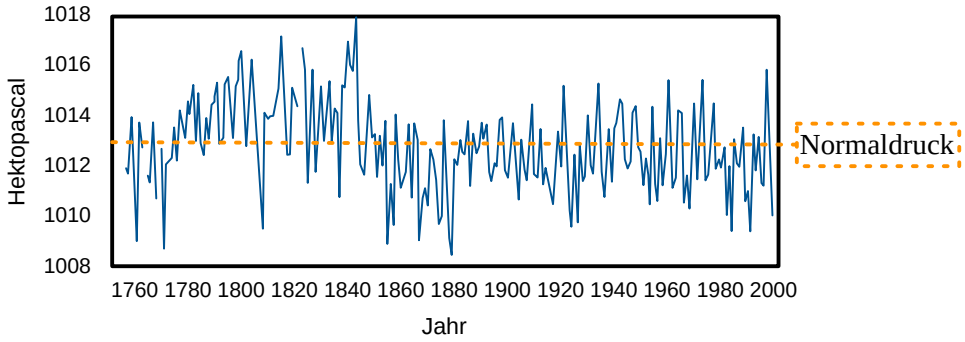


Abbildung 5.26: Luftdruck in hPa in Stockholm von 1756-1998 (Quelle: [107] S. 198).

Der Druck pro Höhenmeter ändert sich in der Nähe des Meeresspiegels schneller als auf einem Berg. Der Grund hierfür ist, dass obere Luftmassen auf die unteren drücken. Der Luftdruck bezüglich der Höhe wird in der barometrischen Höhenformel 5.1 durch eine Exponentialfunktion beschrieben [108] .

$$p(z) = p(0)exp\left(-\frac{mgh}{kT}\right) \tag{5.1}$$

$p(0)$: Normaldruck von $1013,25\ hPa$
m: molare Masse von $0,029\ kg/Mole$ für trockene Luft
g: Erdbeschleunigung von $9,807\ m/s^2$
h: Höhe in Meter
k: Boltzmann Konstante von $8,3145\ J/(MoleK)$
T: Temperatur von $294\ K(21°)$

Der Stephansdom in Wien liegt auf einer Meeresspiegelhöhe von circa 170 Meter. Der Südturm ist bis auf eine Höhe von 72 Meter öffentlich zugänglich, und liegt somit auf circa 242 Meter

Meereshöhe. Somit ergibt sich am Grund des Stephansdoms ein Luftdruck von $993,4\ hPa$ und für den 72 Meter höheren Südturm[4] ein Luftdruck von $985,1\ hPa$. Das entspricht in dieser Höhe einem Unterschied von circa $1\ hPa$ alle neun Meter. Ein solcher Sensor kann somit zur Detektion von raschen Höhenunterschieden genutzt werden. Der Barometer Sensor wurde erstmals im *Samsung Galaxy Nexus I9250* mit der Android 4.0 API aufgenommen (Oktober 2011) und wird seitdem in allen neueren Android Smartphones verbaut. Der Grund für die Integration eines Barometers in ein Smartphone ist eine schnellerer GPS-Lokalisierung. Durch den Luftdruck kann die ungefähre Höhe abgeschätzt werden. Dank der Höheninformation kann die Position schneller berechnet werden.

In Android Lifelogging wird der erste gemessene Wert herangezogen. Jeder *Subscriber* wartet maximal zwei Sekunden auf den aktuellen Luftdruck und gibt den Wert weiter.

5.9.4 BrightnessSensor

Ein Helligkeitssensor misst die umgebende Beleuchtungsstärke in der Einheit Lux. Dabei kommt ein Fotowiderstand - Light Dependent Resistor (LDR) zum Einsatz. Bei diesem werden durch das eintreffende Licht Elektronen frei und Widerstand verringert sich. Der primäre Zweck eines solchen Sensors ist es, die Displayhelligkeit zu steuern, um Energie zu sparen. Für Android Lifelogging ermittelt die Klasse *BrightnessSensorService* die Helligkeit. Die einzelnen *Subscriber* abonnieren den Service und warten maximal zwei Sekunden, bis die Helligkeitsdaten eintreffen. Der erste gemessene Wert beendet die Messung.

5.9.5 CameraSensor

Mithilfe der Klasse *CameraSensorService* können Bilder der verfügbaren Kameras aufgenommen werden. Damit ein Bild gemacht werden kann, müssen der Pfad, die Bildgröße und die gewünschte Kamera angegeben werden. Abhängig, ob die hintere oder, falls verfügbar, die vordere Kamera benutzt wird, kann die Lage während der Aufnahme erfasst und berechnet werden. Anhand dieser Lage wird das Bild entsprechend rotiert.

$Einschränkungen$**:** Da die Gerätetreiber der einzelnen Smartphones nicht einheitlich implementiert wurden, gibt es mehrere Probleme. Es ist nicht mit jedem Android Device möglich, Bilder ohne eine Vorschauanzeige zu machen. Beispielsweise ist es bei dem *Nexus S* Telefon zwar möglich, Bilder ohne Vorschau zu machen, dabei funktioniert der Weißabgleich aber nicht korrekt. Damit die automatische Aufnahme von Bildern dennoch funktioniert, wurde eine eigene Activity *PictureActivity* realisiert, die für entsprechende Smartphones eine Vorschau anzeigt. Wünschenswert wären zusätzliche Optionen der Android API wie: ISO oder Blendeneinstellungen.

5.9.6 GpsSensor/NetSensor

Zur Ermittlung der Position in geographischer Länge und Breite (siehe Abbildung 5.27) können Satelliten- beziehungsweise WLAN- und sendemastenbasierte Lokalisierungsverfahren verwen-

[4] Berechnung in Wolfram Alpha: `http://goo.gl/b4gtt`

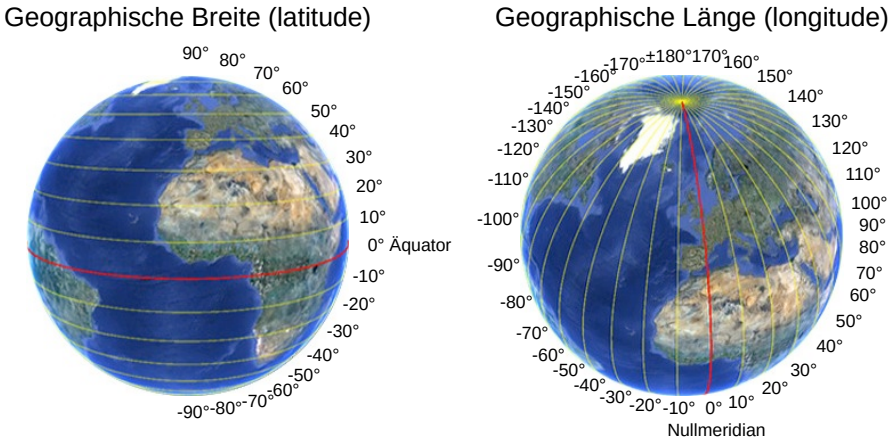


Abbildung 5.27: Die geographische Breite ± 90 ° und geographische Länge ± 180 ° der Erde.

det werden. Ein mithilfe der LLDL definierter *Subscriber* sucht dabei für eine gewisse Anzahl an Sekunden nach neuen Positionsdaten. Eine Möglichkeit ist es, alle Positionsrohdaten zu speichern. Um die Genauigkeit der Lokalisierung zu erhöhen, ist die optionale Berechnung des Mittelwertes oder des Medians einer Datenreihe möglich. Dazu wird jeweils per LLDL die minimale Anzahl von benötigten Messwerten angegeben. Dies entspricht einem minimalen Vertrauenswert. Ein zweiter Wert hilft Energie zu sparen, indem die Datenerfassung beendet wird, sobald genausoviele Daten erfasst wurden, wie der Wert angibt. Neuere Handys können dank Global Navigation Satellite System (GNSS) mehr Satellitensysteme gleichzeitig zur Lokalisierung verwenden. Damit kann die Zeit bis zur ersten Lokalisierung verringert werden.

Einschränkungen: Der GPS-Sensor benötigt viel Energie, und die Initialisierungszeit schwankt stark. Manchmal ist die erste Lokalisierung schon nach circa fünf Sekunden möglich. Falls aber kein Assisted-GPS-Dienst verfügbar ist, dauert es bis zu zehn Minuten. Bei Innenräumen ist eine schlechte bis gar keine Erfassung möglich. Der NetSensor hingegen ermittelt die Position schnell. Die ermittelten Daten sind aber ungenauer und funktionieren nur mit aktiver Datenverbindung.

5.9.7 GyroscopeSensor

Ein Kreiselinstrument (Gyroskop) misst Winkeländerungen pro Zeiteinheit. Bei klassischen Kreiseln dreht sich eine Masse schnell um eine Achse und stabilisiert die längste Achse. Einer der genauesten Kreisel basiert auf dem sogenannten Sagnac-Effekt [109]. Dabei wird ein Laser in einem rotierenden System gleichzeitig, mithilfe eines halbdurchlässigen Spiegels getrennt, in und gegen die Rotationsrichtung geschickt. Der in Rotationsrichtung laufende Laserstrahl legt dabei einen geringfügig weiteren Weg zurück. Am Ende werden die Laserstrahlen wieder zusammengeführt, wobei durch den Laufzeitunterschied die Wellen zueinander phasenverschoben ankommen und Interferenzen entstehen [110]. Durch Veränderung der Interferenzen kann auf die Rotationsrichtung und somit auf die Winkeländerung geschlossen werden. Solche Gyrosko-

pe sind sehr genau mit einer Abweichung von bis zu unter einem Grad pro Stunde. Weiter gibt es noch zwei ähnliche optische Kreisel, den faseroptischen- (FOG) und Ringlaser-Kreisel (RLG). Letzterer erreicht eine Genauigkeit von unter $0,001$ Grad pro Stunde Abweichung.

In modernen Smartphones werden keine rotierenden Teile oder Laser sondern eine Variante eines Vibrations-Kreisels eingesetzt [111]. Diese Systeme bestehen aus einer schwingenden Mikro Mechanik, welche durch einen Piezo-Kristall in Schwingung versetzt wird [112]. Erstmals wurde ein Gyroskop 2010 im iPhone 4 als Eingabe für Spiele verbaut.

5.9.8 RotationSensor

Nur anhand des Beschleunigungssensors ist die räumliche Bestimmung der Lage nicht möglich. Deshalb wird zusätzlich noch ein elektronischer Kompass verwendet. Das Funktionsprinzip eines solchen Kompasses wie beispielsweise eines AK8973 basiert auf dem Hall-Effekt (siehe Abbildung 5.28). Dieser beschreibt eine proportional zum Magnetfeld auftretende Spannung (Hallspannung) eines von einem statischen Magnetfeld durchflossenen Leiters. Mit der Android

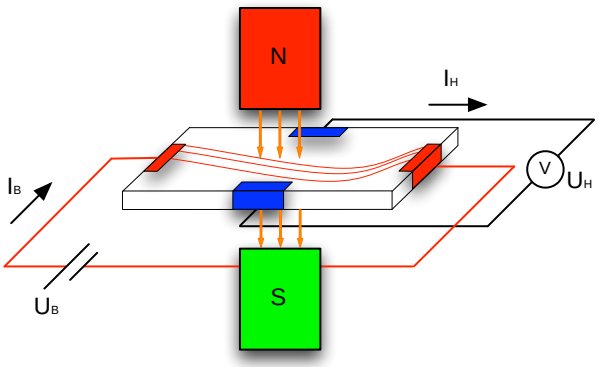


Abbildung 5.28: Veranschaulichung des Hall-Effekts, (In Anlehnung an: [113] S. 447)

API lassen sich die Daten des elektronischen Kompasses über ein Abonnement der *SensorManager* Klasse mit dem Type *TYPE_MAGNETIC_FIELD* ermitteln. Diese Daten entsprechen einem Vektor, der in Richtung der Inklination zeigt und somit nicht notwendigerweise auf der XY-Ebene des Smartphones liegt. Wenn die aktuellen Vektoren der Beschleunigung und des Kompasses vorhanden sind, lässt sich die Lage des Smartphones errechnen (siehe Abbildung 5.29). Dazu wird zuerst der invertierte X-Vektor als Kreuzprodukt von Beschleunigung und Kompassvektor gebildet. Die Y-Achse ergibt sich dann aus dem negierten Beschleunigungsvektor und die Z-Achse bildet das Kreuzprodukt aus X- und Y-Achse.

Einschränkungen: Da die Erdbeschleunigung nur ohne relative Beschleunigung akkurat gemessen wird, resultiert daraus eine Verfälschung der Lageberechnung. In Innenräumen können Überlagerungen des Erdmagnetfeldes durch stromdurchflossene Leiter auftreten. Des Weiteren

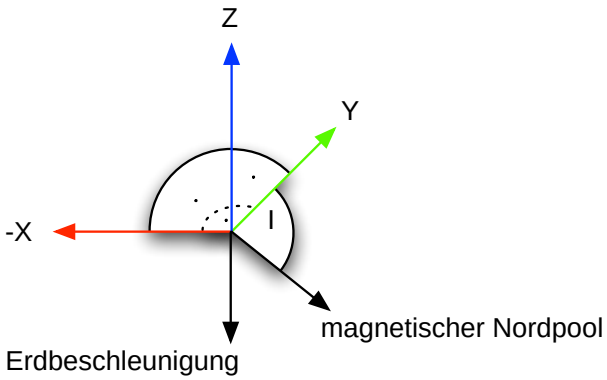


Abbildung 5.29: Berechnung der Lage. I: Inklination, X, Y, Z, Erdbeschleunigung, magnetischer Nordpol.

wird die sich ändernde Abweichung zwischen dem geografischen und magnetischen Nordpol[5] nicht berücksichtigt. Die Android API verfügt aber über die Klasse *GeomagneticField* mit deren Hilfe die Abweichung über das *World Magnetic Model* kompensiert wird. Einziger Nachteil dabei ist, dass die ungefähre Position für die Berechnung notwendig ist. Die Erfassung der aktuellen Lage erfolgt aus einer Kombination aus Beschleunigungsvektor und Magnetfeldvektor. Die Lage wird als *OpenGL* kompatible 3x3 Matrix abgespeichert.

[5]Mit *magnetischem Nordpol* ist der geomagnetische arktische Pol gemeint.

```
<src type="location" maxSearchTime="10" method="gps" precision="median[3,5]">
        <dependency locationAccuracy="[0,200]">
                <writeDb data="location"/>
        </dependency>
</src>
```

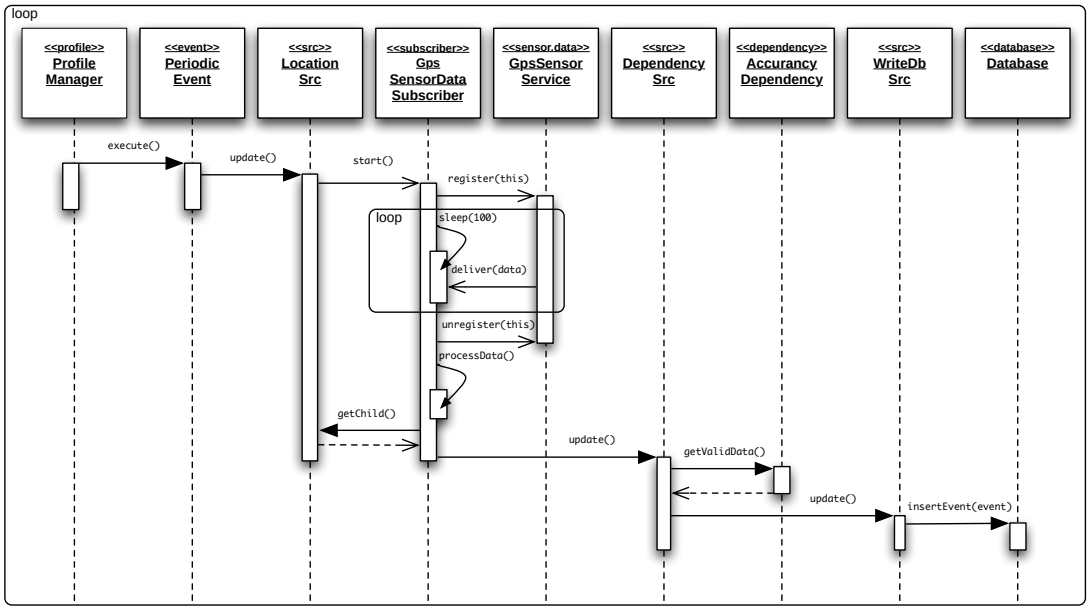


Abbildung 5.30: Synchrones Update der GPS-Position.

5.10 Ablauf synchroner und asynchroner Datenerfassung

Der interne Programmablauf wird durch die LLDL-Deklaration spezifiziert. Dabei wird je nach Spezifikation ein Konstrukt von Java-Objekten erstellt. Folgend werden zwei Beispielkonfigurationen für synchrone und asynchrone Sensoren gezeigt. Dabei erfolgt zuerst die Deklaration der XML-Konfiguration in LLDL und darunter wird ein Sequenzdiagramm visualisiert, das den internen Ablauf repräsentiert.

In Abbildung 5.30 ist eine synchrone Erfassung der Position ersichtlich. Dabei wird für maximal zehn Sekunden die GPS-Position ermittelt und mithilfe eines Medianfilter der Größe fünf gefiltert. Ist die Abweichung gleich oder kleiner als 200 Meter erfolgt der Datenbankeintrag. In Abbildung 5.31 ist eine asynchrone Erfassung von eingehenden SMS-Nachrichten ersichtlich.

5.11 Entwicklungsumgebung

Als Entwicklungsrechner stehen ein Mac OS X 10.7 und ein Windows 7-Rechner zur Verfügung. Dabei wird die Entwicklungsumgebung Eclipse 4.2.1 *Juno* mit dem Java Development

```
<src type="SmsIn">
        <writeDb data="sms"/>
</src>
```

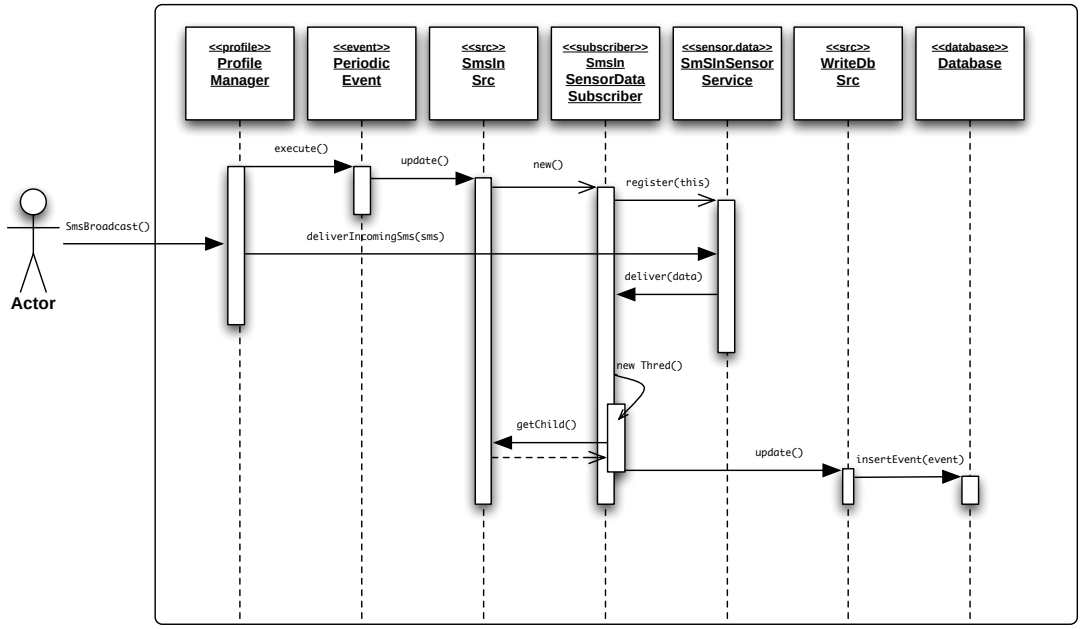


Abbildung 5.31: Asynchrones Update durch eine eintreffende SMS-Nachricht.

Kit (Mac OS X: 1.6) verwendet. Für die Android-Entwicklung wird das von Google kostenlos zur Verfügung gestellte Software Development Kit (SDK) und Eclipse Plug-in Android Development Toolkit (ADT) verwendet. Weiters wird Apache Ant für das automatisierte Umwandeln von Vektorgrafiken auf Rastergrafiken verwendet. Zur Erstellung des SSL-Zertifikats wurde die Software *portecle* verwendet (siehe Kapitel A.2, Seite 126).

Eclipse Ist eine Open Source entwickelte und in Java realisierte Entwicklungsumgebung [114]

Apache Ant Hierbei handelt es sich um ein in Java implementiertes Batch-Werkzeug zur Erstellung von automatischen Programmabläufen. Dazu wird eine XML-Datei definiert, in der die Programmkonstrukte definiert werden. *Apache Ant* wurde von der Apache Software Foundation entwickelt und steht unter der *Apache 2.0* Lizenz zur Verfügung [115].

Mogrify Ein Bestandteil der Grafikbibliothek ImageMagick. Mogrify ermöglicht eine Skalierung von SVG Grafen und Umwandlung in das PNG Bildformat [116].

SIPS Scriptable Image Processing System (Sips) dient zum Umwandeln von PDF Dateien und ist Bestandteil vom Apple Mac OS X Betriebssystem [117].

Portecle Portecle ist eine unter der GPL-2.0 Lizenz stehende Software zum Erstellen, Konvertieren und Editieren von Zertifikaten [118].

5.11.1 Smartphoneanforderungen

Zum Testen der Software ist ein Simulator ungeeignet. Sämtliche Sensoren müssen am Gerät getestet werden, da diese nur schwer simuliert werden können. Zum Testen stand folgende Android 4.1 aufwärtskompatible Hardware zur Verfügung:

Galaxy S III: Mit dem *Samsung GALAXY S III I9300* Smartphone lässt sich der tägliche Gebrauch der Software simulieren. Zudem sind alle aktuellen Sensoren des Android-SDK verfügbar.

Nexus 7: Das Tablet *Google Nexus 7* ist aufgrund der geringen Ausmaße und aufgrund des überdimensionierten Akkus besonders für das Testen von Konfigurationen, die viel Energie benötigen, geeignet.

KAPITEL 6

Ergebnisse

In diesem Kapitel werden die Ergebnisse des praktischen Teiles der Diplomarbeit präsentiert. Sämtliche Funktionen von ALL (siehe Abbildung 6.1) wie das Webinterface, die Datenbank und die Visualisierung der Daten werden gezeigt. Schließlich erfolgt eine Auswertung aus verschiedenen Szenarien der Lokalisierung.

Abbildung 6.1: Logo der Android Lifelogging App.

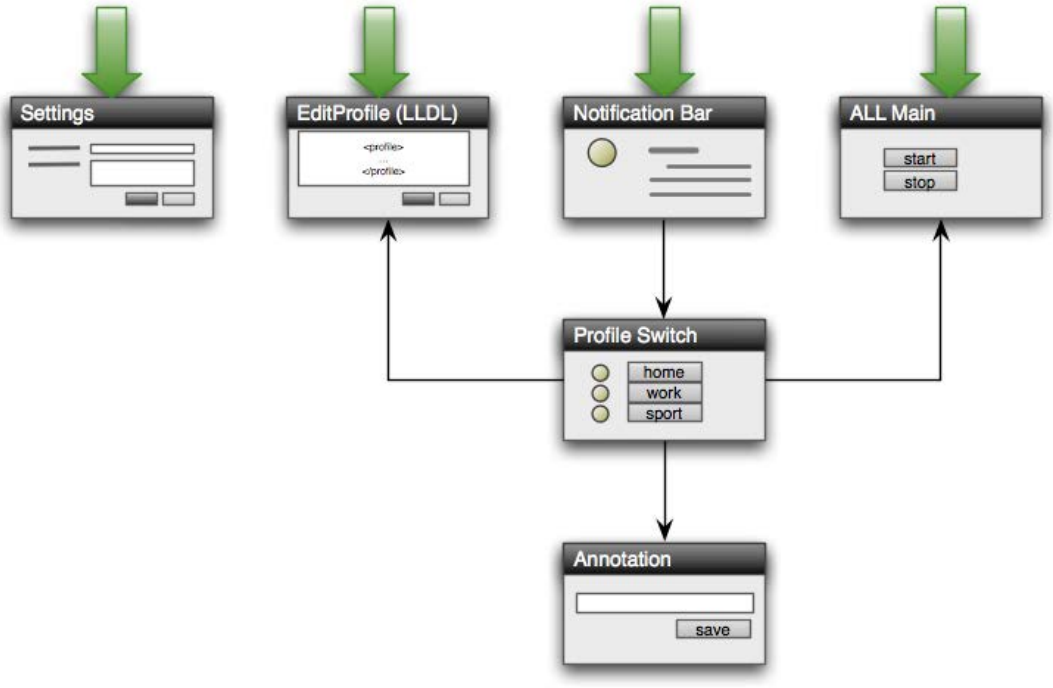


Abbildung 6.2: Android Lifelogging GUI Übersicht.

