

Bachelorarbeit

Erreichbarkeit von Datenschutzhinweisen in mobilen Anwendungen

vorgelegt von

Jannis Dammann

geb. am 10. März 1999 in Hamburg

Matrikelnummer 7061777

MIN-Fakultät, Fachbereich Informatik

Studiengang B.Sc. Informatik

eingereicht am 17. Oktober 2020

Betreuer: Maximilian Blochberger, M.Sc

Erstgutachter: Prof. Dr.-Ing. Hannes Federrath

Zweitgutachter: Dr.-Ing. Daniel Demmler

Aufgabenstellung

Nach Artikel 13 und 14 der Datenschutz-Grundverordnung (DSGVO) müssen Verantwortliche die betroffene Person über die Datenverarbeitung informieren. Nutzer*innen (Betroffene) erhalten daher beim ersten Start einer mobilen Anwendung Informationen darüber, welche Daten von der Anwendung verarbeitet werden. Laut Artikel 12 DSGVO müssen Verantwortliche geeignete Maßnahmen treffen, um dem*der Betroffenen Informationen nach Artikel 13 und 14 in leicht zugänglicher Form zu übermitteln. Das heißt, dass dem*der Nutzer*in auch nach dem ersten Start einer Anwendung diese Informationen weiterhin zugänglich sein müssen.

Ziel der Arbeit ist es, eine Methodik zu entwickeln, mit der der Navigationsaufwand beim Abruf von Datenschutzhinweisen für die betroffene Person gemessen werden kann. Es sollen Android-Anwendungen mittels dynamischer Analyse untersucht werden.

Zusammenfassung

In dieser Arbeit wird, ausgehend von den Bestimmungen der in 2018 in Kraft getretenen Datenschutz-Grundverordnung der Europäischen Union, die Erreichbarkeit von Datenschutzhinweisen in mobilen Anwendungen betrachtet. Die Arbeit stellt das dynamische Analyseverfahren *DataPolicyRipper* vor, welches mithilfe des Frameworks „UI Automator“ [Doc19f] entwickelt wurde. Das Verfahren nutzt die Methodik des Graphical User Interface (GUI) Ripping, um mobile Anwendungen des Betriebssystems Android automatisiert nach Datenschutzhinweisen zu durchsuchen.

Weiterhin wird eine größere Anzahl an Anwendungen dieses Betriebssystems genutzt, um die Funktion von *DataPolicyRipper* zu evaluieren. Ebenso werden diese Anwendungen analysiert, um Erkenntnisse in Bezug auf die derzeitige Einbettung von Datenschutzhinweisen zu erlangen. Die wichtigsten Erkenntnisse dieser Analyse sind, dass aktuell keine einheitlichen Best Practices in der Bereitstellung von Datenschutzhinweisen und -erklärungen ersichtlich sind und die Einbettung dieser in der durchschnittlichen analysierten Anwendung Verbesserung bedarf. Weiterhin hat die Nutzerzahl einer Anwendung keinen ersichtlichen Einfluss auf die Erreichbarkeit der Datenschutzhinweise. Nutzer*innen werden durch die Analyseergebnisse auf Techniken hingewiesen, mit denen die Datenschutzhinweise in Anwendungen häufig zu erreichen sind. Für die Zukunft wünschenswert wäre eine Harmonisierung der Einbindung der Datenschutzhinweise und -erklärungen in den mobilen Anwendungen. Dies könnte durch den Betreiber des Betriebssystems oder durch ein politisches Organ erfolgen.

Der Quellcode der in dieser Arbeit vorgestellten Programme sowie Dokumentierungen der Analyseergebnisse werden aus Gründen der Transparenz in einem Verzeichnis auf der Webseite „GitHub“ zur Einsicht bereitgestellt [Git20].

Inhaltsverzeichnis

1 Einleitung	5
2 Verwandte Literatur	7
3 Vorgehensweise	8
4 Ausführung per Hand	9
4.1 Ausführungsbeispiel anhand einer Anwendung	9
4.2 Das Problem als Suchgraph	12
4.3 Ergebnisse weiterer Anwendungen	13
5 Technische Grundlagen	16
5.1 Android GUI	16
5.1.1 Technischer Aufbau	16
5.1.2 Wichtige Eigenschaften der GUI-Elemente	18
5.2 GUI Ripping	19
6 Automatisierung des Verfahrens	21
6.1 Verwendete Software	21
6.1.1 UI Automator	22
6.2 Funktionsweise von <i>DataPolicyRipper</i>	23
6.2.1 Komponente: Kanten	24
6.2.2 Komponente: Knoten	27
6.2.3 Komponente: Breitensuche	32
6.2.4 Komponente: Ausführende Klasse	36
6.3 Durchführung von <i>DataPolicyRipper</i>	37
7 Limitationen von <i>DataPolicyRipper</i>	42
7.1 Hürden bei der Entwicklung	42
7.2 Grenzen der Methodik	47
8 Ergebnisse der Analyse	50
8.1 Gesamtbetrachtung aller Anwendungen	51
8.2 Anwendungen vom selben Hersteller	57
8.3 Anwendungen gleicher Art	61
9 Fazit	64
9.1 Zukünftige Arbeiten	64
9.2 Bewertung der Analyseergebnisse	65
Literatur	67

1 Einleitung

Am 14. April 2016 wurde die Datenschutz-Grundverordnung (DSGVO) der Europäischen Union (EU) vom Europäischen Parlament angenommen und ist nach einer zweijährigen Übergangsphase am 25. Mai 2018 europaweit in Kraft getreten [Wir20]. Dies bedeutet, dass die Verordnung 2016/679 der Europäischen Union an die Stelle von bisherigen nationalen Gesetzen tritt [Fed20, S.26] und die EU-Datenschutzrichtlinie aus dem Jahr 1995 ersetzt [Wir20].

Die DSGVO hat das Ziel, das europäische Datenschutzrecht zu harmonisieren und ist für öffentliche und private Stellen gültig [Fed20, S.26]. In den Artikeln 13 und 14 des Gesetzestextes werden die Informationspflichten des*der Verantwortlichen bei der Erhebung von personenbezogenen Daten gegenüber der betroffenen Person geregelt. Hierzu gehören unter anderem der Zweck der Verarbeitung, die Speicherzeit der personenbezogenen Daten sowie die Hinweise auf die Rechte des/der Betroffenen in Bezug auf die Daten [Med19, S.48-52]. Diese Informationen müssen dem/der Betroffenen zum Zeitpunkt der Erhebung der Daten kundgegeben und zur Verfügung gestellt werden [Fed20, S.63-64].

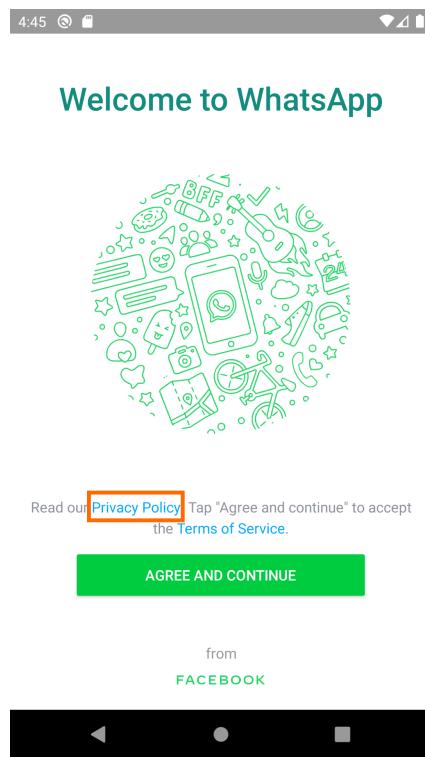


Abbildung 1.1: Anwendung „WhatsApp“: Datenschutzhinweise bei erstem Start

Da die Daten zum Zeitpunkt der Erhebung mitgeteilt beziehungsweise zur Verfügung gestellt werden müssen, werden dem/der Nutzer*in üblicherweise beim ersten Start einer Anwendung oder auch beim ersten Aufruf einer Webseite diese Informationen mitgeteilt. In Abbildung 1.1 ist als Beispiel die Mitteilung der Anwendung des Nachrichtendienstes „WhatsApp“ gezeigt. Interessante Details in Ansichten aus Anwendungen, auf die in dieser Arbeit eingegangen wird,

sind orangefarben markiert. Hier ist der Verweis auf die Datenschutzhinweise der Anwendung hervorgehoben. Der Nutzer muss in der Regel diesen Datenschutzrichtlinien vor Benutzung der Anwendung zustimmen. Dies ist im Falle der Anwendung „WhatsApp“ ebenfalls nötig.

In Artikel 12 der Verordnung wird festgelegt, dass die in den Artikeln 13 und 14 beschriebenen Daten, zusätzlich zu dieser Art der Benachrichtigung, dem/der Nutzer*in in „präziser, transparenter, verständlicher und leicht zugänglicher Form“ [Med19, S.146] zur Verfügung gestellt werden müssen. Dies kann auch in digitaler Form geschehen [Med19, S.146]. Über die in Artikel 12 formulierten Richtlinien hinaus wird nicht weiter spezifiziert, wie Informationen in leicht zugänglicher Form nach der Mitteilung beim ersten Start der Anwendung präsentiert werden sollen.

Diese Überlegung ist der Ausgangspunkt dieser Arbeit. Insbesondere wird ein Verfahren mit dem Namen *DataPolicyRipper* entwickelt, welches die Zugänglichkeit der Datenschutzhinweise in mobilen Anwendungen misst. Für die Analyse der Bereitstellung der Datenschutzhinweise nach Artikel 12 werden mobile Anwendungen des Betriebssystems Android genutzt. Die Android-Plattform wurde für die Entwicklung des Verfahrens ausgewählt, da diese auf dem heutigen Markt am weitesten verbreitet ist. Seit 2014 ist das Betriebssystem konstant auf über 50 Prozent aller mobilen Endgeräte installiert und hat derzeit sogar einen Marktanteil von circa 75 Prozent [Sta20c] [Sta20a].

Das entwickelte Verfahren wird im Laufe der Arbeit vorgestellt und die Ergebnisse der Analyse werden vorgestellt und diskutiert. Ebenfalls werden diese Ergebnisse für die Evaluation der Funktionalität des Verfahrens *DataPolicyRipper* genutzt. Daraufhin wird erörtert, auf welche Kriterien und Einzelheiten Nutzer*innen bei der Suche nach Datenschutzhinweisen in mobilen Anwendungen des Betriebssystems Android achten sollten. Insbesondere werden am Ende der Arbeit Best Practices vorgeschlagen, die von Entwickler*innen verfolgt werden könnten.

2 Verwandte Literatur

Um diese Arbeit in den Kontext des derzeitigen wissenschaftlichen Standes einzuordnen, soll ein Blick auf die verwandte Literatur geworfen werden. Zwei wesentliche Verwandtschaften finden sich zum einen in der Analysemethodik, die für dieses Thema genutzt wird, und zum anderen in der Betrachtung der Datenschutzhinweise in Anwendungen diverser Plattformen.

Die Methoden, die in dieser Arbeit verwendet werden, um Datenschutzhinweise in mobilen Anwendungen automatisiert zu erreichen, wurden bereits in einigen wissenschaftlichen Arbeiten thematisiert. Hierbei handelt es sich um Arbeiten, die die dynamische Analyse von Benutzerschnittstellen in mobilen Anwendungen näher betrachten. Genau genommen ist dies die dynamische Analysemethodik des Graphical User Interface (GUI) Rippings. Hier zu nennen sind die im folgenden Absatz beschriebenen Autor*innen und ihre wissenschaftlichen Beiträge zum Thema des GUI Rippings, besonders in Bezug auf mobile Anwendungen.

Im Rahmen der *Working Conferences on Reverse Engineering* wurden unter anderem von Memon in den Jahren 2003 [MBN03] und 2013 [Mem+13] Arbeiten veröffentlicht, die die Grundlagen der Methodik näher erläutern. Eine Arbeit von Guerreiro mit dem Titel *GUI ripping of iOS mobile applications* behandelt insbesondere das GUI Ripping von Anwendungen des Betriebssystems iOS, liefert jedoch auch allgemeine Informationen zu der Thematik [Gue16]. Außerdem relevant für die Analyse von Android-Anwendungen sind die Arbeiten von Amalfitano et al., die Beiträge zur *IEEE/AEC International Conference on Automated Software Engineering* darstellen. In diesen werden zwei unterschiedliche Verfahren des GUI Rippings vorgestellt, die Android-Anwendungen automatisiert testen [Ama+12b][Ama+12a].

Abzugrenzen sind die genannten Literaturen von dieser Arbeit durch die Zielsetzung, die mit der dynamischen Analyse erreicht werden soll. Im Normalfall wird GUI Ripping dazu eingesetzt, Anwendungen automatisiert zu testen und auf Fehlerfälle der Benutzerschnittstelle hin zu überprüfen. In dieser Arbeit soll die Methodik dazu genutzt werden, Datenschutzhinweise und -einstellungen in den Anwendungen zu finden.

Somit folgt die zweite Verwandtschaft in der aktuellen wissenschaftlichen Literatur. Unter dem Stichwort „Usable Privacy“ wurden mehrere Arbeiten veröffentlicht, die das Thema der Datenschutzerklärungen und deren Inhalte in verschiedenen Softwaresystemen behandeln. Wie beispielsweise der Konferenzbeitrag zur *Conference on Computational Linguistics* von Liu et al. [Liu+14] beschäftigt sich der Großteil dieser Arbeiten mit den natürlichsprachlichen Inhalten und Formulierungen der Erklärungen und deren Verständlichkeit. Weitere Arbeiten aus diesem Bereich sind *The usable privacy policy project*, eine Arbeit der Autor*innen Sadeh et al., die ein Verfahren zur automatischen Analyse und übersichtlichen Bereitstellung von Datenschutzhinweisen verschiedener Webseiten vorstellt [Sad+13]. Zum anderen sind ebenfalls Beiträge zu den *Proceedings on Privacy Enhancing Technologies*, wie der Beitrag *MAPS: Scaling privacy compliance analysis to a million apps* von Zimmek et al. [Zim+19], zu nennen. Dieser behandelt den automatisierten Vergleich von Datenschutzhinweisen von mobilen Anwendungen und den Daten, die diese kollektieren [Zim+19]. Diese Arbeit ist demnach eng mit der folgenden verwandt.

3 Vorgehensweise

Der Fokus dieser Arbeit liegt auf der Entwicklung des Verfahrens zum Messen des Navigationsaufwands in mobilen Anwendungen auf dem Betriebssystem Android. Im Verlauf der Arbeit werden die verschiedenen Schritte in der Entwicklung vorgestellt und näher betrachtet. Die grundlegende Idee hinter der Methodik ist das Messen der Interface-Interaktionen, die der Benutzer tätigen muss, um vom Start der Anwendung zu den Datenschutzhinweisen zu gelangen.

Begonnen wird im Kapitel *Ausführung per Hand* mit der händischen Ausführung des Verfahrens bei einer kleinen Auswahl an Anwendungen. Ziel dieses Schrittes ist es, einen generellen Überblick darüber zu erhalten, wie Anwendungsentwickler*innen die Datenschutzhinweise in ihre Programme einpflegen. Insbesondere wird dieses Kapitel dazu genutzt, die Idee hinter dem Verfahren näher zu betrachten und vorzustellen. Weiterhin werden die Interaktionen, die im Verfahren gemessen werden, thematisiert. Hierzu werden die Anwendungen explorativ durchsucht, um erste Vermutungen zu erhalten, wie Datenschutzhinweise in Anwendungen erreicht werden. Diese Erkenntnisse werden darauffolgend im weiteren Verlauf der Arbeit verwendet.

Hierzu wird im Kapitel *Technische Grundlagen* die Funktionsweise und der Aufbau des Graphical User Interface (GUI) im Betriebssystem Android thematisiert und weiterhin die Methodik vorgestellt, die für die dynamische Analyse der Anwendungen im Verfahren genutzt wird: das GUI-Ripping. Sind diese beiden Grundlagen geklärt, kann die *Automatisierung des Verfahrens* durch dessen Entwicklung und Funktionsweise beschrieben werden.

Ist das Programm zum automatisierten Verfahren thematisiert worden, können im nächsten Abschnitt *Ergebnisse der Analyse* eine große Anzahl an Anwendungen analysiert und die Ergebnisse dieser Analyse betrachtet werden. Für die Auswahl der zu analysierenden Anwendungen sowohl im Schritt *Ausführung per Hand* als auch bei der Analyse durch das automatisierte Verfahren wird der Google Play Store benutzt. Dies ist der primäre App-Store auf der Android-Plattform, welcher im Jahr 2019 über zwei Milliarden monatliche Nutzer*innen hatte [Pla19, S.5]. Die Website „AndroidRank“ liefert Daten wie Nutzerzahlen, Durchschnittsbewertungen und andere Statistiken zu den Anwendungen aus dem Play Store und bietet die Möglichkeit, die Anwendungen anhand dieser Kriterien zu sortieren [And19b].

4 Ausführung per Hand

Um die Entwicklung des Verfahrens zu starten, soll, wie bereits im vorangegangenen Kapitel *Vorgehensweise* erklärt, das Verfahren mit einer kleinen Auswahl an mobilen Anwendungen per Hand ausgeführt werden. Die Auswahl, die getroffen wurde, besteht aus zehn Anwendungen aus dem primären App-Store auf dem Betriebssystem Android, dem Google Play Store. Hierzu wird die Statistik der meistinstallierten Anwendungen der Website „AndroidRank“ genutzt [And19b].

4.1 Ausführungsbeispiel anhand einer Anwendung

In dieser Statistik, die nach der Anzahl an Installationen der Anwendungen auf Android-Geräten sortiert ist, ist „WhatsApp Messenger“ die Anwendung, die am meisten Nutzer*innen verzeichnet [And19b]. Dieser wird jedoch nicht betrachtet, da für die Nutzung des Nachrichten-Dienstes eine Telefonnummer nötig ist. Die Anwendung mit den zweitmeisten Nutzern*innen gehört zum selben Mutterkonzern, dem Social-Media-Riesen „Facebook“ [Ber19]. Ob die beiden Anwendungen aus diesem Grund eine ähnliche Art der Bereitstellung der Datenschutzhinweise verwenden, ist jedoch unklar. Diese Thematik wird im weiteren Verlauf des Kapitels näher betrachtet und ebenfalls im Kapitel *Ergebnisse der Analyse* thematisiert. In diesem Fall handelt es sich um die mobile Anwendung „Facebook“ [And19b]. Mit dieser soll nun das grundlegende Prinzip hinter der zu entwickelnden Methodik ausführlich verdeutlicht werden.

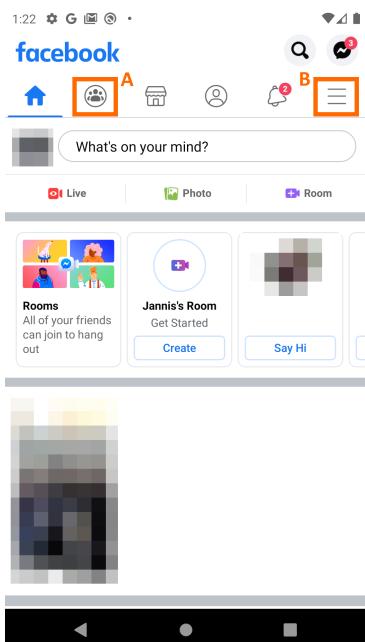


Abbildung 4.1: Startbildschirm

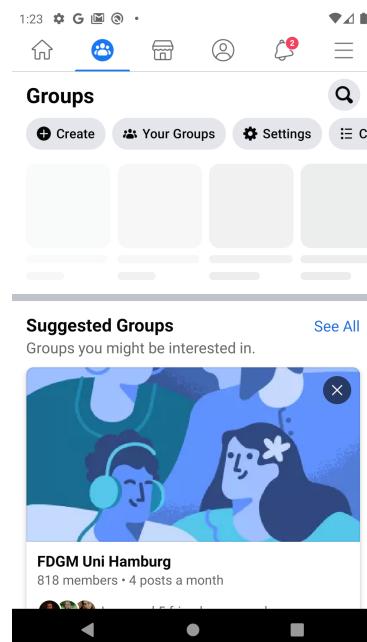


Abbildung 4.2: Groups

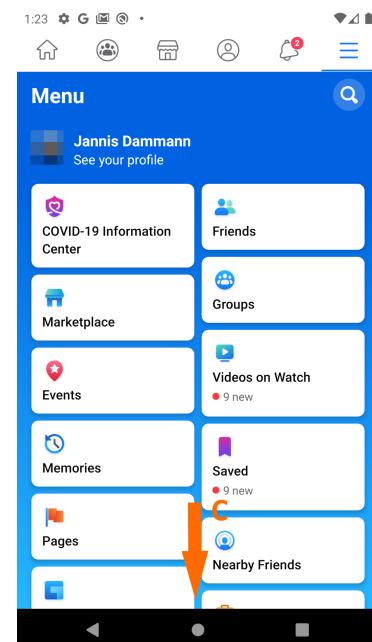


Abbildung 4.3: Menü(1)

Die für diese Erklärung wichtigen Stellen in der GUI sind in den Abbildungen durch orange-farbene Rechtecke mit Buchstaben gekennzeichnet, die Rechtecke und Buchstaben sind keine Elemente der Anwendungsoberfläche. Sensible und persönliche Daten sind durch Verpixelungen unkenntlich gemacht. Zwei Abbildungen, die derselben Ansicht der Anwendung angehören, jedoch unterschiedliche Funktionen bereitstellen, werden durch eingeklammerte Zahlen in der Bildunterschrift unterschieden.

Öffnet man die Anwendung des sozialen Netzwerks und ist in dieser bereits angemeldet, wird die in Abbildung 4.1 zu sehende Ansicht des Newsfeeds gezeigt. In dieser ist es möglich, eine Vielzahl an Benutzerinteraktionen durch den Touchscreen des Smartphones auszuführen. Beispiele dieser wären das Scrollen durch den Newsfeed oder das Betätigen einer der sechs Buttons in der Toolbar, von denen zwei in der Abbildung mit **A** und **B** markiert sind.

Wird beispielsweise der Button **A** betätigt, gelangt man zu der in Abbildung 4.2 dargestellten Seite der Anwendung, in der Nutzer*innen Inhalte angezeigt werden, die mit Gruppen in dem sozialen Netzwerk zu tun haben. Da jedoch in diesem Verfahren nach Datenschutzhinweisen gesucht wird und hier sehr wahrscheinlich keine zu finden sein werden, sollte eine andere Ansicht erforscht werden.

Da die naheliegendere Platzierung für Datenschutzhinweise im Menü beziehungsweise in den Einstellungen der Anwendung ist, könnte auf der Suche nach diesen zum Beispiel der Button **B** ausgewählt werden. Dieser führt zur Menüansicht der Anwendung, die in Abbildung 4.3 zu sehen ist. Um neben den hier platzierten Kacheln weitere Menüoptionen aufrufen zu können, wird in dieser Ansicht weiter nach unten gescrollt, hier mit dem Pfeil an Stelle **C** dargestellt. Dies führt zu Abbildung 4.4. Hier können nun mehrere Optionsseiten geöffnet werden, beispielsweise **Settings & Privacy** an Stelle **D**, in welcher die Platzierung der Datenschutzhinweise vermutet wird. Diese Benutzerinteraktion klappt das in Abbildung 4.5 dargestellte Untermenü auf. Um zu den Einstellungen der Anwendung zu gelangen, wird hier die Schaltfläche **F** mit der Aufschrift **Settings** betätigt.

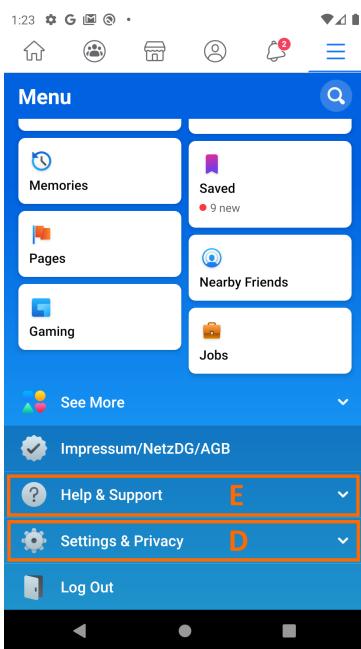


Abbildung 4.4: Menü(2)

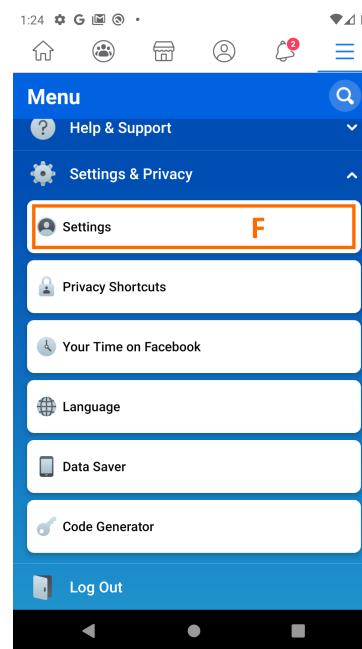


Abbildung 4.5: Menü(3)

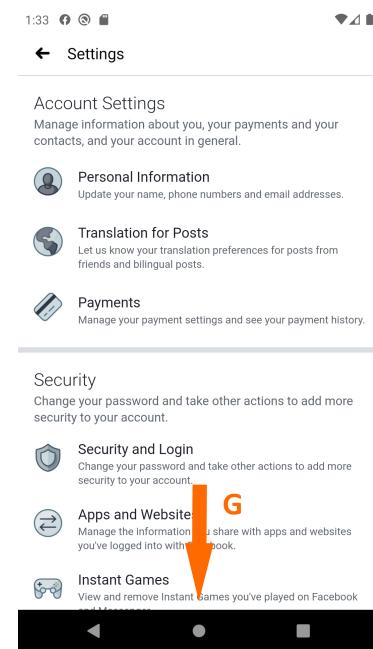


Abbildung 4.6: Optionen(1)

Nachdem durch das Auswählen des Buttons **F** die Einstellungen geöffnet wurden, befindet sich die Anwendung in dem Zustand, der in Abbildung 4.6 zu sehen ist. Hier sind anfänglich keine Optionen verfügbar, die auf Datenschutzhinweise hindeuten. Wird in den Einstellungen weiter nach unten gescrollt (**G**), werden weitere Optionen angezeigt, zu sehen in Abbildung 4.7. In der letzten Einstellungsrubrik **Legal and Policies** werden Unterseiten für die jeweiligen rechtlichen Erklärungen der Anwendung, wie beispielsweise die Allgemeinen Geschäftsbedingungen oder die Cookie-Richtlinien, angezeigt. Neben den beiden genannten werden auch die Datenschutzbedingungen, mit **H** markiert, angeboten. Wird dieser Punkt ausgewählt, werden die in Abbildung 4.9 gezeigten Datenschutzbedingungen und -hinweise geöffnet und das Ziel der Suche ist erreicht.

