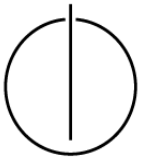
FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Analyse zur Identifizierung von Benutzern/Geräten anhand verschiedener Faktoren und Integration in ein Framework

Norbert Schmidbartl



FAKULTÄT FÜR INFORMATIK

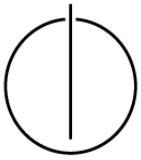
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Analyse zur Identifizierung von Benutzern/Geräten anhand verschiedener Faktoren und Integration in ein Framework

Analysis to identify users/devices based on different criteria and integration into a framework

Bearbeiter:	Norbert Schmidbartl
Aufgabensteller:	Prof. Dr. Uwe Baumgarten
Betreuer:	Nils T. Kannengießer, M.Sc.
Abgabedatum:	15.02.2015



Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Garching, den 15.02.2015

Norbert Schmidbartl

Zusammenfassung

Diese Arbeit beschäftigt sich mit der Analyse, Implementierung und Evaluation von Methoden, die der Identifizierung von Geräten und Nutzern dienen, sowie deren Integration in ein Cloud-basiertes Android-Framework. Android selbst bietet nur minimalen Schutz gegen Urheberrechtsverletzungen. Durch die Identifizierung von Nutzern und Geräten können bestehende Schutzmechanismen für Anwendungen ergänzt und zudem sensible Nutzerdaten geschützt werden. Der Einsatz von Identifikationsverfahren ermöglicht individuellere, benutzerfreundlichere Anwendungs-Inhalte, welche den Gewohnheiten eines Nutzers entgegenkommen. Die Grundidee des Frameworks besteht darin, spezifische Informationen über Geräte und Nutzer zu extrahieren und diese anschließend mit eindeutigen Identifikationsnummern zu verknüpfen. Es wird ein breites Spektrum an Informationsquellen analysiert, wie beispielsweise Hardware-Eigenschaften, Sensordaten, Positionsdaten, Kameradaten, Mikrofondaten sowie Nutzer- und Systemdateien. Zudem werden signifikante Eigenschaften und Verhaltensweisen eines Nutzers untersucht, um benutzerspezifische Profile für die spätere Verifikation und Identifizierung zu erstellen. Die Ergebnisse der einzelnen Identifizierungsmethoden werden gewichtet und kombiniert, um so ein aussagekräftiges Gesamtergebnis zu erhalten. Vor allem in Situationen, in denen einzelne Informationsquellen nicht zur Verfügung stehen, oder essentielle Nutzer- oder Gerätedaten plötzlich geändert oder sogar gelöscht werden, können sich durch die Kombination der Methoden Vorteile ergeben. Die Evaluation hat gezeigt, dass das Android-Framework in der Lage ist, Nutzer- und Geräte sowohl kurz- als auch langfristig zuverlässig zu identifizieren.

Abstract

This thesis describes the analysis, implementation, and verification of methods—all of which can be used for user as well as device identification. These are also used for integrating such identifications into a cloud-based Android framework. Android itself provides only minimal protection against copyright infringement. Passive user and device identification can help to enhance the existing protection mechanisms for mobile applications as well as sensitive user data. Additionally, identification processes can be used to create more user-friendly and individualized Android applications to meet individual user habits. The basic idea of the framework is to extract specific user and device information, and also to link the aggregated data to unique identification numbers. The identification methods analyze a wide range of information on mobile devices, such as specific hardware specifications, sensor data, location data, camera and microphone input, and user and system-specific data. The framework examines information on the behavior and characteristics of devices and users to create distinguishable profiles for later identification. The weighted and combined results of the identification methods can be used to calculate the overall result of the identification process. The combination of methods leads to crucial advantages in cases where common device or user information is not available, is altered, or is not reliable. The evaluation has shown that the Android framework is able to reliably identify users and devices based on characteristic and persistent sources of information.

Inhaltsverzeichnis

1. Einführung und Hintergründe

Mobile Geräte sind mittlerweile aus dem alltäglichen Leben nicht mehr wegzudenken. In den Jahren 2013 bis 2014 ist die Anzahl an Menschen, die in Deutschland ein Smartphone nutzen, um 25 Prozent gestiegen [1]. Prognosen gehen davon aus, dass im Jahre 2016 mehr als 2 Milliarden Menschen weltweit ein Smartphone nutzen werden [2]. Es ist zudem zu beobachten, dass mobile Geräte auch bei Softwareentwicklern immer beliebter werden. So entwickeln bereits mehr als 50% der Software-Firmen weltweit Anwendungen für Smartphones oder Tablets [3]. Mit einem Zuwachs von über 700 000 Entwicklern von 2013 auf 2014 konnten somit im Jahre 2014 knapp 9 Millionen Software-Entwickler für mobile Anwendung gezählt werden [3]. Das mobile Betriebssystem Android gilt hierbei als das am meisten genutzten mobile Betriebssystem weltweit, mit einem Marktanteil von etwa 85% [4].

Smartphones und Tablets stehen dem Menschen heutzutage in unzähligen Situationen als mobile Helfer zur Seite, und lösen ehemals komplizierte Probleme des Alltags. Informationen jeglicher Art sind dank des mobilen Internets auf Smartphones und Tablets jederzeit abrufbar. Ob als Navigationsgerät, als mobile Kommunikationszentrale, beispielsweise für E-Mails oder soziale Netzwerke, als E-Book-Reader, MP3-Spieler, tragbare Spielekonsole oder auch als Fernsehgerät mit On-Demand-Funktionalität, die Anzahl an Anwendungen, die ein handelsübliche Smartphone oder Tablet unterstützt, ist in den letzten Jahren stetig gestiegen. Immer mehr Menschen wissen diesen Mehrwert zu schätzen, wobei sich die Nutzerpräferenzen im Laufe der Zeit entsprechend gewandelt haben [5].

Für Softwareentwickler entstehen mit der zunehmenden Weiterentwicklung und Verbreitung von Android-Geräten sowie der damit einhergehenden Veränderung von Geschäftsmodellen neue Herausforderungen und Probleme. Sicherheitsrelevante Aspekte gewinnen bei der Entwicklung mobiler Anwendungen immer mehr an Bedeutung [6]. Diese Arbeit beschäftigt sich diesbezüglich mit den Möglichkeiten der zuverlässigen Nutzer- und Geräteidentifizierung, und der Entwicklung eines Android-Frameworks, welches, wie in den nachfolgenden Kapiteln dargestellt, sowohl Entwicklern als auch Nutzern mobiler Anwendungen entgegenkommen kann.

1.1 Motivation

Die Identifizierung von Nutzern und Geräten stellt für eine ganze Reihe an Problemstellungen eine Möglichkeit dar, den Umgang mit mobilen Anwendungen noch sicherer und benutzerfreundlicher zu gestalten. Im Folgenden wird erläutert, welche Ausgangssituationen hinter der Motivation für die Entwicklung eines Frameworks für die Identifizierung von Nutzern und Geräten stecken.

Ein Problem, mit dem sich Android-Entwickler im Allgemeinen auseinander setzen müssen, sobald ein kommerzielles Produkt veröffentlicht werden soll, ist der unzureichende Kopierschutz von Anwendungen, den Android standardmäßig bietet [7]. So wurde berichtet, dass bei mehr als 90% der registrierten Installationen einer kostenpflichtigen Anwendung der Kopierschutz umgangen werden konnte [7]. Es handelt sich hierbei nicht um Einzelfälle, raubkopierte Android-Anwendungen stellen ein grundlegendes Problem dar. Statistiken zeigen, dass der standardmäßige Schutz von Anwendungen auf Android-Geräten nach wie vor vernachlässigt wird [7]. Der unzureichende Kopierschutz ist allerdings nur ein Teil der sicherheitsrelevanten Schwachstellen des Android-Betriebssystems. Einer der Gründe für die Sicherheitsmängel dürfte sein, dass Android auf Linux basiert, und somit neben den Stärken des Betriebssystems auch einige Schwächen mit sich bringt. Ein Beispiel hierfür wäre die Anfälligkeit für bestimmte Linux-Kernel-Exploits [8]. Auch von dem kürzlich bekannt gewordenen "DoubleDirect"-Problem sind unter anderem Geräte mit dem Android-Betriebssystem betroffen [9]. Android-Anwendungen basieren zudem in der Regel auf Java, weshalb sich ungeschützte Android-Anwendungen ohne Probleme dekompilieren und somit auch modifizieren lassen [8]. So kann beobachtet werden, dass

zu bestimmten Anwendung immer wieder unautorisierte, leicht modifizierte Versionen auftauchen, die nicht von den ursprünglichen Entwicklern stammen, und den ursprünglichen Kopierschutz aushebeln [10].

Softwarepakete wie beispielsweise „ProGuard“ können helfen, die Dekompilierung von Java-Anwendungen zu beschränken [8]. Eine Möglichkeit, diese Sicherheitsmechanismen sinnvoll zu unterstützen, ist die Identifizierung von Nutzern und Geräten, die mit einer Anwendungsinstallation in Verbindung gebracht werden können. Die Identifizierung von Nutzern und Geräten ermöglicht es Anwendungen, auf unrechtmäßige Installationen oder sonstige Urheberrechtsverletzungen angemessen zu reagieren. Erst durch die eindeutige und zuverlässige Zuordnung von Anwendungsinstallationen zu einer Einzelperson oder einer Gruppe von Nutzern und Geräten werden Lizenzierungssysteme sinnvoll und gerecht.

Die eindeutige Zuordnung eines spezifischen Nutzers zu einer Anwendungsinstallation auf einem Gerät kann aber nicht nur aus der Sicht der Android-Entwickler sinnvoll sein. Es liegt auch im Interesse der Anwendungsnutzer, dass Geräte und Nutzer in bestimmten Szenarien von Anwendungen identifiziert werden können. So kann beispielsweise davon ausgegangen werden, dass mobile Geräte in Zukunft vermehrt sicherheitsrelevante Aufgaben übernehmen werden [6]. Auch heutzutage schon werden Smartphones und Tablets für Online-Banking oder die Verwaltung vertraulicher Daten eingesetzt, und mobile Bezahlsysteme gewinnen immer mehr an Bedeutung [6]. Android-Anwendungen, die sicherheitsrelevante Funktionen anbieten, sollten eine gewisse Verbindlichkeit sowie einen umfassenden Zugangsschutz garantieren können, ohne die Benutzerfreundlichkeit mehr als nötig einzuschränken. Die passive Identifizierung von Nutzern kann helfen, die Benutzung von mobilen Geräten einfacher und benutzerfreundlicher zu gestalten. So können Authentifizierungsvorgänge, die das aktive Eingreifen der Nutzer erfordern, durch passive Identifizierungsvorgänge unterstützt und vereinfacht werden. Ein Nutzer müsste sich beispielsweise immer nur dann aktiv authentifizieren, wenn feststellt wird, dass ein Nutzerwechsel stattgefunden hat. Die Anzahl an Authentifizierungsvorgängen könnte dadurch deutlich reduziert werden. Zudem existieren nach wie vor bestimmte Szenarien, in denen bisherige Authentifizierungsverfahren versagen, da sie im Grunde immer nur auf einer einzigen Informationsquelle basieren. Das Anliegen dieser Arbeit ist es, ein breites Spektrum an Verfahren zu analysieren und zu implementieren, die als Ganzes eine deutlich höhere Zuverlässigkeit bieten können als bisherige Methoden, die voneinander isoliert ausgeführt werden.

Ein weiteres Problem, welches immer mehr Android-Nutzer betrifft, ist der Diebstahl von mobilen Geräten [11]. Viele Nutzer wissen auch aus eigener Erfahrung, dass der Diebstahl eines mobilen Geräts neben dem Wertverlust vor allem wegen der verlorenen gespeicherten Daten sehr ärgerlich sein kann. Passive Identifizierungsmethoden, durch die vor allem kurzfristig stattfindenden Nutzerwechsel erkannt werden können, ermöglichen die vorsorgliche Sperrung von Smartphones oder Tablets bei Verdacht. Dadurch können persönliche Nutzerdaten vor Fremdzugriff geschützt werden. Zudem lässt sich durch geeignete Identifizierungsmethoden der Diebstahl eines Geräts gezielt zurückverfolgen. Zu diesem Zweck bieten sich vor allem biometrische Verfahren an, die beispielsweise passiv feststellbaren Verhaltensänderungen analysieren. Falls unerwarteterweise ein unbekannter Nutzer ein Gerät bedient, lassen sich durch diese Methoden entsprechende Schutzmaßnahmen einleiten. Auch die nachträgliche Zuordnung von Nutzer und Geräten, beispielsweise im Bereich der mobilen IT-Forensik, kann in diesem Zusammenhang sinnvoll sein [12]. Nutzer und Geräte sollten in diesen Szenarien im Zweifelsfall auch längerfristig eindeutig identifiziert werden können.

Durch die zunehmende Nutzung von Smartphones und Tablets werden immer mehr Informationen des alltäglichen Lebens auf mobile Geräte ausgelagert, Smartphones und Tablets werden somit persönlicher

und individueller. Anwendungen können die persönlichen Präferenzen eines Nutzers zumindest zum Teil erfassen und sich inhaltlich an die Gewohnheiten und Wünsche des Nutzers anpassen. Sogenannte Recommender-Systeme wählen anhand zuvor gesammelter Informationen über den Nutzer neue, passende Inhalte, die anschließend als Vorschläge präsentiert werden können [13]. Diese Systeme können beispielsweise bei Online-Shops oder Musik- und Filmanwendungen eingesetzt werden. Die langfristige Identifizierung eines bestimmten Nutzers und dessen Gewohnheiten ist hierbei Grundvoraussetzung, um die Anwendung individueller und benutzerspezifischer gestalten zu können [13].

1.2 Zielsetzung

Ziel dieser Arbeit ist die Analyse von Möglichkeiten zur Identifizierung von Benutzern und Geräten anhand verschiedener Faktoren, und die Implementierung eines Android-Frameworks, welches die Ergebnisse der vorhergehenden Untersuchungen umsetzt. Die Performanz und Zuverlässigkeit der implementierten Methoden und des Frameworks als Ganzes wurden zudem im Rahmen dieser Arbeit evaluiert. Das Android-Framework muss eine ganz Reihe an Anforderungen erfüllen, um für die Identifizierung von Nutzern und Geräten sinnvoll eingesetzt werden zu können. Nachfolgend werden die wichtigsten Zielsetzungen, die bei der Implementierung und Evaluierung des Frameworks beachtet wurden, erläutert.

Das Framework soll, wie bereits beschrieben, sowohl für die Nutzer- als auch Geräteidentifizierung verwendet werden können. Im Rahmen der Arbeit soll untersucht werden, welche Informationsquellen sich prinzipiell für die Identifizierung eignen, und ob sich die jeweils gewonnenen Daten der Nutzer- oder der Geräteidentifizierung zuordnen lassen.

Eine weitere Anforderung an die vorgestellten Methoden ist, dass die Nutzer nicht aktiv mit dem Framework interagieren müssen, um den Identifizierungsvorgang durchzuführen. Das Framework soll Nutzer und Geräte wiedererkennen können, ohne dass der Nutzer aktiv eingreifen muss. Es soll somit eine passiven Nutzer- und Geräteidentifizierung angeboten werden, und das Framework wurde für diesen Einsatzzweck optimiert.

Es zeigt sich, dass sich Identifizierungsmethoden grundsätzlich in zwei Kategorien einordnen lassen: Ein Teil der Verfahren ermöglicht eine kurzfristige Identifizierung von Nutzern, und es besteht die Möglichkeit, dass Nutzer und Geräte zeitnahe erkannt werden können. Es existieren allerdings auch Methoden, die vor allem für längerfristige Nutzer- und Geräteidentifizierungen eingesetzt werden können. In Kapitel 3 wird näher auf die entsprechenden Eigenschaften der vorgestellten Methoden eingegangen. Zielsetzung des Frameworks ist es, beide Typen, also sowohl kurz- als auch längerfristige Identifizierungsmethoden zu kombinieren, um mithilfe der spezifischen Stärken der jeweiligen Methoden ein möglichst breites Spektrum an Anwendungsmöglichkeiten abzudecken.

Das Android-Framework soll zudem die Identifizierung nicht nur isoliert auf einem Geräte ausführen können, sondern die Menge aller Daten, die von unterschiedlichen Anwendungsinstallationen gesammelt werden, zentral und ganzheitlich analysieren. Das Framework soll somit nicht nur die Identität eines Nutzers oder Geräts verifizieren können, sondern diese auch in einer größeren Menge an Daten von unterschiedlichen Nutzern und Geräten identifizieren (one-to-many). Die eingehenden Daten, die für die Identifizierung verwendet werden können, werden deshalb zentral auf einem Server mit allen zuvor gesammelten Referenzdaten gespeichert und verglichen, und passende Datenpaare werden extrahieren.

Eine weitere Anforderung an die Implementierung war, dass sensible Nutzerdaten durch das Framework so gut wie möglich geschützt werden sollen. Durch den Einsatz von Verschlüsselungs- und Pseudonomisierungstechniken soll die unrechtmäßige Extraktion von persönlichen Daten aus dem Gesamtsystem unmöglich gemacht werden.

Des Weiteren soll das Framework auf handelsüblichen Android-Smartphones und -Tablet lauffähig sein. Um den Einsatz des Frameworks auf einem großen Auswahl an Android-Geräten zu ermöglichen, wurde das Projekt so konzipiert, dass Features, die von bestimmten Geräten nicht unterstützt werden, bei Bedarf von diesen deaktiviert werden, und somit keine Fehlermeldungen bzw. Anwendungsabstürze verursachen können. Das Framework als Ganzes soll dabei aber trotzdem einsatzfähig bleiben. Um diese Anforderung zu realisieren, greift das Framework auf eine Vielzahl möglichst unabhängiger Verfahren zurück. Es hat sich gezeigt, dass sich durch die Kombination der Methoden die Ausfallsicherheit, die Performanz sowie die Zuverlässigkeit des Identifizierungsvorgangs steigern lassen. Die Implementierung soll hierbei so umgesetzt werden, dass das Android-Framework bei Bedarf um neue Methoden erweitert werden kann. Im Verlauf der Evaluation wurde das Framework für den tatsächlichen Einsatz auf mobilen Geräten optimiert, und die Implementierung wurde an unterschiedliche Hardware-Konfigurationen und Android-Versionen angepasst.

1.3 Aufbau der Arbeit

Die Masterarbeit untergliedert sich in insgesamt sechs Kapitel, die im Folgenden kurz vorgestellt werden.

Einführung und Hintergründe

In Kapitel 1 wird zu Beginn auf die Motivation und Zielsetzung, die hinter dieser Arbeit stehen, eingegangen. Nachfolgend werden die technischen Eigenschaften und aktuellen Umstände, die das mobile Betriebssystem Android betreffen, näher erläutert. In diesem Kapitel sollen wichtige Grundlagen und Hintergrundwissen vermittelt werden, die dem Verständnis der Arbeit dienen.

Ablauf der Datenerfassung und Identifizierung

In Kapitel 2 wir die Vorgehensweise und das Gesamtkonzept des Frameworks für die Identifizierung von Nutzern und Geräten vorgestellt. In diesem Abschnitt soll ein kurzer Überblick über die prinzipielle Arbeitsweise des Frameworks vermittelt werden.

Mögliche Optionen zur Erfassung und Identifizierung von Benutzern und Geräten

Kapitel 3 beschäftigt sich detailliert mit einer Reihe von Identifizierungsmethoden und möglichen Informationsquellen. Es wird erläutert, wie sich diese Methoden für die Identifizierung von Nutzern oder Geräten einsetzten lassen, und deren tatsächliche Verwendbarkeit für das Android-Framework näher untersucht.

Entwicklung eines Android-Frameworks

Kapitel 4 geht auf die Implementierung des Android-Frameworks ein, und beschreibt, wie die vorgestellten Identifizierungsmethoden in das Framework eingebettet wurden. Zudem wird auf die konkrete Umsetzung des Java Servlets auf der Serverseite eingegangen.

Evaluierung des Frameworks

In Kapitel 5 wir im Detail dargestellt und erläutert, welche Ergebnisse im Rahmen der Evaluation mithilfe einer Beispiel-App erzielt werden konnten.

Zusammenfassung und Ausblick

Kapitel 6 umfasst eine abschließende Zusammenfassung der Erkenntnisse, die in dieser Arbeit gewonnen wurden, sowie einen kurzen Ausblick auf zukunftsrelevante Themen.

Eine CD, auf der eine vollständige, lauffähige Version der Implementierung des Android-Frameworks und des Java Servlets, sowie der Programmcode der Beispiel-App und der Visualisierungs-Anwendung enthalten ist, wurde dieser Arbeit beigelegt.

1.4 Android

1.4.1 Hintergründe und Nutzerzahlen zu Android

Das Android-Betriebssystem wurde im Oktober 2003 von der Android, Inc. und dessen Mitbegründer, Andy Rubin, ins Leben gerufen und im August 2005 von Google Inc. gekauft [14]. Android ist ein quelloffenes, frei verfügbares System und gilt als das Hauptprodukt der Open Handset Alliance, ein aus 87 Hard- und Softwareunternehmen bestehendes Konsortium, welches die weitere Entwicklung von Android vorantreibt und steuert [15].

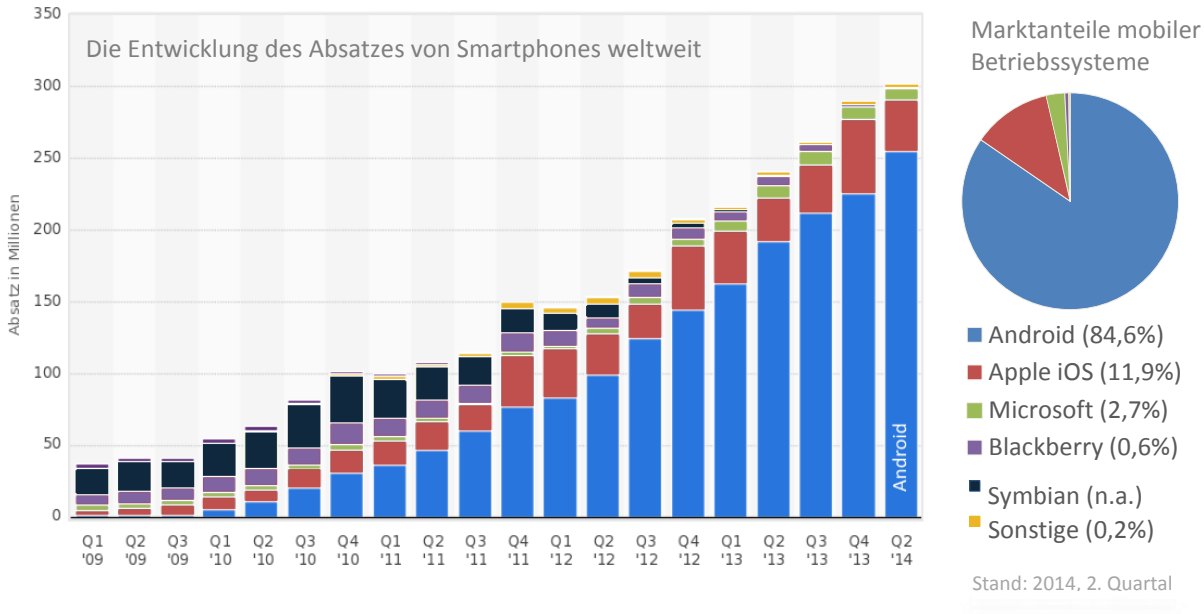


Abbildung 1: Entwicklung des Smartphone-Absatzes [16], Marktanteile mobiler Betriebssysteme [4]

Das auf dem Linux-Kernel basierende System, das vor allem für mobile Touchscreen-Geräte konzipiert wurde, ist mittlerweile das am meisten genutzte mobile Betriebssystem weltweit, wie in Abbildung 1 dargestellt [4]. Im zweiten Quartal 2014 waren auf 84,6 Prozent aller ausgelieferten Smartphones Android vorinstalliert, womit der Marktanteil im Vergleich zum Vorjahresquartal um 33,6 Prozent gesteigert werden konnte [4]. Android übertrifft mittlerweile die zusammengerechneten Verkaufszahlen der Betriebssysteme iOS, Mac OS X und Windows [17].

Tabelle 1: Verbreitung der jeweiligen Android-Versionen [18] , Stand: Januar 2015

Codename	Version	API	Anteil
Lollipop	5.0	21	1,6%
KitKat	4.4	19	39.7%
Jelly Bean	4.1.x - 4.3	16-18	44,5%
Ice Cream Sandwich	4.0.3 - 4.0.4	15	6.4%
Gingerbread	2.3.3 - 3.3.7	10	7.4%
Froyo	2.2	8	0.4%

Seitdem das erste Android-Smartphone im Oktober 2008 auf den Markt kam, sind zahlreiche Updates für Android erschienen. Bisherige Fehler wurden beseitigt, und der Funktionsumfang des Betriebssystems wurde erweitert. Die aktuelle Version, Android 5.0, erhielt die Bezeichnung „Lollipop“, und ist seit Oktober 2014 verfügbar [19]. Da einige Hersteller modifizierte Versionen von Android auf ihre Geräten vorinstallieren und die Entwicklung im Bereich der Hardware für mobile Geräte in einem enormen Tempo

voranschreitet, müssen Teile der Software für jedes Update neu angepasst oder ersetzt werden. Die Bereitstellung neuer Android-Versionen wird dadurch für einige Geräte erheblich verzögert [20]. Tabelle 1 zeigt, welchen Nutzeranteil die Android-Versionen jeweils haben. Die größte Verbreitung findet noch immer Android 4.1.x, 4,2.x, sowie 4.3, Codename Jelly Bean“, mit insgesamt 44,5 Prozent (Stand: Januar 2015), gefolgt von Android 4.4, „KitKat“, mit 39.7 Prozent. Android-Versionen vor 2.2 oder einem Anteil von weniger als 0,1% wurden bei der Datenerhebung nicht beachtet. [18]

Bei der Entwicklung einer App kann auf essentielle Android-Komponenten zugegriffen werden, die als Bausteine die grundlegende Struktur und das Verhalten einer Anwendung mitbestimmen. Zudem werden durch diese Komponenten Schnittstellen und Eintrittspunkte für das System definiert. Es existieren insgesamt vier dieser essentiellen Bausteine: Activities, Services, Broadcast-Receivers und Content-Providers [21]. Da diese Bausteine für die Entwicklung und das Verständnis des Android-Frameworks eine Grundvoraussetzung darstellen, findet sich im Glossar dieser Arbeit Erklärungen zu den grundsätzlichen Funktionsweisen der einzelnen Komponenten.

1.4.2 Die Android - Architektur

Die Architektur von Android lässt sich in vier aufeinander aufbauende Schichten untergliedern. Dazu zählen die Applikationsschicht (oberste Schicht), das Application-Framework, die Android Laufzeitumgebung, der Dalvik Virtual Machine bzw. ab Android-Version 5.0 die Android Runtime (ART), die nativen Bibliotheken sowie in der untersten Schicht der Linux-Kernel. Im Folgenden wird kurz auf die unterschiedlichen Schichten des Betriebssystems eingegangen [22] :

Linux Kernel

Der Linux Kernel stellt die unterste Schicht des Android-Software-Stacks dar. Zu Beginn wurde noch der Linux-Kernel 2.6 verwendet, ab Android 4.x kommt ein 3.x-Kernel zum Einsatz. Die Hardware-Abstraktion für die restlichen Softwarekomponenten wird von dieser Schicht bereitgestellt. Die Gerätetreiber des Systems sind somit Teil des Kernels, wodurch Android eine der Hauptstärken Linux-basierter Systeme übernimmt. So sind beispielsweise Treiber für die Kamera, Wireless Local Area Network (WLAN), Bluetooth, USB, Flash Memory und die Sensoren enthalten. Schnittstellen zur Netzwerkkommunikation sowie die Prozess-, Speicher- und Energieverwaltung sind auch Teil des Linux-Kernels. [22]

Laufzeitumgebung und native Bibliotheken

Der Jede Anwendung, die auf einem Android-Gerät ausgeführt wurde, lief bis einschließlich Android 4.4 in einer eigenen Instanz der Dalvik-Virtual-Machine (Dalvik-VM). Die Dalvik-VM basiert auf der Standard-Java-VM, wurde aber von Google für den Einsatz auf mobilen Geräten optimiert. Die Laufzeitumgebung lässt sich in die Schicht, die auf dem Linux-Kernel aufbaut, einordnen. [22] Ab Android 5.0, „Lollipop“, kommt anstelle der Dalvik-VM die sogenannte Android Runtime (ART) zum Einsatz. Im Gegensatz zur Dalvik-VM wird der Bytecode nicht erst während der Ausführung in den native Befehlssatz umgewandelt (Just-In-Time), sondern bereits bei der Installation einer Anwendung (Ahead-Of-Time). Die Dauer eines Anwendungsstarts sowie die Ausführgeschwindigkeit werden dadurch optimiert, wobei allerdings dadurch der Speicherbedarf einer Anwendung steigt.

Die Kern-Bibliotheken basieren auf C/C++ und Java, und umfassen Bibliotheken der Laufzeitumgebung, klassische Java-Libraries sowie weitere Android-typische Bibliotheken. Die Java-basierten Bibliotheken stellen eine Sammlung von Programmierschnittstellen (APIs) dar, mithilfe derer Android-Entwickler auf grundlegende Funktionalitäten des Systems zuzugreifen können. Beispiele für diese Art von Bibliotheken sind android.app (Application-Model), android.net (Netzwerkkommunikation), android.hardware (Zugriff auf Hardware-Komponenten, wie zum Beispiel den Beschleunigungs- und Lichtsensoren) und

android.database (Zugriff auf Datenbanken), sowie viele weitere [22]. Diese Java-basierten Schnittstellen dienen hauptsächlich dazu, den Entwicklern die Funktionalitäten der in C/C++ verfassten Bibliotheken in Java zugänglich zu machen. Die in C/C++ geschriebenen Komponenten hingegen erledigen den eigentlichen Teil der Arbeit, wie zum Beispiel die Secure-Socket-Layer-Kommunikation, das Datenbankmanagement, Audio- und Video-Wiedergabe, Grafikoperationen, sowie viele andere Aufgaben. Falls ein direkter Zugriff in Java-Code auf Bibliotheken, die nicht in Java verfasst wurde, erfolgen soll, kann zudem das Java Native Interface (JNI) bzw. das Android Native Development Kit (NDK) verwendet werden. [22]

Application - Framework

Das Application-Framework, das eine Schicht über der Laufzeitumgebung einzuordnen ist, enthält eine Sammlung von essentiellen Services, und stellt damit eine Umgebung dar, in der die Android-Anwendungen (Apps) laufen und verwaltet werden können. Die Wiederverwendbarkeit der Anwendungskomponenten, sowie deren Austauschbarkeit bzw. Ersetzbarkeit wird durch das Application-Framework ermöglicht. [22] Zu den Kernelementen des Frameworks gehören unter anderem der Content-Provider (erlaubt den Austausch von Daten zwischen Anwendungen), der Activity-Manager (verwaltet Activity-Stack und Lebenszyklus der Anwendungen), der Package-Manager (liefert Informationen über installierte Apps), der Telephony-Manager (stellt Dienste für Telefonie und Informationen zum Mobilfunkanbieter bereit), der Sensor-Manager (liefert Sensordaten), der Location-Manager (liefert Positionsdaten), der Resource-Manager (dient der Verwaltung von Ressourcen), das View-System (Sammlung von Views), sowie der Notifications-Manager (für Meldungen zuständig). [22]

Applikationsschicht

Die Applikationsschicht stellt die oberste Schicht der Android-Architektur dar, und umfasst im Grunde alle Softwarekomponenten, die der Anwender auf seinem Android-Gerät tatsächlich sieht. Dazu zählen die nativen Applikationen, die von der jeweiligen Android-Implementierung bereitgestellt werden, wie zum Beispiel der Webbrowser, das Mailprogramm, der Kalender, die Kontakte und Telefonfunktionen. Zudem werden auch alle Apps, die von Drittanbietern angeboten werden, zu der Applikationsschicht gezählt. [22]

1.5 Java Servlets

Servlets sind plattformunabhängige, serverseitige Software-Komponenten, die in Java verfasst werden. Sie werden als Teilkomponente innerhalb eines Webservers ausgeführt, wobei der Server Java und die Java Virtual Machine (JVM) unterstützen muss, wie beispielsweise Apache Tomcat oder der Websphere Application Server. Die Java Servlets werden bei Bedarf in die JVM des Webservers geladen und gestartet. Sobald ein Servlet ausgeführt wird, können eingehende Anfragen von diesem entgegengenommen werden. Von dem Servlet wird dynamisch eine Antwortnachricht generiert, welche an den Client zurückgesendet wird, wie in Abbildung 2 dargestellt. Java Servlets können auf Datenbanksysteme (wie beispielsweise MySQL) zugreifen, um Daten persistent zu speichern. [23]

Der Prozessablauf einer Client-Servlet-Verbindung gestaltet sich hierbei wie folgt [23]:

1. Der Client sendet eine Anfrage an den Server
2. Der Server sendet die Informationen, die bei der Anfrage übermittelt wurden, weiter an das Servlet
3. Das Servlet reagiert auf die Anfrage und generiert dynamisch eine Antwort, welche von der jeweiligen Client-Anfrage abhängt, und übergibt diese dem Server.
4. Der Server schickt die Antwort des Servlets zurück an den Client.

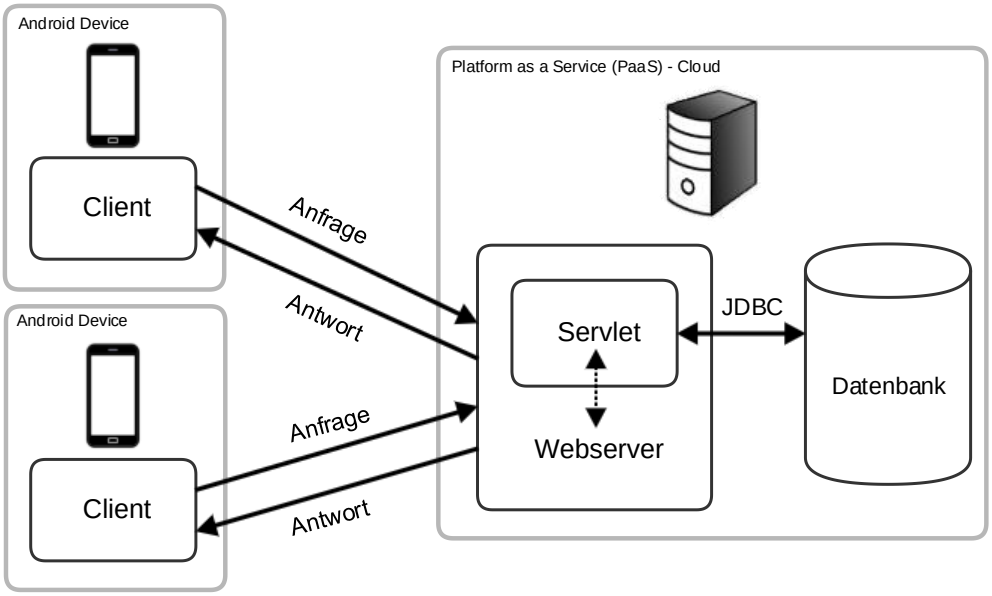


Abbildung 2: Verbindung zwischen Client, Servlet und Datenbank

Da Servlets in Java geschrieben werden, kann auf eine Vielzahl von *Application Programming Interfaces* (APIs), wie zum Beispiel Java Database Connectivity[1] (JDBC) zugegriffen werden. Servlets sind mit herkömmlichen Common Gateway Interface[2] (CGI) vergleichbar, bieten aber insbesondere bei der Entwicklung eines Identifizierungs-Frameworks für Android folgende Vorteile:

1. Auf Java basierend: Servlets können von allen Vorteilen profitieren, die eine moderne, objektorientierte Programmiersprache mit sich bringt. Zudem wird der Datenaustausch zu anderen Java-basierten Programmen vereinfacht, so können bestimmte Java-Objekte serialisiert und diese zum Beispiel direkt an Android-Anwendungen geschickt (oder empfangen) und verarbeiten werden.
2. Persistenz und Performanz: Java Servlets werden im Gegensatz zu CGI nur einmalig von dem Server in die JVM geladen, wobei jede Client-Anfrage einen neuen Thread[3] erzeugt. Es können somit beispielsweise Datenbankverbindungen gehalten, und Daten im Hauptspeicher verarbeitet werden, die mehrere Client-Anfragen überdauern.
3. Plattformunabhängig: Da Servets auf Java basieren, sind sie nicht von einer bestimmten Plattform abhängig. Servlets können somit von verschiedenen Webservern auf unterschiedlichen Betriebssystemen ausgeführt werden. Dies kann vor allem bei Cloud-basierten, skalierbaren Anwendungen von Vorteil sein, insbesondere dann, wenn eine gewisse Flexibilität aufgrund von sich schnell ändernden Anforderungen vorausgesetzt wird. [23]

Bei der Implementierung kann auf eine Reihe an vordefinierten Schnittstellen, Klassen und Methoden zurückgegriffen werden. Falls beispielsweise das Servlet (wie in den meisten Fällen) HTTP-Anfragen verarbeiten soll, kann eine Klasse erstellt werden, die die `javax.servlet.http.HttpServlet`-Schnittstelle implementiert [23]. Informationen zu den Client-Anfragen werden in einem `HttpServletRequest`-Objekt gekapselt. Das Objekt enthält alle Daten, die von einem Client an das

[1] JDBC, eine universelle Java - Datenbankschnittstelle
[2] CGI, dient dem dynamischen Datenaustausch zwischen Client und Webserver.
[3] Java-Thread, sequentieller Ausführungsstrang, der nebenläufig zu anderen Threads ausgeführt werden kann.

Servlet übermittelt werden sollen, wie zum Beispiel Header-Daten, aber auch die eigentlichen Nutzdaten.

Die `HttpServletRequest`-Klasse stellt zudem Methoden bereit, die für die Extraktion von Informationen aus den Anfragedaten (wie zum Beispiel die Parameter einer Anfrage) genutzt werden können [23]. Alle Informationen einer Antwort, die das Servlet dynamisch je nach Anfrage erzeugt, können in einem `HttpServletResponse`-Objekt übergeben werden. Das Objekt kann beispielsweise (HTML-)Text kapseln, wobei aber auch jeder andere serialisierbare Datentyp denkbar ist. Im Header kann beschrieben werden, um welchen Typ von Daten es sich hierbei handelt, so dass der Client anschließend passend auf die Antwortdaten reagieren und diese weiterverarbeiten kann [23].

Jedes Mal, wenn eine Anfrage eintrifft, wird ein neuer Servlet-Thread gestartet, wodurch mehrere Anfragen zeitglich beantwortet werden können. Die durch die `javax.servlet.http.HttpServlet`-Schnittstelle beschriebenen Methoden `doPost` (für HTTP POST - Anfragen) und `doGet` (für HTTP GET - Anfragen) werden nach Eingang einer Anfrage mit einem entsprechenden `HttpServletRequest`- und `HttpServletResponse`-Objekt als Parameter aufgerufen. Die Verarbeitung der Anfrage und die Generierung der Antwortdaten erfolgt dann innerhalb dieser Methoden [23].

2 Ablauf der Datenerfassung und Identifizierung

In den nachfolgenden Kapiteln werden die einzelnen Teilschritte vorgestellt, die bei der Identifikation von Nutzern und Geräten von dem Android-Framework und dem Java-Servlet ausgeführt werden, wie in Abbildung 3 dargestellt.

2.1 Grundsätzliche Vorgehensweise

Sobald das Framework als Teil einer Anwendung das erste Mal auf einem mobilen Gerät gestartet wird, kann mit der Erfassung relevanter Daten begonnen werden. Um Nutzer und Geräte identifizieren zu können, müssen zu Beginn erst einmal genügend Informationen gesammelt werden, die den zu identifizierenden Nutzer oder das Gerät auf eine ganz bestimmte Art und Weise charakterisieren. Keine der in dieser Arbeit vorgestellten Methoden benötigt in diesem Zusammenhang den aktiven Eingriff des Nutzers, es handelt sich somit um eine passive Nutzer- und Geräteidentifizierung. Alle Daten, die für die Identifizierung benötigt werden, können aus Informationsquellen gewonnen werden, die in der Regel auf handelsüblichen Android-Geräten verfügbar sind und sich zum Teil aus den individuellen Verhaltensweisen oder Eigenschaften der Nutzer oder Geräte ergeben. Die anfängliche Datenaggregation und Vorverarbeitung der Rohdaten wird als Enrolment-Phase bezeichnet. Die Enrolment-Phase umfasst zudem auch die Merkmalsextraktion sowie Speicherung der aggregierten Daten.