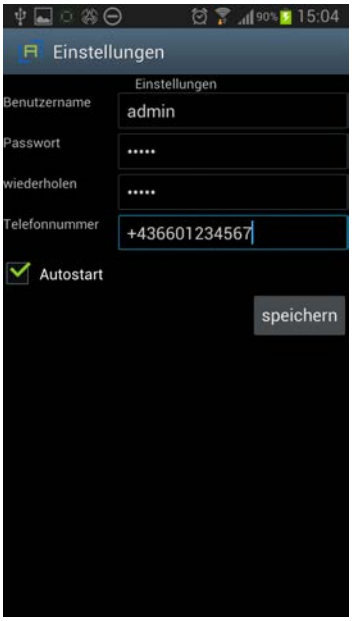
6.1 ALL Benutzeroberfläche am Smartphone

Für die Installation von ALL ist ein geeignetes Android Smartphone ab Version 2.3 notwendig. Der volle Funktionsumfang, wie die Kalenderunterstützung und erweiterte Informationen in der Benachrichtigungsleiste, setzt jedoch mindestens die Version 4.1 und eine aktive mobile Datenverbindung voraus.

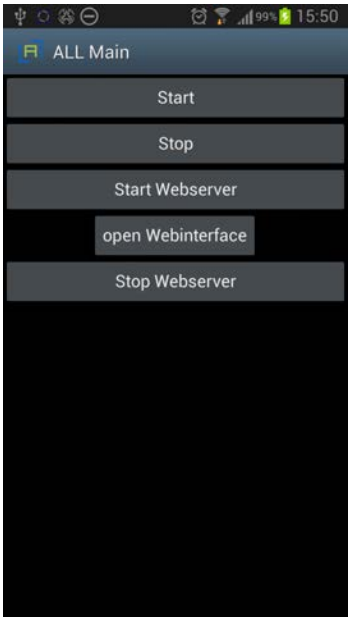
Wie in Abbildung 6.2 visualisiert, bietet ALL vier Einstiegspunkte: *Settings*, *EditProfile*, *Notification Bar* und *All Main* zum Betrieb der Anwendung. Weiters sind noch die internen Aktionen *Profile Switch* und *Annotation* über die *Notification Bar* aufrufbar.

Bei erstmaligem Start, beziehungsweise solange die grundlegenden Einstellungen nicht gesetzt wurden, sind die Einstellungen über *Settings* (siehe Abbildung 6.3a) zu treffen. Wobei Benutzername, Kennwort, Telefonnummer und die Autostart-Option auszufüllen, beziehungsweise auszuwählen sind. Mithilfe von *ALL Main* (siehe Abbildung 6.3b) werden der Hintergrunddienst und das Webservice gestartet, beziehungsweise gestoppt. Bei aktiviertem Hintergrunddienst wird in der Benachrichtigungsleiste eine Benachrichtigung über das aktive Profil mit Zusatzinformationen angezeigt. Am linken Rand wird dazu ein Kreis-Symbol visualisiert (siehe Abbildung 6.3c). Die Farbe des Kreises repräsentiert dabei die ID des aktiven Profils.

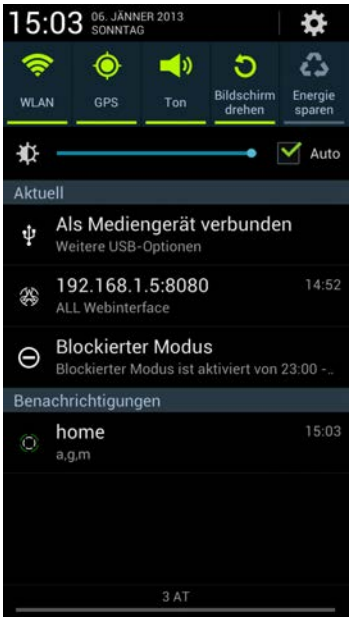
Entsprechend der Anordnung in der LLDL-Deklaration werden eine fortlaufende ID und eine korrespondierende Farbe der einzelnen Profile vergeben (siehe Abbildung 6.4c). Ein zwei-



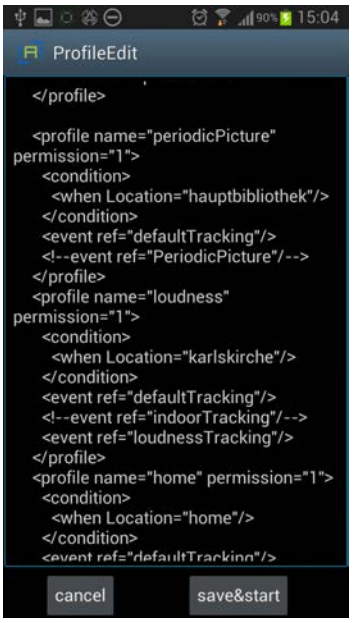
(a) Grundlegenden Einstellung von ALL.



(b) Hauptmenü von ALL zum Starten und Beenden der Hintergrunddienste.

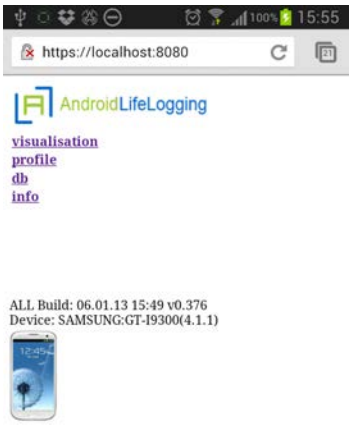


(c) Benachrichtigung über das aktive Profil.



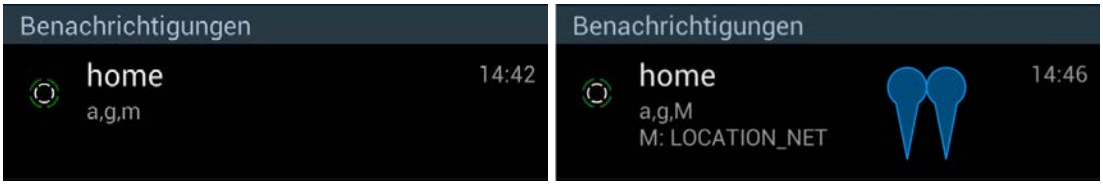
(d) Deklaration der LLDL in der Smartphone GUI.

(e) Aktivierung eines anderen Profils.



(f) Web-GUI des Webinterfaces am Smartphone.

Abbildung 6.3: Android Lifelogging GUI Hauptinteraktionen.



(a) Benachrichtigung des aktiven Profils. (b) Detailansicht durch Fingergeste nach unten.

(c) Farb-Ringe der Profil IDs 1 bis 10.

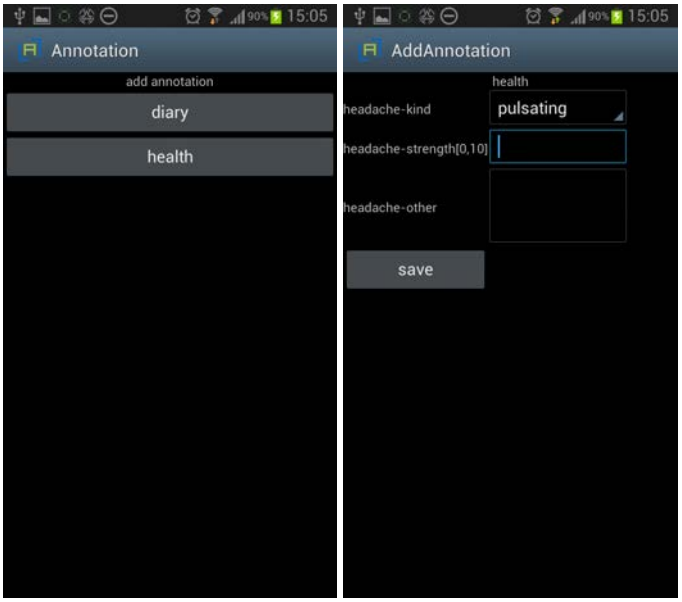
Abbildung 6.4: Benachrichtigung von Android Lifelogging.

ter weißer, kleinerer Kreis befindet sich im inneren des Farb-Kreises. Dieser visualisiert eine aktive Datenerfassung, indem sich der weiße Kreis dreht. Rechts davon wird die Uhrzeit der letzten Aktualisierung angezeigt. In einer größeren Schriftart wird zudem der Name des aktuellen Profils und darunter anhand von Buchstaben die verwendeten Sensoren (siehe Abbildung 6.4a) angezeigt. In Tabelle 6.1 sind sämtliche Buchstaben und zugehörigen Sensoren ersichtlich.

Tabelle 6.1: Verwendete Sensoren und zugehörige Buchstaben.

Buchstabe	Sensor
a	CALL
b	SMS
c	MAKE_VIDEO
d	MAKE_PICTURE
e	ACCELERATION
f	AUDIO
g	AUDIO_CALL
h	BAROMETER
i	BRIGHTNESS
j	CAMERA_FRONT
k	CAMERA_BACK
l	LOCATION_GPS
m	LOCATION_NET
n	GYROSCOPE
o	LOUDNESS
p	ROTATION

Sobald ein Sensor aktiv ist, wird der entsprechende Kleinbuchstabe zu einem Großbuchstaben. Bei Smartphones ab Android 4.1 kann zusätzlich eine Legende der aktiven Sensoren mithilfe



(a) Auswahl einer Anmerkung. (b) Eintragung einer Anmerkung.

Abbildung 6.5: Auswahl und Erstellung einer Anmerkung.

einer zwei Fingergeste nach unten angezeigt werden (siehe Abbildung 6.4b). Über *EditProfile* kann mithilfe der LLDL die Konfiguration bearbeitet werden (siehe Abbildung 6.3d).

Um ein manuelles Profil zu aktivieren, muss zuerst in der Benachrichtigungsleiste (siehe Abbildung 6.3a) auf die Profil-Benachrichtigung gedrückt werden. Dadurch gelangt man zur *Profile Switch*-Auswahl (siehe Abbildung 6.3c). Dort kann das gewünschte Profil durch Berührung aktiviert werden. Die Erstellung einer Anmerkung, der Aufruf von *ALL Main* oder *EditProfile* sind ebenfalls möglich. Zusätzlich zur visuellen Kreisrepräsentation des Profiles wird bei jedem Profil-Wechsel ein haptisches Feedback mit dem Vibrationsmotor in Form eines Morsecodes signalisiert.

Zur Erstellung einer Anmerkung muss zunächst über die Benachrichtigungsleiste *Profile Switch* aufgerufen werden. Dort wird durch Berührung von *Annotation* die Auswahl der Anmerkungen angezeigt (siehe Abbildung 6.5a). Folgend kann die Anmerkung entsprechend der LLDL hinzugefügt werden (siehe Abbildung 6.5b). Über den Einstiegspunkt *ALL Main* (siehe Abbildung 6.3b) kann das Webinterface über *öffne Webinterface* geöffnet werden (siehe Abbildung 6.3f).

6.2 Webinterface

Zur Visualisierung des aktivierten Webservices wird in der Benachrichtigungsleiste eine Benachrichtigung angezeigt. Dazu wird ein Symbol bestehend aus drei kreisförmig angeordneten „W“ visualisiert (siehe Abbildung 6.6). Ebenfalls werden die IP-Adresse und der zugehörige

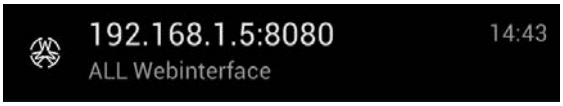


Abbildung 6.6: Benachrichtigung über aktiviertes Webinterface und URL.

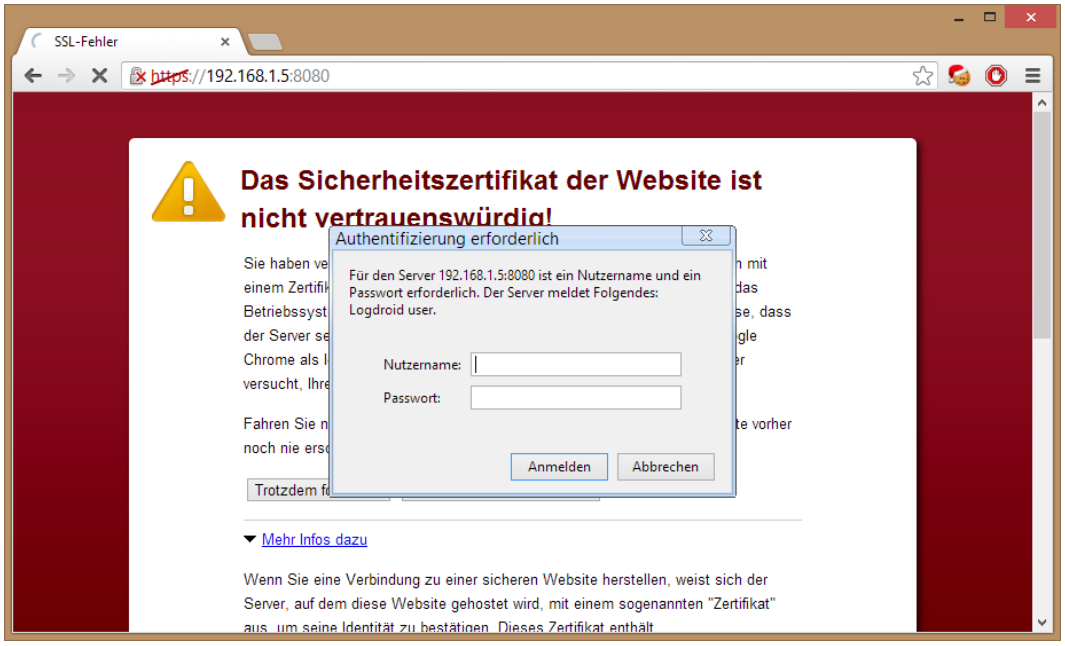


Abbildung 6.7: Webinterface Passwortschutz und Sicherheitswarnung.

Port des Webinterfaces angezeigt. Um mithilfe des Browsers auf das Webinterface zugreifen zu können, muss sich der Computer im gleichen Subnetz befinden. Beispielsweise indem das Smartphone und der Computer mit dem gleichen WLAN-Router verbunden sind. Als Browser wurde *Google Chrome*[6] verwendet. Im Browser ist die URL bestehend aus dem Protokoll *HTTPS*, IP-Adresse und Port einzugeben: *https://{IP}:{Port}*. Dementsprechend entspricht die Adresse aus Abbildung 6.6: *https://192.168.1.5:8080*

Anfangs wird im Browser eine Warnung angezeigt, dass das Zertifikat nicht vertrauenswürdig sei (siehe Abbildung 6.7). Der Grund dafür ist, dass dieses selbst signiert wurde (siehe Kapitel A.2, Seite 126). Im Folgenden muss durch *Trotzdem fortfahren* das Zertifikat als vertrauenswürdig bestätigt werden. Danach ist eine Authentifizierung mithilfe von Benutzername und Passwort erforderlich, welche in der Android GUI *Einstellungen* (siehe Abbildung 6.3a) konfigurierbar sind.

Startseite des Webinterfaces

Auf der Startseite des Webinterfaces befindet sich links immer die Navigationsleiste mit der

[6]Google Chrome Version 24.0.1312.57 m.

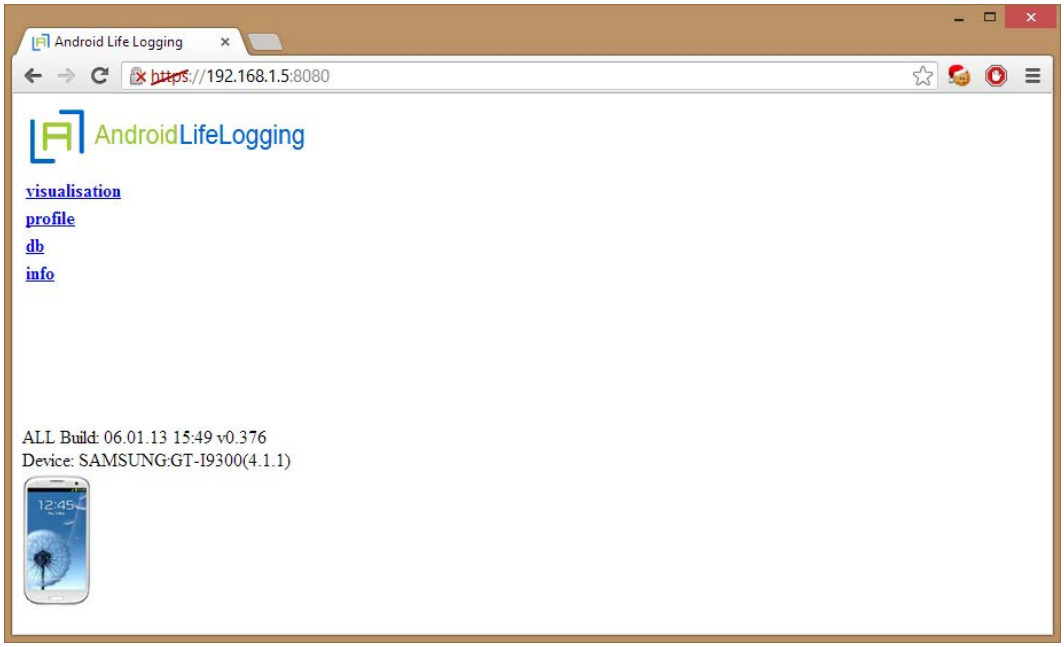


Abbildung 6.8: Startseite des Webinterfaces mit Navigation.

ALL Versions-Information und einer Abbildung des Smartphones. Wobei die Navigation in verschiedene Teile gegliedert ist:

visualisation	Verschiedene Arten der Visualisierung.
profile	Konfiguration mithilfe der LLDL.
db	Betrachtung, Durchsuchen und Datenaustausch der Datenbank.
info	Informationen über auftretende Fehler und Stabilität des Systems.

Profil
Wie zuvor in der Android-GUI ist die Konfiguration von ALL ebenfalls über das Webinterface möglich. Zur besseren Visualisierung werden dabei die jeweiligen XML-Tags eingefärbt (siehe Abbildung 6.9). Strukturelle Elemente von ALL werden zudem in der korrespondierenden Farbe (siehe Kapitel 5.6, Seite 58) hinterlegt.

Analyze
Zur besseren Übersicht bietet die Ansicht *analyze* eine abstrahierte Version der XML Deklaration. Dabei können drei verschiedene Detailgrade betrachtet werden (siehe Abbildung 6.10).

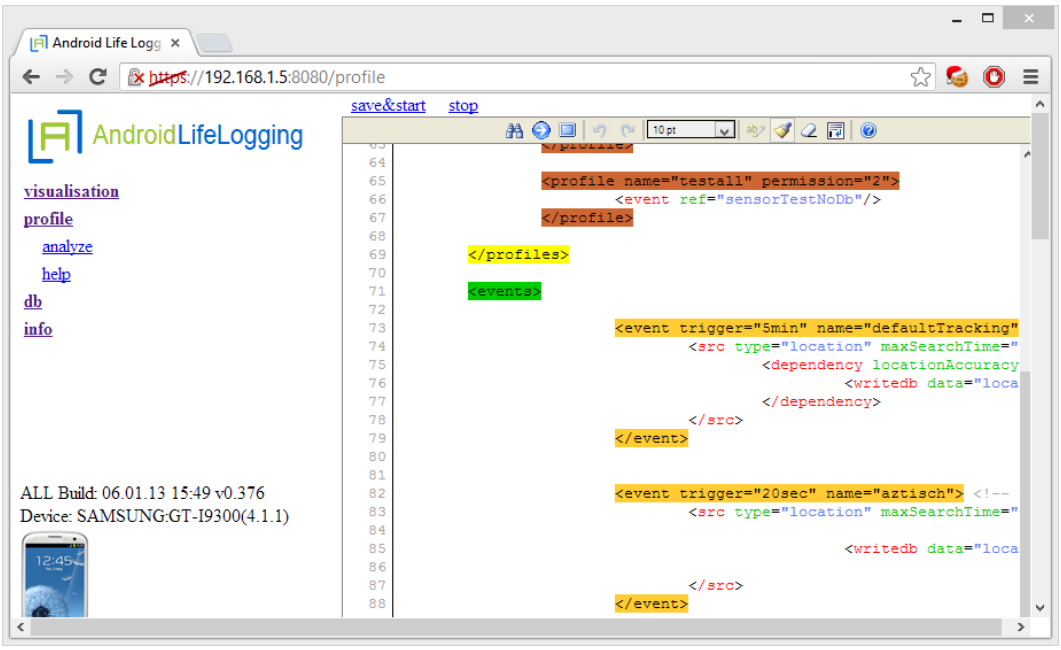
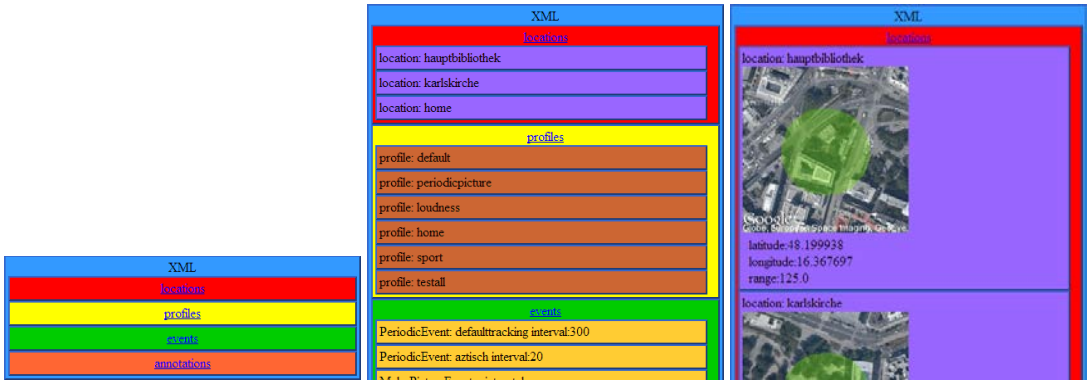


Abbildung 6.9: Farbige Hervorhebung der LLDL-Deklaration.



(a) *detail:1* (b) *detail:2* (c) *detail:3*

Abbildung 6.10: Detailgrade der LLDL-Abstraktion.

6.3 Datenbankschnittstelle

Die SQLite Datenbank bietet eine dynamische Abfrage der erfassten Daten. Da kein direkter Zugriff auf die Datenbank möglich ist, wurde ein webbasierter Datenbankmanager in Java realisiert. Dabei werden Abfragen, Import und Export sämtlicher Daten unterstützt.

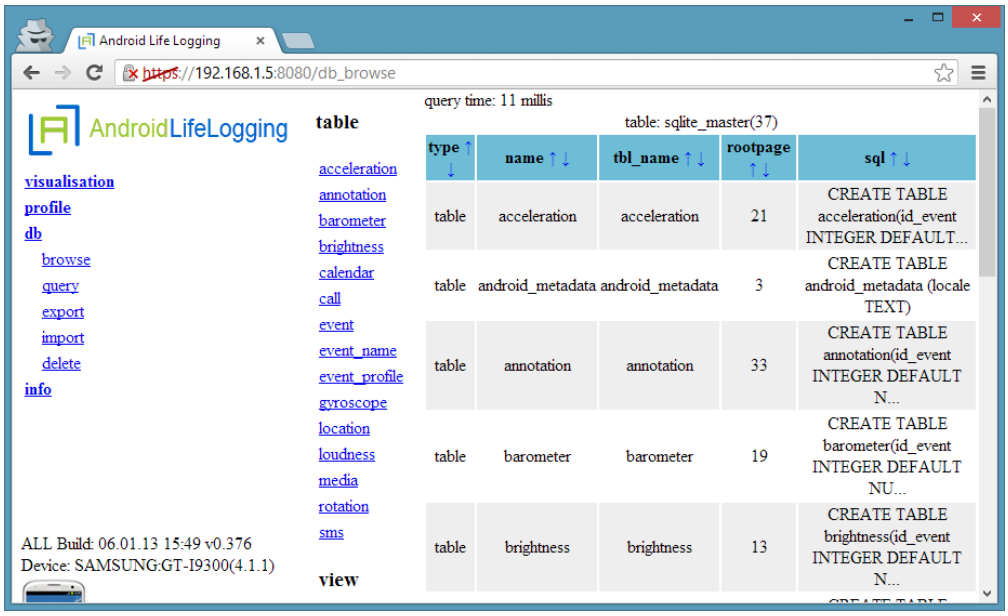


Abbildung 6.11: Web-basierter Datenbankbrowser mit Übersicht der Tabellen.