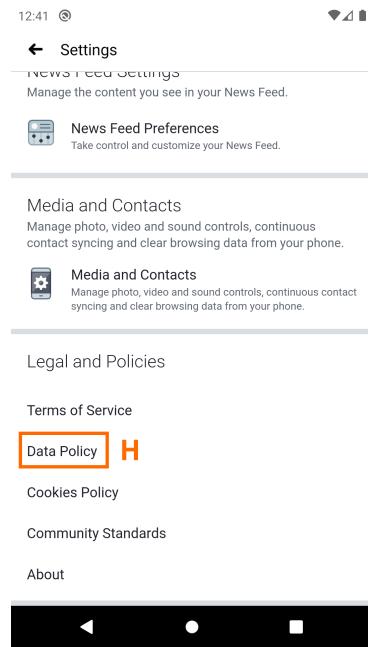


Abbildung 4.7: Optionen(2)

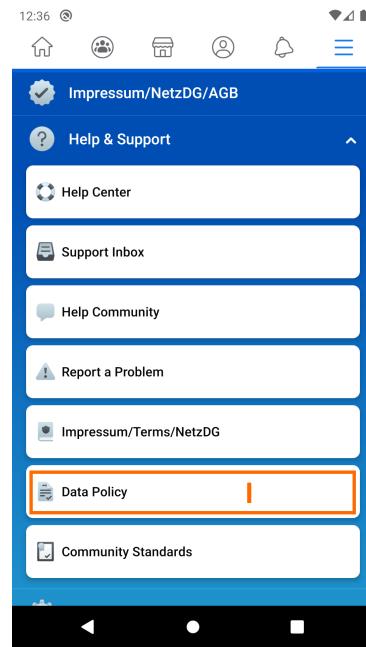


Abbildung 4.8: Menü(4)

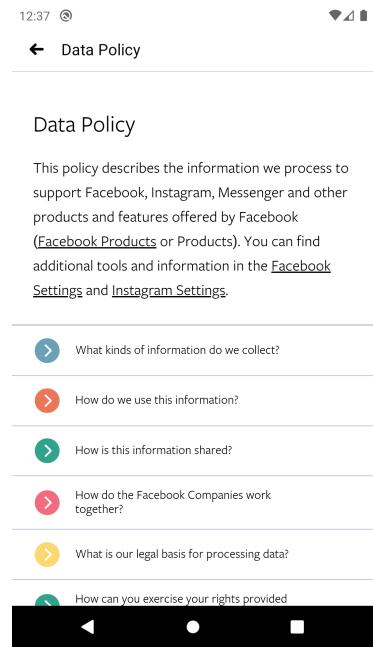


Abbildung 4.9: Datenschutz

Dieser eben beschriebene Suchvorgang nach den Datenschutzhinweisen in der Anwendung des sozialen Netzwerks lässt sich ebenfalls durch die Abfolge der Benutzerinteraktionen als Pfad ausdrücken. Hier wäre das der Pfad $P_1 = \{B, C, D, F, G, H\}$, wenn das Scrollen innerhalb einer Seite als eine Interaktion gewertet wird. Der Pfad hat eine Länge von $L_1=6$, also müssen sechs Benutzerinteraktionen ausgeführt werden, um vom Start der Anwendung zu den Datenschutzhinweisen zu gelangen. Die Interaktion **A** wird nicht berücksichtigt. Sie würde den Pfad nur unnötig verlängern, da der Menübutton **B** sowohl aus der Ansicht in Abbildung 4.1 als auch aus der Gruppen-Ansicht in Abbildung 4.2 verfügbar ist. Die Suche nach Datenschutzhinweisen wird somit nur auf die nötigen Interaktionen reduziert. Ist eine Interaktion überflüssig, wird sie nicht beachtet. Dies wird im nächsten Abschnitt *Das Problem als Suchgraph* sowie im weiteren Verlauf der Arbeit näher erläutert. Wichtig für die explorative Analyse ist hier jedoch anzumerken, dass Interaktionen, die sowohl in einer vorherigen Ansicht des Pfades als auch in einer späteren Ansicht getätigten werden können, in der späteren Ansicht nicht durchsucht werden müssen, da sie ebenfalls aus der früheren Ansicht erreicht werden können.

Es gibt in diesem Fall noch einen Pfad, der weniger als sechs Schritte benötigt, um zum Ziel der Suche zu gelangen. Wird in der Menüansicht in Abbildung 4.4 anstatt Interaktion **D** der Button **Help & Support** (**E**) gewählt, öffnet sich das in Abbildung 4.8 dargestellte Untermenü. Hier befindet sich ein Unterpunkt mit der Aufschrift **Data Policy** an Stelle **I**, welcher ebenfalls

zu den Datenschutzbedingungen führt. Somit wäre der dazugehörige Pfad $P_2 = \{B, C, E, I\}$ mit der Länge $L_2=4$ um zwei Benutzerinteraktionen kürzer als Pfad P_1 . P_2 ist somit der kürzeste Pfad von Interaktionen zwischen dem Startbildschirm und den Datenschutzhinweisen in der Anwendung „Facebook“.

4.2 Das Problem als Suchgraph

Das Problem der Suche nach Datenschutzhinweisen lässt sich als Suche in einem Suchgraphen vorstellen und visualisieren. Ein Ausschnitt von einem aus den bisherigen Abbildungen zur Anwendung „Facebook“ zusammengesetzten Graph für das derzeitige Beispiel ist in Abbildung 4.10 dargestellt.

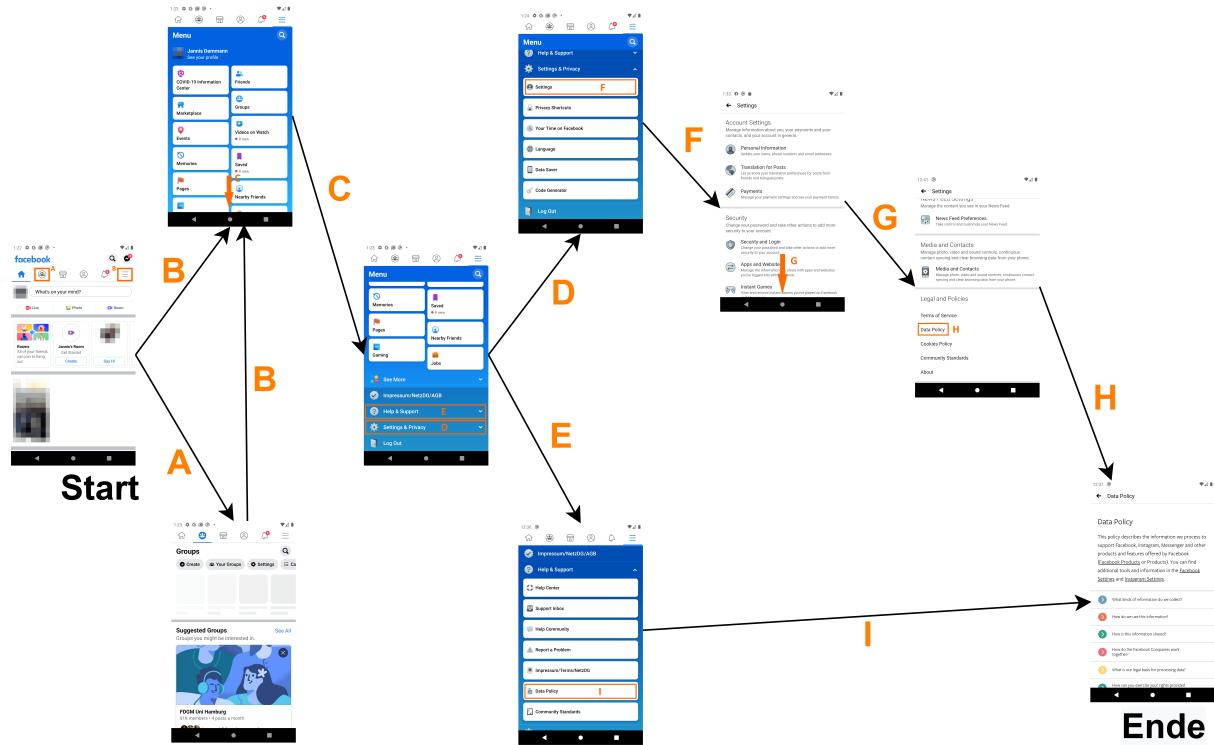


Abbildung 4.10: Suchgraph zum Anwendungsbeispiel Facebook

Ein gerichteter Graph $G(V,E)$ besteht aus einer Menge V an Knoten und einer Menge E von geordneten Paaren aus Knoten, die Kanten genannt werden [PM10, S.75]. In diesem Fall bestehen die Knoten aus den unterschiedlichen Ansichten, die in den bisherigen Abbildungen dargestellt wurden. Eine Kante führt von einer Ansicht zur nächsten und repräsentiert eine der in den Abbildungen markierten Benutzerinteraktionen. Die Beschriftungen in dem aktuellen Suchgraphen sind analog zu denen, die in den Abbildungen des vorherigen Abschnitts *Ausführungsbeispiel anhand einer Anwendung* zu finden sind.

Um in einem Graphen eine Suche durchzuführen, werden ein Startknoten sowie ein Endknoten benötigt [PM10, S.75]. Hier ist der Startknoten durch den Startbildschirm der Anwendung aus Abbildung 4.1 repräsentiert. Die Datenschutzhinweise, zu sehen in Abbildung 4.9, bilden den Endknoten des Graphen. Beides ist in der Abbildung durch Schriftzüge gekennzeichnet. Sind diese Definitionen nun geklärt, ist es möglich, in diesem Graphen die Suche nach einem kürzesten Pfad vom Startknoten bis zum Endknoten durchzuführen.

In dem Beispielgraphen zur Anwendung des sozialen Netzwerks ist leicht zu erkennen, dass die bisherigen gefundenen Lösungen hier ebenfalls geltend sind. Der Pfad P_1 führt mit sechs Schritten vom Startknoten bis zum Endknoten, während der Pfad $P_2=\{\mathbf{B}, \mathbf{C}, \mathbf{E}, \mathbf{I}\}$ mit der Länge $L_2=4$ den kürzesten Pfad darstellt.

Zudem handelt es sich bei einem Suchgraphen zur Suche nach Datenschutzhinweisen um einen gerichteten, azyklischen Graphen, also einen gerichteten Graphen, der keine Zyklen enthält [PM10, S.75]. Ein Zyklus in einem Graphen ist ein Pfad, der mit dem Knoten endet, mit dem er ebenfalls beginnt [PM10, S.75]. In dem in dieser Arbeit betrachteten Fall ist dies nicht sinnvoll, da wir einen bereits betrachteten Knoten nicht noch einmal durchsuchen möchten. Wir haben im Idealfall bereits alle relevanten von diesem ausgehenden Benutzerinteraktionen erforscht. Ein Beispiel hierfür wäre, wenn wir durch Interaktion \mathbf{A} von Abbildung 4.1 zu Abbildung 4.2 gelangen, dann aber eine potentielle Benutzerinteraktion \mathbf{N} ausführen würden, wie die *Zurück*-Taste der Android-Bedienleiste, mit der wir zurück zum Startbildschirm der Anwendung gelangen. Jeder Pfad, der mit $P_3=\{\mathbf{A}, \mathbf{N}\}$ startet, ist nicht zielführend, da er einen Zyklus darstellt.

Das Problem als Suche nach einem kürzesten Pfad in einem Graphen zu betrachten, hat den Vorteil, dass es für diese Art an Suchproblemen schon diverse bewährte Lösungsverfahren, wie beispielsweise die Tiefen- oder Breitensuche, gibt [PM10]. Diese können in der *Automatisierung des Verfahrens* genutzt werden und sind somit im nächsten Kapitel relevant.

4.3 Ergebnisse weiterer Anwendungen

Um dieses Kapitel abzuschließen, wurden neun weitere Anwendungen per Hand, analog zu dem im Voraus ausführlich beschriebenen Verfahren, analysiert. Bei den analysierten Anwendungen handelt es sich um weitere Beispiele aus der Statistik zu den Nutzerzahlen [And19b]. Hierbei wurde darauf geachtet, dass maximal zwei Anwendungen des selben Entwicklerhauses gewählt wurden. In einem solchen Fall wurden dann die beiden Anwendungen mit den meisten Nutzerzahlen ausgewählt und die weiteren Anwendungen des Entwicklers wurden durch die darauffolgenden Anwendungen mit den höchsten Nutzerzahlen ersetzt. Eine Ausnahme bildet die Anwendung „Google Play Services“, welche sich auf Platz eins der Statistik befindet. Hierbei handelt es sich um einen Dienst, der im Hintergrund auf allen Geräten der Plattform Android läuft [And19a]. Aus diesem Grund konnte dieser Dienst nicht analysiert werden. Die Ergebnisse der Analyse sind in der Tabelle aus Abbildung 4.11 abgebildet.

Die Anwendungen sind in der Tabelle absteigend nach ihrer Installierungsanzahl auf Basis der Webseite AndroidRank [And19b] vom 28.05.2020 sortiert. Die Installationszahlen der Anwendungen sind hinter deren Namen eingetragen. Die Nutzungskomplexität in Bezug auf die Platzierung von Datenschutzrichtlinien und -hinweisen entspricht dem kürzesten Pfad, welcher in der dritten Spalte für die jeweilige Anwendung spezifiziert wird. Die Angaben der Pfade entsprechen der Anzahl der Benutzerinteraktionen und beinhalten das Scrollen als Interaktion. Zusätzlich zu den Angaben der kürzesten Pfade ist in der Tabelle die Länge des ersten gefundenen Pades in der händischen Analyse angegeben sowie gegebenenfalls die Länge weiterer gefundener Möglichkeiten, zu den Datenschutzrichtlinien zu gelangen.

Die Werte für die Länge der kürzesten Pfade der getesteten Anwendungen liegen zwischen zwei und sechs Benutzerinteraktionen. Der Durchschnittswert über alle zehn Anwendungen beträgt **4.3** Interaktionen. Der Median liegt bei **4**. Es ist zudem keine Korrelation zwischen

Anwendung	erster gefundener Pfad	kürzester Pfad	evtl. weitere Pfade
Facebook (>5 Mrd.)	6	4	erster Pfad(6)
Youtube (>5 Mrd.)	2	2	4
Google Chrome (>5 Mrd.)	6	6	nein
Instagram (>1 Mrd.)	5	5	Einstellungssuche(5)
Subway Surfers (>1 Mrd.)	4	4	nein
Candy Crush (>1 Mrd.)	6	6	nein
Snapchat (>1 Mrd.)	4	4	nein
TikTok (>1 Mrd.)	3	3	5
Skype (>1 Mrd.)	5	5	nein
Samsung Gallery (>1 Mrd.)	4	4	nein

Tabelle 4.1: Ergebnisse der händischen Analyse

Installationsanzahl und der Länge des kürzesten Pfades zu erkennen. In nur vier Anwendungen konnte eine weitere Möglichkeit gefunden werden, die Datenschutzhinweise zu öffnen.

Im Folgenden werden einige Besonderheiten angemerkt, die bei der händischen Analyse aufgefallen sind. Bei Anwendungen, die vom selben Entwicklungsunternehmen stammen, gibt es sowohl Beispiele, die die Platzierung von Datenschutzhinweisen ähnlich lösen, als auch eines, in dem sehr unterschiedliche Wege in unterschiedlichen Diensten genutzt werden. Sowohl „Facebook“ als auch „Instagram“ gehören zum Social-Media-Konzern Facebook [Rus12]. Der kürzeste Pfad in der Anwendung von Instagram ist zwar um eine Benutzerinteraktion länger als die von Facebook, das grundlegende Prinzip, die Datenschutzhinweise zu erreichen ist jedoch analog. Als Gegenbeispiel sind die beiden Dienste von Google zu nennen. Während die Lösung für die Videoplattform „Youtube“ mit der niedrigsten Anzahl an Interaktionen implementiert wird, sind die Datenschutzbedingungen im Webbrowser „Google Chrome“ mit der höchsten Anzahl an Interaktionen aus allen getesteten Anwendungen zu erreichen.

Die eben genannte Anwendung „Youtube“ ist als Positivbeispiel zu nennen. Öffnet man aus dem Startbildschirm des Dienstes das Menü, wird direkt am unteren Ende der Anzeige ein Button (Y) bereitgestellt, der auf die Datenschutzbedingungen verweist. Dies ist in Abbildung 4.11 zu sehen. Weiterhin bietet die Anwendung die Möglichkeit, die Hinweise ähnlich wie bei anderen Beispielen über die Einstellungen mit vier Benutzerinteraktionen zu erreichen.

Eine weitere Besonderheit ist bei der Anwendung „Instagram“ zu finden. Wie in Abbildung 4.12 gezeigt, kann zusätzlich zum ersten gefundenen Weg, welcher auch der kürzeste Pfad ist, in den Einstellungen über eine Suchleiste gesucht werden. Hierüber ist es auch möglich, die Datenschutzbedingungen aufzufinden. Wird das Eingeben der Textzeichen als eine einzige Benutzerinteraktion gewertet, benötigt dieser Weg ebenfalls fünf Benutzerinteraktionen. Dieses Konzept könnte demnach ebenfalls in der Automatisierung des Verfahrens betrachtet werden.

Einige Entwickler*innen binden ihre Datenschutzhinweise nicht als eigenständige Anzeige in die Anwendung ein. Stattdessen öffnet das Betätigen des Buttons, der zu den Datenschutzbedingungen führt, eine Webseite, auf der diese angezeigt werden. Hierzu gehören unter anderem das Spiel „Subway Surfers“ und das Videokonferenzprogramm „Skype“ von Microsoft. Letzteres ist in Abbildung 4.13 dargestellt.

Allgemein lässt sich beobachten, dass die Ansätze für die Platzierung der Hinweise sich bei den getesteten Beispielen ähneln. Im Normalfall muss, wie im Abschnitt *Ausführungsbeispiel anhand einer Anwendung* gezeigt, das Menü der mobilen Anwendung geöffnet werden. Dieses Prinzip

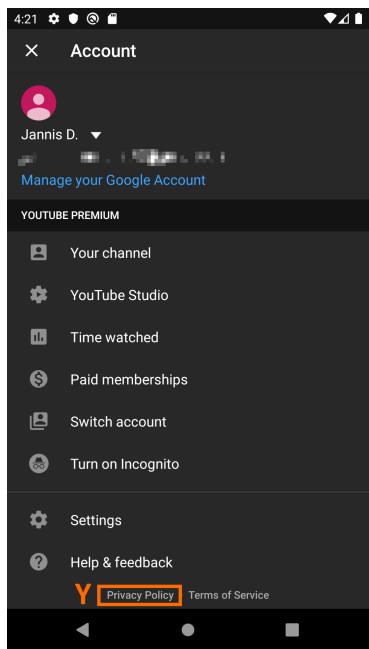


Abbildung 4.11: Youtube

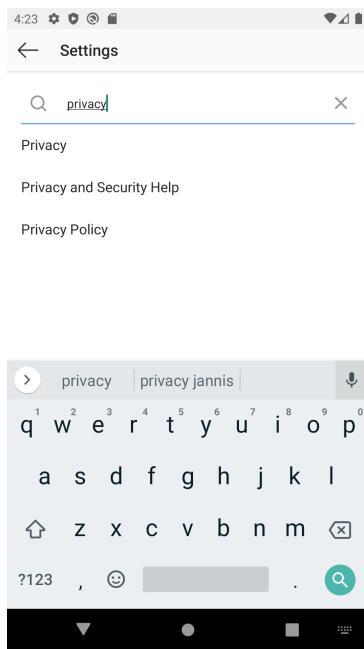


Abbildung 4.12: Instagram



Abbildung 4.13: Skype

ist in allen der getesteten Anwendungen wiederzufinden. Wie es ab diesem Punkt weiter geht, ist nicht einheitlich geregelt, es gibt jedoch einige Implementationen, die ähnliche Techniken verfolgen. In einem Großteil der Anwendung müssen vom Menü aus die Einstellungen geöffnet werden. Hier werden dann die Hinweise entweder als einzelner Einstellungspunkt eingepflegt oder es müssen noch weitere Interaktionen getätigert werden, wie beispielsweise das Öffnen der Hilfe-Seite der Anwendung. Dies wird ebenfalls häufig implementiert. Müssen die Einstellungen nicht geöffnet werden, sind die Datenschutzhinweise bei den getesteten Anwendungen direkt auffindbar. Dies ist bei „Youtube“ und „TikTok“ der Fall gewesen.

Diese Erkenntnisse sollen nun helfen, im weiteren Verlauf die *Automatisierung des Verfahrens* durchzuführen und die Suche bei der Automatisierung zu verkürzen.

5 Technische Grundlagen

Dieses Kapitel behandelt die Grundlagen, die für die programmiertechnische Automatisierung des zuvor per Hand ausgeführten Verfahrens benötigt werden. Dies ist zum einen der technische Aufbau der *Android GUI* und einige für die Automatisierung relevanten Aspekte in Bezug auf diese. Zum anderen wird die Funktionsweise der dynamischen Analysemethodik, welche die Grundlage für das Verfahren in dieser Arbeit bildet, im Abschnitt *GUI Ripping* behandelt.

Programmiertechnische Begrifflichkeiten, die in dieser Arbeit definiert werden und auf welche ein besonderer Fokus liegt, werden bei ihrer ersten Definition hervorgehoben, wie dieses **Beispiel** zeigt. Im weiteren Textverlauf werden diese lediglich geneigt dargestellt. Dieses *Beispiel* veranschaulicht die Schreibweise. Technische Begrifflichkeiten wie Strings, Rückgabewerte oder Klassennamen aus Frameworks werden in Kapitalschrift angegeben, an folgendem **BEISPIEL** verdeutlicht.

5.1 Android GUI

Wie in der Online-Dokumentation für Android-Entwickler*innen beschrieben, ist die GUI einer Anwendung all das, was der*die Benutzer*in sehen kann und ebenso alles, womit Interaktionen durch den*die Benutzer*in möglich sind [Doc19g]. Somit stellt sie die Schnittstelle zwischen dem*der Benutzer*in und einer Anwendung auf einem Endgerät dar [Tho16]. Die von der Schnittstelle akzeptierten Eingaben sind die möglichen Benutzerinteraktionen auf dem Endgerät. Für die Zwecke dieser Arbeit werden lediglich die zwei primären Gesten auf dem Touchscreen eines Gerätes betrachtet: das Klicken sowie das Scrollen. Wie die beiden Interaktionen in der GUI implementiert werden, wird im folgenden näher erläutert.

Für die Entwicklung der GUI stellt der Mutterkonzern und Entwickler des Betriebssystems Android, „Google“, die Designsprache „Material Design“ zur Verfügung. Über diese trifft der Konzern die Aussage „Material ist ein anpassbares System von Richtlinien, Komponenten und Werkzeugen, die die Best Practices für das Design von Benutzeroberflächen unterstützen“ [Goo]. Diese Richtlinien stellen jedoch nur Empfehlungen dar und sind kein Zwang für Entwickler*innen [Tho16]. Android ist somit eine offene Plattform, Entwickler*innen können über das Design ihrer GUI frei entscheiden ohne festen Richtlinien folgen zu müssen.

5.1.1 Technischer Aufbau

Trotz dieser Freiheit gibt es einige Eigenschaften, die die Interfaces von allen Android-Anwendungen gemeinsam haben: ihren technischen Aufbau. Die folgenden Bemerkungen dieses Absatzes beziehen sich auf die Online-Dokumentation zu Aktivitäten für Android-Entwickler*innen [Doc19c]. Eine Anwendung des Betriebssystems besteht aus einer oder mehreren **Activities** beziehungsweise **Aktivitäten**. Diese erfüllen jeweils einen Zweck für das Programm, wie beispielsweise die Implementierung des Startbildschirms oder der Anwendungseinstellungen. Im

Normalfall wird eine *Aktivität* dafür genutzt, eine Seite der Anwendung zu implementieren. Zudem stellt eine *Aktivität* den Bereich bereit, in dem die Benutzeroberfläche dargestellt wird. Eine *Aktivität* nimmt üblicherweise den ganzen Bildschirm des Endgeräts ein [Tho16].

Die GUI, die von einer *Aktivität* angezeigt wird, besteht aus zwei Arten von Elementen: **View** und **ViewGroup** [Doc20e]. Ein **View**-Objekt bildet ein Element in der GUI ab, es kann von Benutzer*innen gesehen werden und es kann mit ihm interagiert werden [Doc20e]. Typische Beispiele aus Anwendungen sind simple Container für Text (TEXTVIEW), BUTTONS oder Bilder (IMAGEVIEW) [Tho16]. **ViewGroup**-Objekte sind Behälter für **View**-Objekte und weitere **ViewGroup**-Objekte [Doc20e]. Beispiele hierfür sind **Layouts**, wie ein LINEARLAYOUT, welches die ihm untergeordneten Elemente vertikal oder horizontal anordnet [Tho16]. Die **View**- und **ViewGroup**-Objekte einer *Aktivität* werden hierarchisch angeordnet, wobei ein **ViewGroup**-Objekt an oberster Stelle angeordnet sein muss [Doc20e]. Eine allgemeine Veranschaulichung für eine solche Hierarchie liefert Abbildung 5.1.

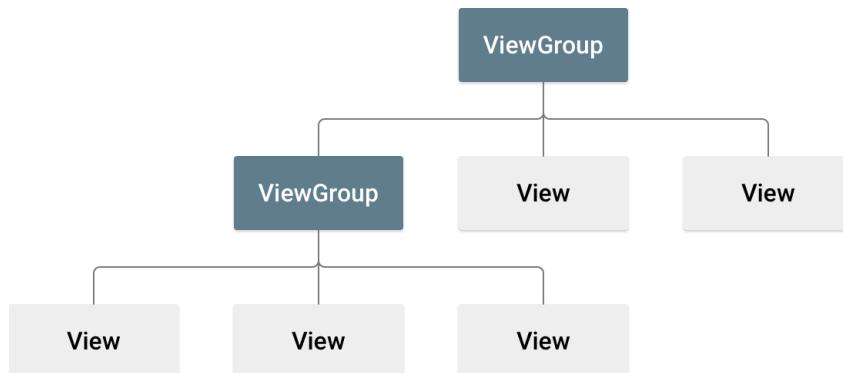


Abbildung 5.1: Grafische Darstellung einer View-Hierarchie [Doc20e]

Die Hierarchie der GUI-Elemente wird im Normalfall bei der Entwicklung in einem Dokument mithilfe eines XML-Standards für die Android-Entwicklung festgelegt [Doc20e]. Diese XML-Hierarchie kann ebenfalls während der Laufzeit der Anwendung abstrahiert werden. Hierfür wird das Werkzeug „UI Automator Viewer“ bereitgestellt [Doc19f]. Als Beispiel ist die Hierarchie der Menüansicht der Anwendung „Youtube“ (Abbildung 4.11) in Abbildung 5.2 zu sehen. Die eckigen Klammern hinter den jeweiligen Namen der UI-Objekte sind nicht zu beachten, sie spezifizieren lediglich deren Maße und deren Position auf dem Bildschirm des Endgeräts. Die Zahlen in den runden Klammern identifizieren die jeweiligen Objekte in der gleichen hierarchischen Ebene. Wie dargestellt, sind alle Objekte der Ansicht einem FRAMELAYOUT untergeordnet. Dieses hat in der hierarchischen Ebene direkt unter sich sechs Kinder, welche mit den Identifikationsnummern (0) bis (5) versehen sind. Lediglich das Kindesobjekt mit der Nummer (2) hat weitere Kindesobjekte in der Hierarchie. Verfolgt man dieses Prinzip weiter, ist jedes zu sehende Element in der Ansicht aus Abbildung 4.11 vorzufinden. Diese Hierarchie verdeutlicht die interne Repräsentation der GUI-Elemente gut. Eine statische Analyse der Anwendungen unter Verwendung der XML-Hierarchie ist nicht möglich. Dies liegt daran, dass der Großteil der zu untersuchenden Anwendungen keinen Zugriff auf ihren Quellcode ermöglichen und die Hierarchie lediglich über das bereits genannten Tool „UI Automator Viewer“ zur Laufzeit, also dynamisch, abstrahiert werden kann.