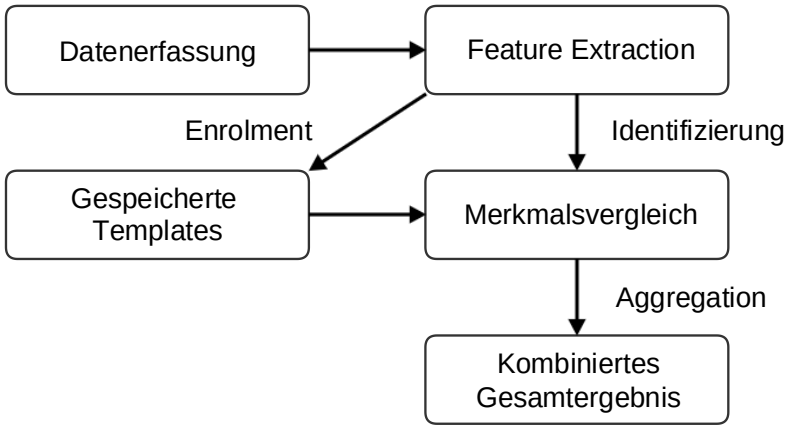


Abbildung 3: Die einzelnen Schritte der Identifikation

Direkt nach der erneuten Installation des Frameworks sind noch keine Informationen darüber bekannt, ob das Gerät oder der Nutzer bereits eine Enrolment-Phase durchlaufen hat, bzw. ob bereits Referenzdaten und entsprechende Identifikationsnummern in der Datenbank vorhanden sind. Das Gerät und der Nutzer erhalten somit nach der Installation neue Identifikationsnummern, und es wird vorerst wie gehabt die Enrolment-Phase gestartet. Allerdings wird im nächsten Schritt, sobald genug Merkmalsvektoren extrahiert wurden, der eigentliche Vorgang der Identifizierung gestartet. Falls der Nutzer oder das Gerät dem Framework bereits bekannt sind, lassen sich diese anhand der gespeicherten Merkmalsvektoren bzw. Referenzdaten identifizieren. In der Identifizierungsphase werden somit neu aggregierte Merkmalsvektoren eines zu identifizierenden Nutzers oder Geräts mit den bereits gespeicherten Referenzdaten verglichen. Das Framework überprüft in bestimmten Abständen, ob ein bestimmtes Gerät seinen Besitzer gewechselt hat. Es werden also die nutzerspezifischen Referenzdaten, die auf einem Gerät zu unterschiedlichen Zeitpunkten gesammelt werden konnten, untereinander verglichen. Falls die Identifizierung des bisherigen Nutzers für ein Gerät fehlschlägt, wird versucht, den neuen Besitzer innerhalb der Menge aller bekannten Nutzer zu identifizieren. Für den Fall dass der neue Nutzer dem Framework noch nicht bekannt sein sollte, werden erneut Referenzdaten gesammelt, um den neuen Nutzer später identifizieren zu können.

Die Ergebnisse, die durch die Merkmalsvergleiche der einzelnen Methoden gewonnen wurden, werden nach der jeweiligen Performanz und Aussagekraft gewichtet, und zu einem Gesamtergebnis kombiniert. Falls das Gesamtergebnis, welches die Wahrscheinlichkeit, dass ein Nutzer oder ein Gerät identifiziert werden konnte, einen gewissen Grenzwert erfüllt, wird die Identifizierung als erfolgreich bewertet. Es besteht in diesem Fall die Möglichkeit, die Referenzdaten der neu identifizierten Nutzer und Geräte mit den gespeicherten Datensätzen zusammenführen.

Weitere Details über den genauen Ablauf der Enrolment- und Identifizierungsphase finden sich in den nachfolgenden Kapiteln.

2.2 Data Aggregation und Feature Extraction

Android-Anwendungen können auf eine Vielzahl von Daten über den jeweiligen Benutzer und das mobile Gerät zugreifen, wie in Kapitel 3 näher beschrieben. Allerding gibt es für die gesammelten Informationen bestimmte Voraussetzungen, die zumindest zum Teil erfüllt sein müssen, damit sie sinnvoll für die Identifikation von Nutzern und Geräten eingesetzt werden können:

- **Universalität**

  Im besten Falle sollte auf jedem Android-Gerät die verwendete Informationsquelle zur Verfügung stehen.

- **Erfass- und Messbarkeit**

  Die verwendeten Informationen sollten auf herkömmlichen Smartphones und Tablets von einer Android-Anwendung abgefragt werden können, und zudem sinnvolle Berechnungen zulassen.

- **Einmaligkeit**

  Die Menge an Informationen, die einer Identifikationsnummer zugeordnet wird, sollte im besten Falle eindeutig sein, und mit keiner Datenmenge eines anderen Nutzers oder Geräts übereinstimmen.

- **Persistenz**

  Die Daten, auf denen die Identifizierungsmethoden basieren, sollten sich über längere Zeiträume nur sehr wenig, im besten Falle gar nicht, verändern. [24]

Es zeigt sich, dass keine der verfügbaren Methoden auf Android-Geräten für sich allein genommen alle vier Kriterien vollständig erfüllen kann, wie in Kapitel 3 näher beschrieben. Jede Methode hat hierbei ihre eigenen Stärken und Schwächen, was sich in erster Linie in den jeweiligen Fehlerraten niederschlägt. Einige Methoden zeichnen sich beispielsweise durch eine besonders starke Persistenz der Informationsquellen aus, müssen aber zum Beispiel bei der Einmaligkeit der Merkmale oder der uneingeschränkten Verfügbarkeit Abstriche machen. Diese Art von Methoden eignet sich insbesondere für *Clusteranalysen,* also der Unterscheidung von Gruppen, wodurch die Auswahl an möglichen Kandidaten in der Regel schon deutlich eingeschränkt werden kann. Andere Methoden hingegen können wiederum genau bei den Kriterien der Einmaligkeit oder allgemeinen Verfügbarkeit punkten, bringen dafür aber wiederum andere Nachteile mit sich. Die Grundidee des Frameworks besteht darin, die Stärken jeder Methode optimal zu nutzen, und die Performanz der einzelnen Methode durch das Zusammenwirken aller Methoden zu verbessen. Die Schwächen einzelnen Methoden sollen durch die Rückkopplung mit anderen Methoden nach einer gewissen Zeit ausgeglichen werden. Wie durch die

Kombination der einzelnen Methoden ein optimales Ergebnis bei der Identifizierung von Nutzer und Geräten erzielt werden kann wird in Kapitel 2.6 näher erläutert.

Bei der Auswahl der Vorgehensweisen sollte ein weiteres Kriterium beachtet werden, welches die Menge aller Methoden als Ganzes betrifft: Die Vorgehensweise der gewählten Methoden sollten möglichst unabhängig voneinander sein. Falls eine direkte Abhängigkeit zwischen den Fehler zweier Methoden besteht, kann sich durch die Kombination dieser Verfahren kein wirklicher Mehrwert für das identifizierungs-Framework ergeben. Für korrekte Identifizierungen ist hingegen eine möglichst große Korrelation zwischen den Methoden wünschenswert.

Die Daten, die von geeigneten Informationsquellen gesammelt werden, werden in der Enrolment-Phase vorverarbeitet (Preprocessing). Es werden Merkmalsvektoren extrahiert, welche die Komplexität und die Menge an Daten, die letztendlich gespeichert und analysiert werden müssen, auf das Notwendigste reduzieren. In Kapitel 3 wird hierzu die Vorgehensweise der einzelnen Methoden erläutert. Die extrahierten Informationen repräsentieren die essentiellen Charakteristika der gesammelten Rohdaten. Sie ermöglichen den effizienten Vergleich, also beispielsweise die Anwendung von fehlertoleranten Ähnlichkeitsmaßen während der eigentlichen Analysephase zur Identifizierung von Nutzern und Geräten.

2.3 Datentransfer

Die Daten, die während der Enrolment-Phase gesammelt und verarbeitet wurden, müssen für die eigentliche Identifizierung von Nutzern und Geräten gespeichert werden. Da eine Vielzahl an Daten von unterschiedlichen Benutzern und Geräten verglichen werden müssen, bietet es sich an, die Informationen auf einem Cloud-basierten Server zu speichern, der per Internet für alle laufenden Anwendungen, die das Identifizierungsframework nutzen, erreichbar ist. Dadurch kann zudem sichergestellt werden, dass die Daten nicht nachträglich auf den Geräten geändert oder sogar gelöscht werden.

Die Daten werden falls möglich gebündelt an das Framework gesendet. Dies ist beispielsweise bei allen Daten, die unmittelbar nach der ersten Installation aggregiert werden können, der Fall. Bei Identifizierungsmethoden, die eine längere Datenerfassungsphase benötigen, werden die Daten jedoch einzeln je nach Verfügbarkeit gesendet. Das Framework ist in der Lage, den bestehenden Datensatz um neu hinzukommende Referenzdaten zu ergänzen.

Während der Identifizierungs-Phase müssen die kombinierten Ergebnisse der Identifizierung zudem vom Java-Servlet zurück an das Android-Framework gesendet werden. Der Datentransfer erfolgt ähnlich wie der Übertragung der Referenzdaten (Serialisierung, dann Komprimierung der Daten), wobei die Pseudonymisierung der Informationen durch die Verwendung von globalen, zufällig generierten Nutzer- und Geräte-IDs sichergestellt wird. Das Framework wertet die Ergebnisse der Identifizierung aus, sobald die Daten vom Java-Servlet empfangen wurden, und speichert diese auf dem mobilen Gerät.

Das Framework stellt zudem eine Schnittstelle zur Verfügung, über welche die empfangen und gesammelten Daten abgefragt werden können. Dadurch wird eine Anbindung des Frameworks an weitere sicherheitsrelevante Komponenten ermöglicht. Anwendungen können zudem basierend auf den bereitgestellten Daten auf spezifische Änderungen reagieren.

Die gesammelten Daten können logischerweise nur bei bestehender Internet-Verbindung an das Java Servlet gesendet werden. Falls keine Internet-Verbindung verfügbar ist, werden die Referenzdaten deshalb zwischenzeitlich lokal auf dem mobilen Gerät gespeichert, bis entweder wieder eine Verbindung mit dem Servlet möglich ist, oder die gespeicherten Informationen durch neu aggregierte oder aktualisierte Referenzdaten lokal ersetzt werden.

Um die Referenzdaten bei bestehender Verbindung an den Server zu senden, werden folgende Schritte durch das Framework und das Java-Servlet ausgeführt:

1. Pseudonymisierung: Um sensible Nutzer- und Gerätedaten zu schützen, werden die Hash-Werte dieser Daten berechnet. Dazu kommt der Secure Hash Algorithm[4] (SHA-256) zum Einsatz [25]. Mithilfe der Hash-Werte lässt sich im Nachhinein überprüfen, ob die ursprünglichen Quelldaten, auf denen die Hash-Werte beruhen, identisch waren.
2. Serialisierung und Komprimierung der Daten: Die Java-Objekte werden serialisiert und komprimiert, das heißt sie werden in eine kompakte Form gebracht, die direkt an das Servlet geschickt und von diesem verarbeitet werden kann. Java-Objekte, die serialisiert werden können, müssen das `java.io.Serializable` - Interface implementieren. Für die Komprimierung der serialisierten Daten kommt die `java.util.zip.GZIPOutputStream` – Klasse zum Einsatz. [26]
3. Datentransfer: Die komprimierten Daten werden per Hypertext Transfer Protocol (HTTP)-Anfrage an das Servlet geschickt. Die Daten werden für die Übertragung mithilfe des Secure Socket Layer[5] (SSL) verschlüsselt. Mithilfe von SSL wird das Servlet zudem gegenüber dem Framework authentifiziert. Eine Android-Anwendung, die das Framework einbindet, kann außerdem einen geheimen Schlüssel übergeben, womit sich dann das Framework wiederum gegenüber dem Servlet authentifizieren kann. Das Servlet muss dafür entsprechend konfiguriert sein. [27] Zudem besteht die Möglichkeit, die Daten bei Bedarf in serialisierter Form an zusätzliche, externe Komponenten mithilfe einer Listener-Schnittstelle zu übergeben.

   Die gesammelten Informationen können entweder per WLAN, oder aber auch, je nach Verfügbarkeit, über eine mobile Internetverbindung gesendet werden. Das Framework kann so konfiguriert werden, dass die Daten nur bei aktiver WLAN-Verbindung gesendet werden. Alle Daten, die das Framework in dieser Konfiguration ohne WLAN-Verbindung sammelt, werden auf dem Gerät selbst in einem Cache zwischengespeichert. Sobald sich das Android-Gerät das nächste Mal mit einem WLAN-Netzwerk verbindet, werden diese Daten an das Servlet geschickt.
4. Dekomprimierung und Deserialisierung: Die Daten werden von dem Servlet empfangen, und mithilfe der `java.util.zip.GZIPInputStream` - Klasse dekomprimiert. Anschließend können die Daten wieder deserialisiert werden. Das Servlet erhält somit ein Java-Objekt, welches identisch ist mit dem ursprünglich vom Framework versendeten Objekt. [28]

2.4 Datenspeicherung

Das Servlet kann mithilfe von JDBC auf verschiedene Datenbanktypen zugreifen, wobei das in dieser Arbeit vorgestellte Framework das Datenbankverwaltungssystem MySQL verwendet [29].

Sobald die Datenobjekte vom Servlet empfangen und wiederhergestellt wurden, können die enthaltenen Informationen in der Datenbank gespeichert werden. Um die eigentliche Identifizierung zu beschleunigen, werden die charakteristischen Features in der Datenbank indexiert. Jeder Feature-Vektor wird in einer separaten Zeile einer Datenbanktabelle gespeichert. Jede Methode, beziehungsweise in einige Fällen eine Gruppe von ähnlichen Methoden erhält hierbei jeweils eine eigene Tabelle. Die Einträge der Tabellen werden per Fremdschlüssel mit der dazugehörigen Identifikationsnummer eines Nutzers oder Geräts assoziiert.

[4] Secure Hash Algorithm, Spezifikation: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf
[5] Secure Socket Layer, Spezifikation: https://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00

Einige Methoden generieren selbst nach Merkmalsextraktion größere Datenmengen. Um trotzdem eine effiziente Analyse dieser Datenmengen zu ermöglichen, wird das Java-Objekt, welches die Informationen hält, serialisiert und separat gespeichert. In der Datenbanktabelle selbst werden, wie bei den anderen Methoden auch, nur die charakteristischsten Merkmale gespeichert und indexiert. Jeder Eintrag in der Tabelle erhält dann eine Referenz auf das gespeicherte Datenobjekt, wie in Abbildung 4 beispielhaft dargestellt. Sobald in der Phase der Nutzer- und Geräteidentifizierung eine Ähnlichkeit zu den in der Tabellen indexierten Merkmalen festgestellt werden kann, wird das Datenobjekt bei Bedarf in den Hauptspeicher geladen und deserialisiert, um eine präzisere Analyse zu ermöglichen. Das Servlet kann so konfiguriert werden, dass eingehende Datenobjekte im Hauptspeicher bleiben. Grund für diese Option ist die Tatsache, dass Hauptspeicherzugriffe deutlich schneller sind als SQL-Anfragen, wodurch Performancegewinne erzielt werden können. Erst wenn der benutzte Arbeitsspeicher einen gewissen Grenzwert überschreitet, werden die Datenobjekte aus dem Hauptspeicher entfernt, und müssen dann bei Bedarf wieder aus der Datenbank oder (bei größeren Objekten) von der Festplatte gelesen werden. Als Folge können unter günstigen Umständen viele aufeinanderfolgende Anfragen komplett im Hauptspeicher bearbeitet werden, beispielsweise wenn von einem Client in kurzer Zeit größere Mengen an Datenpaketen empfangen werden.

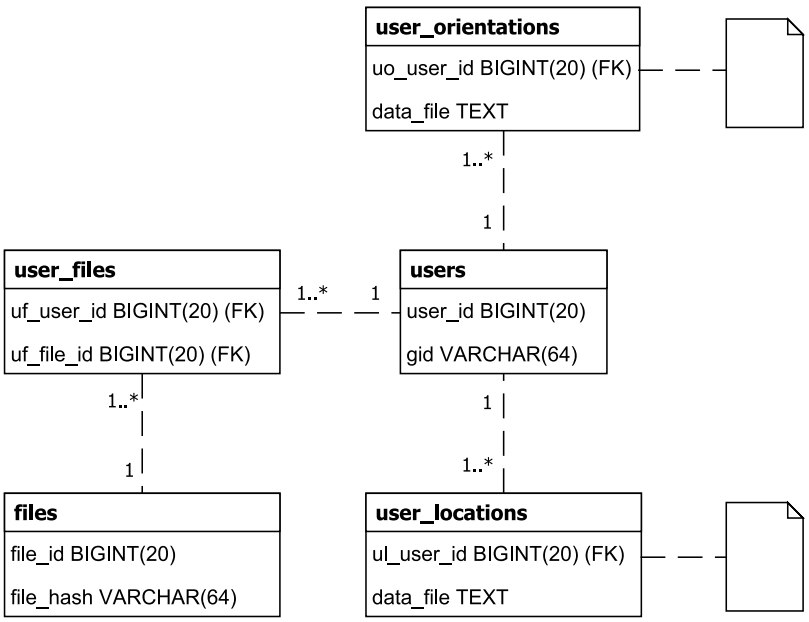


Abbildung 4: Ausschnitt aus der grundlegenden Datenbankstruktur

2.5 Vergleich der Daten

Sobald die Enrolment-Phase abgeschlossen ist, kann mit der eigentlichen Identifizierung von Benutzern und Geräten begonnen werden. Dabei ist zu beachten, dass jede Methode (wie in Kapitel 3 vorgestellt) unterschiedlich viel Zeit benötigt, bis genug Daten für die Identifizierung gesammelt wurden. Einige Methoden können schon nach wenigen Sekunden für die Identifizierung eingesetzt werden. Andere Verfahren, insbesondere biometrische Methoden, benötigen mehr Zeit für die Enrolment-Phase. Diese Identifizierungsmethoden können erst dann für die Identifizierung verwendet werden, nachdem das Framework ein ganz bestimmtes Nutzerverhalten aufzeichnen konnte. Dies hat zur Folge, dass zu Beginn noch nicht alle Methoden für die eigentliche Identifizierung zur Verfügung stehen. Umso länger ein Nutzer jedoch das Framework auf seinem mobilen Gerät installiert hat, desto mehr Methoden stehen nach und nach zur Verfügung. Die Aussagekraft und Präzision des Identifizierungsvorgangs steigt mit der Zeit, bis

im besten Fall zu einem Zeitpunkt alle Methoden genug Daten gesammelt haben, um für identifizierungszwecke eingesetzt zu werden.

Sobald genug Daten gesammelt wurden, um mit der eigentlichen Identifizierung zu beginnen, werden neu aggregierte Merkmale mit den bereits gespeicherten verglichen, wie in Abbildung 5 dargestellt. Dieser Vorgang wird zentral von einem Java-Servlet ausgeführt, wobei die Merkmalsdaten aller identifizierter Objekte wie in Kapitel 1.5 beschrieben in einer MySQL-Serverdatenbank gespeichert werden. Die Daten, die während der Enrolment-Phase gesammelt wurden, werden bei Bedarf durch neu aggregierte Daten ersetzt, falls ein Benutzer oder Gerät durch den kombinierten Identifizierungsvorgang eindeutig einem Referenzdatensatz zugeordnet werden konnte. Dadurch wird die Aktualität der gespeicherten Datensätze sichergestellt.

Um den Prozess der Identifizierung zu beschleunigen, wird mithilfe der Merkmal-Indizes eine Menge von möglichen Kandidaten aus der Datenbank geladen. Als Indizes werden hierbei Merkmale gewählt, die durch SQL-Anfragen effizient abgefragt werden können. Jedes Element, also jeder Datensatz dieser Menge wird anschließend mit dem neu extrahierten Datensatz verglichen. Welches algorithmische Verfahren hierbei zum Einsatz kommt, hängt von der spezifischen Informationsquelle und somit der verwendeten Methode ab, welche in Kapitel 3 näher beschrieben werden.

Jede Methode, die im Rahmen der Identifizierung zum Einsatz kommt, liefert einen Wert zurück, der beschreibt, mit welcher Wahrscheinlichkeit ein Nutzer oder Gerät identifiziert werden konnte. Der Vergleich von Nutzern und Geräten erfolgt paarweise, wobei geeignete Datenstrukturen und Thresholds dabei helfen, den Vorgang zu beschleunigen. Das Framework führt eine sogenannte *Diskriminanzanalyse* durch, unterscheidet also anhand von Merkmalen bestimmte Gruppen von Nutzern oder Geräten. Im besten Fall beinhaltet eine solche Gruppe mit hoher Wahrscheinlichkeit nur einen einzigen Nutzer oder Gerät. Es kann in der Regel die Ähnlichkeit (oder äquivalent dazu, die Distanz) zweier Datensätze berechnet werden, also beispielweise zu wie viel Prozent die verglichenen Daten unter bestimmten Voraussetzungen übereinstimmen bzw. voneinander abweichen. Details über die Berechnungen, die im Verlauf der Identifizierung benötigt werden, finden sich in Kapitel 3.

Die Ergebnisse der einzelnen Methoden werden anschließend in einer Datenbanktabelle für die spätere Verwendung gespeichert. Jeder Wert, der von einer Methode zurückgeliefert wird, beschreibt die Ähnlichkeit zweier Datensätze. Die verglichenen Datensätze können entweder jeweils zwei Geräten oder aber auch jeweils zwei Nutzern zugeordnet werden. Das Ergebnis einer Methode kann somit in einer Datenbanktabelle in einer entsprechenden Zeile gespeichert werden. Primärschlüssel sind hierbei die beiden Identifikationsnummern, die den verglichenen Datensätzen zugeordnet wurden. Jede weitere Spalte enthält schließlich den zurückgelieferten Wert einer spezifischen Methode. Im nächsten Kapitel wird erläutert, wie das Zwischenspeichern der Ergebnisse dabei helfen kann, den Identifizierungsvorgang bzw. die Kombination einzelner Ergebnisse zu beschleunigen.

Ein Teil der Methoden kann neben der Identifizierung auch für die Verifizierung eines Nutzers eingesetzt werden. So können beispielsweise Verhaltensänderungen erkannt und der Anwendung mitgeteilt werden. In einigen Fällen kann aufgrund der Verhaltensänderung auf einen Nutzerwechsel geschlossen werden. Allerdings kann erst mittelfristig eine aussagekräftige Wahrscheinlichkeit für einen Nutzerwechsel vom Framework bereitgestellt werden, da Verhaltensänderungen durchaus registriert werden können, ohne dass ein Gerät seinen Besitzer wechselt. Für die längerfristige Identifizierung werden prinzipiell gemittelte Werte verwendet, wodurch kurzfristige Änderungen der Referenzdaten keine fehlerhaften Rückschlüsse innerhalb des Identifizierungsvorgangs verursachen.

Ein weiterer Wert, der von einigen Methoden zurückliefert wird, beschreibt die Aussagekraft der

gemessenen Werte. Falls beispielsweise zwei Datensätze komplett übereinstimmen, sich diese Daten aber auch für so gut wie jeden anderen Nutzer oder Gerät finden lassen, kann aus dem Ergebnis kein wirklicher Rückschluss auf die Identität des Nutzers oder des Geräts gezogen werden. Falls hingegen ein Datensatz eine gewisse Einmaligkeit unter allen Nutzern oder Geräten aufweist, hat eine Übereinstimmung eine große Aussagekraft in der Identifizierungsphase. Der Wert repräsentiert somit die relative Häufigkeit aggregierter Merkmale, ein wichtiges Kriterium, welches die Aussagekraft einer Methode maßgeblich bestimmt. Analog zu den Ergebnissen, die die Ähnlichkeit zweier Datensätze beschreiben, wird dieser Wert auch in der entsprechenden Datenbanktabelle für die spätere Verwendung gespeichert.

2.6 Kombination der Ergebnisse

Im vorhergehenden Kapitel wurde beschrieben, wie die einzelnen Methoden die gespeicherten Referenzdaten mit neu aggregierten Merkmalen vergleichen. Die Rückgabewerte der einzelnen Vergleichsmethoden, werden im nächsten Schritt validiert, gewichtet und anschließend kombiniert, wie in Abbildung 5 dargestellt. Die spezifische Gewichtung orientiert sich an der jeweiligen Güte der Identifizierungsmethoden, welche sich in den Fehlerraten widerspiegelt.

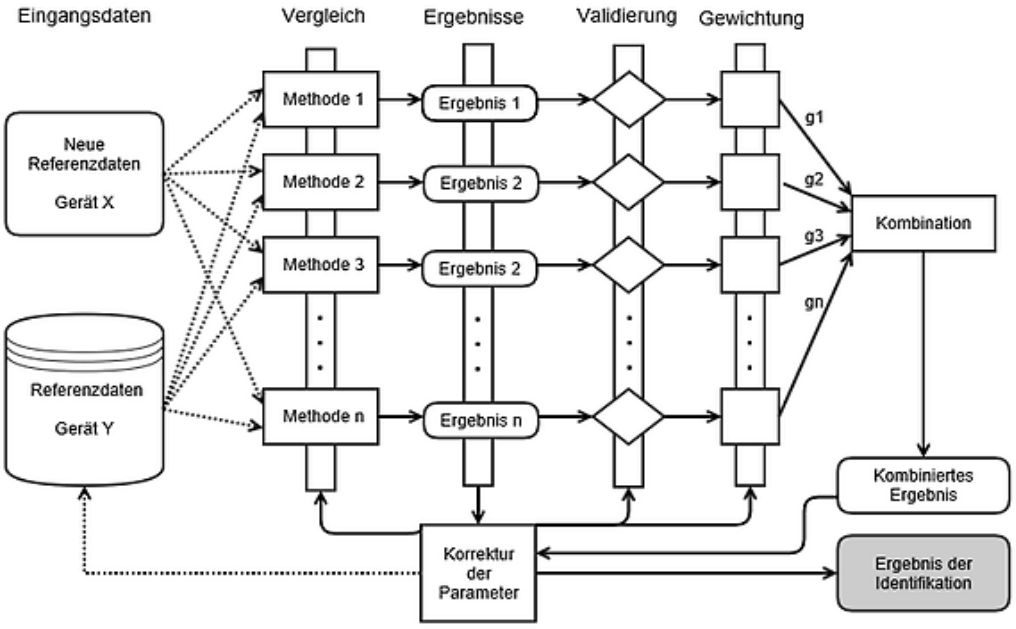


Abbildung 5: Kombination der Identifizierungsmethoden

Die einzelnen Identifizierungsmethoden, die im Rahmen dieser Arbeit untersucht wurden, können prinzipiell als *Klassifikatoren* vom Typ-3 betrachtet werden. Diese Art von Klassifikator gibt neben der Zuordnung zu einer Liste von Klassen zusätzliche Plausibilitäts- bzw. Wahrscheinlichkeitswerte für die jeweiligen Zuordnungen zurück [30]. Es bestehen prinzipiell mehrere plausible Möglichkeiten, die Ergebnisse eines Typ-3 Klassifikators zu kombinieren: So können beispielsweise Maximal- oder Minimal-Werte verwendet werden. Auch die Verwendung des Medians oder die Mittelung aller Wahrscheinlichkeiten kann in bestimmten Szenarien Sinn machen. Die *Bayes'sche Kombinationsregel* hingegen bildet das Produkt über alle Wahrscheinlichkeiten, eine Methode, die vor allem dann Sinn macht, wenn alle Verfahren möglichst unabhängig voneinander sind, und die einzelne Methoden für sich genommen konstante Fehlerraten aufweisen. Bei der Entwicklung des Frameworks wurde jedoch die Entscheidung getroffen, die Wahrscheinlichkeiten der einzelnen Methoden zu gewichten und anschließend zu kombinieren. Die Gewichtung der einzelnen Wahrscheinlichkeiten hat den

entscheidenden Vorteil, dass die spezifischen Fehlerraten bzw. die Güte der jeweiligen Methoden bei der Kombination beachtet werden können. Die Berechnung der Gesamtwahrscheinlichkeit, dass x und y als identisch betrachtet werden können gestaltet sich wie folgt:

$$p_{gesamt}(x,y) = f\big(p_1(x,y), p_2(x,y), p_3(x,y), \dots, p_n(x,y)\big) = \sum_{i=1}^{n} p_i(x,y) * w(i) \tag{2.1}$$

wobei die Identifizierungsmethoden und die jeweilige Gewichtung folgende Bedingungen für jede mögliche Kombination aus Nutzern oder Geräte erfüllen müssen:

$$p_i(x,y) \in [0,1]$$
$$\sum_{i=1}^{n} w(i) = 1 \tag{2.2}$$

Falls eine Methode (noch) nicht verfügbar ist, wird das entsprechende Gewicht gleich null gesetzt, und die Gewichtung der anderen Methoden entsprechend angepasst. Abbildung 5 zeigt den prinzipiellen Vorgang der Kombination der gewichteten Wahrscheinlichkeitswerte. In Kapitel 4.3 wird erläutert, wie diese Vorgehensweise bei der Implementierung des Java Servlets in das Gesamtsystem eingebettet wird.

Das Framework nimmt mit der Zeit Anpassungen der Gewichtungen der einzelnen Identifizierungsmethoden vor, wobei sich allerdings die Frage stellt, welche Gewichtungen zu Beginn verwendet werden sollen. Es lässt sich feststellen, dass die Güte der Verfahren zum großen Teilen auch vom Kontext und somit von der Anwendung selbst abhängt, in der das Framework tatsächlich eingesetzt wird. Denkbare Android-Anwendungen, bei denen die Menge und Qualität der jeweils verfügbaren Referenzdaten direkt mit dem jeweiligen Verwendungszweck zusammenhängt wären beispielsweise Foto-Apps, Outdoor-App, Navigations-Apps, sowie Apps für die Verwaltung von Musik, Dateien oder Kontaktdaten. Es erscheint daher sinnvoll, dass die initialen Gewichtungen der Identifizierungsmethoden bei Bedarf von der spezifischen Anwendung festgelegt werden können. Eine Foto-App könnte so beispielsweise Methoden, welche auf die interne Kamera zugreifen, besonders stark gewichten. Bei der Implementierung des Frameworks wurde diese Besonderheit beachtet und eine entsprechende Schnittstelle eingerichtet. Des Weiteren besteht für Anwendungen die Möglichkeit, die Einzelergebnisse der Identifizierungsmethoden neben dem Gesamtergebnis abzufragen. Anwendungsentwicklern wird somit zusätzlich ermöglicht, eine anwendungsspezifische Kombinationsfunktion bei Bedarf selbst zu implementieren.

Tabelle 2: Die Ergebnisse des Identifizierungsverfahrens in der Datenbanktabelle (Beispiel)

id_1 (FK)	id_2 (FK)	m_combined	m_1	m_2	m_3	m_4	m_5	m_6
a	b	ab_combined	ab_1	ab_2	ab_3	ab_4	ab_5	ab_6
c	d	cd_combined	cd_1	cd_2	cd_3	cd_4	cd_5	cd_6
e	f	ef_combined	ef_1	ef_2	ef_3	ef_4	ef_5	ef_6

Das kombinierte Ergebnis kann, wie auch die Einzelwerte der jeweiligen Methoden, in einer Datenbanktabelle gespeichert werden. Primärschlüssel ist hierbei wiederum die Kombination der beiden Identifikationsnummern der verglichenen Nutzer bzw. Geräte, wie in Tabelle 2 beispielhaft dargestellt. Die Felder in der Tabelle enthalten die Identifikationsnummern, sowie alle berechneten Endergebnisse der Methoden und das kombinierte Engergebnis. In Tabelle 2 werden die Zusammenhänge durch

Platzhalter der einzelnen Spalten verdeutlicht.

Es kann unter Umständen sein, dass die Verfügbarkeit bzw. das Fehlen eines Merkmals wiederum selbst als konstantes Merkmal (z.B. Hardware-Eigenschaft) interpretiert werden kann Tabelle 3, Zeilen 2 bis 4). So kann beispielsweise ausgeschlossen werden, dass auf einem Gerät ohne SD-Kartenslot plötzlich die Seriennummer einer SD-Karte extrahiert werden kann (Stand: November 2014).

Weitere Kriterien für die Qualität und Verwendbarkeit eines Merkmals sind die Universalität und die Erfassbarkeit der benötigten Informationen, wie in Kapitel 2.2 beschrieben. Die meisten der in dieser Arbeit vorgestellten Methoden können auf fast allen handelsüblichen Android-Geräten eingesetzt werden. Es gibt jedoch Szenarien, in welchen einzelne Methoden keine Ergebnisse liefern. Falls eine Methode zu einem bestimmten Zeitpunkt während der Identifizierung nicht verfügbar ist, orientiert sich die Vorgehensweise des Frameworks an den Vorgaben welche in Tabelle 3 beschrieben werden.

Es gibt jedoch auch Fälle, in denen bestimmte Methoden keine Werte liefern, obwohl dies zu einem früheren oder späteren Zeitpunkt der Fall war (Tabelle 3, Zeilen 5 bis 8). Falls aber in der Datenbank während der Enrolment-Phase und auch danach noch keine Merkmale zu einer Methode gespeichert werden konnten, ist die Wahrscheinlichkeit hoch, dass diese Methode auf dem Gerät bzw. bei genau diesem Benutzer ganz einfach nicht zur Verfügung steht, was sich in der Gewichtung in Tabelle 3, Zeile 2 wiederspiegelt. Wenn Datensätze für die Identifizierung neu aggregiert werden, kann es allerdings sein, dass ausnahmsweise genau zu diesem Zeitpunkt eine Methode fehlschlägt, ein Umstand, der bei der Gewichtung in Tabelle 3, Zeilen 3 und 4 berücksichtigt wird. Falls eine Methode noch keine Rückschlüsse auf die Ähnlichkeit zweier Geräte oder Benutzer zulässt, da beispielsweise die entsprechende Methode noch auf bestimmte Nutzerhandlungen wartet, wird die entsprechende Methode vorerst ignoriert, und aus dem gewichteten Endergebnis der Identifizierung herausgerechnet.

Tabelle 3: Vorgehensweise bei fehlenden Daten

Daten (Identifizierung)	**Daten (Datenbank)**	**Verfügbarkeit der Methode**	**Ähnlichkeit**	**Relative Gewichtung**
vorhanden	vorhanden	-	Berechnung	Berechnung
vorhanden	fehlt	konstant	gering	hoch
fehlt	vorhanden	konstant	gering	hoch
fehlt	fehlt	konstant	hoch	durchschnittlich
vorhanden	fehlt	variabel*	gering	hoch
fehlt	vorhanden	variabel	gering	durchschnittlich
fehlt	fehlt	variabel*	hoch	gering

*die Verfügbarkeit der Methode ist prinzipiell variabel, es wird aber bleibende Nichtverfügbarkeit angenommen, da in diesen Fällen während der gesamten Zeitspanne der Enrolment-Phase (und auch danach) keine Daten gesammelt werden konnten.

Da für einen Teil der Methoden auf handelsüblichen Geräten sofort nach Installation der Anwendung genügend Referenzdaten für die Identifizierung zur Verfügung stehen, kann das Framework eine sofortige Einschätzung für die Wahrscheinlichkeit einer Identifizierung geben. Allerdings nimmt die Anzahl an verfügbaren Methoden mit der Zeit zu, wodurch prinzipiell erst nach einer gewissen Zeitspanne die zuverlässigsten Ergebnisse bei der Identifikation erzielt werden.

3 Mögliche Optionen zur Erfassung und Identifizierung von Benutzern und Geräten

Wie in Kapitel 2.2 beschrieben, sollten Informationsquellen, auf deren Basis die Identifizierung von Nutzern und Geräten durchgeführt werden kann, einige Kriterien erfüllen: Universalität, Erfass- und Messbarkeit, sowie Einmaligkeit und Persistenz. Bei der Auswahl geeigneter Methoden sollte somit darauf geachtet werden, dass die Informationen auf einem Großteil der handelsüblichen Android-Geräten von einer Anwendung bzw. dem Framework abgefragt und entsprechend verarbeitet werden kann. Im Allgemeinen bieten sich für mobile Geräte mehrere Möglichkeiten an, eine Identifizierung durchzuführen. Zum einen können auf einem Android-Gerät interne Identifikationsnummern abgefragt werden, wie aus Abbildung 6 ersichtlich. Systeminterne Identifikationsnummern werden von Android selbst bzw. einem System-Service erstellt, und sollten zumindest in der Theorie auf jedem Gerät verfügbar sein. Allerdings besteht die Möglichkeit, dass Android-Nutzer diese Identifikationsnummern zurücksetzen oder verändern können. Es kann zudem unter Umständen weitere Probleme mit der Persistenz dieser Werte geben, zum Beispiel wenn ein Android-Gerät in den Werkszustand zurückgesetzt wird, oder ein Update auf einem Gerät durchgeführt wird. Außerdem sorgen Implementierungsfehler dafür, dass die Werte teilweise mehrmals vergeben werden, oder aber gar nicht erst von einer Anwendung abgefragt werden können. Somit bringen die verfügbaren Identifikationsnummern neben dem eigentlichen Nutzen für das Framework auch eine Reihe von Nachteilen mit sich, wie in Kapitel 3.1 im Detail beschrieben. Trotz der Einschränkung eignen sich die Identifikationsnummern als Informationsquelle für das Android-Framework. So kann, falls ein Teil dieser Informationen auf einem Android-Gerät zur Verfügung stehen sollte, in Kombination mit weiteren Identifizierungsmethoden die Anzahl der in Frage kommenden Nutzer und Geräte deutlich eingeschränkt werden.

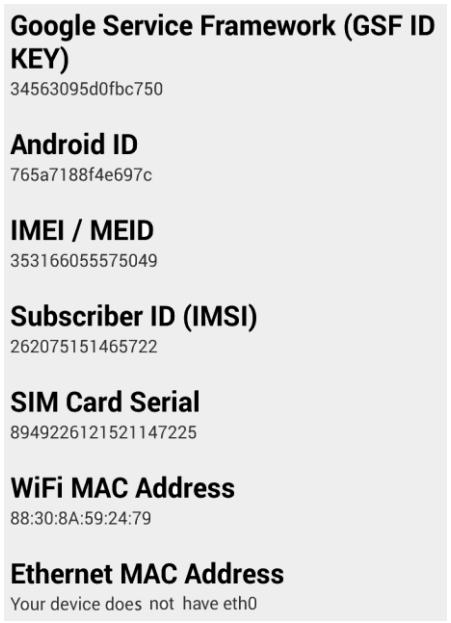


Abbildung 6: Interne Identifikationsnummern und Hardware-Adressen

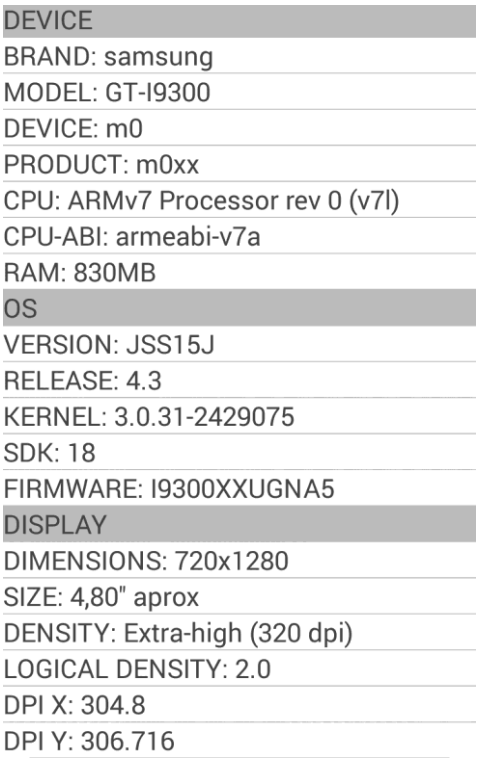


Abbildung 7: Hardware-Eigenschaften eines mobilen Geräts

Eine weitere Möglichkeit, verwendbare Daten zu extrahieren, besteht darin, bestimmte Informationen über die Hardware- Spezifikation des Geräts abzufragen, wie in Abbildung 7 dargestellt. Neben diesen Informationen lassen sich auch Hardware-Adressen und Seriennummern bestimmter Bauteile bzw.

Komponenten erfassen (Abbildung 6), und im Rahmen des Identifizierungsvorgangs verwenden. Details über die Vor- und Nachteile dieser Informationsquellen finden sich in Kapitel 3.2.

Mit handelsüblichen Android-Geräten kann eine Vielzahl an Verbindungen zu anderen Geräten bzw. Drahtlosadaptern aufgebaut werden. WLAN und Bluetooth zählt hierbei zu den am weit verbreitetsten Technologien, diese haben sich als Standard auf mobilen Geräten durchgesetzt. Auf Android-Geräten lassen sich viele Daten über die Eigenheiten dieser Informationsquellen extrahieren, wie in Abbildung 8 und Abbildung 9 zu erkennen ist. Abbildung 8 beschreibt beispielsweise die Signalstärken umliegender WLAN-Netze. Die gewonnene Information kann unter anderen dazu gewonnen werden, mithilfe von Triangulation die Position des Smartphones oder Tablets im Raum zu berechnen. Interessant ist hierbei, dass sich ein Teil der Methoden, welche von diesen Informationsquellen Gebrauch macht, der Nutzer-Identifizierung zuordnen lässt, während ein anderer Teil wiederum der Geräte-Identifizierung zuzuordnen ist. Das liegt daran, dass die Eigenschaften der Adapter analysiert werden können, aber auch die Menge aller Verbindungen, die von einem Nutzer mithilfe des Adapters hergestellt wurden. Kapitel 3.1.3, 3.1.4, 3.1.12 sowie 3.2.2 setzten sich im Detail mit möglichen Vorgehensweisen auseinander, die auf diesen Informationsquellen aufbauen.