Auf der *browse*-Seite können sämtliche SQL-Tabellen und deren Inhalt betrachtet (siehe Abbildung 6.12) und nach ID sortiert werden (siehe Abbildung 6.11).

Mithilfe der *query* Seite erfolgt eine spezifische Abfrage (siehe Abbildung 6.13). Zusätzlich zur SQLite Abfrage wurde eine grafische Visualisierungsmöglichkeit hinzugefügt, die mit folgenden Parametern spezifiziert wird:

[draw:line(min)] Visualisiert einen Linien-Plot (siehe Abbildung 6.14).

[calcdist] Entfernung zu definierten Orten (siehe Abbildung 6.15).

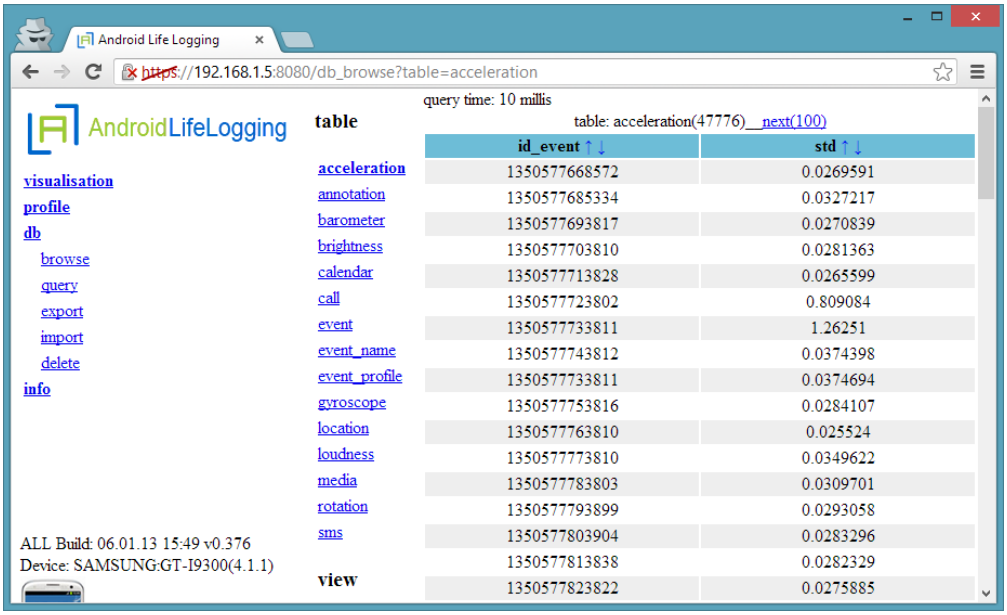


Abbildung 6.12: Datenbank Rohdaten der Tabelle acceleration.

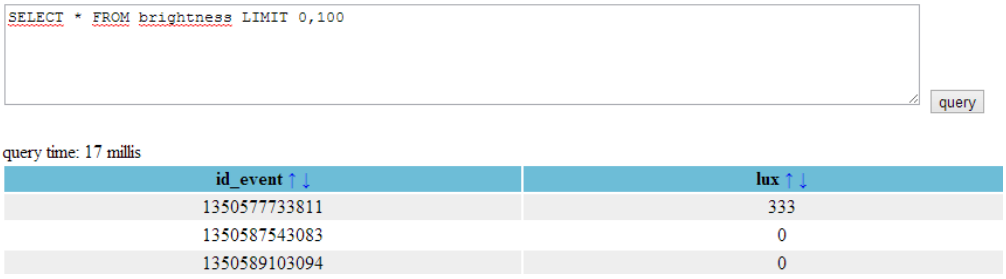


Abbildung 6.13: SQL-Abfrage über das Webinterface.

Auf der *export*-Seite können die gesammelten Daten in verschiedene Dateiformate exportiert werden. Folgende Export-Optionen stehen zur Verfügung:

all Vollständiger 256 Bit verschlüsselter Export[7] inklusive Medien und Profilen.

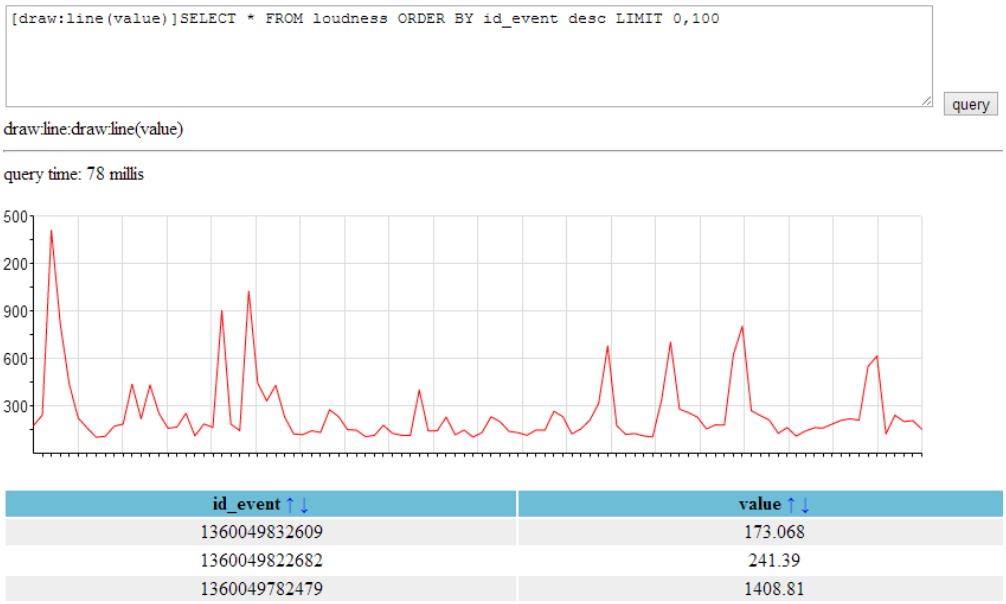
gpx Unverschlüsselter Export von Geodaten im Gpx-Dateiformat.

kml Unverschlüsselter Export von Geodaten im Kml-Dateiformat.

sql Unverschlüsselte Kopie der aktuellen SQLite Datenbank (Datenbankdump).

Durch *import* kann eine Sicherung im *all* Dateiformat importiert werden. Dabei können der

[7] Der Export erfolgt im Zip Datencontainer und als Dateiendung All.



id_event ↑↓	value ↑↓
1360049832609	173.068
1360049822682	241.39
1360049782479	1408.81

Abbildung 6.14: SQL-Abfrage mit Linienplot.

dist_hauptbibliothek	dist_karlskirche	dist_home	altitude ↑↓	speed ↑↓	accuracy ↑↓	id_name ↑↓	id_profile ↑↓	permission ↑↓
8998,1	9163,2	7,6	-1	-1	37.908	110616	107431	1
8999,2	9164,3	6,5	-1	-1	36.9945	110616	107431	1
9001,0	9166,1	4,8	-1	-1	35.9165	110616	107431	1
8993,2	9158,3	13,5	-1	-1	38.954	110616	107431	1
8994,5	9159,6	11,9	-1	-1	42.44	110616	107431	1

Abbildung 6.15: SQL-Abfrage mit Berechnung der Entfernung zu den definierten Orten.

Speicherort von Medien-Dateien und optional ein Passwort angegeben werden (siehe Abbildung 6.16). Das Passwort setzt sich dabei aus *username:passwort* der Android GUI Konfiguration zusammen. Falls kein Passwort angegeben ist, wird die aus der ALL spezifizierte Kombination aus Benutzername und Passwort verwendet.

Auf der Seite *delete* kann die komplette SQLite Datenbank gelöscht werden. Die Medien-Dateien bleiben aber erhalten und können separat über einen Dateibrowser entfernt werden.

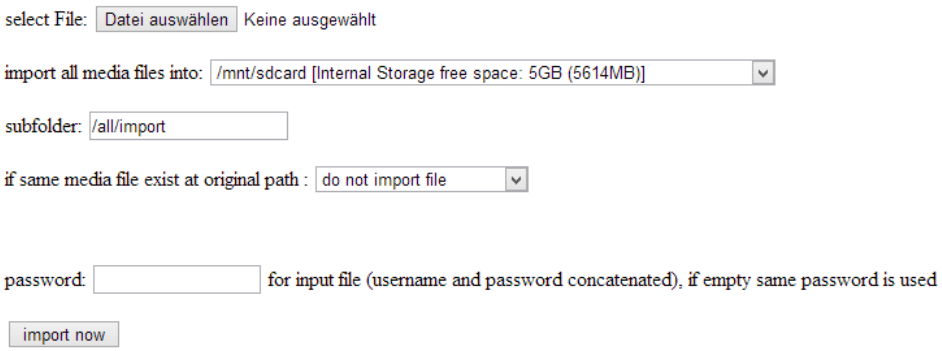


Abbildung 6.16: Import einer Sicherung im All Dateiformat.

6.4 Visualisierung von Lifelogging Daten

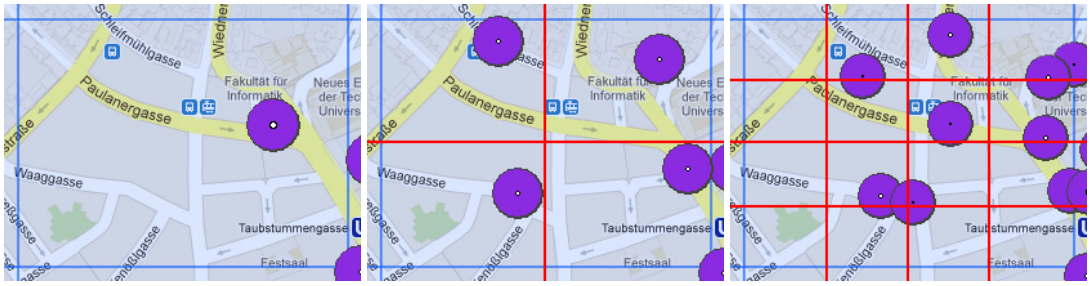
Eine beliebige Visualisierung der erfassten Daten kann, wie bereits erwähnt, als Linienplot des erweiterten Befehlssatz erstellt werden. Für die Veranschaulichung von Bildern und Positionen wurden zusätzliche Visualisierungsmöglichkeiten geschaffen. Im Folgenden werden die Ansichten der Landkarte und der Zeitraffer-Ansicht vorgestellt.

6.4.1 Landkarte

Diese Visualisierungsart zeigt die erfassten Daten auf einer Landkarte. Die Bedingung ist, dass die zu visualisierenden Daten mindestens einer Position zugehören. Zur Visualisierung der Positionsdaten wird die API von *Google Maps* verwendet, wobei ein zusätzlicher ALL-Layer (siehe Abbildung 6.19a) eine Veränderung der Darstellungsoptionen ermöglicht. Wird eine große Anzahl von Positionsdaten visualisiert, entstehen Probleme. Die Sichtbarkeit wird aufgrund der Datenmenge beeinträchtigt und die Darstellung ist rechenintensiv. Deswegen ist eine Filterung der Daten notwendig.

$subdivision$**:** Zur Reduzierung der Datenmenge erfolgt über einen veränderbaren Detailheitsgrad ein automatisches Clustering der Positionsdaten. Dies wird durch das Sampling von geographischer Breite und geographischer Länge realisiert. Bei einer niedrigen Auflösung werden wenige und bei einer hohen Auflösung viele Daten angezeigt. Der Parameter *subdivision* (siehe Abbildung 6.17) spezifiziert dabei den Detailheitsgrad. Im ALL-Layer der Landkarte ist der *subdivision* Parameter veränderbar. Durch das Sampling der Positionen resultieren rechteckige Bereiche[8] auf der Google Maps Landkarte. Die Anzahl der Rechtecke n ergibt sich durch die Formel $n = 2^{subdivision}$. Über die Option *show tiles* im ALL-Layer (siehe Abbildung 6.19a) werden die Rechtecke (siehe Abbildung 6.20b) mit dem Detailheitsgrad *subdivision:0* angezeigt. Für jedes Rechteck wird der Mittelwert der darin befindlichen Positionen berechnet und als Datenkreis visualisiert.

[8]Ein rechteckiger Bereich in Google Maps entspricht aufgrund der nichtlinearen Abbildung meist keinem Rechteck auf der Erdoberfläche.



(a) *subdivision*:0 (b) *subdivision*:1 (c) *subdivision*:2

Abbildung 6.17: Visualisierung unterschiedlicher Detailheitsgrade.

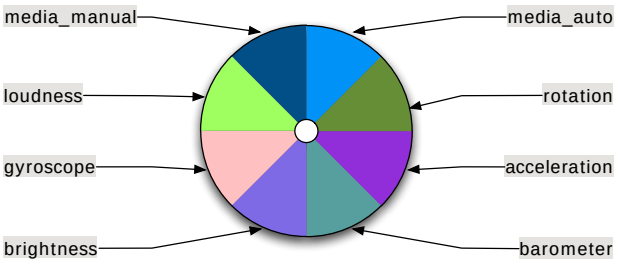
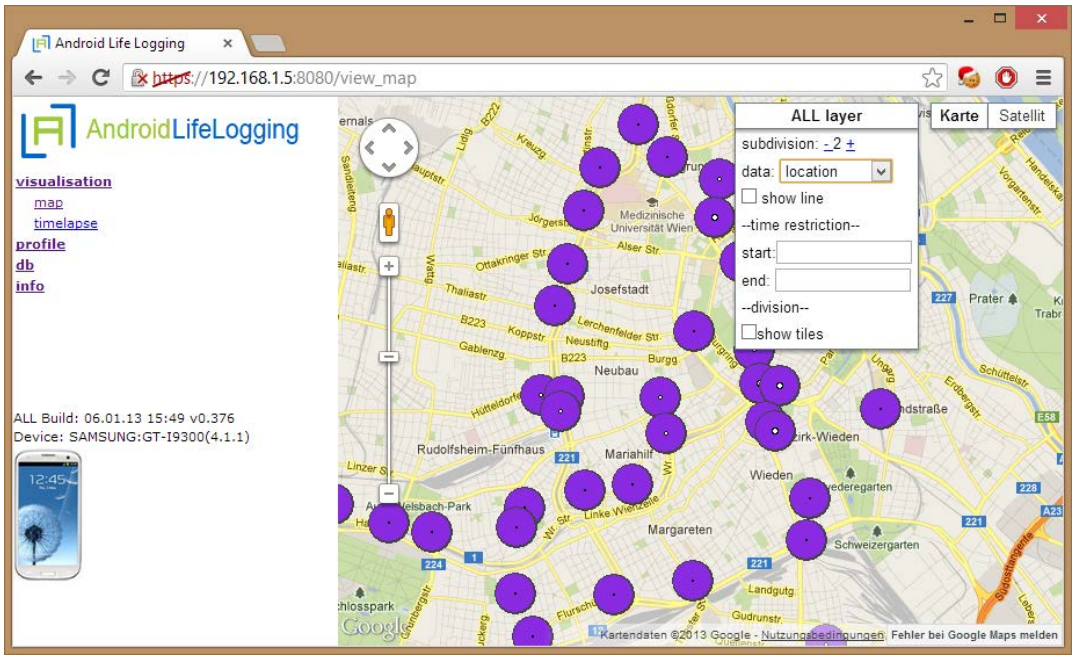


Abbildung 6.18: Farben des Datenkreises mit zugehörigen Datentypen.

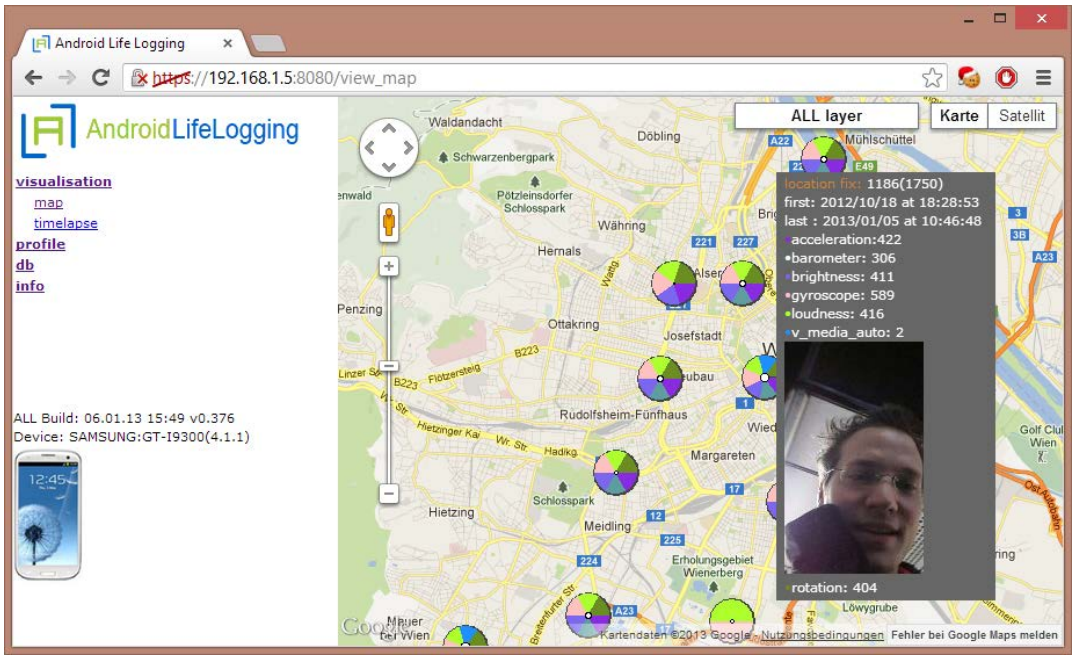
data: Jeder Datenkreis kann verschiedene Datentypen beinhalten. Je nach Anzahl der verfügbaren Datentypen wird der Datenkreis (siehe Abbildung 6.18) in regelmäßige Kreissektoren unterteilt und eingefärbt. Dabei repräsentieren die Farben im Datenkreis die Datentypen. Durch einen Mausklick oder Mouse-Over werden Zeit und Datum der Datenerfassung wie auch Anzahl und Sensortyp angezeigt (siehe Abbildung 6.19b). Im inneren des Datenkreises befindet sich ein weißer Kreis dessen Radius die Anzahl der zusammengefügten Rohdaten repräsentiert. Über die Auswahl *data* im ALL-Layer ist eine Filterung nach Datentyp möglich. Dazu sind in Tabelle 6.2 die verfügbaren Datenquellen und zugehörige Datentypen ersichtlich.

Tabelle 6.2: Auswahl der Datenquelle und korrespondierende Datentypen.

Datenquelle	visualisierte Datentypen
location	Positionsdaten
acceleration	Beschleunigungsdaten
barometer	Luftdruck
brightness	Helligkeit
loudness	Lautstärke
wiw	alle verfügbaren

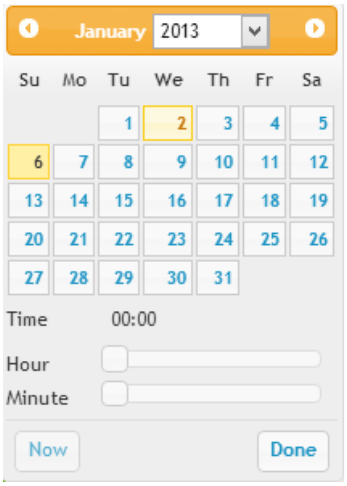


(a) Visualisierung der Positionen auf der Landkarte.

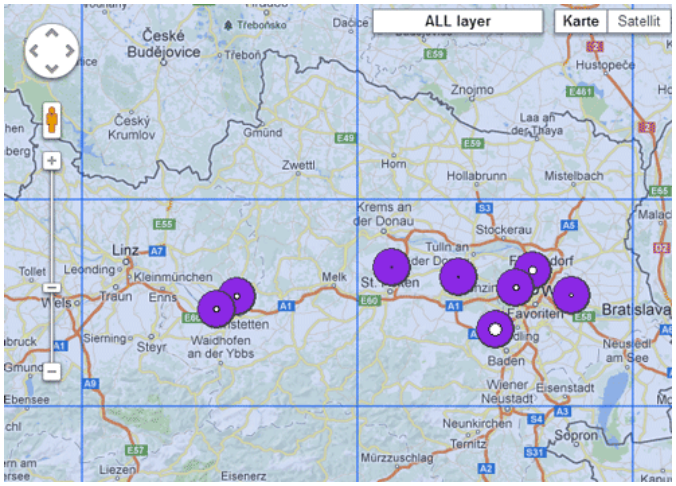


(b) Detailinformationen des Datenkreises.

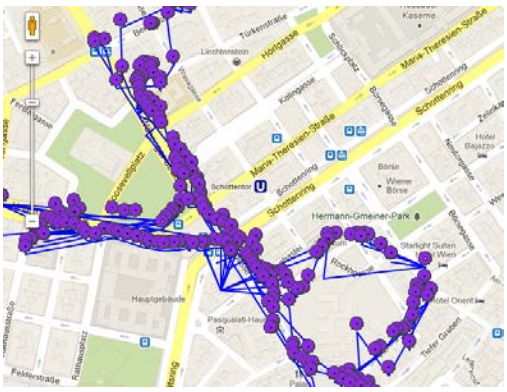
Abbildung 6.19: Visualisierung der Daten auf einer Landkarte.



(a) Zeitliche Einschränkung der Daten.



(b) Unterteilung der Erdoberfläche.



(c) Weg, der durch eine Linie dargestellt wird.

Abbildung 6.20: Landkarten Zeitbegrenzung, Clustering und Weg-Darstellung.

show line: Die Option *show line* im ALL-Layer ermöglicht eine zusätzliche Visualisierung der Positionsdaten als Weg. Hierbei erfolgt kein Clustering der Positionsdaten und somit werden die Ausreißer der Rohdaten sichtbar.

time restriction: Eine zeitliche Einschränkung ist über die Felder *start / end* im ALL-Layer möglich (siehe Abbildung 6.20a). Dabei können entweder textuell oder über die GUI das Datum und die Uhrzeit gewählt werden.

Die Datenkreise können auch in der Street View Ansicht (siehe Abbildung 6.21) von Google Maps visualisiert werden. Zum Beispiel ist es dadurch möglich, besuchte Orte mit den aufgenommenen Bildern zu vergleichen, oder einen Ort zu betrachten, wie er früher ausgesehen hat. Auf der *export*-Seite des Webinterfaces kann durch den Kml- oder Gpx-Export eine Google

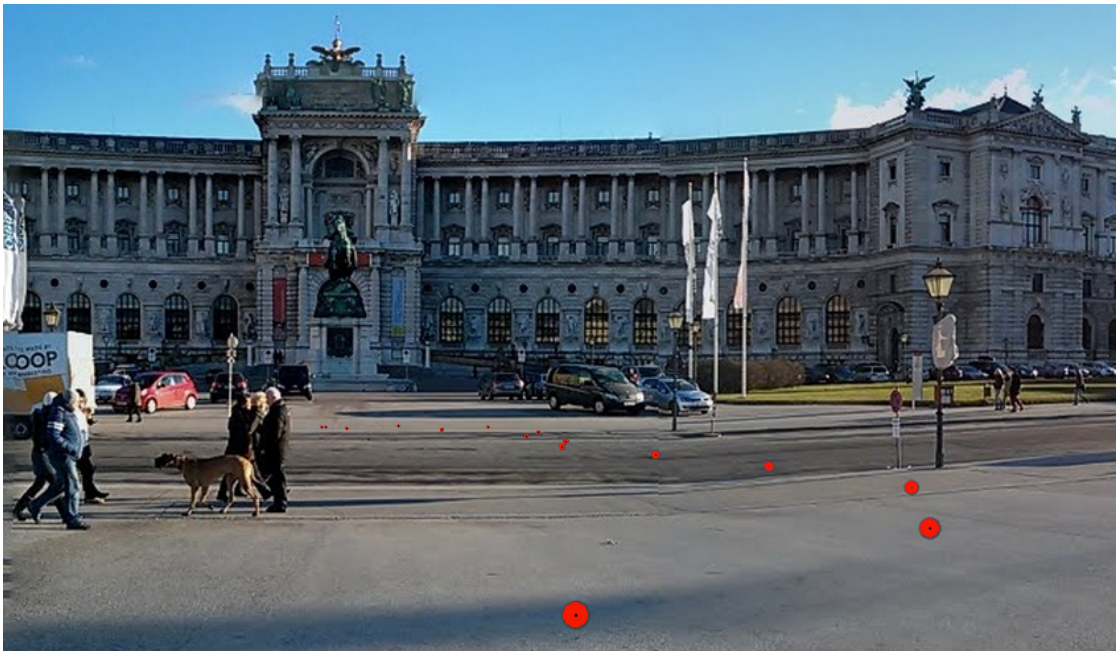
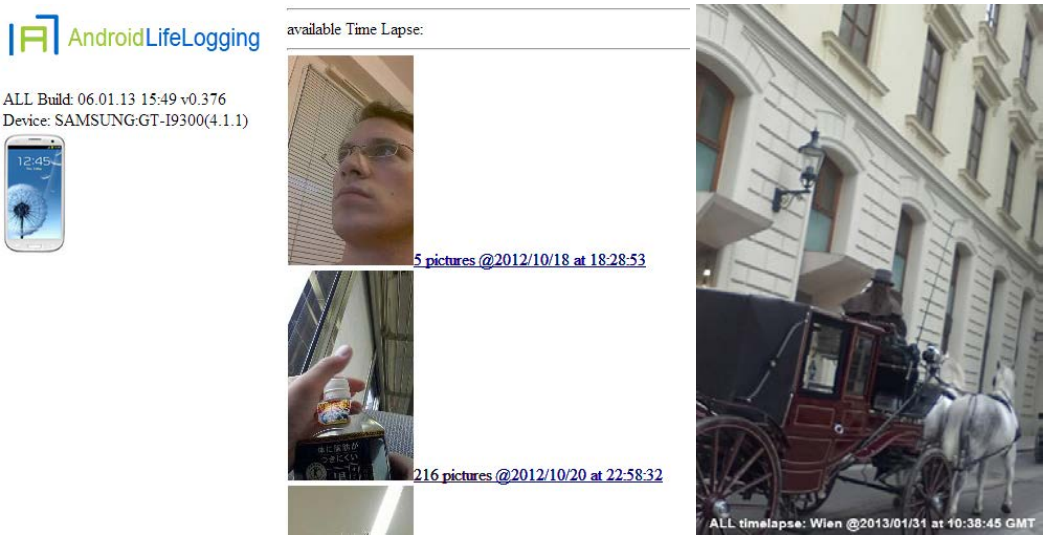


Abbildung 6.21: Street View Ansicht mit Datenkreisen die besuchte Positionen visualisieren.