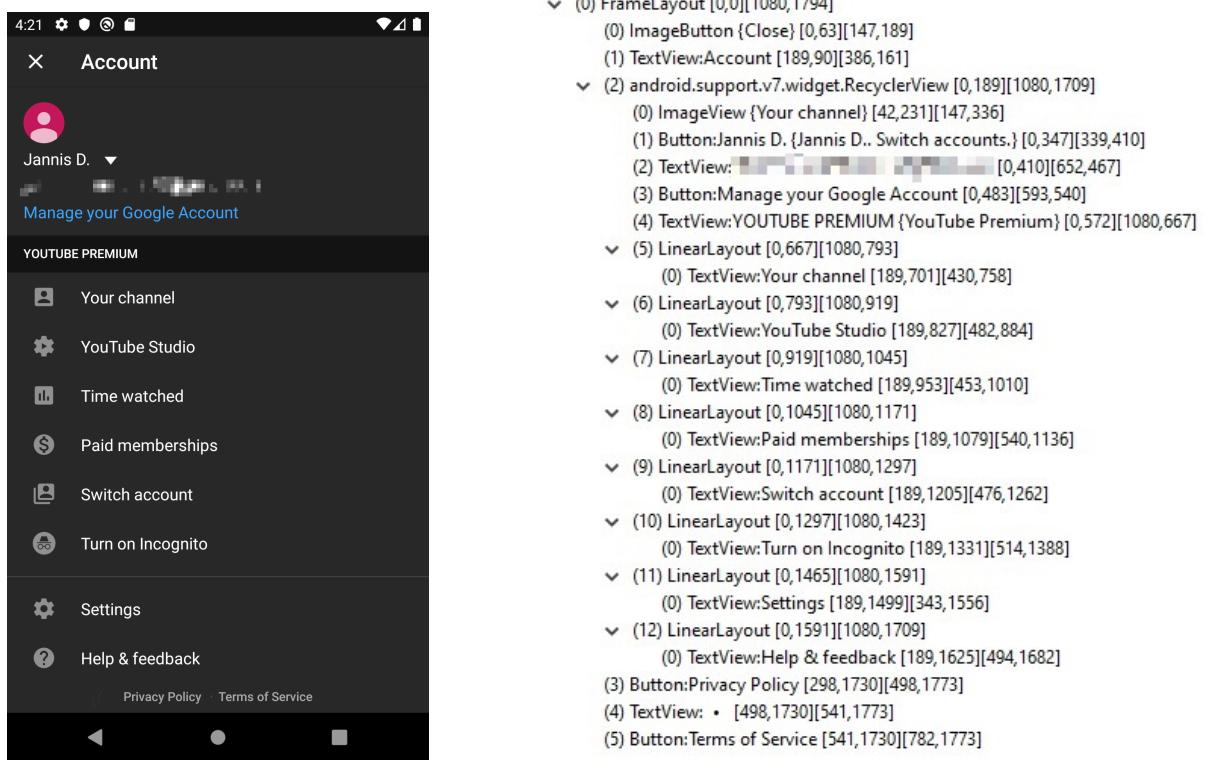


Abbildung 5.2: Abstrahierte Hierarchie der Youtube-Menüansicht

5.1.2 Wichtige Eigenschaften der GUI-Elemente

Die für die Automatisierung des Verfahrens in dieser Arbeit interessanten Eigenschaften der beiden Arten von GUI-Elementen sind die Attribute, die ebenfalls in dem bereits genannten XML-Standard festgelegt werden. Es gibt einige gemeinsame Eigenschaften, die *View*- und *ViewGroup*-Objekte teilen. Hierzu ist vor allem das boole'sche Attribut ***Clickable*** zu nennen, welches angibt, ob ein *View*-Objekt durch eine Touch-Interaktion des Benutzers betätigt werden kann [Doc20j]. Bei der dynamischen Analyse der Anwendung sind Objekte mit dem Wert TRUE für diese Eigenschaft von besonderer Bedeutung, da dies die Elemente sind, mit denen interagiert werden kann. Ein weiteres Attribut mit hoher Relevanz ist die ***ContentDescription***. Dies ist ein Text, den Entwickler*innen einem Objekt zuweisen können, welcher den Inhalt des Views in Worten beschreibt [Doc20j]. Dies ist vor allem bei Objekten relevant, die Piktogramme darstellen. Beispielhaft hierfür sind die in Abbildung 4.1 zu sehenden Navigationsbuttons der Anwendung „Facebook“ (Interaktionen **A** und **B**).

Besondere Formen der *ViewGroup*-Elemente sind die Klassen SCROLLVIEW und HORIZONTALSCROLLVIEW. Diese erweitern beide das FRAMELAYOUT und ermöglichen es, eine *ViewGroup* wie beispielsweise ein LINEARLAYOUT als Objekt unter sich in der Hierarchie zu halten, durch das der*die Benutzer*in dann scrollen kann [Doc20g][Doc20d]. Wie der Name schon sagt, besteht der Unterschied zwischen den beiden Klassen in der Orientierung des Scrollens. Da *ViewGroup*-Objekte bis auf das Scrollen lediglich für die Anordnung der *View*-Objekte auf dem Bildschirm des Endgeräts zuständig sind, werden sie in dieser Arbeit nicht näher betrachtet. Bei der Suche nach den Datenschutzhinweisen in der Anwendung ist einzig die Eigenschaft des Scrollens relevant. Somit werden nun noch einige Eigenschaften der *View*-Objekte thematisiert.

Die in diesem Absatz getätigten Bemerkungen beziehen sich auf die Online-Dokumentation für Android-Entwickler*innen bezüglich der View-Klasse [Doc20j]. Als für diese Arbeit interessante Attribute sind zum einen die Eigenschaft **Text**, welche für einen TEXTVIEW den dargestellten String beinhaltet, und zum anderen das Integer-Attribut der **Visibility** zu nennen, welches spezifiziert, ob ein View-Objekt auf dem Bildschirm sichtbar, unsichtbar oder nicht vorhanden ist. Ein unsichtbares Element nimmt Platz in der GUI ein, während ein nicht vorhandenes Objekt keinen Platz einnimmt. In dieser Arbeit sind nur die Objekte von Bedeutung, die für den*die Nutzer*in sichtbar sind (VISIBILITY=0).

Weitere Grundlagen bezüglich der Android GUI sowie deren technischen Aufbaus sind nicht für die Zwecke dieser Arbeit relevant und werden aus diesem Grund nicht betrachtet.

5.2 GUI Ripping

Die in dieser Arbeit verwendete Methodik zur Automatisierung der Suche nach Datenschutzhinweisen ist an das GUI Ripping angelehnt. Hierbei handelt es sich um ein Verfahren der dynamischen Analyse, also einer Testtechnik, welche eine Anwendung während ihrer Laufzeit betrachtet [Aca14]. Da es sich um eine Art des Testens von Software handelt, wird GUI Ripping klassischerweise dazu genutzt, Fehler in der Anwendung zu finden [MBN03]. Wie der Name der Methodik bereits nahelegt, wird auf Fehler in der Benutzerschnittstelle getestet. Durch GUI Ripping wird „mittels Reverse Engineering ein Modell des Arbeitsflusses der Anwendung abstrahiert“ [Mem+13]. Dieses kann darauffolgend zum Generieren automatischer Testfälle genutzt werden [MBN03]. Somit gehört diese Variante des Testens von Benutzerschnittstellen zu modellbasierten Ansätzen des Testens [Gue16].

Beim GUI Ripping wird im Normalfall jede Ansicht der Anwendung als ein Zustand in einem Zustandsgraphen angesehen. Dieser Zustandsgraph oder auch „GUI Tree“ [Mem+13] repräsentiert das Modell der gesamten GUI der Anwendung. In einem Zustand werden alle Objekte, die sich in der jeweiligen Ansicht der GUI befinden, zusammen mit ihren Eigenschaften (Position auf dem Bildschirm, Farbe, Clickable, etc.) sowie deren zugehörigen Werten gespeichert [Gue16]. Durch das automatische Ausführen von Benutzerinteraktionen wird von einem Zustand in den nächsten gewechselt und der nächste Zustand kann wieder gespeichert werden [Gue16]. In welcher Reihenfolge die Benutzerinteraktionen gewählt und ausgeführt werden, kann entweder zufällig oder nach einer bestimmten Heuristik geschehen. Amalfitano et al. haben für ihr Verfahren des GUI Ripping für Android-Anwendungen beispielsweise die Explorationsstrategien der Tiefensuche sowie Breitensuche gewählt [Ama+12a].

Hier ist anzumerken, dass die erhöhte Komplexität heutiger Anwendungen es erschwert, einen kompletten GUI Tree für die Anwendungen zu abstrahieren. Zum Beispiel kommt in vielen Social-Media-Anwendungen ein News Feed zum Einsatz, in dem wiederholt nach unten gescrollt werden kann und der somit einen endlosen Pfad in dem GUI Tree darstellen würde. Dies könnte auch in der Anwendung „Facebook“ in der Ansicht passieren, die in Abbildung 4.1 zu sehen ist. Somit ist es wichtig, ein Kriterium für den Abschluss der Suche festzulegen, wie zum Beispiel eine maximale Tiefe der Suche [Ama+12a].

Die Übergänge oder Kanten von einem Zustand in den nächsten stellen die Benutzerinteraktionen dar, die automatisiert (durch eine Explorationsstrategie) ausgeführt werden [Gue16]. Als eine vereinfachte grafische Darstellung eines GUI Trees ist der bereits behandelte und in Abbildung 4.10 gezeigte Graph zu nennen.

Zusammenfassend wird bei der Methodik des GUI Ripping die Benutzerschnittstelle automatisiert durchlaufen, indem alle möglichen Benutzerinteraktionen ausgeführt werden, und so ein Modell der GUI entwickelt, welches alle Ansichten der Anwendung enthält [MBN03]. Diese Grundlagen sollen im Folgenden helfen, die Automatisierung des im Fokus dieser Arbeit stehenden Verfahrens besser verstehen zu können.

6 Automatisierung des Verfahrens

Dieses Kapitel behandelt die softwaretechnisch automatisierte Variante des Verfahrens, welche unter dem Namen *DataPolicyRipper* für diese Arbeit entwickelt wurde. Zuerst wird die *Verwendete Software* vorgestellt. Anschließend wird sowohl die *Funktionsweise von DataPolicyRipper* als auch die *Durchführung von DataPolicyRipper* anhand eines Beispiels ausführlich behandelt. Somit dient dieser Abschnitt der Arbeit hauptsächlich der Erklärung des Programms *DataPolicyRipper*, welches für die im nächsten Kapitel thematisierten *Ergebnisse der Analyse* genutzt wird.

6.1 Verwendete Software

Zur Implementierung von *DataPolicyRipper* wird die offizielle integrierte Entwicklungsumgebung (IDE) für die Android-Plattform genutzt: Android Studio [Doc20f]. Diese IDE bietet zusätzlich zu einem klassischen Code-Editor und Debug-Funktionen weiterhin einen Emulator für Android-Geräte [Doc20f]. Dieser Emulator ermöglicht es, Anwendungen aus dem Google Play Store zu installieren und dient dazu, das Programm *DataPolicyRipper* auszuführen ohne ein separates Gerät zu benötigen.

Wie im vorangegangenen Kapitel beschrieben, handelt es sich beim GUI Ripping um ein dynamisches Testverfahren für Anwendungen. Somit wird das Verfahren als instrumentierter Test einer ansonsten funktionslosen mobilen Anwendung verwirklicht und in der Programmiersprache Java geschrieben. Eine testgetriebene Entwicklung durch beispielsweise den Test-First-Ansatz [Pro20] ist aus diesem Grund für *DataPolicyRipper* nicht möglich gewesen. Da es sich bei dem Verfahren um einen Test handelt ist das Testen der Software nur sehr begrenzt möglich. Ihre Korrektheit wird durch die Analyseergebnisse im Kapitel *Ergebnisse der Analyse* evaluiert.

Die offizielle Dokumentation für Entwickler*innen für die Android-Plattform bezeichnet instrumentierte Testverfahren als Tests, die auf einem physischen oder emulierten Gerät ausgeführt werden [Doc20b]. Somit werden instrumentierte Tests dazu genutzt, das Verhalten einer Anwendung im Kontext der Bedienung auf einem laufenden Gerät dynamisch zu überprüfen. Für solche Arten von Tests von Android-Anwendungen werden zahlreiche Frameworks bereitgestellt. Im Folgenden werden einige der verschiedenen Möglichkeiten des instrumentierten Testens für die Android-Plattform vorgestellt und im Anschluss die für diese Arbeit gewählte Implementierung behandelt.

Das offizielle Tool „Espresso“ ermöglicht es Entwickler*innen, für eine einzelne Anwendung Benutzerinteraktionen zu simulieren und so die GUI zu testen [Doc20i]. Dieses Framework benötigt jedoch die konkreten Namen für die jeweiligen View-Objekte und *Aktivitäten* und stellt somit einen White-Box-Test dar [Doc20i]. Dies bedeutet, dass „Espresso“ für die Zwecke dieser Arbeit nicht genutzt werden kann, da der Quellcode der Anwendungen, die getestet werden sollen, nicht frei zugänglich ist.

Weiterhin wird von Android das Tool „App Crawler“ angeboten [Doc19a]. Hierbei handelt es sich um ein Programm, welches automatisch den Zustandsraum der Anwendung erforscht, indem es Benutzerinteraktionen simuliert, bis keine weiteren Interaktionen möglich sind oder eine bestimmte Zeit überschritten ist [Doc19a]. Dieses Tool eignet sich trotz dieser Funktionen nicht für die automatisierte Suche nach Datenschutzhinweisen, da das Tool nur sehr beschränkt anpassbar ist und lediglich die grundsätzliche Funktionalität der GUI einer Anwendung testen soll [Doc19a].

Die im Kapitel *Verwandte Literatur* genannten Arbeiten von Amalfitano *Using GUI ripping for automated testing of Android applications* und *A toolset for GUI testing of Android applications*, welche Implementierungen des GUI Ripping für Android-Anwendungen vorstellen, nutzen „Robotium“ [Ama+12a][Ama+12b]. Hierbei handelt es sich um ein Open Source Framework für die Automatisierung von Tests von Android-Anwendungen [Rad16b]. Das Framework unterstützt das Black-Box-Testen von Anwendungen, deren Quelltext nicht verfügbar ist und funktioniert für alle Arten von mobilen Anwendungen [Rad16b]. Ebenso ist die benötigte Funktionalität für das GUI Ripping durch die genannten Quellen erprobt. Nachteilig ist, dass für die Nutzung von „Robotium“ das ANDROID PACKAGE(APK) [Doc19b] der zu testenden Anwendung zur Verfügung stehen muss [Rad16a]. Dieses muss somit im Voraus heruntergeladen werden und der Google Play Store kann nicht zum Herunterladen der Anwendungen auf das Gerät genutzt werden. Weiterhin kann ausschließlich eine einzige Anwendung durch das Framework getestet werden [Rad16a]. Dies bedeutet, dass für das Testen mehrerer Anwendungen für jede Anwendung ein eigenes Projekt in der Entwicklungsumgebung angelegt werden muss [Rad16a]. Für die automatisierte Suche nach Datenschutzhinweisen in mehreren Anwendungen hintereinander ist das Framework „Robotium“ nicht geeignet.

6.1.1 UI Automator

Im vorangegangenen Kapitel wurde bereits das Programm „UI Automator Viewer“ genutzt, um die XML-Hierarchie einer Ansicht während der Laufzeit einer Anwendung zu abstrahieren und anzuzeigen. Dieses Werkzeug ist Teil des offiziell von Android bereitgestellten Frameworks „UI Automator“ [Doc19f]. Dieses wird zur Implementierung des in dieser Arbeit vorgestellten Verfahrens *DataPolicyRipper* genutzt. Dieser Abschnitt wird die Grundkonzepte und Programmierschnittstellen des Werkzeuges erläutern.

Die Bemerkungen der nächsten beiden Absätze beziehen sich auf die offizielle Entwicklerdokumentation des Frameworks [Doc19f]. „UI Automator“ ermöglicht es, automatisierte Tests für die Android-Plattform zu schreiben, die sowohl die Funktionalität von Anwendungen überprüfen als auch Interaktionen in System-Anwendungen des Betriebssystems ausführen. Hierfür werden keine näheren Kenntnisse über die Implementierung der jeweiligen zu testenden Anwendung benötigt. Da mehrere Anwendungen in einem einzelnen Testfall getestet werden können, eignet sich das Framework ebenfalls für das Durchsuchen mehrerer Anwendungen hintereinander.

Zum Ausführen der Tests muss ein Objekt vom Typ **UiDevice** instanziert werden. Über dieses kann auf das Gerät zugegriffen werden, auf dem der Test läuft. Weiterhin wird dieses genutzt, um die drei Softkeys des Betriebssystems Android zu betätigen, wie sie beispielsweise in den Screenshots aus der Anwendung Facebook in den Abbildungen 4.1 bis 4.3 in der unteren Leiste der Abbildungen zu sehen sind. Die Programmierschnittstelle (API) **UiObject** repräsentiert ein Element der GUI. Dies können sowohl Objekte vom Typ **View** als auch vom Typ **ViewGroup** sein, wie sie im Abschnitt *Technischer Aufbau* beschrieben wurden. Mit diesen kann über die

API interagiert werden und es ist möglich, Informationen über ihre Eigenschaften zu erhalten. Für GUI-Container der Art *ViewGroup*, mit denen durch Scrollen interagiert wird, bietet das Framework die API ***UiScrollable***. Diese ermöglicht das Scrollen durch die jeweiligen GUI-Container. Eine weitere wichtige API beinhaltet Methoden, über die eine Instanz eines *UiDevice* nach GUI-Elementen durchsucht werden kann. Dies ist die Schnittstelle ***UiSelector***, welche Abfragen nach bestimmten Attributen der GUI-Elemente ausführen kann und Objekte der Typen *UiObject* und *UiScrollable* instanziieren kann.

Dies sind alle für das Verständnis der folgenden Abschnitte zur Erklärung der *Funktionsweise von DataPolicyRipper* benötigten Grundlagen zu der für die Automatisierung der Suche nach Datenschutzhinweisen verwendete Software.

6.2 Funktionsweise von *DataPolicyRipper*

Die Grundlegende Funktionsweise des automatisierten Verfahrens ist an das bereits beschriebene GUI Ripping angelehnt. Jedoch ist es für die Suche nach Datenschutzhinweisen nicht nötig, das gesamte Modell der GUI zu abstrahieren. Lediglich der Pfad im Modell, der zu den Datenschutzhinweisen führt, ist relevant. Um sowohl Rechenzeit als auch Speicherbedarf einzusparen, wird das Modell der GUI, beziehungsweise der Zustandsraum der Suche, zur Laufzeit des Verfahrens generiert. Die Suche wird für die Zwecke dieser Arbeit als Suche in einem Graphen betrachtet, wie sie von Poole und Mackworth in deren Buch zu den Grundlagen der *Artificial Intelligence* beschrieben wird [PM10]. Das in Abbildung 6.1 gezeigte Code-Beispiel implementiert eine Breitensuche, welche das Fundament für die Suche in der Automatisierung des Verfahrens bildet. Der Pseudo-Code ist an den von Poole und Mackworth vorgestellten Generischen Suchalgorithmus angelehnt und auf die Breitensuche angepasst [PM10, S.79].

Die Eingabeparameter für die Suche sind in den Zeilen 3 bis 6 aufgeführt. Dies sind der Graph $G(N, E)$ mit Knoten ***N*** und Kanten ***E***, in dem die Suche ausgeführt wird, ein Startknoten ***S***, von dem aus die Suche ausgeführt wird, eine Zielfunktion ***goal***, die angibt, ob ein Knoten ein Zielknoten ist, sowie die maximale Tiefe, bis zu der gesucht werden soll. Lokal wird die Breitensuche ausgeführt, indem alle zu durchsuchenden Pfade in einer Liste gespeichert werden, der ***Frontier***. Diese besteht zum Anfang der Suche aus einem Pfad, der lediglich den Startknoten enthält. Der Hauptbestandteil der Prozedur besteht aus einer do-while-Schleife, die ausgeführt wird, solange die *Frontier* Pfade enthält und die aktuelle Tiefe nicht die maximale Tiefe überschritten hat [Zeile 22].

In den Zeilen 13 bis 21 ist der Körper der Schleife zu sehen. Die *Frontier* ist bei einer Breitensuche als eine Warteschlange nach dem First-In-First-Out-Prinzip (FIFO) implementiert [PM10, S.84]. Dies bedeutet, dass der Pfad als erstes betrachtet wird, der am frühesten der *Frontier* hinzugefügt wurde [Zeile 14]. Dieser Pfad $P=(S, \dots, n)$ wird aus der *Frontier* entfernt und es wird überprüft, ob es sich bei dem letzten Knoten des Pfades ***n*** um einen Zielknoten handelt. Ist dies der Fall, wird ***P*** von der Suche als Lösung ausgegeben und die Suche wird beendet [Zeile 16-18]. Ansonsten wird ***n*** durchsucht, indem für jeden Kindesknoten ***c*** von ***n*** ein Pfad $P_c=(S, \dots, n, c)$ der *Frontier* hinzugefügt wird [Zeile 19-21]. Die aktuelle Tiefe ist die jeweilige Länge des aktuell zu durchsuchenden Pfades ***P*** [Zeile 15]. Ist diese größer als die maximal erlaubte Tiefe oder wurden alle Knoten durchsucht, die *Frontier* ist demnach leer, wird die Suche abgebrochen und NULL als Ergebnis ausgegeben [Zeile 22-23].

```

1   procedure Breadth-First-Search(G, S, goal, MaxDepth)
2     Inputs
3       G: Graph mit Knoten N und Kanten E
4       S: Startknoten
5       goal: Boolesche Ziel-Funktion
6       MaxDepth: Tiefe, bis zu der gesucht wird
7     Output
8       Kürzester Pfad von S zu einem Zielknoten, sonst null
9     Local
10    Frontier: Liste von Pfaden
11    Depth: Aktuelle Tiefe
12    Frontier = { (S) }
13    do {
14      select and remove first path (S, ..., n) from Frontier
15      Depth = Length of (S, ..., n)
16      if goal(n) {
17        return (S, ..., n)
18      }
19      for all children c of n in G {
20        add (S, ..., n, c) to Frontier
21      }
22    } while (Frontier != null || Depth <= MaxDepth)
23    return null

```

Abbildung 6.1: Pseudo-Code für Breitensuche in Graphen

Die Breitensuche wurde als Implementierung für diese Arbeit gewählt, da diese in jedem Fall den kürzesten Pfad vom Startknoten bis zu einem Zielknoten findet, sollte mindestens ein Zielknoten existieren [PM10, S.86]. Ausgehend von diesem Pseudo-Code der Breitensuche werden für die Automatisierung des Verfahrens in *DataPolicyRipper* in dieser Arbeit vier implementierte Komponenten beziehungsweise Klassen benötigt. Hierzu gehören die beiden Bestandteile des Graphen **G(N,E)**, die Knoten **N** und die Kanten **E**. Weiterhin gehören noch die an den beschriebenen Pseudo-Code angelehnte Klasse für die Breitensuche sowie eine Klasse, die für das Instanziieren und Starten der Suche in mehreren Anwendungen zuständig ist, zum *DataPolicyRipper* dazu. Im Folgenden wird auf jede dieser vier Komponenten eingegangen und es werden einige Besonderheiten ihrer Implementierungen aufgeführt.

6.2.1 Komponente: Kanten

In Abbildung 6.2 ist ein Ausschnitt des Klassendiagramms der Anwendung gezeigt, welches die für die Funktionalität einer Kante benötigten Klassen abbildet. Die Notation der in dieser Arbeit abgebildeten Klassendiagramme folgt der Unified Modelling Language (UML) der Version 2.5 [Inn13]. Konstruktoren von Klassen sind in Blau gekennzeichnet. Methoden mit dem Rückgabetyp VOID in Java werden ohne Rückgabewert angegeben.

Eine Kante in dem Graphen, der die GUI einer Anwendung repräsentiert, wird durch die abstrakte Klasse **AbstractEdgeInteraction** in *DataPolicyRipper* implementiert. Kanten in dem Graphen sind die Interaktionen, die in einer Ansicht der GUI ausgeführt werden können. Zu den Feldern der Klasse gehört ein Integerwert, der das Level der Präferenz der Kante angibt (**preference**). Je höher der Wert, desto eher wird die Kante durchsucht. Präferierte Kanten sind beispielsweise Interaktionen, welche Schlüsselwörter als *Text* oder *ContentDescription* haben,

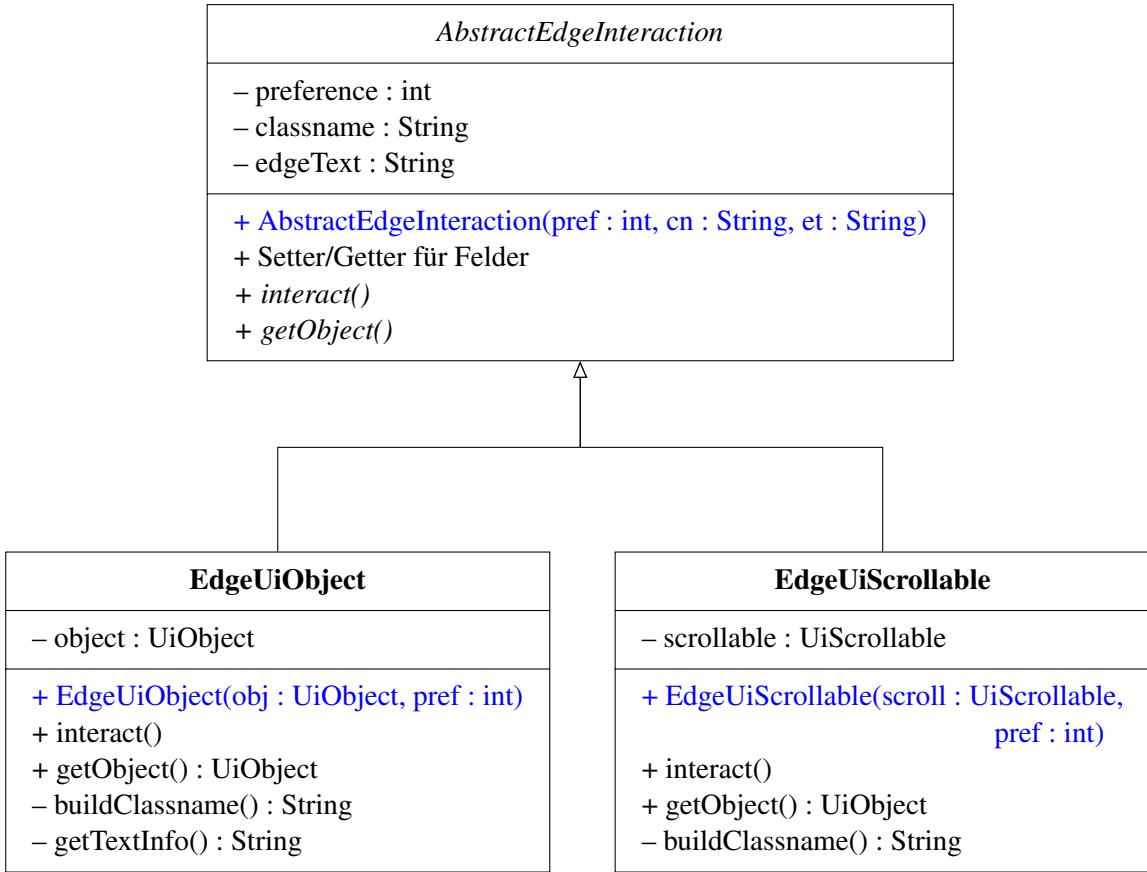


Abbildung 6.2: Abbildung der Komponente **Kante** als UML-Klassendiagramm

die wahrscheinlich zu Datenschutzhinweisen führen könnten. Auf die Werte der Präferenz und die Schlüsselwörter wird im nächsten Abschnitt *Komponente: Knoten* näher eingegangen. Weiterhin hält die Klasse Informationen über den Typ des GUI-Elementes als String. Dies ist der Klassenname (**classname**) der Art von *View* oder *ViewGroup*, der das Objekt angehört. Das Feld **edgeText** vom Typ String beinhaltet den Text oder die *ContentDescription* einer Interaktion, sollte das zugehörige GUI-Element eines der beiden Attribute besitzen.