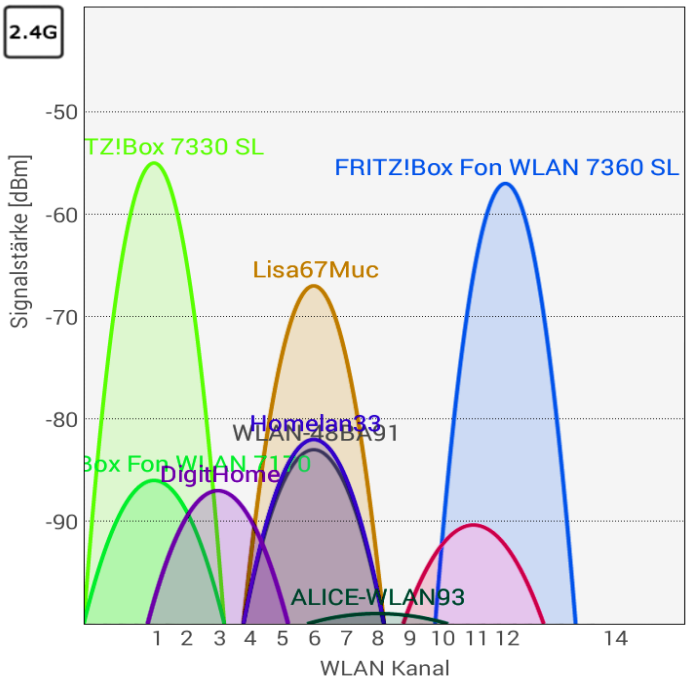


Abbildung 8: Signalstärken von WLAN-Netzen

SSID: "FRITZ!Box Fon WLAN 7360 SL"
External IP: 93.211.77.142
IP address: 192.168.178.22
Gateway: 192.168.178.1
Netmask: 255.255.255.0
DHCP server: 192.168.178.1
DHCP lease: 10 days
DNS 1: 192.168.178.1
DNS 2: 0.0.0.0
Link speed: 19 Mbps
MAC: 88:30:8A:59:24:79
SSID MAC: 24:65:11:AD:AD:DC
RSSI: -80 dBm

Abbildung 9: Eigenschaften und Konfiguration eines verbundenen Routers

Die Daten, die auf den Speichermedien eines Android-Geräts abgelegt wurden, können sich unter Umständen auch als Informationsquelle für Nutzer- und Geräteidentifizierungen eignen. Die Daten werden beispielsweise entweder direkt vom Nutzer erzeugt, oder wurden vom Android-Gerät selbst bzw. dessen Hersteller auf den internen Speicher geschrieben. So unterstützen beispielsweise Daten, die nicht direkt mit einem bestimmten Nutzer, dafür aber mit einem Gerät in Verbindung gebracht werden können, unter bestimmten Umständen die Geräteidentifizierung, wie in Kapitel 3.2 beschrieben. Es existiert allerdings auf einem Android-Geräte in der Regel auch eine große Menge an Daten, die dem Nutzer direkt zugeordnet werden können. So installieren Nutzer zum Beispiel eine individuelle Auswahl an Anwendungen, sie speichern Musik, Dokumente, aber auch Kontaktinformationen und viele weitere persönliche Informationen auf ihrem Gerät. Es erscheint sinnvoll, neben den Daten, die ein Nutzer selbst generiert hat, auch weitere Daten über den Nutzer, die jedoch vom Gerät erstellt wurden, zu untersuchen. So zeichnen Android-Geräte diverse Nutzer-Aktivitäten auf, wie beispielsweise ein- und

ausgehende Anrufe, inklusive Zeitstempel, sowie den SMS-Verlauf. In Kapitel 3.1 wird im Detail erläutert, wie sich diese Informationsquellen für die Nutzeridentifizierung einsetzen lassen.

Android-Geräte können mithilfe von Sensoren Informationen über die Umgebung extrahieren, in der sich das Gerät befindet, wie in den Abbildung 11 und Abbildung 10 dargestellt. Falls Anwendungen auf diese Informationen zugreifen möchten, wird eine entsprechende *SensorEventListener*-Schnittstelle angeboten. Es können zudem Informationen über den aktuellen Zustand des Geräts in Erfahrung gebracht werden. Die Art und Weise, wie ein mobiles Gerät tagtäglich benutzt wird, spiegeln einen Teil des persönlichen Alltags eines Menschen wieder. Auf Android-Geräten ist in der Regel eine Vielzahl an Sensoren verbaut, die einen kontinuierlichen Strom an detaillierten Informationen liefern, und somit implizite Informationen über die direkte Benutzung des Telefons oder das allgemeine Verhalten des Nutzers liefern. Im Glossar dieser Arbeit befindet sich kurze Erläuterungen zu den innerhalb des Frameworks eingesetzten Sensoren [31]

Accelerometer	Compass
X: 0,814 m/s²	83,0º (E)
Y: 4,453 m/s²	Pitch: -26,9º
Z: 8,753 m/s²	Roll: -5,3º
Gyroscope	**Magnetic field sensor**
X: 0,212 rad/sec	X: -18,6 uT
Y: -0,259 rad/sec	Y: -13,2 uT
Z: -0,164 rad/sec	Z: -28.7 uT
Proximity sensor	**Pressure sensor**
Far	951,973 hPa
Light sensor	
8.0 lx	

Abbildung 11: Verfügbare Sensordaten

Satellites	Accuracy
11/12	12,0 m
Latitude	**Longitude**
48º3'7"	11º36'6"
Altitude	**Bearing**
637,7 m	127,0º (SE)
Speed	
0,0 km/h	

Abbildung 10: Daten des GPS-Sensors

Der Beschleunigungssensor liefert beispielsweise dreidimensionale Sensorwerte zurück, welche die Beschleunigung im Raum relativ zur Ausrichtung des Geräts zu einem ganz bestimmten Zeitpunkt beschreiben. Abbildung 12 zeigt, wie sich die Komponenten bestimmter Sensorwerte, die relativ zu Ausrichtung des mobilen Geräts stehen, in ein entsprechendes Koordinatensystem einordnen lassen. [32]

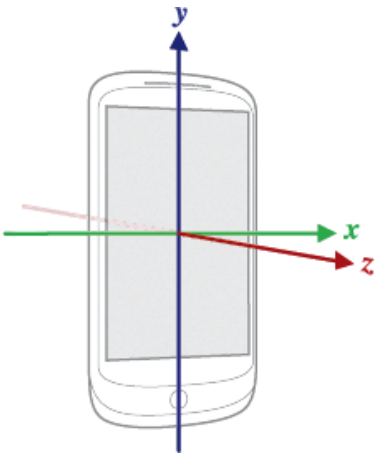


Abbildung 12: Koordinatensystem der SensorEvent API [32]

Neben den klassischen Sensoren, die in Echtzeit Informationen über die Bewegungen, Position sowie die direkte Umgebung des mobilen Geräts liefern, existieren weitere Bauteile, die im weiteren Sinne auch der Kategorie der Hardware-Sensoren zugeordnet werden können. Dazu zählen unter anderem der GPS-Sensor, welcher die absolute, globale Position eines mobilen Geräts zurückliefern kann, sowie Kamerasensoren und Mikrophone, mithelfe derer die audiovisuellen Eindrücke der Umgebung digital abgebildet werden. Ein Teil der verfügbaren Sensoren, die auf einem handelsüblichen Android-Gerät zur Verfügung stehen, können für die Identifizierung von Nutzern und Geräten eingesetzt werden. In den Kapiteln 3.1.11, 3.1.12, 3.1.13, 3.2.6, 3.2.7, 3.3.4 sowie 3.3.5 wird eingehend erläutert, wie sich die Daten der Sensoren für den kombinierten Identifizierungsvorgang nutzen lassen.

Die Sicherheitsrichtlinien von Android schränken die Auswahl an Informationsquellen ein, die ohne die Zustimmung des Nutzers verwendet werden können. Für einen Teil der verfügbaren Informationen werden sogenannte *Permissions* benötigt. Ohne *Permission* ist ein Zugriff auf die entsprechenden Informationsquellen nicht möglich. Eine *Permission* wird in einer AndroidManifest.xml - Datei spezifiziert, die jeder Android-Anwendung beigefügt werden muss. Es kann beschrieben werden, auf welche Komponenten und Informationen die Anwendung zugreifen möchte. Vor der Installation einer Anwendung wird dem Nutzer dann die Liste aller benötigter *Permissions* angezeigt. [33]

Die meisten der in dieser Arbeit vorgestellten Methoden lassen sich in zwei Gruppen gliedern: Methoden, die der Benutzeridentifizierung dienen, sowie solche, die für die Identifizierung von Geräten in Frage kommen, wie in Abbildung 13 dargestellt. Allerdings gibt es auch Informationsquellen, die der Identifizierung dienen können, aber nicht einem Benutzer oder Geräte zugeordnet werden können, sondern nur der spezifischen Kombination aus Benutzer und Gerät. Die genutzten Informationen hängen bei diesen Methoden sowohl vom Benutzer, als auch vom jeweiligen Gerat ab.

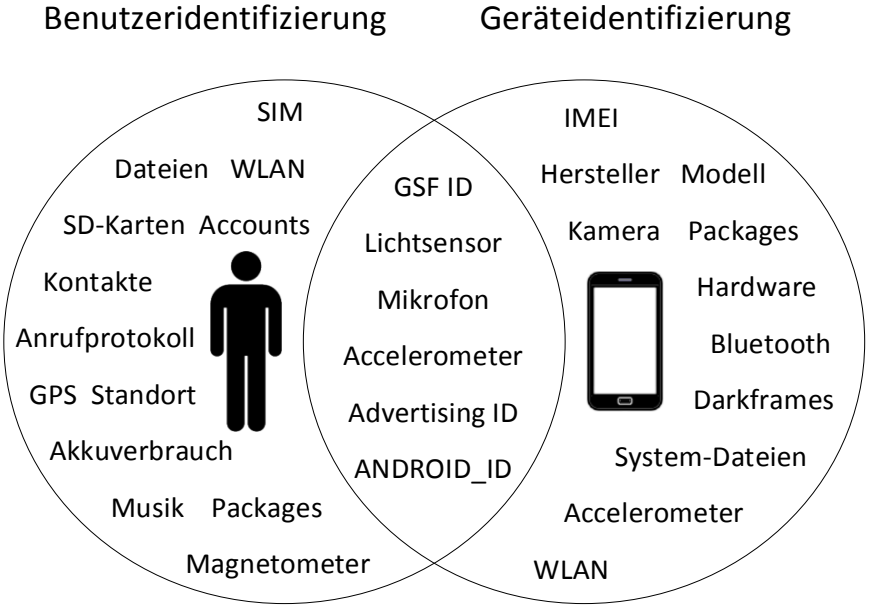


Abbildung 13: Nutzer- und Geräteidentifizierung (Details dazu in den Kapiteln 3.1 - 3.3)

Abbildung 13 zeigt eine Reihe von Methoden und Informationsquellen, die in den nachfolgenden Kapiteln näher erläutert werden. Ideen bezüglich dieser Methoden wurden mit Herrn Prof. Dr. Baumgarten, Herrn Prof. Dr. Sejun Song sowie Herrn Nils T. Kannengießer gemeinsam diskutiert. Eine Zusammenfassung der Vor- und Nachteile der jeweiligen Methoden, die sich unter Anderem während der Evaluation des Frameworks gezeigt haben, findet sich in Kapitel 3.4. Im Rahmen dieser Arbeit wurden zu allen vorgestellten Informationsquellen die entsprechenden Methoden implementiert und getestet. In Kapitel 4 ist neben der Beschreibung der Implementierungen auch eine Tabelle zu finden, welche die Package- und Dateinamen der vorgestellten Methode enthält (Tabelle 7).

3.1 Methoden zur Identifizierung von Benutzern

In den nachfolgenden Kapiteln werden Methoden erläutert und bewertet, die zur Identifizierung von Benutzern dienen. Die Informationen, die für die jeweilige Identifizierung verwendet werden, basieren auf den Benutzern selbst. Sie werden von den Benutzern teilweise selbst generiert, beschreiben dessen Verhalten oder Besitz, oder umfassen einen Teil der Daten, die der Nutzer auf dem Gerät selbst abgelegt hat. Alle Informationen, die von den in diesem Kapitel vorgestellten Methoden verwendet werden,

können einem spezifischen Benutzer, unabhängig von den benutzen Geräten, direkt zugeordnet werden.

3.1.1 Subscriber Identity Module

Mithilfe von Subscriber Identity Modulen (SIM) können Mobilfunkbetreiber (Mobile Network Operator, MNO) Nutzer mobiler Endgeräte identifizieren und authentifizieren [34]. Auf Sim-Karten sind spezifische Informationen gespeichert, die dem Nutzer zugeordnet und von Android-Anwendungen ausgelesen werden können. Zu den gespeicherten Informationen gehören unter anderem:

- Integrated Circuit Card Identifier (ICCID): 20-22 stellige Identifikationsnummer, mit deren Hilfe SIM-Karten eindeutig identifiziert werden können.
- International Mobile Subscriber Identity (IMSI): Eindeutige Identifikationsnummer, die den Nutzer gegenüber dem Mobilfunknetzwerk identifiziert.
- Abbreviated Dialing Numbers (ADN): Auf der SIM-Karte gespeicherte Telefonnummern [34]

Die Informationen, die auf den SIM-Karten gespeichert sind, können von dem Framework verwendet werden, um benutzerspezifische Profile zu erstellen. Da SIM-Karten so konzipiert wurden, dass sie den Mobilfunkbetreibern bei der Identifizierung sowie Authentifizierung von Nutzern helfen, erfüllen sie prinzipiell auch für das in dieser Arbeit vorgestellte Framework wichtige Voraussetzungen für die Nutzeridentifizierung. Da sich eine SIM-Karte meistens im Besitz von genau einer Person befindet, und SIM-Karten oftmals behalten werden, selbst wenn das mobile Gerät gewechselt wird, werden die Informationen, die auf einer SIM-Karte gespeichert sind, in erster Linie einem Benutzer (und nur indirekt einem Gerät) zugeordnet.

Universalität und Erfassbarkeit

Jedes Android-Smartphone besitzt per Definition mindestens einen SIM-Kartenslot. Die mobilen Geräte werden entweder direkt mit SIM-Karte geliefert, oder der Benutzer kauft diese nach. IMSI- und ICCID-Nummer sind somit auf so gut wie jedem Android-Smartphone verfügbar. Android Tablets hingegen sind nicht immer mit einem SIM-Kartenslot ausgestatten. In solchen Fällen greift das Framework auf andere anwendbare Methoden zurück.

Falls ein Android-Gerät mindestens einen SIM-Kartenslot besitzt, besteht für Android-Anwendungen die Möglichkeit, Informationen wie beispielsweise ICCID- und IMSI-Nummern auszulesen (Listing 1). Wie dargestellt, bietet Android spezielle Funktionen an, womit Anwendungen mithilfe des *TelephonyManagers* auf diese Daten zugreifen zu können. Das Framework, beziehungsweise die Anwendung die das Framework einbindet, benötigt allerdings (ab Android SDK-Version 4) eine *Permission*, um auf diese Informationen zugreifen zu können [35]:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE">
```

Der nachfolgende Quellcode-Ausschnitt beschreibt, wie neben der IMSI-Nummer (`getSubscriberId`) die Seriennummer (`getSimSerialNumber`) sowie die Telefonnummer (`getSimSerialNumber`) einer SIM-Karte ausgelesen werden können. Die Daten der SIM-Karte können für die Verwendung innerhalb des Android-Frameworks zudem mit Hash-Werten geschützt werden:

Die tatsächliche Telefonnummer kann allerdings nicht immer ohne weiteres abgefragt werden kann, ein Großteil der Mobilfunkbetreiber verschleiert die echte Telefonnummer des Nutzers [36]. Es existieren aber alternative Vorgehensweisen, die die Extraktion der Telefonnummer unter bestimmten Bedingungen trotzdem ermöglichen, wie in Kapitel 3.1.6 erläutert.

Um die Identifizierung durchzuführen, können die aggregierten Hash-Werte paarweise miteinander verglichen werden, wie in Kapitel 4 im Detail erläutert.

Listing 1: Methode für die Extraktion der Daten einer SIM-Karte

```
01 public static SIMItem getSIMData(Context context) {
02    TelephonyManager phoneManager = (TelephonyManager) context
03                .getSystemService(Context.TELEPHONY_SERVICE);
04    return new SIMItem(HashGenerator.hash(phoneManager.getSubscriberId()),
05                HashGenerator.hash(phoneManager.getSimSerialNumber()),
06                HashGenerator.hash(phoneManager.getLine1Number()));
07 }
```

Einmaligkeit und Persistenz

Die ICCID- und IMSI-Nummern einer spezifischen SIM-Karte sind einmalig, und somit bestens als unterstützende Information für die Nutzeridentifikation geeignet, wie in Tabelle 4 dargestellt. Einige Geräte unterstützen mehr als eine SIM-Karte. Einem Nutzer kann somit unter Umständen mehr als eine ICCID- oder IMSI-Nummer zugeordnet werden. Diese Besonderheit löst das Framework, indem jeder Benutzer in der entsprechenden Datenbanktabelle einen Verweis auf eine Menge von SIM-Karten hält.

Die Beziehung zwischen SIM-Karte und deren Benutzer kann hilfreich sein, reicht jedoch alleine nicht für eine persistente, längerfristige Identifizierung von Nutzern aus. Ein Problem, dass sich vor allem bei der längerfristigen Identifikation von Benutzern zeigt, ist, dass Nutzer den Mobilfunkanbieter und somit ihre SIM-Karte jederzeit wechseln können. In solchen Fällen muss das Framework auf andere Methoden der Identifikation zurückgreifen, um den Wechsel einer SIM-Karte zu erkennen, und die Beziehung zwischen der neuen Menge an SIM-Karten-Informationen und dem entsprechenden Benutzer aktualisieren.

3.1.2 Benutzerkonten

Auf Android-Geräte wird eine Liste von Account-Daten gespeichert und verwaltet. Diese Liste an Zugangsdaten kann von Android-Anwendungen genutzt werden, um Nutzer online zu authentifizieren. Die Anmeldeinformationen, die durch den Benutzer eingegeben werden (Benutzername und Passwort) werden von speziellen Authentifizierungsmodulen verwaltet. Anwendungen können mithilfe des *Account Managers* eine Liste aller gespeicherten Benutzerkonten anfordern [37]. Die Menge aller verfügbaren Benutzerkonten, die auf einem Gerät gespeichert wurden, kann verwendet werden, um den jeweiligen Benutzern eindeutige sowie tolerante Identifikationsnummern zuzuweisen.

Universalität und Erfassbarkeit

Nicht auf allen Android-Geräten sind spezifische Informationen zu Benutzerkonten verfügbar. Es besteht die Möglichkeit, dass einige Benutzer noch keine Anmeldeinformationen auf ihrem Smartphone oder Tablets eingeben haben. Sobald der Nutzer aber auf seinem mobilen Gerät Online-Dienste nutzen möchte, die ein Benutzerkonto erfordern, kann das Framework die Information für die Identifizierung des Nutzers abfragen. Einige Android-Anwendungen anonymisieren die Informationen, die über ein Benutzerkonto abgerufen werden können, so ist es teilweise nicht möglich, aus den verfügbaren Daten unterscheidbare Identifikationsnummern zu erstellen. Trotzdem existiert eine Vielzahl an mobilen Anwendungen, die verwendbare Benutzerkonten-Informationen bereitstellen, darunter auch einige der meistgenutzten Anwendungen weltweit. Falls benutzerspezifische Kontoinformationen auf einem Gerät verfügbar sind, eignen sich diese bestens dazu, die Erstellung einzigartiger Identifikationsnummern zu unterstützen, wie in Kapitel 3.4 dargestellt. Die aggregierten Informationen stellen Mengen von Benutzerkonten dar. In Kapitel 4 wird erläutert, wie ausgehend von Mengenvergleichen Werte für den Identifizierungsvorgang auf Android-Geräten berechnet werden können.

Android-Anwendungen benötigen für die Verwendung der Benutzerkonten-Informationen folgende *Permissions*:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS">
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS">
```

Eine Liste der Benutzerkonten-Daten kann, sobald die oben genannten *Permissions* von dem Nutzer bestätigt wurden, von der entsprechenden Anwendung mithilfe des *AccountManagers* abgefragt werden [38].

Einmaligkeit und Persistenz

Auf Android-Geräten kann davon ausgegangen werden, dass die spezifischen Informationen eines Benutzerkontos genau einem einzigen Nutzer zugeordnet werden können. Anwendungen wird es so beispielsweise ermöglicht, persönliche Informationen der Nutzer auf individuellen Konten abzulegen. Die Informationen, die von der Identifizierungsmethode verwendet werden, können somit jeweils einer Einzelperson zugeordnet werden, eine für die Verwendung innerhalb des Frameworks optimale Bedingung.

Benutzer können auf ihren Geräten verschiedenen Online-Konten unabhängig voneinander verwaltet, und zudem komplett neue Konten erstellen, diese aber auch jederzeit wieder löschen. Das Framework muss deshalb die Referenzdaten dieser Methode regelmäßig aktualisieren, und mit den Ergebnissen anderer Methoden abgleichen. [37]

3.1.3 Bluetooth

Android-Anwendungen ist es möglich, eine Liste aller gekoppelten Bluetooth-Geräte eines Geräts anzufordern und zu analysieren. Jeder Bluetooth-Adapter verfügt über eine spezifische Media Access Control-Adresse (MAC), die für jedes Gerät eindeutig sein sollte. Zudem besitzen Bluetooth-Adapter einen Anzeigenamen. [39] Die Liste aller verbundener Bluetooth-Geräten (Abbildung 14) kann verwendet werden, um dem Nutzer eine individuelle Identifikationsnummer zuzuweisen.

Universalität und Erfassbarkeit

So gut wie jedes moderne Android-Smartphone oder -Tablet unterstützt Bluetooth-Verbindungen zu anderen Geräten. Benutzeridentifikation basierend auf angeschlossenen Bluetooth-Geräten ist allerdings nur dann möglich, wenn der Benutzer neben seinem Bluetooth-fähigem Android-Gerät mindestens ein weiteres Bluetooth-Gerät besitzt, und diese miteinander verbunden sind. Sobald dies der Fall ist, können Android-Anwendungen auf die Liste verbundener Bluetooth-Geräte zugreifen. Allerdings benötigt eine Android-Anwendung mindestens eine *Permission*, um die Liste aller verbundenen Bluetooth-Geräte abzufragen. Um Bluetooth zu aktivieren, wird eine weitere *Permission* benötigt [40]:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Eine Liste aller per Bluetooth verbundenen Geräte kann, sobald die *Permission* genehmigt wurde wie in Listing 2 beschrieben auf einem Android-Gerät von einer Anwendung angefordert werden.

Die Berechnungen, die während des Identifizierungsvorgangs ausgeführt werden müssen, basieren bei dieser Identifizierungsmethode somit aus einer Menge von Einträgen. In Kapitel 4 wird beschrieben, welche Vorgehensweise das Framework in solchen Fällen wählt.

Listing 2: Anforderung einer Liste aller mit dem Gerät verbundener Geräte

```
01 BluetoothAdapter bluetoothAdapter = BluetoothAdapter
02    .getDefaultAdapter();
03 if(bluetoothAdapter != null) {
04    Set<BluetoothDevice> pairedDevices = bluetoothAdapter
05          .getBondedDevices();
06 }
```

Einmaligkeit und Persistenz

Die MAC-Adresse eines Bluetooth-Adapters sollte im Prinzip einzigartig sein. Allerdings kann es aufgrund diverser Implementierungsfehler dazu kommen, dass eine MAC-Adresse mehrmals vergeben wird. Berichte über dieses Phänomen und die damit verbundenen Probleme sind in diversen Internet-Foren zu finden. Da der Fall, dass bestimmte MAC-Adressen mehrmals vergeben wurden, allerdings nicht allzu häufig auftritt, ist die Methode trotzdem dazu geeignet, eine Identifikationsnummer mit einer gewissen Zuverlässigkeit einem Nutzer zuzuordnen.

Es kann davon ausgegangen werden, dass ein Benutzer eines Android-Geräts die meisten der konfigurierten Bluetooth-Verbindungen selbst eingerichtet hat. Es ist zudem wahrscheinlich, dass dieser Benutzer eine Untergruppe oder möglicherweise alle gekoppelten Bluetooth-Geräte besitzt. Benutzer können Bluetooth-Geräte aber auch untereinander teilen. Obwohl einige Bluetooth-Geräte wie beispielsweise Headsets oder Wearables in den allermeisten Fällen nur einem einzigen Benutzer gehören, kann es sein, dass einige Bluetooth-Geräte, wie zum Beispiel Audio-Dongles von mehreren Personen genutzt werden. Durch diese Tatsache wird zwar die Einmaligkeit einer einzigen Bluetooth-Verbindung eingeschränkt, die spezifische Kombination aus allen verbunden Geräten hingegen erscheint hingegen sehr individuell, abhängig von der jeweiligen Anzahl verbundener Geräte.

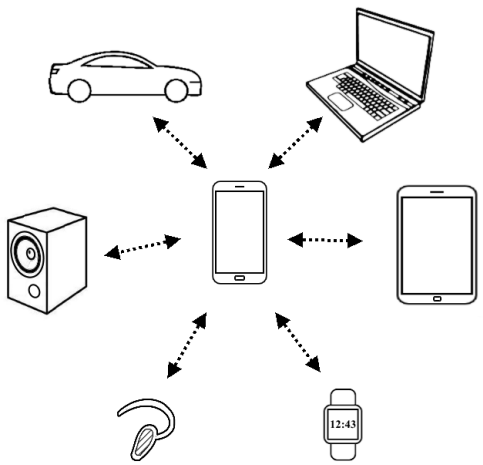


Abbildung 14: Verbundene Bluetooth-Geräte

Benutzer können neue Smartphones oder Tablets kaufen, aber auch entsorgen. Allerdings sollte die Liste der Bluetooth-Verbindungen immer noch zu einem gewissen Grad übereinstimmt, selbst wenn der Benutzer sein Smartphone oder Tablet ersetzt. Die Liste aller verbundenen Bluetooth-Geräte kann sich mit der Zeit jedoch trotzdem ändern. Falls ein Großteil der verbundenen Geräte in der Liste erhalten bleibt, kann das Framework auf die Änderungen passend reagieren und die Referenzdaten aktualisieren. Doch selbst wenn der Identifikationsprozess in diesen Szenarien in der Lage ist, Änderungen in der Menge aller angeschlossenen Geräte zu erkennen und entsprechend zu reagieren, kann eine die ausschließlich auf Bluetooth-Verbindungen basierende Identifizierung keine hundertprozentige Sicherheit gewährleisten.

3.1.4 WLAN

Verglichen mit dem Identifikationsprozess der auf angeschlossene Bluetooth-Geräte basiert, gestaltet sich die Methode, die die Liste konfigurierter WLAN-Netzwerke analysiert, relativ ähnlich. Es werden so Informationen zu den konfigurierten beziehungsweise verbundenen WLAN-Netzwerken, wie

beispielsweise MAC-Adressen und Service Set Identifiers (SSID)[6], ausgewertet. Allerdings bringt diese Methode dabei eigene Vor- und Nachteile mit sich. [41]

Universalität und Erfassbarkeit

Es lassen sich auf fast jedem Smartphone oder Tablet Informationen zu mindestens einem drahtlosen Netzwerk auslesen, wie beispielsweise zu dem WLAN-Netzwerk, welches der Benutzer bei sich zu Hause konfiguriert hat. WLAN-Adapter sind in so gut wie jedem Android-Gerät verbaut. Somit ist für diese Methode die einzige Voraussetzung, dass der Nutzer mindestens ein WLAN-Netz auf seinem Smartphone oder Tablet konfiguriert hat.

Android erlaubt es Anwendungen, eine Liste aller konfigurierter WLAN-Netzwerke mithilfe eines sogenannten *WifiManagers* abzufragen, wie nachfolgend beschrieben (Listing 3).

Listing 3: Anforderung einer Liste aller konfigurierten WLAN-Netzwerke

```
01 WifiManager wifiManager = (WifiManager) context
02           .getSystemService(Context.WIFI_SERVICE);
03 if(wifiManager!=null){
04    List<WifiConfiguration> wifiConfigs = wifiManager
05                .getConfiguredNetworks();
06 }
```

Die aggregierten Referenzdaten werden somit durch abzählbare Mengen repräsentiert, und können für den späteren Identifizierungsvorgang als Grundlage dienen, um Ähnlichkeiten paarweise zu analysieren.

Einmaligkeit und Persistenz

Die Einmaligkeit der gespeicherten Informationen über die einzelnen WLAN-Netzwerke kann variieren, abhängig von der Anzahl an Benutzern, die sich ein bestimmtes Netzwerk teilen. Dennoch ist in den meisten Fällen die spezifische Kombination der konfigurierten Netzwerke auf Smartphones und Tablets dazu geeignet, aussagekräftige Identifikationsprofile für Benutzer zu erstellen. Die Qualität und Verwendbarkeit dieser Identifikationsprofile steigt, falls der Benutzer in erster Linie Informationen über private WLAN-Netzwerke auf seinem Smartphone gespeichert hat, wohingegen die Verwendung öffentlicher Netzwerke die Eindeutigkeit der Identifikationsprofile tendenziell einschränkt. Allerdings lässt sich selbst in dem eher seltenen Fall, dass Nutzer ausschließlich öffentliche WLAN-Netzwerke nutzen, die gewonnene Information über konfigurierte Netzwerke dazu nutzen, die Menge aller Nutzer in kleinere Cluster zu unterteilen. In Kombination mit weiteren Identifizierungsmethoden kann diese Unterteilung signifikante Vorteile mit sich bringen, wie in Kapitel 2.6 näher erläutert.

Selbst wenn ein Nutzer neue WLAN-Netzwerke auf seinem mobilen Gerät konfiguriert, ändert sich die Liste als Ganzes tendenziell eher wenig. Besonders die MAC-Adressen von privaten Netzwerken bleiben normalerweise über längere Zeiträume konstant, da Router in der Regel mehrere Jahre in Gebrauch sind bevor ein neues Gerät gekauft werden muss. Sobald sich ein Teil der konfigurierten WLAN-Netzwerke für einen Benutzer ändert, wird das Framework die Referenzdaten, die in der Server-Datenbank gespeichert sind, aktualisieren. Die Einmaligkeit und Persistenz der verwendeten MAC-Adressen wird zumindest in der Theorie garantiert, allerdings gibt es aufgrund fehlerhafter Umsetzungen seitens bestimmter Hersteller gewisse Einschränkungen hinsichtlich dieser Kriterien, wie in Kapitel 3.2.2 näher erläutert. Wie auch bei Bluetooth finden sich zu dieser Problematik viele Berichte in Internet-Foren. Da sich das Problem aber trotzdem in Grenzen hält, und ein Nutzer in der Regel nicht nur ein einziges WLAN-Netzwerk nutzt,

[6] Service Set Identifier: der Name eines Service Sets bzw. Wireless Access Points

kann davon ausgegangen werden, dass mit hoher Wahrscheinlichkeit zumindest eine Teilmenge der analysierten WLAN-Netzwerke wirklich einzigartige und persistente MAC-Adressen aufweisen. Zudem kann das Framework nach einiger Zeit fehlerhafte MAC-analysieren und ignorieren, um die Fehlerrate dieser Identifizierungsmethode zu verbessern. Das Verfahren ist deshalb in Kombination mit anderen Identifizierungsmethoden trotzdem sehr gut geeignet, Nutzer unterscheidbar und somit identifizierbar zu machen.

3.1.5 Secure Digital (SD) Memory Card

Jede SD-Karte hat eine eigene Seriennummer, die von Android-Anwendungen ausgelesen werden kann. Damit dieses Identifikationsverfahren auf einem Android-Gerät eingesetzt werden kann, müssen einige Voraussetzung erfüllt sein: Zum einen muss das Android-Gerät SD-Karten unterstützen, und der Benutzer sollte zudem die SD-Karte über einen längeren Zeitraum nutzen, im besten Fall also auch dann wenn er sein Smartphone oder Tablet wechselt. [42]

Universalität und Erfassbarkeit

Obwohl Secure Digital (SD) - Karten von einigen mobilen Geräten nicht unterstützt werden, ist eine Vielzahl an Smartphones und Tablets in der Lage, mit diesen Speicherkarten umzugehen. Android-Anwendungen können die Seriennummern eingesetzter SD-Karte unter bestimmten Umständen auslesen. Android selbst bietet allerdings offiziell keine Möglichkeit an, die Seriennummer einer SD-Karte zu extrahieren. Falls eine SD-Karte verwendet wird, existieren aber eine Reihe von Workarounds, mithilfe derer sich die gewünschte Information auf bestimmten Geräten auslesen lassen. Falls eine Android-Anwendung beispielsweise auf "`/sys/class/mmc_host/mmc#/mmc#:*/cid`" oder "`/sys/block/mmcblk#/device/cid`" zugreifen kann (Wildcards müssen der Speicherkarte entsprechend ersetzt werden), lässt sich unter diesen Pfaden die Seriennummer der externen SD-Karte extrahieren. Da der tatsächliche Pfad aber je nach Modell und Konfiguration variieren kann, muss das Framework die bekannten Verzeichnisse der Reihe nach durchgehen, um die passende Datei zu finden. Die Methode wurde deshalb so konzipiert, dass nachträglich noch Verzeichnisse ergänzt werden können, um die Verfügbarkeit dieser Vorgehensweise weiter zu erhöhen. Um auf den externen Speicher zugreifen zu können, benötigt das Framework eine *Permission*:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE">
```

Einmaligkeit und Persistenz

Die Seriennummer kann verwendet werden, um den Besitzer der entsprechenden SD-Karte zuverlässig und eindeutig zu identifizieren. Durch die Seriennummer einer SD-Karte kann diese eindeutig identifiziert werden. Da sich eine SD-Karte in den allermeisten Fällen im Besitz von genau einem Benutzer befindet, und ausschließlich von diesem benutzt wird, kann die Seriennummern genau einer Person zugeordnet werden. Die Einmaligkeit der Seriennummer kann somit einen wichtigen Beitrag für die Identifizierung von Benutzern liefern.

Die meisten mobilen Geräte werden nicht mit SD-Karten ausgeliefert, weshalb die Anwender diese Speicherkarten oftmals separat kaufen müssen. Es ist wahrscheinlich, dass die Benutzer ihre SD-Karten selbst dann noch weiter behalten werden, wenn sie ihr mobiles Gerät irgendwann später entsorgen oder verkaufen werden. Da die meisten Android-Geräte dasselbe Speicherkarten-Format unterstützen, sind Benutzer häufig in der Lage, ihre alten Speicherkarten wiederzuverwenden. Die SD Association gibt zudem die Lebensdauer einer SD-Karte unter Normalbedingungen abhängig vom Hersteller mit 10 Jahren oder mehr an [43]. Somit kann davon ausgegangen werden, dass sich die Methode über einen längeren Zeitraum für die Identifizierung von Nutzern verwenden lässt.

3.1.6 Kontakte und Kommunikation

Die Menge der auf einem Android-Gerät gespeicherten Kontakte kann als benutzerspezifischer Fingerabdruck des sozialen Umfelds eines Benutzers betrachtet werden. So können Kontaktinformationen über Kollegen, Freunde und Familie verwendet werden, um einem Benutzer eine individuelle Identifikationsnummer zuzuordnen.

Auf einem Android-Gerät gibt es mehrere Möglichkeiten, Informationen über die sozialen Kontakte eines Nutzers zu extrahieren. Zum einen können Kontaktdaten entweder direkt im Gerätespeicher oder auf der SIM-Karte in einer Kontaktliste gespeichert werden [34]. Android-Anwendungen können diese Liste anfordern, unabhängig davon, wo die Kontaktinformationen tatsächlich abgelegt wurden. [44]

Den Personen in der Kontaktliste muss für die Verwendung innerhalb des Identifizierungsvorgangs eine eindeutige Identifikationsnummer zugewiesen werden. Falls vorhanden, kann diese Nummer direkt aus der Telefonnummer des Kontakts generiert werden. Um Unterschiede zwischen eigentlich identischen Telefonnummern aufgrund von Länder- und Ortsvorwahlen zu beseitigen, wird eine Java-Bibliothek eingesetzt, die von Google zu diesem Zweck entwickelt wurde: *libphonenumber.* Falls die Telefonnummer eines Kontakts nicht vorhanden sein sollte, wird auf E-Mail-Adressen zurückgegriffen, falls diese gespeichert wurden. [45]

Neben den gespeicherten Kontakten eines Nutzers lassen sich auch Informationen über das Anrufprotoll und der SMS-History auslesen. Zudem erkennt das Framework, falls sich zwei Benutzer einen WLAN-Zugang geteilt haben, oder durch die gemeinsame Verwendung von Bluetooth-Geräten in Verbindung gebracht werden können. Das in dieser Arbeit vorgestellte Framework greift somit auf fünf Informationsquellen zurück, um Daten über das soziale Umfeld eines Nutzers zu extrahieren: Kontaktliste, Anrufprotoll, SMS-History, gemeinsam genutzte WLAN-Verbindungen und Bluetooth-Adapter. Die gewonnenen Informationen aus diesen fünf Quellen können kombiniert werden, um die Aussagekraft der Identifizierungsmethode zu steigern. So kann aus der Information, dass zwei Personen sich kennen (oder zumindest in irgendeiner Weise in Verbindung gebracht werden können), durch die Häufigkeit und die zeitliche Verteilung der Anrufe und Kurznachrichten eine spezifische Gewichtung verliehen werden. Personen, mit denen ein Benutzer häufiger kommuniziert, erhalten eine höhere Gewichtung. Zudem wird, umso weiter der letzte Anruf bzw. die letzte Nachricht zwischen den beiden Personen in der Vergangenheit liegt, die Gewichtung der Information über die jeweilige Beziehung reduziert. Durch dieses Verfahren wird nicht nur allein der topologische Aufbau des sozialen Netzwerks eines Nutzers als Informationsquelle für die Identifizierung verwendet, sondern auch die Intensität der Kommunikation zwischen Benutzern untereinander berücksichtigt.

Universalität und Erfassbarkeit

Viele Benutzer von mobilen Geräten speichern auf ihren Smartphones Kontaktdaten. Obwohl das Smartphone mittlerweile deutlich mehr Funktionen bietet als nur das Telefonieren oder die Möglichkeit, Nachrichten zu verschicken, dient es den allermeisten Benutzern in erster Linie als bevorzugtes Kommunikationsmittel. Es kann somit davon ausgegangen werden, dass die in diesem Kapitel vorgestellte Methode auf annähernd allen Smartphones angewendet werden kann.

Die Einträge der Anruf- und Kontaktlisten können mithilfe eines *ContentResolver* auf einem Android-Gerät ausgelesen werden. Die Art der Anfrage wird jeweils durch einen Uniform Resource Identifier (URI) definiert, welcher beim Aufruf der entsprechenden Methode übergeben werden muss. Die benötigten Informationen können von Android-Anwendungen verwendet werden, wobei hierfür allerdings bis zu drei *Permission* benötigt werden.

Die benötigten *Permissions* für diese Identifizierungsmethode lauten wie folgt:

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission. READ_SMS"/>
<uses-permission android:name="android.permission. READ_CALL_LOG"/>
```

Die Methode kann so konfiguriert werden, dass nur ein Teil der *Permissions* erforderlich ist, wodurch dann dementsprechend nicht alle Informationsquellen genutzt werden können. Die Unterscheidbarkeit der Benutzer und somit die Qualität der Identifizierung steigt in der Regel mit der Menge der Verfügbaren Daten. In jedem Fall wird für diese Methode auf Android mindestens eine *Permission* benötigt.

Einmaligkeit und Persistenz

Die Gesamtheit aller Kontakte, und die Gewichtung der einzelnen Kontakte anhand des Anrufprotokolls und der SMS-History ergibt in ihrer individuellen Zusammenstellung eine einzigartige Informationsquelle, anhand derer sich Benutzer unterscheiden und identifizieren lassen. Wichtig ist hierbei, dass der Benutzer bereits ausreichend Kontakte auf seinem mobilen Gerät gespeichert hat, und mit diesen im besten Fall regelmäßig kommuniziert.

Da das soziale Umfeld und vor allem der engere Freundeskreis eines Benutzers sich in der Regel nur langsam ändert, kann davon ausgegangen werden, dass die Informationen, auf denen die Identifizierungsmethode basiert, über einen längeren Zeitraum relativ konstant bleibt, und somit für die Identifizierung eines Benutzers geeignet ist. Falls ein Nutzer Kontaktinformationen zu Familienmitgliedern gespeichert hat, und mit diesen regelmäßig telefoniert, kann das Framework auf einen persistenten, benutzerspezifischen Kern an Kontaktinformationen zurückgreifen. Das Framework, das in dieser Arbeit vorgestellt wird, erkennt zudem kleinere Änderungen, und aktualisiert dementsprechend die Referenzdaten. Es kann jedoch vorkommen, dass sich das Umfeld eines Benutzers, beispielsweise durch einen Umzug oder Arbeitsplatzwechsel, in kurzer Zeit stark ändert. In diesen Fällen kann das Framework auf die Ergebnisse der anderen Identifizierungsmethoden zurückgreifen, um die entbrechenden Daten zu aktualisieren.