Abbildung 6.22: Visualisierung eines Weges mit Google Earth.



(a) Übersicht zeitlich aufeinanderfolgender Bilder. (b) Zeitraffer der einzelnen Bilder.

Abbildung 6.23: Zeitraffer Übersicht von ALL.

Earth kompatible Datei erzeugt werden. Durch das Importieren der Datei in Google Earth (siehe Abbildung 6.22) können die besuchten Orte mit den verfügbaren 3D-Modellen betrachtet werden. Ebenso kann eine Animation abgespielt werden, wobei im Flugmodus die besuchten Orte abgeflogen werden.

6.4.2 Zeitraffer

Die zweite Visualisierungsart zeigt hintereinander aufgenommene Bilder als Zeitrafferaufnahme. Dabei werden die Bilder fortlaufend überblendet. Zuerst wird eine Übersicht zusammengehöriger Bilder mit der Bilderanzahl und Datum aufgelistet (siehe Abbildung 6.23a). Nach der Auswahl wird der eigentliche Zeitraffer angezeigt (siehe Abbildung 6.23b). Dabei werden im rechten unteren Eck der Ort und das Datum angezeigt. Ein Videobeispiel ist auf Youtube verfügbar[9] und zeigt einen Selbstversuch zur visuellen Erfassung eines Nachmittags in Wien.

6.5 Datenfilterung

Selbstversuche mit ALL zeigen, dass die automatische Erfassung der Kamera eine große Anzahl von Bildern erzeugt. Aus diesem Grund wird eine Methode zur Filterung redundanter Bilder vorgestellt. Ebenso wird die Möglichkeit der Anonymisierung von Gesichtern gezeigt.

[9]Beispiel eines Bild-Zeitraffer: `http://www.youtube.com/watch?v=V642P9OI6F4`

6.5.1 Filterung redundanter Bilder

Um Speicherplatz zu sparen, kann es sinnvoll sein, redundante Bilder aus einer Zeitraffer Aufnahme zu löschen. Dazu wurde ein einfacher Ansatz getestet, der jeweils drei aufeinanderfolgende Bilder untersucht. Falls das mittlere Bild dem ersten und letzten entsprechend ähnlich ist, wird dieses entfernt (siehe Abbildung 6.24). Als Ähnlichkeitsmaß wird Histogramm-Intersection (Formel 6.1) verwendet [119]. Die einzelnen Bilder werden dazu zuerst auf eine Bildgröße von 32x32 Pixel skaliert und in den *NRGB* Farbraum transformiert [120]. Der Grund dafür ist, dass dadurch die Belichtung der Bilder keinen großen Einfluss hat. Die Histogramme werden dann erzeugt, indem die Pixel in je acht Klassen (bins) pro Farbkanal eingeteilt werden. Bei dieser Methode resultieren jeweils zwei Bilder, die den Anfang und das Ende einer ähnlichen Bildreihe repräsentieren.

Abbildung 6.24: Entfernung redundanter Bilder mithilfe der Histogramm-Intersection.

$$K_{int}(A, B) = \sum_{i=1}^{m} min\{a_i, b_i\} \tag{6.1}$$

K_{int}: Histogramm-Intersection.
A: Histogramm des ersten Bildes.
B: Histogramm des zweiten Bildes.
a_i: Elemente aus dem Histogramm A.
b_i: Elemente aus dem Histogramm B.

6.5.2 Anonymisierung

Zur Anonymisierung der Gesichter kann eine zusätzliche Option *priv* verwendet werden. Dabei werden zuerst mithilfe der Android API-Gesichtserkennung die Gesichter lokalisiert und danach mit einer grünen Box *priv=1* umrandet oder mit *priv=2* entfernt (siehe Abbildung 6.25). Die verfügbare Android API Gesichtserkennung kann jedoch nicht konfiguriert werden, dementsprechend ist sie nicht optimal geeignet.

(a) Bild ohne Gesichter. (b) Bild mit Gesichtern und falsch erkanntem Gesicht. (c) Entfernung von Gesichtern.

Abbildung 6.25: Zeitraffer-Bilder mit Gesichtserkennung für Anonymisierung.

6.6 Auswertung der Datenqualität

Im Zuge der Arbeit wurde festgestellt, dass es Probleme mit der Lokalisierung gibt. Vor allem auftretende Ausreißer verfälschen die Koordinaten der Messergebnisse. Deswegen wird folgend die Genauigkeit der Lokalisierung untersucht, zusätzlich werden Möglichkeiten erarbeitet, um diese Ausreißer zu filtern. Weiters wird der erfasste Luftdruck betrachtet, um zu sehen, ob eine schnelle Änderung erkannt wird.

6.6.1 Genauigkeit der Lokalisierung

Für die Ermittlung der Positionsdaten wurden mehrere Experimente an einem Ort durchgeführt. Dabei konnten die statische und die mobile Erfassung der Positionsdaten mithilfe unterschiedlicher Verfahren getestet werden. Die Datenerfassung wurde mit dem Samsung GALAXY S III I9300 und der ALL Anwendung durchgeführt. Die folgenden Abbildungen der Experimente sind jeweils so verzerrt, dass sie annähernd einem rechteckigen Bereich auf der Erdoberfläche entsprechen.

Experiment 1: Im Innenraum einer Dachgeschosswohnung wurden mit der kombinierten WLAN- und Sendemastenlokalisierung alle zehn Sekunden Positionsdaten ermittelt. Die Wohnung befindet sich in einem Wohnkomplex, der eine Vielzahl (> 7) in Reichweite befindlicher

WLAN-Netzwerke aufweist. Über einen Zeitraum von 50 Minuten konnten somit 300 Positionsdaten erfasst werden.

Das Ziel des ersten Experiments war die Verteilung der gemessenen Positionsdaten, die mit der kombinierten WLAN- und Sendemastenlokalisierung erfasst wurden, zu visualisieren. Dazu wurden die Eigenwerte und die Eigenvektoren der Positionsdaten, die die Richtung der größten Datenvarianz angeben, berechnet. Da der Eigenwert in Richtung des Eigenvektors die Standardabweichung repräsentiert, wurden drei Ausgleichsellipse eingezeichnet (siehe Abbildung 6.26).

Unter der Annahme, dass die Daten normalverteilt sind, entsprechen diese Ellipsen der ein-, zwei-, und dreifachen Sigma-Umgebung der Normalverteilung. Dementsprechend befinden sich in der dreifachen Sigma-Umgebung $99,73\ \%$ aller Messwerte. Die Ausdehnung dieses Bereichs entspricht Horizontal (Geografische Länge) circa $2 * 10^{-4}\ Grad$ und Vertikal (Geografische Breite) $4 * 10^{-4}\ Grad$. Dies entspricht einer Abweichung von $\pm 11\ m$.

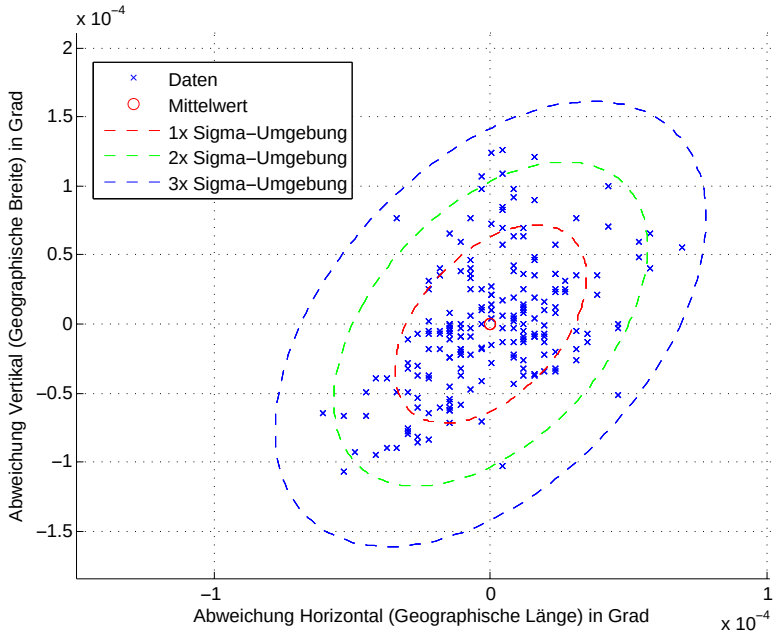


Abbildung 6.26: Verteilung der WLAN/Net-Rohdaten, Dachgeschoss (300 Messwerte).

Experiment 2: Diesmal fand die Datenerfassung mit freier Sicht zum Himmel unter Verwendung der GPS-Lokalisierung statt. Jede Sekunde wurde dabei, über einen Zeitraum von 17 Minuten, die Position erfasst.

Das Ziel des zweiten Experiments war es wieder die Verteilung zu visualisieren und mit dem ersten Experiment zu vergleichen. Ebenso wurde die Verteilung wie in Experiment 1 berechnet und die Sigma-Umgebungen als Ellipsen (siehe Abbildung 6.27) eingezeichnet. Die einzelnen Messwerte bilden scheinbar einen Pfad. Dies ist auf die im zivilen Bereich beabsichtigte Ungenauigkeit von GPS und Störungen der Signalausbreitung zurückzuführen [57]. Über eine länger Zeit wird auch hier eine Normalverteilung angenommen. Die genaue Verteilung ist Die Abwei-

chung von circa $\pm 22\ m$ ist etwa doppelt so groß wie in Experiment 1.

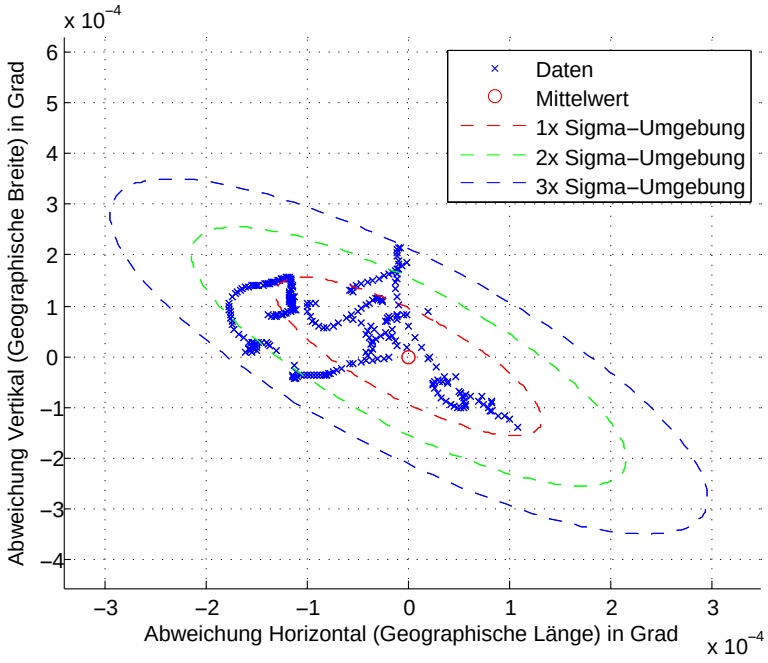


Abbildung 6.27: Verteilung der GPS-Rohdaten, freie Sicht zum Himmel (1000 Messwerte).

Experiment 3: Aufgrund der ermittelten Genauigkeit von Experiment 1 erfolgte eine Kombination von Messungen in zwei verschiedenen Räumen der Dachgeschosswohnung. In diesem dritten Experiment wurden jeweils 300 Messwerte in Raum 1 und in Raum 2 erfasst. Die Entfernung zwischen den Erfassungspositionen beträgt circa $10\ m$.

Ziel des Experiments war es, zu evaluieren, ob eine Lokalisierung beziehungsweise Trennung der Räume mit den Positionsdaten möglich ist. Mithilfe des k-Means-Algorithmus konnten zwei Clusterzentren der Messwerte ermittelt werden. Zusätzlich zu den Rohdaten der Messwerte wird in Abbildung 6.28 der Grundriss der Wohnung visualisiert. Weil die exakte Position der Wohnung nicht zur Verfügung stand, wurden die Rohdaten und der Grundriss zueinander ausgerichtet. Dabei wurde angenommen, dass die Mittelwerte der Clusterzentren und der Erfassungspositionen gleich sind.

Eine Trennung der Räume konnte mithilfe einer Messreihe ermittelt werden. Jedoch entspricht der Abstand zwischen den Clusterzentren von circa $20\ m$ nicht der Realität. Auch wurde eine Abänderung des Experiments getestet. Durch Verringerung des Abstands zwischen den Erfassungsorten auf $5\ m$ war eine Trennung der Räume aufgrund der Ungenauigkeit nicht mehr möglich.

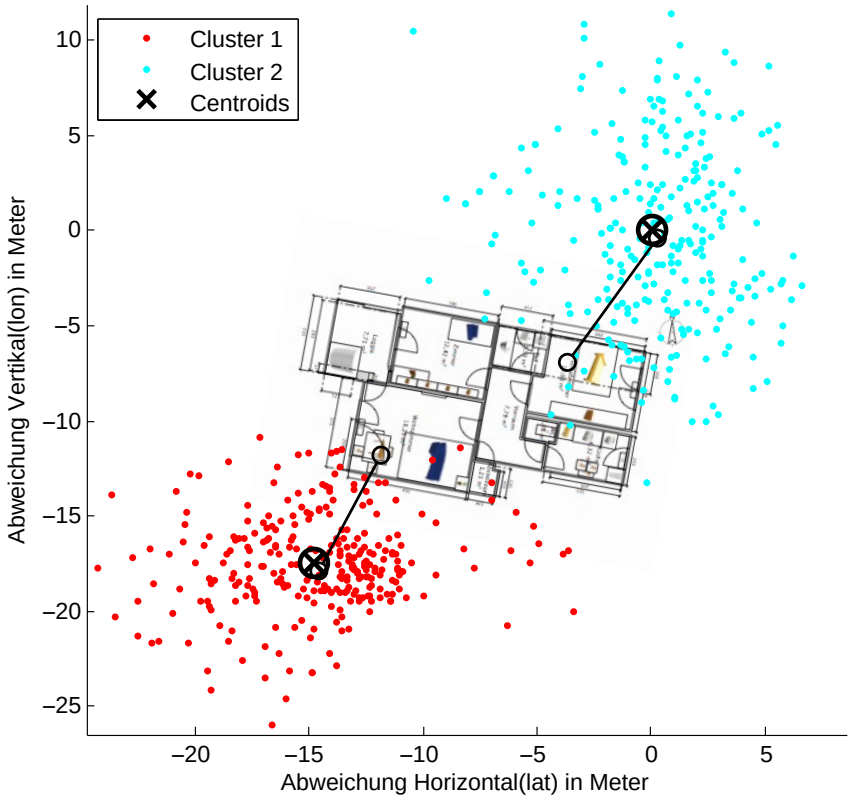
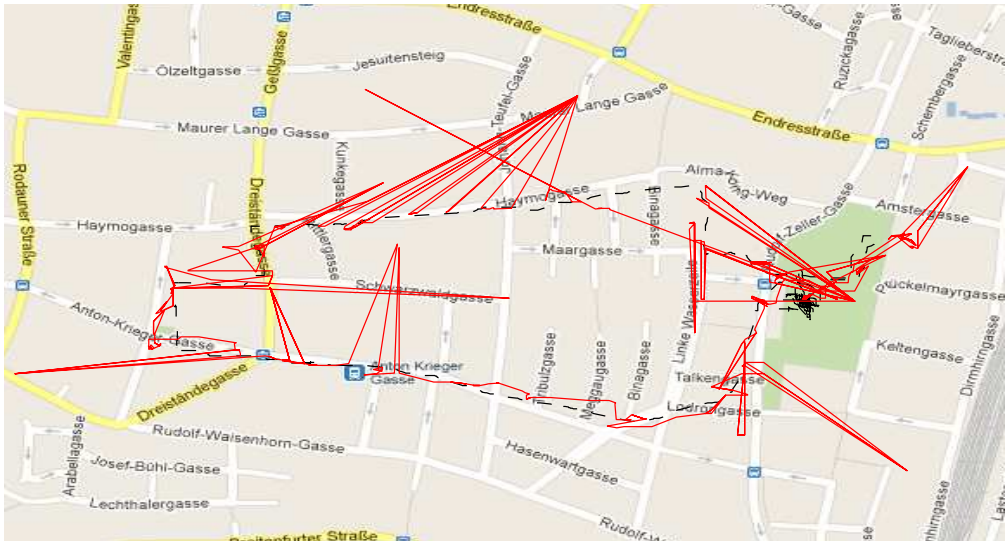


Abbildung 6.28: Visualisierung von zwei mithilfe des k-Means-Algorithmus ermittelten Clusterzentren (Kreise mit X), die jeweils mit den korrespondierenden Originalpositionen (Kreise ohne X) verbunden sind.

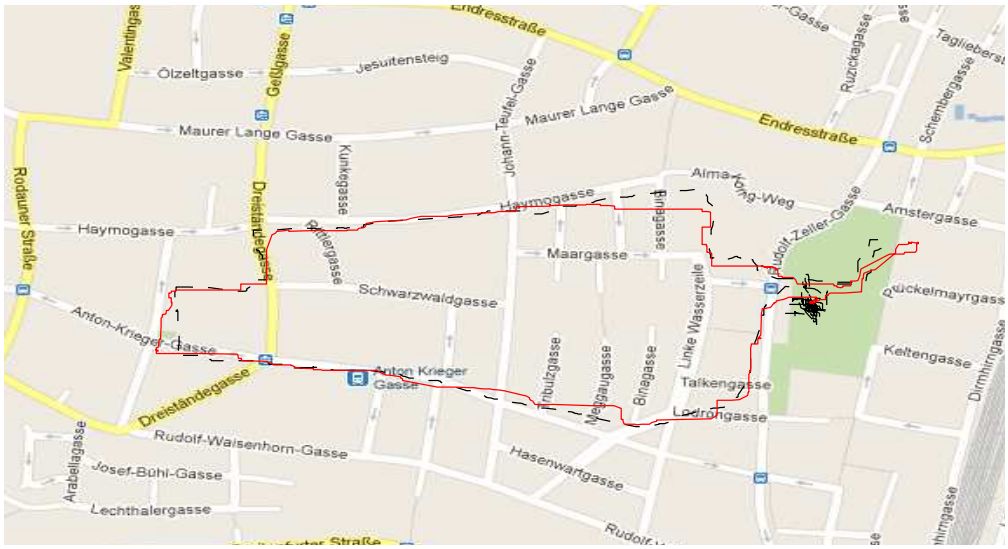
Experiment 4: Im nächsten Experiment wird die mobile Lokalisierung untersucht. Dazu wurde eine festgelegte Route abgegangen, und jede Sekunde die Position mit der kombinierten WLAN- und Sendemastenlokalisierung ermittelt.

Ein Medianfilter ermöglicht die Entfernung von Ausreißern. Das Ziel des Experiments war die Ermittlung einer empirischen Median-Filtergröße. Diese Filtergröße gibt die Anzahl der zur Filterung verwendeten Daten an. Ebenso sollten die Ausreißer der mobilen Erfassung visualisiert werden.

Für die Ermittlung einer entsprechenden Median-Filtergröße wurde der Zusammenhang zwischen Anzahl der Messwerte und Fehlerabweichung in Meter untersucht. Dazu wurde die Position auf einer fest vorgelegten Route erfasst (siehe Abbildung 6.29a). Es konnte empirisch ermittelt werden, dass ein Median Filter der Größe 33, wie in Abbildung 6.29b ersichtlich, die großen Ausreißer entfernt.



(a) Rohdaten der kombinierten WLAN/Net-Positionsdaten.



(b) Medianfilter der Größe 33.

Abbildung 6.29: Route mit WLAN/Net-Lokalisierung. Die strichlierte Linie repräsentiert den tatsächlichen Weg. Die durchgehende Linie entspricht den gemessenen Positionen.

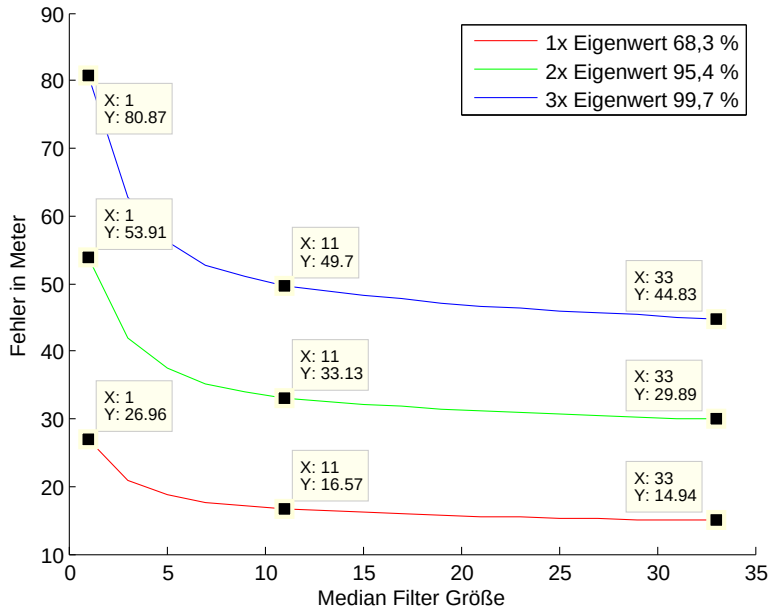


Abbildung 6.30: Zu erwartender Fehler entsprechend der Median-Filtergröße.

Resultate der Experimente Die Ergebnisse der Experimente zeigen, dass die Lokalisierung verschiedener Methoden unterschiedlich genau sind. Ungenauigkeiten und Ausreißer verfälschen dabei die Lifeloggingdaten, deshalb ist das Entfernen von Ausreißern notwendig. Ein Medianfilter ermöglicht dies.

Untersuchung der Median-Filtergröße Die LLDL bietet die Möglichkeit Positionsdaten mit einem Medianfilter zu filtern. Deshalb wurde eine statistisch Untersuchung der Filtergröße durchgeführt. Dazu wurden wie in Experiment 1 Positionsdaten mithilfe der WLAN- und Sendemastenlokalisierung im Dachgeschoss der Wohnung aufgezeichnet. Um genauere Werte zu erhalten wurden diesmal 1.000 Messwerte erfasst.

Das Ziel dieser Untersuchung ist die Ermittlung der Filtergröße im Zusammenhang der zu erwartenden Fehlerabweichung. Beispielsweise soll ermittelt werden, wie groß der zu erwartende Fehler in Meter ist, wenn ein Medianfilter der Größe 11 benutzt wird. Die Rohdaten wurden dazu in ansteigender Median-Filtergröße gefiltert, dies entspricht der X-Achse der Abbildung 6.30. Zur Ermittlung des Fehlers in Meter wurde der Mittelwert der Daten herangezogen. Die drei Linien repräsentieren die ein-, zwei-, und dreifache Sigma-Umgebung, das bedeutet bei der untersten Linie, dass $68,3\ \%$ der Messwerte unter Verwendung des Medianfilters einen entsprechenden Fehler in Meter aufweisen. Beispielsweise ist zu $68,3\ \%$ mit einer Abweichung von $49,7\ m$ zu rechnen wenn die Daten mit einem Medianfilter der Größe 11 gefiltert wurden.