Neben Methoden, die für das Setzen und Ausgeben der Werte der Felder der Klasse zuständig sind (**Setter/Getter**), definiert die Klasse *AbstractEdgeInteraction* zwei abstrakte Methoden. Dies sind die Methode **interact()**, welche die eigentliche Interaktion mit dem GUI-Element realisiert, und ein *Getter* für ein *UiObject*.

Als Interaktion in der GUI werden die zwei Möglichkeiten Klicken und Scrollen beachtet. Für Elemente der GUI, mit denen durch Klicken interagiert werden kann, werden Objekte vom Typ *UiObject* genutzt, während für das Scrollen Objekte vom Typ *UiScrollable* zum Einsatz kommen. Demnach wird für beide Arten der Interaktion jeweils eine Erweiterung der abstrakten Klasse benötigt. Die Klasse **EdgeUiObject** erweitert die abstrakte Überklasse für Kanten, die ein *UiObject* halten, während die Klasse **EdgeUiScrollable** dies für Kanten der Interaktion Scrollen tut. Diese hält demnach als Feld ein Objekt vom Typ *UiScrollable*. Beide Arten der erbenden Klassen werden über das Objekt der jeweiligen Interaktion sowie einen Integer instanziert. Der Integer-Wert gibt, wie oben beschrieben, die Präferenz an.

Beim Aufruf des Konstruktors beider Klassen werden über den Konstruktor der übergeordneten, vererbenden Klasse deren Felder initialisiert. Für das Instanziieren des Strings *classname*

nutzen beide Komponenten die Methode ***buildClassname()***, welche den Typen des jeweiligen *UiScrollables* oder *UiObjects* ermittelt. Zusätzlich wird in *EdgeUiObject* durch die Methode ***getTextInfo()*** der String für das Feld *edgeText* der übergeordneten Klasse gesetzt. Ein *UiScrolable* beinhaltet weder einen Text noch eine *ContentDescription* und somit wird das Feld hier als String mit dem Wert "SCROLL" interpretiert.

Beide erbenden Klassen müssen die abstrakten Methoden der vererbenden Klasse implementieren. Demnach sorgt die Methode zum Ausführen der Interaktion, je nach Typ der Klasse, für einen Klick auf das *UiObject* oder das Scrollen innerhalb eines *UiScrollables*. Die Methode ***getObject()*** gibt in der Klasse *EdgeUiObject* das gespeicherte *UiObject* aus, während in der Klasse *EdgeUiScrolable* der Rückgabewert NULL zurückgegeben wird. Diese Methode dient dem Vergleich der verschiedenen Kanten, welche in der Komponente **Knoten** genutzt wird.

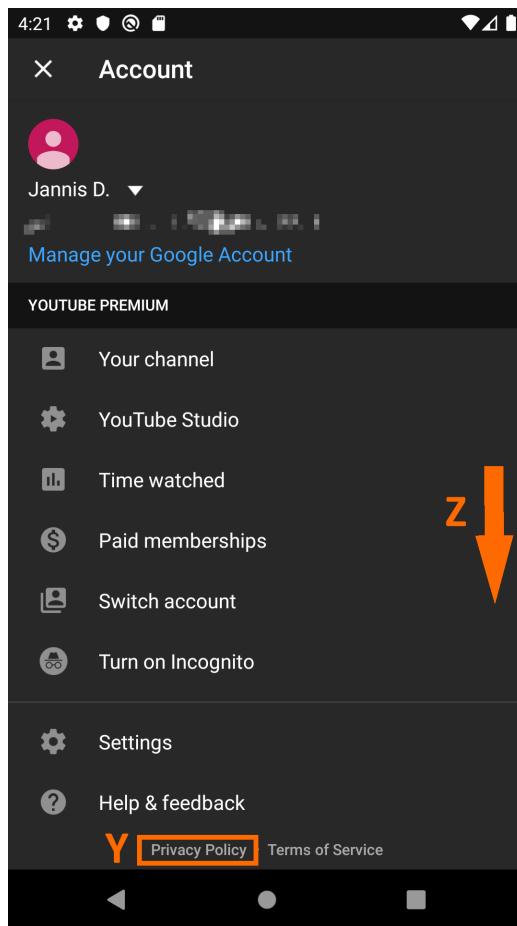


Abbildung 6.3: Ansicht aus der Anwendung „Youtube“ zur Verdeutlichung der Konzepte

In Abbildung 6.3 ist eine Ansicht aus der Anwendung „Youtube“ zu sehen. Hier ist als Beispiel die Interaktion **Y** markiert. Anhand dieser Interaktion soll nun nochmals das Konzept der Kante in *DataPolicyRipper* beispielhaft verdeutlicht werden. Da es sich bei der Interaktion nicht um ein Scrollen handelt, wird diese durch ein *EdgeUiObject* realisiert. Das Feld ***object*** der Klasse ist hier das *UiObject*, welches den Button repräsentiert. Dieses wird bei der Initialisierung der Klasse zusammen mit einer Präferenz als Integer-Wert übergeben und die Felder der abstrakten Klasse werden entsprechend initialisiert. Über die Methode *buildClassname()* wird das String-Feld *classname* der Überklasse ermittelt. Dies ist in diesem Fall der String "BUTTON". Das Feld *edgeText* ist im Falle der Interaktion **Y** der String "PRIVACY POLICY" und wird mithilfe der

privaten Methode `getTextInfo()` gesetzt. Die implementierte Methode `interact()` sorgt hier dafür, dass auf den Button **Y** geklickt wird.

Als weiteres hypothetisches Beispiel ist die Interaktion **Z**, das Scrollen in der Ansicht, angegeben. Diese Interaktion würde durch die Klasse `EdgeUiScrollable` implementiert werden. Das übergeordnete `FRAMELAYOUT` der Anwendung könnte beispielsweise ein `UiScrollable` sein und wäre somit das Objekt für das Feld **scrollable**. Der String für das Feld `classname` würde über die private Methode `buildClassname()` ermittelt werden und wäre der Klassenname der Interaktion ("FRAMELAYOUT"). Für den `edgeText` würde im Falle eines `EdgeUiScrollables` ein String mit Wert "SCROLL" übergeben werden. Die Methode `getObject()` würde NULL zurückgeben und die Methode `interact()` würde für das Scrollen in dem `UiScrollable` sorgen.

6.2.2 Komponente: Knoten

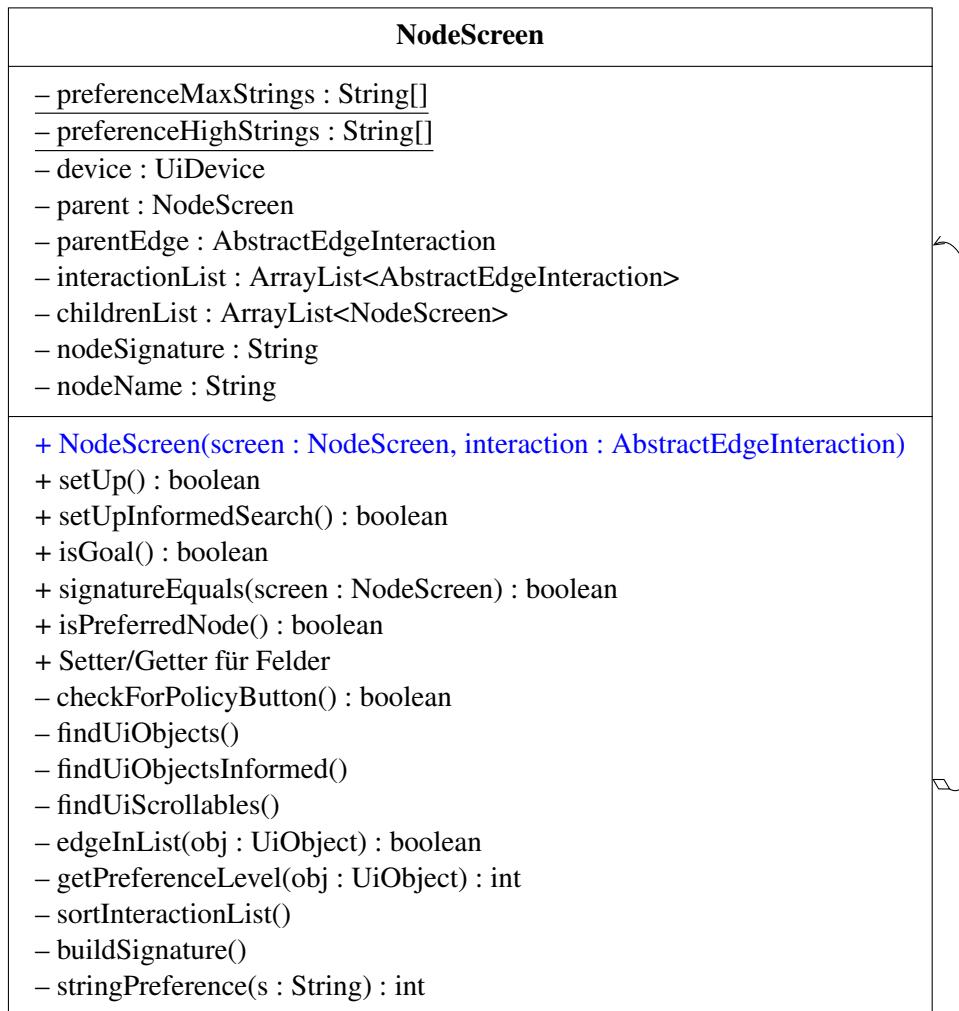


Abbildung 6.4: Abbildung der Komponente **Knoten** als UML-Klassendiagramm

Die Komponente der Knoten des Graphen wird in der Klasse **NodeScreen** realisiert. Abbildung 6.4 zeigt die zugehörige Klasse aus dem UML-Klassendiagramm. Die Klasse realisiert sowohl die Speicherung der zur Suche benötigten Informationen über einen Knoten als auch die rekursive Instanziierung aller von diesem Knoten aus erreichbaren Kanten und Kindesknoten.

Ein Knoten in dem Graphen enthält Informationen darüber, welches sein Elternknoten ist sowie welche Kante vom Elternknoten aus zu diesem Knoten führt. Dies wird durch die beiden Felder **parent** und **parentEdge** realisiert, welche jeweils eine Instanz der Klasse *NodeScreen* oder ein Objekt vom Typ *AbstractEdgeInteraction* halten.

Weiterhin speichert ein Knoten Informationen darüber, welche Kanten von ihm aus zu welchen Kindesknoten im Graphen führen. Hierfür sind die beiden Listen **interactionList** und **childrenList** zuständig. Die *interactionList* beinhaltet alle Elemente des Typs *AbstractEdgeInteraction*, also alle Interaktionen, die in der jeweiligen Ansicht der GUI möglich sind. Das Feld *childrenList* besteht aus den jeweiligen Instanzen der Klasse *NodeScreen*, zu denen die jeweiligen Interaktionen aus der *interactionList* führen. Die *interactionList* wird beim Durchsuchen eines Knotens instanziert und im Nachhinein wird für jede Interaktion in der Liste ein neues *NodeScreen*-Objekt in der *childrenList* initialisiert. Zwei weitere Felder der Klasse mit Typ String halten zum einen Informationen über den Namen oder die Bezeichnung des Knotens (**nodeName**) und zum anderen eine Signatur, welche eine String-Repräsentation eines Knotens darstellt (**nodeSignature**). Das *UiDevice* **device** wird dafür benötigt, Zugriff auf die GUI-Elemente des Gerätes zu erhalten und nach diesen in der Ansicht zu suchen.

Die beiden statischen Felder **preferenceMaxStrings** und **preferenceHighStrings** sind Arrays aus Strings, welche im *Text* oder in der *ContentDescription* von Interaktionen enthalten sein könnten, die zu Datenschutzhinweisen führen. Interaktionen, die diese Strings in den beschriebenen Attributen enthalten, werden bevorzugt untersucht. Allgemein wird zwischen vier Stufen in der Präferenz von Interaktionen unterschieden. Die höchste Stufe (**Maximal**) sind Interaktionen, die direkt zu Datenschutzhinweisen führen sollten. Darauf folgend sind Interaktionen, die in bereits untersuchten Anwendungen auf dem Pfad zu den Datenschutzhinweisen lagen (**Hoch**). Als nächstes werden Kanten des Typs *EdgeUiScrollable* präferiert (**Mittel**) und die niedrigste Stufe bilden alle restlichen Interaktionen (**Niedrig**). Aus den im Kapitel *Ausführung per Hand* getesteten Anwendungen wurden die in Tabelle 6.1 aufgeführten Stichworte für die Präferenzstufen Maximal und Hoch entnommen.

Präferenz	Stichworte
Maximal	Data Policy, Privacy Policy, Privacy Statement, Policies
Hoch	Menu, Profile, Settings, Account, Privacy, Help, Options, About, Navigation, Drawer, Legal Information, Legal, Me

Tabelle 6.1: Stichworte für die Präferenzstufen Hoch und Maximal

Ein Objekt der Klasse *NodeScreen* wird mit zwei Parametern instanziert. Hierbei handelt es sich um den *NodeScreen*, welcher der Elternknoten des zu instanzierenden Knoten ist, sowie die Interaktion, die vom Elternknoten aus zu dem Objektknoten führt. Neben Setter- und Getter-Methoden für die relevanten Felder der Klasse bietet ein *NodeScreen* einige öffentliche, von anderen Klassen nutzbare Methoden. Dies sind alles Methoden mit einem boole'schen Rückgabewert. Die Prozedur **isPreferredNode()** ermittelt, ob es sich bei dem Knoten um einen präferierten Knoten handelt.

Eine weitere Methode überprüft, ob sich ein weiterer Knoten mit der aktuellen Instanz der Klasse gleicht (**signatureEquals(screen: NodeScreen)**). Hierfür wird das Feld der *nodeSignature* mit der Signatur des übergebenen Knotens verglichen. Stimmen die beiden Signaturen überein, werden die Knoten als gleich betrachtet. Dieses Verfahren ist relevant für das Vermeiden von

```

1  private void buildSignature() throws UiObjectNotFoundException {
2      nodeSignature = "";
3      UiObject textViewName = device.findObject(new
4          UiSelector().className("android.widget.TextView")
5          .clickable(false).instance(0));
6
7      if (textViewName != null) {
8          nodeName = textViewName.getText();
9          nodeSignature = nodeSignature.concat(nodeName + ";");
10     }
11     for (AbstractEdgeInteraction e: interactionList) {
12         if (nodeSignature.length() <= 500) {
13             nodeSignature = nodeSignature.concat(e.getClassName()
14                 + ";" + e.getEdgeText() + ";");
15         }
16     }
17 }
```

Abbildung 6.5: Code-Ausschnitt aus Klasse **NodeScreen** zur Methode *buildSignature()*

Zyklen in den Pfaden der Suche, welches durch die Komponente: *Breitensuche* realisiert und demnach im nächsten Abschnitt näher betrachtet wird.

Die *nodeSignature* sowie der *nodeName* eines *NodeScreen* werden in der privaten Methode ***buildSignature()*** gebildet. Diese ist in Abbildung 6.5 als Code-Ausschnitt aus der Klasse zu sehen. Angefangen wird durch die Initialisierung der Signatur als leerer String [Zeile 2]. Das Feld *nodeName* repräsentiert eine Überschrift in der derzeitigen Ansicht. Um eine solche zu ermitteln, wird durch einen *UiSelector* in dem *UiDevice* nach der ersten Instanz eines *UiObjects* gesucht, welches dem Typen TEXTVIEW angehört und welches nicht durch Klicken betätigt werden kann [Zeile 3-5]. Existiert ein solches *UiObject*, wird der Text von diesem als *nodeName* des Knotens gesetzt sowie der Signatur des Knotens hinzugefügt [Zeile 7-10]. Weiterhin besteht die Signatur aus den Klassennamen und Beschreibungstexten aller Interaktionen, die von dem *NodeScreen* aus möglich sind. Somit wird eine Schleife genutzt, um für jede *AbstractEdgeInteraction* aus der Interaktionsliste des Knotens deren Beschreibungstext und Klassenname der Signatur hinzuzufügen [Zeile 11-15]. Die Signatur darf maximal eine Länge von 500 Zeichen besitzen [Zeile 12].

Die boole'sche Prozedur ***isGoal()*** gibt wieder, ob es sich bei einem *NodeScreen* um einen Zielknoten handelt. Hierfür wird, analog zu der in Abbildung 6.5 erklärten Ermittlung des Feldes *nodeName*, überprüft, ob sich ein *UiObject* vom Typ TEXTVIEW auf dem *UiDevice* befindet, welches die Stichworte „Data Policy“ oder „Privacy Policy“ als Text enthält. Hierbei handelt es sich jedoch nicht um eine Interaktion. Somit darf das *UiObject* nicht *Clickable* sein und befindet sich nicht in der *interactionList*.

Die beiden noch nicht behandelten öffentlichen Methoden der Klasse ***setUp()*** und ***setUpInformedSearch()*** führen das Durchsuchen eines Knotens durch. Der Unterschied zwischen ihnen wird im nächsten Unterkapitel erklärt. Beide Methoden nutzen mehrere der privaten Methoden der Klasse, um die möglichen Interaktionen des Knotens zu ermitteln und die Felder *interactionList* und *childrenList* zu initialisieren. Nach der Ermittlung der möglichen Interaktionen wird die Prozedur zum Sortieren der Interaktionsliste nach der Präferenz der Kanten aufgerufen(***sortInteractionList()***) und die Signatur des Knotens durch die bereits beschriebene Vorgehensweise gebildet.

```

1  private void findUiObjects() throws UiObjectNotFoundException {
2      UiObject obj;
3      int i;
4      findUiObjectsInformed();
5      i = 0;
6      do {
7          obj = null;
8          obj = device.findObject(new UiSelector()
9              .clickable(true).instance(i));
10
11         if (obj != null && obj.exists() && !edgeInList(obj)) {
12             int pref;
13             pref = getPreferenceLevel(obj);
14             interactionlist.add(new EdgeUiObject(obj, pref));
15         }
16         i++;
17     } while (obj != null && obj.exists());
18 }
```

Abbildung 6.6: Code-Ausschnitt aus Klasse **NodeScreen** zur Methode *findUiObjects()*

Die privaten Methoden zum Ermitteln der möglichen Interaktionen (**findUiObjects()**, **findUiObjectsInformed()**, **findUiScrollables()**) funktionieren nach einem ähnlichen Schema, das beispielhaft an der in Abbildung 6.6 abgebildeten Methode erklärt wird. Hierfür werden zwei lokale Variablen benötigt: eine Variable des Typen, nach dem gesucht werden soll (in diesem Fall *UiObject*) [Zeile 2] sowie einen Integer (*i*), welcher mit dem Wert 0 initialisiert wird [Zeile 3-5]. Der Hauptteil der Methodik besteht aus einer do-while-Schleife, welche ausgeführt wird, solange die Variable des gesuchten Objektes nicht NULL ist und das Objekt auf dem Gerät derzeit zu sehen ist [Zeile 16].

Im Körper der Schleife [Zeile 6-15] wird die Objektvariable auf den Wert NULL gesetzt und danach auf dem *UiDevice* durch einen *UiSelector* nach Objekten gesucht, die *Clickable* sind und die *i*-te Instanz dieses *UiSelectors* auf dem *UiDevice* sind [Zeile 8-9]. Sollte die gefundene Instanz des *UiObjects* nicht NULL und auf dem Gerät derzeit vorhanden sein, wird durch die Methode **edgeInlist(obj: UiObject)** überprüft, ob sich das Objekt bereits in der Interaktionsliste befindet [Zeile 11]. Ist dies nicht der Fall, wird die Präferenz für das Objekt ermittelt (**getPreferenceLevel(obj: UiObject)**) und der Interaktionsliste eine neue entsprechende Kante mit der ermittelten Präferenz hinzugefügt [Zeile 12-14]. Weiterhin wird die Integer-Variable inkrementiert, um im nächsten Durchlauf der Schleife eine mögliche weitere Instanz eines *UiObjects* zu finden, welche dem bereits beschriebenen *UiSelector* entspricht [Zeile 16].

Die beiden weiteren Methoden zum Ermitteln der möglichen Interaktionen funktionieren analog zu der eben beschriebenen Prozedur. *findUiObjectsInformed()* sucht lediglich nach präferierten Interaktionen, während *findUiScrollables()* Kanten vom Typ *EdgeUiScrollable* ermittelt. Die beiden bisher behandelten Komponenten der Kanten und Knoten des Graphen werden in der im nächsten Abschnitt beschriebenen Komponente von *DataPolicyRipper* dazu genutzt, die Suche nach Datenschutzanwendungen in einer einzelnen Anwendung durchzuführen.

Abbildung 6.7 zeigt einen Ausschnitt aus der Anwendung „Youtube“, der schon mehrmals in dieser Arbeit referenziert wurde. In diesem sind alle möglichen Interaktionen farblich markiert und auf der rechten Seite der Abbildung ist eine Legende zur Erklärung der genutzten Farben

zu finden. Im Folgenden werden die zuvor erklärten Konzepte der Klasse `NodeScreen` in der Anwendung `DataPolicyRipper` am Beispiel der Abbildung verdeutlicht.



Abbildung 6.7: Ansicht aus der Anwendung „Youtube“ zur Verdeutlichung der Konzepte

Ein Objekt der Klasse `NodeScreen`, welches die abgebildete Ansicht repräsentiert, wird beim Durchsuchen des Knotens `parent` instanziert, von dem aus es durch die Interaktion `parentEdge` erreicht wird. Beide Felder werden bei der Instanziierung übergeben und gesetzt. Der Name des Knotens wird, wie bereits beschrieben, im Feld `nodeName` als String gespeichert und durch die Methode `buildSignature()` ermittelt, welche in den Methoden `setUp()` und `setUpInformedSearch()` aufgerufen wird. In diesem Beispiel ist der Name des Knotens der String "ACCOUNT". Dies ist die Überschrift, die sich mittig oben in der Ansicht befindet.

In der Liste `interactionList` sind die in der Ansicht durch farbige Rechtecke oder Pfeile markierten Interaktionen gespeichert. Ein Pfeil repräsentiert eine Interaktion durch Scrollen. Die Farbe der Interaktionen in der Abbildung codiert ihre jeweilige Präferenz. Die Präferenzstufen sind in der Legende auf der rechten Seite der Abbildung aufgeführt. Das Durchsuchen eines Knotens durch eine der beiden Methoden `setUp()` und `setUpInformedSearch()` nutzt unter anderem die Methoden zur Initialisierung der `interactionList`. Die Methode `findUiObjectsInformed()` findet und initialisiert lediglich die Objekte vom Typ `EdgeUiObject` mit Präferenzlevel **Maximal** und **Hoch**. Durch `findUiObjects()` werden alle Objekte vom Typ `EdgeUiObject` gefunden und initialisiert (**Maximal**, **Hoch**, **Niedrig**). Die dritte Methode zur Instanziierung der Interaktionen, `findUiScrollables()`, ermittelt alle möglichen Interaktionen vom Typ Scrolling (`EdgeUiScrollable`) und instanziert diese mit Präferenz **Mittel**.

Im Anschluss an die Initialisierung aller Interaktionen in der *interactionList* wird beim Durchsuchen des Knotens die *interactionList* durch die Methode *sortInteractionList()* nach den Präferenzen sortiert und für jede Interaktion in der Liste wird ein Objekt vom Typ *NodeScreen* mit dieser Interaktion als *parentEdge* und dem aktuellen *NodeScreen*-Objekt als *parent* instanziert und in der *childrenList* gespeichert. Daraufhin wird die Signatur des Knotens durch die Methode *buildSignature()* gebildet. Diese besteht aus dem Namen des Knotens sowie den Feldern *edgeText* und *classname* der Interaktionen aus der *interactionList*. In diesem Fall würde die Signatur mit dem String "ACCOUNT;BUTTON;PRIVACY POLICY;TEXTVIEW;SETTINGS;BUTTON;MANAGE YOUR GOOGLE ACCOUNT;TEXTVIEW;SWITCH ACCOUNT;TEXTVIEW;HELP & FEEDBACK;弗AMELAYOUT;SCROLL;" beginnen. Dies sind als Beispiel die Klassennamen und Beschreibungstexte für die ersten sechs Interaktionen der *interactionList*. Der String würde für die weiteren Interaktionen mit der Präferenzstufe **Niedrig** weitergeführt werden.

6.2.3 Komponente: Breitensuche

Nachdem die beiden Komponenten des Programms *DataPolicyRipper* behandelt wurden, welche den Suchgraphen der Anwendung realisieren, wird nun die Komponente der Suche in diesem Graphen näher betrachtet. Diese wird durch die Klasse **BFSSearch** implementiert. Ein Klassendiagramm, das diese Komponente veranschaulicht, ist in Abbildung 6.8 gezeigt.