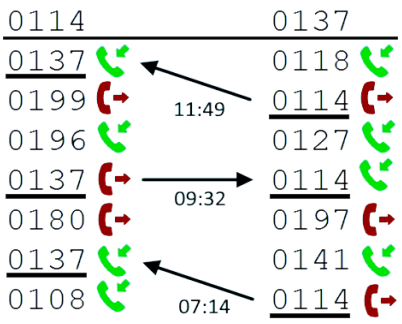


Abbildung 15: Vergleich von Anrufprotokollen (Beispiel)

Neben den bereits beschriebenen Informationen, die über das sozialen Umfeld eines Nutzers gesammelt werden können, lässt sich unter bestimmten Umständen noch eine weitere wichtige Information aus den Anrufprotokollen und der SMS-History extrahieren. Android-Anwendungen ist es in der Regel nicht möglich, die Telefonnummer eines Benutzers direkt abzufragen. Es erscheint sinnvoll, die Anrufprotokolle zweier Benutzer zu analysieren, um so die jeweiligen Telefonnummern der Benutzer auf andere Weise zu extrahieren. Wie in Abbildung 15 dargestellt, können die Anrufprotokolle zweier Benutzer direkt miteinander verglichen werden. Bei diesem Vorgang werden von beiden Benutzern alle übereinstimmenden Einträge mit identische Anrufzeiten und gleicher Anrufdauer extrahiert. Da es Aufgrund von Verzögerungen innerhalb des Mobilfunknetzes zu Zeitunterschienden kommen kann, wird der Abgleich mit einer Toleranz von wenigen Sekunden durchgeführt. Analog dazu lassen sich auch Paare von versendeten und empfangenen Nachrichten extrahieren. Anschließend wird überprüft, ob bei einigen der gesammelten Einträge die Telefonnummern bei beiden Benutzern jeweils identisch bleiben. Ab einer gewissen Anzahl an Anrufen bzw. Nachrichten kann mit einer sehr hohen Wahrscheinlichkeit davon ausgegangen werden, dass die beiden Personen in diesen Zeitabschnitten miteinander kommuniziert haben. Die Telefonnummern beider Benutzer lassen

sich in diesem Fall wechselseitig aus den Anrufprotokollen bzw. der SMS-History extrahieren.

3.1.7 Mobile Anwendungen

Die Anzahl an Android-Anwendungen, die jährlich von Google Play heruntergeladen und auf mobilen Geräten installiert werden, steigt stetig. Zudem nimmt das Angebot an Anwendungen, die für die Android-Plattform entwickelt wurden, kontinuierlich zu. So waren Ende 2014 über 1 400 000 Anwendungen auf Google Play verfügbar [46]. Zudem überstieg schon 2013 die Anzahl der weltweit heruntergeladenen Anwendungen die 50 Milliardengrenze, wie in Abbildung 16 dargestellt [47].

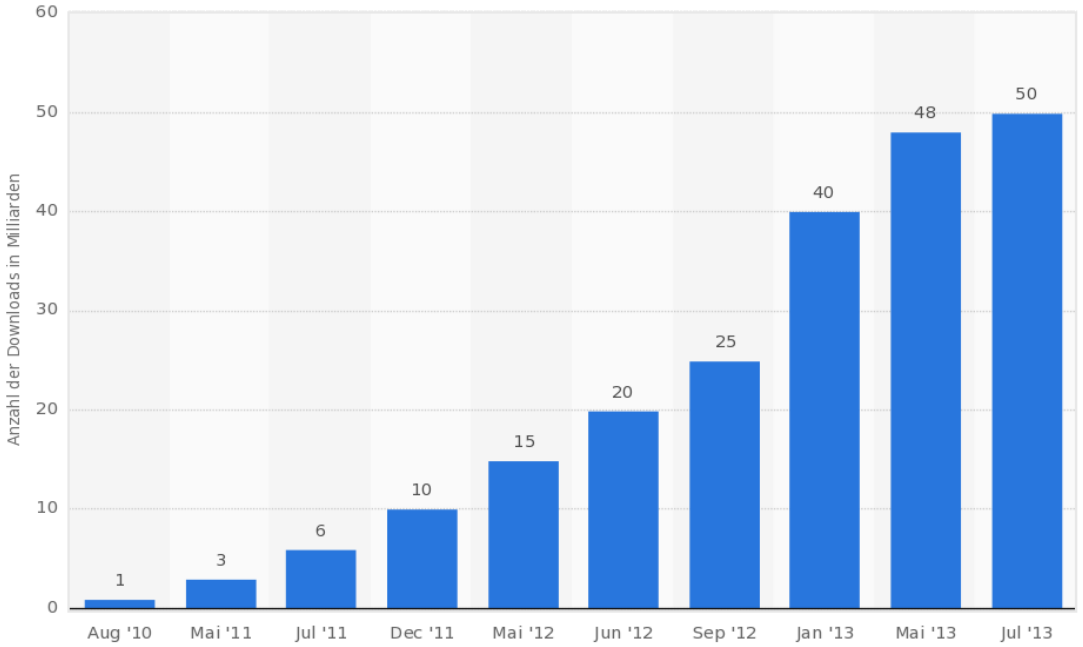


Abbildung 16: Entwicklung der Downloadzahlen von Anwendungen auf Google Play [47]

Nach einer Studie, die im Auftrag von Google Inc. von der Ipsos MediaCT in Partnerschaft mit der Mobile Marketing Association und dem Interactive Advertising Bureau durchgeführt wurde, hatte bis Ende 2013 ein durchschnittlicher Android-Nutzer im Schnitt 23 Anwendungen auf seinem Smartphone installiert [48].

Das Framework untersucht sowohl die Auswahl installierter Anwendungen sowie deren individuelle Benutzung auf einem mobilen Gerät.

Universalität und Erfassbarkeit

Die Methode, welche die installierten Anwendungen untersucht, teilt die Gruppe aller Nutzer in kleinere Cluster ein: Nutzer, die exakt dieselben Android-Anwendungen auf ihrem mobilen Gerät installiert haben, werden in einem Cluster zusammengefasst. Im optimalen Fall enthält ein Cluster genau einen Nutzer. Da Nutzer, die noch gar keine eigenen Anwendungen installiert haben, in einem eigenen Cluster gruppiert werden, kann diese Methode auf jedem Android-Gerät angewendet werden, unabhängig davon wie viele Anwendungen ein Nutzer installiert hat.

Android-Anwendungen können eine Liste allen installierten Packages anfordern (Listing 4). Zudem lässt sich abfragen, ob es sich bei einem Package um eine vorinstallierte Anwendung, also beispielsweise einem System-Package, oder aber um eine vom Nutzer installierte Anwendung handelt. Die Android-Anwendung benötigt keine besondere *Permission*, um diese Information anzufordern. Nachfolgend wird gezeigt, wie auf Android-Geräten eine Liste aller Packages angefordert werden kann. Das Framework unterscheidet zwischen Anwendungen, die vom Nutzer selbst installiert wurden, sowie Packages, die als

Teil des Systems angesehen werden können. Jeder Eintrag der Liste enthält ein Feld, welches beschreibt, um welchen Typ es sich jeweils handelt. Falls dieser Eintrag den Wert `ApplicationInfo.FLAG_SYSTEM` nicht enthält, können Informationen über die mobile Anwendung für die Nutzeridentifizierung verwendet werden.

Listing 4: Anforderung einer Liste aller mobilen Anwendungen auf einem Gerät

```
01 List<PackageInfo> installedPackages = context.getPackageManager()
02          .getInstalledPackages(0);
03 for (int i = 0; i < installedPackages.size(); i++) {
04    PackageInfo p = installedPackages.get(i);
05    [...]
06    ApplicationInfo ainfo = context.getPackageManager()
07    .getApplicationInfo(p.packageName, 0);
08    if ((ainfo.flags & ApplicationInfo.FLAG_SYSTEM) != 0) {
09          systemApp = true;
10    }
11    else{
12          systemApp = false;
13    }
14    [...]
15 }
```

Einmaligkeit und Persistenz

Die tendenzielle Einmaligkeit der benutzerspezifischen Kombination aus installierten Anwendungen hängt mit der Anzahl der installierten Anwendungen zusammen. Umso mehr Anwendungen ein Nutzer auf seinem mobilen Gerät installiert, desto einmaliger wird seine Auswahl an Anwendungen im Vergleich zu den Installationen anderer Benutzer. Wie zu Beginn dieses Kapitels erwähnt, existieren mittlerweile über 1 400 000 Anwendungen im Google Play Store, wobei ein durchschnittlicher Android-Nutzer in etwa 23 Anwendungen auf seinem mobilen Gerät installiert hat. Diese Zahlen sprechen für eine sehr hohe Einmaligkeit der benutzerspezifischen Kombinationen der installierten Anwendungen. Für einen durchschnittlichen Android-Nutzer gibt es insgesamt mehr als $8.87916054354 \times 10^{118}$ Möglichkeiten, aus 1 400 000 verfügbaren Anwendungen genau 23 auszuwählen bzw. diese auf seinem Gerät zu installieren:

$$\frac{1400000!}{((1400000 - 23)! * 23!)} \approx 8.87916054354 \times 10^{118}$$

Die Wahrscheinlichkeit, dass zwei (durchschnittliche) Nutzer exakt dieselben Anwendungen installieren, wäre bei einer Gleichverteilung der Häufigkeiten dieser Kombinationen so gut wie ausgeschlossen. Die Einmaligkeit der einzelnen Kombinationsmöglichkeiten wird in Wirklichkeit aber durch die tatsächliche Verteilung der Downloadzahlen der einzelnen Android-Anwendungen relativiert. Abbildung 17 zeigt die entsprechende Verteilung Ende 2014.

Bei den Anwendungsinstallationen handelt es sich annähernd um eine Pareto-Verteilung. Das bedeutet, dass schätzungsweise 10% aller Anwendungen für etwa 90% aller Downloads verantwortlich sind. Die Installationen, die sich auf den mobilen Geräten der meisten Nutzer finden lassen, konzentrieren sich somit nur auf einen Bruchteil der tatsächlich verfügbaren Anwendungen. Die vorliegende Pareto-Verteilung schränkt somit die Einmaligkeit einer benutzerspezifischen Auswahl an Anwendungen tendenziell ein. Trotzdem erzielt die Methode, wie auch in der Evaluation gezeigt, sehr gute Ergebnisse, wenn es um die Identifizierung von Nutzern geht. Dies liegt vor allem daran, dass trotz der eher ungünstigen Verteilung der Downloadzahlen die absolute Menge an Downloads und der verfügbaren

Anwendungen so hoch ist, dass die daraus resultierende Dispersion der Kombinationen die Identifizierung eines Nutzers oder zumindest ein sehr feines Clustering zulässt.

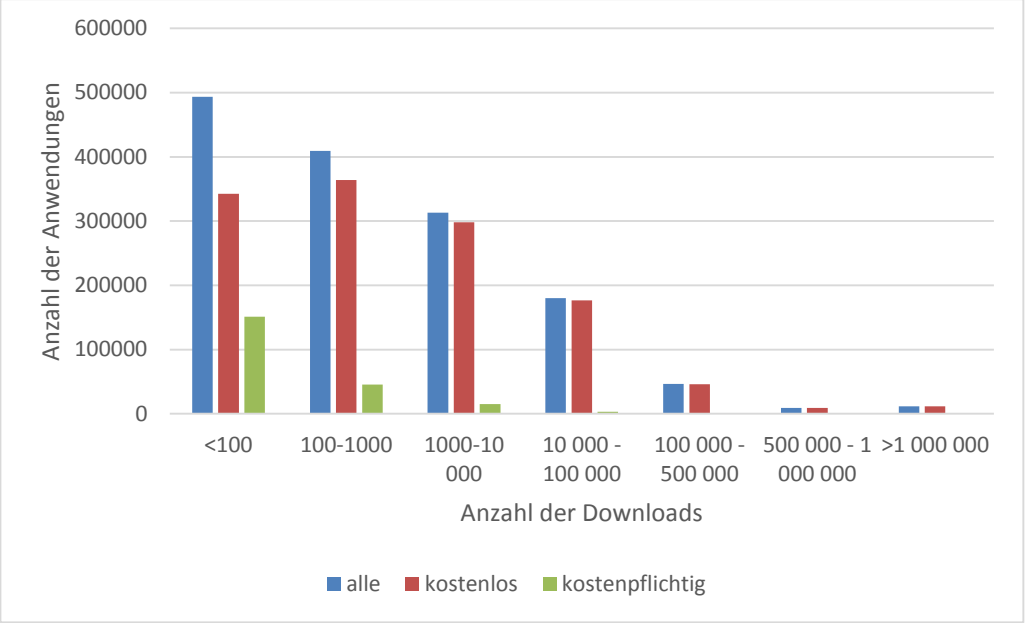


Abbildung 17: Verteilung der Downloadzahlen auf Google Play

Viele Anwendungen werden im alltäglichen Leben von den Nutzern regelmäßig verwendet. Selbst wenn ein Nutzer sein mobiles Gerät wechselt, ist davon auszugehen, dass der Nutzer viele der oft genutzten Anwendungen neu installieren wird, oder die Anwendungen sogar auf all seinen Geräten synchronisiert.

3.1.8 Nutzerdateien

Viele Benutzer speichern Dateien auf ihren Smartphones oder Tablets, wie zum Beispiel persönliche Dokumente, Notizen, Bilder oder auch Filme. Pseudonymisierte Hash-Werte dieser Dateien können generiert und gesammelt werden, um Benutzer anhand der spezifischen Kombination von gespeicherten Nutzerdaten zu identifizieren.

Universalität und Erfassbarkeit

Handelsübliche Smartphones werden mittlerweile mit vergleichsweise viel internen Speicherplatz ausgestattet. Den Nutzern wird dadurch ermöglicht, große Datenmengen auf ihren Smartphones zu speichern. Sobald ein Nutzer persönliche Dateien auf seinem Gerät gespeichert hat, kann das Framework die gespeicherten Informationen nutzen, um benutzerspezifische Profile zu erstellen. Damit Android-Anwendungen auf den Speicher eines Android-Geräts zugreifen können, ist allerdings folgende *Permission* erforderlich:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Einmaligkeit und Persistenz

Obwohl einige dieser Daten mit Sicherheit auch von unterschiedlichen Benutzern auf mehreren Geräten geteilt und gespeichert wurden, können durch diese Vorgehensweise spezifische Cluster von Benutzern ermittelt werden, die dabei helfen benutzerspezifische Profile zu erstellen. Besonders Dateien, die vom jeweiligen Benutzer selbst erstellt wurden, stellen einzigartige Informationsquellen für diese Identifizierungsmethode dar. Viele mobile Anwendungen ermöglichen es Nutzern, Dateien direkt auf ihrem Smartphone oder Tablet zu erstellen oder zu ändern.

Falls der Benutzer diese spezifische Dateien mit anderen Geräten, wie zum Beispiel mit einem neu

gekauften Gerät, synchronisiert, ist das Identifizierungsverfahren sogar in der Lage, Benutzer anhand ihrer Dateien geräteübergreifend zu identifizieren.

3.1.9 Musik

Viele Android-Geräte können in der Regel auch als mobile Musikabspielgeräte verwendet werden. Die Benutzer können Musik auf ihren Smartphones oder Tablets speichern. Es bietet sich die Möglichkeit an, aus den gespeicherten Musik-Daten individuelle Fingerabdrücke zu generieren, welche den Musikgeschmack des Benutzers repräsentieren.

Universalität und Erfassbarkeit

Nach einer Studie des Bundesverband Musikindustrie e. V. hörten im Jahre 2012 fast 80% aller Smartphone-Nutzer auf ihrem Geräten unterwegs Musik, bei den jüngeren Smartphone-Besitzern (16-24 Jahre) sogar bis zu 90%. Damit ist das Smartphone das beliebteste mobile Musik-Abspielgerät, sogar noch vor MP3-Playern. [49]

Ein beträchtlicher Teil der Nutzer hat die Musik-Dateien direkt auf dem Smartphone oder Tablet gespeichert. Die benötigten Informationen über den individuellen Musikgeschmack des Nutzers lassen sich somit auf vielen mobilen Android-Geräten finden.

Auf dem Android-Betriebssystem ist es Anwendungen möglich, bestimme Meta-Daten zur gespeicherten Musik, wie beispielsweise Musiktitel, Künstlernamen oder den Titel von Alben abzufragen. Auf diese Informationen, die in Form von Metadaten gespeichert sind, kann mit Hilfe der *MediaMetadataRetriever*-Klasse und einem *Content-Provider* zugegriffen werden [50]. So kann zum Beispiel, wie in unserem Framework implementiert, eine Liste aller Künstler erstellt werden, deren Musiktitel auf einem Gerät gespeichert sind. Listing 5 zeigt ein entsprechendes Quellcode-Beispiel. Die Listeneinträge werden entsprechend der Häufigkeit der Titel, die dem jeweiligen Künstler zugeordnet werden können, gewichtet. Künstler, die ein Nutzer besonders gerne hört, gehen somit bei der späteren Analyse stärker in die Berechnung von Ähnlichkeiten mit ein. Die Ähnlichkeit zwischen Listen, und der Vergleich der jeweiligen Gewichtungen können somit den Identifikationsprozess von Benutzern unterstützen.

Listing 5: Vorgehensweise, um die Künstler aller Musikstücke zu extrahieren

```
01 Cursor cursor = context.getContentResolver().query(
02          MediaStore.Audio.Media.EXTERNAL_CONTENT_URI, null, null, null,
03          AudioColumns.ARTIST + " ASC");
04 if (cursor != null) {
05    while (cursor.moveToNext()) {
06          String artist = cursor.getString(cursor
07                    .getColumnIndex(AudioColumns.ARTIST));
08          [...]
09    }
10 }
```

Einmaligkeit und Persistenz

Diese Musiksammlungen repräsentieren durch ihre spezifische Zusammenstellung gewissermaßen den charakteristischen Musikgeschmacks des Nutzers. Die Auswahl der Musik ist ein individuelles Phänomen, und kann daher als unterstützende Informationsquelle für die Benutzeridentifikation dienen. Die Einmaligkeit der gesammelten Information hängt dabei stark vom jeweiligen Benutzer ab, wobei Benutzer mit einem sehr individuellen Musikgeschmack eindeutiger identifiziert werden können als Mainstream-Hörer. Es ist aber in jedem Fall möglich Cluster dem Musikgeschmack entsprechend zu erstellen.

Die individuellen Präferenzen, die sich in den gespeicherten Häufigkeitsverteilung der Musiktitel eines

Künstlers wiederfinden lassen, verändern sich in der Regel langsam, analog zum Musikgeschmack vieler Nutzer. Allerdings hängt dies natürlich auch von den individuellen Hörgewohnheiten des jeweiligen Nutzers ab. Das Framework ist in der Lage, die Informationen über den Musikgeschmack eines Nutzers zu aktualisieren, falls Änderungen festgestellt wurden.

3.1.10 Akku

Android-Anwendungen können detaillierte Informationen über den aktuellen Zustand des Akkus anfordern. Es ist unter anderem möglich, den aktuellen Ladezustand des Akkus, die Ladedauer sowie das Temperaturniveau des Akkus zu überwachen. Diese Informationen können in Kombination mit Zeitstempeln dazu verwendet werden, ein individuelles Nutzungsprofil für jeden Benutzer zu erstellen. In bestimmten Zeitabschnitten kann der Zustand des Akkus signifikante Informationen über die Nutzung des Smartphones liefern, zum Beispiel, wenn der Benutzer auf seinem mobilen Gerät Videos während der Kaffeepausen abspielt, oder sein Gerät als Navigationsgerät nutzt. Aus den gemittelten Zeitstempeln und die Dauer der Zeitabschnitte, in denen ein Benutzer sein mobiles Gerät jeden Tag lädt, können auch individuelle Verhaltensprofile für jeden Nutzer erstellt werden. Die gewonnenen Informationen werden von dem Framework verwendet, um den Identifikationsprozess zu unterstützen. [51]

Einmaligkeit und Persistenz

Die Zeit, wann und für wie lange ein Benutzer sein mobiles Gerät lädt, ist sehr individuell. Die Aufzeichnungen des Energieverbrauchs eines mobilen Geräts repräsentieren viele spezifische Eigenschaften eines Nutzers. Die gesammelten Referenzdaten können somit als einmalig betrachtet werden.

Die Persistenz dieser Methode hängt stark von den Eigenschaften des jeweiligen Benutzers ab. Angenommen ein Benutzer geht jeden Tag seinen geregelten Tagesablauf nach, und führt regelmäßig gewisse Routinen aus (wie z.B. Smartphone-Nutzung in der Kaffeepause), kann davon ausgegangen werden, dass die gesammelten Referenzdaten über einen längeren Zeitraum für diesen Benutzer charakteristisch bleiben. Es ist natürlich vorstellbar, dass es Benutzer gibt, die in ihrem Alltag weniger Routinen folgen. In diesem Falle wird es komplizierter, verwendbare Referenzdaten zu sammeln, die mit dem zukünftigen Verhalten des Benutzers abgeglichen werden können. Das Framework kann Unregelmäßigkeiten erkennen, und passt die Gewichtung der Methode entsprechend an.

Universalität und Messbarkeit

Jedes *mobile* Android-Gerät besitzt per Definition einen Akku. Benutzer müssen ihre Geräte von Zeit zu Zeit laden, und erzeugen durch Benutzung ihres Geräts individuelle Energieverbrauchskurven. Die benötigten Informationen sind somit auf jedem Android-Gerät vorhanden.

Anwendungen können auf Android-Smartphones einen sogenannten *Broadcast-Receivers* erstellen, der die Anwendung über Änderungen des aktuellen Zustands der Batterie informiert. Die Anwendung benötigt hierfür allerdings eine *Permission* [51]:

```
<uses-permission android:name="android.permission.BATTERY_STATS"/>
```

3.1.11 Schritt-Erkennung

Die bereitgestellten Daten des Beschleunigungsmessers eines Android-Geräts können analysiert werden, um bestimmte Tätigkeiten oder Bewegungen des Benutzers zu erkennen. Eine dieser Aktivitäten, die eine Android-Anwendung beobachten und analysieren kann, ist ein Spaziergang bzw. die einzelnen Schritte, die ein Nutzers mit dem Gerät tätigt, vorausgesetzt der Nutzer trägt sein Smartphone bei sich in der Hosentasche.

Universalität und Messbarkeit

Beschleunigungssensoren sind auf den meisten handelsüblichen Android-Geräten vorhanden. Die Unterstützung von Sensoren gehörte schon frühzeitig zum Konzept von Android. So war bereits auf dem ersten erhältlichen Android-Smartphone, HTC Dream, ein Beschleunigungssensor vorhanden. Der Sensor wird bereits ab Android 1.5 bzw. API Level 3 von dem Android Betriebssystem unterstützt. Zusätzlich bieten einige Android-Geräte ab KitKat (Android 4.4) bestimmten Hardwarekomponenten, welche die Schritte eines Nutzers erkennen und zählen können, ohne das zusätzliche Software-Implementierungen nötig sind. Das Framework greift je nach Android-Version auf den Beschleunigungssensor oder aber die integrierte Schritterkennung von Android zu, je nachdem welche Informationsquellen auf dem Gerät zur Verfügung stehen

Die Daten, die von dem Beschleunigungssensoren geliefert werden, können mithilfe eines *SensorEventListeners* dem Framework übergeben werden, wobei Android keine zusätzliche *Persmission* für das Abfragen der Daten erfordert [52]. Die Auswertung der Schritterkennung kann somit ohne Einschränkung auf handelsüblichen Android-Geräten durchgeführt werden. Die Qualität der von dieser Identifizierungsmethode verwendeten Information hängt hierbei stark mit den Gewohnheiten des Nutzers zusammen. Wie auch bei anderen biometrischen Methoden, die das Nutzerverhalten analysieren, werden bessere Ergebnisse erzielt falls ein Nutzer einen teilweise routinierteren Tagesablauf verfolgt.

Die Identifizierungsmethode kann in erster Linie auf Smartphones und Phablets Anwendung finden. Da Tablets in der Regel nicht direkt am Körper getragen werden, gestaltet sich die Erkennung von Nutzerschritten deutlich komplizierter, und wird teilweise sogar ganz unmöglich. Auf Android-Geräten, die der Nutzer in der Regel nicht in der Hosentasche trägt, muss deshalb auf andere Identifizierungsmethoden zurückgegriffen werden.

Einmaligkeit und Persistenz

Die individuelle Anzahl an Schritten, die ein Benutzer in der Regel pro Tag tätigt, kann mithilfe des Beschleunigungssensors von einer Android Anwendung gezählt werden. Anschließend ist es möglich, Statistiken über die Verteilung der Schritte und somit über das individuelle Verhalten eines Nutzers zu erstellen. Die gesammelten Informationen zu den gemittelten Zeiten des ersten und des letzten Schritt des Benutzers am Tag, als auch die Verteilung der Schritte insgesamt, kann verwendet werden, um benutzerspezifische Profile zu erstellen. Diese können mit der entsprechenden Identifikationsnummer verknüpft werden. Die Informationen, die durch die Schritterkennung analysiert werden, sind prinzipiell sehr individuell und deshalb unterscheidbar. Allerdings lässt die Methode für sich allein genommen in der Praxis noch keine eindeutige Identifizierung zu, da für eine exakte Identifizierung das Verhalten eines Nutzers über einen sehr langen Zeitraum beobachtet werden müsste. Als ergänzende Methode, die im Zweifelsfall den Identifikationsprozess um nützliche Informationen erweitert, kann das Verfahren durchaus sinnvoll für das Framework eingesetzt werden.

Die Anzahl der Schritte, die ein Nutzer an einem bestimmten Tag tätigt, kann abhängig von den Gewohnheiten des Nutzers variieren. Die Identifizierungsmethode liefert deshalb bessere und persistentere Ergebnisse, falls die kumulierten und gemittelten Werte für jeden Tag anstelle einzelner Messwerte verwendet werden. Dies gilt vor allem dann, wenn die Identifizierungsmethode über einen längeren Zeitraum eingesetzt wird.

Falls ein Nutzer seinen Lebensstil in etwa beibehält, und somit jeden Tag in etwa die gleiche Strecke mit einer konstanten zeitlichen Verteilung zu Fuß zurücklegt, wird die Methode über diese Zeiträume hinweg ähnliche Verhaltensmuster aufzeichnen können. Der Nutzer kann somit längerfristig den gesammelten

Referenzdaten zugeordnet werden. Die Persistenz der verwendeten Information hängt dabei stark von den Gewohnheiten des Nutzers ab, beispielsweise davon wieviel Routine in dem durchschnittlichen Alltags eines Nutzers steckt.

Um den Stromverbrauch zu minimieren und die Genauigkeit zu erhöhen, ist es sinnvoll, die Falsch-Positiv-Rate des Schritt-Detektors so weit wie möglich zu reduzieren. Die Informationen des Lichtsensors können verwendet werden, um die Analyse des Beschleunigungs-Signals zu starten oder zu deaktivieren, da das Gerät typischerweise wenig Licht in der Innentasche des Nutzers empfängt. Jeder Schritt des Nutzers erzeugt ein bestimmtes Muster, welches mit dem Beschleunigungssensor beobachtet werden kann. Es ist möglich, Informationen über die Anzahl sowie die zeitliche Verteilung der Schritte eines Nutzers aus dem Beschleunigungssignal zu extrahieren.

3.1.12 Positionsdaten

Auf handelsüblichen Android-Geräten besteht die Möglichkeit, die Menge aller Orte, die ein Nutzer in regelmäßigen Abständen besucht, zu erfassen. Die in diesem Kapitel vorgestellte Vorgehensweise verwendet diese Information, um Nutzer über längere Zeiträume zu identifizieren.

Universalität und Messbarkeit

Es gibt eine ganze Reihe an Möglichkeiten, den aktuellen Standort eines Nutzers auf Android-Geräten zu bestimmen. Neben der klassischen Lokalisierung mittels des Global Positioning System (GPS) kann auch auf Informationen über umliegende Funknetze zurückgegriffen werden. Die in der Nähe liegenden Mobilfunkmasten und WLAN-Netzwerke erlauben es Android, mithilfe der *Google Locations API* die Position des Smartphones oder des Tablets zu bestimmen, selbst wenn kein GPS-Empfang vorhanden ist. Die Signalstärken der umliegenden Funknetze (Abbildung 8) ermöglichen eine präzise Ortung mittels Triangulation. Zum Einsatz kommt hier der sogenannte *Fused Location Provider,* welcher von Google für diesen Zweck zur Verfügung gestellt wird [53].

Die folgenden *Permissions* werden von dem Framework angefordert, um die Ortungsfunktionalität der Android-Geräte bestmöglich nutzen zu können [53]:

```
<uses-permission android:name="android.permission.ACCESS_GPS" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name=
"com.google.android.providers.gsf.permission.READ_GSERVICES">
```

Außerdem kann festgelegt werden, mit welcher Genauigkeit die Ortskoordinaten angefragt werden sollen. Folgende *Permissions* kommen hierfür zum Einsatz [53]:

```
<uses-permission android:name="android.permission.ACCESS_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Einmaligkeit und Persistenz

Einige der charakteristischsten, erfassbaren Informationen, die den Alltag eines Benutzers beschreiben sind die verschiedenen Orte, die der Benutzer in regelmäßigen Abständen besucht. Die Arbeit, das Zuhause des Nutzers sowie die in der Nähe gelegenen Einkaufsmöglichkeiten stellen eine einzigartige, benutzerspezifische Menge an Informationen dar. Die Kombination der Positionsdaten ergibt somit für jeden Nutzer ein sehr individuelles Bild, vorausgesetzt das Framework konnte bereits genug Ortskoordinaten des Nutzers erfassen.

Da ein Nutzer in den allermeisten Fällen einen festen Wohnsitzt hat, und diesen in der Regel eher selten

wechselt, kann von einer anhaltenden Persistenz dieser Identifizierungsmethode ausgegangen werden. Arbeitsplatzwechsel sind in der Regel auch nicht übermäßig häufig, ein weiterer Umstand, der zu der Beständigkeit der verwendeten Informationsquellen beiträgt. Neben dem Wohnsitz und dem Arbeitsplatz gibt es eine ganze Reihe weiterer Orte, die ein Nutzer regelmäßig und über längere Zeiträume hinweg besucht. Ein Großteil dieser Orte liegt höchstwahrscheinlich in der näheren Umgebung des Wohnsitzes, wie zum Beispiel Einkaufsmöglichkeiten, Kontakte in der Nachbarschaft oder auch Orte die für Freizeitaktivitäten in Frage kommen. Die gesammelten Positionsdaten bleiben in der Regel über längere Zeiträume konstant, vorausgesetzt der Benutzer wechselt nicht seinen Wohnsitz oder Arbeitsplatz. Falls der Nutzer aber umziehen oder sich eine neue Arbeitsstelle suchen sollte ist das Framework auf die Ergebnisse anderer Identifizierungsmethoden angewiesen, um den Nutzer trotzdem noch identifizieren zu können. Die gesammelten Standortdaten müssen in diesem Fall durch neu aggregierte Referenzdaten ersetzt werden.

Datenaggregation

Der erste Schritt eines Frameworks, welches diese Identifizierungsmethode verwendet, ist das Sammeln von Daten über die aktuellen Standorte des Benutzers in bestimmten Zeitintervallen. Android-Geräte haben mehrere Möglichkeiten, den aktuellen Standort zu erfassen: basierend auf GPS-Daten oder aber auch basierend auf Daten zu den verfügbaren Mobilfunk- oder WLAN-Netzen. Durch Triangulation kann der jeweilige Standort annähernd berechnet werden. Googles Lokalisierungsdienst bietet hierbei für Android Anwendung eine bestmögliche Kombination der Positionsdaten verschiedenen Quellen an [53].

Aufgrund der Tatsache, dass die vom Framework gesammelten Standorte mit anderen Mengen von Positionsdaten verglichen werden müssen, ist es erforderlich, die gesammelten Positionen in einer effizienten Datenstruktur zu speichern. Es stehen viele bekannte baumähnliche Datenstrukturen zur Verfügung, welche unter Umständen zu diesem Zweck verwendet werden können. Das hier präsentierte Framework verwendet Quadtree-Strukturen, wie in Abbildung 18 dargestellt. Eine kurze Beschreibung der Quadtree-Datenstruktur findet sich im Glossar dieser Arbeit. Quadtrees sind in der Regel vergleichsweise schnell und speichersparend, und eignen sich deshalb auch besonders für den Einsatz auf mobilen Geräten.

Für den Fall, dass ein Benutzer über sehr große Entfernungen reist, fügt das Framework die neu gefundenen Orte in einen neuen Quadtree ein, dem der entsprechende Bereich zugeordnet ist.

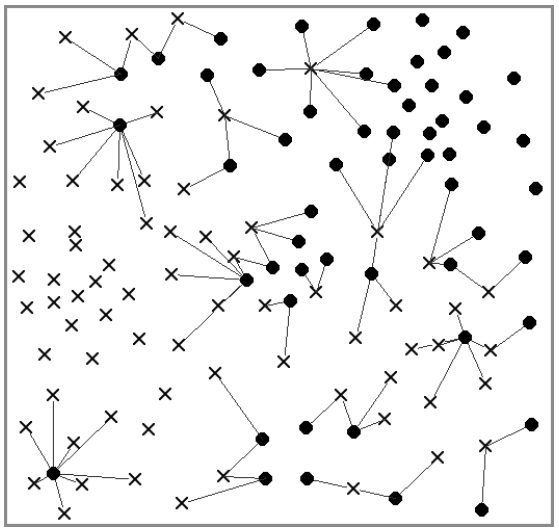


Abbildung 19: Nearest neighbor graph

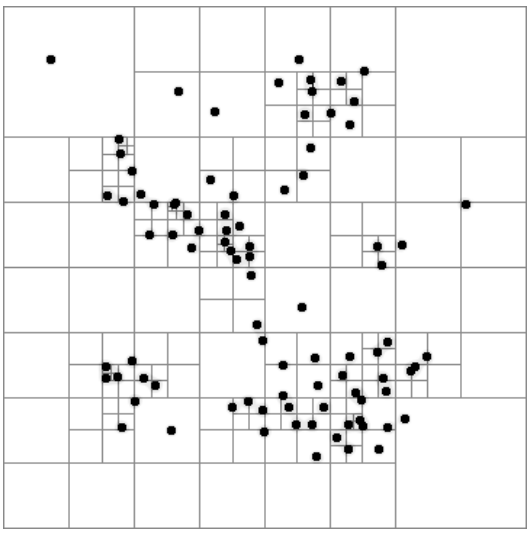


Abbildung 18: In einem Quadtree gespeicherte Positionsdaten

Diese Methode hat mehrere Vorteile: die Laufzeit der Methode kann im Allgemeinen reduziert werden, weil die Orte, die sehr weit weg von den alltäglichen Standorte sind jeweils in verschiedenen Quadtrees automatisch gruppiert werden. Darüber hinaus können zentrierte Positionen des Gebiets verwendet werden, um geeignete Datenbank-Indizes zu erstellen, die helfen können den Identifikationsprozess zu beschleunigen.

Identifikationsverfahren

Sobald die Aggregation der Daten beendet ist, können die gespeicherten Informationen verwendet werden, um einen Benutzer anhand der Orte, die er regelmäßig besucht zu identifizieren. Sobald ein bekannter Benutzer das Identifikations-Framework erneut installiert, werden seine aktuellen Standorte in einem Quadtree (oder einer ähnlichen Datenstruktur) gespeichert. Zu diesem Zeitpunkt weiß das Framework noch nicht, ob der Benutzer bereits bekannt ist oder nicht. Das Framework vergleicht den Satz der neu gesammelten Orte mit den Daten, die bereits in der Datenbank gespeichert sind. Zunächst werden die Benutzer mit grob übereinstimmenden Bereichen aus der Datenbank ausgewählt. Auf der Grundlage dieser Gruppen von Benutzerstandorten können Nächste-Nachbarn-Graphen (NNG) zwischen Standortdaten des neuen Benutzers und Standortdaten, die von bereits bekannten Benutzern erstellt worden sind, aufgebaut werden, wie in Abbildung 19 dargestellt. Um die Laufzeit des NNG-Algorithmus zu reduzieren, ist es sinnvoll, die nächsten Nachbarn nur bei einer Entfernung unter einem bestimmten Schwellenwert zu betrachten. Der durchschnittliche Abstand zwischen den nächsten Nachbarn-Standorten in Kombination mit dem Prozentsatz von Standorten, die den vorgegebenen Schwellenwert erfüllen, bietet eine angemessene und genaue Abstandsmessung, die verwendet werden kann, um Benutzer zu identifizieren.

3.1.13 Geräte-Orientierung

Auf Android-Geräten mit integrierten Beschleunigungsmesser und Magnetometer können Anwendungen aus den von diesen Sensoren [31] gewonnenen Informationen die absolute Ausrichtung des Smartphones oder Tablets berechnen.

Universalität und Messbarkeit

Beschleunigungssensoren werden wie in Kapitel 3.1.11 beschrieben seit Android 1.5 unterstützt. Das Gleiche gilt auch für das Magnetometer, welches für die Berechnung der Ausrichtung eines mobilen Geräts zusätzlich benötigt wird. Magnetometer sind wie die Beschleunigungssensoren mittlerweile Standard bei mobilen Android-Geräten, weshalb die Identifizierungsmethode auf einer sehr großen Auswahl an Geräten durchgeführt werden kann.

Android-Anwendungen können auf die Sensordaten, die von den Beschleunigungssensoren sowie dem Magnetometer geliefert werden, zugreifen. Die absolute Orientierung kann anschließend berechnet werden. Sobald ein Nutzer sein Gerät ablegt, kann eine Anwendung die Ausrichtung des Geräts berechnen und speichern, es können somit ohne größere Einschränkungen Referenzdaten für die Nutzeridentifizierung gesammelt werden. Ein weiterer Vorteil dieser Identifizierungsmethode ist, dass weder für Beschleunigungssensoren noch Magnetometer *Permissions* benötigt werden, und die Identifizierung eines Nutzers somit ohne zusätzliche Berechtigungen durchgeführt werden kann.

Einmaligkeit und Persistenz

Die gemittelten alltäglichen Orientierungen der Geräte, in welchen diese wiederholt ausgerichtet werden, stellen einen eindeutigen Fingerabdruck dar, welcher helfen kann Benutzer über einen bestimmten Zeitraum zu identifizieren. Benutzer verwenden zum Beispiel Dockingstationen, oder sie platzieren ihre Smartphones in Halterungen, wenn sie für ein paar Stunden auf der Autobahn fahren.

Einige Benutzer legen ihre Geräte jeden Tag auf Bürotische, Nachttische oder zu Hause auf dem Schreibtisch.

Nutzer gewöhnen sich oft einen festen Platz für ihr Smartphone oder Tablet an, um dieses bei Bedarf schnell zur Hand zu haben. Jede diese Orientierungen kann abgerufen und gespeichert werden und daraus nach einer gewissen Zeit einzigartige, gemittelte Profile des jeweiligen Gerätes erstellt werden, die einen Teil des alltäglichen Verhaltens des Benutzer repräsentieren.

Das in dieser Arbeit vorgestellte Framework implementiert diesen neuen Ansatz, um benutzerspezifische Profile mithilfe der Beobachtung von Geräteorientierungen zu erstellen.

Datenaggregation

Das Framework berechnet die aktuelle dreidimensionale Ausrichtung der Geräte nach bestimmten Zeiträumen. Anschließend zeichnet das Framework kontinuierlich die darauffolgenden Geräte-Ausrichtungen auf. Für den Fall, dass die Änderung der Ausrichtung unterhalb einer bestimmten Schwelle bleibt, wird der aktuelle Orientierungsmesswert gespeichert. Die Ausrichtung eines Geräts kann durch die drei Euler-Winkel beschrieben werden. Um kumulierte Werte über diese Winkel zu speichern, ist es sinnvoll, die Informationen in ein dreidimensionales Array zu speichern. Jede Dimension stellt einen der angegebenen Winkel da. Die in den entsprechenden Feldern des Arrays gespeicherten Informationen sind Zählerwerte der beobachteten Geräteorientierungen. Jedes Mal, wenn sich das Gerät in einer bestimmten Ausrichtung befindet wird der Zähler des jeweiligen Array-Elements erhöht. Nach einer gewissen Zeit enthält der dreidimensionale Würfel eine Häufigkeitsverteilung aller aufgezeichneten Orientierungen, wie in Abbildung 20 dargestellt.

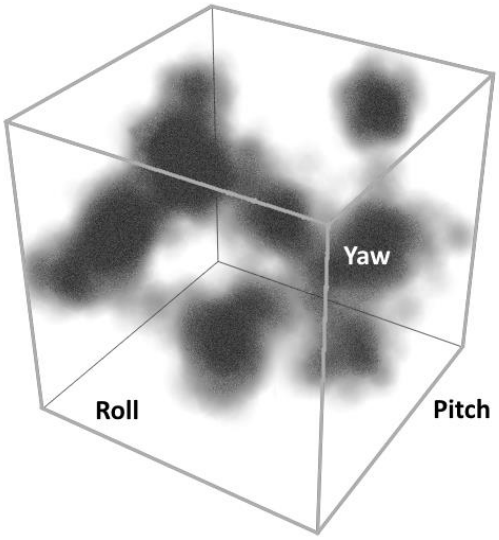


Abbildung 20: Visualisierung der Häufigkeitsverteilungen von Orientierungsdaten (3D-Array)

Identifikationsverfahren

Um Benutzer auf Grundlage der Verteilung der Geräteorientierungen zu identifizieren, müssen zunächst genügend Daten gesammelt und in dem dreidimensionalen Array gespeichert werden. Eine Bewertung darüber, wie viel sich die Verteilungen der Ausrichtungen zwischen Nutzern unterscheiden, kann im nächsten Schritt berechnet werden.

Um Störungen, verursacht durch geringfügige Abweichungen der Orientierungen, zu vermeiden wird ein Low-Pass-Filter auf des dreidimensionale Array angewendet. Danach können die im dreidimensionalen Array gespeicherten gemittelten Unterschiede der einzelnen Einträge der Häufigkeitsverteilung berechnet werden. Der Wert stellt eine geeignete Abstandsmessung dar, welche die Ähnlichkeit der kumulierten Geräteausrichtungen, beobachtet über einen bestimmten Zeitraum, beschreibt. Unsere Auswertung hat gezeigt, dass die Häufigkeitsverteilung der Orientierungen tatsächlich verwendet werden kann um einzigartige und unterscheidbare Benutzerprofile zu erstellen.

3.2 Methoden zur Identifizierung von Geräten

3.2.1 Hardware-Spezifikation

Auf Android-Geräten ist es für Anwendungen möglich, die Modellnummer sowie den Hersteller des Gerätes herauszufinden, was als Kennzeichen für das verwendete Hardware-Setup verwendet werden kann. Jedoch können Geräteversionen variieren, wenn es um spezifische Kennzahlen wie dem verfügbaren Speicherpatz, der CPU, verbauter Netzwerktechnologien und anderen Hardware-Einheiten geht. In den meisten Fällen verkaufen die Hersteller die sich unterscheidenden Versionen in verschiedenen Ländern oder Kontinenten. Es macht Sinn, diese spezifischen Hardware-Informationen zusammen mit Daten über das Modell und dem Hersteller zu speichern. So können Geräte möglichst fein geclustert werden. Das Clustering auf Basis von Hardware-Spezifikationen ist zuverlässig und basiert nicht auf Wahrscheinlichkeitsberechnungen. Es ist deshalb nützlich um falsche Klassifikationen auszusortieren.

Universalität und Erfassbarkeit

Informationen über die Hardware lassen sich auf jedem handelsüblichen Android-Gerät auslesen. Die Informationen können ohne *Permission* von Android-Anwendungen abgefragt werden. Da die Informationen sofort nach dem Start einer Anwendung erfasst werden können, eignet sich diese Methode hervorragend, um schon frühzeitig kleinere Cluster von Geräten, die mithilfe anderer Methoden verglichen werden müssen, zu erstellen.

Einmaligkeit und Persistenz

Da von einem Modell eines mobilen Geräts in der Regel eine hohe Stückzahl produziert wird, kann die Kombination der verbauten Hardware nicht als einmalig betrachtet werden. Allerdings können Informationen über die Hardware eines Geräts verwendet werden, um die zu identifizierenden Geräte in kleinere Clustern einzuteilen. Die Unterscheidung in Gerätegruppen mit gleicher Hardware kann dabei helfen, die Ergebnisse der anderen Methoden zu verbessern, indem die Falsch-Positiv- sowie Falsch-Negativ-Rate gesenkt werden. Das Gesamtergebnis einer Geräteidentifizierung kann dadurch weiter optimiert werden. Falls die Anzahl an Geräten, die identifiziert werden sollen, sehr groß ist, wird es zudem deutlich schwieriger, möglichst einzigartige, unterscheidbare Merkmale für die Identifizierung zu finden. Jedes Kriterium, welches die Menge der zu identifizierenden Geräte in kleinere Cluster unterteilt, hilft, die Genauigkeit des gesamten Identifizierungsprozesses zu erhöhen.

Die Hardwarekonfiguration eines mobilen Gerätes ändert sich nur in den seltensten Fällen. Selbst falls ein Gerät repariert werden muss, werden in der Regel baugleiche Ersatzteile eingesetzt, wodurch sich die ursprüngliche Hardwarekonfiguration nicht ändert.

In Zukunft könnten allerdings durch neue Entwicklungen auch Android-Geräte auf den Markt kommen, welche den Austausch von Hardware-Komponenten wesentlich erleichtern. Das bringt für die Identifizierung von einzelnen Geräten Vor- und Nachteile mit sich. Zum einen steigt die Individualität der jeweiligen Hardware-Konfigurationen, wodurch sich die Menge der zu identifizierenden Geräte in noch viel kleinere Cluster unterteilen lässt. Dadurch steigen die Performanz und die Präzision des gesamten Identifizierungsvorgangs, da weniger potentielle Kandidaten miteinander verglichen werden müssen. Allerdings kann auch davon ausgegangen werden, dass die Benutzer einzelne Hardwarekomponenten regelmäßig tauschen werden, wodurch die Persistenz der genutzten Informationen eingeschränkt wird. Das Framework, das in dieser Arbeit vorgestellt wird, wurde bereits so konzipiert, dass mit wenig Änderungsaufwand auch Geräte mit sich ändernden Hardwarekonfigurationen erfasst werden können. Im Moment wird allerdings noch davon ausgegangen, dass die jeweilige Hardwarekonfiguration unverändert bleibt.

3.2.2 Bluetooth und WLAN

Falls Bluetooth oder WLAN auf einem Android-Gerät verfügbar ist, können die entsprechende MAC-Adressen und der Anzeigenamen bzw. die SSID des spezifischen Adapters von einer Android-Anwendung ausgelesen werden [40]. Die Identifikationsnummer, die das Gerät identifiziert, kann dann mit diesen Informationen verknüpft werden.

Universalität und Erfassbarkeit

Wie bereits in den Kapitel 3.1.3 und 3.1.4 beschrieben, hat sich Bluetooth und WLAN bei handelsüblichen Android-Geräten mittlerweile als Standard durchgesetzt. Die Identifizierungsmethoden können somit auf fast allen Smartphones und Tablets angewendet werden.

Für die Identifizierung anhand der Bluetooth-MAC-Adresse muss Bluetooth auf dem Gerät aktiviert sein, um die erforderlichen Daten zu sammeln. Falls Bluetooth nicht aktiviert sein sollte, wird eine zusätzliche Android-Erlaubnis benötigt, um die MAC-Adresse des Bluetooth-Adapters ermitteln zu können.

Um Daten über den WLAN-Adapter sammeln zu können werden weitere *Permissions* benötigt:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

Einmaligkeit und Persistenz

Die MAC-Adresse eines Bluetooth-Adapters in Kombination mit dem Anzeigenamen kann zumindest in der Theorie als einmalig angesehen werden. Allerdings gibt es wie bereits in Kapitel 3.1.3 geschildert Ausnahmen. Auch die auf den meisten Geräten verfügbaren WLAN-Adapter besitzen zumindest in der Theorie eindeutige MAC-Adressen, welche, analog zu der Vorgehensweise mit Bluetooth-Adaptern, von einer Anwendung gespeichert und analysiert werden können, um das Gerät zu identifizieren. Diese MAC-Adressen können sich allerdings ändern, oder sind nicht wirklich eindeutig. Einige Hersteller produzieren Geräte mit gleicher MAC-Adresse, oder sogar Geräte auf denen nach jedem Neustart neue zufällige MAC-Adressen generiert werden.

Das Framework ist in diesen Fällen in der Lage, sich wiederholende MAC-Adressen zu erkennen, und auf andere Identifizierungsmethoden auszuweichen. Falls die MAC-Adresse des Bluetooth-Adapters den technischen Vorgaben entspricht und tatsächlich als einmalig betrachtet werden kann, stellt die Identifizierungsmethode allerdings eine hervorragende Möglichkeit dar, Geräte eindeutig zu identifizieren.

MAC-Adressen sollten sich per Definition nicht ändern und somit über die gesamte Lebensdauer eines mobilen Geräts konstant bleiben. Allerdings gibt es auch hier, ähnlich zu den Problemen mit der Einzigartigkeit der MAC-Adressen, Abweichungen von den Vorgaben. Die Anzahl der fehlerhaften Implementierungen halten sich aber in Grenzen, wodurch die Identifizierungsmethode in der Regel auf eine konstante Informationsquelle zurückgreifen kann.

3.2.3 International Mobile Equipment Identity (IMEI)

Die International Mobile Equipment Identity (IMEI) ist eine 15-stellige Zahl, die verwendet wird, um GMTS- oder UMTS-Geräte eindeutig identifizieren zu können [54].

Universalität und Erfassbarkeit

Jedes mobile Gerät, welches einen SIM-Karten-Slot besitzt, kann unter Verwendung der entsprechenden IMEI-Nummer des Gerätes identifiziert werden (Listing 6) [55]. Android-Anwendungen benötigen folgende *Permission*, um auf die Daten zugreifen zu können:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE " />
```