Die Untersuchung zeigt, dass die Verwendung großer Medianfilter die Ergebnisse verbessert. Anfangs verringert sich der Fehler signifikant mit der Erhöhung der Filtergröße. Ab einer Filtergröße von circa 21 steigt die Genauigkeit nur noch geringfügig.

6.6.2 Luftdruck

In diesem Experiment wurden der Luftdruck und die Position im mobilen Einsatz aufgezeichnet. Aus den Positionen und dem gemessenen Luftdruck lässt sich der Zusammenhang zwischen Höhe und Luftdruck visualisieren. Dazu wurde mithilfe der Google Elevation API die Höhe der Erdoberfläche ermittelt [121]. Wie in Abbildung 6.31 ersichtlich, lässt sich anhand der barometrischen Formel 5.1 auf den zu erwartenden Luftdruck schließen. Die Messung wurde innerhalb von vier Stunden im Stadtbereich von Wien durchgeführt. Das Experiment zeigt beispielsweise Störungen in öffentlichen Verkehrsmitteln.

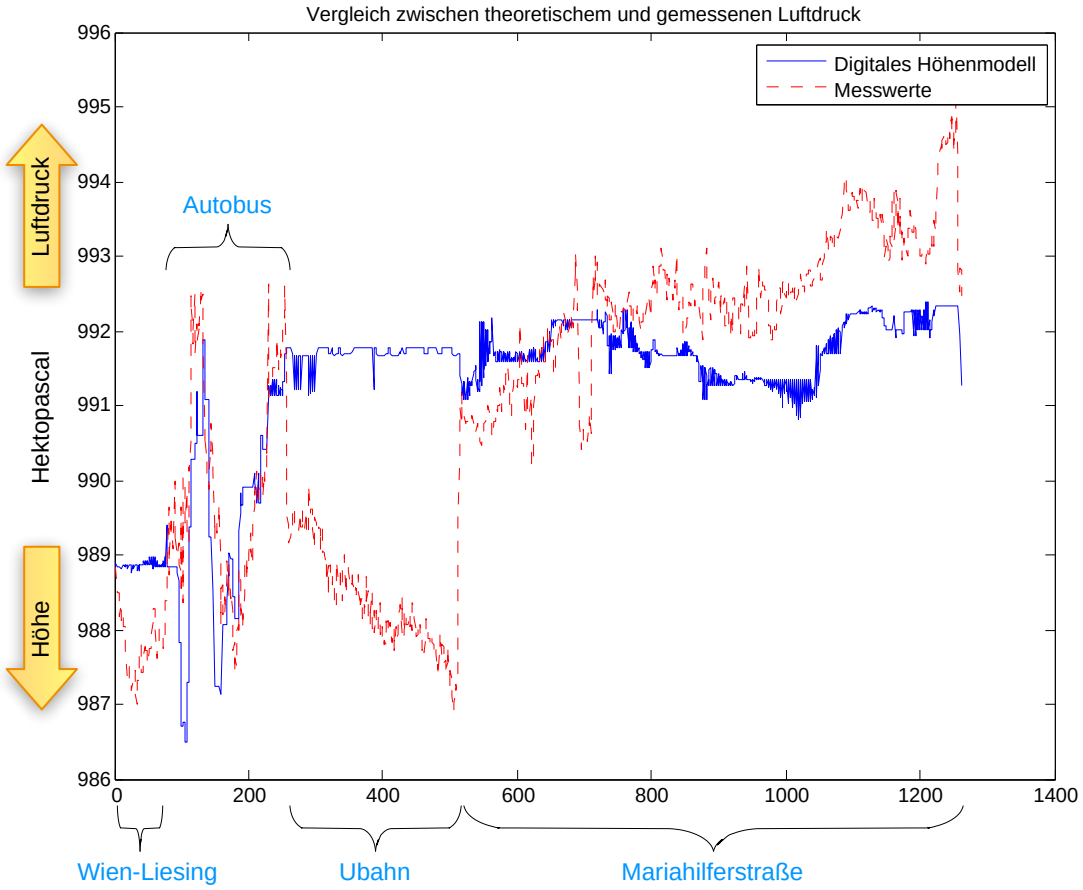


Abbildung 6.31: Vergleich zwischen dem zu erwartenden Luftdruck anhand der Höhe und den gemessenen Werten.

KAPITEL 7

Zusammenfassung

Diese Diplomarbeit beschäftigt sich mit mobilen Lifeloggingsystemen und der Realisierung einer Android basierten Lifeloggingarchitektur. Dabei wurden verschiedene Aspekte bestehender Ansätze und Möglichkeiten der Indizierung durch inhaltliche Analyse untersucht.

Es wurde ein flexibel konfigurierbares Lifeloggingarchitektur geschaffen, das mithilfe der LLDL umfassend konfigurierbar ist. Das orts- und zeitabhängige System ermöglicht eine energieoptimierte Datenerfassung. Sämtliche Sensoren, die am Smartphone verfügbar sind, können erfasst, visualisiert und ausgewertet werden. Das realisierte System kann beliebige Sensordaten beliebig konfigurierbar aufzeichnen. Verschiedene Visualisierungsarten ermöglichen es die erfassten Daten gefiltert anzuzeigen. Ebenso ist durch den Import und Export ein inkrementeller Datenaustausch möglich.

Die Android-Plattform bietet zuverlässige Möglichkeiten, Prozesse im Hintergrund laufen zu lassen, welche die Grundlage einer automatischen Datenerfassung darstellen. Ebenso können die Kamera, der Audiorekorder und sämtliche Sensordaten ohne Interaktion des Benutzers erfasst werden. Das größte Problem ist die geringe Standby-Zeit gängiger Smartphones. Deshalb wurde bei energieintensiver Erfassung ein tragbares Ladegerät verwendet. Dank der LLDL ist es aber auch möglich, eine energiesparende Erfassung zu konfigurieren. Eine Erfassung, die zum Beispiel alle zwanzig Minuten Position, Luftdruck, Telefongespräche und SMS-Nachrichten erfasst, belastet das Smartphone mit circa zehn Prozent der Akkuleistung. Hingegen belastet eine Erfassung aller Daten alle zehn Sekunden den Energiehaushalt des Smartphones dramatisch. Trotz eines mobilen Ladegeräts sinkt dabei der Akkustatus um circa zehn Prozent pro Stunde. Es gibt noch eine Vielzahl von Optimierungsmöglichkeiten. Beispielsweise könnten mithilfe einer eigenen internen Datenbank sämtliche bekannte Sendemasten und zugehörige Positionen erfasst werden. Damit könnte eine bekannte Position ohne Datenverbindung ermittelt werden. Zukünftig könnten Lifeloggingsysteme allgegenwärtig sein. Durch die fortschreitende Miniaturisierung wird der Energieverbrauch weiter sinken. Somit kann eine genauere und präzisere Datenerfassung erfolgen.

ANHANG A

Anhang

A.1 Klassendiagramme

at.hochi.android.all.broadcastreceiver Im Package *broadcastreceiver* (siehe Abbildung A.2) befinden sich Komponenten, die für den Empfang von asynchronen System- oder Anwendungsdaten zuständig sind. Damit ein Empfänger eine Nachricht erhält, muss die Klasse *BroadcastReceiver* erweitert werden, und sich für spezifische Dienste registrieren. Dies geschieht je nach Art des Dienstes entweder programmiertechnisch zur Laufzeit oder, durch einen Eintrag in der *AndroidManifest.xml* Datei. Beispielsweise empfängt der zur Laufzeit abonnierte *ScheduleReceiver* regelmäßig Nachrichten vom System-Zeitgeber und ermöglicht damit einen geregelten Ablauf der Anwendung.

at.hochi.android.all.database Das Package *database* (siehe Abbildung A.3) ist für die Verwendung der SQLite Datenbank zuständig. Die Singletonklasse *Database* ermöglicht einen synchronisierten Zugriff auf die Daten. Das bedeutet, der Zugriff ist für einen einzelnen Prozess

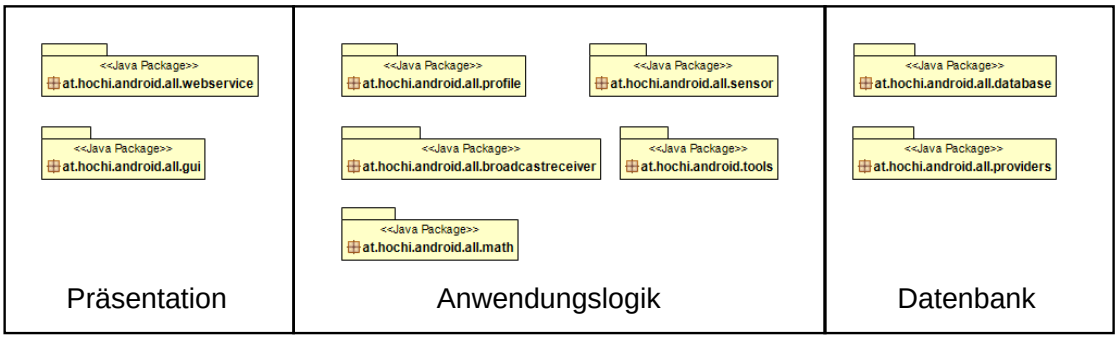


Abbildung A.1: MVC Unterteilung von ALL in Pakete.

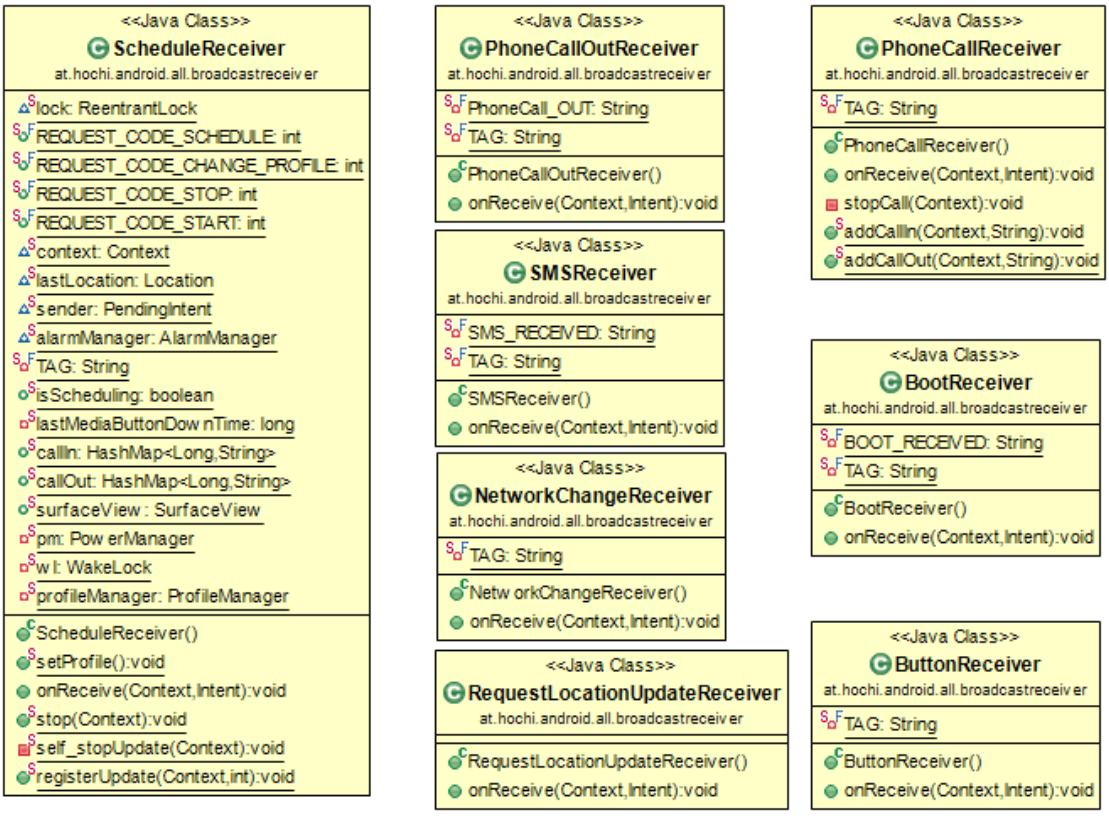


Abbildung A.2: Package *at.hochi.android.all.broadcastreceiver*.

synchronisiert, dennoch ermöglicht Android die gleichzeitige Verwendung von unterschiedlichen Prozessen. Da simultane Lese- und Schreibzugriffe bei Android Lifelogging erfolgen, diese aber synchronisiert sind, werden die SQLite Beschränkungen eingehalten.

at.hochi.android.all.database.entry Das Package *entry* (siehe Abbildung A.4) bildet die SLQ-Tabellen in objektorientierte Klassen ab. Jede Klasse leitet sich dabei von der abstrakten *EventReferenceTable* ab. Mit den zugehörigen Getter- und Settermethoden erfolgt der Zugriff auf die Daten. Durch die Übergabe einer abgeleiteten *EventReferenceTable* an die *Database* Methode *insertEventReferenceTable* wird der Eintrag in die SQLite Datenbank gespeichert. Einzige Ausnahme ist die Tabelle *settings*, sie wird eigens im *at.hochi.android.all.providers* Package behandelt, damit sie den gleichzeitigen Zugriff ermöglicht.

at.hochi.android.all.database.exchange Dieses Package (siehe Abbildung A.5) ist für den Datenaustausch zuständig. Dabei werden der Import und der Export der ALL spezifischen Datei abgearbeitet. Zusätzlich wird ebenfalls der Export des *Kml* und *Gpx* Dateiformates unterstützt.

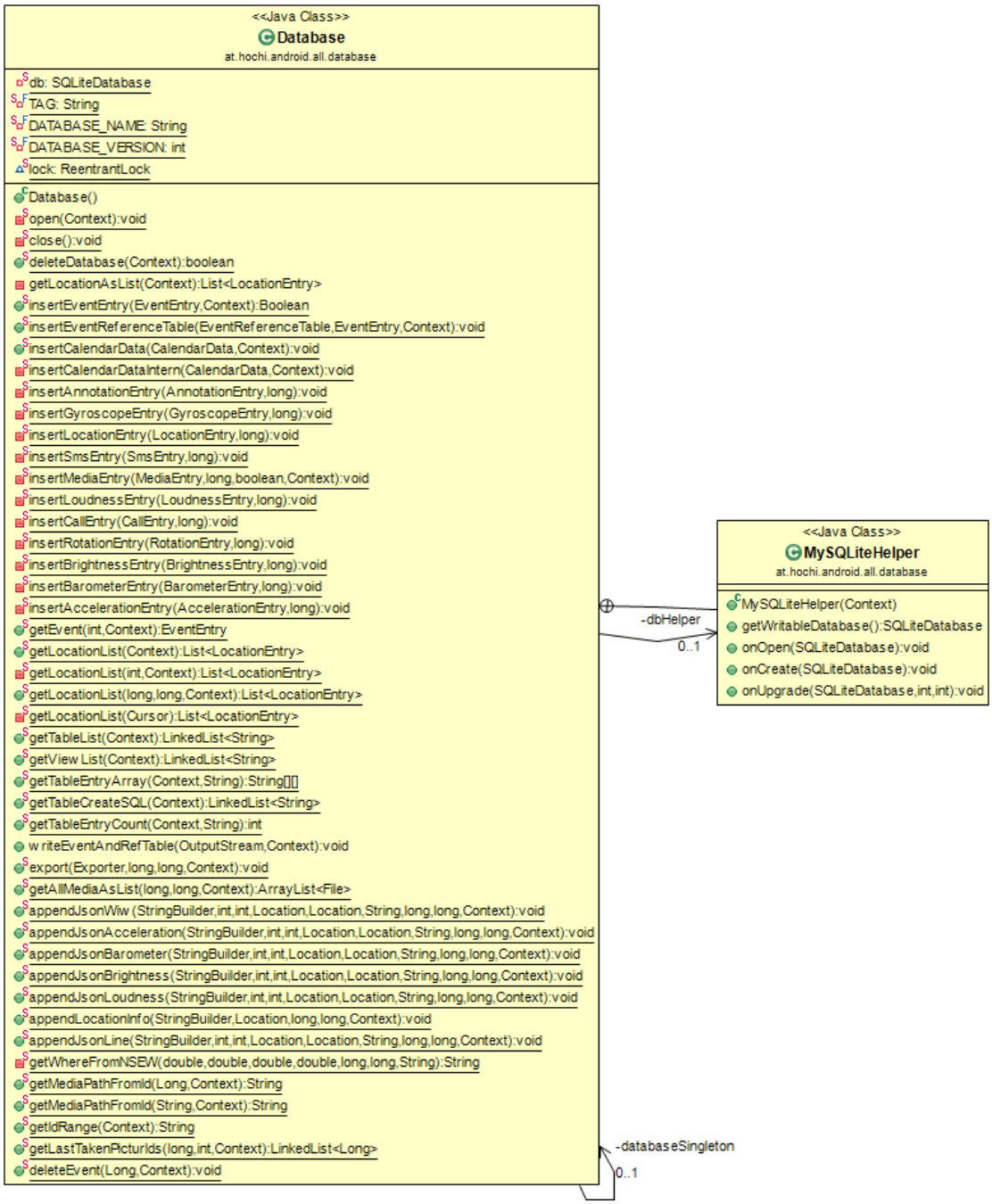


Abbildung A.3: Package *at.hochi.android.all.database*.

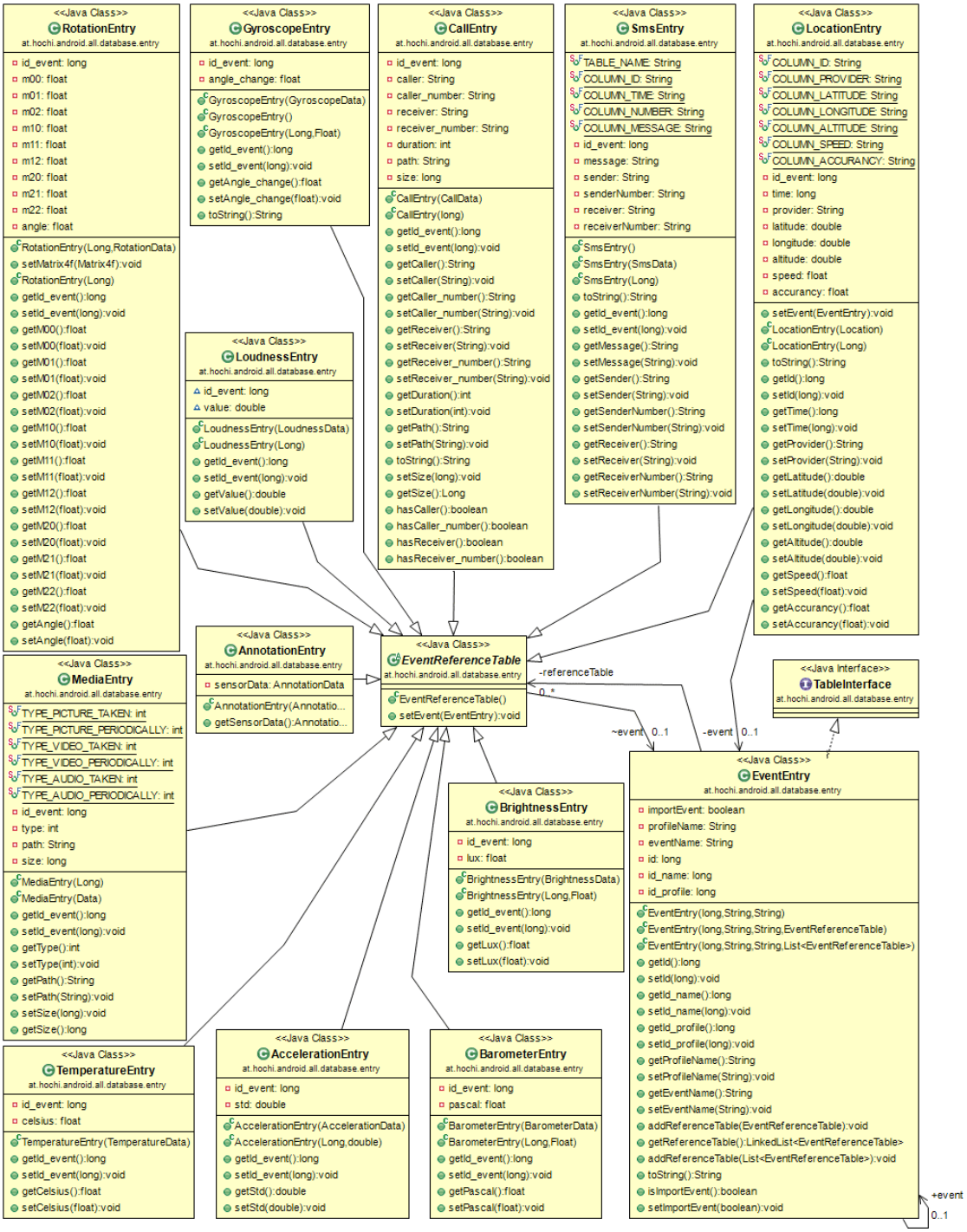


Abbildung A.4: Package *at.hochi.android.all.database.entry*.

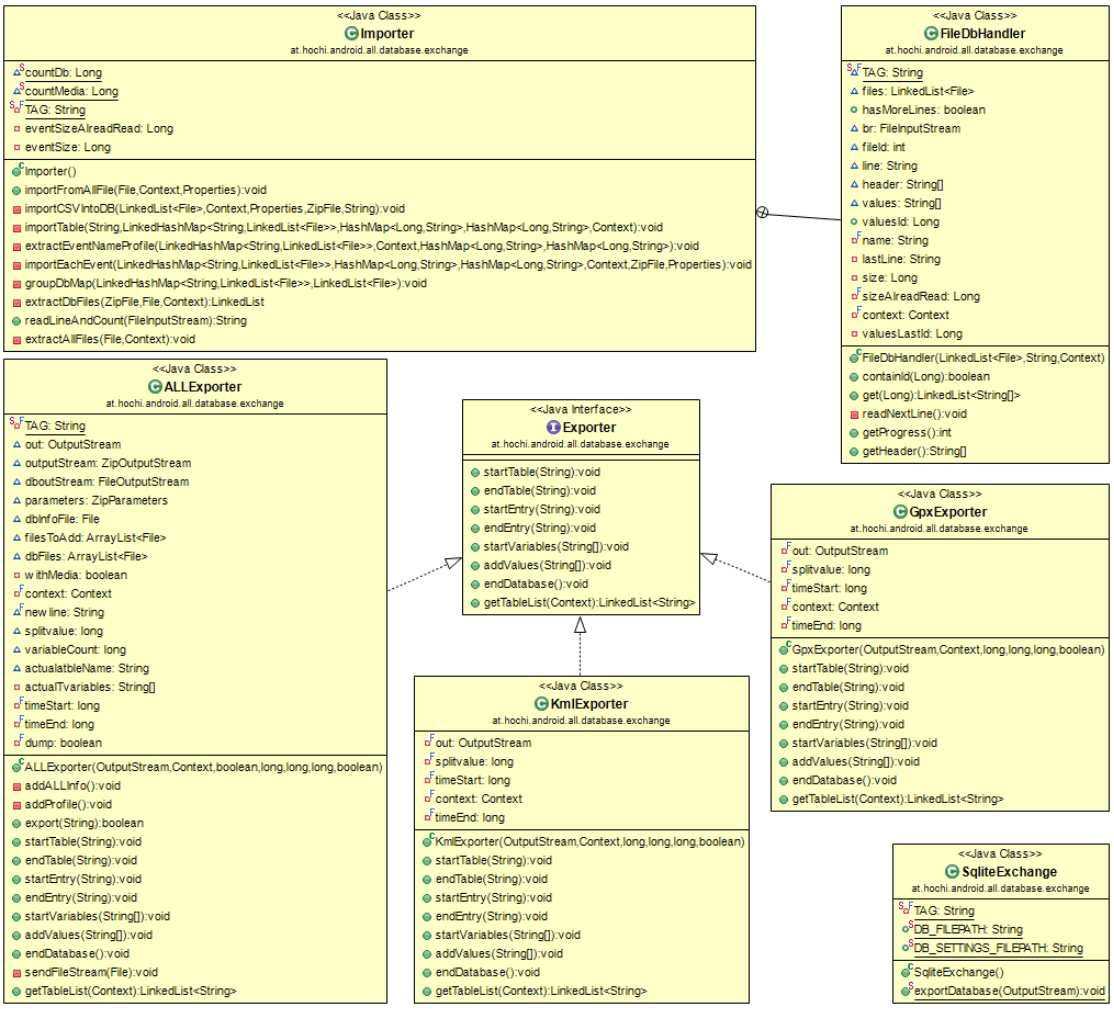


Abbildung A.5: Package *at.hochi.android.all.database.exchange*.

at.hochi.android.all.gui Im *gui* Package (siehe Abbildung A.6) werden sämtliche Interaktionen der grafischen Benutzerschnittstellen abgearbeitet. Die grafische Gestaltung der Benutzeroberfläche erfolgte per XML. Die *ALLActivity* Klasse ist der Startpunkt der Android Lifelogging-Anwendung ALL und wird als erste *Activity* ausgeführt.

at.hochi.android.all.gui.notification Das *notification* Package (siehe Abbildung A.7) ist für das Anzeigen der Statusmeldungen zuständig.

at.hochi.android.all.profile Repräsentiert Profile der LLDL Konfiguration (siehe Abbildung A.8).