Ein Objekt der Klasse *BFSSearch* führt eine Breitensuche nach Datenschutzhinweisen in einer Anwendung aus, wie sie bereits anhand der Abbildung 6.1 erklärt wurde. Hierfür werden mehrere Felder benötigt. Der statische Integer-Wert **maxDepth** legt die maximale Tiefe fest, bis zu der im Graphen gesucht werden soll. Im Kapitel *Ausführung per Hand* wurden zehn verschiedene Anwendungen per Hand durchsucht. Die Länge der gefundenen Pfade (siehe Tabelle 4.1) betrug maximal sechs Interaktionen. Demnach ist die maximale Tiefe bei der Suche auf neun Interaktionen festgelegt, um noch Spielraum für längere Pfade zu erlauben.

Ein Objekt vom Typ *BFSSearch* wird mit einem *NodeScreen*, einem *UiObject* und einem boole'schen Wert initialisiert. Das *NodeScreen*-Objekt ist der Knoten, von dem aus die Suche ausgeführt wird, also das Feld **startNode**. Das übergebene *UiObject* bildet das Feld **appIcon** ab, welches der Button zum Ausführen der zu durchsuchenden Anwendung auf dem Startbildschirm des Gerätes ist. Das Feld **appPkg** vom Typ String speichert den Namen der zu durchsuchenden Anwendung. Der übergebene boole'sche Wert gibt wieder, ob es sich bei der Suche um eine heuristische oder uninformierte Suche handelt. Dieser wird im Feld **informedSearch** gehalten. Eine heuristische beziehungsweise informierte Suche beachtet lediglich die Kanten in der Anwendung mit den Präferenzstufen **Mittel**, **Hoch** und **Maximal**. Wird wiederum eine uninformierte Suche ausgeführt, dann werden alle von einer Ansicht aus möglichen Interaktionen durchsucht.

In der **frontier** werden analog zur bereits behandelten Breitensuche alle noch zu durchsuchenden Pfade gespeichert. Ein Pfad wird als ArrayList mit *NodeScreen*-Objekten realisiert. Die beiden Felder **currentPath** und **currentPathCopy** werden dazu genutzt, den derzeitigen Pfad zu speichern, um diesen zu durchsuchen und zu erweitern. In der Liste **visitedNodes** werden alle Knoten gespeichert, die bereits durchsucht wurden. Die Variable **currentInteraction** vom Typ *AbstractEdgeInteraction* hält die Interaktion, die aktuell durchsucht wird. Der boole'sche Wert **finished** gibt an, ob die Suche beendet ist.

Die Klasse benötigt weiterhin zwei Integer-Felder, welche dazu genutzt werden, die Position in der *frontier* zu speichern und zu aktualisieren, an der der erste Pfad mit der aktuell durch-

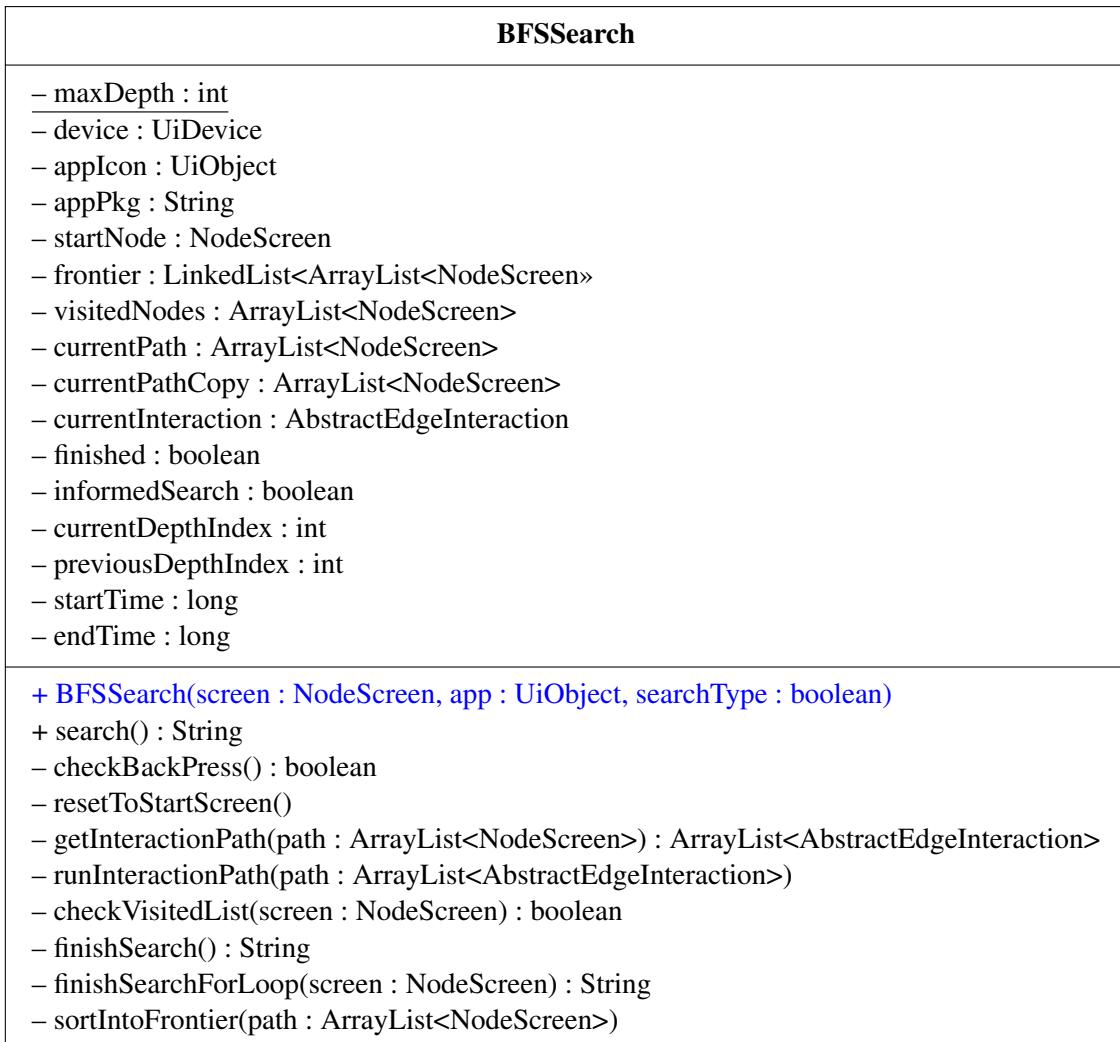


Abbildung 6.8: Abbildung der Komponente **Suche** als UML-Klassendiagramm

suchten Länge liegt (**currentDepthIndex**, **previousDepthIndex**). Dies wird genutzt, wenn ein Knoten durchsucht wird und dieser mindestens eine Interaktion mit hoher Präferenz in seiner Interaktionsliste hält. In diesem Fall wird der erweiterte Pfad so in die *frontier* eingesortiert, dass dieser der erste Pfad in der *frontier* mit der eigenen Länge ist. Die letzten beiden von der Klasse benötigten Felder sind vom Typ Long und werden für das Zeitmanagement der Suche genutzt (**startTime**, **endTime**). Diese wird maximal 30 Minuten ausgeführt. Nach Überschreiten der 30 Minuten wird die Suche abgebrochen.

Die Hauptlogik der Suche in einer Anwendung wird in der Methode **search()** ausgeführt. Als Rückgabetyp wird ein String ausgegeben, der das Ergebnis der Suche abbildet. Dieser String repräsentiert demnach den Pfad zu den Datenschutzhinweisen. Die Methode führt analog zur bereits behandelten Prozedur (Abbildung 6.1) eine Breitensuche aus. Diese ist an die bereits eingeführten Komponenten zur Suche nach Datenschutzhinweisen in einer Anwendung angepasst. Der Codeausschnitt in Abbildung 6.9 zeigt die for-Schleife aus der Methode **search()**, welche die angepasste Variante der Schleife aus den Zeilen 19 bis 21 aus der Abbildung 6.1 darstellt. Die Schleife führt die Erweiterung des aktuellen Pfades (*currentPath*) durch.

Für die Prozedur wird eine lokale Variable **lastItem** vom Typ *NodeScreen* benötigt, die den letzten Knoten auf dem aktuell zu durchsuchenden Pfad speichert [Zeile 1]. Die Schleife wird

einmal für jedes Kind aus der Kindesliste dieses letzten Knotens ausgeführt [Zeile 2]. Der Körper der Schleife beginnt mit der Initialisierung des Feldes *currentPathCopy* als Kopie des *currentPath*. Die derzeitige Interaktion (*currentInteraction*) befindet sich in der Interaktionsliste des *lastItem* an derselben Position, an der sich der derzeitige Kindesknoten in der Kindesliste befindet [Zeile 4-5]. Nach der Initialisierung wird mit dieser Interaktion interagiert [Zeile 6].

```

1   NodeScreen lastItem = currentPath.get(currentPath.length()-1);
2   for (NodeScreen child: lastItem.getChildrenList()) {
3       currentPathCopy = new ArrayList<NodeScreen>(currentPath);
4       currentInteraction = lastItem.getInteractionList()
5           .get(lastItem.getChildrenList().indexOf(child));
6       currentInteraction.interact();
7
8       if (informedSearch) {
9           if (child.setUpInformedSearch()) {
10               return finishSearchForLoop(child);
11           }
12       } else {
13           if(child.setUp()) {
14               return finishSearchForLoop(child);
15           }
16       }
17       currentPathCopy.add(child);
18       if (frontier.size() == 0 || currentPathCopy.size()
19 > frontier.get(currentDepthIndex).size()) {
20           previousDepthIndex = currentDepthIndex;
21           currentDepthIndex = frontier.size();
22       }
23
24       if (!informedSearch && child.isPreferredNode()) {
25           sortIntoFrontier(currentPathCopy);
26       } else {
27           frontier.add(currentPathCopy);
28       }
29       device.pressBack();
30       if (checkVisitedList(child)) {
31           frontier.remove(currentPathCopy);
32           currentDepthIndex = previousDepthIndex;
33       } else if (checkBackPress()) {
34           frontier.remove(currentPathCopy);
35           currentDepthIndex = previousDepthIndex;
36       } else {
37           visitedNodes.add(child);
38       }
39       resetToStartScreen();
40       runInteractionPath(getInteractionPath(currentPath));
41   }

```

Abbildung 6.9: Code-Ausschnitt aus Klasse **BFS****Search** zur Suche in einem Knoten

Im Folgenden wird das Objekt des Kindesknotens (**child**), welches bereits beim Durchsuchen des Elternknotens (*lastItem*) instanziert wurde, nach Interaktionen durchsucht. Dies funktioniert durch die im vorangegangenen Abschnitt *Komponente: Knoten* bereits beschriebenen Methoden *setUp()* und *setUpInformedSearch()*. Je nachdem, ob die derzeitige Suche informiert oder uninformativ durchgeführt wird, wird eine der beiden Methoden aufgerufen [Zeile 8-16]. Beide

dieser Methoden liefern einen boole'schen Rückgabewert. Wird eine Interaktion gefunden, welche zu Datenschutzhinweisen führt, wird TRUE zurückgegeben und die private Methode **finishSearchForLoop(screen : NodeScreen)** wird mit dem durchsuchten Knoten aufgerufen [Zeile 10, Zeile 14].

Die beiden Methoden **finishSearch()** und **finishSearchForLoop(screen : NodeScreen)** beenden die Suche, wenn Datenschutzhinweise gefunden wurden. Hierbei wird der Rückgabestring generiert, welcher den Pfad zu den Datenschutzhinweisen repräsentiert. Die Prozedur **finishSearch()** wird genutzt, wenn das Erreichen der Datenschutzhinweise durch einen Pfad erst außerhalb der for-Schleife erkannt wird [Abbildung 6.1: Zeile 16-18]. Der Rückgabestring wird aus den Texten aller Interaktionen des Zielpfades zusammengesetzt. Hat eine Interaktion keinen edge-Text, wird auf die Überschrift aus dem *nodeName* des entsprechenden *NodeScreen*-Objektes zurückgegriffen.

Nach dem Aufbereiten und Durchsuchen des Knotens *child* wird die Kopie des aktuellen Pfads um diesen Knoten erweitert [Zeile 17]. Ist der erweiterte Pfad länger als die bisherigen Pfade in der *frontier*, werden die Felder *currentDepthIndex* und *previousDepthIndex* aktualisiert [Zeile 18-22]. Im nächsten Schritt wird der erweiterte Pfad der *frontier* hinzugefügt [Zeile 27]. Handelt es sich um eine uninformierte Suche und ist der Knoten *child* ein präferierter Knoten, wird die private Methode **sortIntoFrontier(path : ArrayList<NodeScreen>)** genutzt, um diesen in die *frontier* einzusortieren [Zeile 24-25]. Hierzu werden die bereits angesprochenen Felder zur Indexspeicherung genutzt. Bei einer informierten Suche ist dies nicht zielführend, da es sich bei jedem Knoten um einen präferierten Knoten handelt.

Nach einer Betätigung des Zurück-Buttons des Gerätes [Zeile 29] wird der zuvor durchsuchte Knoten der Liste mit den bereits besuchten Knoten hinzugefügt [Zeile 37]. Dies geschieht jedoch nur, falls zwei Sonderfälle nicht durchlaufen werden sollten [Zeile 30-35]: Bei dem ersten Sonderfall wurde der Knoten bereits zuvor durchsucht und befindet sich in der Liste [Zeile 30]. Dies wird durch die Methode **checkVisitedList(screen : NodeScreen)** überprüft. Diese vergleicht den Knoten mittels seiner Signatur mit allen bereits besuchten Knoten in der Liste. Der zweite Sonderfall tritt ein, wenn durch das Betätigen des Zurück-Buttons der Startbildschirm der Android-Plattform erreicht wurde [Zeile 33]. Die hierfür verwendete Methode **checkBackPressed()** wird im Kapitel *Limitationen von DataPolicyRipper* weiter behandelt und aus diesem Grund hier nicht näher betrachtet. In beiden Sonderfällen wird der Pfad wieder von der *frontier* entfernt und die Felder zur Indexspeicherung werden zurückgesetzt [Zeile 31-32, Zeile 34-35].

Zum Abschluss der Schleife wird durch die Methode **resetToStartScreen()** zur Startansicht der Anwendung zurücknavigiert und die beiden Methoden **runInteractionPath(path : ArrayList<AbstractEdgeInteraction>)** und **getInteractionPath(path : ArrayList<NodeScreen>)** werden genutzt, um zurück zur aktuell zu erweiternden Ansicht *lastItem* zu gelangen [Zeile 39-40]. Die erste Methode führt eine Abfolge an Interaktionen aus und die zweite ermittelt die Abfolge an Interaktionen, die zu einem Pfad an *NodeScreen*-Objekten gehören.

Ergebnisfall	Stringformat
Zielpfad gefunden	appPkg + ":Start;" + [edgeText; or nodeName;]* + "DataPolicy;"
Keinen Pfad gefunden	appPkg + ":Es wurden keine Datenschutzhinweise gefunden."
Zeitlimit erreicht	appPkg + ":Das Zeitlimit wurde überschritten."

Tabelle 6.2: Format der Ausgabestrings der Methode **search()** der Klasse **BFSSearch**

Die Suche durch die Klasse *BFSSearch* kann drei verschiedene Ergebnisfälle erreichen. Je nach Ergebnisfall wird ein anderer Typ an Ausgabestring von der Methode *search()* zurückgegeben. Die drei Fälle sind in der Tabelle 6.2 aufgeführt. Im ersten Fall findet die Suche einen Pfad zu Datenschutzhinweisen und es wird ein String ausgegeben, welcher diesen Pfad repräsentiert. Zur Zuordnung des gefundenen Pfades zu der Anwendung wird in jedem der drei Fälle der Packagename der Anwendung an den Anfang des Strings gesetzt. Weiterhin beginnt jeder Pfad mit dem String "START" und endet mit dem String "DATAPOLICY". Die einzelnen Knoten in dem Pfad werden durch ein Semikolon getrennt und entweder durch den *edgeText* der ausgeführten Interaktion oder den *nodeName* der erreichten Ansicht repräsentiert.

Wurde kein Pfad gefunden, tritt der zweite Fall ein. Hier wird lediglich der String "ES WURDEN KEINE DATENSCHUTZHINWEISE GEFUNDEN." mit dem Packagennamen der Anwendung ausgegeben. Im dritten und letzten Fall wurde das Zeitlimit bei der Suche erreicht und diese abgebrochen. Hier wird der String "DAS ZEITLIMIT WURDE ÜBERSCHRITTEN." zurückgegeben. Diese Unterscheidung dient der Analyse der Ergebnisse und der Erweiterung von *DataPolicyRipper*.

6.2.4 Komponente: Ausführende Klasse

Dieser Abschnitt widmet sich der letzten Komponente des Programms *DataPolicyRipper*. Die Komponente ist für das Ausführen der Suche nach Datenschutzhinweisen in mehreren Anwendungen hintereinander zuständig. Wie im Klassendiagramm zu der Klasse ***DataPolicyRipper*** aus Abbildung 6.10 zu sehen, handelt es sich bei dieser Komponente um eine Testklasse. Dies liegt an dem bereits im Abschnitt *Verwendete Software* genannten Umstand, dass das Verfahren *DataPolicyRipper* als ein instrumentierter Test einer ansonsten funktionslosen Android-Anwendung realisiert ist.

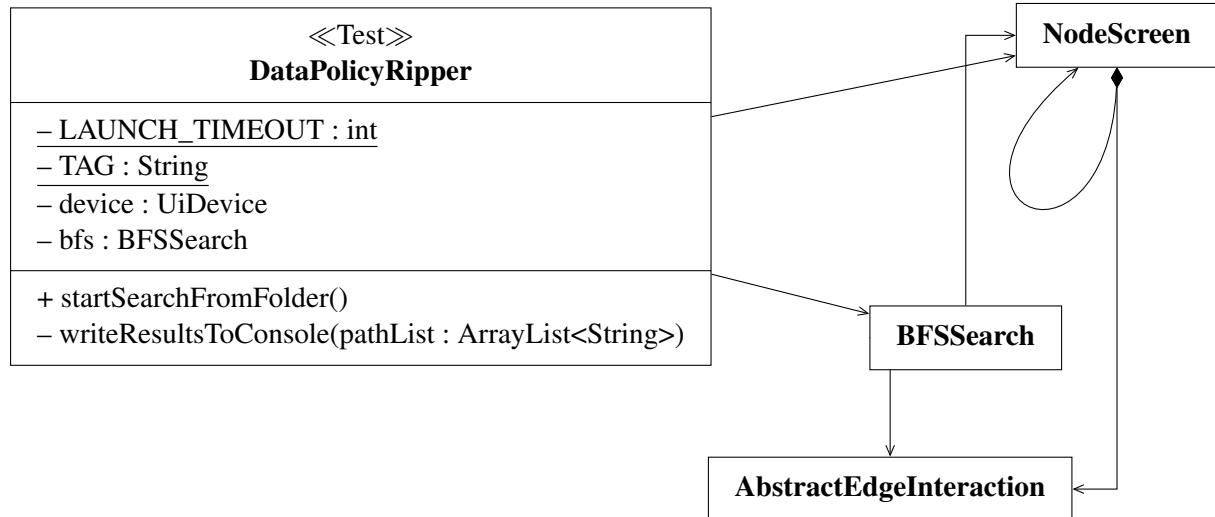


Abbildung 6.10: UML-Klassendiagramm zur Komponente **Ausführende Klasse**

Für die Suche wird eine Instanz vom Typ *UiDevice* benötigt. Das Feld ***device*** wird in diesem Fall genutzt, um durch Betätigung des Home-Buttons zum Hauptbildschirm der Android-Plattform zu gelangen und die Suche von dort aus zu starten. Das Starten vom Hauptbildschirm wird als Best Practice für die Testautomatisierung mit dem Framework UI Automator vorgeschlagen [Doc20h]. Weiterhin wird das Feld vom Typ *UiDevice* für das Auffinden der App-Icons der zu

testenden Anwendungen genutzt. Das Feld ***bfs*** vom Typ *BFS*Search ist für die Suche in den verschiedenen Anwendungen nötig.

Die Testklasse nutzt zwei Methoden für die Suche nach Datenschutzhinweisen. Die Prozedur ***startSearchFromFolder()*** ist eine Testmethode und führt die verschiedenen Suchdurchgänge durch. Hierfür wird zum Hauptbildschirm der Android-Plattform navigiert und danach werden alle Anwendungen, die sich in einem Ordner mit dem Namen „Tested Apps“ befinden, durchsucht. Somit muss vor der Ausführung von *DataPolicyRipper* dieser Ordner mit allen zu durchsuchenden Anwendungen angelegt werden. Die Ergebnisse der Suche werden in einer lokalen ArrayList als Strings gespeichert. Wurden alle Anwendungen durchsucht, wird diese Ergebnisliste an die zweite Methode der Klasse weitergegeben. Die Methode ***writeResultsToConsole(pathList : ArrayList<String>)*** ist demnach dafür zuständig, die Ergebnisse aus der ArrayList in den Log der Android-Plattform auszugeben. Dieser Log kann in AndroidStudio ausgelesen werden. Hierfür kann nach dem Stichwort „Results“ gefiltert werden.

Um die Laufzeit zu verringern, werden in der Methode *startSearchFromFolder()* primär informierte Suchen durchgeführt. Für jede der zu durchsuchenden Anwendungen wird zuerst eine informierte Suche durchgeführt. Nur in dem Fall, in dem keine Datenschutzhinweise gefunden wurden, wird eine uninformede Suche für diese Anwendung durchgeführt. Wurde bereits bei der informierten Suche das Zeitlimit überschritten oder wurde ein Pfad zu Datenschutzhinweisen gefunden, wird keine uninformede Suche durchgeführt.

Um abschließend eine Zusammenfassung und einen Überblick der Funktionsweise von *DataPolicyRipper* zu liefern, ist in Abbildung 6.10 zusätzlich zu dem Klassendiagramm zur Klasse *DataPolicyRipper* das Zusammenspiel der gesamten Komponenten durch das Klassendiagramm dargestellt. Es ist zu erkennen, dass die Testklasse eine Instanz vom Typ *BFS*Search nutzt, um die Suchen auszuführen. Um diese Instanzen zu initialisieren wird zusätzlich ein Objekt vom Typ *NodeScreen* benötigt. Die Suchklasse *BFS*Search greift sowohl auf Methoden und Objekte der Klasse *NodeScreen* als auch auf Methoden und Objekte vom Typ *AbstractEdgeInteraction* zurück. Diese werden für die technische Realisierung des Suchraumes der zu durchsuchenden Anwendung genutzt. Instanzen der Klasse *NodeScreen* sind rekursive Datenstrukturen und bestehen demnach aus mehreren Objekten ihres eigenen Typs. Weiterhin besteht ein *NodeScreen*-Objekt aus Objekten vom Typ *AbstractEdgeInteraction*. Zusammenfassend sind demnach die Komponenten *NodeScreen* und *AbstractEdgeInteraction* dafür zuständig, den Suchgraphen der zu durchsuchenden Anwendungen technisch abzubilden, während die Komponente *BFS*Search für die Suche in diesem Suchgraphen genutzt wird. Die Testklasse *DataPolicyRipper* führt lediglich das Programm *DataPolicyRipper* aus, indem durch sie mehrere Anwendungen hintereinander durchsucht werden.

Mit dieser Komponente ist die technische Beschreibung der Anwendung *DataPolicyRipper* abgeschlossen, da alle vier Teile der Anwendung betrachtet wurden. Einige Besonderheiten werden zusätzlich im Kapitel *Limitationen von DataPolicyRipper* behandelt, da spezifische Lösungen für Hürden bei der Suche nach Datenschutzhinweisen implementiert wurden.

6.3 Durchführung von *DataPolicyRipper*

Dieser Abschnitt soll das Kapitel *Automatisierung des Verfahrens* abschließen, indem ein Beispiel für eine Durchführung des Programms *DataPolicyRipper* vorgestellt wird. Da es sich um eine Testanwendung handelt, muss diese durch Ausführen der Testklasse *DataPolicyRipper*

gestartet werden. In Abbildung 6.11 ist ein Screenshot aus Android Studio abgebildet. Hier sind zwei Auswahlfelder mit Dropdown-Menüs zu sehen. In dem linken Menü wird die auszuführende Klasse oder Methode ausgewählt. In diesem Fall ist das die eben genannte Testklasse. In der rechten Auswahl wird das Gerät, auf dem die Testklasse ausgeführt werden soll, spezifiziert. Dies kann, wie in diesem Fall, ein Emulator sein, oder auch ein Android-Gerät, das mit dem ausführenden Computer verbunden ist. Sind die beiden Dropdown-Menüs korrekt ausgewählt, kann die instrumentierte Testklasse durch Betätigung des „Ausführen“-Buttons gestartet werden. Dieser ist durch ein orangefarbenes Rechteck in der Abbildung markiert.

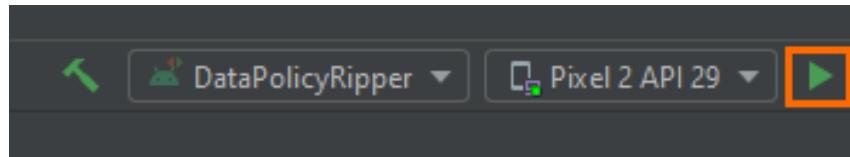


Abbildung 6.11: Start von *DataPolicyRipper*

Bevor *DataPolicyRipper* ausgeführt werden kann, müssen die zu durchsuchenden Anwendungen vorbereitet werden. Dies erfolgt, indem die Anwendungen in einem Ordner auf dem Hauptbildschirm der Android-Plattform hinterlegt werden. Dieser Ordner wird mit dem Namen „Tested Apps“ benannt. Ein Beispiel für einen solchen Ordner ist in Abbildung 6.12 mit einem grün gefärbten Rechteck zu sehen. In diesem Ordner können beliebig viele Anwendungen hinterlegt werden. Im Beispiel aus Abbildung 6.13 befinden sich elf verschiedene Anwendungen in dem Ordner.