Alle Smartphones sollten dem allgemeinen Verständnis nach mit einem SIM-Kartenslot ausgestattet sein, weshalb das Verfahren vor allem für diese Geräteklasse besonders gut geeignet ist. Tablets und andere Android-Geräte, die womöglich über keinen SIM-Karten-Slot verfügen, sind für diese Identifizierungsmethode hingegen nicht gut geeignet. Es sollte in diesen Fall immer eine alternative Identifizierungsnummer existieren, wie in Kapitel 3.2.4 beschrieben.

Einmaligkeit und Persistenz

Auch wenn der Identifikationsprozess auf Grundlage der IMEI-Nummern ein sehr zuverlässiges Verfahren zu sein scheint, gibt es einige Nachteile: Die IMEI-Nummer kann leicht von den Benutzern geändert werden, wodurch sowohl die Persistenz als auch die Einmaligkeit der IMEI-Nummer nicht mehr gewährleistet werden kann. Zudem können die Geräte mehr als eine IMEI-Nummer besitzen, je nachdem wie viele SIM Kartensteckplätze vorhanden sind. Es muss deshalb immer eine Menge an IMEI-Nummern einen Benutzer zugeordnet werden. Darüber hinaus wird zwar theoretisch die Einzigartigkeit der IMEI-Nummer gewährleistet, aber in der Realität kann immer noch eine Anzahl von Geräten mit gleichen IMEI-Nummern gefunden werden. Dies ist auf falsche Implementierungen und der Tatsache, dass Benutzer in der Lage sind die IMEI-Nummer auf dem Gerät zu ändern, zurückzuführen.

Listing 6: Quellcode um die IMEI eines Geräts abzufragen

```
01 private static String getIMEI(Context context){
02     if(context!=null){
03           TelephonyManager phoneManager = (TelephonyManager) context
04                       .getSystemService(Context.TELEPHONY_SERVICE);
05           if(phoneManager!=null){
06                 return phoneManager.getDeviceId();
07           }
08     }
09     return "";
10 }
```

3.2.4 Seriennummer

Falls ein Gerät nicht in der Lage ist, eine einzigartige IMEI-Nummer bereitzustellen, wird von den Geräteherstellern gefordert, dass das Gerät stattdessen eine einzigartige Geräte-ID bzw. Seriennummer besitzt und bereitstellt. Dies ist beispielsweise bei Tablets oder Media-Playern ohne SIM-Karten-Slot der Fall.

Universalität und Erfassbarkeit

Die Seriennummer kann ergänzend zur IMEI-Nummer für die Identifizierung von Geräten benutzt werden. In Kombination decken die beiden Identifizierungsmethoden einen sehr großen Teil der Android-Geräte ab. In der Realität ist die Anzahl der Geräte mit Seriennummer immer noch relativ gering, weshalb stattdessen in diesen Fällen auf die IMEI, wie in Kapitel 3.2.3 beschrieben, zurückgegriffen werden muss.

Die Seriennummer kann von einer Android Anwendung abgefragt werden, wobei Android hierfür keine speziellen Methoden zur Verfügung stellt. Da die Seriennummer nur auf einem Teil der Android-Geräte zur Verfügung steht, muss bei der Abfrage der Daten darauf geachtet werden, dass die Identifizierungsmethode keine Fehlermeldungen verursacht (Listing 7).

Einmaligkeit und Persistenz

Wie die meisten Gerätekennungen ist die Seriennummer eines Android-Geräts nicht immer eindeutig oder persistent. Grund dafür ist, dass die Seriennummer von Benutzern vorübergehend geändert werden kann, falls Root-Rechte auf dem Gerät aktiviert sind. Auf handelsüblichen Android-Geräten kann allerdings davon ausgegangen werden, dass die Identifizierungsmethode über einen längeren Zeitraum Geräte eindeutig identifizieren kann, falls eine Seriennummer vorhanden ist.

Listing 7: Quellcode um die Seriennummer eines Geräts auszulesen

```
01 private static String getSerialNumber(){
02     String serialNumber = "";
03     try {
04           Class<?> c = Class.forName("android.os.SystemProperties");
05           Method g = c.getMethod("get", String.class, String.class);
06           serialNumber = (String) (g.invoke(c, "ro.serialno", "unknown"));
07     } catch (Exception e) {
08           Log.v(TAG,"Exception",e);
09     }
10     return serialNumber;
11 }
```

3.2.5 System-Pakete und -Dateien

Jedes Android-Gerät wird mit einer bestimmten Auswahl vorinstallierter System-Pakete (Java Packages) ausgeliefert. Zusätzlich werden auf einem Smartphone oder Tablet herstellerspezifische Systemdateien vorab gespeichert. Informationen über die vorinstallierte Software-Zusammenstellung sowie System-Dateien können von dem Framework extrahiert werden, um den jeweiligen Geräten spezifische Eigenschaften zuzuordnen und diese dadurch unterscheidbar zu machen.

Universalität und Erfassbarkeit

Informationen über vorinstallierte System-Packages und -Dateien können auf jedem handelsüblichen Android-Gerät extrahiert werden. Neben den Packages und Dateien, die standardmäßig vorab mit Android auf Smartphones und Tablets gespeichert werden, existieren eine ganze Reihe herstellerspezifischer Daten, welche noch vor der Auslieferung eines Geräts auf diesem gespeichert werden.
Die Hersteller von Smartphones und Tablets konfigurieren ihre mobilen Geräte so, dass die Packages bereits vor der ersten Verwendung des Smartphones oder Tablets automatisch installiert werden. Die benötigten Informationen sind somit von Anfang an auf jedem Android-Smartphone vorhanden. Android-Anwendungen können eine Liste aller installierten System-Packages extrahieren. Es wird ein Zeitstempel der Installation und der letzten Aktualisierung eines System-Packages gespeichert, sowie die Bezeichnung des *Namespace*, der durch das jeweiligen Package repräsentiert wird. Zudem ist es möglich, Informationen über bestimme Systemdateien anzufordern. Auch hier bietet sich die Speicherung eines Zeitstempels an. In Kombination dazu wird wie in Kapitel 3.1.8 beschrieben ein zusätzlicher Hash-Werte der Systemdateien erstellt. Anschließend besteht die Möglichkeit, eine gerätespezifische Identifikationsnummer der Zusammenstellung aller vorinstallierten Systempakete bzw. Systemdateien zuzuordnen.

Um auf die vorinstallierten Packages zuzugreifen wird folgende *Permission* benötigt:

```
<uses-permission android:name="android.permission.GET_TASKS" />
```

Die Analyse der Systemdateien erfordert folgende *Permission*:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Einmaligkeit und Persistenz

Die Informationen über die Systempakete bzw. Dateien als Ganzes unterscheiden sich von Gerät zu Gerät, und können somit dabei helfen, unterschiedliche Smartphones und Tablets zu identifizieren. Die Kombination von Hash-Werten, Package-Typen und den jeweiligen Zeitstempeln stellen voneinander unterscheidbaren Informationen dar. Vor allem durch die Verwendung von Zeitstempel wird eine gewisse Einmaligkeit gewährleistet. Die Kombination der installierten System-Packages und -Dateien an sich stellt zudem auch ohne Zeitstempel schon eine sehr gute Möglichkeit dar, die Geräte zumindest in sehr kleine Cluster zu unterteilen, was den Rechenaufwand der anderen Identifizierungsmethoden deutlich reduziert.

Die Informationen, auf denen diese Identifizierungsmethoden basieren, ändern sich in der Regel selten. Es können allerdings Situationen auftreten, in denen zumindest ein Teil der verwendeten Informationen nicht mehr aktuell ist, beispielsweise wenn ein System-Update durchgeführt wird, oder das mobile Gerät auf die Werkseinstellungen zurückgesetzt wird. Vor allem die gespeicherten Zeitstempel können sich in diesen Fällen ändern, wobei die individuelle Zusammenstellung der herstellerspezifischen System-Packages und Dateien zumindest im Kern teilweise erhalten bleiben sollte. Das Framework sollte mithilfe anderer Identifikationsmethoden erkennen, wenn die Zuordnung der Informationen zu einem bestimmten Gerät nicht mehr möglich ist, und die betroffenen Referenzdaten auf den aktuellen Stand bringen.

3.2.6 Pixelfehler

Pixelfehler sind auf vielen Displays oder Bildern, die mithilfe der verbauten digitalen Kamerasensoren angefertigt wurden, zu beobachten. Die Farbe dieser Pixel unterscheidet sich im Allgemeinen drastisch von den Umgebenden, und ist auf allen Bildern, die mit dem entsprechenden Kameramodul gemacht wurde, zu erkennen. [56]

Universalität und Erfassbarkeit

Auf jedem Android-Gerät mit mindestens einer Kamera, kann die spezifische Verteilung der Pixelfehler ausgewertet werden. Android-Anwendungen können Bilder direkt von der Kamera abfragen, ohne dass der Nutzer aktiv eingreifen muss. Zudem können Bilder, die sich in der Galerie des Smartphones oder des Tablets befinden, auf Pixelfehler untersucht werden. Bis auf einige Nischenprodukte [57] sind mittlerweile so gut wie alle Android-Geräte mit mindestens einer Kamera ausgestattet. So ist die Identifizierungsmethode selbst auf Modellen aus den preisgünstigeren Segmenten anwendbar.

Obwohl Pixelfehler bekannt sind und den meisten Nutzern inzwischen ein Begriff sein sollten, ist praktisch nur ein Teil der Android-Geräte davon betroffen. Das Auftreten von Pixelfehlern und deren Verteilung unterliegen zufälligen Größen. Falls durch die Identifizierungsmethode keine Pixelfehler gefunden werden können, wird diese Information über deren Fehlen anstelle der eigentlichen Daten über Pixelfehler von der Identifizierungsmethode verwendet, um die Menge aller Geräte zu clustern.

Android selbst ist so konzipiert, dass Anwendungen die Bilddaten der Kamera im *Joint Photographic Experts Group* (JPEG) oder einem anderen verlustbehafteten Bildformat übermittelt bekommen. Es besteht zwar in der Theorie die Möglichkeit, auch die Rohdaten eines Bildes anzufordern, allerdings wird dies auf einem Großteil der Android-Geräte nicht unterstützt. Das liegt zum einen an dem begrenzten Arbeitsspeicher, der Android-Anwendungen in der Regel zur Verfügung steht. Zum anderen liegt es daran, dass für mobile Anwendungen Bilder in guter JPEG-Qualität in den meisten Fällen völlig ausreichend sind, und deshalb kein großer Bedarf nach den Rohdaten eines Bildes von Seiten der Anwender besteht. Da das Identifikations-Framework jedoch auch einzelne Pixel analysieren muss, um brauchbare

Informationen aus den Bildern zu extrahieren, müssen einige Besonderheiten beachtet werden. Die fehlerhaften Pixel erscheinen in komprimierten Bildformaten etwas verwaschen. Die Farbe des fehlerhaften Pixels kann sich in diesen Fällen auch auf die Farben der benachbarten Pixel auswirken, und umgekehrt. Bei der Erkennung der Pixelfehler muss diese Tatsache beachtet werden und die Algorithmen entsprechend angepasst werden.

Während der Evaluation des Frameworks har sich gezeigt, dass auf handelsüblichen Android-Geräten in der Regel nur ein, maximal zwei unkomprimierte Bilder in voller Auflösung in den verfügbaren Arbeitsspeicher passen. Falls durch eine Anwendung mehr Bilder geladen werden, wird eine Fehlermeldung ausgegeben. Bei den Implementierungen der Methoden, die auf hochauflösende Bilddaten zugreifen, musste diese Einschränkung beachtet werden. Die Untersuchung eines Bildes auf Pixelfehler wurde deshalb so umgesetzt werden, dass nur die Pixel-Daten eines einzelnen Fotos während der Analyse benötigt werden. Es werden also insbesondere keine weiteren Kopien des Bildes verwendet. Persistent gespeichert werden muss bei dieser Methode nur eine Liste der gefunden Pixelfehler. In Kapitel 3.2.7, bei der Untersuchung von Dark-Frames, zeigt sich eine ähnliche Problematik.

Um auf die Kamera eines Android-Geräts zugreifen zu können, wird folgende *Permission* benötigt:

```
<uses-permission android:name="android.permission.CAMERA" />
```

Einmaligkeit und Persistenz

Die Verteilung der fehlerhaften Pixel kann verwendet werden, um eine Identifikationsnummer für ein bestimmtes Gerät mit dieser Information zu verknüpfen. Die Einmaligkeit dieser Verteilung steigt mit der Anzahl der gefundenen Pixelfehler, wobei schon bei einem einzigen Pixelfehler durch die hohe Pixelanzahl eines Bildes von einer sehr großen Einzigartigkeit ausgegangen werden kann.

Falls für Kamerasensor festgestellt werden konnte, das Bilder mit Pixelfehler zurückliefert werden, kann davon ausgegangen werden, dass die fehlerhaften Pixel auch weiterhin auf Bildern auftauchen werden. Da Pixelfehler durch persistente, fehlerhafte Hardware-Eigenschaften verursacht werden, verschwindet die Ursache nur in den seltensten Fällen. Es besteht allerdings die Möglichkeit, dass durch ein Firmware-Update oder eine Anpassung der Kamerasoftware fehlerhafte Pixel aus den zurückgelieferten Bildern herausgerechnet werden, bevor die Identifizierungsmethode Zugriff auf die Bilddaten hat. Zudem kann sich die Anzahl der gefundenen Pixelfehler mit der Zeit erhöhen. Das Identifikations-Framework muss deshalb so konzipiert sein, dass Veränderungen in der Anzahl der Pixel die Falsch-Negativ-Rate des Verfahrens nicht erhöhen. In diese Fälle kommen die Ergebnisse der anderen Identifikationsmethoden zum Einsatz.

Datenaggregation

Bilder aus der Galerie oder Bilddaten, die direkt von der integrierten Kamera stammen, können analysiert werden, um fehlerhafte Pixel zu finden, die für diese Kamera charakteristisch sind. Bilder, die im Rahmen der Dark-Frame-Analyse angefertigt wurden, eignen sich besonders gut für die Pixelfehler-Erkennung, da in dunklen Bildern die RGG-Werte fehlerhafter Pixel in der Regel besonders deutlich von den Farbwerten der umliegenden Pixel abweichen. Die Position der Pixelfehler kann erfasst werden, indem die Farbverteilung sowie die Farbabweichung einzelner Pixel analysiert wird. Dazu werden die Farbwerte umliegender Pixel betrachtet und Abweichungen davon analysiert. Eine Methode, durch die mögliche Pixelfehler anhand der umliegenden Pixel erkannt werden können, wird in Listing 8 aufgeführt.

Die Informationen über die fehlerhaften Pixel können gespeichert und mit den bereits gespeicherten Pixel verglichen werden. Jedes Mal, wenn die Koordinaten eines Pixels mit denen eines bereits gespeicherten Pixels übereinstimmen, wird ein dazugehöriger Zähler hochgezählt. Die Zählvariable kann verwendet

werden, um nach einer bestimmten Zeit die Anzahl fälschlicherweise gespeicherten Pixel zu reduzieren. Durch dieses Verfahren wird die Falsch-Negativ- und Falsch-Positiv-Rate der Pixelfehler-Erkennung mit zunehmender Anzahl der zur Verfügung stehenden Bilder deutlich reduziert.

Listing 8: Algorithmus, um mögliche Pixelfehler zu erkennen

```
01 private boolean  isDeadPixel(Bitmap bmp,int x,int y){
02     int rgb       = bmp.getPixel(x, y);
03     int red       = Color.red(rgb);
04     int green     = Color.green(rgb);
05     int blue      = Color.blue(rgb);
06     int num       = 0;
07     double aRed   = 0;
08     double aBlue  = 0;
09     double aGreen = 0;
10     for (int px = x-pixelErrorArea; px <= x+pixelErrorArea; px++) {
11           for (int py = y-pixelErrorArea; py <= y+pixelErrorArea; py++) {
12                 if(px!=x||py!=y){
13                       if(px >= 0 && px < bmp.getWidth()){
14                             if(py >= 0 && py < bmp.getHeight()){
15                                   int pRGB = bmp.getPixel(px, py);
16                                   aRed = aRed+Color.red(pRGB);
17                                   aBlue = aBlue+Color.blue(pRGB);
18                                   aGreen = aGreen+ Color.green(pRGB);
19                                   num++;
20                             }
21                       }
22                 }
23           }
24     }
25     if(num>0){
26           aRed = aRed/num;
27           aBlue = aBlue/num;
28           aGreen = aGreen/num;
29           if (Math.abs(red-aRed) >  pixelErrorThreshold
30                       || Math.abs(green-aGreen) >  pixelErrorThreshold
31                       || Math.abs(blue-aBlue)  > pixelErrorThreshold) {
32                       return true;
33           }
34     }
35     return false;
36 }
```

Identifikationsverfahren

Sobald die Daten über fehlerhafte Pixel gesammelt sind, kann dieser Satz von Pixeln mit den bereits in der Cloud-Datenbank gespeicherten Datensätzen verglichen werden. Für den Fall, dass zwei Gruppen übereinstimmen, kann das Gerät mit einer sehr hohen Sicherheit identifiziert werden. Damit ein Gerät als identifiziert eingestuft werden kann, sollte die Anzahl der Pixel paarweise übereinstimmen.

3.2.7 Dark-Frames

Dark-Frames werden in der digitalen Fotografie zur Rauschreduzierung bei Aufnahmen mit langer Belichtungszeit verwendet. Dark-Frames sind Bilder, welche in kompletter Dunkelheit und in der Regel mit geschlossenem Objektiv, aufgenommen wurden [58]. Abbildung 21 zeigt das Bildrauschen eines Dark-Frames, welches auf einem mobilen Gerät erstellt wurde. Wie in der Evaluation festgestellt, ist oft mit bloßem Auge ist zu erkennen, dass sogar bauchgleiche Kameras jeweils individuelle Muster erzeugen.

Einige Eigenschaften des aufgezeichneten Bildrauschens bleiben über die Zeit unverändert, wobei dieses

Phänomen *Fixed Pattern Noise* genannt wird. Mithilfe von Dark-Frames können Fixed Pattern Noise - Informationen extrahiert und analysiert werden. Die gesammelten analysierten Daten von Dark-Frames können mit einer entsprechenden Geräte-ID verknüpft werden.

Universalität und Erfassbarkeit

Wie bereits in Kapitel 3.2.6 beschrieben sind Kameras in Android-Geräten sehr weit verbreitet. Es wird, analog zu der Analyse von Pixelfehlern, eine *Permission* benötigt, um auf die internen Kameras zugreifen zu können. Fixed Pattern Noise - Informationen sind bedingt durch die spezifischen Hardwareeigenschaften der Bildsensoren für jede Kamera zu finden. Das Identifikationsverfahren lässt sich somit auf jedem Smartphone oder Tablet anwenden, welches über mindestens eine Kamera verfügt.

Einmaligkeit und Persistenz

Fixed Pattern Noise - Informationen zeichnen sich durch ihre sehr spezifischen Eigenschaften aus. Es ist zu beobachten, dass selbst auf baugleichen Kameras ein zum Teil sehr unterschiedliches Bildrauschen extrahiert werden kann. Auch wenn sich die Umwelteinflüsse ändern, oder die komplette Software eines Gerät geändert wird, bleiben die probabilistischen Attribute der Muster in den beobachteten Dark-Frames nahezu gleich. Sie eignen sich daher sehr gut, um registrierte Geräte zu identifizieren. Die verwendeten Informationen zeigten auch im Verlaufe der Evaluation durchgehend eine sehr hohe Persistenz.

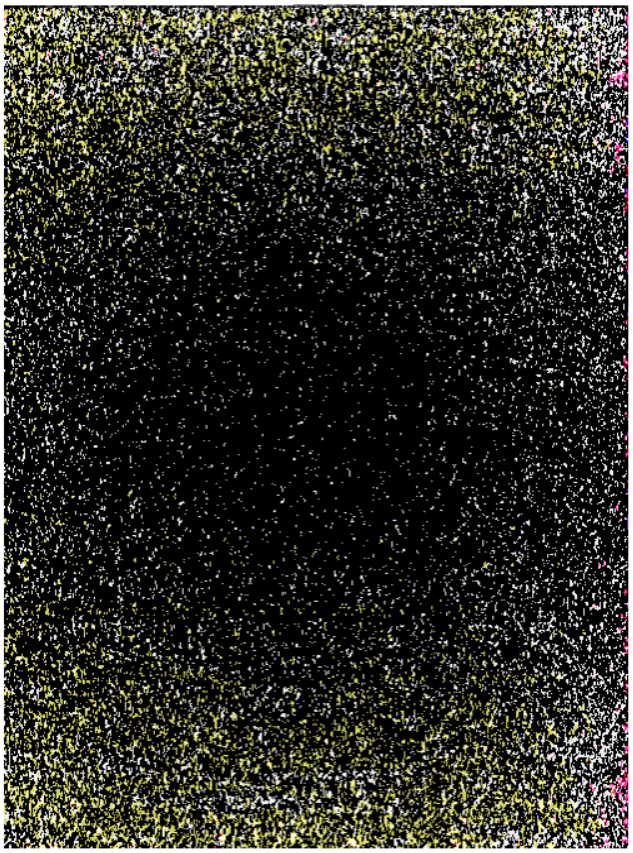


Abbildung 21: Kameraspezifisches Bildrauschen

Datenaggregation

Berechtigte Android-Anwendungen können auf die internen Kameras des Gerätes zugreifen. In den meisten Fällen befindet sich eine Kamera auf der Rückseite und eine auf der Vorderseite des Geräts. Basierend auf den Werten des Lichtsensor nimmt das Framework Bilder in der Dunkelheit auf, zum Beispiel in der Nacht oder wenn der Benutzer sein Gerät in der Tasche trägt. Die durchschnittliche Farbe des Bildes kann anschließend berechnet werden. Für die Verwendung als Dark-Frame sollte das Bild eine möglichst geringe Helligkeit aufweisen. Wenn die durchschnittliche Farbe einen bestimmten Schwellenwert einhält, wird das Bild auf dem Handy gespeichert. Nach einer gewissen Zeit, wenn mehrere geeignete Bilder aufgenommen worden sind, wird die Durchschnittsfarbe der einzelnen Pixel auf der Grundlage der aufgenommenen Bilder berechnet, wie in Listing 9 dargestellt. Die aggregierten Daten werden normalisiert, und in einer Dark-Frame-Datei gespeichert. Der Dark-Frame wird in einer Datenbank abgelegt und kann durch eine Identifizierungsnummer mit dem Gerät verknüpft werden.

Listing 9: Methode für das Zusammenfügen mehrerer Bilder (Dark-Frame-Analyse)

```
01 private void addDarkFrame(Bitmap bmp){
02     Canvas canvas = new Canvas(darkFrame.getImage());
03     Paint paint = new Paint();
04     int alpha = (int) ((1.0 / (darkFrame.getWeight() + 1.0)) * 255.0);
05     if (alpha < 2) {
06           alpha = 2;
07     }
08     paint.setAlpha(alpha);
09     canvas.drawBitmap(bmp, 0, 0, paint);
10     darkFrame.increaseWeight();
11 }
```

Identifikationsprozess

Nachdem ein Dark-Frame auf einem bestimmten Gerät erstellt wurde, können dessen Daten mit bereits in der Datenbank gespeicherten Dark-Frames verglichen werden. Die Ähnlichkeit kann berechnet werden, indem die gemittelten Unterschiede zwischen den Standardabweichungen und die Farbwerte der einzelnen Pixel verglichen werden. Der Vergleich der Histogramme der einzelnen Dark-Frames ergibt auch Abstandswerte, die die Ähnlichkeit der Dark-Frames repräsentieren. Für den Fall, dass die Farbe-Distributionen, die Standardabweichungen, die Farbwerte selbst und die Histogramme zu einem gewissen Grad übereinstimmen, kann das Gerät mit einer definierten Wahrscheinlichkeit identifiziert werden.

3.3 Methoden zur kombinierten Identifizierung von Geräten und Benutzern

Neben den Methoden, die jeweils ausschließlich der Identifizierung von Nutzern oder der Identifizierung von Geräten dienen, gibt es eine dritte Kategorie an Identifizierungsmethoden. Einige Informationen, die für die Identifizierung verwendet werden können, basieren sowohl auf Daten, die von einem bestimmten Nutzer abhängen, als auch Informationen, die einem bestimmten Gerät direkt zugeordnet werden können.

Die Aussagekraft der Informationen wird dadurch erhöht. Wenn man die Eigenschaft der Einmaligkeit der Referenzdaten betrachtet, lässt sich sagen, dass durch die spezifische Kombination von Nutzer und Gerät sehr gute Ergebnisse erzielt werden können. Die verwendeten Informationen sind sehr spezifisch, und lassen sich im besten Fall nur exakt einer Nutzer-Geräte-Kombination zuordnen. Allerdings wird die Persistenz der Referenzdaten durch die Voraussetzung, dass eine bestimmte Kombination aus Nutzer und Gerät vorliegen muss, eingeschränkt. Sobald ein Nutzer das Gerät wechselt, ist keine der in den folgenden Kapiteln vorgestellten Methoden in der Lage, Nutzer oder Gerät separat wiederzuerkennen. Das Framework muss in diesen Fällen auf Methoden zurückgreifen, die in den Kapiteln 3.1 und 3.2 vorgestellt wurden.

3.3.1 Advertising ID

Auf Android-Geräten gibt es mehrere Identifikationsnummern, die von dem System selbst bereitgestellt werden. Die *Google Advertising-ID* kann von Entwicklern verwendet werden, um ihre Android-Anwendungen zu monetarisieren.

Universalität und Erfassbarkeit

Advertising-IDs werden vom Google-Play-Dienst ab Version 4.0+ unterstützt, und können von Android-Anwendungen jederzeit ausgelesen und für Identifizierungszwecke verwendet werden (Listing 10). So können beispielsweise Werbenetzwerk-Kunden auf diese String-Bezeichner zurückgreifen, um Nutzer zu identifizieren. Die Advertising-ID stellt somit eine sehr gute Möglichkeit dar, das Identifizierungs-

Framework um eine weitere Informationsquelle zu ergänzen.

Persistenz und Einmaligkeit

Google versichert, dass jeder Android-Nutzer seine eigene, einzigartige Advertising-ID erhält. Allerdings können Nutzer diese ID zu jedem Zeitpunkt leicht zurücksetzen, womit die Dauerhaftigkeit und Einmaligkeit dieses *Universally Unique Identifiers* (UUID) nicht immer garantiert werden kann. Das Framework ist somit auch auf die Ergebnisse der anderen Identifizierungsmethoden angewiesen, selbst wenn die Advertising ID auf einem Gerät zur Verfügung steht.

Listing 10: Quellcode um die GSF ID abzufragen

```
01 private static String getGsfAndroidId(Context context) {
02    if (context != null) {
03          String ID_KEY = "android_id";
04          String params[] = { ID_KEY };
05          Cursor c = context.getContentResolver().query(GSFURI,
06          null, null,params, null);
07          if (c != null) {
08                if (!c.moveToFirst() || c.getColumnCount() < 2) {
09                      return "";
10                }
11                try {
12                      return HashGenerator.hash(
13                      Long.toHexString(Long.parseLong(c.getString(1))));
14                } catch (NumberFormatException e) {
15                      return "";
16                }
17          }
18    }
19    return "";
20 }
```

3.3.2 ANDROID_ID

Jedes Mal, wenn ein Benutzer sein Android-Gerät einrichtet, wird automatisch eine ANDROID_ID erzeugt. Die ANDROID_ID wird durch einen zufällig generierten 64-Bit Wert repräsentiert. Diese kann zu Identifizierung eines Gerätes und des aktuellen Benutzers verwendet werden.

Universalität und Erfassbarkeit

Die ANDROID_ID wird ab API Level 3 auf Android-Geräten erzeugt, und ist somit auf handelsüblichen Android-Geräten immer vorhanden. Anwendungen können die ANDROID_ID jederzeit auslesen, ohne dass eine zusätzliche *Permission* benötigt wird (Listing 11).

Einmaligkeit und Persistenz

Aufgrund von Implementierungsfehler größerer Hersteller besitzen einige Geräte dieselbe ANDROID_ID. Es kann somit nicht immer von einer absoluten Einmaligkeit der ANDROID_ID ausgegangen werden. Wie auch bei der Analyse von MAC-Adressen ist das Framework in der Lage, mehrmals vorkommende ANDROID_IDs nach einiger Zeit zu erkennen und zu ignorieren. Ein weiteres Problem, welches die Persistenz der ANDROID_ID betrifft, ist, dass sich die erzeugte Identifikationsnummer ändert, sobald ein Gerät auf Werkseinstellungen zurückgesetzt wird. Auch in diesem Fall muss das Framework auf andere Identifikationsmethoden zurückgreifen, um fehlerhafte Ergebnisse zu vermeiden. Ab Android 4.2 erhält jeder Benutzer seine eigene ANDROID_ID, so dass einem bestimmten Gerät mehr als eine Identifikationsnummern zugeordnet sein kann. Diese Besonderheit der Zuordnung muss bei der Implementierung eines Identifikationsverfahrens beachtet werden.

Listing 11: Quellcode um die ANDROID_ID eines Geräts abzufragen

```
01 import android.provider.Settings.Secure;
02
03 private static String getAndroidID(Context context){
04    if(context!=null){
05          String android_id = Secure.getString(
06          context.getContentResolver(),Secure.ANDROID_ID);
07          if(isAndroidIDSuitable(android_id)){
08                return android_id;
09          }
10    }
11    return "";
12 }
```

3.3.3 Google Service Framework ID

Die Google-Service Framework (GSF) bietet eine einzigartige 64-Bit-Identifikationsnummer, die von Android-Anwendungen verwendet werden kann, um bestimmte Benutzer und Geräte zu identifizieren.

Universalität und Erfassbarkeit

Die GSF-ID wird nach dem Zufallsprinzip beim ersten GSF-Login erzeugt und liefert eine zuverlässige Möglichkeit, Benutzer und Android-Geräten zu unterscheiden. Die ID ist auf jedem Android-Gerät verfügbar, welches das Google-Service Framework installiert hat. Da viele sehr häufig genutzte Anwendungen auf das Google-Service Framework angewiesen sind, kann davon ausgegangen sein, dass auf handelsüblichen Geräten diese ID in der Regel zur Verfügung steht. Um die GSF-ID auszulesen, wird keine zusätzliche *Permission* benötigt.

Einmaligkeit und Persistenz

Die GSF-ID kann sich jedoch ändern, beispielsweise falls das Gerät zwischenzeitlich von einem Nutzer oder während einer Reparatur auf Werkseinstellungen zurückgesetzt wird, wie es auch bei der ANDROID_ID und der Advertising-ID der Fall ist. Die GSF-ID bleibt zudem nicht konstant, falls ein Nutzer alle Daten des Google-Service Frameworks auf dem Gerät entfernen sollte. Es zeigt sich, dass die GSF-ID, die ANDROID_ID sowie die Advertising-ID wertvolle Informationen für die Identifizierung von Nutzern und Geräten liefern können, aber nur in Kombination mit anderen Identifizierungsmethoden langfristig zuverlässige Ergebnisse liefern.

3.3.4 Environmental Sound Analysis

Android-Anwendungen haben Zugriff auf die eingebauten Mikrofone eines Android-Geräts. Das Identifikations-Framework kann somit auf aufgenommene Daten der Geräusche, die in der direkten Umgebung des Nutzers auftreten, zugreifen. Informationen über die spezifischen Geräuschkulissen, die von dem Framework aufgezeichnet wurden, können anschließend mit Identifikationsnummern verknüpft werden.

Da die Eigenschaften der unterschiedlichen Mikrofone das ausgezeichnete Klangspektrum zusätzlich beeinflussen können, sind die Daten, auf denen diese Methode beruht, nicht nur von der Umgebung des Nutzers abhängig. Geräte-spezifische Hardware-Eigenschaften haben auch Einfluss auf die aggregierten Referenzdaten, weshalb diese Methode vor allem dazu geeignet ist, eine Kombination aus Nutzer und Gerät zu identifizieren.