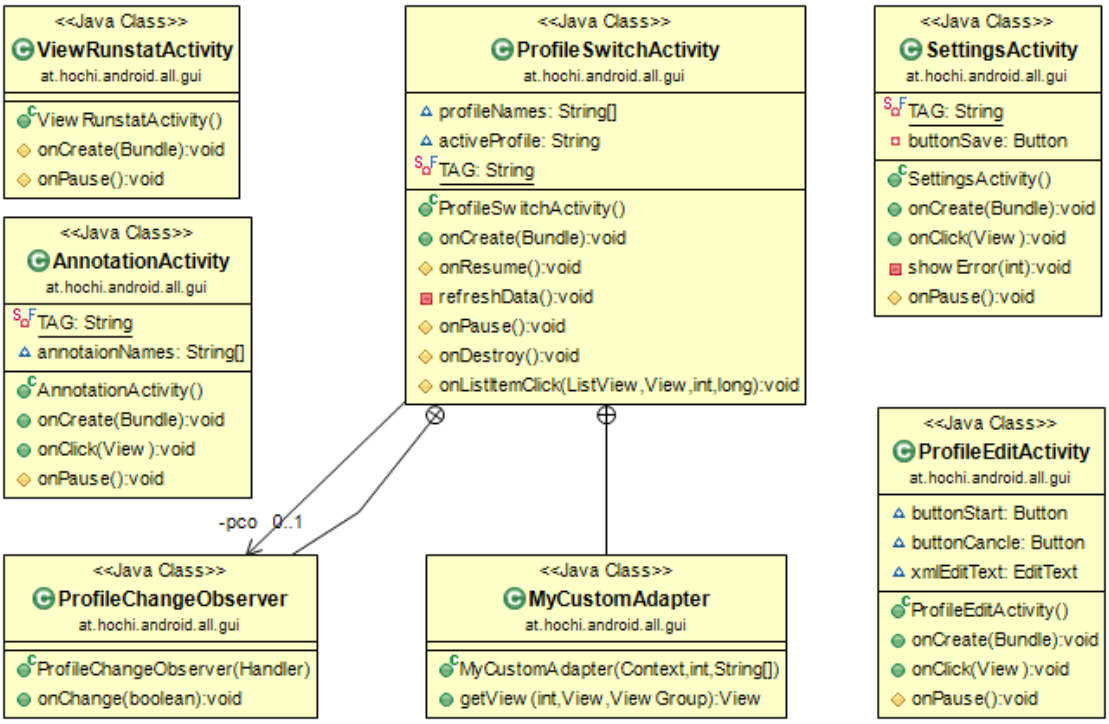


Abbildung A.6: Package *at.hochi.android.all.gui*.

at.hochi.android.all.profile.condition Dieses Package (siehe Abbildung A.9) repräsentiert die Bedingungen, welche für die Aktivierung von Profilen zuständig sind.

at.hochi.android.all.profile.event Das *event* Package (siehe Abbildung A.10) enthält Klassen, die synchrone und asynchrone Events repräsentieren und für den Ablauf von Profilen notwendig sind.

at.hochi.android.all.profile.note Im *note* Package (siehe Abbildung A.11) sind die einzelnen Notiztypen wie TextNote vertreten.

at.hochi.android.all.profile.src Das *src* Package (siehe Abbildung A.12) repräsentiert die einzelnen Sensorquellen eines Profiles.

at.hochi.android.all.profile.src.dependency Durch das *dependency* Package (siehe Abbildung A.13) werden Abhängigkeiten für Profilupdates definiert. Zum Beispiel kann ein Update erfolgen, wenn die Umgebungshelligkeit groß genug ist.

at.hochi.android.all.sensor Im *sensor* Package (siehe Abbildung A.14) werden die eigentlichen synchronen und asynchronen Sensoren behandelt.

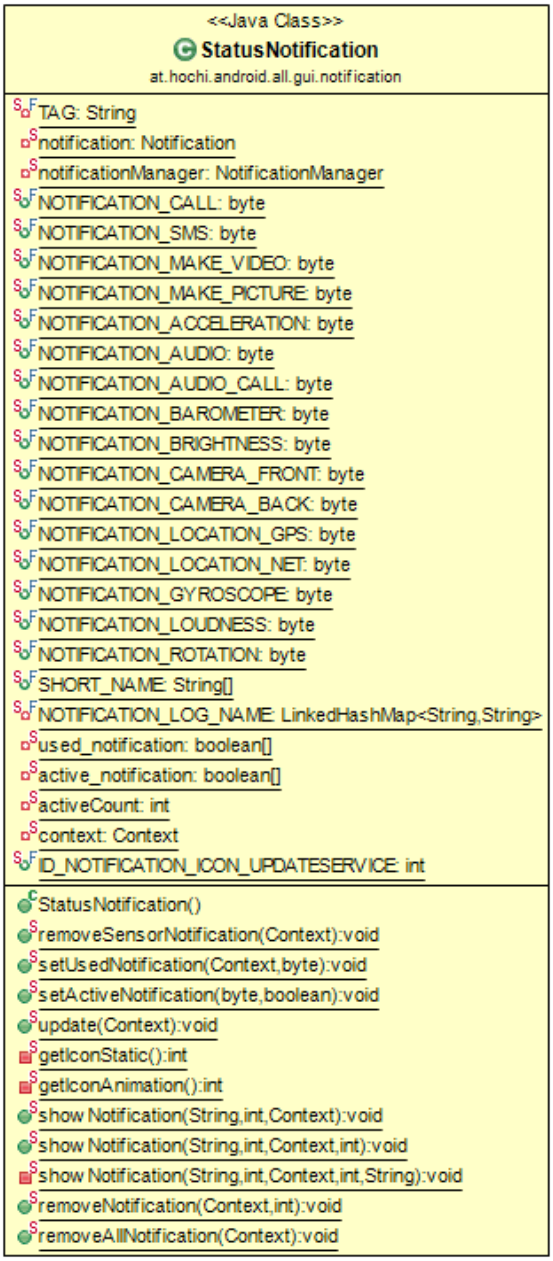


Abbildung A.7: Package *at.hochi.android.all.gui.notification*.

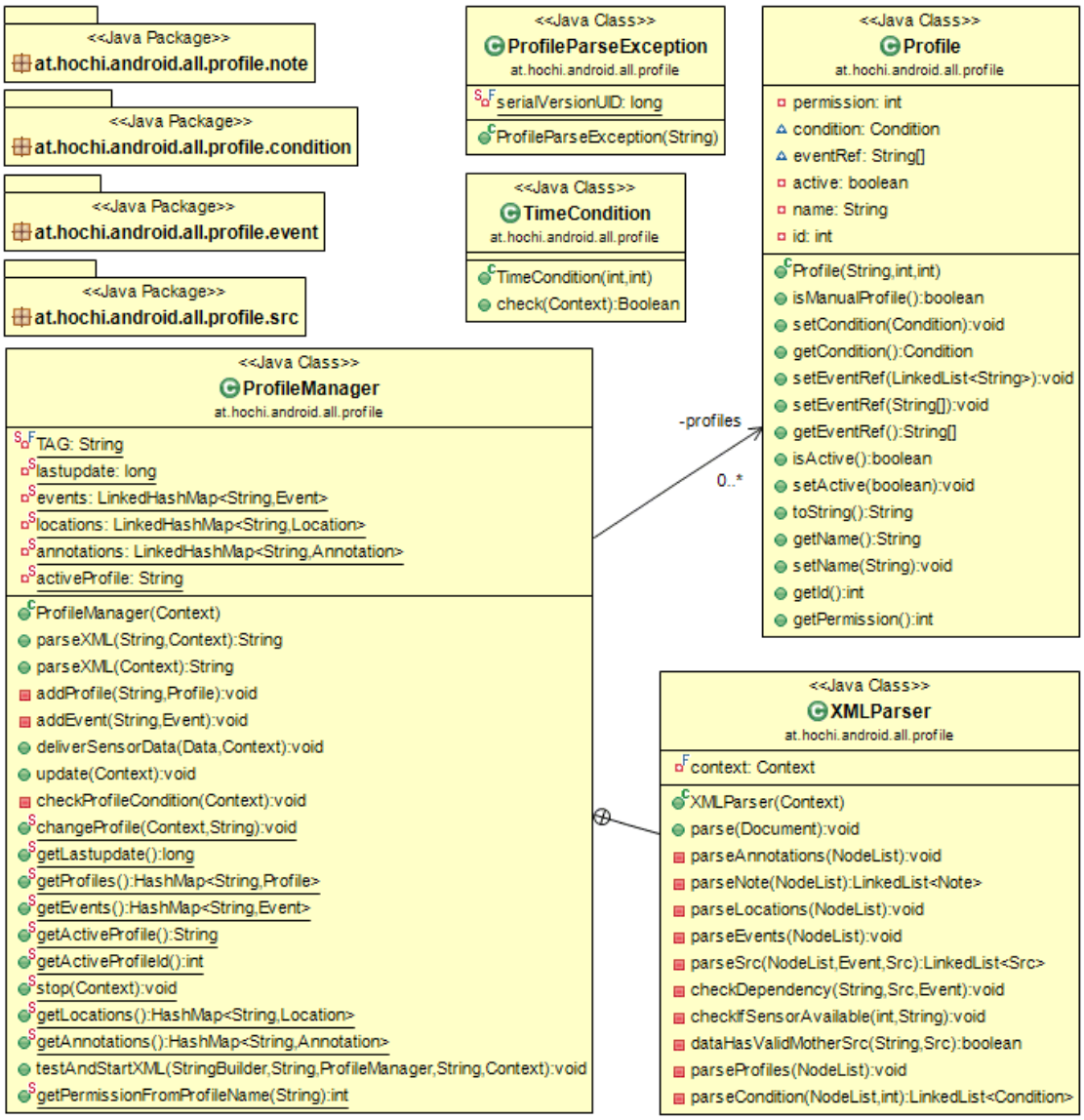


Abbildung A.8: Package *at.hochi.android.all.profile*.

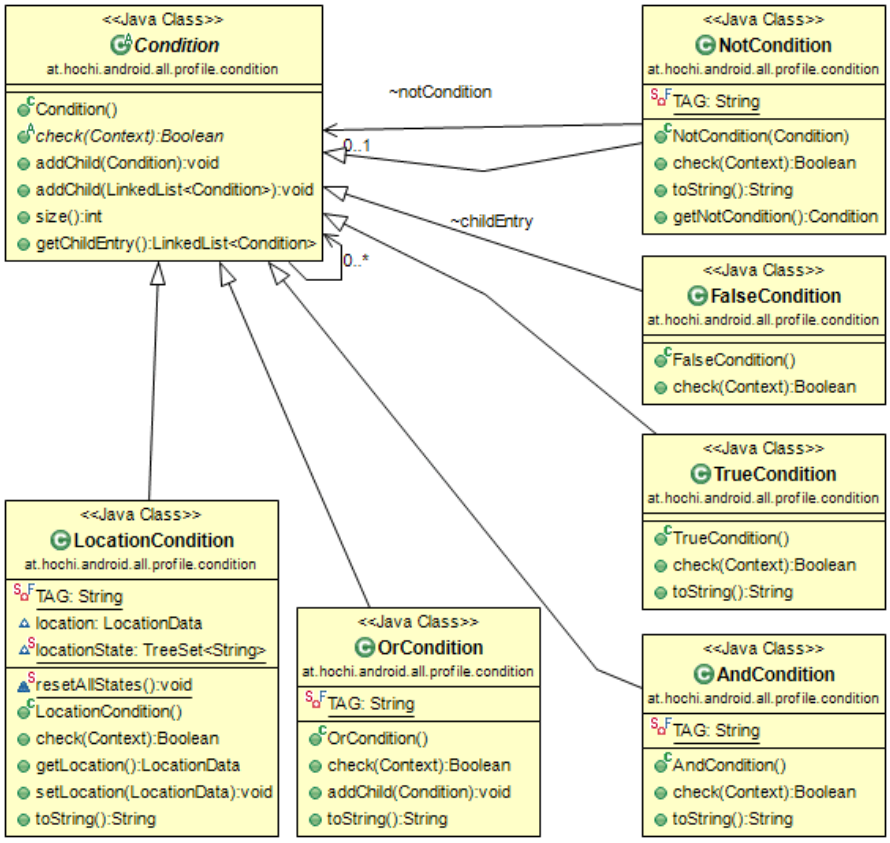


Abbildung A.9: Package *at.hochi.android.all.profile.condition*.

at.hochi.android.all.sensor.asynchronous Im *asynchronous* Package (siehe Abbildung A.15) werden die asynchronen Sensoren definiert, die unter anderem für die Verarbeitung von eintreffenden SMS-Nachrichten zuständig sind.

at.hochi.android.all.sensor.data Das Package (siehe Abbildung A.16) kapselt die einzelnen Sensordaten.

at.hochi.android.all.sensor.synchron Das *synchron* Package (siehe Abbildung A.17) repräsentiert die synchronen Sensoren, beispielsweise für die Ermittlung der Positionsdaten.

at.hochi.android.all.webservice Im *webservice* Package (siehe Abbildung A.18) befinden sich sämtliche Klassen die das Webinterface realisieren.

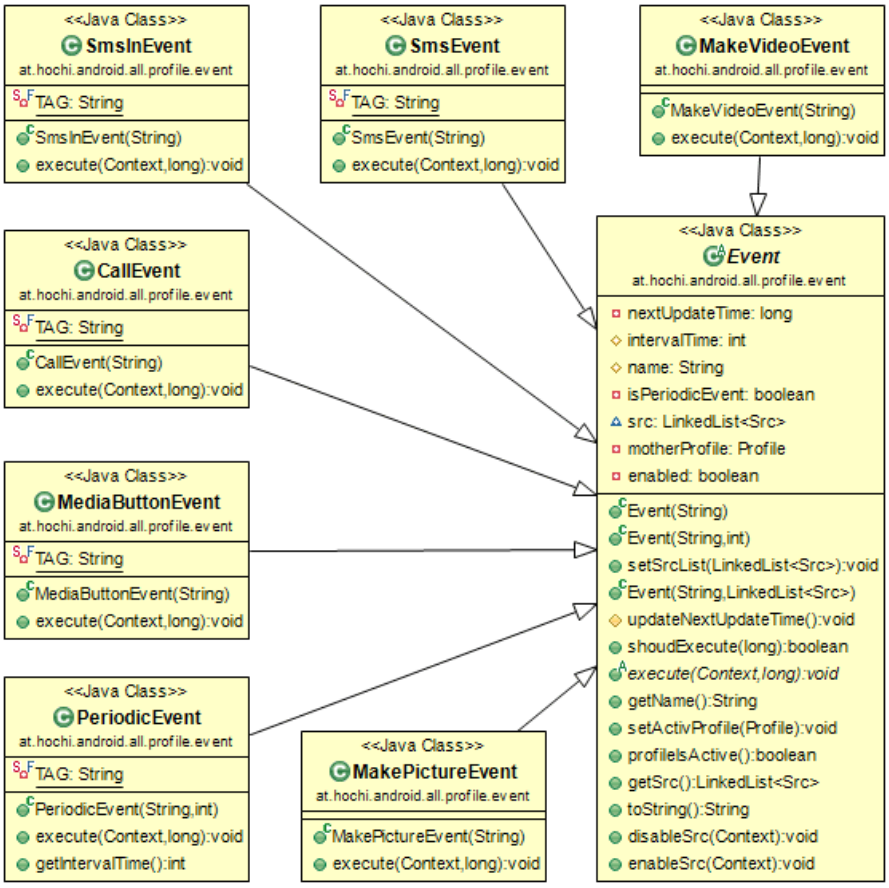


Abbildung A.10: Package *at.hochi.android.all.profile.event*.

A.2 Erstellung des SSL-Zertifikates

Für das HTTPS-Kommunikationsprotokoll des Webinterfaces musste ein eigenes SSL-Zertifikat erstellt werden. Dazu diente das Programm Portecle, das die Erstellung, Bearbeitung und Konvertierung unterschiedlichster Zertifikate unterstützt [118]. Damit das Zertifikat von der Java Klasse java.security.KeyStore benutzt werden kann, muss es im BKS-Dateiformat vorliegen und das zugehörige Passwort bekannt sein. Zuerst muss ein neuer Keystore vom Type BKS erstellt werden. Dann wird mithilfe des *Generate Key Pairs* ein neuer Keystore erstellt. Weiters werden das Passwort gesetzt, und das asymmetrische Schlüsselpaar erstellt.

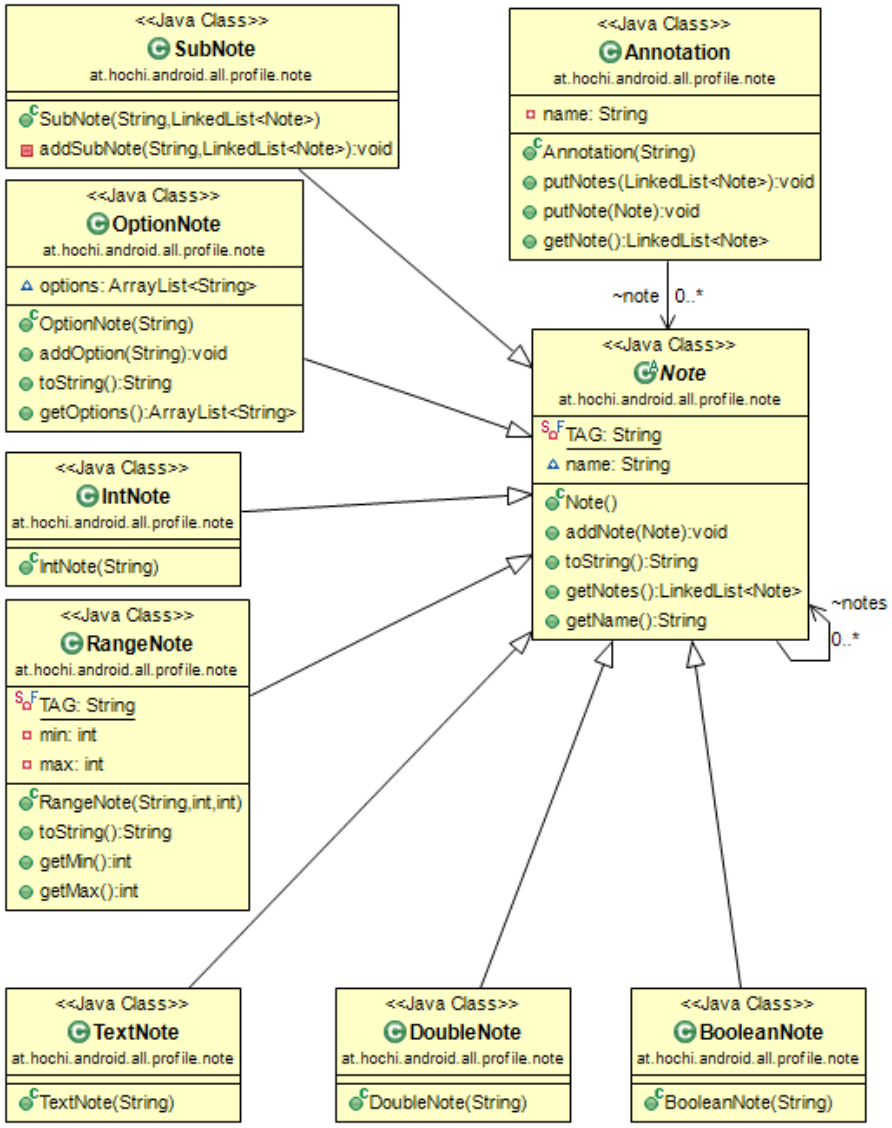


Abbildung A.11: Package *at.hochi.android.all.profile.note*.

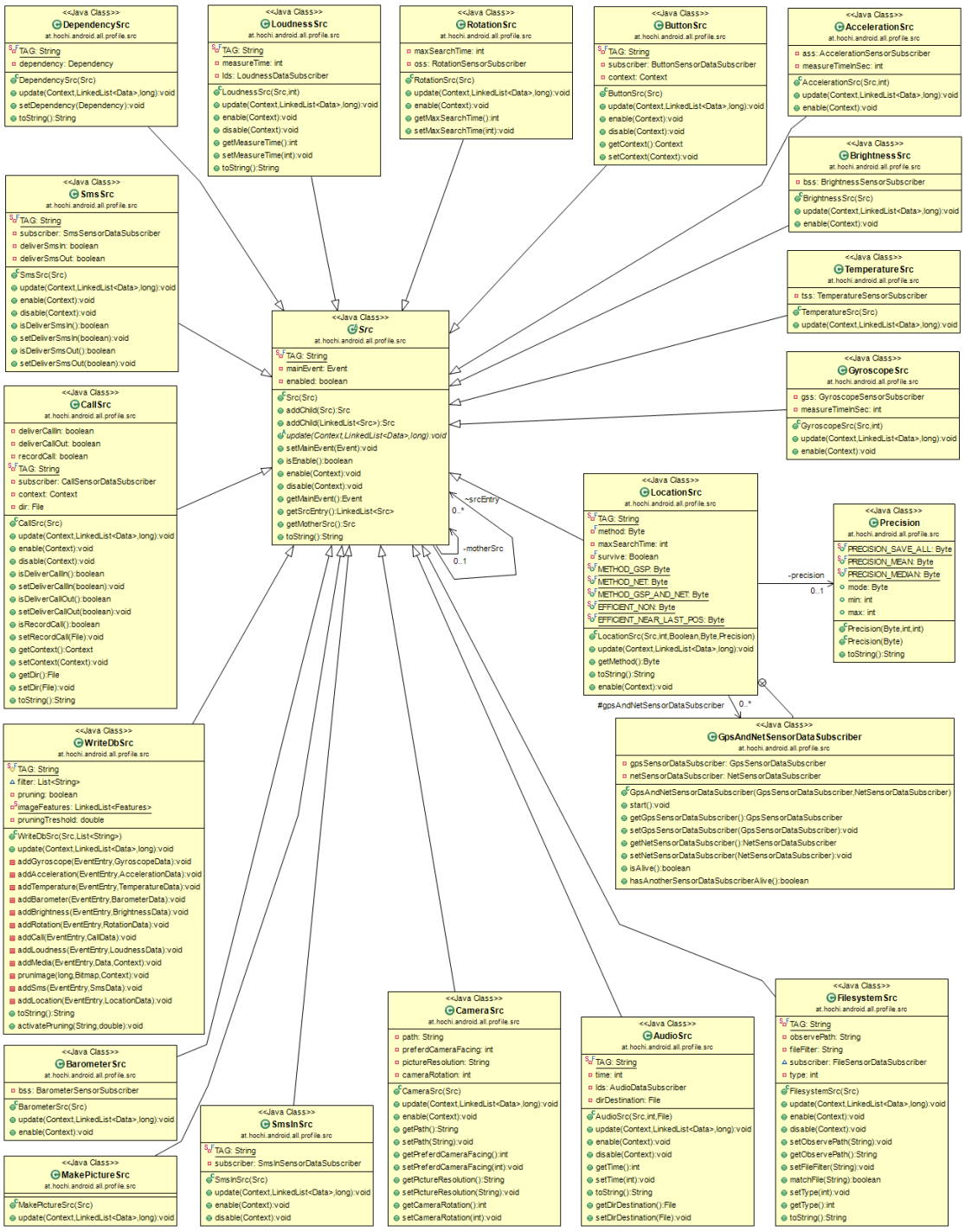


Abbildung A.12: Package *at.hochi.android.all.profile.src*.

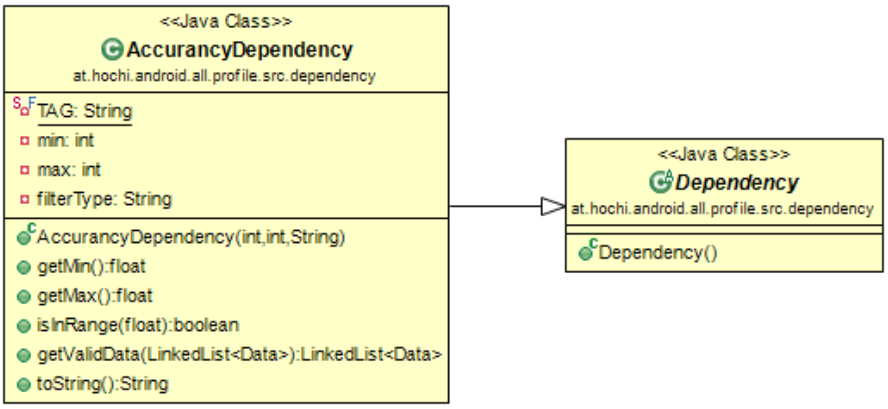


Abbildung A.13: Package *at.hochi.android.all.profile.src.dependency*.