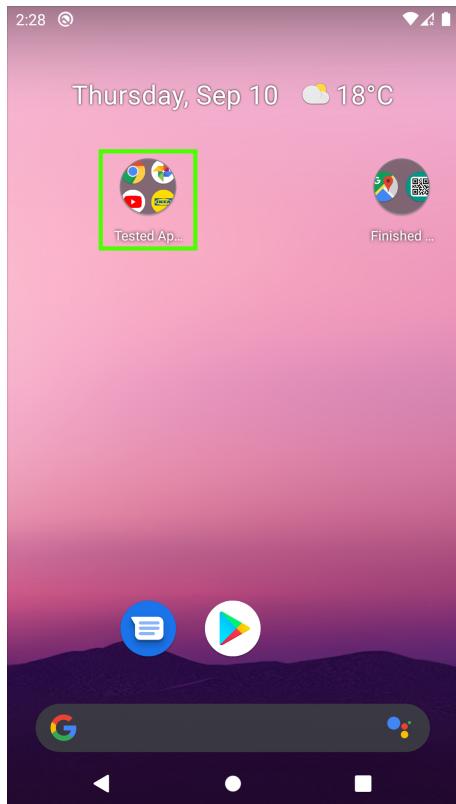


Abbildung 6.12: Hauptbildschirm mit Ordner

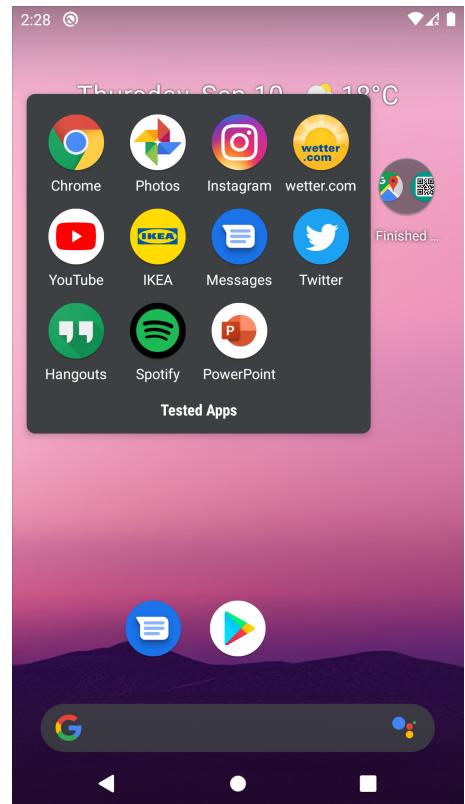
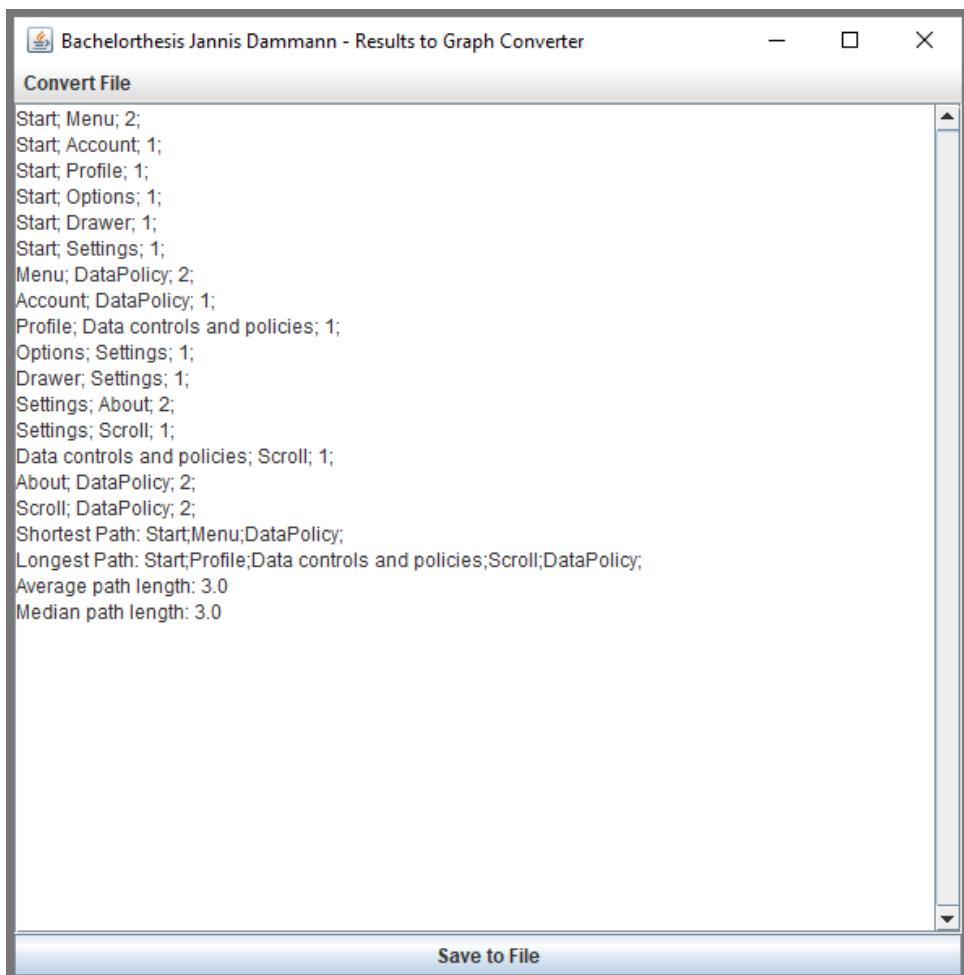


Abbildung 6.13: Ordner „Tested Apps“

Weiterhin sollte jede Anwendung vor der Suche einmal gestartet werden. Wenn nötig, muss sich mit einem Benutzerkonto angemeldet werden. Sind diese Schritte erfolgt, kann *DataPolicyRipper* wie beschrieben ausgeführt werden. Nach dem erfolgreichen Abschluss der Suche nach Datenschutzhinweisen in allen hinterlegten Anwendungen kann der Log der Android-Plattform in Android Studio verwendet werden, um die Ergebnisse anzuzeigen. Ein Beispiel für ein Ergebnis zur Suche in der Anwendung „Youtube“ wäre folgende Ausgabe: 2020-08-15 01:36:06.349 11755-11784/COM.EXAMPLE.BACHELORTHEISANDROIDPROJECT I/RESULTS: COM.GOOGLE.ANDROID.YOUTUBE:START;ACCOUNT;DATAPOLICY;. Die Ergebnisse können in einem Textdokument gespeichert werden. In der Durchführung von *DataPolicyRipper* mit den angegebenen Anwendungen wurde in sieben dieser Anwendungen ein Pfad zu Datenschutzhinweisen gefunden.



The screenshot shows a window titled "Bachelorthesis Jannis Dammann - Results to Graph Converter". The main area contains a text box labeled "Convert File" with the following content:

```

Start; Menu; 2;
Start; Account; 1;
Start; Profile; 1;
Start; Options; 1;
Start; Drawer; 1;
Start; Settings; 1;
Menu; DataPolicy; 2;
Account; DataPolicy; 1;
Profile; Data controls and policies; 1;
Options; Settings; 1;
Drawer; Settings; 1;
Settings; About; 2;
Settings; Scroll; 1;
Data controls and policies; Scroll; 1;
About; DataPolicy; 2;
Scroll; DataPolicy; 2;
Shortest Path: Start;Menu;DataPolicy;
Longest Path: Start;Profile;Data controls and policies;Scroll;DataPolicy;
Average path length: 3.0
Median path length: 3.0

```

At the bottom of the window, there is a "Save to File" button.

Abbildung 6.14: Ausgabe von *ResultConverter* für die Beispieldurchführung

Um relevante zu analysierende Ergebnisse zu erhalten, wurde ein separates Programm *ResultConverter* in Java entwickelt. Dieses kann dazu genutzt werden, beliebig viele, in einem Textdokument gespeicherte Ergebnisse zu konvertieren. Das Programm *ResultConverter* kalkuliert für jede Kombination von Knoten in den verschiedenen Pfaden die Häufigkeit, mit der diese Kombination vorkommt. Für die Beispieldurchführung aus den Abbildungen 6.12 und 6.13 ist das Ergebnis des *ResultConverter* in Abbildung 6.14 zu sehen. In jeder Zeile des Textfeldes ist ein Knotenpaar aufgeführt. Hinter dem jeweiligen Knotenpaar steht die Häufigkeit dieses Paares. Das Ergebnis aus der ersten Zeile START; MENU; 2; bedeutet, dass, um in zwei der sieben

Anwendungen zu den Datenschutzhinweisen zu gelangen, vom Startbildschirm der Anwendung als erster Schritt in das Menü navigiert werden muss.

Diese Ausgaben können für die visualisierte Aufbereitung der Suchergebnisse in einem Graphen verwendet werden. Abbildung 6.15 zeigt einen gewichteten, gerichteten Graphen, der aus den in Abbildung 6.14 zu sehenden Ergebnissen erstellt wurde.

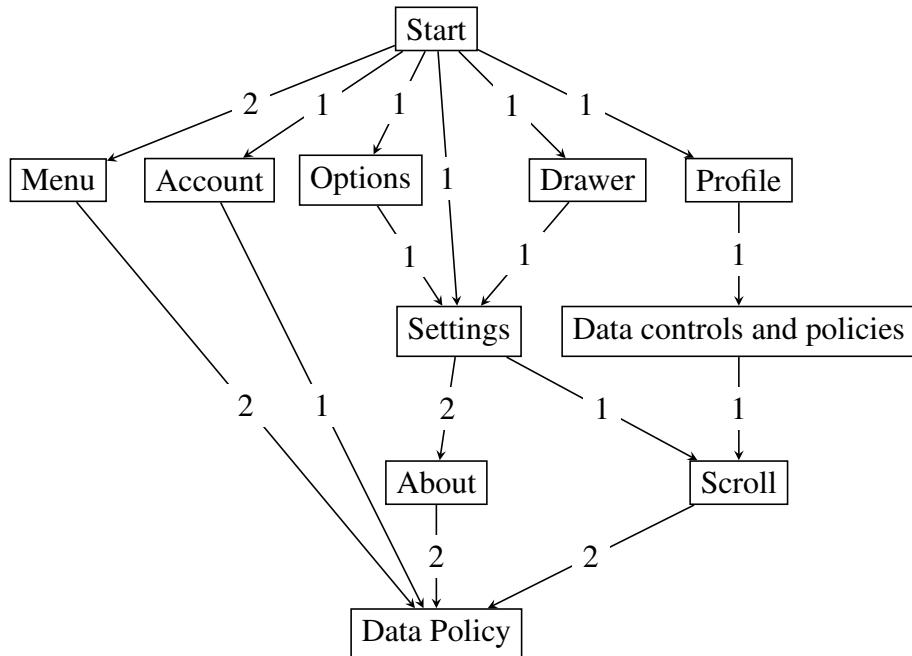


Abbildung 6.15: Ergebnisgraph für die Beispieldurchführung

Jeder Knoten in dem Graphen steht für einen Knoten in dem Pfad zu Datenschutzhinweisen in mindestens einer der Anwendungen. Die Beschriftung eines Knotens ist der Name des Knotens auf dem Pfad. Durch das Programm *ResultConverter* werden die Namen der Knoten teilweise generalisiert. Sollten die Knoten aus der Ausgabe von *DataPolicyRipper* bestimmte Stichworte wie „Menu“, „Account“, „Settings“ oder „Profile“ enthalten, werden diese durch *ResultConverter* auf die entsprechenden Stichworte verkürzt.

Eine Kante in dem Graphen visualisiert eine Interaktion, welche von einem Knoten aus zu einem anderen führt und ist demnach gerichtet. Ebenfalls sind die Kanten gewichtet. Das Gewicht einer Kante gibt an, wie häufig die Interaktion zwischen den beiden entsprechenden Knoten auf den Pfaden zu den Datenschutzhinweisen der durchsuchten Anwendungen vorgekommen ist. Zusätzlich sind, aus Gründen der Übersichtlichkeit, in einigen Graphen in der folgenden Arbeit Kanten mit höherem Gewicht breiter gezeichnet. Die Kante mit Gewicht 1 zwischen den beiden Knoten **Start** und **Account** gibt demnach an, dass in den sieben gefundenen Pfaden zu Datenschutzhinweisen auf einem dieser Pfade vom Startbildschirm der Anwendung zur Account-Übersicht navigiert werden muss.

Durch die erste Beispieldurchführung von *DataPolicyRipper* wurde demnach bei sieben der elf durchsuchten Anwendungen ein Pfad zu Datenschutzhinweisen identifiziert. Der kürzeste Pfad der sieben gefundenen Pfade benötigt zwei Interaktionen, um das Ziel der Suche zu erreichen. In der Anwendung mit dem längsten Pfad werden vier Interaktionen benötigt, um zu den Datenschutzhinweisen zu gelangen. Die Durchschnittslänge der Pfade in der Beispieldurchführung beträgt drei Interaktionen. Der Median liegt ebenfalls bei drei Interaktionen.

Im Kapitel *Ergebnisse der Analyse* wird die beschriebene Visualisierung der Ergebnisse in einem gerichteten, gewichteten Graphen genutzt. Nachdem in diesem Kapitel die technische Funktionsweise des Programms *DataPolicyRipper* erklärt wurde, wird weiterhin im folgenden Kapitel auf einige *Limitationen von DataPolicyRipper* eingegangen.

7 Limitationen von *DataPolicyRipper*

In diesem Kapitel wird auf die Limitationen des für diese Arbeit entwickelten Programms *DataPolicyRipper* eingegangen. Zuvor werden im ersten Abschnitt des Kapitels einige *Hürden bei der Entwicklung* des Programms dargelegt und welche spezifischen Lösungen für diese Probleme angewandt wurden.

7.1 Hürden bei der Entwicklung

Um einen Überblick über die bei der Entwicklung von *DataPolicyRipper* gemeisterten Komplikationen zu geben, wird im Folgenden eine Auswahl dieser Komplikationen vorgestellt. Die erste Hürde bei der Entwicklung stellt die Implementation einer Breitensuche in dem Verfahren dar. In Abbildung 7.1 sind zwei gerichtete Graphen zu sehen. Beide Graphen sind analog zueinander aufgebaut. Die Beschriftung der Knoten repräsentiert jeweils eine Durchsuchungsreihenfolge der Knoten durch eines von zwei Suchverfahren. Der linke Graph visualisiert die Reihenfolge, in der die Knoten bei einer Breitensuche durchsucht werden, während der rechte Graph dies für eine Tiefensuche veranschaulicht. In beiden Fällen ist der Knoten **0** der Startknoten der Suche.



Abbildung 7.1: Reihenfolge der durchsuchten Knoten in verschiedenen Suchverfahren

In einer Breitensuche werden, wie im linken Graphen zu erkennen, zunächst alle Knoten einer Tiefe durchsucht. Erst darauffolgend werden die Pfade auf die nächsthöhere Tiefe erweitert und die Knoten auf dieser höheren Tiefe durchsucht. In einer Tiefensuche wird zunächst ein Pfad komplett durchsucht. Dies führt dazu, dass in dem Graphen der rechte Teilgraph unter dem Startknoten **0** erst durchsucht wird, nachdem der linke Teilgraph komplett durchsucht wurde.

Eine Tiefensuche durch eine Anwendung des Betriebssystems Android ist mit weniger Aufwand zu implementieren als eine Breitensuche. Dies liegt an dem eben beschriebenen Umstand. In einer Breitensuche muss beispielsweise nach Durchsuchen des Knotens **2** zurück zum Startknoten navigiert werden. Darauffolgend kann dann vom Startknoten aus der bereits durchsuchte Knoten **1** durchlaufen werden, um zum nächsten Knoten **3** zu gelangen. Dieser ständige Wechsel der Teilgraphen in der Breitensuche führt zu einem erhöhten Aufwand an Navigation durch die Anwendung. Es muss in komplexeren Anwendungen wiederholt zum Startbildschirm der

Anwendung zurücknavigiert werden, da häufig in andere Teilgraphen gewechselt wird. Ebenfalls müssen die zu erweiternden Pfade regelmäßig bis zur aktuell zu durchsuchenden Tiefe durchlaufen werden. In einer Tiefensuche muss dies nur bei einem Wechsel der Teilgraphen geschehen. Die Breitensuche wurde als Implementation gewählt, da diese im Gegensatz zur Tiefensuche in jedem Fall den kürzesten Pfad zu einem Zielknoten findet [PM10][S.86].

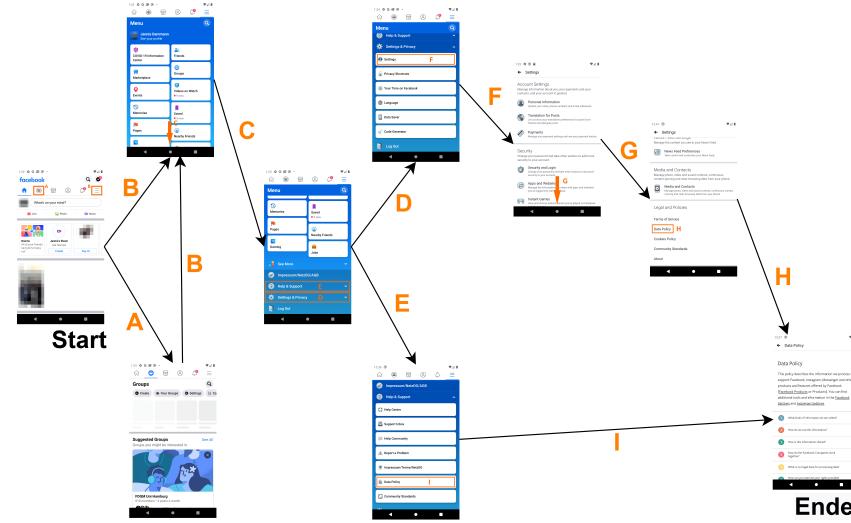


Abbildung 7.2: Suchgraph zum Anwendungsbeispiel Facebook

Um diese Unterschiede anhand eines Anwendungsbeispiels zu beschreiben, kann der im Kapitel *Ausführung per Hand* erstellte Graph zur Anwendung „Facebook“ verwendet werden, welcher aus Gründen der Übersichtlichkeit nochmals in Abbildung 7.2 zu sehen ist. Eine Tiefensuche würde zuerst den Pfad $P_1 = \{A, B, C, D, F, G, H\}$ komplett erweitern und durchsuchen. Demnach würde die Tiefensuche im schlechtesten Fall den längsten Pfad zu den Datenschutzhinweisen finden. Wird einmal angenommen, dass es sich bei der Ansicht **Ende** nicht um einen Zielknoten handelt, so würde nach der Durchsuchung von P_1 zurück navigiert und der Pfad $P_2 = \{A, B, C, E, I\}$ vollständig durchsucht werden. Erst wenn der vom Startknoten mit der Interaktion **A** ausgehende Teilgraph vollständig durchsucht worden wäre, würde der mit Interaktion **B** startende Teilgraph durchsucht und die Pfade $P_3 = \{B, C, D, F, G, H\}$ und $P_4 = \{B, C, E, I\}$ nacheinander vollständig durchsucht werden.

Schritt	Aktuell erweiterter Pfad
1	{A}
2	{B}
3	{A, B}
4	{B, C}
5	{A, B, C}
6	{B, C, D}
7	{B, C, E}
8	{A, B, C, D}
9	{A, B, C, E}
10	{B, C, D, F}
11	{B, C, E, I}

Tabelle 7.1: Ablauf einer Breitensuche anhand des Beispielgraphen

Im Falle einer Breitensuche werden die Pfade nicht hintereinander, sondern parallel zueinander durchsucht. Die Pfade werden lediglich bis zur aktuellen Tiefe erweitert. Die Tabelle 7.1 zeigt die Schritte, in denen eine Breitensuche in dem Beispiel der Anwendung „Facebook“ die Pfade erweitert. Wie zu erkennen ist, werden die Pfade nacheinander bis zur aktuell durchsuchten Tiefe erweitert. Dies führt dazu, dass in allen Schritten bis auf **7**, **9** und **11** zurück zum Startbildschirm navigiert werden muss, um in den jeweils anderen mit Interaktionen **A** oder **B** beginnenden Teilgraphen zu wechseln.

In *DataPolicyRipper* wurden einige Methoden für diesen ständigen Wechsel der Teilgraphen entwickelt. Dies sind die Methoden *runInteractionPath()*, *getInteractionPath()* und *resetToStartScreen()* der Klasse *BFSSearch*. Die beiden ersten Methoden werden dazu genutzt, vom Startbildschirm zum aktuell zu durchsuchenden Knoten zu gelangen. Die dritte Methode *resetToStartScreen()* realisiert das Zurückkehren zum Startbildschirm von einer beliebigen Ansicht der Anwendung.

```

1  private void resetToStartScreen() {
2      int i = 0;
3      do {
4          device.pressBack();
5          i++;
6          if (i >= 10) {
7              device.pressHome();
8              device.pressRecentApps();
9              UiObject clearButton = device.findObject(new
10                 UiSelector().text("Clear all").clickable(true));
11              UiScrollable listView = new UiScrollable(new
12                 UiSelector().className("android.widget.ListView") .
13                 scrollable(true));
14              listView.setAsHorizontalList();
15              if (listView != null) {
16                  do {
17                      listView.scrollToBeginning(4);
18                  } while(clearButton != null &&
19                      clearButton.exists());
20                      clearButton.clickAndwaitForNewWindow();
21                  } else {
22                      device.pressHome();
23                  }
24              }
25          } while (!checkBackPress());
26      }

```

Abbildung 7.3: Code-Ausschnitt der Methode *resetToStartScreen()*

Abbildung 7.3 zeigt den Code zu der Methode *resetToStartScreen()*. Für die beschriebene Komplikation der Implementierung der Breitensuche ist die if-Anweisung in dem Code-Ausschnitt nicht relevant [Zeile 6-24]. Für das Zurücknavigieren zum Startbildschirm wird eine do-while-Schleife genutzt. In der do-while-Schleife wird jeweils einmal der Zurück-Button des Geräts betätigt [Zeile 4]. Dies wird ausgeführt, bis die Methode *checkBackPress()* TRUE ausgibt [Zeile 25]. Diese Prozedur überprüft, ob der Hauptbildschirm der Android-Plattform erreicht wurde und wenn dies der Fall ist, öffnet die Methode die zu durchsuchende Anwendung erneut und die Prozedur ist beendet.

Somit wird durch die Methode der Zurück-Button wiederholt betätigt bis die Anwendung geschlossen wurde. Im Nachhinein wird die Anwendung erneut geöffnet und der Startbildschirm der Anwendung ist erreicht. Im vorangegangenen Kapitel wurde in Abbildung 6.9 ein Teil der Methode `search()` der Klasse `BFSSearch` behandelt. In diesem Code-Ausschnitt ist zu sehen, dass nach jeder Erweiterung eines Pfades die drei zuvor genannten Methoden für den Wechsel der Teilgraphen ausgeführt werden.

Eine weitere Komplikation, die bei der Entwicklung auftrat, stellen Dialoge dar, die nicht durch den Zurück-Button geschlossen werden können. Dies sind klassischerweise Dialoge vom Typ ALERTDIALOG der Android-Plattform [Doc20a]. Objekte von diesem Typ besitzen das Attribut **Cancelable**, welches angibt, ob das Schließen des Dialogs durch Betätigung der Zurück-Taste möglich ist [Doc20c]. Wie ein solcher Dialog aussehen kann, zeigt Abbildung 7.4. Der Dialog ist in der Abbildung mit einem orangefarbenen Rechteck gekennzeichnet.

Das Auftreten solcher Dialoge wird in `DataPolicyRipper` ebenfalls in der bereits beschriebenen Methode `resetToStartScreen()` gehandhabt. Die hierfür benötigte if-Anweisung ist in Abbildung 7.2 gezeigt [Zeile 6-24]. Tritt ein ALERTDIALOG auf, verhindert dieser, dass durch Betätigung des Zurück-Buttons zum Hauptbildschirm der Android-Plattform navigiert werden kann. Somit hält die Methode `resetToStartScreen()` einen Integer-Wert, welcher die Anzahl der Schleifeniterationen inkrementiert. Wurde bereits zehnmal der Zurück-Button betätigt, muss ein ALERTDIALOG vorliegen, da die Suche insgesamt nur bis zu einer Tiefe von neun Interaktionen sucht und sonst der Hauptbildschirm bereits erreicht worden wäre. Somit wird die if-Anweisung betreten [Zeile 6]. Der Code in der if-Anweisung navigiert durch Betätigung des Home-Buttons zum Hauptbildschirm [Zeile 7]. Dies ist auch möglich, wenn ein ALERTDIALOG vorliegt, da die Anwendung in diesem Fall geschlossen wird. Würde die Anwendung jedoch nun sofort wieder geöffnet werden, ist der ALERTDIALOG weiterhin vorhanden. Demnach muss die Anwendung komplett angehalten werden. Hierfür wird der dritte der Navigationsbuttons der Android-Plattform betätigt [Zeile 8]. Dies ist der Button „Recent Apps“, welcher alle zuletzt geöffneten Anwendungen anzeigt. Diese Ansicht ist in Abbildung 7.5 zu sehen.

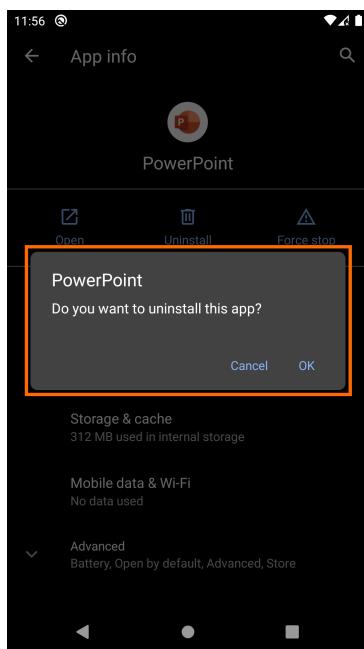


Abbildung 7.4: AlertDialog

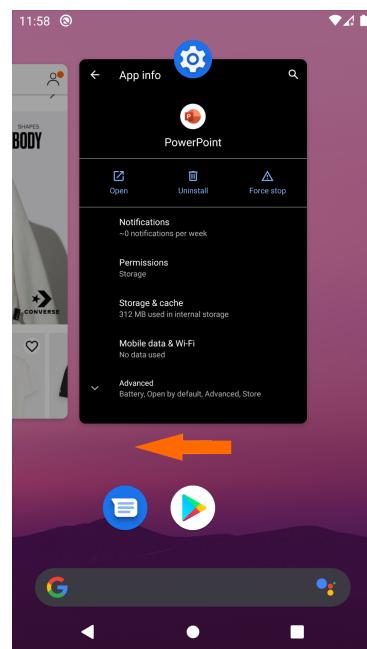


Abbildung 7.5: Lösung(1)

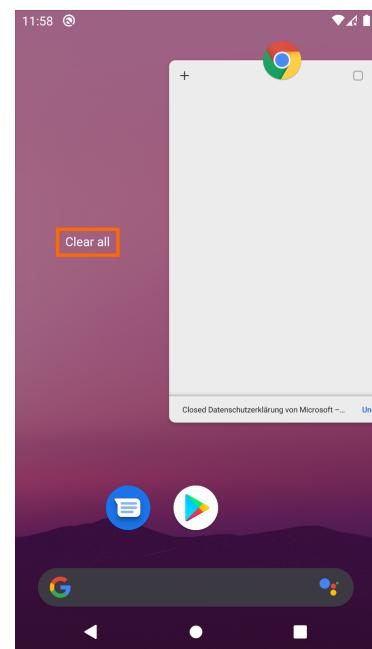


Abbildung 7.6: Lösung(2)

Die grafische Darstellung der Durchführung der Methode in den Abbildungen 7.4 und 7.5 zeigt, dass in der Ansicht „Recent Apps“ nach links gescrollt wird, bis ein Button mit der Beschriftung "CLEAR ALL" [Zeile 9-19] zu sehen ist. Dieser ist in Abbildung 7.6 orangefarben hervorgehoben. Durch Betätigung dieses Buttons [Zeile 20] werden alle derzeit geöffneten Anwendungen geschlossen. Im Nachhinein wird zum Hauptbildschirm navigiert [Zeile 22]. In diesem Fall ist die Abbruchbedingung der do-while-Schleife erfüllt und die Methode *checkBackPress()* öffnet die zu durchsuchende Anwendung erneut. Nun ist der ALERTDIALOG geschlossen, da die Anwendung neu gestartet wurde.