Universalität und Erfassbarkeit

Mikrofone sind auf jedem Smartphone vorhanden, und auch Tablets werden standardmäßig damit

ausgestattet. Mithilfe dieser Mikrofone wird es Android-Anwendungen ermöglicht, eine Vielzahl an Geräuschen des Alltags aufnehmen. Das Framework kann die Identifizierungsmethode somit auf jedem handelsüblichen Gerät anwenden. Voraussetzung hierfür ist folgende *Permission*:

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
```

Es ist fast unmöglich, alle Daten, die durch das Mikrofon erhobenen wurden, über einen unbegrenzten Zeitraum zu speichern. Eine Möglichkeit, benutzerspezifische Fingerabdrücke der Klangumgebungen des Nutzers zu erstellen, ist es kontinuierliche, einzelne und wiederkehrende Geräusche des täglichen Lebens zu erkennen und zu extrahieren. Beispiele für diese Art von Klang können durch CPU-Lüfter, einer Kaffeemaschine, von PKW-Motoren oder Zügen, welche der Benutzer täglich verwendet, erzeugt werden.

Einmaligkeit und Persistenz

Obwohl es Alltagsgeräusche gibt, die in der Umwelt einiger Nutzer mit Sicherheit wiederholt vorkommen können, zeichnen sich die Kombinationen aller Geräusche für jeden Nutzer durch ihre hohe Einzigartigkeit aus. Für dann Fall das genügend Referenzdaten gesammelt wurden kann davon ausgegangen werden, dass eine einmalige Menge an Informationen vorliegt, mithilfe derer Nutzer und Geräte identifiziert werden können.

Wenn man die Gesamtheit aller Alltagsgeräusche eines Nutzers betrachtet, wird sich die Information auf der diese Identifizierungsmethode basiert, in der Regel nur langsam ändern. Da das Framework passende Filter einsetzt und so nur sich oft wiederholende Geräusche in die Datenbank als Referenzwerte mit aufnimmt, wird zudem die Speicherung von seltenen, störenden Geräuschen vermieden. Falls die Umwelt und der Alltagsablauf eines Nutzers sich nicht zu sehr ändert, kann von einer konstanten Performanz dieser Methode ausgegangen werden. Die verwendeten Referenzdaten müssen allerdings von dem Framework in regelmäßigen Abständen aktualisiert werden, um die Fehlerraten dieser Identifizierungsmethode zu reduzieren.

Datenaggregation

Das Framework schätzt die Variation der aktuellen Geräusche durch die Berechnung der Standardabweichung der Frequenzwerte im zeitlichen Verlauf. Wenn sich das Geräusch nicht zu sehr (Schwellenwert) ändert, speichert das Framework dessen Frequenzspektrum (Abbildung 22)

Das Frequenzspektrum des Audiosignals kann mithilfe der Schnellen Fourier-Transformation berechnet werden, welche wie folgt lautet:

$$f(m) = \sum_{k=0}^{2n-1} d(k) e^{-\frac{2\pi i}{2n} mk} \; m = 0, \dots, 2n-1 \tag{3.1}$$

Wobei die Feldern $d(k)$ mit $k = 0, \dots, 2n-1$ die Eingangsdaten darstellen, welche vom Mikrofon des mobilen Geräts empfangen wurden [59]. In den Felder *f(m)* mit $k = 0, \dots, 2n-1$ wird durch die Anwendung der Formel das Frequenzspektrum des aufgenommenen Alltagsgeräusch gespeichert.

Die Frequenzspektren sind in ähnlichen Gruppen gruppiert. Für den Fall, dass die Daten eines Geräusches mit einem bereits gespeicherten Frequenzspektrum übereinstimmen, wird ein entsprechender Zähler erhöht. Der Zähler kann verwendet werden, um kontinuierliche, aber nicht wiederkehrende Geräusche nach einer gewissen Zeit zu löschen.

Identifikationsverfahren

Zunächst muss die Datenaggregation des neu registrierten Benutzers abgeschlossen sein, das bedeutet es müssen genügend Daten über die Umgebungsgeräusche in der Datenbank gespeichert werden. Im nächsten Schritt werden die gespeicherten Frequenzspektren und die neu gefundenen Spektren paarweise verglichen. Für den Fall, dass die gemittelte Euklidische Distanz zweier Frequenzspektren einen bestimmten Schwellwert unterschreitet, gibt der Identifizierungsprozess die Wahrscheinlichkeit zurück, welche der gemittelten Ähnlichkeit der Umgebungsgeräusche entspricht.

Vorausgesetzt, dass das Framework genügend Daten gesammelt hat, sind die Ergebnisse der Identifikation dieser Methode in der Lage Benutzer basierend auf ihren Umgebungsgeräuschen zu unterscheiden.

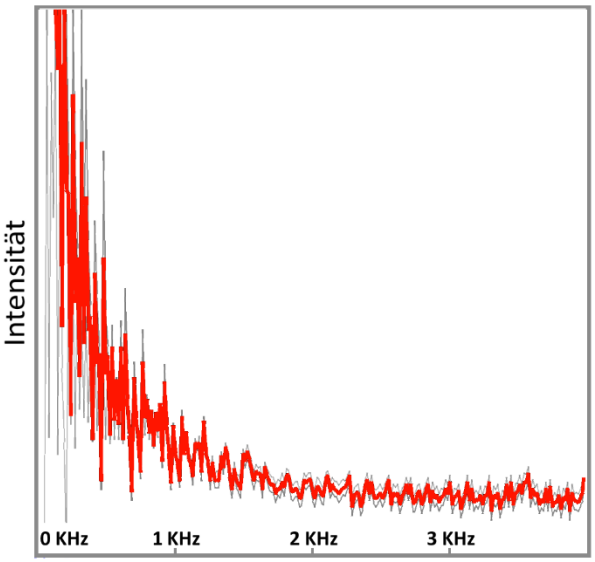


Abbildung 22: Frequenzspektrum eines Audiosamples

3.3.5 Gait Recognition

Die Schritte, welche ein Benutzer jeden Tag tätigt, können verwendet werden, um Benutzerprofile zu erstellen. Allerdings ist eine Analyse der Verteilung der Schritte nicht die einzige Möglichkeit, Informationen über bestimmte Gehgewohnheiten zu sammeln. Die Sensorwerte, die der Beschleunigungsmesser während des Zeitintervalls eines Schrittes bietet, können zusätzlich analysiert werden. Das Ziel dieses biometrische Ansatzes ist die Identifizierung von Benutzern auf der Grundlage ihrer individuellen Gangarten.

Universalität und Messbarkeit

Wie schon in Kapitel 3.1.11 erläutert, sind Beschleunigungssensoren auf Android-Geräten Standard, die Identifizierungsmethode kann somit auf einer großen Auswahl an Geräten Referenzdaten extrahieren. Die Genauigkeit dieser Methode steigt mit der Abtastrate der Beschleunigungssensoren. Die Präzision und Geschwindigkeit, mit der die Beschleunigungssensoren Daten liefern, hat sich seit der ersten Android-Version deutlich verbessert. Dies wird in den Hardwarespezifikationen der Hersteller beschreiben und hat sich auch in der Evaluation der Methoden bestätigt. Auf allen Geräten, die während der Evaluation im Einsatz waren, reichten die Abtastraten der Beschleunigungssensoren aus, um anhand der Daten die Schritte eines Nutzers zu erkennen. Es hat sich gezeigt, dass die Methode am besten funktioniert, wenn das Smartphone möglichst nahe am Körper getragen wird, und das Gerät relativ zum Körper des Nutzers gesehen in einer Position bleibt. Die Methode eignet sich somit in erster Linie für Smartphones, die im alltäglichen Gebrauch in der Hosentasche eines Nutzers getragen werden. Für die Erfassung der Referenzdaten werden keine zusätzlichen *Permissions* benötigt.

Einmaligkeit und Persistenz

Die Gangart eines Menschen ist sehr individuell. Die gesammelten Daten zeichnen sich somit durch eine sehr hohe Einmaligkeit aus. Durch die genaue Analyse des menschlichen Ganges können Individuen mit sehr hoher Präzision unterschieden werden [60]. Da die Messwerte, die den Gang der Nutzer beschreiben, allerdings auch von den Eigenschaften der Sensoren sowie der verbauten Hardware der Geräte abhängt, kann diese Methode in erster Linie für die Identifizierung spezifischen Nutzer-Gerät-

Kombinationen eingesetzt werden.

Die Eigenschaften, die den Gang eines Menschen auszeichnen, ändern sich in der Regel nur langsam. Das Framework aktualisiert auch während der Identifizierungsphase kontinuierlich die Menge der vorhandenen Referenzdaten, wodurch sich die Identifizierungsmethode auf leichte Veränderungen einstellen kann und langfristig gute Ergebnisse liefert. In seltenen Fällen kann es jedoch sein, dass sich der individuelle Gang eines Nutzer plötzlich ändert, beispielsweise nach einem Unfall, oder falls sich die körperliche Verfassung eines Nutzers ändert. Das Framework greift in solch einem Szenario auf die Informationen der anderen Methoden zurück, und bringt die entsprechenden Referenzdaten auf den neuesten Stand.

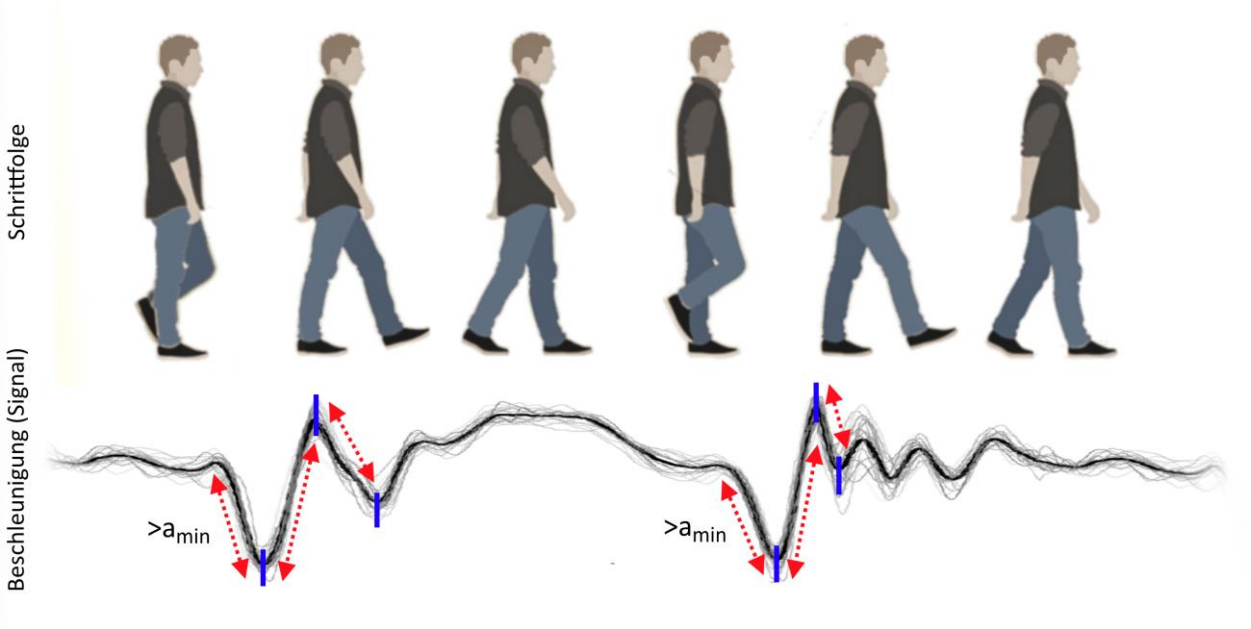


Abbildung 23: Bewegungsablauf und Betrag der Beschleunigung in X-, Y- und Z-Richtung

Datenaggregation

Informationen darüber, wann ein Schritt getätigt wurde, kann durch die Analyse der lokalen Extrema der Beschleunigungssequenzen erhalten werden. Ein Zyklus besteht aus zwei aufeinander folgenden Schritten, die extrahiert werden können, wie in Abbildung 23 dargestellt. Das Schritt-Signal wird normalisiert (Ausmaß und Dauer), um ähnliche Signal-Daten zu vergleichen und clustern zu können. Um die unterschiedlichen Laufzeiten der Schritte anzupassen, wird kubische Interpolation verwendet.

Ein charakteristischer Wert für den Unterschied zwischen den neu gesammelten Daten und den bereits gespeicherten Daten wird mithilfe von *Dynamic-Time-Warping* (DTW) berechnet, welches einen paarweisen Distanzwert für zeitliche Sequenzen zurückliefert [61]. DTW wurde für die Clusterung der Sequenzen als Distanzmaß gewählt, weil damit auch Schritte unterschiedlicher Geschwindigkeit und Dauer mithilfe eines geeigneten Zeitverzerrungspfades verglichen werden können. Den Abstand zwischen zwei Sequenzen X bzw. Y unterschiedlicher Dauer wird als Summe lokaler Distanzen $d_{i,j} = d(x_i,y_j)$ entlang des Zeitverzerrungspfades bestimmt. Als lokale Distanz wird in der Implementierung des Android-Framework die *Manhattan-Distanz*[7] verwend, was vor allem auf mobilen Geräten Geschwindigkeitsvorteile mit sich bringt.

[7] Manhattan-Distanz: Entspricht der Summe über die absoluten Differenzen der Einzelkoordinate zweier Punkte.

Ähnliche Schrittdaten werden gruppiert. Jedes Mal, wenn ein Benutzer bei einem Spaziergang sein Gerät in eine andere Tasche steckt oder seine Schritt- oder Laufgeschwindigkeit variiert (Schwellenwert), werden neue Schritt-Cluster erstellt. Mithilfe von DTW wird erreicht, dass Schritte unabhängig von diesen beiden Eigenschaften in gemeinsame Clustern eingeteilt werden können, was den Speicherverbrauch des Verfahrens insgesamt reduziert.

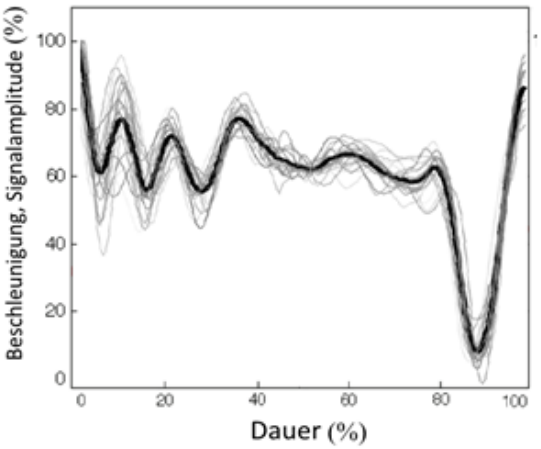


Abbildung 24: Schritt-Cluster

Der Durchschnittswert der Beschleunigung wird für jeden Cluster errechnet. Die Genauigkeit der gemittelten Werte kann iterativ verbessert werden, indem das Gewicht der Daten jedes einzelnen Schritts je nach Ähnlichkeit mit dem zuvor berechneten Durchschnitt angepasst wird. Um die Datenmenge zu reduzieren, ist die Anzahl der gespeicherten Samples pro Cluster beschränkt. Daten mit dem größten Abstand zu den gemittelten Werten werden verworfen.

Identifikationsverfahren

Bei dem Identifikationsprozess werden die neu gesammelten Sequenzen mit den gespeicherten Referenzdaten verglichen. Um den Algorithmus zu beschleunigen, werden die gemittelten Werte der Signalstärken zuerst mit Hilfe von DTW verglichen. Für den Fall, dass der Unterschied im Durchschnitt einen bestimmten Schwellenwert einhält, werden die aufgezeichneten Werte des Beschleunigungsmessers einzeln verglichen, und ein Ähnlichkeitswert für jedes Paar Schritte durch Einsatz von DTW berechnet. Mohammad Derawi und Patrick Bours zeigten in ihrer Arbeit, dass der einzelne Vergleich deutlich niedrigere Fehlerquoten bietet als der begrenzte Vergleich der Mittelwerte [60]. Für den Fall, dass der minimale Unterschied eine zweite Schwelle einhält, kann der Benutzer mit einer bestimmten Wahrscheinlichkeit identifiziert werden.

3.4 Vergleich der Methoden

Die folgenden Tabellen fassen die Eigenschaften der einzelnen Methoden zusammen, die bei der Implementierung und späteren Verwendung beachtet werden müssen. Unterschieden wird zwischen Nutzeridentifikation (Tabelle 4), Geräteidentifikation (Tabelle 5), und kombinierte Nutzer- und Geräteidentifikation (Tabelle 6). Neben der Einmaligkeit werden die Persistenz, die Verfügbarkeit und der Zeitaufwand für die Datenaggregation bewertet. Die Bewertung der Methoden berücksichtigt Erfahrungswerte aus der Evaluation sowie verfügbare Statistiken ergeben sich zum Teil auch aus der Ableitung logischer Zusammenhänge.

Bewertung der Methoden nach den Kriterien in Tabelle 4, Tabelle 5 und Tabelle 6:

- **++** → Das Kriterium kann als besondere Stärke der Methode angesehen werden.
- **+** → Die Methode weist überdurchschnittliche Eigenschaften hinsichtlich dieses Kriteriums auf.
- **o** → Das Kriterium wird von der Methode durchschnittlich erfüllt.
- **-** → Die Methode kann das Kriterium nur eingeschränkt erfüllen.
- **--** → Das Kriterium stellt eine Schwachstelle der Methode dar.

Es wird zudem aufgelistet, wie viele *Permissions* für den Einsatz der jeweiligen Methoden benötigt werden.

Tabelle 4: Benutzeridentifizierung - Vergleich der Methoden

	Einmaligkeit	Persistenz	Verfügbarkeit	Zeitaufwand	Permissions
SIM	++	+	+	++	1
Accounts	++	++	+	+	1
Bluetooth	+	+	+	+	1-2
WLAN	+	+	+	+	1-2
SD-Karte	++	+	-	+	1
Kontakte	+	+	+	+	1
Call-Log	+	+	+	+	1
SMS History	+	+	+	+	1
Packages	o	o	++	+	0
Dateien	+	o	o	+	1
Musik	+	-	o	+	0
Akku	o	o	++	o	1
Schritterkennung	o	+	++	o	0
Locations	++	++	++	o	1-5
Orientations	++	+	++	o	0

Eine hohe Verfügbarkeit der verwendeten Methoden auf handelsüblichen Android-Geräten ist in der Regel gegeben, wie aus Tabelle 4 ersichtlich ist. Nur die Methode, die auf SD- Karten zurückgreift, ist in ihrer Verfügbarkeit ein wenig eingeschränkt, kann aber durch gute Persistenz-Eigenschaften und die Einmaligkeit der Seriennummer einer SD-Karte punkten.

Es ist ersichtlich, dass eine ganze Reihe an Methoden das Kriterium der Einmaligkeit der verwendeten Informationen erfüllt. Es lassen sich somit Nutzer besonders gut anhand dieser Informationsquellen von anderen unterscheiden. Bei den biometrischen Verfahren kann hinzugefügt werden muss, dass die verwendeten Informationsquellen eine tatsächliche Einmaligkeit aufweisen, sich aber bei den mathematischen Vergleichen der Merkmalsvektoren naturgemäß falsch-negative und falsch-positive Resultate nicht zu hundert Prozent vermeiden lassen.

Bei der Bewertung des Zeitaufwands für die Enrolment-Phase lässt sich sagen, dass Methoden, die das Verhalten der Nutzer untersuchen, tendenziell länger brauchen als Methoden, die bereits vorhandene Daten auf einem gerät untersuchen. Biometrische Verfahren benötigen ist gemeinsam dass die verwendeten Referenzdaten erst einmal von der entsprechenden Person gesammelt werden müssen, bevor die Person anschließend identifiziert werden kann.

Aus Tabelle 5 lassen sich die Bewertungen für die Methoden, die für die Geräteidentifizierung in Frage kommen, ablesen. Im Gegensatz zu den Methoden, die einen Benutzer identifizieren soll, werden bei der Geräteidentifikation vor allem die Eigenschaften eines Geräts, wohingegen bei Nutzers oftmals das spezifische Verhalten durch biometrische Verfahren untersucht wird.

Tabelle 5: Geräteidentifizierung- Vergleich der Methoden

	Einmaligkeit	Persistenz	Verfügbarkeit	Zeitaufwand	Persmissions
Hardware	--	++	++	++	0
Bluetooth	++	+	+	++	1-2
WLAN	++	+	+	++	1-2
IMEI	++	+	+	++	1
Serial Number	++	++	--	++	0
Packages	++	o	++	++	1
Files	++	o	++	++	1
Pixels	++	++	-	++	1
Dark-Frames	+	+	+	o	1

Bis auf die Hardware-Spezifikation zeichnen sich alle Methoden durch die Einmaligkeit der Referenzdaten aus. Hardware-Spezifikationen können trotzdem benutzt werden, um die Menge aller Geräte zu Clustern, um so die Performanz und Qualität der anderen Methoden zu steigern.

Da sich die physischen Eigenschaften eines Smartphones oder Tablets in der Regel nicht ändern, zeigen einige Methoden eine besonders gute Persistenz. Bei Seriennummern und MAC-Adressen ist die Persistenz nur eingeschränkt, weil sich diese unter bestimmten Umständen vom Nutzer ändern lassen, oder sich beim Zurücksetzen auf Werkseinstellungen ändern können, wie auch die Eigenschaften der gespeicherten Dateien und Packages. Hardwareeigenschaften zeigen naturgemäß eine deutlich bessere Persistenz als Informationen, die auf Software oder Daten basieren. Die meisten Methoden sind sofort nach der Installation des Frameworks verfügbar. Die Methode, die Dark-Frames verwendet, benötigt Bilder, die in der absoluten Dunkelheit von den internen Kameras angefertigt wurden, weshalb unter Umständen gewartet werden muss bevor ein Gerät anhand dieser Methode identifiziert werden kann.

Tabelle 6: Kombinierte Identifizierung von Nutzern und Geräten- Vergleich der Methoden

	Einmaligkeit	Persistenz	Verfügbarkeit	Zeitaufwand	AP*
Gait Recognition	++	+	o	o	1
Sound	++	+	+	o	1
Advertising ID	++	-	+	++	0
ANDROID_ID	++	o	+	++	0
GSF ID	++	o	+	++	0

Die Methoden, die in Tabelle 6 beschrieben werden, basieren sowohl auf Informationen des jeweiligen Nutzers, als auch des spezifischen Geräts. Dies hat zur Folge, dass die Referenzdaten sich durch eine sehr hohe Einmaligkeit auszeichnen. Da die Methode von Nutzer als auch Gerät abhängt, ist die Persistenz der verwendeten Methoden eher mittelmäßig. Die gesicherten Referenzdaten müssen öfters aktualisiert werden, um die Falsch-Negativ-Rate zu senken. Die Verfügbarkeit der Methoden ist auf den allermeisten Geräten gegeben, wobei die Methode, die Daten vom Nutzer oder der Umwelt sammeln etwas länger brauchen, bis genug Referenzdaten für die Identifizierung gesammelt wurden.

4 Entwicklung eines Android-Frameworks

Im Rahmen dieser Arbeit wurde ein Android-Framework implementierten, welches die vorgestellten Methoden verwendet, um Benutzer und Geräte zu identifizieren. Das Framework kann in Android-Anwendungen als Android-Library aufgenommen werden, oder alternativ als Java-Archiv, welches die Klassenbibliothek des Frameworks enthält. Neben dem Android-Framework wurde eine Java-Servlet entwickelt, welches die gesammelten Daten zentral in einem Cloud-Server abspeichert und vergleicht.

Bei der Implementierung des Frameworks sind die Überlegungen aus Kapitel 3 direkt in die Entscheidungsfindung mit eingeflossen. Der Ablauf der Identifizierung als Ganzes orientiert sich stark an der prinzipiellen Vorgehensweise welche in Kapitel 2 dargestellt wurde. Im Folgenden wird erläutert, welche Entscheidungen hinsichtlich der Implementierung des Frameworks getroffen wurden, und wie das Gesamtsystem den Anforderungen entsprechend gestaltet wurde.

Die Komponenten des Systems wurden mithilfe der Eclipse - Entwicklungsumgebung (IDE) entwickelt. Die Entscheidung für Eclipse brachte im Verlauf der Implementierungsarbeit einige spezifische Vorteile mit sich. So konnte die gesamte Implementierung des Systems innerhalb einer Entwicklungsumgebung durchgeführt werden, da Eclipse sowohl für die Entwicklung des Android-Frameworks als auch die Implementierung des Java Servlets geeignet ist. So zeichnet sich Eclipse durch eine große Auswahl an anforderungsspezifischen Plug-Ins aus. Bibliotheken, die in beiden Teilprojekten benötigt wurden, ließen sich einheitlich verwalten. Das Android-Framework und das Java-Servlet teilen sich zudem eine Menge gemeinsamer Projektdateien. Komponentenspezifische Daten, wie beispielsweise Packages, Klassen oder Ressourcen konnten, falls benötigt, ohne Probleme von einem Teilprojekt in das andere importiert werden. Die Entwicklung des Gesamtsystems gestaltete sich somit sehr einheitlich, und mögliche Kompatibilitätsprobleme zwischen den Teilprojekten wurden durch eine komponentenübergreifende Verwaltung der Projektressourcen, der Konfigurationen und Bibliothek-Versionen von Anfang an vermieden werden. Die Debugging-Phase der Implementierung wurde zudem durch die Verwendung einer einzigen Entwicklungsumgebung vereinfacht. [62]

Um Anwendungen für die Android-Plattform zu entwickeln, muss das Android Development Tools (ADT) - Plugin in Eclipse integriert werden [63]. Für die Implementierung des Servlets wurde zudem ein Tomcat-Server (v 7.0) innerhalb der Eclipse-DIE eingerichtet, und für Testzwecke ein lokaler MySQL-Server für die Anbindung an das Java-Servlet konfiguriert. Die Entwicklung des Gesamtsystems konnte somit vorerst lokal auf einem einzigen Arbeitsrechner durchgeführt werden. Die URL-Adresse des Java-Servlets bzw. des entsprechenden Servers kann in der anwendungsspezifischen Konfiguration des Android-Frameworks entsprechend festgelegt werden. So wurde für die spätere Evaluation des Systems das Java-Servlet dann innerhalb eines Cloud-Servers mit globaler IP-Adresse ausgeführt.

Die Entscheidung, als Server-Komponente ein Java-Servlet einzusetzen, hat sich als auf lange Sicht nützliche und zeitsparende Alternative erwiesen. So ist es beispielsweise möglich, serialisierbare Objekte zwischen dem Android-Framework und dem Java-Servlet in komprimierter Form zu tauschen. Java stellt hierfür geeignete Schnittstellen und Klassen zur Verfügung. Wie bereits erwähnt, teilen sich Android-Framework und Java-Servlet gemeinsame Quellcode-Dateien und Projektressourcen. Als die Implementierung der anwendungsspezifischen Methoden für den Datenaustausch und die Datenwiederherstellung fertiggestellt waren, gestaltete sich daher die Weiterentwicklung des Gesamtsystems sehr effizient. Die Wartung sowie die Wiederverwendbarkeit der Teilkomponenten des Systems werden durch die gemeinsame Programmiersprache zusätzlich vereinfacht.

4.1 Architektur

Das im Rahmen dieser Arbeit entwickelte System zur Identifizierung von Nutzern und Geräten lässt sich grundsätzlich in zwei Hauptkomponenten unterteilen. Zum einen kommt ein Android-Framework zum Einsatz, welches auf Android-Geräten eingebunden in eine Anwendung installiert werden kann. Das Framework muss so konfiguriert werden, dass eine Verbindung mit einem Java-Servlet hergestellt werden kann, sobald Referenzdaten über das Gerät oder den Nutzer gesammelt wurden. Die Konfiguration der Implementierung hängt somit direkt von der entsprechenden Anwendung ab, da bei der Einbindung des Frameworks die Parameter der Konfiguration, wie beispielsweise dir URL des Servers, direkt übergeben werden müssen. Das Java-Servlet stellt die zweite Hauptkomponente des Gesamtsystems dar. In der Regel kommuniziert eine Gruppe von Android-Frameworks mit einem Java-Servlet, welches in der Cloud ausgeführt werden kann.

Abbildung 25 zeigt einen Überblick über das Gesamtsystem. Neben den Klassen, die für die Datenaggregation (Android-Framework) sowie der eigentlichen Identifizierung von Nutzern und Geräten zuständig sind (Java Servlet), existiert eine Reihe weiterer essentieller Komponenten. Die entsprechenden Klassen sind in erster Linie für die Übermittlung, Speicherung und Verwaltung der gesammelten Referenzdaten zuständig.

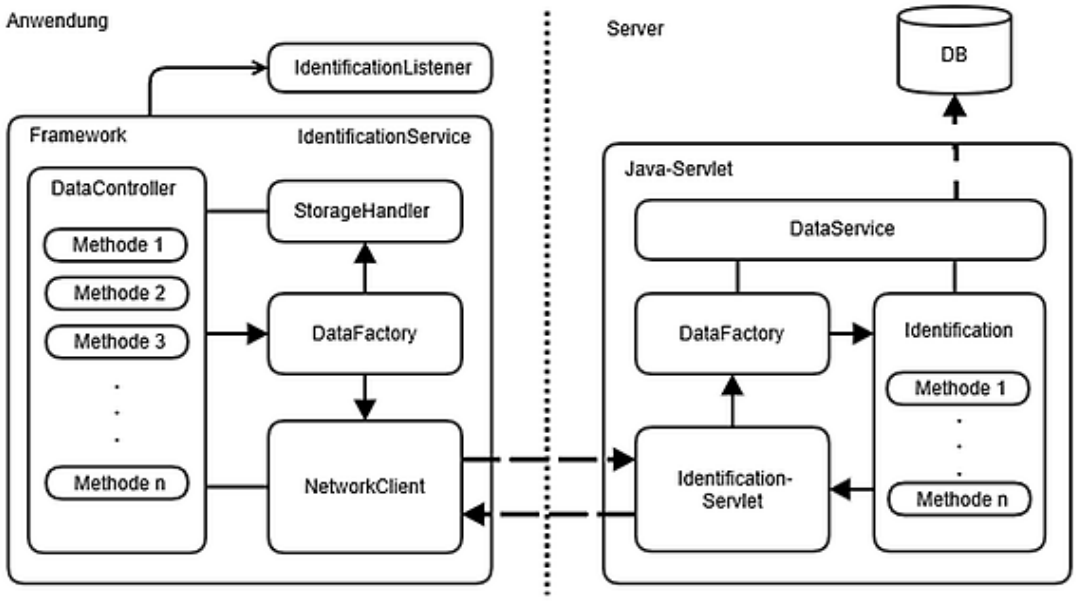


Abbildung 25: Aufbau des Gesamtsystems

In den folgenden Kapiteln wird auf die dargestellten Details näher eingegangen, und die entsprechenden Implementierungen der Teilkomponenten vorgestellt.

4.2 Android-Framework

Das Android-Framework sammelt und analysiert die Informationen welcher zur Identifizierung erforderlich sind. Datenaggregation, Vorverarbeitung, Feature-Extraktion, Pseudonymisierung, Serialisierung und Kompression der gesammelten Daten erfolgt durch das Android-Framework.

Einige Methoden, welche in dieser Arbeit vorgestellt werden, benötigen spezielle *Android-Permissions* um korrekt zu arbeiten. Das Framework kann jedoch so konfiguriert werden, dass die Menge der verwendeten Methoden nur die Berechtigungen benötigt, die bereits durch die Android Anwendung selbst erforderlich sind.

IdentificationService

Da das Framework prinzipiell keine direkte Interaktion mit den Nutzer benötigt, und längerfristig bzw. durchgehend im Hintergrund Informationen über Nutzer und Gerät sammeln soll, bietet sich als Basis für den Aufbau des Frameworks ein sogenannter Android-Service an. Die Implementierung des Services, welcher als Member-Klassen die wesentlichen Funktionen des Frameworks beinhaltet, wird innerhalb des Frameworks als *IdentificationService* bezeichnet, und findet sich im Package `com.tum.ident` in der Datei `IdentificationService.java` wieder. Der *IdentificationService* kann wie in Kapitel 4.4 erläutert an die eigentliche Anwendung gebunden und bei Bedarf gestartet werden. Da das Framework aber auch von Beginn an, also auch unmittelbar nach einem Neustart des Android-Geräts im Hintergrund laufen sollte, wird der *IdentificationService* zudem auch innerhalb einer *Broadcast*-Klasse gestartet, welche die *BroadcastReceiver* - Schnittstelle implementiert. Damit ein *BroadcastReceiver* nach dem Bootvorgang benachrichtig werden kann, wird einen eine *Permission* benötigt:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

Außerdem muss Android per *AndroidManifest* mitgeteilt werden, welche Klasse innerhalb der Anwendung die abstrakte *BroadcastReceiver*-Klasse ableitet, und nach dem Start des Geräts aufgerufen und benachrichtigt werden soll. Dies geschieht mithilfe eines sogenannten *Intent-Filters*.

DataController

Die wichtigste Member-Klasse des *IdentificationService* wird als *DataController* bezeichnet. Die *DataController*-Klasse beinhaltet alle Klassen, welche für die eigentliche Datenaggregation zuständig sind. Der *DataController* sammelt die Daten, die von den einzelnen Methoden aggregiert wurden.
Eine weitere Aufgabe des *DataController* besteht darin, in gewissen Abständen die gesammelten Referenzdaten zu erneuern. Jede Identifizierungsmethode erhält zu diesem Zweck eine Timer-Variable, in der gespeichert wird, wann das letzte Mal die gesammelten Daten aktualisiert wurden. Eine Aktualisierung der Daten durch den *DataController* kann zudem auch dann stattfinden, falls bedeutende Änderungen der Informationsquellen festgestellt werden konnten.

Einige Daten können sofort nach Start des Frameworks gesammelt und an das Servlet geschickt werden. Die gesammelten Daten der Methoden, die auf solche Informationen zugreifen, werden vom *DataController* gebündelt an das Servlet geschickt. Die Referenzdaten von Methoden, die über einen längeren Zeitraum Daten sammeln müssen, bevor sie für die Identifizierung eingesetzt werden können, werden hingegen in einzelnen Paketen an das Servlet übertragen. Dies geschieht immer dann, wenn sich der aktuelle Datensatz durch neu hinzugekommene Informationen signifikant geändert hat, oder bestimmten Zeitintervallen, falls eine sofortige Übertragung fehlschlägt.

Das Framework greift auf Sensor- und Positionsdaten zurück, um einen Teil der Referenzdaten zu aggregieren. Die Daten können von Framework in der Regel mithilfe von *Listenern* angefordert werden. Für die *Listener*, welche Daten beispielsweise von einem Bewegungssensor oder Magnetometer anfordern, kann spezifiziert werden, wie schnell die Sensordaten dem *Listener* übergeben werden sollen. Die verfügbare maximale Geschwindigkeit hängt hierbei natürlich auch mit den jeweiligen Hardware-Eigenschaften des Geräts zusammen

Ein Teil der Member-Klassen stellen die Funktionalität der in Kapitel 3 vorgestellten Identifizierungsmethoden zur Verfügung. Die Implementierungen dieser Methoden, welche für die Datenaggregation innerhalb des Android-Frameworks verwendet werden, finden sich in folgenden Packages bzw. Klassen (Tabelle 7), und sind auf der beigelegten CD enthalten.

Tabelle 7: Implementierungen der Methoden

Methode	Package	Klassen
SIM:	*.sim	SIMData.java, SIMItem.java
Benutzerkonten:	*.accounts	AccountData.java, AccountItem.java
Bluetooth:	*.bluetooth	BluetoothData.java, BluetoothItem.java
WLAN:	*.wlan	WLANData.java, WLANItem.java
SD-Card:	*.system	SystemData.java, SDCardItem.java
Kontakte	*.calllog *.contacts *.sms	CallLogData.java, CallLogItem.java ContactData.java, ContactItem.java SMSData.java, SMSItem.java
Anwendungen:	*.apps	PackageData.java, PackageItem.java
Nutzerdateien:	*.files	FileData.java, FileItem.java, FileItemList.java
Musik:	*.music	MusicData.java, MusicItem.java, MusicItemList.java
Akku:	*.battery	BatteryData.java, BatteryItem.java, BatteryItemList.java
Schritt-Erkennung:	*.gait *.stepdetector	StepCounter.java, StepStatistics.java StepDetector.java
Positionsdaten:	*.locations	LocationData.java, LocationItem.java,
Geräte-Orientierung:	*.orientation	OrientationData.java, OrientationItem.java, LocationArea.java, LocationAreaList.java
Hardware:	*.system	SystemData.java, Device.java
IMEI:	*.system	SystemData.java, DeviceIDItem.java
Seriennummer:	*.system	SystemData.java, DeviceIDItem.java
System-Pakete und -Dateien:	*.apps	PackageData.java, PackageItem.java
Pixelfehler:	*.camera	CameraData.java, CameraPixel.java, CameraPixelList.java
Dark-Frames:	*.camera	CameraData.java, DarkFrame.java
Advertising ID:	*.system	SystemData.java, DeviceIDItem.java
ANDROID_ID:	*.system	SystemData.java, DeviceIDItem.java
GSF ID:	*.system	SystemData.java, DeviceIDItem.java
Sound Analysis:	*.spectrum	SpectrumData.java, SpectrumItem.java, SpectrumItemList.java
Gait Recognition:	*.gait	StepData.java, StepItem.java, StepCluster.java, StepClusterList.java, StepStatistics.java

Die in Tabelle 7 vorgestellten Klassen sind für die Aggregation der benötigten Informationen zuständig. Die gesammelten Daten werden in serialisierbare Klassen gespeichert. Ein Teil der entsprechenden Implementierungen ist in den Packages `*.user`, `*.device` sowie `*.userdevice` zu finden.

Die *DataController* Klasse implementiert die *Runnable*-Schnittstelle. Alle Identifizierungsmethoden werden innerhalb der entsprechenden `run`-Methode nebenläufig ausgeführt, ohne die eigentliche Android-Anwendung im Ablauf der Ausführung zu stören.

DataFactory

Die aggregierten Daten werden an den *DataController* übergeben. Dieser serialisiert und komprimiert die Objekte mithilfe der *DataFactory*-Klasse. Die Referenzdaten können anschließend an das Java-Servlet übertragen werden. Außerdem werden die erhaltenen Daten im internen Speicher des Android-Geräts für die spätere Verwendung zwischengespeichert.

StorageHandler

Mithilfe der *StorageHandler*-Klasse werden die Referenzdaten vom *DataController* lokal zwischengespeichert und bei Bedarf wieder geladen. Die Daten werden also sowohl der Servlet-Datenbank als auch auf dem mobilen Gerät gespeichert. Dies ist nötig, da zum einen eine Internetverbindung nicht immer verfügbar ist, und die neu aggregierten Referenzdaten zu einem späteren Zeitpunkt an das Servlet geschickt werden sollen. Identifizierungsmethoden, die über längere Zeiträume Information sammeln, ersetzen zudem die schon gesammelten Daten nicht komplett, sondern ergänzen diese nach und nach. Beispiele für diese Art von Identifizierungsmethode wären die Verfahren welche die Positionen der Geräte oder die Schritte der Nutzer untersuchen. Das Framework überprüft die gespeicherten Daten auf Integrität, und kann dadurch Änderungen, welche durch externe Eingriffe vorgenommen wurden, erkennen. Um die Daten auf dem Gerät zu speichern benötigt das Framework folgende *Permission*:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Die Implementierung des *StorageHandlers* sowie weitere Details findet sich im Package `com.tum.ident.storage` in der Datei `StorageHandler.java`.

NetworkClient

Falls eine Internetverbindung zum Java-Servlet verfügbar ist, können die gesammelten Referenzdaten an übermittelt werden. Das Framework kann so konfiguriert werden, dass die Daten nur bei bestehender WLAN-Verbindung und nicht bei mobilem Internet übertragen werden.
Der *DataController* übergibt die serialisierten, komprimierten Daten, die von der *DataFactory*-Klasse generiert werden, der *NetworkClient*-Klasse. Die *NetworkClient*-Klasse übernimmt dann die verschlüsselte Übertragung der Daten an das Servlet. Um eine verschlüsselte SSL-Verbindung zwischen dem Android-Framework und dem Java-Servlet aufzubauen, wurde mithilfe des *Java Keytools* eine .keystore-Datei erstellt. Der Tomcat-Server wurde so konfiguriert, dass er nur Anfragen über sichere Verbindungen annimmt. Das Android-Framework benutzt den erstellten *KeyStore*, um die Referenzdaten verschlüsselt und somit vor Fremdzugriff geschützt an das Java-Servlet zu übertragen.

IdentificationListener

Um Informationen über den aktuellen Zustand der Identifizierung zu erhalten, muss die Anwendung, welche das Framework einbindet, die *IdentificationListener* - Schnittstelle implementieren. Sobald das Framework Daten vom Servlet erhält, werden diese intern gespeichert, ausgewertet und mithilfe der IdentificationListener-Schnittstelle an die Anwendung übertragen. Dadurch kann sichergestellt werden, dass die Anwendung zeitnah auf die eingehenden Ergebnisse der Identifizierung reagieren kann.

4.3 Java-Servlet

Das Java-Servlet wird auf einem Server in der Cloud ausgeführt. Die von der Android-Bibliothek gesammelten Daten werden durch das Servlet in einer Datenbank gespeichert. Deserialisierung, Dekompression, Datenspeicherung, Template-Vergleich und die Kombination der Ergebnisse erfolgt durch das Java-Servlet. Ankommende Vorlagen werden mit den gespeicherten Daten verglichen. Die Ergebnisse der einzelnen Methoden werden kombiniert zu einem Endergebnis, wie in Kapitel 3.4 beschrieben. Für den Fall, dass ein Benutzer oder Gerät identifiziert werden kann, sendet das Java-Servlet eine entsprechende Meldung zurück an die Android-Bibliothek, die die Android Anwendung über das Ergebnis informiert.

IdentificationServlet

Wie in Kapitel 1.5 beschrieben, muss das IdentificationServlet, welches die Referenzdaten vom Android-Framework erhält und die entsprechenden Ergebnisse des Identifizierungsvorgangs zurückschickt, eine `doPost` - Methode implementieren, welche in der `javax.servlet.http.HttpServlet`-Schnittstelle enthalten ist. Listing 12 zeigt die Implementierung der `doPost` - Methode innerhalb des IdentificationServlets

Listing 12: doPost-Methode des IdentificationServlets