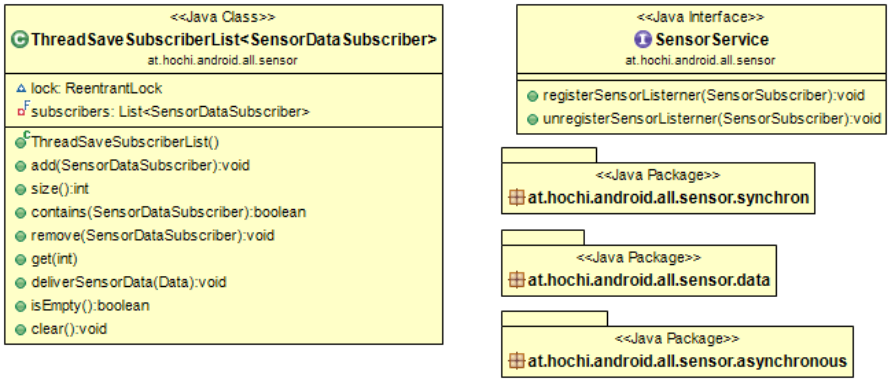


Abbildung A.14: Package *at.hochi.android.all.sensor*.

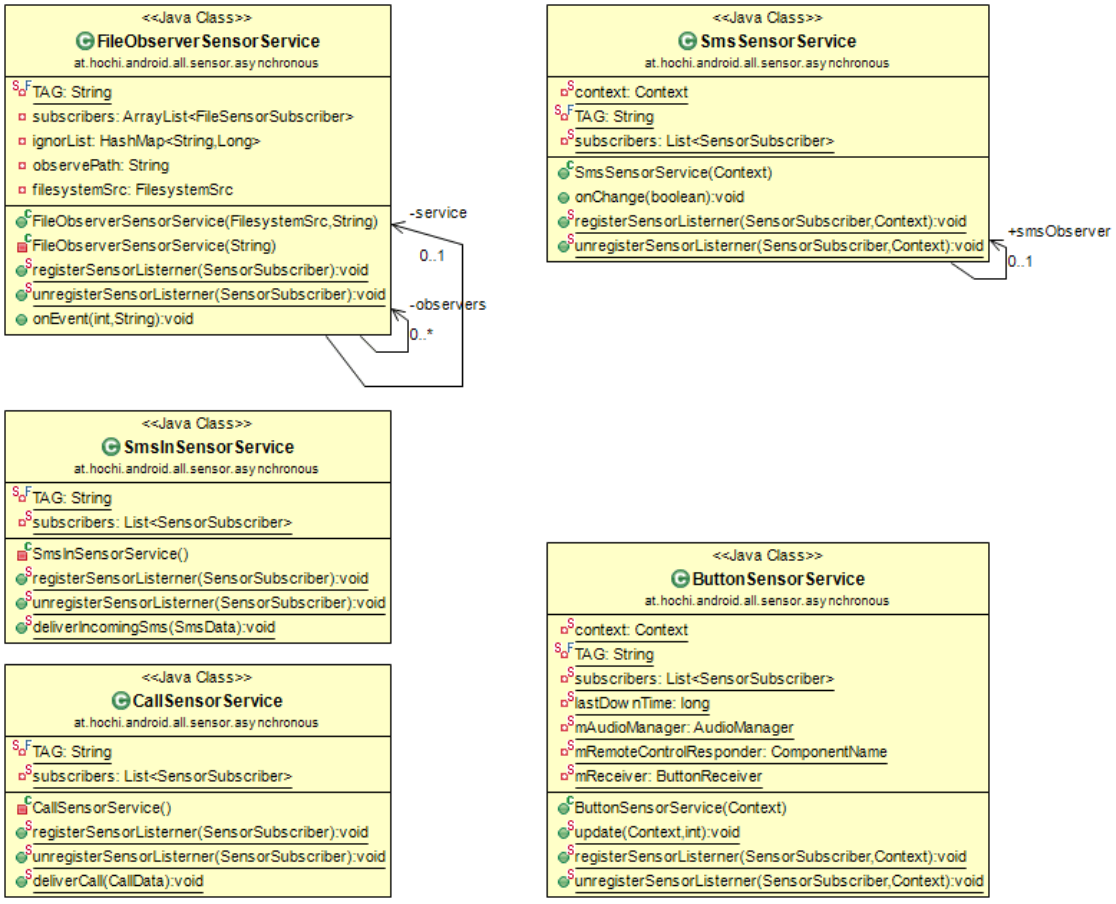


Abbildung A.15: Package *at.hochi.android.all.sensor.asynchronous*.

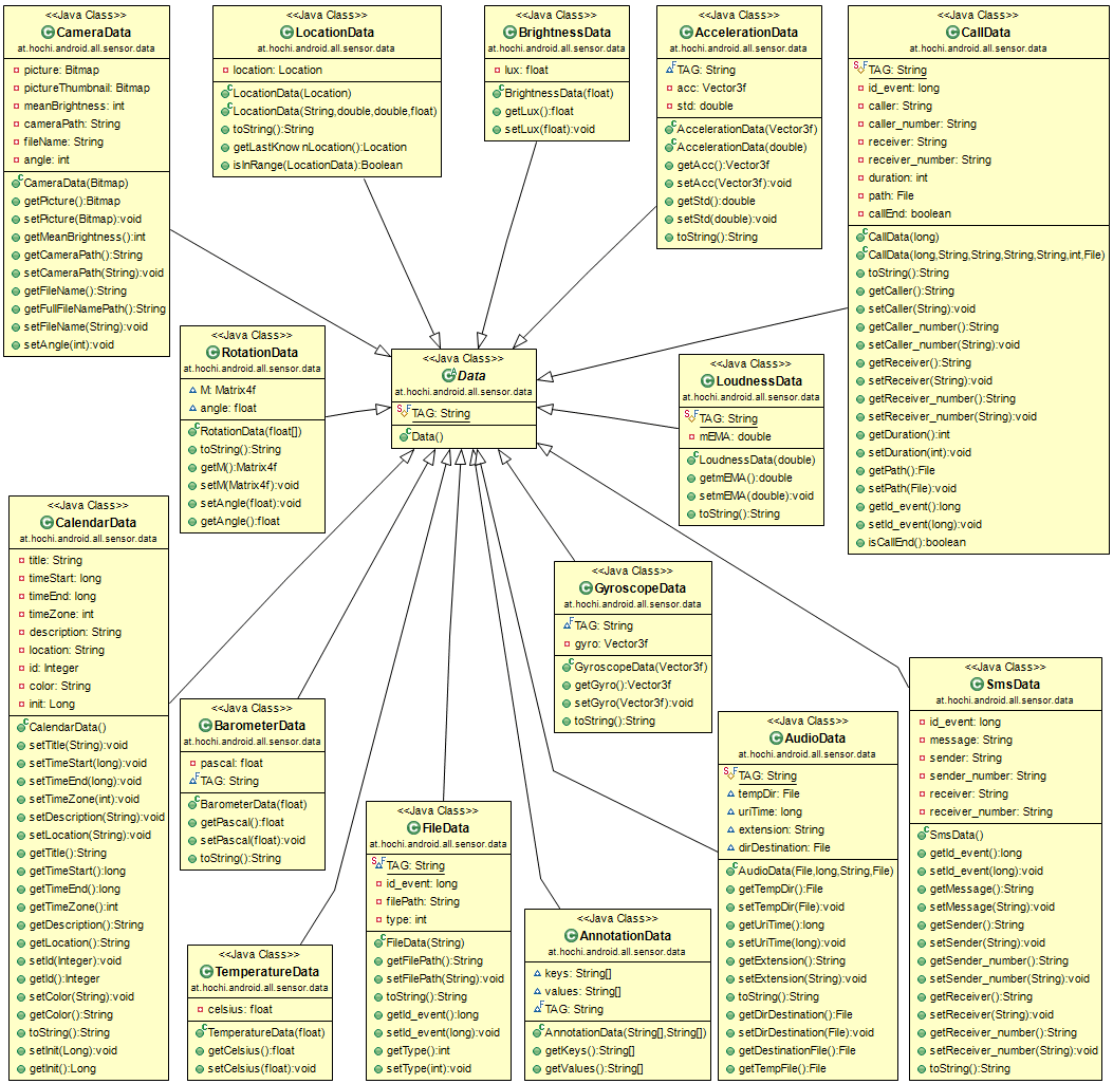


Abbildung A.16: Package *at.hochi.android.all.sensor.data*.

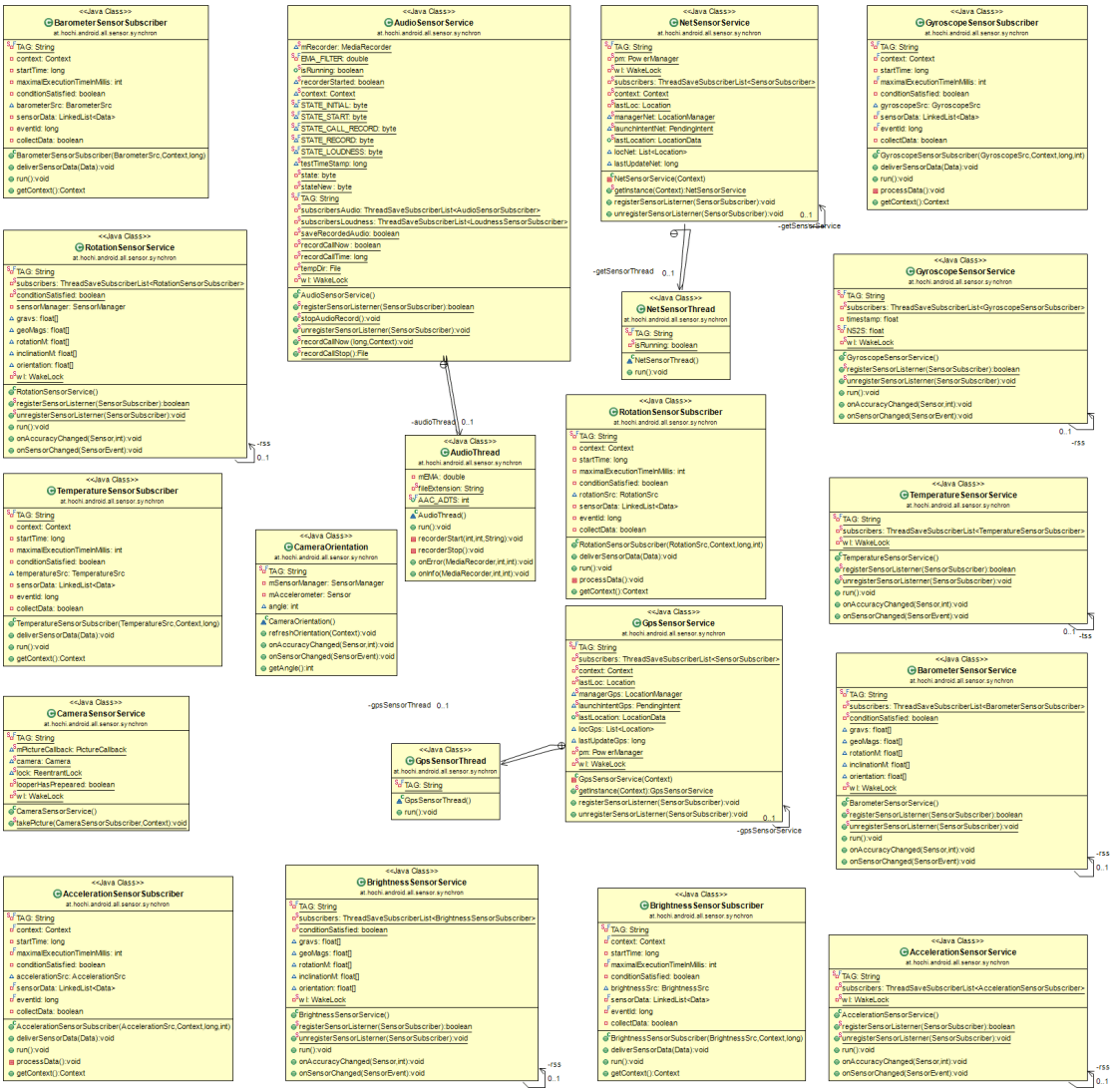


Abbildung A.17: Package *at.hochi.android.all.sensor.synchron*.

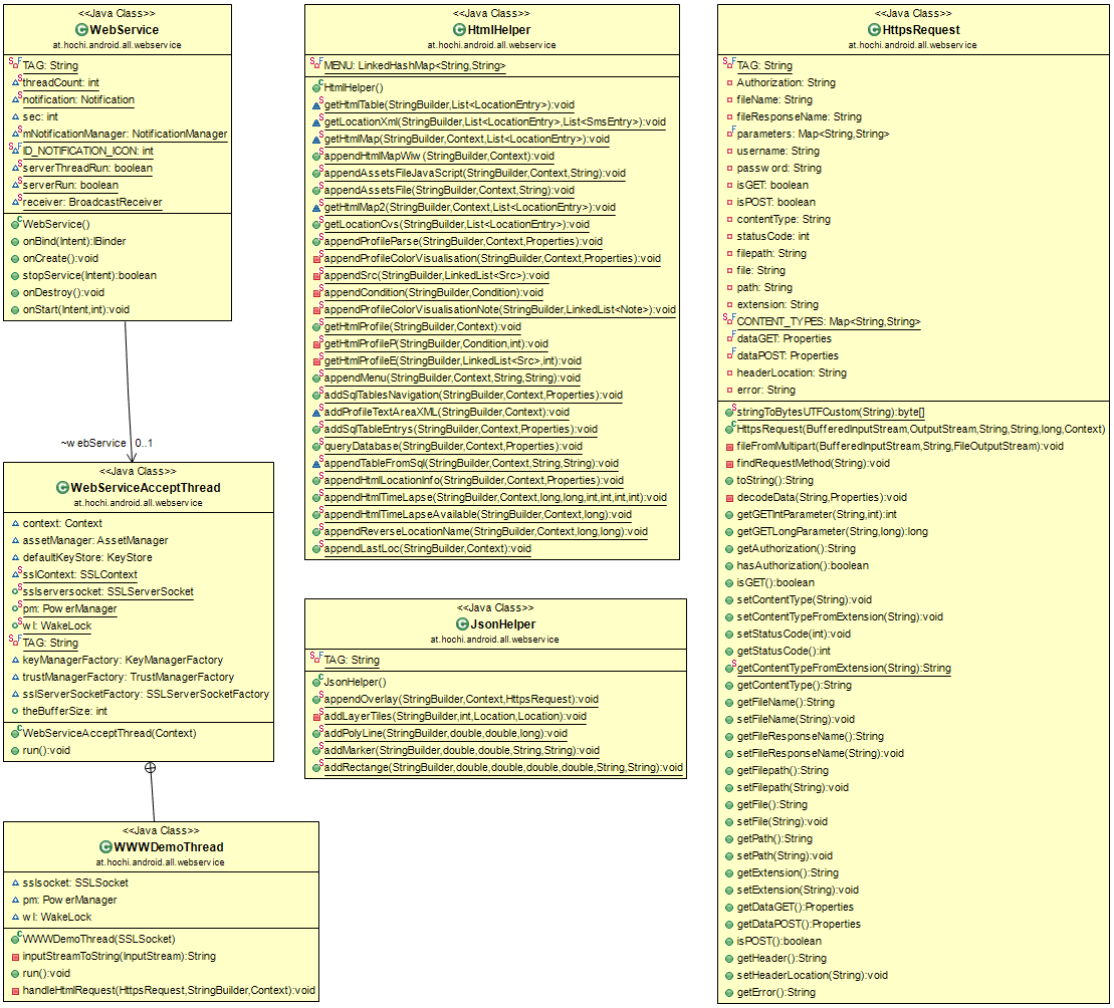


Abbildung A.18: Package *at.hochi.android.all.webservice*.

Abbildungsverzeichnis

1.1 Vannevar Bush Idee von MEMEX 2
1.2 Entwicklung Manns tragbarer Computersysteme. 3
1.3 Entwicklung tragbarer Brillensysteme. 3
1.4 Grundliegender Ablauf von Lifeloggingsystemen. 4
1.5 Zusammenhang zwischen Aufwand und Anzahl um Daten zu erfassen. 5
1.6 Ergebnisse einer Studie zum Erinnerungsvermögen. 7

2.1 *Eudaemons' shoe* mobiler Roulette-Computer. 13
2.2 Studie zur Erforschung tragbarer Computersysteme und neuer Sensoren. 14
2.3 Experimentelles Beleuchtungssystem zur künstlerischen Gestaltung. 15
2.4 Manns energiesparendes erneuertes System 15
2.5 Life Übertragungssystem und Internetplattform aus dem Jahr 1994. 16
2.6 Schematische Darstellung der Eyetap-Technologie 17
2.7 Entwicklung der Eyetap-Prototypen. 17
2.8 AREMAC-Anwendung und Prototyp. 18
2.9 ParcTab *Forget Me Not* Prototyp. 19
2.10 Erste tragbare PDA-Systeme. 20
2.11 Ortung mithilfe von vier Satelliten 22
2.12 Strategie zur Lokalisierung in Android. 22
2.13 Eventdetektion und Gruppierung zu Blöcken. 24
2.14 Bildfolge mit Histogramm und Differenzen. 25
2.15 Fehlerhafte Eventdetektion. 26
2.16 Vergleich mithilfe von Bildblöcken 26
2.17 Zeitrafferansicht von *SenseCam*-Bildern. 27
2.18 Weitere Visualisierungsmethoden von *MyLifeBits*. 28
2.19 Visualisierung von Bildern anhand ihrer Eventlänge 28
2.20 Orts- und zeitspezifische Visualisierung von GPS-Positionen 29
2.21 Gesichtserkennung und Verschleierung mithilfe eines Gesichtsdetektors. 30

3.1 Memory Prosthesis Audio Lifeloggingsystems. 32
3.2 StartleCam Prototyp mit Verlauf des Hautwiderstandes. 33
3.3 Der interne Aufbau der SenseCam. 33
3.4 Aussehen, Trageweise und Bilder von SenseCam. 34

3.5 Google Project Glass Prototypen. 35
3.6 Tragbares Gehirnaktivitäts-Messgerät. 35
3.7 Aizawa Lifeloggingsystem-Komponenten. 36
3.8 iGlogger System auf verschiedenen Smartphones. 37
3.9 Blackberry WMMS. 37
3.10 Tragbare Gesundheitsüberwachung Lifeloggingeinheit. 38
3.11 Prototyp eines Systems, das anhand von Vibrationen Oberflächen erkennt. 38
3.12 Struktureller Aufbau und GUI von MyExperience. 39
3.13 Beispielkonfiguration von UbiqLog. 41
3.14 GUI von UbiqLog. 41

4.1 Erhältliche Systeme: Vicon Revue, Autographer und Memoto. 46
4.2 Strukturell Aufbau und GUI von MyExperience. 47

5.1 ALL Anwendungsfalldiagramm zur Veranschaulichung des Systems. 52
5.2 Architektur und grundlegende Konzepte von ALL. 54
5.3 Datenbank Schema von Android Lifelogging. 56
5.4 Allgemeiner Überblick über die ALL-Architektur 58
5.5 Grundlegende Struktur der XML-Konfiguration und abstrahierte Darstellung. . . . 59
5.6 Zwei *locations* Elemente mit IDs. 61
5.7 XML-Deklaration einzelner Profile. 62
5.8 Deklaration eines automatischen Profils. 63
5.9 Deklaration eines manuellen Profils. 63
5.10 Allgemeine Struktur des Elements *events*. 64
5.11 XML-Deklaration eines synchronen Events. 65
5.12 XML-Deklaration eines asynchronen Events. 66
5.13 Spezifikation der *src* Referenzen. 66
5.14 Spezifikation von *dependency* und *writeDb* Knoten. 67
5.15 Grundlegende *annotations*-Spezifikation. 68
5.16 Beispiel einer *annotation*-Deklaration. 68
5.17 Erstellung einer komplexeren *annotation* Spezifikation. 69
5.18 Regelmäßiger Trigger durch Echtzeituhr. 72
5.19 Abonnement mehrerer SensorSubscriber eines SensorService. 72
5.20 Asynchroner Trigger des ButtonSensor. 73
5.21 Asynchrone Abfolge des CallSensor. 74
5.22 Aufruf von Inotify an den FileObserver. 75
5.23 SmsSensor Benachrichtigung von empfangenen und gesendeten SMS-Nachrichten. 75
5.24 Schematischer Aufbau eines Beschleunigungssensors Mikrosystemes. 77
5.25 Zustände der AudioSensor Klasse. 80
5.26 Luftdruck in hPa in Stockholm von 1756 - 1998. 80
5.27 Die geographische Breite ± 90 ° und geographische Länge ± 180 ° der Erde. 82
5.28 Veranschaulichung des Hall-Effekts. 83
5.29 Berechnung der Lage mithilfe von Erdbeschleunigung und Magnetismus. 84
5.30 Synchrones Update der GPS-Position. 85

5.31 Asynchrones Update durch eine eintreffende SMS-Nachricht. 86

6.1 Logo der Android Lifelogging App. 89
6.2 Android Lifelogging GUI Übersicht. 90
6.3 Android Lifelogging GUI Hauptinteraktionen. 91
6.4 Benachrichtigung von Android Lifelogging. 92
6.5 Auswahl und Erstellung einer Anmerkung. 93
6.6 Benachrichtigung über aktiviertes Webinterface und URL. 94
6.7 Webinterface Passwortschutz und Sicherheitswarnung. 94
6.8 Startseite des Webinterfaces mit Navigation. 95
6.9 Farbige Hervorhebung der LLDL-Deklaration. 96
6.10 Detailgrade der LLDL-Abstraktion. 96
6.11 Web-basierter Datenbankbrowser mit Übersicht der Tabellen. 97
6.12 Datenbank Rohdaten der Tabelle acceleration. 98
6.13 SQL-Abfrage über das Webinterface. 98
6.14 SQL-Abfrage mit Linienplot. 99
6.15 SQL-Abfrage mit Berechnung der Entfernung zu den definierten Orten. 99
6.16 Import einer Sicherung im All Dateiformat. 100
6.17 Visualisierung unterschiedlicher Detailheitsgrade. 101
6.18 Farben des Datenkreises mit zugehörigen Datentypen. 101
6.19 Visualisierung der Daten auf einer Landkarte. 102
6.20 Landkarten Zeitbegrenzung, Clustering und Weg-Darstellung. 103
6.21 Street View Ansicht mit Datenkreisen die besuchte Positionen visualisieren. 104
6.22 Visualisierung eines Weges mit Google Earth. 104
6.23 Zeitraffer Übersicht von ALL. 105
6.24 Entfernung redundanter Bilder mithilfe der Histogramm-Intersection. 106
6.25 Zeitraffer-Bilder mit Gesichtserkennung für Anonymisierung. 107
6.26 Verteilung der WLAN/Net-Rohdaten, Dachgeschoss (300 Messwerte). 108
6.27 Verteilung der GPS-Rohdaten, freie Sicht zum Himmel (1000 Messwerte). 109
6.28 Zwei mithilfe des k-Means-Algorithmus ermittelten Clusterzentren. 110
6.29 Route mit WLAN/Net-Lokalisierung. 111
6.30 Zu erwartender Fehler entsprechend der Median-Filtergröße. 112
6.31 Vergleich zwischen dem zu erwartenden Luftdruck 113

A.1 MVC Unterteilung von ALL in Pakete. 117
A.2 Package *at.hochi.android.all.broadcastreceiver*. 118
A.3 Package *at.hochi.android.all.database*. 119
A.4 Package *at.hochi.android.all.database.entry*. 120
A.5 Package *at.hochi.android.all.database.exchange*. 121
A.6 Package *at.hochi.android.all.gui*. 122
A.7 Package *at.hochi.android.all.gui.notification*. 123
A.8 Package *at.hochi.android.all.profile*. 124
A.9 Package *at.hochi.android.all.profile.condition*. 125
A.10 Package *at.hochi.android.all.profile.event*. 126

A.11 Package *at.hochi.android.all.profile.note*. 127
A.12 Package *at.hochi.android.all.profile.src*. 128
A.13 Package *at.hochi.android.all.profile.src.dependency*. 129
A.14 Package *at.hochi.android.all.sensor*. 129
A.15 Package *at.hochi.android.all.sensor.asynchronous*. 130
A.16 Package *at.hochi.android.all.sensor.data*. 131
A.17 Package *at.hochi.android.all.sensor.synchron*. 132
A.18 Package *at.hochi.android.all.webservice*. 133