Die zwei letzten Komplikationen, auf die in diesem Abschnitt eingegangen wird, werden durch Systemeinstellungen des Betriebssystems Android behoben. Die erste dieser Hürden bei der Entwicklung stellt die Tastatur des Gerätes dar. Bei Betätigen einer Interaktion, welche zu einer Suchleiste führt, wird die Tastatur von Android automatisch geöffnet. Diese verdeckt einen großen Teil des Bildschirms und sorgt somit dafür, dass überflüssig häufig durch Ansichten gescrollt werden muss. Weiterhin stellen die Buttons der Tastatur eine relevante Anzahl an Interaktionen dar, welche bei einer uninformatierten Suche beachtet werden und somit die Suche deutlich verlängern. Dieses Problem wird vermieden, indem vor der Durchführung von *Data-PolicyRipper* die Tastatur im Betriebssystem deaktiviert wird. Diese wird demnach nicht mehr automatisch geöffnet und muss nicht beachtet werden.

Die zweite der genannten Komplikationen ist der „Picture-in-picture-Modus“ [Doc19e], den einige Anwendungen auf der Android-Plattform nutzen. Hierbei handelt es sich um eine Funktion, die es Nutzern*innen ermöglicht, den Inhalt einer Anwendung in einem kleinen Fenster über den Inhalten anderer Anwendungen zu nutzen. Diese Funktion dient dem Multitasking und wird häufig für das Anzeigen von Videos genutzt [Doc19e]. In den beiden linken Screenshots aus Abbildung 7.7 ist dies am Beispiel der Anwendung „Youtube“ zu sehen. In der linken Ansicht wird ein Video abgespielt. Wird nun der Zurück-Button betätigt, schließt sich die Anwendung nicht. Stattdessen wird das Video in einem kleinen Fenster auf dem Hauptbildschirm der Android-Plattform abgespielt, wie in der mittleren Ansicht orangefarben markiert.

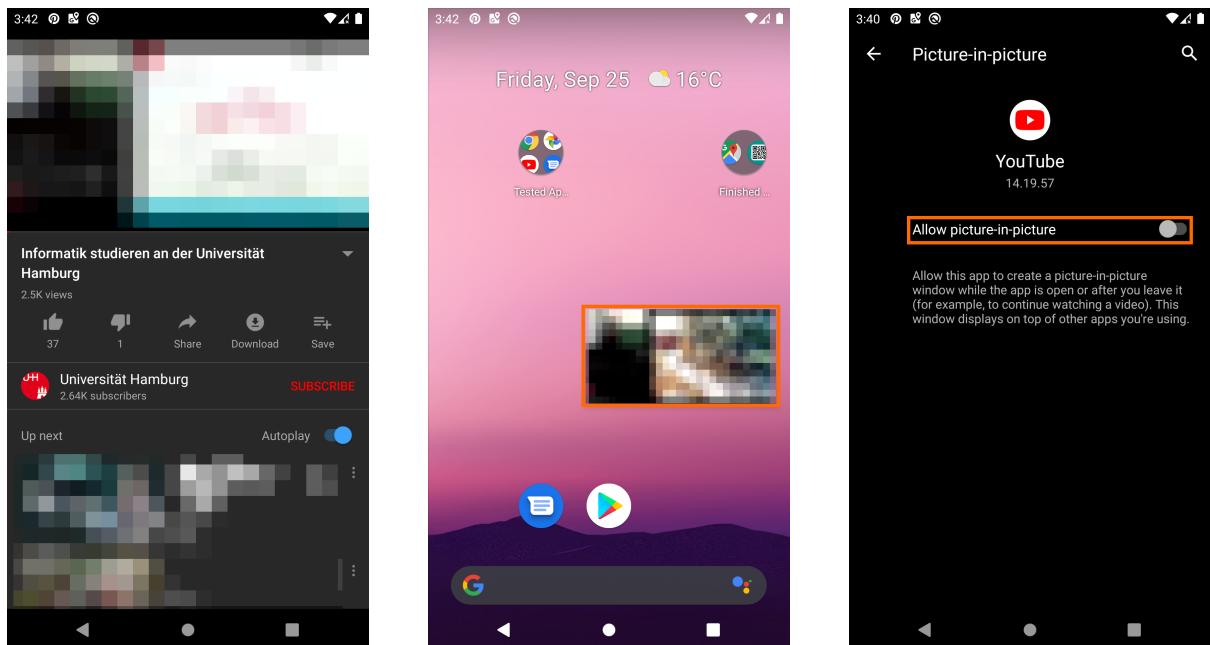


Abbildung 7.7: Picture-in-picture-Modus am Beispiel der Anwendung „Youtube“

Sollte eine Anwendung, die durch *DataPolicyRipper* nach Datenschutzhinweisen durchsucht wird, in diesen Multitasking-Modus gelangen, kann die Anwendung nicht vom Hauptbildschirm aus erneut geöffnet werden, da diese technisch gesehen bereits ausgeführt wird. Aus diesem Grund muss diese Funktion in den Systemeinstellungen der Android-Plattform für jede der zu durchsuchenden Anwendungen deaktiviert werden. Hierzu wird, wie in der rechten Ansicht aus Abbildung 7.7 gezeigt, die Auswahl „Allow picture-in-picture“ deaktiviert.

7.2 Grenzen der Methodik

Zusätzlich zu den im vorangegangenen Abschnitt beschriebenen Komplikationen, welche in der Entwicklung gelöst werden konnten, hat das Programm *DataPolicyRipper* einige Limitationen. Dies sind Umstände, unter denen die Suche nach Datenschutzhinweisen keine Ergebnisse erzielt.

Die erste und relevanteste dieser Limitationen ist durch das genutzte Framework „UI Automator bedingt“. Da die bevorzugte Suchmethode durch die im Abschnitt *Komponente: Knoten* behandelten Schlagwörter heuristisch ist, liefert *DataPolicyRipper* in Anwendungen, welche keine *ContentDescription* für Piktogramme benutzen, keine Ergebnisse. Die Wahrscheinlichkeit, dass durch den zweiten Suchdurchlauf mit einer uninformativen Methodik Datenschutzhinweise gefunden werden, ist gering. Dies liegt daran, dass Anwendungen im Normalfall einen Großteil an möglichen Interaktionen bieten und somit die Laufzeit deutlich zu lang ist. Ein Beispiel für eine Anwendung, welche keine *ContentDescription* für Piktogramme auf dem Pfad zu Datenschutzhinweisen liefert, ist die Anwendung des sozialen Netzwerks „Facebook“.

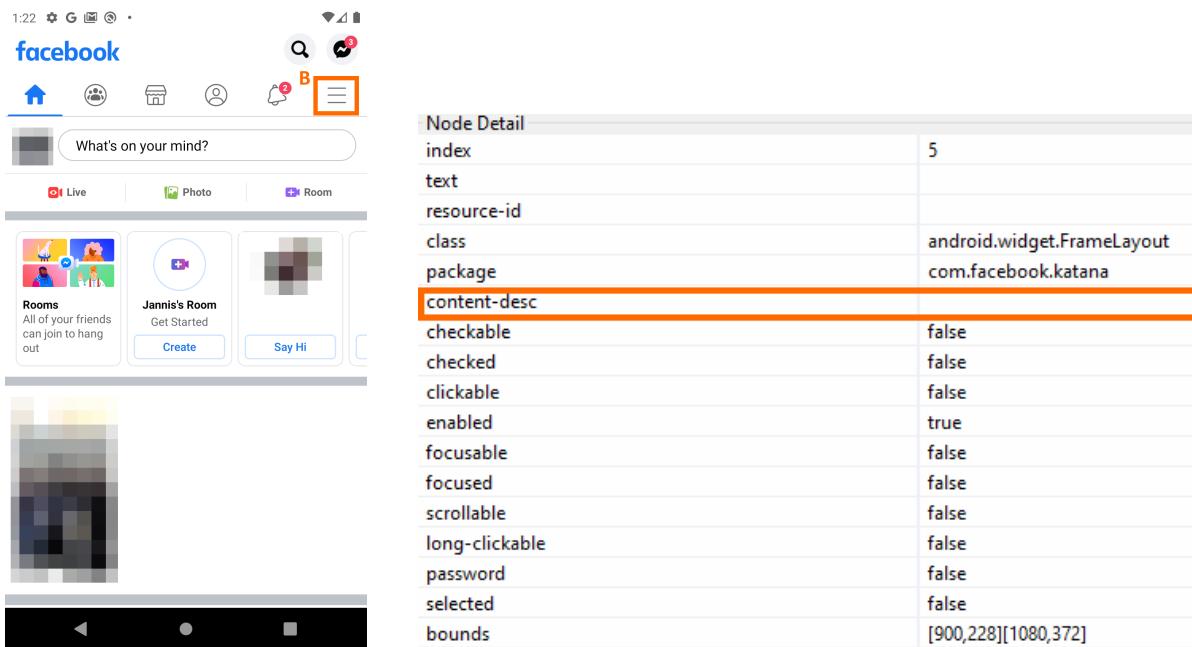


Abbildung 7.8: Eigenschaften der Interaktion **B** in der Anwendung „Facebook“

Abbildung 7.8 zeigt auf der rechten Seite die durch das Tool „UI Automator Viewer“ [Doc19f] abstrahierten Eigenschaften der Interaktion **B**. Diese ist in der linken Ansicht der Anwendung markiert und bildet die erste Interaktion auf dem Pfad zu den Datenschutzhinweisen (siehe Abschnitt *Ausführungsbeispiel anhand einer Anwendung*). Die *ContentDescription* der Interaktion

ist ebenfalls orangefarben hervorgehoben. Hier ist zu erkennen, dass diese leer ist. Dies führt dazu, dass *DataPolicyRipper* in der Anwendung des sozialen Netzwerks keine Ergebnisse liefert. In der uninformierten Suche wird hier das Zeitlimit überschritten.

Weitere typische Anwendungen, welche keine *ContentDescription* für Interaktionen bereitstellen, sind Spiele. Diese bieten häufig ihren eigenen Stil für die GUI an und auf diese kann nicht zugegriffen werden. Acht der zwanzig meistinstallierten Anwendungen laut der Statistik der Webseite „AndroidRank“ sind Spiele [And19b]. In allen dieser Spiele konnten aus diesem Grund keine Datenschutzhinweise gefunden werden.

Das Fehlen einer *ContentDescription* bei Piktogrammen in Anwendungen ist negativ anzumerken. Die *ContentDescription* ist vor allem für die Accessibility-Nutzung durch Personen mit Seh-, Hör- oder anderen Behinderungen relevant [Doc19d]. Das Attribut wird beispielsweise dafür genutzt, um, mithilfe besonderer Geräte, Nutzer*innen diese Beschreibungen vorzulesen [Doc19d]. Demnach ist eine fehlende *ContentDescription* nicht nur für die Nutzung von *DataPolicyRipper* limitierend, sondern ebenfalls eine Problematik für die allgemeine Nutzung der Anwendung.

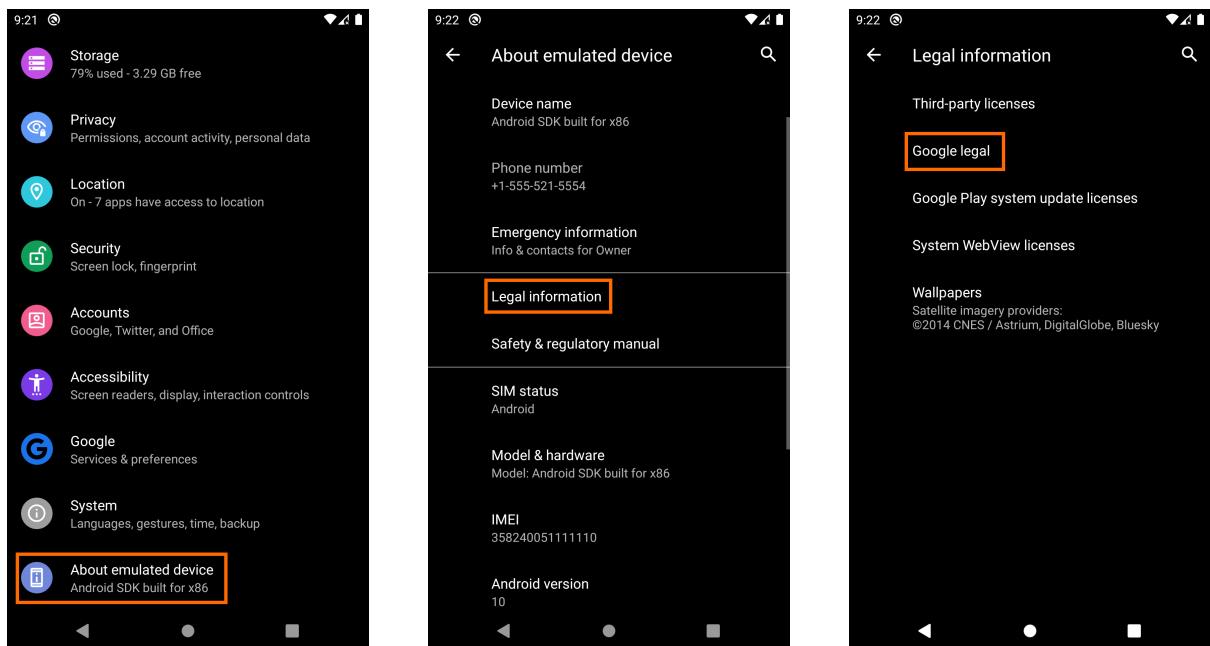


Abbildung 7.9: Datenschutzhinweise in den Systemeinstellungen

Die Sprache einer Anwendung stellt ebenfalls eine Limitation für das Verfahren dar. Für die Schlagwörter der heuristischen Suche wurden englische Wörter gewählt. Dies sorgt demnach dafür, dass Anwendungen in einer anderen Sprache nur uninformativ durchsucht werden können und in seltenen Fällen Datenschutzhinweise gefunden werden. Da Englisch die meistgesprochene Sprache der Welt ist [Eth20] und insbesondere im digitalen Kontext vorherrschend ist, wurde sich in der Entwicklung für diese Schlagwörter entschieden.

Die letzte hier behandelte Art von Anwendungen, in denen durch *DataPolicyRipper* keine Ergebnisse gefunden werden, sind Anwendungen, welche keinen Pfad zu Datenschutzhinweisen besitzen. Dies sind beispielsweise Systemanwendungen, die auf Geräten vorinstalliert sind. Die Datenschutzhinweise, die sich auf diese Anwendungen beziehen, könnten dann beispielsweise in den Systemeinstellungen des Geräts zu finden sein. Die Ansichten aus Abbildung 7.9 veranschaulichen durch orangefarben hervorgehobene Interaktionen, wie diese Datenschutzhinweise

zum Beispiel in dem für *DataPolicyRipper* genutzten Emulator in den Systemeinstellungen erreicht werden können. Nach Betätigung der Interaktion in der letzten Ansicht kann die Datenschutzrichtlinie aus mehreren zu Googles Diensten zugehörigen Richtlinien ausgewählt werden.

Theoretisch kann *DataPolicyRipper* auf allen neueren Geräten mit dem Betriebssystem Android ausgeführt werden. Es wurde jedoch mit dem Emulator von Android Studio und somit für Android-Geräte von Google entwickelt und für diese optimiert. Die unterschiedlichen Anpassungen der verschiedenen Hersteller an das Betriebssystem könnten möglicherweise dafür sorgen, dass einige Aspekte von *DataPolicyRipper* nicht angemessen funktionieren. Aufgrund von Beschränkungen in der Auswahl der Mobiltelefone konnte das Verfahren lediglich zusätzlich mit dem Modell Galaxy S7 von Samsung [Deu20] getestet werden. Mit diesem funktionierte es nach einigen problemspezifischen Anpassungen des Hauptbildschirms durch die Installation und Verwendung eines anpassbaren Hauptbildschirms beziehungsweise „Launchers“ [Sto20e]. Somit sollte *DataPolicyRipper* im besten Fall mit dem durch Android Studio bereitgestellten Emulator genutzt werden.

8 Ergebnisse der Analyse

Dieses Kapitel wird dazu genutzt, die Analyseergebnisse der Durchführung des Verfahrens mit einer größeren Auswahl an Anwendungen genauer zu betrachten. Diese Ergebnisse wurden durch die Durchführung von *DataPolicyRipper* mit den Anwendungen geliefert. Da jedoch aufgrund der im vorherigen Kapitel genannten *Limitationen von DataPolicyRipper* in einigen dieser Anwendungen keine Pfade zu Datenschutzhinweisen gefunden werden, wurden diese Anwendungen nochmals per Hand analysiert.

Insgesamt wurden die 100 am häufigsten installierten Anwendungen der Android-Plattform betrachtet. Die Auswahl wurde anhand der Statistik der meistinstallierten Anwendungen nach der Webseite „AndroidRank“ vom Stand des 28. September 2020 getätigt [And19b]. Diese wurden mittels *DataPolicyRipper* durchsucht. Wurde mit diesem Verfahren kein Pfad zu Datenschutzhinweisen in einer Anwendung gefunden, ist diese per Hand durchsucht und der eventuell gefundene Pfad notiert worden.

Bei 17 dieser Anwendungen handelt es sich um Systemanwendungen oder Systemdienste, welche entweder zum Betriebssystem Android oder zu den Systemanwendungen vom Hersteller Samsung gehören. In diesen 17 Anwendungen ist kein Pfad zu Datenschutzhinweisen zu finden, da diese in den Systemeinstellungen des Geräts hinterlegt sind. Demnach wurde der Pfad zu den Datenschutzrichtlinien in den Systemeinstellungen der jeweiligen Geräte als Ergebnis für diese Anwendungen verwendet. Sieben der 100 meistinstallierten Anwendungen sind nur unter Angabe einer Mobiltelefonnummer nutzbar. Demnach wurden diese Anwendungen lediglich analog zu der im Kapitel *Ausführung per Hand* vorgestellten Vorgehensweise analysiert. Zusätzlich konnte eine weitere Anwendung ebenfalls lediglich per Hand analysiert werden. Der Grund hierfür ist, dass diese Anwendungen lediglich mit einem kostenpflichtigen Account genutzt werden konnte. Für die Nutzung zweier weiterer Anwendungen stand im ersten Fall kein kompatibles Gerät zur Verfügung und im zweiten Fall war ein betriebliches Benutzerkonto vonnöten. Diese beiden Anwendungen konnten demnach nicht analysiert werden. Weiterhin konnten in drei anderen Anwendungen weder durch *DataPolicyRipper* noch durch die händische Analyse Datenschutzhinweise gefunden werden.

Die Abdeckung von Anwendungen, in denen durch *DataPolicyRipper* Datenschutzhinweise gefunden wurde, beträgt circa 40 Prozent. Bei dieser Berechnung wurden die bisher genannten Systemanwendungen und -dienste sowie die dreizehn weiteren Sonderfälle außer Acht gelassen. Dies ergibt insgesamt 70 Anwendungen, in denen durch das Verfahren Datenschutzhinweise gefunden werden könnten. Die genaue Aufschlüsselung der verschiedenen Ausgaben von *DataPolicyRipper* sind in Tabelle 8.1 angegeben.

Die Ausgaben des Verfahrens bilden drei unterschiedliche Fälle: Es wurden Datenschutzhinweise gefunden, das Zeitlimit wurde überschritten oder es konnten keine Datenschutzhinweise gefunden werden. Weiterhin sind in der Tabelle Spiele gesondert aufgeführt. Von diesen gab es in der Liste der meistinstallierten Anwendungen zehn Stück. Diese bilden, wie im Kapitel *Limitationen von DataPolicyRipper* angemerkt, durch die fehlende *ContentDescription* eine Komplikation, in der *DataPolicyRipper* keine Datenschutzhinweise findet. Das Überschreiten

Fall	Häufigkeit	Prozentualer Anteil
Datenschutzhinweise gefunden	27	38.6 %
Zeitlimit überschritten	13	18.6 %
Keine Datenschutzhinweise gefunden	20	28.5 %
Spiel (Keine Hinweise gefunden)	10	14.3 %

Tabelle 8.1: Evaluation von *DataPolicyRipper*

des Zeitlimits weist im Normalfall auf eine Anwendung hin, die komplexer Natur ist und generell viele Interaktionen besitzt. In diesen dauert die Suche nach Datenschutzhinweisen demnach zu lange und wird abgebrochen. Dies war in 18.6 Prozent der analysierten Anwendungen der Fall. In den restlichen 28.5 Prozent der Anwendungen wurden keine Datenschutzhinweise gefunden. Der Grund hierfür können, analog zu den Spieleanwendungen, fehlende *ContentDescription*-Attribute für Piktogramme sein. Weitere Erklärungsmöglichkeiten sind ungünstig auftretende Dialoge, die die Betätigung von relevanten Interaktionen verhindern, oder Interaktionen, die nicht bekannt sind. Fälle, in denen durch *DataPolicyRipper* Ergebnisse gefunden wurden, jedoch per Hand keine Datenschutzhinweise auffindbar waren, gab es nicht. Demnach sollte die Korrektheit von gefundenen Pfaden durch *DataPolicyRipper* gegeben sein.

Im Folgenden werden die Ergebnisse aus allen durchsuchten Anwendungen gebündelt betrachtet. Weiterhin werden *Anwendungen vom selben Hersteller* miteinander verglichen und es werden einige *Anwendungen gleicher Art* analysiert, um zu erfahren, ob diese Gemeinsamkeiten in der Einbindung der Datenschutzhinweise aufweisen.

8.1 Gesamtbetrachtung aller Anwendungen

Wie zuvor angegeben, konnten in 95 der 100 meistinstallierten Anwendungen entweder durch *DataPolicyRipper* oder durch händische Analyse Ergebnisse gefunden werden. Die Interaktionspfade zu den Datenschutzhinweisen haben Längen zwischen zwei und sechs Interaktionen. Die durchschnittliche Pfadlänge aller 95 Pfade beträgt **3.66** Interaktionen. Weiterhin liegt der Median dieser Ergebnisspfade bei **4** Interaktionen.

In Abbildung 8.1 ist der aus den Ergebnissen der Analyse erstellte Graph zu sehen. Aus Gründen der Übersichtlichkeit sind Kanten mit Gewichtung **1** oder **2** ohne Gewicht gezeichnet. Ist eine Kante beidseitig, beziehungsweise zeigt sie mit einer Pfeilspitze in beide Richtungen, ist diese mit zwei Gewichten datiert. Die Gewichte sind mit einem Trennstrich visuell voneinander getrennt und blau markiert. Das an vorderer Stelle stehende Gewicht ist die Häufigkeit, mit der vom nördlich gelegenen Knoten zum südlich gelegenen Knoten navigiert wird. Das zweite Gewicht gibt die Häufigkeit der Gegenrichtung an. Somit bedeutet beispielsweise das Gewicht **24/5** zwischen den Knoten **Settings** und **Scroll**, dass es 24 Ergebnispfade gab, in denen von **Settings** eine Interaktion des Scrollens ausgeführt werden muss. Demnach lagen zusätzlich noch fünf Fälle vor, in denen nach einem Scrollen auf einen Einstellungsbutton geklickt werden muss.

Einige der Interaktionsnamen benötigen nähere Erklärung. Der Knoten **Daily** repräsentiert zwei Spieleanwendungen vom selben Entwickler, in denen erst eine Ansicht mit dem Namen „Daily Challenge“ geöffnet werden muss, um die Datenschutzhinweise zu erreichen. Ein weiterer nicht

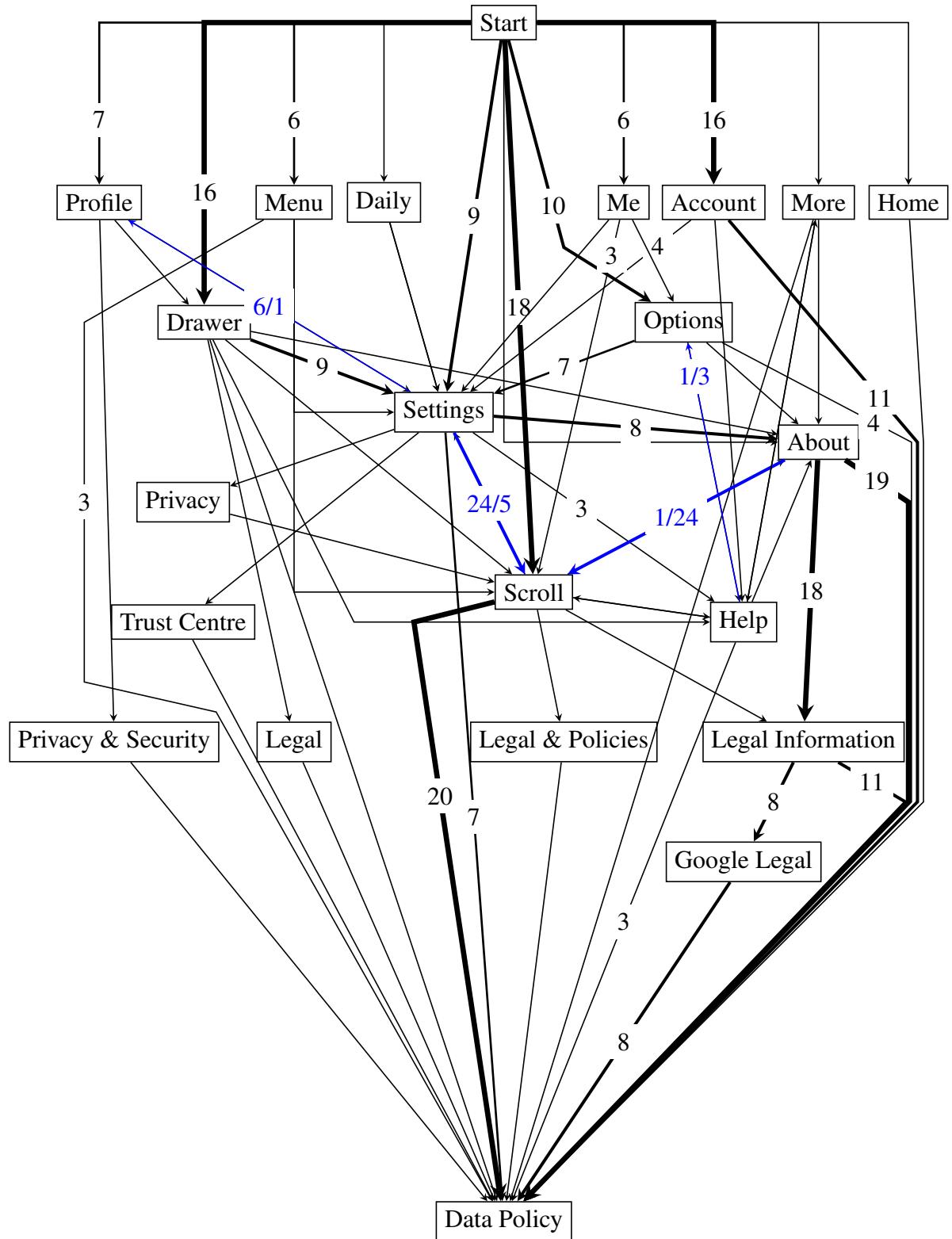


Abbildung 8.1: Ergebnisgraph für die Gesamtbetrachtung

selbsterklärender Name einer Interaktion ist der Knoten **Drawer**. Bei diesen Interaktionen handelt es sich um sogenannte Navigation Drawer. Diese öffnen typischerweise ein Menü, welches sich von einer Seite über die bisherige Ansicht schiebt. Die Interaktion ist normalerweise durch

drei horizontal übereinander platzierte Linien gekennzeichnet. Ein Beispiel ist in Abbildung 8.2 dargestellt.