```
01 protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
02    if(prepared==false){
03          prepared = initDatabase();
04    }
05    if(prepared){
06          byte[] answer = DataFactory.newRequest(request);
07          OutputStream out=response.getOutputStream();
08          if(out!=null){
09                out.write(answer);
10              out.close();
11          }
12    }
13    else{
14          [...]
15    }
16 }
```

Falls beim Start des Servlets noch keine Verbindung zur MySQL-Datenbank aufgebaut werden konnte, wird beim Aufruf der `doPost` - Methode des IdentificationServlets der Verbdingsaufbau erneut initialisiert. Anschließend wird geprüft, ob die Datenbank alle benötigten Tabellen enthält und der Konfiguration des Servlets entspricht. Falls die Vorbereitung der Datenbank erfolgreich war, werden die Daten der Anfrage der *DataFactory* übergeben. Das serialisierte Ergebnis der Anfrage wird zurückgegeben, und mithilfe eines `OutputStreams` als Rückgabewert dem Android-Framework zurückgesendet.

DataFactory

Die *DataFactory*-Klasse enthält Member-Methoden, um die ankommenden Daten wiederherzustellen. Die Daten werden erst dekomprimiert und anschließend deserialisiert. Bei der Deserialisierung wird die Version der Datenstrukturen überprüft. Diese muss mit den Versionen des Android-Frameworks übereinstimmen. Nachdem ein Objekt erfolgreich wiederhergestellt wurde, ruft die *DataFactory* eine objektspezifische Methode auf, welche die weitere Bearbeitung der Anfrage übernimmt. Die Anfragen werden zudem in drei Typen unterschieden. Das Android-Framework kann zum einen komplett neuen

Datensatz an das Servlet schicken. In diesem Fall müssen neue identifikationsnummern für das entsprechende Gerät und den Nutzer erstellt werden, welche den neu gesammelten Referenzdaten in der Datenbanktabelle zugeordnet werden. Des Weiteren kann das Android-Framework dem Servlet mitteilen, dass ein bestehender Datensatz aktualisiert werden soll. Die bereits gespeicherten Referenzdaten werden in diesem Fall durch die neu eintreffenden Informationen ersetzt. Falls ein neuer Datensatz eintrifft aktualisiert das Java-Servlet zudem die Einträge in der Datenbank, in denen die Ähnlichkeit zweier Referenzdatensätze abgelegt wird. Als dritte Option besteht die Möglichkeit dass das Framework eine Aufforderung zur Identifizierung eines Nutzers oder Geräts an das Servlet schickt. Es werden die entsprechenden Einträge der Datenbanktabelle zurückgeliefert, und an das Android-Framework gesendet.

DataService

Die *DataService*-Klasse ist für die Kommunikation mit der MySQL-Datenbank zuständig. Die Klasse stellt Funktionen zur Verfügung, welche zum Einfügen und Aktualisieren von Daten in die Datenbank verwendet werden. Zudem können mithilfe der *DataService*-Klasse Daten aus der Datenbank abgefragt werden. Der *DataService*-Klasse können ohne weiteres Datenobjekte, die direkt von einem Android-Framework empfangen wurden, übergeben werden. Die *DataService*-Klasse extrahiert die Daten der entsprechenden Member-Variablen, und fügt die Daten der passenden Datenbanktabelle hinzu. Größere Datenmengen wie beispielsweise Informationen über Dark-Frames werden in separaten Dateien gespeichert, und nur ein Verweis auf die Datei in der Datenbank gespeichert. Dem *DataService* muss neben dem Objekt selbst immer auch eine entsprechende Identifikationsnummer übergeben werden, damit die einzutragenden Referenzdaten mit einem Nutzer oder Gerät assoziiert werden können.

Identification

Die eigentliche Identifikation von Geräten und Nutzern erfolgt durch das Java Servlet, wohingegen die Datenaggregation von dem Android-Framework übernommen wird. Für den Identifikationsvorgang wird innerhalb des Java Servlets eine eigene Klasse bereitgestellt, welche im Folgenden vorgestellt werden.

Sobald neue Referenzdaten in die Datenbank eingefügt werden, muss die Tabelle, welche die Resultate der einzelnen Methoden sowie das Gesamtergebnis der Identifikation enthält aktualisiert werden. Das Java-Servlet lädt dazu aus der Datenbank alle Daten, die mit den neu hinzugefügten Referenzdaten gemeinsame Charakteristika aufweisen. Wie in Kapitel 2.4 beschrieben, werden die aussagekräftigsten Merkmale eines Referenzdatensatzes in der Datenbank indexiert. Für einen Teil der Daten, beispielsweise bei den pseudonomisierten Informationen, kann daraufhin das Endresultat für die jeweilige Methode direkt berechnet werden. Falls identische pseudonomisierte Daten in der Datenbank gefunden wurden, handelt es sich um einen teilweisen Match. Diese Art von Methoden aggregiert diesbezüglich Daten, die als Listen oder Mengen von pseudonomisierten Daten dargestellt werden können. Um zwei dieser (gewichteten) Gruppen miteinander zu vergleichen, wird der in Listing 14 vorgestellte Algorithmus verwendet. Der Ergebniswert beschreibt die Ähnlichkeit zwischen zwei Mengen oder Listen in einem Bereich zwischen 0 (unähnlich) und 1 (identisch).

Für Methoden, deren Referenzdaten auf Informationsquellen wie beispielweise Sensoren, Ortskoordinaten oder Eigenschaften der Kamerabilder basieren, muss eine andere Vorgehensweise gewählt werden. Die Daten können in diesem Fall nicht als Menge von diskreten Werten interpretiert werden, weshalb die jeweiligen Berechnungen sehr methodenspezifisch gewählt werden müssen. Die prinzipiellen Vorgehensweisen der einzelnen Methoden wird in Kapitel 3 näher erläutert. Weitere Details zu den konkreten Implementierungen der Identifizierungsmethoden finden sich auf der CD, die dieser Arbeit beigelegt wurde.

Listing 13: Algorithmus zum Vergleich der Daten zweier IDs

```
01 public double calculateMatch(String id1,
                                String id2){
02  Methods methods = getMethods(id1,id2);
03  methods.sort();

04  double match = 0;
05  double weight = 0;
06  double[] m = new double[methods.size()];
07  for(Method method : methods.getList()){
08   if(method.isAvailable()&&method.isActive()){
09    m[method.i()] =  method.compare(id1,id2);

10    if(m[method.i()] == 0){
11     if(method.getFalseNegativeRate() == 0){
12      match = 0;
13      weight = 1;
14      break;
15     }
16    } else if(m[method.i()] == 1){
17     if(method.getFalsePositiveRate() == 0){
18      match = 1;
19      weight = 1;
20      break;
21     }
22    }
23    match = match+
              m[method.i()]*method.getWeight();
24    weight = weight + method.getWeight();
25   }
26  }

27  match = match/weight;

28  for(Method method : methods){
29   if(method.isAvailable()&&method.isActive()){
30    method.adjustWeight(m[method.i()],match);
31    method.update();
32   }
33  }

34  return match;
35 }
```

Das Java-Servlet berechnet aus den Einzelergebnissen der methodenspezifischen Vergleiche ein Gesamtergebnis, wie in Listing 13 dargestellt. Die grundlegenden Ideen, die hinter dieser Implementierung stehen, werden in Kapitel 2.6 erläutert.

Sobald alle Berechnungen des paarweisen Vergleichs abgeschlossen wurden, wird abhängig davon, wie ähnlich sich die Gesamtmengen der Referenzdaten tatsächlich sind, das Gesamtergebnis in einer separaten Datenbanktabelle zwischengespeichert. Es werden nur Paare von Nutzer- bzw. Geräte-IDs abgespeichert, falls das Endresultat einen gewissen Grenzwert erfüllt. Damit wird verhindert, dass zu wirklich jeder möglichen Kombination aus Nutzern oder Geräten das Ergebnis der Identifizierung im Detail gespeichert werden muss, was einen Speicherverbrauch von $O(n^2)$ bedeuten würde. Es werden somit nur netzähnliche Cluster von ähnlichen Nutzer- oder Geräte-Paaren in der Datenbank gespeichert.

Die Speicherung der Ergebnisse hat den Vorteil, dass die Berechnungen nicht jedes Mal neu berechnet werden müssen, wenn ein Android-Framework eine Identifizierungsanfrage stellt. Stattdessen werden die benötigten Daten mithilfe des *DataService* einfach aus der Datenbank geladen, und an das Android-Framework geschickt. Das Android-Framework informiert daraufhin die Anwendung mithilfe des *IdentificationListeners* über das Ergebnis der Identifizierung.

Listing 14: Algorithmus zum Vergleich zweier Mengen

```
01 public double compare(IdentificationSet s1,
                         IdentificationSet s2){
02   double sum = 0, weight = 0;
03   ArrayList<SetItem> union =
              s1.copyList().addAll(s2.copyList());
04   ArrayList<SetItem> intersection =
           s1.copyList().retainAll(s2.copyList());
05   for(SetItem i : union){
06     sum = sum + i.weight();
07   }
08   if(sum>0){
09     for(SetItem i : intersection){
10       weight = weight + i.weight();
11     }
12     return weight/sum;
13   }
14   else return 1;
17 }
```

4.4 Konfiguration und Einbindung des Frameworks

Damit das Framework in eine Android-Anwendung eingebunden werden kann, müssen einige Voraussetzungen erfüllt sein, welche im Folgenden erläutert werden. Bei der Entwicklung des Android-Frameworks wurde allerdings darauf geachtet, dass bei der Einbindung der Implementierung unnötige Arbeit für die Entwickler der mobilen Anwendung entfällt. Auf der beigelegten CD befindet sich zudem eine lauffähige Beispiel-App (Kapitel 5.1) inklusive Quellcode, in der das Framework für die Evaluation testweise integriert wurde und die tatsächliche Verwendung der Bibliothek veranschaulichen kann.

In dem Android-Manifest der Anwendung müssen zum einen der *IdentificationService* sowie der BroadcastReceiver des Frameworks dem System bekannt gemacht werden. Des Weiteren müssen alle *Permissions*, die vom Framework benötigt werden, in das Manifest der Anwendung eingetragen werden. Der Umfang des Frameworks kann zum Teil so angepasst werden, dass nur *Permissions* benötigt werden, die auch von der Anwendung selbst angefragt werden. Da das Framework aber für den Identifizierungsvorgang die Daten logischerweise an das Java-Servlet schicken muss, und bei nicht vorhandener Internetverbindung die Daten abspeichern muss, werden in jedem Fall folgende *Permissions* für die Einbindung des Android-Frameworks benötigt:

```
<uses-permission android:name="android.permission.INTERNET">
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Falls das Android-Manifest entsprechend angepasst wurde, kann das Framework in das Projekt eingebunden und ausgeführt werden. Das Framework wird ab dem ersten Start der Anwendung ausgeführt. Die Anwendung muss dazu dem Framework einige Parameter übergeben, und den *IdentificationService* starten, wie in Listing 15 beispielhaft dargestellt.

Die übergebenen Parameter werden von dem Framework gespeichert, und werden bei einem Neustart des Geräts erneut verwendet. Die Parameter des *IdentificationService* können aber auch nachträglich noch bei Bedarf durch einen Aufruf der in Listing 15 aufgeführten Methoden geändert werden.

Das Android-Framework sorgt in der Standard-Konfiguration automatisch dafür, dass der *IdentificationService* dauerhaft ausgeführt wird. Das Framework wird unmittelbar nach einem Neustart des mobilen Geräts gestartet.

Listing 15: Einbindung des Android-Frameworks

```
01 import com.tum.ident.IdentificationListener;
02 import com.tum.ident.IdentificationService;
03
04 [...]
05
06 private void startIdentificationService(Context context,
                                          String serverURL,
                                          SurfaceHolder surfaceHolder,
                                          MyIdentificationListener listener ){
07     IdentificationService.setServerURL(serverURL);
08     IdentificationService.setSurfaceHolder(surfaceHolder);
09     IdentificationService.setListener(listener);
10     IdentificationService.bind(context);
11 }
```

Um Daten über den Identifizierungsvorgang an sich sowie die Identifizierungsnummer für das mobile Gerät bzw. den aktuellen Nutzer zu erhalten, sollte zudem die *IdentificationListener*-Schnittstelle implementiert und dem *IdentificationService* wie in Listing 15 dargestellt übergeben werden. Die Anwendung kann so über die Ergebnisse von Identifizierungsvorgängen informiert werde, und erhält bei Bedarf Informationen über die gesammelten Referenzdaten. Die internen Komponenten des Android-Frameworks müssen in der Regel von der Android-Anwendung nicht direkt angesprochen werden, so sind die öffentlichen Schnittstellen des *IdentificationService,* wie in Abbildung 26 dargestellt, neben der *IdentificationListener*-Schnittstelle hauptsächlich für die Kommunikation mit dem Android-Framework zuständig.

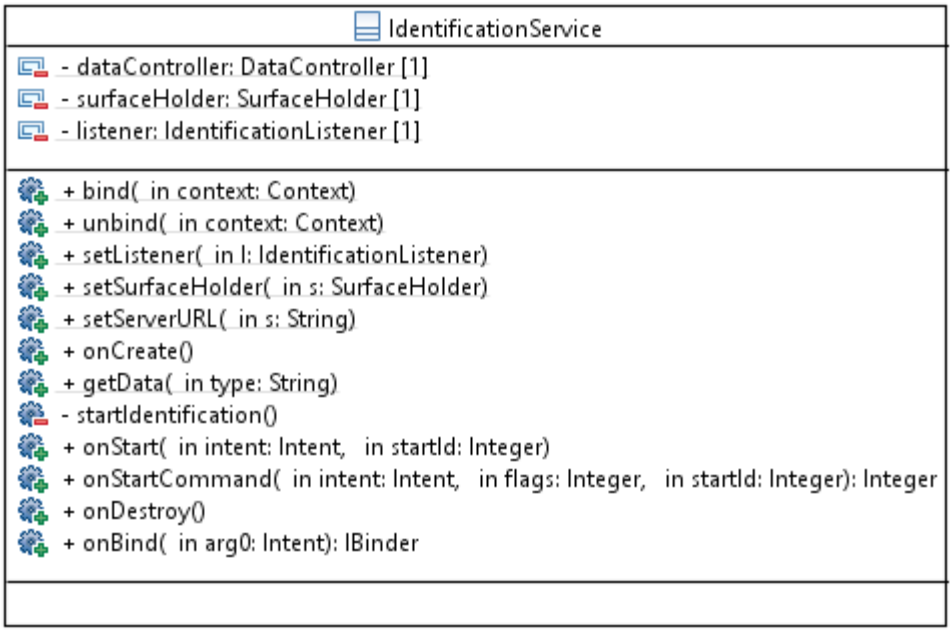


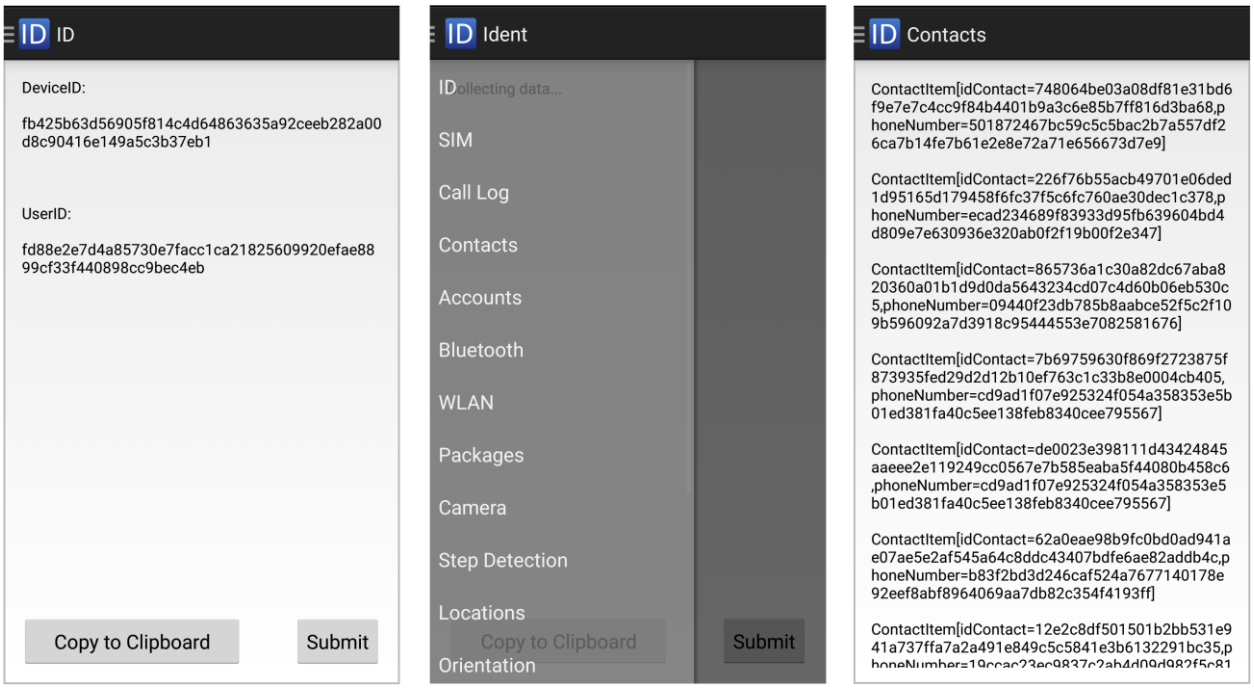
Abbildung 26: Klassendiagramm des IdentificationService

5 Evaluierung des Frameworks

Um die Performanz und die Zuverlässigkeit der Implementierungen zu testen, wurde eine Beispiel-App entwickelt, welche das in Kapitel 4 vorgestellte Android-Framework einbindet. Eine Gruppe von acht Teilnehmern installierte diese Anwendung im Rahmen der Evaluierung auf ihren Smartphones. Innerhalb eines Zeitraums von zwei Wochen konnten Referenzdaten gesammelt und Nutzer- und Geräteidentifizierungen durchgeführt werden. In den nachfolgenden Kapiteln wird die Entwicklung und Auswertung der Beispiel-App beschrieben, und anschließend die essentiellen Ergebnisse der Evaluierung vorgestellt. Es wird zudem auf die Visualisierungen der gesammelten Daten durch die Beispiel-App eingegangen.

5.1 Entwicklung einer Beispiel- App

Die Beispiel-App, die für die Evaluierung des Frameworks entwickelt wurde, muss einige Voraussetzungen erfüllen, damit sie innerhalb des zweiwöchigen Zeitrahmes sinnvoll für die Verbesserung des Frameworks und Auswertung der Endresultate verwendet werden konnte. Zum einen sollte sichergestellt werden, dass das Framework auch wirklich durchgehend auf den mobilen Geräten der Teilnehmer ausgeführt wird. Zudem soll die Beispiel-App in Echtzeit Informationen über den aktuellen Stand der Identifizierung liefern. Um die Korrektheit der implementierten Vorgehensweisen zu prüfen sollten des Weiteren für jede Identifizierungsmethode detaillierte Informationen über die gesammelten Daten innerhalb der Beispiel-App einsehbar sein. Referenzdaten, die nicht ohne weiteres auf ihre Korrektheit überprüft werden können, sollten zudem anschaulich dargestellt werden.



(a) Geräte- und Nutzer-ID (b) Auswahlmenü (c) Pseudonymisierte Daten

Abbildung 27: Screenshots der Beispiel-App

Das Framework wurde wie in Kapitel 4.4 erläutert in die Beispiel-App eingebunden. Die Beispiel-App erhält über einen *IdentificationListener* Benachrichtigungen über den aktuellen Stand der Identifizierung. Wie in Abbildung 27 (a) dargestellt können beispielweise die Geräte- und Nutzer-IDs ohne Zeitverzögerung angezeigt werden, sobald die Werte auf dem Java-Servlet generiert und an die Beispiel-App bzw. das Framework übermittelt wurden. Den Teilnehmern wurde in dieser App-Ansicht zudem eine Möglichkeit angeboten, die Identifikationsnummern in die Zwischenablage des Geräts zu kopieren, um

die Korrektheit der späteren Identifizierung später eindeutig überprüfen zu können.

Die Beispiel-App greift auf die gespeicherten Referenzdaten zurück, um diese den Nutzern zu präsentieren. Sensible Daten werden hierbei, wie in Abbildung 27 (c) dargestellt, umgehend durch das Framework vor jeglicher Verwendung oder Weitergabe pseudonomisiert. Abbildung 27Abbildung 27 (b) zeigt das Auswahlmenü der Beispiel-App, mithilfe dessen die jeweilige Identifizierungsmethoden bzw. die dazugehörigen Referenzdaten ausgewählt werden können. Die Auswahl und Darstellung der Referenzdaten wurde mithilfe von Fragmenten umgesetzt.

Alle Referenzdaten, die in erster Linie physikalische Gegebenheiten beschreiben, werden innerhalb der Beispiel-App visualisiert (Abbildung 28). Die aggregierten Daten konnten somit zeitnah interpretiert, eingeordnet bzw. bewertet werden.

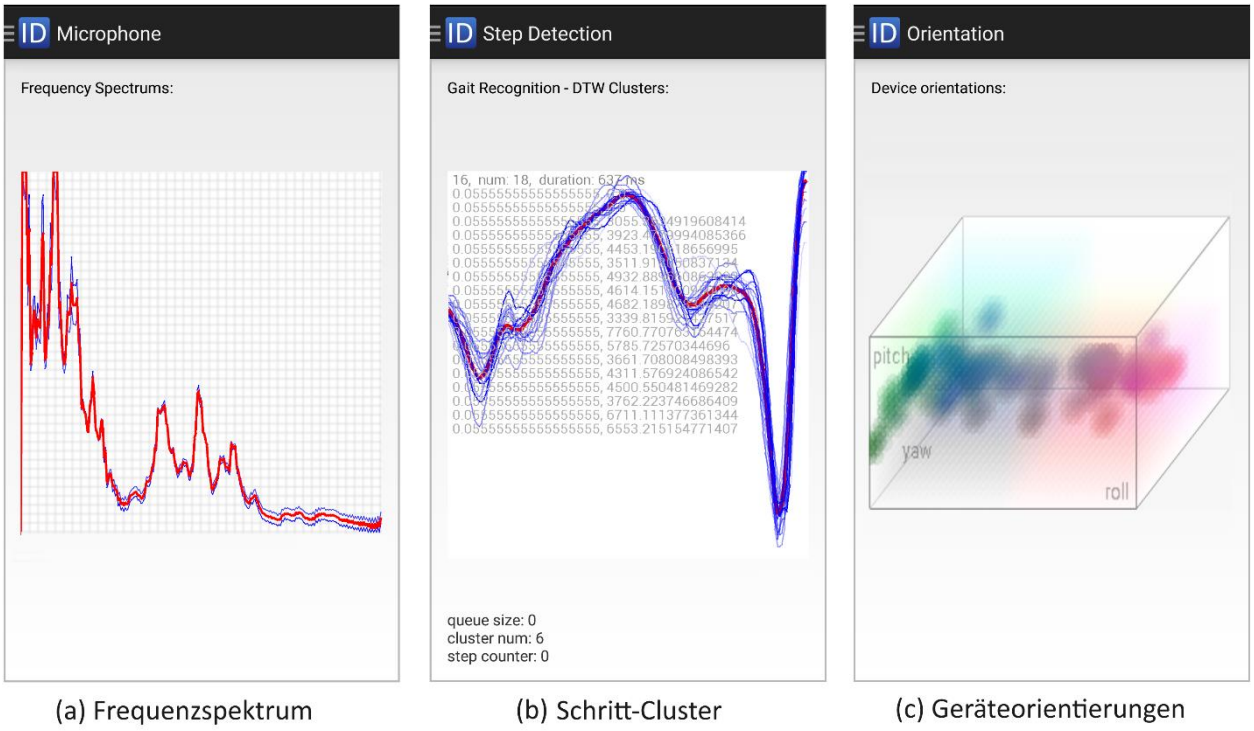


Abbildung 28: Screenshots der Visualisierungen

Die Beispiel-App ist nicht nur in der Lage, bei Bedarf Daten von dem Framework abzufragen, sondern wird zudem umgehend informiert, falls neue Daten gesammelt und angezeigt werden können. Die Teilnehmer der Evaluation konnten so mithilfe der Beispiel-App schnellstmöglich überprüfen, ob für eine Methode korrekte Referenzdaten gesammelt werden konnten. Die Teilnehmergaben konnten so während der Evaluation Rückmeldungen über dem Anschein nach fehlerhafte oder fehlende Daten geben.

Um das Framework möglichst schnell an das Feedback der Studenten anpassen zu können, sowie um auf Fehlermeldungen umgehend reagieren zu können, wurde eine Update-Funktionalität für die Beispiel-App implementiert. Dabei wurden zwei Ansätze gewählt: Zum einen wurde die Beispiel-App den Teilnehmern auf Google Play als Beta-Version angeboten. Laut Google werden Updates automatisch an die Geräte der Teilnehmer verteilt, ohne dass die Beispiel-App manuell auf Updates überprüft werden muss. Bei längerfristigen Testphasen mag die Nutzung dieser Google-Play-Funktionalität durchaus sinnvoll sein, allerdings kann festgestellt werden, dass Google teilweise mehr als 24 Stunden braucht, um das hochgeladene Update den Nutzern verfügbar zu machen. Angesichts der Tatsache, dass die Evaluation über einen Zeitraum von zwei Wochen ablaufen sollte, war die Zeit, die für ein Update über den Google Play - Dienst veranschlagt wurde, zu viel. Es wurde deshalb zusätzlich eine eigene Lösung implementiert,

um das Update-Problem zu lösen. Die Beispiel-App überprüft in gewissen Zeitintervallen selbst, ob eine neue Version zur Verfügung steht. Dazu wird eine Datei, in der die aktuelle Version gespeichert ist, von einem Webserver heruntergeladen und mit der internen Versionsnummer verglichen. Falls die Versionsnummern nicht übereinstimmen, wird ein Notification-Sound abgespielt, und eine Meldung über die neue Version dem Nutzer angezeigt. Die Teilnehmer konnten so neue APK-Dateien umgehend über den mobilen herunterladen und installieren.

Um die Stabilität des Frameworks zu verbessern, wurde die Beispiel-App mit einem Analyse-Tool ausgestattet, welches detaillierte Fehlermeldungen der Anwendung an einen Server übermittelt. Für die Beispiel-App wurde dazu ACRA und Acralyzer (CouchApp[8]) eingesetzt. ACRA wird in mehr als 40 Millionen Android-Anwendungen weltweit eingesetzt. Es handelt sich hierbei um ein quelloffenes System, welches im Rahmen der Beispiel-App-Entwicklung für Crash-Reports zuständig war. Die gesendeten Daten können über ein Web-Interface eingesehen werden, wie in Abbildung 29 dargestellt. [64]

5.2 Verlauf der Evaluation

Eine Gruppe von acht Studenten installierte die Beispiel-App auf ihren Smartphones. Zudem wurde die Beispiel-App auf zehn weiteren Geräten installiert, um das Framework auf Fehler zu überprüfen und weitere Details über die Qualität der Geräteidentifizierung im Speziellen zu sammeln. Es handelte sich hierbei um sechs Smartphones sowie drei Tablets. Auf allen getesteten Geräten lief das Android-Framework stabil. Folgende Modelle kamen für die Untersuchung der Geräteidentifizierung zum Einsatz:

Smartphones

- Nexus One (PB99100), Android-Version 2.3
- Samsung Galaxy W (GT-I8150), Android 2.3
- Galaxy Nexus (GT-I9250), Android-Version 4.4
- Samsung GALAXY S III (GT-I9300), Android-Version 4.4, drei Testgeräte
- Samsung GALAXY S 5 (SM-G900), Android-Versionen 4.4, Update auf Android 5.0

Tablets

- Nexus 7 (ME370T),Android 4.1
- Sony Xperia Tablet S (SGPT121), Android 4.0
- Samsung Galaxy Tab 10.1N (GT-P7511), Android 4.0

Ein Teil Android-Versionen bzw. Gerätespezifikationen, die von den Teilnehmern der Evaluation verwendet wurden, sind durch die Fehlermeldungen des ARCA-Frameworks nebenbei erfasst worden. Da aber die entsprechenden Referenzdaten der Teilnehmer durch das Framework grundsätzlich pseudonomisiert wurden, sind basierend auf den Daten-Einträgen in der MySQL-Tabelle keine weiteren Rückschlüsse auf bestimmte Hardware-Details möglich. In der Datenbank stehen diesbezüglich nur eine Reihe unterscheidbarer Hash-Werte. Die einzige Information, die im Sinne der Evaluation den Teilnehmern explizit und auch außerhalb des Frameworks zugeordnet wurde, waren die individuellen Identifikationsnummern, die manuell aus der Beispiel-App vor und nach der Analysephase von den Teilnehmern kopiert werden sollten (siehe Kapitel 5.1, Abbildung 27 (a)).

Obwohl das Framework auf dem mobilen Gerät, mit dem während der Implementierungsphase hauptsächlich entwickelt und getestet wurde, vor der Durchführung der Evaluation ohne jegliche

[8] CouchApp: Ein Web-Application-Framework für CouchDB (Datenbankmanagementsystem), auf Javascript basierend

Abstürze bzw. Probleme funktionierte, registrierte ACRA in den ersten Tagen während der Datenerfassungsphase einige Fehlermeldungen von unterschiedlichen Android-Versionen und Geräten, wie in Abbildung 29 dargestellt. Die Fehler sind zum Teil auf den unterschiedlichen Funktionsumfang der Android-Versionen zurückzuführen, oder wurden beispielsweise durch inkompatible, externe Java-Bibliotheken verursacht. Die Anzahl der Fehlermeldungen konnte allerdings in den ersten Tagen schon deutlich reduziert werden. Nach einer Woche, also zu Beginn der Analysephase wurde von ACRA bereits keine einzige Fehlermeldung mehr registriert. Es kann also davon ausgegangen werden, dass das Framework auf den zur Verfügung stehenden Geräten und Software-Versionen ohne Einschränkungen stabil läuft.

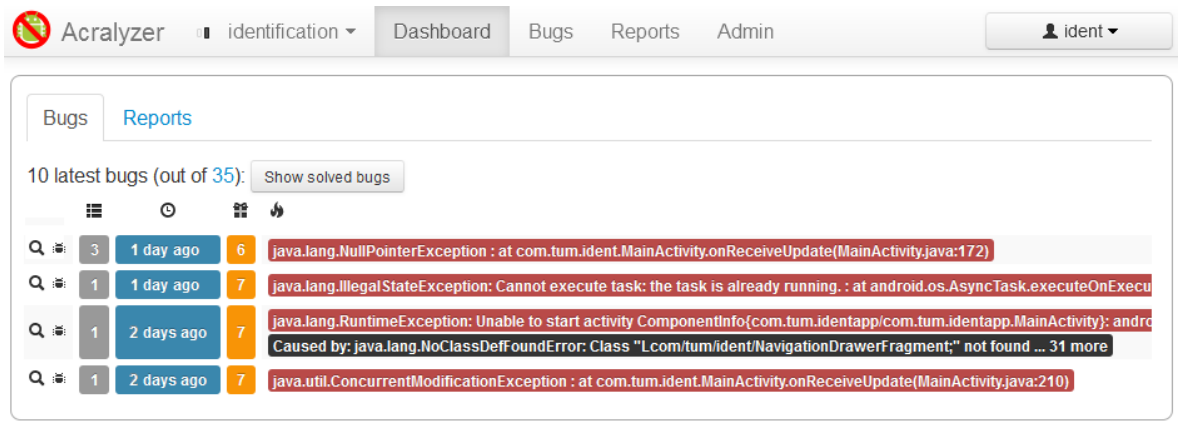


Abbildung 29: Web-Interface des Acralyzers [65]

Neben den Fehlermeldungen, die von ARCA aufgezeichnet und übermittelt werden konnten, gab es zu Beginn der Evaluation einige gerätespezifische Probleme. Ein Problem, das im Laufe der Evaluation gelöst werden musste, war, dass bestimmte Smartphones nur mithilfe von relativ aufwendigen Workarounds auf bestimmte Sensoren zugreifen konnte, sobald der Sperrbildschirm des mobilen Geräts aktiviert war. So zählt der ab Android KitKat verfügbaren Schritt-Erkennungssensor zwar weiterhin mit, falls der Sperrbildschirm aktiviert wird, Android-Anwendungen werden aber erst bei erneuter Aktivierung des Bildschirms über die Anzahl gegangener Schritte informiert. Für die Untersuchung der Schritte in Echtzeit kann der Schritt-Erkennungssensor deshalb nicht ohne Einschränkungen eingesetzt werden. Als Alternative wurde deshalb eine eigene Klasse für die Schritterkennung auf Software-Ebene implementiert, wodurch auch die Kompatibilität der entsprechenden Implementierung mit älteren Android-Versionen sichergestellt werden konnte.

Des Weiteren bleibt anzumerken, dass an der Evaluation des Frameworks Zwillinge als Nutzer er Beispiel-App teilgenommen haben. Beide Teilnehmer wohnen zudem im selben Haus, und benutzen während der Evaluationsphase baugleiche Android-Smartphones. Der Freundeskreis und die freizeitlichen Aktivitäten zeigen zudem große Übereinstimmungen. Diese Bedingungen können somit als mögliches Worst-Case-Szenario für die eindeutige Identifizierung von Nutzern und Geräten durch das Framework betrachtet werden, weshalb die Performanz des Frameworks im Umgang mit diesen beiden Nutzern besondere Beachtung fand. Es konnte trotz der Ähnlichkeiten der Nutzer und Geräte genug unterscheidbare Merkmale gesammelt werden, um durch der Kombination der aggregierten Informationen innerhalb der zwei Wochen spezifische Nutzerprofile zu erstellen.

Nachdem die anfänglichen Probleme gelöst werden konnten, die durch die unterschiedlichen Konfigurationen und Eigenschaften der Geräte verursacht wurden und ARCA keine neuen Fehlermeldungen mehr von den Geräten der Teilnehmer verzeichnete, konnte nach und nach beobachtet werden, wie durch das Java-Servlets immer mehr Informationen in der MySQL-Datenbank abgelegt

wurden. Die initiale Datenerfassungsphase konnte somit ab diesem Moment für alle verfügbaren Smartphones als vollständig gestartet betrachtet werden.

Der generellen Ablauf der Evaluation kann in zwei zeitlich getrennte Abschnitte unterteilt werden: Zu Beginn der Evaluationsphase wurden neben der Fehlerbehebung in erster Linie Referenzdaten für die spätere Identifizierung gesammelt. Diese initiale Datenerfassungsphase dauerte insgesamt eine Woche. Es hat sich gezeigt, dass ein Teil der Methoden wie erwartet sofort nach Installation bzw. Start der Anwendung Referenzdaten das Java-Servlet geschickt hat. Verfahren, welche die Verhaltensweisen oder andere biometrische Faktoren untersuchten brauchten erwartungsgemäß länger, bis unterscheidbare Merkmalsvektoren extrahiert werden konnten.

Zu Beginn der darauffolgenden Woche deinstallierten die Studenten die Beispiel-App, und entfernten alle Anwendungsdaten von ihren Smartphones. Danach wurde die Beispiel-App erneut installiert. Das Framework begann daraufhin erneut Referenzdaten zu sammeln. Die neu gewonnenen Informationen wurden wie geplant mit den Daten, die bereits in der MySQL-Datenbank gespeichert waren, abgeglichen. Für jedes Gerät und jeden Nutzer konnte so eine Datenreihe angelegt werden, in der die Ähnlichkeiten zwischen Geräten und Nutzern gespeichert wurden.

Es bleibt festzuhalten dass am Ende der einwöchigen Datenerfassungsphase zu allen in dieser Arbeit vorgestellten Methoden Referenzdaten gesammelt werden und die entsprechenden Identifizierungsmethoden durchgeführt werden konnten. Im nachfolgenden Kapitel werden die Erkenntnisse, die während der Evaluation gewonnen werden konnten, im Detail vorgestellt. Mithilfe der Fehlermeldungen, die durch das ACRA-Framework erfasst werden konnten, konnte das Framework soweit optimiert und angepasst werden, dass auf allen Testgeräten das Framework eingesetzt werden kann (Android 2.2 - Android 5.0) . Das Framework bzw. die implementierten Methoden liefen während der gesamten Analysephase auf den für Evaluation bereitgestellten Smartphones stabil.

5.3 Ergebnisse der Evaluation

Die Resultate der Analysephase, sowie die Erkenntnisse, die durch die Untersuchung der erfassten Referenzdaten gewonnen werden konnten, werden in diesem Kapitel näher erläutert. Wie in Kapitel 3 dargestellt, setzt das Framework auf eine ganze Reihe an Informationsquellen, um eine möglichst breit gefächerte Menge an unterscheidbaren Merkmalen für die Identifizierung von Nutzern und Geräten zu erfassen. Neben den Daten, die direkt und ohne zeitliche Verzögerung von den Geräten extrahiert werden konnten, wurde eine auch Merkmalsvektoren gesammelt, die beispielsweise auf Sensorwerten basieren und für sich physikalische Entsprechungen in der Umgebung des mobilen Geräts finden lassen. Diese Art von Daten wurden mithilfe von Visualisierungen innerhalb der Beispiel-App veranschaulich.

Sensor-,Positions- und Kameradaten

Wie aus Abbildung 30 ersichtlich, war das Framework in der Lage, die Signalwerte der Beschleunigungssensoren für die Analyse von Schrittfolgen bzw. des spezifischen Gangs (Gait Recognition) der Nutzer zu verwendet, wie in Kapitel 3.1.11 erläutert.

Durch die Visualisierung der gesammelten Daten wurde ersichtlich, dass die jeweiligen Schritte nicht nur erkannt, sondern die Sensorwerte auch präzise in zeitlich korrekte Abschnitte unterteilt wurden. Um die Korrektheit der Schritt-Erkennung zu überprüfen, wurden neben der Visualisierung der Referenzdaten akustische Rückmeldungen integriert. Jedes Mal, wenn das Framework einen Schritt erkannt hat, wurde bei Bedarf während der Evaluation ein kurzer Ton abgespielt. So konnte zeitnah überprüft werden, ob die Schritterkennung zuverlässige Ergebnisse liefert, selbst wenn sich das Smartphone in der Hosentasche befand.

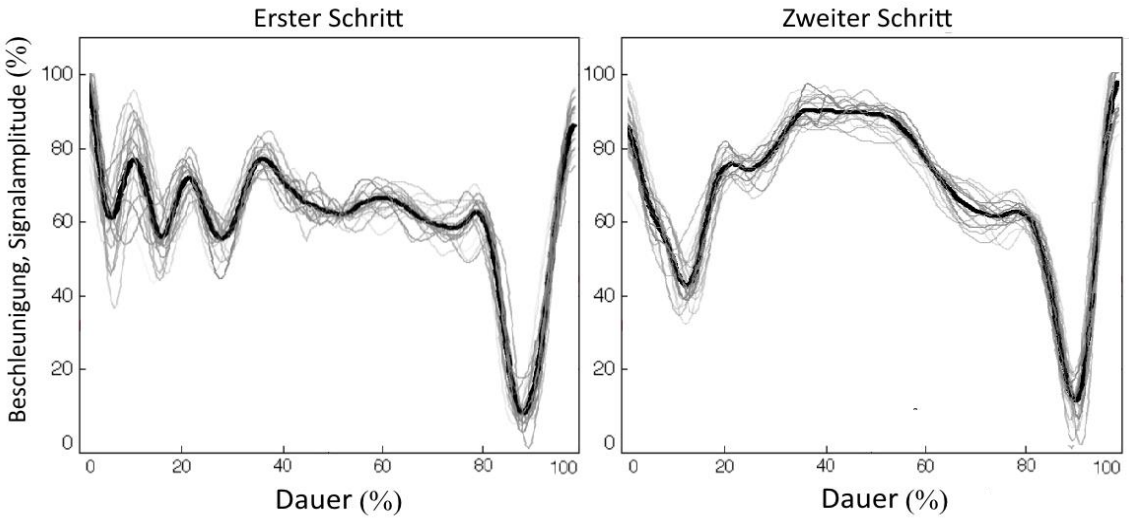


Abbildung 30: Gemittelte Beschleunigungswerte der Schritte eines Nutzers

Die aufgezeichneten Schritte zeigen im Allgemeinen unterscheidbare, nutzerspezifische Charakteristika. Allerdings konnte beobachtet werden, dass für einige Nutzer eine relativ große Datenmenge für diese Methode gesammelt werden musste. Das lag in der Regel daran, dass die betroffenen Nutzer ihr Smartphone immer in unterschiedlichen Taschen am Körper getragen haben. Für jede dieser Taschen musste ein neuer Schritt-Cluster erstellt werden, was den Speicherbedarf der Methode entsprechend erhöhte.