Quellen- und Literaturverzeichnis

[1] BUSH, Vannevar: As We May Think. In: *Atlantic Monthly* (1945), Juli, S. 1–12

[2] *1945 Vannevar Bush*. http://www.contrainfo.com/1296/en-1945-vannevar-bush-predijo-internet, Abruf: 20. Jänner 2013

[3] *Eudaemonic*. http://www.wearcam.org, Abruf: 20. Jänner 2013

[4] MANN, Steve: An historical account of the 'WearComp' and 'WearCam' inventions developed for applications in 'Personal Imaging'. In: *Wearable Computers* IEEE, 1997

[5] *Slashgear: From Cyborgs to Project Glass: the Augmented Reality Story*. http://goo.gl/0aysR, Abruf: 20. Jänner 2013

[6] *Google[x] started Project Glass*. https://plus.google.com/+projectglass/posts, Abruf: 20. Jänner 2013

[7] MANN, Steve ; NIEDZVIECKI, H: *Cyborg: digital destiny and human possibility in the age of the wearable computer*. Computing Reviews, 2001. – ISBN 0385658265

[8] *Nlogging, Microblogging, Lifelogging. Evolution?* http://www.kzero.co.uk/blog/blogging-microblogging-lifelogging-evolution/, Abruf: 20. Jänner 2013

[9] HODGES, S. ; WILLIAMS, L. ; BERRY, E. ; IZADI, S. ; SRINIVASAN, J. ; BUTLER, A. ; SMYTH, G. ; KAPUR, N. ; WOOD, K.: SenseCam: A retrospective memory aid. In: *UbiComp 2006: Ubiquitous Computing* (2006), S. 177–193

[10] SELLEN, A.J. ; FOGG, A. ; AITKEN, M. ; HODGES, S. ; ROTHER, C. ; WOOD, K.: Do life-logging technologies support memory for the past? an experimental study using sensecam. In: *Proceedings of the SIGCHI conference on Human factors in computing systems* ACM, 2007, S. 81–90

[11] *What Happens to Your Facebook Profile When You Die? - TIME*. http://www.time.com/time/business/article/0,8599,1932803,00.html, Abruf: 20. Jänner 2013

[12] *Sleep Time | Azumio*. http://www.azumio.com/apps/sleep-time/, Abruf: 20. Jänner 2013

[13] *The Best Running Apps | Men's Health*. http://www.menshealth.com/techlust/best-running-apps, Abruf: 20. Jänner 2013

[14] *Google Now gets you just the right information at just the right time.* http://www.google.com/landing/now/, Abruf: 20. Jänner 2013

[15] *emWave2*. http://www.heartmathstore.com/category/emwave2/, Abruf: 20. Jänner 2013

[16] SMEATON, A.F.: The unexpected applications of sensors: Home energy and lifestyle analysis. In: *ESF Exploratory Workshop on The Internet of Things for a Sustainable Future, Vielsalm (Belgium)*, 2011

[17] DAI, J. ; BAI, X. ; YANG, Z. ; SHEN, Z. ; XUAN, D.: Mobile phone-based pervasive fall detection. In: *Personal and ubiquitous computing* 14 (2010), Nr. 7, S. 633–643. – Springer-Verlag

[18] VISHWAKARMA, Vinay ; MANDAL, Chittaranjan ; SURAL, Shamik ; GHOSH, Ashish (Hrsg.) ; DE, Rajat K. (Hrsg.) ; PAL, Sankar K. (Hrsg.): *Lecture Notes in Computer Science*. Bd. 4815: *Automatic detection of human fall in video*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2007

[19] *LifeLogging | Quantified Self*. http://quantifiedself.com/lifelogging/, Abruf: 20. Jänner 2013

[20] ROSE: The Design of a Real-Time, Multimodal Biofeedback System for Stroke Patient Rehabilitation. In: *Proceedings of the 14th annual ACM international conference on Multimedia* (2006), S. 763–772

[21] WU, Pang ; PENG, Huan-Kai ; ZHU, Jiang ; ZHANG, Ying: Senscare: Semi-automatic activity summarization system for elderly care. In: *Mobile Computing, Applications, and Services* (2012), S. 1–19

[22] ROSE: Food Log by Analyzing Food Images. In: *Proceeding of the 16th ACM international conference on Multimedia* (2008), August, S. 1–2

[23] LEE, Matthew L. ; DEY, Anind K.: Lifelogging memory appliance for people with episodic memory impairment. In: *UbiComp* (2008), S. 44–53. – ACM

[24] KIKHIA, B ; HALLBERG, J ; SYNNES, K ; SANI, Z: Context-aware Life-Logging for Persons with Mild Dementia. In: *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, IEEE, 2009, S. 6183–6186

[25] BAJRAMI, G ; DERAWI, M O. ; BOURS, P: Towards an automatic gait recognition system using activity recognition (wearable based). In: *3rd International Workshop on Security and Communication Networks (IWSCN)* (2011)

[26] *TED Blog | FBI, here I am! Hasan Elahi on TED.com*. http://blog.ted.com/2011/10/31/fbi-here-i-am-hasan-elahi-on-ted-com/, Abruf: 20. Jänner 2013

[27] BYRNE, Daragh ; DOHERTY, Aiden R. ; JONES, Gareth J. F. ; SMEATON, Alan F. ; KUMPULAINEN, Sanna ; JÄRVELIN, Kalervo: The SenseCam as a tool for task observation, British Computer Society, 2008, S. 19–22

[28] *Project Glass: Live Demo At Google I/O - YouTube*. http://www.youtube.com/watch?v=D7TB8b2t3QE

[29] *Forschungsfeld: Digitale Gedächtnisse*. http://www.stefan-selke.de/nc/forschung/feld/forschungsfelder/digitale-gedaecht-nisse.html, Abruf: 20. Jänner 2013

[30] *Eudaemonic*. http://www.wearcam.org/eudaemonic, Abruf: 20. Jänner 2013

[31] THORP, E.O.: The invention of the first wearable computer. In: *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, IEEE, 1998, S. 4–8

[32] BRABYN, J A.: Laboratory studies of aided blind mobility. In: *University of Canterbury. Electrical Engineering* (1978)

[33] *Virtual Research Systems - User's Guide VR4*. : *Virtual Research Systems - User's Guide VR4*, http://www.virtualresearch.com/Acrobat_files/VR_VR4man.pdf

[34] LANE, F.S.: Obscene Profits: The Entrepreneurs of Pornography in the Cyber Age, Routledge New York, 2001. – ISBN 0415920965, S. 252–254

[35] *A review of head-mounted display devices, etc.* http://wearcam.org/head-mounted-displays.html, Abruf: 20. Jänner 2013

[36] *Smithsonian Museum in the United States, and the Science Museum (e.g. Wellcome Wing, where WearComp was shown) in the U.K.* http://wearcam.org/dusting/mztv/mztv_proposal.htm, Abruf: 20. Jänner 2013

[37] MANN, S.: Continuous lifelong capture of personal experience with EyeTap. In: *Proceedings of the the 1st ACM workshop on Continuous archival and retrieval of personal experiences* ACM, 2004, S. 1–21

[38] MANN, Steve ; FUNG, James ; MONCRIEFF, Eric: EyeTap technology for wireless electronic news gathering. In: *ACM SIGMOBILE Mobile Computing and Communications Review* 3 (1999), Oktober, Nr. 4, S. 19–26

[39] *EyeTap Personal Imaging Lab*. http://eyetap.org, Abruf: 20. Jänner 2013

[40] MANN, W.S.G.: *Eye-tap for electronic newsgathering, documentary video, photojournalism, and personal safety*. September 2 2003. – US Patent 6,614,408

[41] *Why man melding with machine a good thing*. http://www.cbc.ca/news/background/tech/cellphones/mann.html, Abruf: 20. Jänner 2013

[42] *One on One: Steve Mann, Wearable Computing Pioneer. Steve Mann's EyeTap glasses.* http://bits.blogs.nytimes.com/2012/08/07/one-on-one-steve-mann-wearable-computing-pioneer/, Abruf: 20. Jänner 2013

[43] *Telepoint*. http://wearcam.org/ece385/telepoin/, Abruf: 20. Jänner 2013

[44] MANN, Steve: Wearable Computing. In: Soegaard, Mads and Dam, Rikke Friis (eds.). "The Encyclopedia of Human-Computer Interaction, 2nd Ed.". Aarhus, Denmark: The Interaction Design Foundation. In: *Journal of Location Based Services* 5 (2011), September, Nr. 3-4, S. 121–137

[45] LAMMING, M. ; FLYNN, M. u. a.: Forget-me-not: Intimate computing in support of human memory. In: *Proc. FRIEND21, 1994 Int. Symp. on Next Generation Human Interface* Citeseer, 1994, S. 1–9

[46] WANT, R ; SCHILIT, B N. ; ADAMS, N I. ; GOLD, R ; PETERSEN, K ; GOLDBERG, D ; ELLIS, J R. ; WEISER, M: An overview of the PARCTAB ubiquitous computing experiment. In: *IEEE Personal Communications* 2 (1995), Nr. 6, S. 28–43

[47] MUÑIZ JR, Albert M. ; SCHAU, Hope J.: Religiosity in the Abandoned Apple Newton Brand Community. In: *Journal of Consumer Research* 31 (2005), März, Nr. 4, S. 737–747

[48] FELDMANN, S. ; KYAMAKYA, K. ; ZAPATER, A. ; LUE, Z.: An indoor Bluetooth-based positioning system: concept, implementation and experimental evaluation. In: *International Conference on Wireless Networks*, 2003, S. 109–113

[49] WAHLSTER, W. ; KRÖNER, A. ; HECKMANN, D.: SharedLife: towards selective sharing of augmented personal memories. In: *Reasoning, Action and Interaction in AI Theories and Systems* (2006), S. 327–342

[50] KERN, Nicky ; SCHIELE, Bernt ; SCHMIDT, Albrecht: Recognizing context for annotating a live life recording. In: *Personal and Ubiquitous Computing* 11 (2006), August, Nr. 4, S. 251–263

[51] BLUM, Mark ; PENTLAND, Alex S. ; TROSTER, Gerhard: InSense: Interest-Based Life Logging. In: *MultiMedia, IEEE* (2006), September, S. 1–9

[52] CHOUDHURY, Tanzeem ; HARRISON, Beverly ; HIGHTOWER, Jeffrey ; LEGRAND, Louis ; RAHIMI, Ali ; REA, Adam ; LANDAY, James A. ; LESTER, Jonathan ; WYATT, Danny ; HAEHNEL, Dirk ; BORRIELLO, Gaetano ; HEMINGWAY, Bruce ; KLASNJA, Predrag ". ; KOSCHER, Karl ; CONSOLVO, Sunny ; LAMARCA, Anthony: An embedded Activity Recognition system. In: *Pervasive Computing, IEEE* (2008), März, S. 1–10

[53] *Apple Newton MessagePad*. http://oldcomputers.net/apple-newton.html. Version: 1993, Abruf: 20. Jänner 2013

[54] *World Changing Gadgets gallery - Palm Pilot 1000 - 1996*. http://www.talktalk.co.uk/technology/galleries/view/technology/worldchanginggadgets/browse/304185, Abruf: 20. Jänner 2013

[55] REDDY, R. ; STCLAIR, G.: The million book digital library project. In: *Computer Science Presentation* (2001)

[56] *Rememex - MyLifeBits - YouTube*. http://www.youtube.com/watch?v=xbwwkXA92i4

[57] SCHILDT, Gerhard H.: *Satellitennavigation*. Lyk Informationstechnik, 2008. – ISBN 3950251804

[58] *The Google Maps Geolocation API - Google Maps API for Business — Google Developers*. https://developers.google.com/maps/documentation/business/geolocation/, Abruf: 20. Jänner 2013

[59] KING, T. ; HAENSELMANN, T. ; KOPF, S. ; EFFELSBERG, W.: Positionierung mit Wireless-LAN und Bluetooth. In: *Praxis der Informationsverarbeitung und Kommunikation* 29 (2006), Nr. 1, S. 9–17

[60] *Location Strategies | Android Developers*. http://developer.android.com/guide/topics/location/strategies.html, Abruf: 20. Jänner 2013

[61] *Nokia Research is courting partners and expanding Bluetooth as part of an initiative on indoor location-based services.* http://www.eetimes.com/electronics-news/4230993/Nokia-tweaks-Bluetooth-for-indoor-navigation, Abruf: 20. Jänner 2013

[62] *Nokia Research Shows Off Indoor Mapping*. http://thenokiablog.com/2011/11/29/nokia-research-indoor-mapping/, Abruf: 20. Jänner 2013

[63] YANG, J. ; JIANG, Y.G. ; HAUPTMANN, A.G. ; NGO, C.W.: Evaluating bag-of-visual-words representations in scene classification. In: *Proceedings of the international workshop on Workshop on multimedia information retrieval* ACM, 2007, S. 197–206

[64] SUNDARAM, H. ; CHANG, S.F.: Audio scene segmentation using multiple features, models and time scales. In: *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on* Bd. 6 IEEE, 2000, S. 2441–2444

[65] MIGLIORE, Davide A. ; MATTEUCCI, Matteo ; NACCARI, Matteo: A revaluation of frame difference in fast and robust motion detection. In: *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks* ACM, 2006, S. 215–218

[66] STRICKER, M. ; ORENGO, M.: Similarity of color images. In: *Proc. SPIE Storage and Retrieval for Image and Video Databases* Bd. 2420 San Diego, CA, 1995, S. 4

[67] DOHERTY, Aiden R. ; BYRNE, Daragh ; SMEATON, Alan F. ; JONES, Gareth J F. ; HUGHES, Mark: Investigating keyframe selection methods in the novel domain of passively captured visual lifelogs. In: *CIVR '08: Proceedings of the 2008 international conference on Content-based image and video retrieval*, ACM Request Permissions, Juli 2008

[68] DOHERTY, A.R. ; SMEATON, A.F. ; LEE, K. ; ELLIS, D.P.W.: Multimodal segmentation of lifelog data. In: *Large Scale Semantic Access to Content (Text, Image, Video, and Sound)* Centre de Hautes Etudes Internationale D'Informatique Documentaire, 2007, S. 21–38

[69] CONAIRE, C. ; BLIGHE, M. ; O'CONNOR, N.: Sensecam image localisation using hierarchical surf trees. In: *Advances in Multimedia Modeling, Springer* (2009), S. 15–26

[70] LOWE, David G.: Object recognition from local scale-invariant features. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on* Bd. 2 Ieee, 1999, S. 1150–1157

[71] BLIGHE, Michael ; DOHERTY, Aiden ; SMEATON, Alan F. ; O'CONNOR, Noel E.: Keyframe Detection in Visual Lifelogs. In: *ACM Proceedings of the 1st international conference on Pervasive Technologies Related to Assistive Environments* (2008), März, S. 1–2

[72] WANNOUS, H. ; DOVGALECS, V. ; MÉGRET, R. ; DAOUDI, M.: Place Recognition via 3D Modeling for Personal Activity Lifelog Using Wearable Camera. In: *Advances in Multimedia Modeling, Springer* (2012), S. 244–254

[73] XIONG, Ziyou ; RADHAKRISHNAN, R ; DIVAKARAN, A ; HUANG, T S.: Comparing MFCC and MPEG-7 audio features for feature extraction, maximum likelihood HMM and entropic prior HMM for sports audio classification. In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP'03)*, IEEE, 2003

[74] STEVE HODGES, James Srinivasan Alex Butler Gavin S.: SenseCam User Manual. (2012), November, S. 1–28

[75] GEMMELL, Jim ; BELL, Gordon ; LUEDER, Roger: MyLifeBits: a personal database for everything. In: *Communications of the ACM* 49 (2006), Nr. 1, S. 88–95

[76] GEMMELL, J. ; BELL, G. ; LUEDER, R. ; DRUCKER, S. ; WONG, C.: MyLifeBits: fulfilling the Memex vision. In: *Proceedings of the tenth ACM international conference on Multimedia* ACM, 2002, S. 235–238

[77] SMEATON, Alan F.: Content vs. Context for Multimedia Semantics: The Case of SenseCam Image Structuring Invited Keynote Paper. (2006), Oktober, S. 1–10

[78] *Everywhere I've Been: Data Portraits Powered by 3.5 years of data and 2.5 million GPS Points*. http://aaronparecki.com/, Abruf: 20. Jänner 2013

[79] NIJHUIS, J A G. ; TER BRUGGE, M H. ; HELMHOLT, K A. ; PLUIM, J P W. ; SPAANENBURG, L ; VENEMA, R S. ; WESTENBERG, M A.: Car License Plate Recognition with Neural Networks and Fuzzy Logic. In: *ICNN'95 - International Conference on Neural Networks*, IEEE, 1995, S. 2232–2236

[80] CHAUDHARI, Jayashri ; CHEUNG, Sen-ching S. ; VENKATESH, M V.: Privacy Protection for Life-log Video. In: *IEEE Workshop* (2007), Mai, S. 1–5

[81] VIOLA, Paul ; JONES, Michael: Rapid object detection using a boosted cascade of simple features. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on* Bd. 1 IEEE, 2001, S. I–511

[82] VEMURI, S. ; SCHMANDT, C. ; BENDER, W. ; TELLEX, S. ; LASSEY, B.: An audio-based personal memory aid. In: *UbiComp 2004: Ubiquitous Computing* (2004), S. 400–417

[83] HEALEY, J. ; PICARD, R.W.: Startlecam: A cybernetic wearable camera. In: *Wearable Computers, 1998. Digest of Papers. Second International Symposium on* IEEE, 1998, S. 42–49

[84] *Federal Communications Commission - OET Exhibits List*. https://apps.fcc.gov/oetcf/eas/reports/ViewExhibitReport.cfm?mode=Exhibits&calledFromFrame=N&application_id=287362&fcc_id=A4R-X1, Abruf: 31. Jänner 2013

[85] *Some questions on Google Project Glass*. http://counternotions.com/2012/05/31/projectglass, Abruf: 20. Jänner 2013

[86] *Besttechinfo: Google's Project Glass Is Focused On Creating Futuristic Augmented Reality Glasses*. http://goo.gl/Qhf13, Abruf: 20. Jänner 2013

[87] AIZAWA, K. ; ISHIJIMA, K. ; SHIINA, M.: Summarizing wearable video. In: *Image Processing, 2001. Proceedings. 2001 International Conference on* Bd. 3 IEEE, 2001, S. 398–401

[88] TANCHAROEN, D. ; PUANGPAKISIRI, W. ; YAMASAKI, T. ; AIZAWA, K.: Life Log Platform for Continuous and Discrete Recording and Retrieval of Personal Media. In: *IEICE technical report. Image engineering* (2007), S. 123–128

[89] LO, Raymond Chun H.: CyborGlogger: A Computational Framework for Real-time CyborGlogging. (2010)

[90] *iGlogger mobile, share what you see.* http://glogger.mobi, Abruf: 20. Jänner 2013

[91] HUI HSIEN WU: Development and evaluation of a Blackberry-Based Wearable Mobility Mobility Monitoring System. In: *Ottawa, Canada* (2012), Dezember, S. 1–168

[92] KIM, Sang-Hyun ; RYOO, Dong-Wan ; BAE, Changseok: *Wearable Healthcare Gadget for Life-Log Service Based on WPAN*. Springer Berlin Heidelberg, 2007

[93] MAKINO, Y. ; MURAO, M. ; MAENO, T.: Life log system based on tactile sound. In: *Haptics: Generating and Perceiving Tangible Sensations* (2010), S. 292–297

[94] FROEHLICH, Jon ; CHEN, Mike Y. ; CONSOLVO, Sunny ; HARRISON, Beverly ; LANDAY, James A.: MyExperience: A System for In situ Tracing and Capturing of User Feedback on Mobile Phones . In: *the 5th international conference*. New York, New York, USA : ACM Press, 2007, S. 57–70

[95] *The MyExperience tool*. `http://myexperience.sourceforge.net/`, Abruf: 20. Jänner 2013

[96] RAWASSIZADEH, R. ; TOMITSCH, M. ; WAC, K. ; TJOA, A.M.: UbiqLog: a generic mobile phone-based life-log framework. In: *Personal and Ubiquitous Computing, Springer* (2012), S. 1–17

[97] *ViconRevue- memories for life*. `http://viconrevue.com/`, Abruf: 20. Jänner 2013

[98] *Autographer- The worlds first intelligent, wearable camera*. `http://www.autographer.com`, Abruf: 20. Jänner 2013

[99] *Memoto Lifelogging Camera. A tiny, automatic camera and app that gives you a searchable and shareable photographic memory.* `http://memoto.com/`, Abruf: 20. Jänner 2013

[100] *Lifelapse- Relive your life*. `http://www.lifelapse.com/`, Abruf: 20. Jänner 2013

[101] *Replaymyday the automatic diary for your iPhone*. `http://replaymyday.info/`, Abruf: 20. Jänner 2013

[102] *funf - An Open Source Sensing Framework.* `http://funf.org`, Abruf: 20. Jänner 2013

[103] LEMKIN, M ; BOSER, B E.: A three-axis micromachined accelerometer with a CMOS position-sense interface and digital offset-trim electronics. In: *IEEE Journal of Solid-State Circuits* 34 (1999), April, Nr. 4, S. 456–468

[104] HULSING, R.H. u. a.: *Micromachined rate and acceleration sensor*. 1993. – US Patent 5,241,861

[105] Bosch Sensortec: BMA150 Digital, triaxial acceleration sensor. (2008), Oktober, S. 1–56

[106] *gesturedroid - Android gesture detection - Google Project Hosting*. `http://code.google.com/p/gesturedroid`, Abruf: 20. Jänner 2013

[107] MOBERG, Anders ; BERGSTRÖM, Hans ; RUIZ KRIGSMAN, Josefin ; SVANERED, Ola: Daily air temperature and pressure series for Stockholm (1756–1998). In: *Climatic Change* 53 (2002), Nr. 1/3, S. 171–212

[108] BERBERAN-SANTOS, M.N. ; BODUNOV, E.N. ; POGLIANI, L.: On the barometric formula. In: *American Journal of Physics* 65 (1997), S. 404

[109] DEPPNER, HEINZ G.: Drehratenmessgeber. In: *STN Atlas Elektronik, Bremen* (1999), Dezember, S. 10–15

[110] WALTHER, T. ; WALTHER, H.: *Was ist Licht?: Von der klassischen Optik zur Quantenoptik*. Bd. 2122. CH Beck, 1999. – ISBN 3406447228

[111] SCHUMACHER, K. ; WALLRABE, U. ; MOHR, J.: *Design, Herstellung und Charakterisierung eines mikromechanischen Gyrometers auf der Basis der LIGA-Technik*. Forschungszentrum Karlsruhe, 1999

[112] YAZDI, N ; AYAZI, F ; NAJAFI, K: Micromachined inertial sensors. In: *Proceedings of the IEEE* 86 (1998), Nr. 8, S. 1640–1659

[113] PAUS, H.J.: Physik in Experimenten und Beispielen, Hanser Verlag, 2007. – ISBN 3446411429, S. 447

[114] *Eclipse - The Eclipse Foundation open source community website.* `http://www.eclipse.org/`, Abruf: 20. Jänner 2013

[115] *Apache Ant*. `http://ant.apache.org/`, Abruf: 20. Jänner 2013

[116] *ImageMagick: Command-line Tools: Mogrify*. `http://www.imagemagick.org/www/mogrify.html`, Abruf: 20. Jänner 2013

[117] *Sips OS X Manual Page*. `https://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man1/sips.1.html`, Abruf: 20. Jänner 2013

[118] *Portecle: Home*. `http://portecle.sourceforge.net/`, Abruf: 20. Jänner 2013

[119] BARLA, A. ; ODONE, F. ; VERRI, A.: Histogram intersection kernel for image classification. In: *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on* Bd. 3 IEEE, 2003, S. III–513

[120] FINLAYSON, Graham ; SCHIELE, Bernt ; CROWLEY, James: Comprehensive colour image normalization. In: *Computer Vision—ECCV'98* (1998), S. 475–490

[121] *The Google Elevation API*. `https://developers.google.com/maps/documentation/elevation`, Abruf: 20. Jänner 2013

Tabellenverzeichnis

3.1 Vergleich der einzelnen Systeme. 42

5.1 Asynchrone Events, zugehörige *Src*, *writeDb*, *dependency* und *Subscriber-Klassen*. 70
5.2 Synchrone Events, zugehörige *Src*, *writeDb*, *dependency* und *Subscriber-Klassen*. . 70

6.1 Verwendete Sensoren und zugehörige Buchstaben. 92
6.2 Auswahl der Datenquelle und korrespondierende Datentypen. 101