Informationen über die Orte, an denen sich die Nutzer während der Evaluation aufgehalten haben, wurden vom Android-Framework gesammelt und im Zuge der Nutzeridentifikation ausgewertet. Die spezifischen Kombinationen der gesammelten Positionsdaten stellten in der Regel einzigartige Merkmale dar, anhand derer sich die einzelnen Nutzer voneinander unterscheiden ließen.

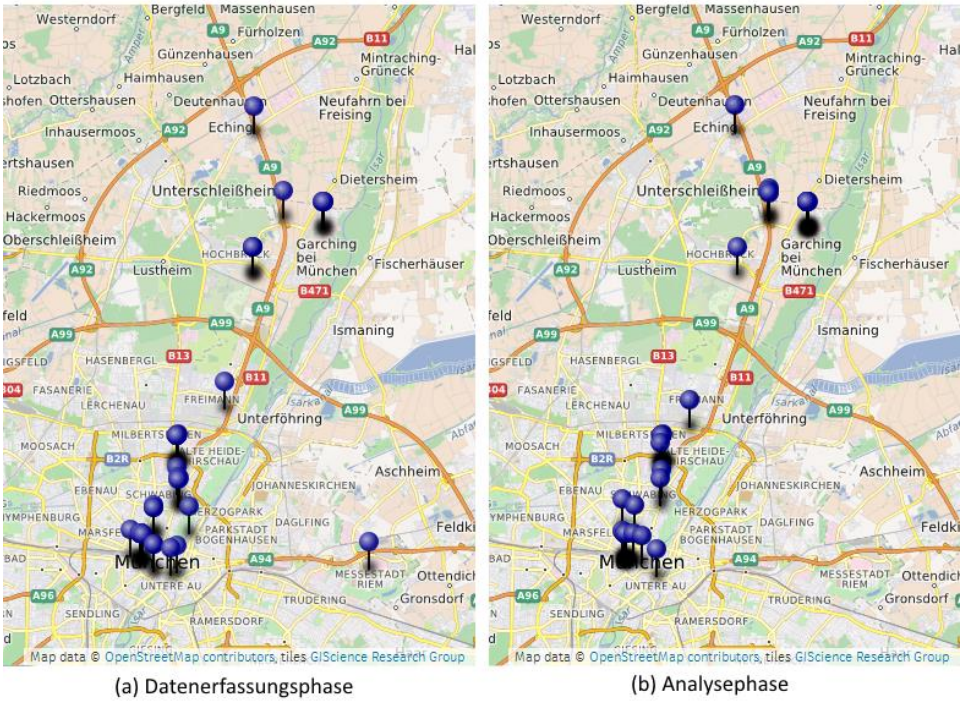


Abbildung 31: Vergleich der aggregierte Positionsdaten eines Nutzers

Die Daten wurden unabhängig vom Gerät gesammelt, es kann davon ausgegangen werden, dass die Informationen, die durch die aggregierten Referenzdaten repräsentiert werden, auch bei einem

möglichen Smartphone-Wechsel relativ konstant geblieben wären. Abbildung 31 zeigt die Positionsdaten einer Nutzer, die während der Datenerfassungsphase gesammelt wurden im Vergleich zu den darauf erfassten Referenzdaten. Es ist ersichtlich, dass sich die Charakteristika der Verteilung der Positionsdaten nicht wesentlichen geändert haben. In Kapitel 3.1.12 wird näher erläutert, wie mithilfe der gesammelten Daten die Identifizierungsmethode durchgeführt wurde.

Eine weitere Methode, deren gesammelte Referenzdaten innerhalb der Beispiel-App visualisiert wurden, setzt sich mit charakteristischen Geräte-Orientierungen auseinander, die im Alltag eines Nutzers regelmäßig auftreten können, wie in Kapitel 3.1.13. Die während der Evaluation gesammelten Referenzdaten zeigen, dass die Geräte-Ausrichtungen direkt mit individuellen Gewohnheiten zusammenhängen. So kann in Abbildung 32 erkannt werden, dass beispielsweise Nutzer 3 anscheinend feste Plätze für die Aufbewahrung seines Smartphones hat. Es wäre denkbardass der Nutzer beispielweise Docking-Stationen nutzt. Nutzer 2 und insbesondere Nutzer 3 weisen hingegen eine deutlich größere Menge an Geräte-Orientierungen vor. Die entscheidende Frage, die sich vor allen in diesen Fällen bei der Evaluation stellt, ist, ob die Geräte-Orientierung zumindest im Mittel einigermaßen konstant bleibt. Wenn man Reihe 1 und Reihe 2 der aggregierten Daten vergleicht, fällt auf, dass die Verteilungen der drei Raumwinkel für jeden User im Kern eine hohe Ähnlichkeit ausweist. Sich wiederholende, benutzerspezifische Verhaltensweisen erwiesen sich während der Evaluationsphase als wertvolle Informationsquellen. Die Werte, die für die paarweise für die Identifizierung der Nutzer durch das Framework berechnet wurden, lassen einen Rückschluss auf bereits bekannte Nutzer zu.

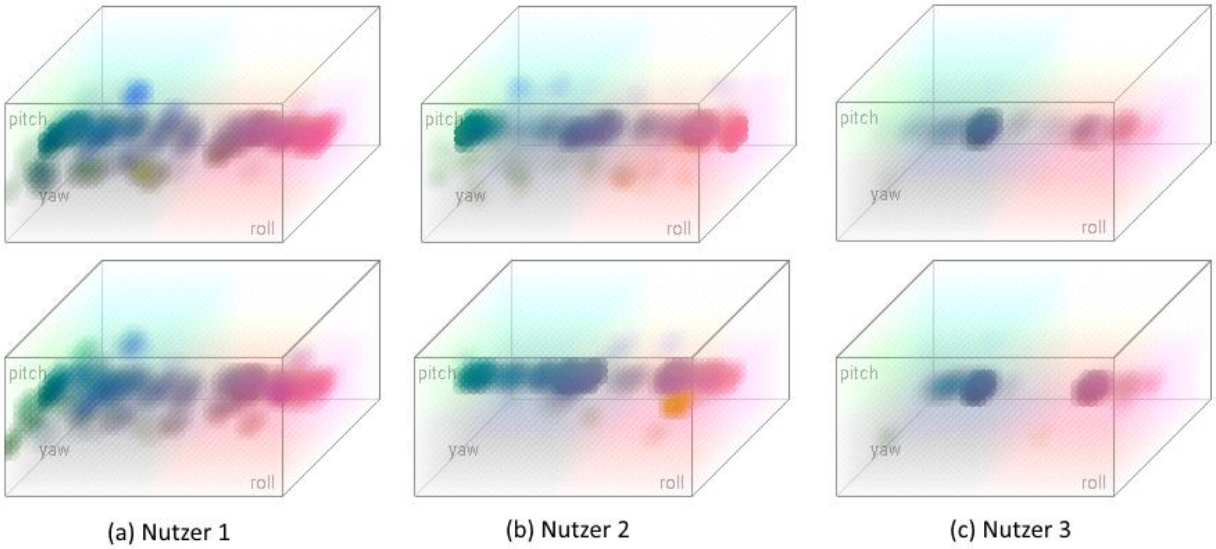


Abbildung 32: Vergleich der aggregierten Geräte-Orientierungen dreier Nutzer

Die Identifizierungsmethoden, welche spezifische Informationen über die eingebauten Kamera extrahiert und analysieren haben, waren nach Abschluss der Analysephase in der Lage, die Geräte anhand der aggregierten Referenzdaten zu unterscheiden. Abbildung 33 veranschaulicht die individuellen Eigenschaften des spezifischen Bildrauschens, was bei drei baugleichen Geräten auftritt. Mit bloßen Auge ist bei dieser Darstellung der Referenzdaten festzustellen, dass sich die eingebauten Kameras und somit die Geräte selbst anhand der Dark-Frame-Analyse mit hoher Wahrscheinlichkeit unterscheiden lassen. Die wahrgenommenen Ähnlichkeiten bzw. Unterscheide bei der Analyse der Dark-Frame-Bilder zeichnete sich auch in den errechneten Werten der entsprechenden Identifizierungsmethode aus. So wurden Dark-Frame-Bilder, die vom selben Gerät stammten, tendenziell eine höhere Ähnlichkeit attestiert als Bilder die mit unterschiedlichen Geräten aufgenommen wurde. Bemerkenswert ist hierbei, dass das Framework sogar bei baugleichen Modellen eine Unterscheidung anhand der Dark-Frame-Bilder zulässt. Das charakteristische Bildrauschen eines Kamerasensors kann mit den physikalischen Eigenschaften der Kamera-Hardware erklärt werden. Das Bildrauschen wird durch spezifische Eigenschaften des

Ausleseverstärkers und die Auswirkungen von Dunkelstrom auf lichtempfindliche Elemente verursacht. Während der Evaluation blieben die Eigenschaften der Dark-Frames auch über längere Zeitspannen konstant. Allerdings kann davon ausgegangen werden das sehr hohen Temperaturschwankungen zumindest in der Theorie Auswirkungen auf die Intensität des feststellbaren Dunkelstroms haben. Die charakteristische, relative Verteilung des Rauschens sollte sich dadurch aber nicht wesentlich ändern, jedoch die absolute Intensität des Rauschen. Da die Farbwerte der Dark-Frames vor der Analyse normiert werden, sollte dieser Faktur nur geringen Einfluss auf das Resultat der Identifizierungsmethode haben, was sich auch während der Evaluation gezeigt hat. Um kleinere Änderungen auszugleichen werden zudem gemittelte Farbwerte aus mehreren Dark-Frame- Bildern berechnet, und die charakteristischen Merkmale extrahiert.

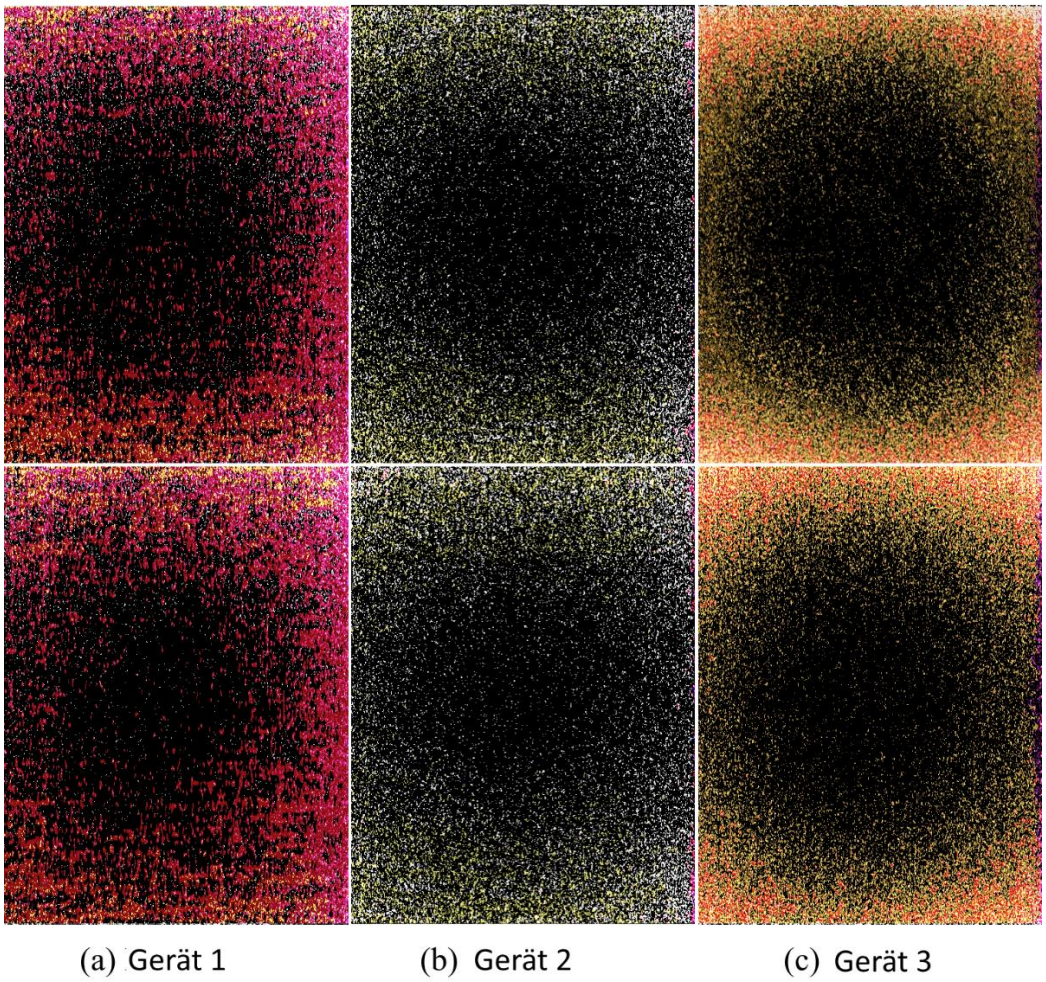


Abbildung 33: Charakteristisches Bildrauschen dreier baugleicher Android-Geräte

Um die Zuverlässigkeit der Pixelfehler-Erkennung zu testen, wurde das Framework zudem mit passenden Bilddaten getestet. Die fehlerhaften Pixel, also sowohl Dead- also auch Hot-Pixel, wurden von der Identifizierungsmethode erkannt und indexiert. Die Positionen den fehlerhaften Pixel waren bei allen getesteten Bildern korrekt. Die gesammelten Referenzdaten charakterisierten somit die entsprechenden Kameras, es konnten nahezu einmalige, unveränderliche Merkmale extrahiert werden.

Pseudonomisierte Nutzer- und Geräte-Daten

Neben den bereits vorgestellten Daten wurde im Laufe der Evaluation auch eine Reihe an Informationen gesammelt, die sich aufgrund der Pseudonomisierung durch das Framework nicht durch plausible Visualisierungen wie zu Beginn des Kapitel beschrieben darstellen lassen. Die pseudonomisierten Daten

können beispielsweise unterscheidbare Einzelwerte wie beispielsweise Seriennummern oder Hardware-Eigenschaften repräsentieren, oder für spezifischen Elemente einer Liste oder Menge stehen (wie beispielsweise die Kontaktdaten, Anruferliste sowie die WLAN- und Bluetooth-Informationen). Typisch für diese Art von Daten ist, dass die benötigten Informationen in den meisten Fällen unmittelbar nach der Installation des Frameworks zur Verfügung gestanden sind. Erste Einschätzungen für die Nutzer- und Geräteidentifizierungen sind somit auch ohne längere Datenerfassungsphase möglich. Die während der initialen Datenerfassungsphase gesammelten Werte gegeben, welche durch das Framework vollständig pseudonomisiert wurden, änderten sich im Verlauf der Evaluation nur wenig. Dies hat zur Folge, dass die Nutzer und Geräte anhand dieser Daten ohne Probleme in der Analysephase wiedererkannt bzw. identifiziert werden konnten. Es ist somit anzunehmen, dass auch bei längeren Zeitspannen ein Großteil der pseudonomisierten Daten, die einem Nutzer oder Gerät zugeordnet werden können, unverändert bleiben, wodurch die Identifizierung anhand dieser Daten auch längerfristig möglich sein sollte. Das Framework reagiert zudem auf Änderungen der pseudonomisierten Referenzdaten, und aktualisiert diese bei Bedarf.

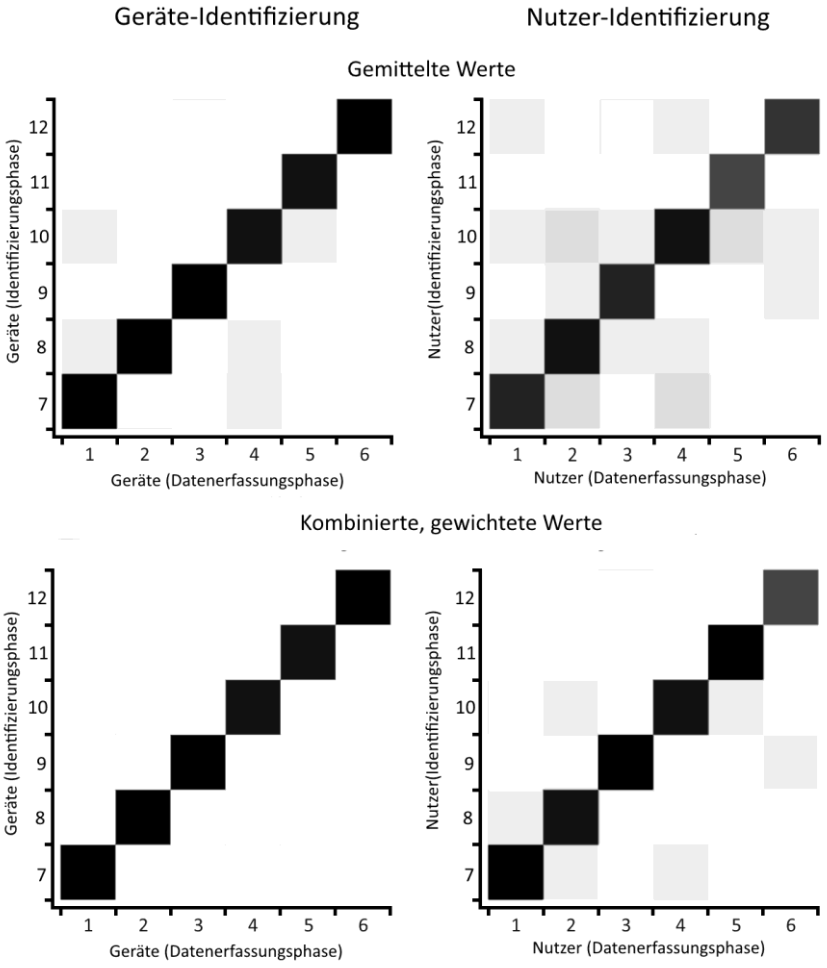


Abbildung 34: Ergebnisse der Identifikation im Vergleich

Mithilfe der gewichteten, kombinierten Rückgabewerte der aller Identifizierungsmethoden konnten die Teilnehmer der Evaluation im Laufe der Analysephase identifiziert werden, wie in Abbildung 34 dargestellt. Die Gewichtung der Identifizierungsmethoden hat sich im direkten Vergleich mit der gemittelten Kombination von Identifizierungsmethoden während des Identifikationsprozesses als geeigneter herausgestellt

Das Framework war somit schon direkt nach der anfänglichen Debugging-Phase für die Identifizierung von Nutzern und Geräten im Rahmen der Evaluation einsatzbereit. Spezifische Parameter der einzelnen Identifizierungsmethoden wurden während der Evaluation so angepasst, dass mit der Zeit bessere Gesamtergebnisse geliefert werden konnten.

6 Zusammenfassung und Ausblick

Die Zielsetzung der Arbeit war die Analyse von Vorgehensweisen für die Identifizierung von Benutzern und Geräten, sowie die Integration der entsprechenden Identifizierungsmethoden in ein Android-Framework. Es wurden mehrere Identifikationsverfahren kombiniert, um die Zuverlässigkeit sowie die Genauigkeit des Frameworks als Ganzes zu optimieren. Im Verlauf der Evaluation hat sich gezeigt, dass die kombinierten, gewichteten Ergebnisse des implementierten Identifikationsverfahrens in der Regel bessere Ergebnisse liefern als einzelne, isoliert betrachtete Methoden, welche zum Teil nur auf einer einzigen Informationsquelle beruhen. Das Framework ist in der Lage, die Gewichtung und Einbindung der einzelnen Methoden der aktuellen Situation anzupassen. So kann sich das Framework mit der Zeit auf eine spezifische Gruppe von Nutzern einstellen, indem Merkmale, die in einem gegebenen Szenario bessere Unterscheidungskriterien bieten, stärker gewichtet werden. Außerdem ist das Framework in der Lage, auf Situationen, in denen bestimmte Methoden auf einem Gerät nicht verfügbar sind, zu reagieren. Dem Framework steht ein breites Spektrum an möglichst unabhängigen Methoden zur Verfügung. Dies ist ein Grund weshalb, selbst wenn nicht alle Methoden eingesetzt werden können, vernünftige Resultate geliefert werden. Das Framework aktualisiert die gesammelten Referenzdaten in regelmäßigen Abständen. Die Genauigkeit der Identifizierung von Nutzern und Geräten kann so selbst bei Änderungen der Informationsquellen über längere Zeiträume sichergestellt werden. Bei der Kombination der Ergebnisse wurde darauf geachtet, dass die spezifischen Identifizierungsmethoden sich hinsichtlich der jeweiligen Fehlerraten (Falsch-Positiv-Rate, Falsch-Negativ-Rate) gegenseitig korrigieren und somit verbessern können. Während der Evaluationsphase konnten die Eigenschaften und Anforderungen des Frameworks besser an die praktischen Einsatzmöglichkeiten angepasst werden. So läuft das Framework mittlerweile auf allen Testgeräten unabhängig von der Android-Version oder den Hardware-Eigenschaften stabil und die Identifizierungsmethoden liefern auf allen getesteten Geräten nachvollziehbare Ergebnisse.

Bei der Entwicklung des Android-Frameworks wurde darauf geachtet, dass die Implementierung des Gesamtsystems möglichst flexibel gestaltet wird. Die Anbindung des Frameworks an andere Projekte ist dadurch ohne größere Einschränkungen möglich. Die *Android-Permissions*, die von dem Framework insgesamt benötigt werden, lassen sich an die Anforderungen der Projekte anpassen. Das Framework stellt zudem Schnittstellen bereit, mithilfe derer auf die aggregierten Daten sowie Identifizierungsergebnisse zur Weiterverarbeitung zugegriffen werden kann. Die Architektur des Android-Frameworks wurde zudem flexibel gestaltet und kann so um weitere Komponenten, wie beispielsweise zusätzliche Identifizierungsmethoden, erweitert werden.

Es existiert eine Reihe von Arbeiten, die sich mit verwandte Themengebieten auseinandersetzen. So konzentrieren sich beispielsweise einige Forscher auf die Identifizierung von Nutzer durch die Analyse des Verhaltens im Zusammenspiel mit Touch-Screen-Geräten [66]. Andere Arbeiten versuchen, die Verhaltensweisen von Benutzer in Computerspielen zu analysieren, um diese dadurch identifizierbar zu machen [67]. Einige Studien verfolgen zudem den Ansatz, die aktuelle Stimmung der Nutzer durch die Untersuchung des Kommunikationsverlauf oder Audio-Inputs zusätzlich zu erfassen. Es existieren zudem Arbeiten, die sich mit der Unterscheidung von Nutzern anhand einer großen Auswahl an spezifischen Verhaltensweisen und körperlichen Aktivitäten, sogenannter Behavioural Biometrics, mithilfe von Sensordaten beschäftigen. So wurde unter anderem gezeigt, dass sich die Daten, die von den Beschleunigungssensoren und Gyroskopen auf mobilen Geräten erfasst werden können, beispielsweise auch für angepasste Spracherkennungssysteme eignen würden [68] [69]. Die Analyse des audiovisuellen Inputs, welcher direkt von dem Nutzer der mobilen Geräte erzeugt wird, wäre ein sinnvoller Ansatz für eine zukünftige Weiterentwicklung des Frameworks. [70]

Angelehnt an die Ergebnisse der Studien ist ein breites Spektrum zukünftiger Funktionalitäten und Einsatzgebiete für das in dieser Arbeit entwickelten Android-Framework zur Identifizierung von Nutzern und Geräten denkbar.

Abkürzungsverzeichnis

ADN	Abbreviated Dialing Numbers
API	Application Programming Interfaces
ART	Android Runtime
CGI	Common Gateway Interface
CPU	Central Processing Unit
DTW	Dynamic Time Warping
GPS	Global Positioning System
GSF	Google Service Framework
HTTP	Hypertext Transfer Protocol
ICCID	Integrated Circuit Card Identifier
IDE	Integrierte Entwicklungsumgebung
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
JDBC	Beispiel Java Database Connectivity
JFIF	JPEG File Interchange Format
JNI	Java Native Interface
JPEG	Joint Photographic Experts Group
MAC	Media Access Control
MNO	Mobile Network Operator
NDK	Android Native Development Kit
NNG	Nearest Neighbor Graph
PaaS	Platform as a Service
SD	Secure Digital
SMS	Short Message Service
SSL	Secure Socket Layer
URI	Uniform Resource Identifier

VM	Virtual Machine
WLAN	Wireless Local Area Networks

Glossar

Im Folgenden werden die wichtigsten Fachbegriffe im Kontext dieser Arbeit erläutert.

Activity	Durch eine Activity kann das Verhalten und der Aufbau einer vom Benutzer bedienbaren, grafischen Oberfläche beschrieben werden.
Beschleunigungs-sensor	Sensoreinheit, welche die relative Beschleunigung in X-,Y- und Z-Richtung misst.
Broadcast-Receiver	Durch Broadcast-Receivers kann auf Meldungen des Systems reagiert werden, sowie zeitgesteuert Prozesse ausgelöst werden. Eine App kann zudem auch selbst Broadcasts starten, um systemweit andere Apps über wichtige Ereignisse zu informieren.
Content-Provider	Mithilfe des Content-Providers können die Daten einer App verwaltet werden. Der Content-Provider dient als Schnittstelle, über die Daten für andere Anwendungen bereitgestellt, oder sogar von diesen modifiziert werden können.
Dead Pixel	Ein Pixel des Kamerasensors, welcher fehlerhafte, meist konstante Werte liefert.
Dynamic Time Warping	Verfahren, welches Wertefolgen unterschiedlicher Länge aufeinander abbildet und abgleicht. DTW kann als Distanzmaß eingesetzt werden.
Falsch-Positiv-Rate	Anteil der fälschlicherweise als zu einer Klasse gehörend bewerteten Objekte; Eigenschaft eines Klassifikators
Falsch-Negativ-Rate	Anteil der fälschlicherweise als nicht zu einer Klasse gehörend bewerteten Objekte; Eigenschaft eines Klassifikators.
Fragment	Teilbaustein einer Activity innerhalb einer Android-Anwendung.
Frequenzspektrum	Zusammensetzung des Signals aus kontinuierlichen Frequenzen.
Hot Pixel	Ein Pixel des Kamerasensor welche vor allem bei langen Belichtungszeiten und hohen ISO-Werten falsche Werte liefert.
Laufzeitumgebung	Die zur Laufzeit verfügbaren bzw. festgelegten Voraussetzungen eines Systems, unter denen Anwendungen ausgeführt werden können.
Lichtsensor	Der Lichtsensor liefert einen Messwert über die Stärke der einfallenden Strahlung zurück.
Magnetometer	Sensor zur Messung magnetischer Flussdichten.
Merkmalsvektor	Merkmale eines Musters in vektorieller Weise zusammengefasst.
Quadtree	Speichersparende, baumartige Datenstruktur, welche hauptsächlich für die Organisation zweidimensionaler Daten eingesetzt wird. Jeder Knoten eines Quadtrees repräsentiert eine rechteckige, zweidimensionale Fläche, welche durch Kind-Knoten rekursiv in kleinere Flächen unterteilt werden kann. Die inneren Knoten eines Quadtrees haben genau vier Kinder.
Service	Ein Service stellt im Gegensatz zur Activity keine Benutzeroberfläche zur Verfügung, sondert läuft als Dienst im Hintergrund, und führt Programmabschnitte aus, die keine Interaktion mit dem Nutzer erfordern. Activities können Services starten, und mit ihnen interagieren.
Triangulation	Eine indirekte Abstandsmessung durch Winkelmessung innerhalb von Dreiecken, welche der Positionsbestimmung dienen kann.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] Statista. *Anzahl der Smartphone-Nutzer in Deutschland in den Jahren 2009 bis 2014 (in Millionen)*. http://de.statista.com/statistik/daten/studie/198959/umfrage/anzahl-der-smartphonenutzer-in-deutschland-seit-2010/. Zugriff am 03.01.2015.

[2] Statista. *Smartphones - Nutzer weltweit 2012-2018 | Prognose*. http://de.statista.com/statistik/daten/studie/309656/umfrage/prognose-zur-anzahl-der-smartphone-nutzer-weltweit/. Zugriff am 05.01.2015.

[3] Evans Data Corporation. *Global Developer Population and Demographic Study 2014 v. 1*. http://evansdata.com/reports/viewRelease.php?reportID=9. Zugriff am 28.01.2015.

[4] Strategy Analytics. 2014. *Android Captured Record 85 Percent Share of Global Smartphone Shipments in Q2 2014*. http://blogs.strategyanalytics.com/WSS/post/2014/07/30/Android-Captured-Record-85-Percent-Share-of-Global-Smartphone-Shipments-in-Q2-2014.aspx. Zugriff am 01.01.2015.

[5] Lopez, C. Trends der Smartphone Nutzung in Deutschland.

[6] Vanessa Pegueros. 2012. *Security of Mobile Banking and Payments*.

[7] Smith, D. 2015. *Android Still Has A Massive Piracy Problem*. http://uk.businessinsider.com/android-piracy-problem-2015-1?r=US. Zugriff am 26.01.2015.

[8] 2014. Nils T. Kannengiesser et al., “Secure Copy Protection For Mobile Apps“, AmiEs 2014, Symposium, Aveiro.

[9] Esteban Pellegrino. *DoubleDirect - Zimperium Discovers Full-Duplex ICMP Redirect Attacks in the Wild*. http://blog.zimperium.com/doubledirect-zimperium-discovers-full-duplex-icmp-redirect-attacks-in-the-wild/. Zugriff am 25.11.2014.

[10] Arxan. State of Mobile App Security - Apps Under Attack.

[11] Consumer Reports News. *Smart phone thefts rose to 3.1 million last year*. http://www.consumerreports.org/cro/news/2014/04/smart-phone-thefts-rose-to-3-1-million-last-year/index.htm. Zugriff am 17.12.2014.

[12] Kevin Curran, Andrew Robinson, Stephen Peacocke, Sean Cassidy. 2010. Mobile Phone Forensic Analysis. *International Journal of Digital Crime and Forensic,* Vol. 2, No. 2.

[13] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. 2011. *Recommender Systems Handbook*. Springer US, Boston, MA.

[14] Ben Elgin. *Google Buys Android for Its Mobile Arsenal*. http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal. Zugriff am 07.12.2014.

[15] 2012. *Alliance Members | Open Handset Alliance*. http://www.openhandsetalliance.com/oha_members.html. Zugriff am 12.01.2015.

[16] IDC., S. *Absatz von Smartphones weltweit vom 1. Quartal 2009 bis zum 2. Quartal 2014 nach Betriebssystem (in Millionen Stück)*. http://de.statista.com/statistik/daten/studie/74592/umfrage/absatz-von-smartphones-weltweit-nach-betriebssystem/. Zugriff am 25.11.2014.

[17] Yarow, J. 2014. *This Chart Shows Google's Incredible Domination Of The World's Computing Platforms*. http://www.businessinsider.com/androids-share-of-the-computing-market-2014-3?IR=T. Zugriff am 05.11.2014.

[18] Google Inc. *Dashboards | Android Developers*. https://developer.android.com/about/dashboards/index.html. Zugriff am 12.01.2015.

[19] Google Inc. 2014. *Android - History*. http://www.android.com/history/. Zugriff am 02.01.2015.

[20] Spence, E. 2015. *Why Is Nobody Using Android 5.0 Lollipop?* http://www.forbes.com/sites/ewanspence/2015/01/11/nobody-is-using-android-lollipop/. Zugriff am 12.01.2015.

[21] *Application Fundamentals | Android Developers*. http://developer.android.com/guide/components/

fundamentals.html. Zugriff am 05.10.2014.
[22] Stefan Brähler. 2010. Analysis of the Android Architecture.
[23] Wahli, U. 2000. *Servlet and JSP programming with IBM WebSphere Studio and VisualAge for Java*. IBM Corporation, International Technical Support Organization, San Jose, Calif.
[24] Eckert, C. 2012. *IT-Sicherheit*. *Konzepte - Verfahren - Protokolle*. Informatik 10-2012. Oldenbourg, München.
[25] NIST Computer Security Division. FIPS 180-4, Secure Hash Standard (SHS).
[26] Oracle Corporation. 2011. *Serializable (Java Platform SE 6)*. https://docs.oracle.com/javase/6/docs/api/java/io/Serializable.html. Zugriff am 10.01.2015.
[27] Rescorla, E. ©2001. *SSL and TLS*. *Designing and building secure systems*. Addison-Wesley, Boston.
[28] Google Inc. *GZIPOutputStream | Android Developers*. http://developer.android.com/reference/java/util/zip/GZIPOutputStream.html. Zugriff am 05.01.2015.
[29] Oracle Corporation. *MySQL : MySQL Connector/J Developer Guide*. http://dev.mysql.com/doc/connector-j/en/index.html. Zugriff am 10.09.2014.
[30] Ludmila I. Kuncheva. Combining Pattern Classifiers.
[31] Google Inc. *Sensors Overview*. http://developer.android.com/guide/topics/sensors/sensors_overview.html,. Zugriff am 25.11.2014.
[32] Google Inc. *SensorEvent | Android Developers*. http://developer.android.com/reference/android/hardware/SensorEvent.html. Zugriff am 31.01.2015.
[33] Google Inc. *System Permissions*. http://developer.android.com/guide/topics/security/permissions.html. Zugriff am 10.01.2015.
[34] 2005. European Telecommunications Standards, “Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) Interface”, June 2005.
[35] Google Inc. *Manifest.permission*. http://developer.android.com/reference/android/Manifest.permission.html. Zugriff am 17.01.2015.
[36] Android Open Source Project - Issue Tracker. *TelephonyManager.getLine1Number returns Null*. https://code.google.com/p/android/issues/detail?id=4339. Zugriff am 02.02.2015.
[37] 2011. Dirk Balfanz, Google, “Identity in the Platform - Thinking Beyond the Browser”, April 2011.
[38] Google Inc. *AccountManager*. http://developer.android.com/reference/android/accounts/AccountManager.html. Zugriff am 02.02.2015.
[39] Brian Wirsing. 2014. Wirsing-Sending And Receiving Data Via Bluetooth With An Android Device.
[40] Google Inc. BluetoothManager.
[41] 1999. Institute of Electrical and Electronics Engineers, “Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, September 1999.
[42] 2003. SanDisk Corporation, “SanDisk Secure Digital Card Product Manual“, December 2003.
[43] SD Association. *FAQs - SD Association*. https://www.sdcard.org/consumers/faq/. Zugriff am 15.01.2015.
[44] Google Inc. *CallLog*. http://developer.android.com/reference/android/provider/CallLog.html. Zugriff am 25.11.2014.
[45] Google Inc. libphonenumber.
[46] AppBrain. *Number of available Android applications*. http://www.appbrain.com/stats/number-of-android-apps. Zugriff am 02.01.2015.
[47] AppBrain. *Downloads on Google Play - AppBrain*. http://www.appbrain.com/stats/android-app-downloads. Zugriff am 03.01.2015.
[48] Google Inc. *Our Mobile Planet*. http://think.withgoogle.com/mobileplanet/de/downloads/. Zugriff

am 11.12.2014.
[49] Bundesverband Musikindustrie e. V. Jahrbuch BVMI 2012.
[50] Google Inc. MediaMetadataRetriever.
[51] Google Inc. Monitoring the Battery Level and Charging State.
[52] Google Inc. *SensorEventListener | Android Developers*. http://developer.android.com/reference/android/hardware/SensorEventListener.html. Zugriff am 17.02.2015.
[53] Google Inc. Location Strategies.
[54] 2014. 3rd Generation Partnership Project, "ETSI TS 122 016 V9.0.1", April 2014.
[55] 2014. Hristo Bojinov at el, „Mobile Device Identification via Sensor Fingerprinting", August 2014.
[56] 2011. Nasim Mansurov, „Dead vs Stuck vs Hot Pixels", August 2011.
[57] nonCamerasmartphones.com. 2014. *All Android Non Camera Smartphones 3G Mobile Phones Review Price.* http://www.noncamerasmartphones.com/. Zugriff am 04.01.2015.
[58] Jeff Medkeff. *Using Image Calibration Techniques to Reduce Noise in Digital Images - photo.net*. http://photo.net/learn/dark_noise/. Zugriff am 12.12.2014.
[59] Andy Böhme. 2005. Fast FOURIER-Transformation.
[60] Derawi, M. and Bours, P. 2013. Gait and activity recognition using commercial phones. *Computers & Security* 39, 137–144.
[61] Salvador, S. and Chan, P. 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis* 11, 5, 561–580.
[62] 2014. *Eclipsepedia*. https://wiki.eclipse.org/Main_Page. Zugriff am 05.02.2015.
[63] Google Inc. *Android Developer Tools | Android Developers*. http://developer.android.com/tools/help/adt.html. Zugriff am 01.02.2015.
[64] *ACRA/acra*. https://github.com/ACRA/acra. Zugriff am 01.02.2015.
[65] *ACRA/acralyzer*. https://github.com/ACRA/acralyzer/wiki. Zugriff am 14.01.2015.
[66] Philipp Mock et al. User Identification Using Raw Sensor Data From Typing on Interactive Displays. In *IUI '14*.
[67] Kuan-Ta Chen et al. User identification based on game play activity patterns. In *NetGames '07*.
[68] Yan Michalevsky, Dan Boneh, Gabi Nakibly. Gyrophone: Recognizing Speech From Gyroscope Signals.
[69] Matic, A. Speech Activity Detection Using Accelerometer.
[70] 2010. J. R. Kwapisz, G. M. Weiss, and S. A. Moore., "Activity recognition using cell phone accelerometers", December 2010.

Listings

Anhang

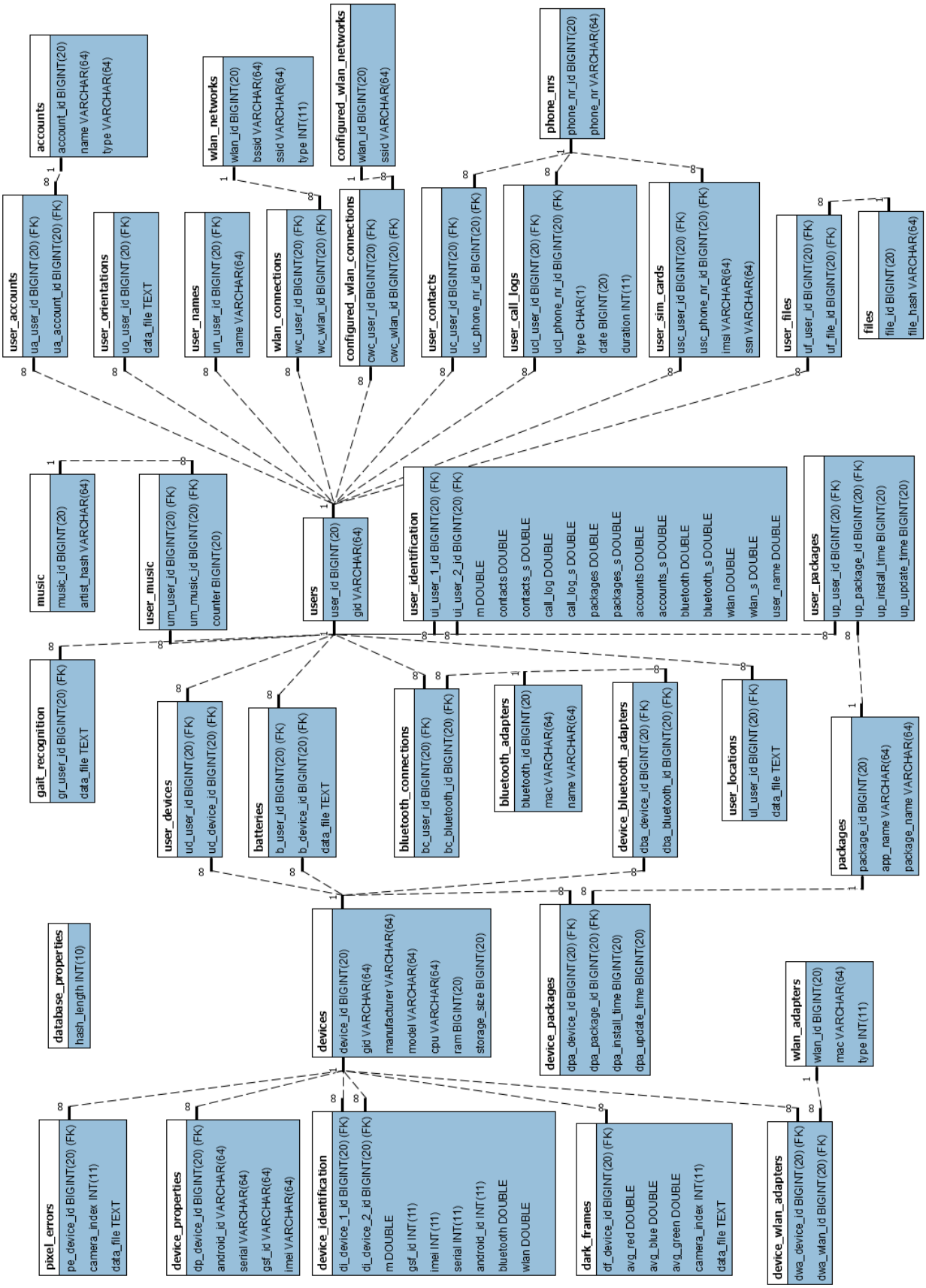


Abbildung 35: Datenbank-Diagramm