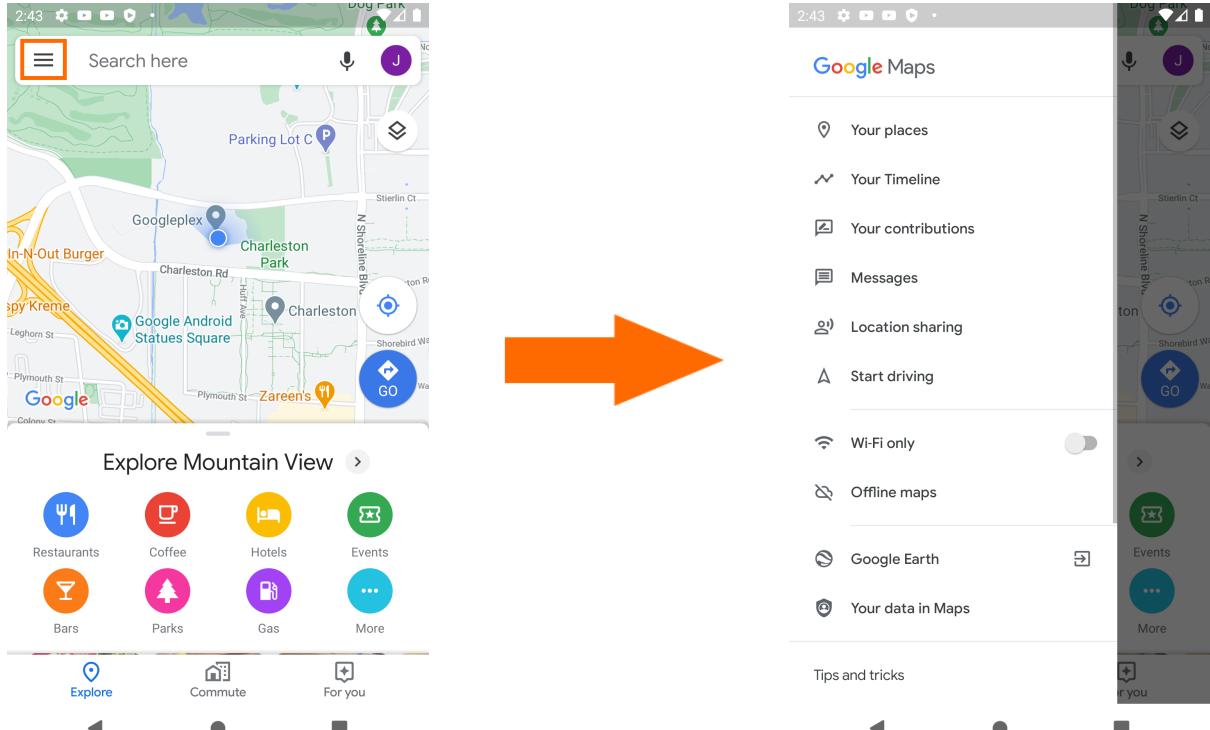


Abbildung 8.2: Veranschaulichung eines **Navigation Drawer** in der Anwendung „Maps“

Ein weiterer Interaktionsname, der definiert werden muss, ist der Knoten **Options**. Dieser ist nicht, wie der Name andeuten könnte, eine klassische Einstellungsansicht. Solche werden durch den Knoten **Settings** repräsentiert. Eine Interaktion, die den Namen **Options** trägt, wird häufig durch ein Piktogramm mit drei übereinander angeordneten Punkten gekennzeichnet. Ein Beispiel ist in der folgenden Abbildung 8.3 gegeben. Die Betätigung der Interaktion öffnet ein Untermenü, welches teilweise über der vorherigen Ansicht angezeigt wird.

Da der Graph aus Abbildung 8.1 aufgrund der hohen Anzahl an analysierten Anwendungen die Ergebnisse ein wenig unübersichtlich visualisiert, ist im Folgenden die Tabelle 8.2 angegeben, welche die am häufigsten vorkommenden Interaktionen auflistet. In über der Hälfte aller Anwendungen muss auf dem Interaktionsweg zu Datenschutzerklärungen gescrollt werden. Da das Scrollen jedoch im Normalfall nicht den thematischen Inhalt einer Ansicht verändert, sondern diesen nur erweitert, sind die weiteren Interaktionen von größerer Bedeutung.

In einem großen Teil der analysierten Anwendungen sind die Datenschutzhinweise durch Aufrufen der Einstellungen zu erreichen. Dies ist demnach eine Interaktion, auf die Nutzer*innen besonders achten sollten. Weiterhin kommt es in mehreren Fällen vor, dass die Datenschutzerklärungen in einer Ansicht mit Informationen über die Anwendung eingebettet sind. Diese kann über das Schlagwort **About** aufgerufen werden. Ungefähr 20 Prozent der getesteten Anwendungen nutzen für die Bereitstellung der Datenschutzhinweise eine Ansicht mit Informationen oder Verlinkungen zu rechtlichen Dokumenten oder Hinweisen. Die zugehörigen Schlagwörter lauten **Legal Information**, **Legal & Policies** oder **Legal**. Weitere mehrmals vorkommende Interaktionen sind der bereits erklärte **Navigation Drawer** sowie das Öffnen einer Ansicht für

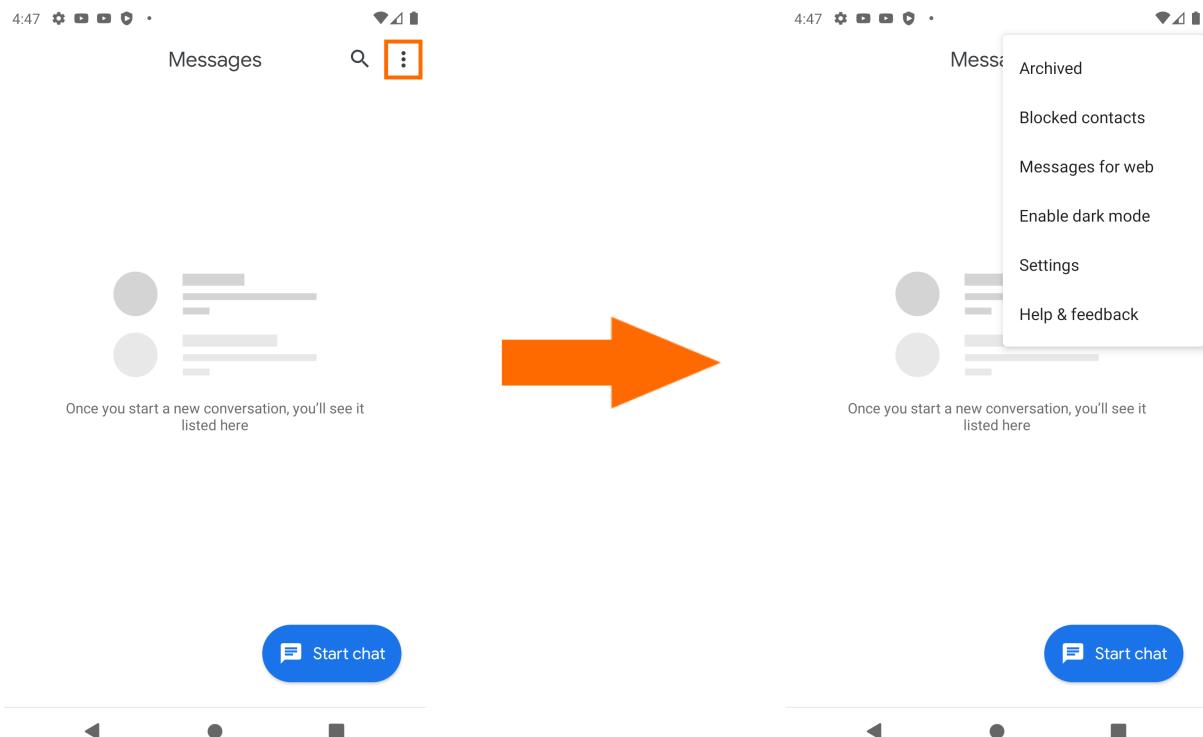


Abbildung 8.3: Veranschaulichung der Interaktion **Options** in der Anwendung „Messages“

Interaktion	Häufigkeit	Prozentualer Anteil
Scroll	54	56.8 %
Settings	45	47.4 %
About	38	40.0 %
Legal Information	19	20.0 %
Drawer	17	17.9 %
Account	16	16.8 %
Options	14	14.7 %
Help	10	10.5 %
Profile	8	8.4 %

Tabelle 8.2: Häufigkeit verschiedener Interaktionen

Einstellungen, die sich auf den **Account** oder das **Profile** der Nutzer*innen beziehen. In einigen wenigen Fällen sind die Datenschutzerklärungen ebenfalls in einer Hilfe-Ansicht zu finden.

Besonders ist in Abbildung 8.1 ebenfalls der Umstand zu erkennen, dass es sich bei den Interaktionen **Scroll** und **Settings** um große Knotenpunkte in dem Graphen handelt. Aus der Einstellungsansicht sind in vergleichsweise wenigen Fällen (7) direkt die Datenschutzhinweise zu erreichen. Somit fungiert diese Interaktion häufig als Übergangspunkt und ist demnach gut als Anhaltspunkt für die Suche nach Datenschutzhinweisen nutzbar. Wie die Einstellungsseite erreicht wird und wie die Datenschutzhinweise von dieser aus zu erreichen sind, ist in den analysierten Fällen unterschiedlich.

Weiterhin ist anzumerken, dass in den analysierten Anwendungen die Häufigkeit der Installationen keine Auswirkungen auf die Komplexität der Erreichung der Datenschutzhinweise

hat. In Abbildung 8.4 ist ein Diagramm zu sehen, welches sowohl die Durchschnittslänge als auch den Median der Ergebnispfade eines bestimmten Anteils der Gesamtanalyse angibt. Die Gesamtanalyse wurde in fünf Abschnitte mit jeweils 19 Anwendungen gegliedert, die nach der Installationszahl sortiert sind. Der erste Abschnitt beinhaltet die 19 meistinstallierten Anwendungen und jeder weitere Abschnitt beinhaltet die nächsten 19 Anwendungen. In der Abbildung ist zu erkennen, dass es in den analysierten Anwendungen keine Korrelation zwischen Installationszahl der Anwendung und der Bereitstellung der Datenschutzhinweise gibt.

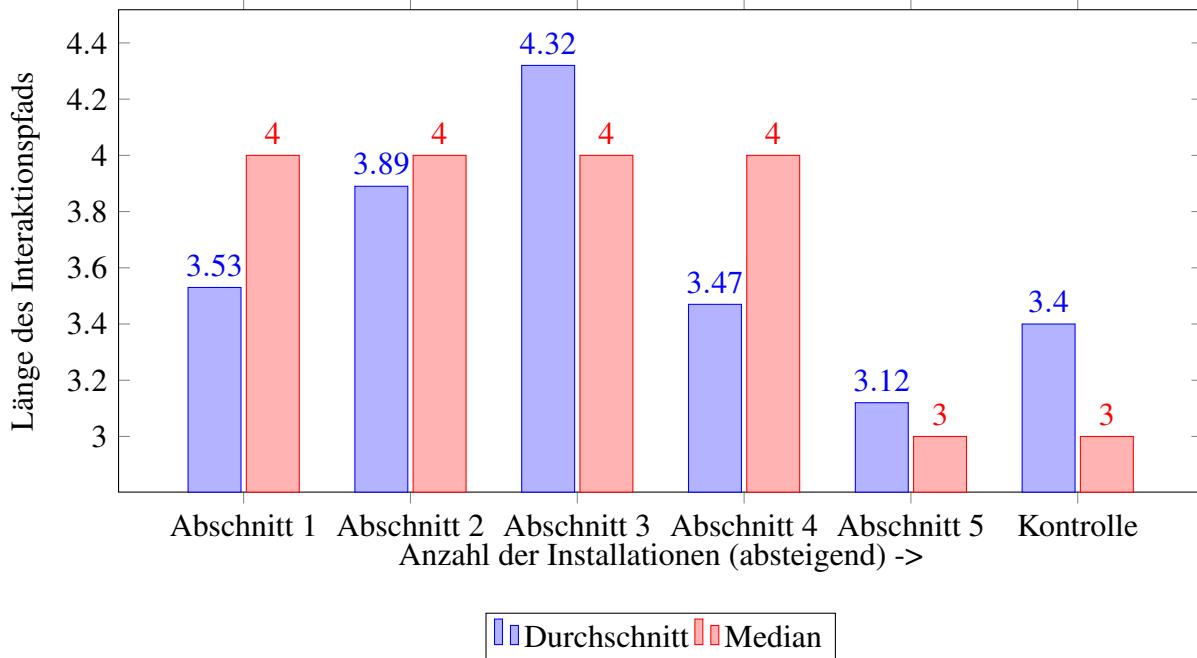


Abbildung 8.4: Durchschnitt und Median der 5 Quantile

Da es sich bei den analysierten Anwendungen um die 100 meistinstallierten Anwendungen auf der Android-Plattform handelt, ist die Statistik eventuell nicht aussagekräftig für alle Anwendungen des Betriebssystems. Aus diesem Grund wurde eine weitere Gruppe von Anwendungen analysiert (**Kontrolle**). Dies sind die untersten Anwendungen in der auf der Webseite „AndroidRank“ aufrufbaren Rangliste. Hierbei handelt es sich demnach um die Anwendungen auf den Plätzen 482 bis 500 [And19b]. Bei den Ergebnissen dieser Anwendungen ist ebenfalls kein wesentlicher Unterschied zu erkennen. Demnach hat bei den analysierten Anwendungen die Häufigkeit der Anwendungsinstallationen keinen Einfluss auf die Komplexität der Einbettung der Datenschutzhinweise.

Kategorie	Interaktionen der Kategorie
Account	Account, Me, Profile
Legal	Google Legal, Legal, Legal & Policies, Legal Information
Privacy	Privacy, Privacy & Security, Trust Centre
Menu	Daily, Drawer, Home, Menu, More, Options

Tabelle 8.3: Zusammenfassende Kategorien der Interaktionen

Um eine besser verständliche Visualisierung der Ergebnisse zu erhalten, können die Ergebnisse aus Abbildung 8.1 zusammenfassend dargestellt werden. Hierfür werden Interaktionen kombiniert betrachtet, die eine semantische Ähnlichkeit besitzen. Eine Übersicht über die Kategorien

und ihre Interaktionen ist in Tabelle 8.3 aufgeführt. Die Kategorie **Account** fasst alle Interaktionen zusammen, die eine Benutzerkontenansicht öffnen. Interaktionen der Kategorie **Legal** führen zu einer Ansicht mit rechtlichen Informationen, während mit **Privacy** Interaktionen betrachtet werden, welche eine Übersicht zu Einstellungen in Bezug auf die Privatsphäre aufrufen. Die letzte Kategorie **Menu** umfasst Interaktionen, die menüartige Ansichten öffnen. Demnach bleiben, neben dem Startknoten und der Datenschutzhinweise, die Interaktionen **Settings**, **Help** und **About** kategoriefrei. Das Scrollen wird hierfür nicht in Betracht gezogen, da es den Kontext einer Ansicht nicht verändert.

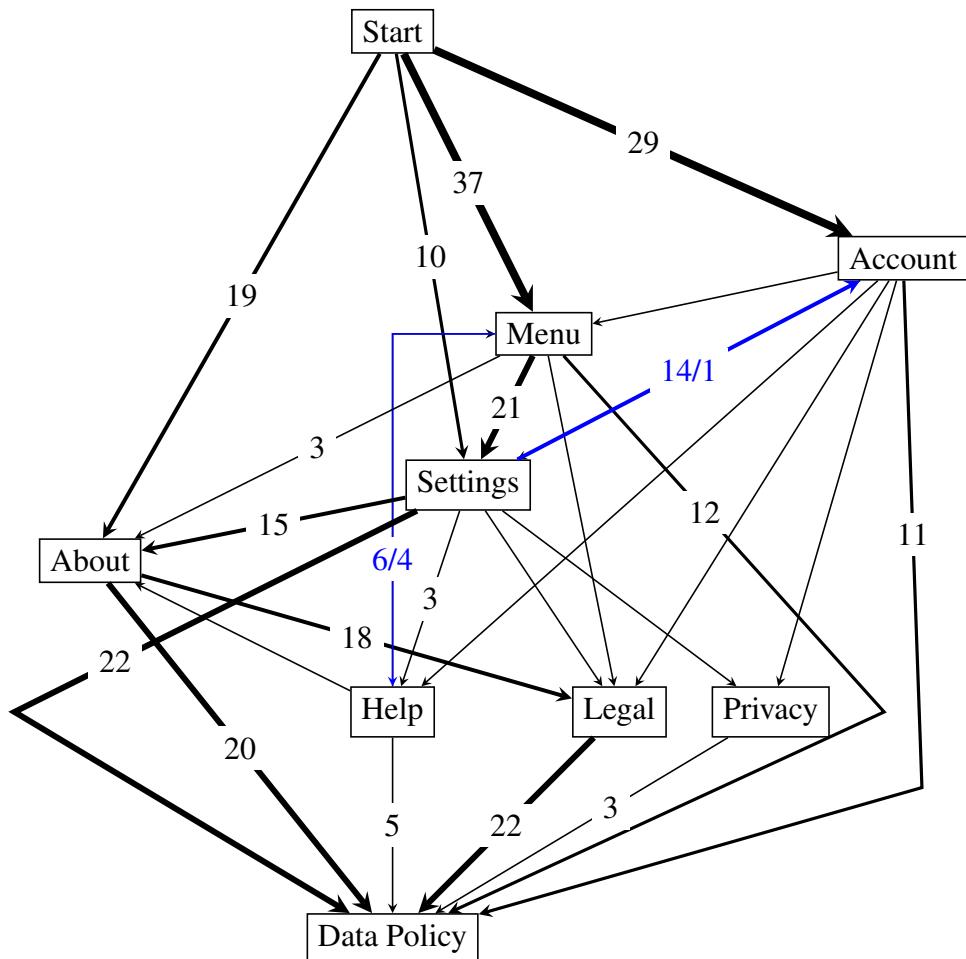


Abbildung 8.5: Kategorische Analyse aller getesteten Anwendungen

Abbildung 8.5 zeigt den Graphen zu der kategorischen Analyse der meistinstallierten Anwendungen. In dieser sind, analog zu dem vorherigen Graphen, Interaktionen der Gewichtung 1 oder 2 ungewichtet dargestellt. Vom Startbildschirm der Anwendung aus wird mit der größten Häufigkeit in 38.9 Prozent der Fälle eine menüartige Ansicht aufgerufen. In 30.5 Prozent aller Anwendungen wird in eine Ansicht zu einem Account navigiert. Darauf folgen die beiden Interaktionen **About** und **Settings**.

Obwohl **Settings** die Interaktion ist, die am seltensten vom Startbildschirm aufgerufen wird, wird diese am häufigsten genutzt. Die Häufigkeit der verschiedenen Interaktionen beziehungsweise Interaktionskategorien ist in Tabelle 8.4 aufgeführt. Neben **Settings** sind menüartige Interaktionen und **About** besonders häufig vertreten. In ungefähr 30 Prozent der Anwendungen befinden sich Ansichten zu einem Benutzeraccount oder eine rechtliche Informationsseite auf dem Weg zu den Datenschutzerklärungen.

Interaktion	Häufigkeit	Prozentualer Anteil
Settings	45	47.4 %
Menu	43	45.3 %
About	38	40.0 %
Account	30	31.6 %
Legal	30	31.6 %
Help	10	10.5 %
Privacy	3	3.2 %

Tabelle 8.4: Häufigkeit der kategorischen Interaktionen

Somit sollten Nutzer*innen auf der Suche nach Datenschutzhinweisen besonders darauf achten, vom Startbildschirm der Anwendungen eine Menüansicht oder eine Ansicht zu einem Benutzerkonto zu öffnen. Darauffolgend sind insbesondere Einstellungsseiten relevant. Von diesen aus sollte besonders auf Informationsseiten mit dem Stichwort **About** und **Legal** geachtet werden. Die Datenschutzhinweise sind am häufigsten direkt in einer Einstellungsseite, einer Informationsseite mit rechtlichen Informationen oder einer reinen Informationsseite (**About**) eingebunden.

8.2 Anwendungen vom selben Hersteller

In diesem Abschnitt werden Anwendungen aus der Liste der 100 meistinstallierten Anwendungen untersucht, die aus demselben Entwicklerhaus stammen. Insbesondere soll analysiert werden, ob es bei Anwendungen, die von demselben Unternehmen entwickelt werden, Gemeinsamkeiten in der Bereitstellung der Datenschutzhinweise gibt. Dies würde Nutzer*innen helfen, da diese bei Anwendungen desselben Herstellers eventuell einem Muster folgen könnten, welches in anderen Anwendungen dieses Herstellers bereits verwendet wurde.

Die erste Gruppe von analysierten Anwendungen eines gemeinsamen Entwicklerhauses sind Anwendungen des Technologieunternehmens „Microsoft“. Hierzu gehören vier Anwendungen der Software-Sparte „Office“ [Mic20a], die Videotelefonat-Anwendung „Skype“ [Mic20d] sowie eine Anwendung für den Cloud-Speicher-Dienst von Microsoft [Mic20b]. Weiterhin wurde die Tastatur-Anwendung „SwiftKey“ [Sto20d] für die Android-Plattform analysiert. Der die Ergebnisse visualisierende Graph ist in Abbildung 8.6 gezeigt.

Die durchschnittliche Pfadlänge in den mobilen Anwendungen des Unternehmens beträgt **3.86** Interaktionen, während der Median der sieben Ergebnisse bei **4** Interaktionen liegt. Somit sind die Datenschutzhinweise in diesen Anwendungen durchschnittlich mit mehr Interaktionen zu erreichen als der Durchschnitt aller analysierten Anwendungen. Sowohl die Interaktion **Settings** als auch die Interaktion **Scroll** sind am häufigsten vertreten. Die Interaktion **Account** liegt auch mehrmals auf Ergebnispfaden. Die restlichen Interaktionen sind jeweils nur einzeln vertreten. Demnach ähneln die Ergebnisse der Interaktionshäufigkeit denen der Gesamtanalyse.

Die Anwendung mit dem kürzesten Ergebnispfad ist die Anwendung „SwiftKey“. In dieser wird der Pfad "START;ACCOUNT;DATAPOLICY;" benötigt, um zu den Datenschutzhinweisen zu gelangen. Der längste Weg ist in der Telefonieanwendung „Skype“ eingebunden. Hier sind fünf Interaktionen nötig. Besonders hervorzuheben ist die zum Office-Paket zugehörige Anwendung

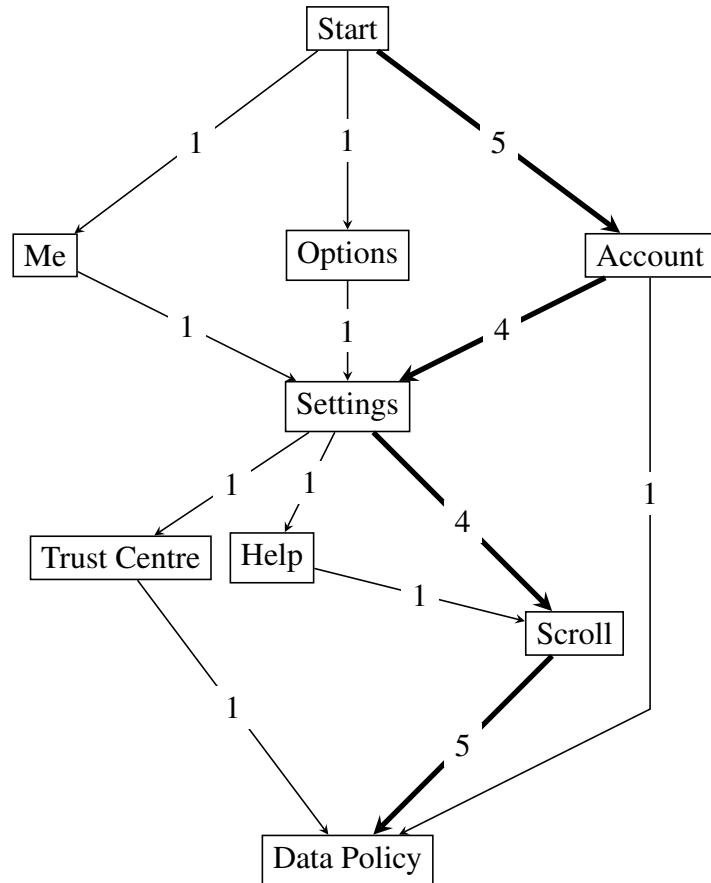


Abbildung 8.6: Ergebnisgraph der Anwendungsanalyse des Unternehmens „Microsoft“

„OneNote“ [Mic20c]. Die Anwendung zum Erstellen von digitalen Notizen bindet die Datenschutzhinweise durch eine Ansicht mit dem Namen **Trust Centre** ein. Diese befindet sich in den Anwendungseinstellungen und verweist sowohl auf die Datenschutzhinweise der Anwendung als auch auf die Einstellungen zur Privatsphäre in der Anwendung.

Die zweite Gruppe von Anwendungen, die einem Unternehmen zugehörig sind, sind die Anwendungen des sozialen Netzwerks „Facebook“. Der Ergebnisgraph der sechs getesteten Dienste ist in Abbildung 8.7 zu sehen.

Die durchschnittliche Pfadlänge in den Anwendungen ist mit **4.33** Interaktionen länger als die Durchschnittslänge aus der Gesamtanalyse. Ebenso übersteigt diese den Wert der zuvor analysierten Gruppe von Anwendungen des Unternehmens „Microsoft“. Der Median liegt bei **4** Interaktionen und gleicht somit dem der Gesamtanalyse. Die einzelnen Pfade sind zwischen vier und fünf Interaktionen lang und liegen demnach alle über dem Durchschnitt der Gesamtanalyse.

Weiterhin ist aus dem Graph erkennbar, dass nur wenige Interaktionen in mehreren Anwendungen verwendet werden. Hieraus lässt sich schließen, dass die Einbettung der Datenschutzerklärungen in den Anwendungen des Unternehmens auf verschiedene Arten gelöst wird. Die beiden Knoten **Settings** und **Scroll** werden mehr als zweimal genutzt und sind somit die am häufigsten auftretenden Interaktionen. Dies stimmt mit der Gesamtanalyse überein.

Anzumerken ist außerdem, dass sich unter den analysierten Anwendungen jeweils zwei Dienste ähneln. Sowohl die Anwendung „Facebook“ als auch der Nachrichtendienst „Messenger“ wurden

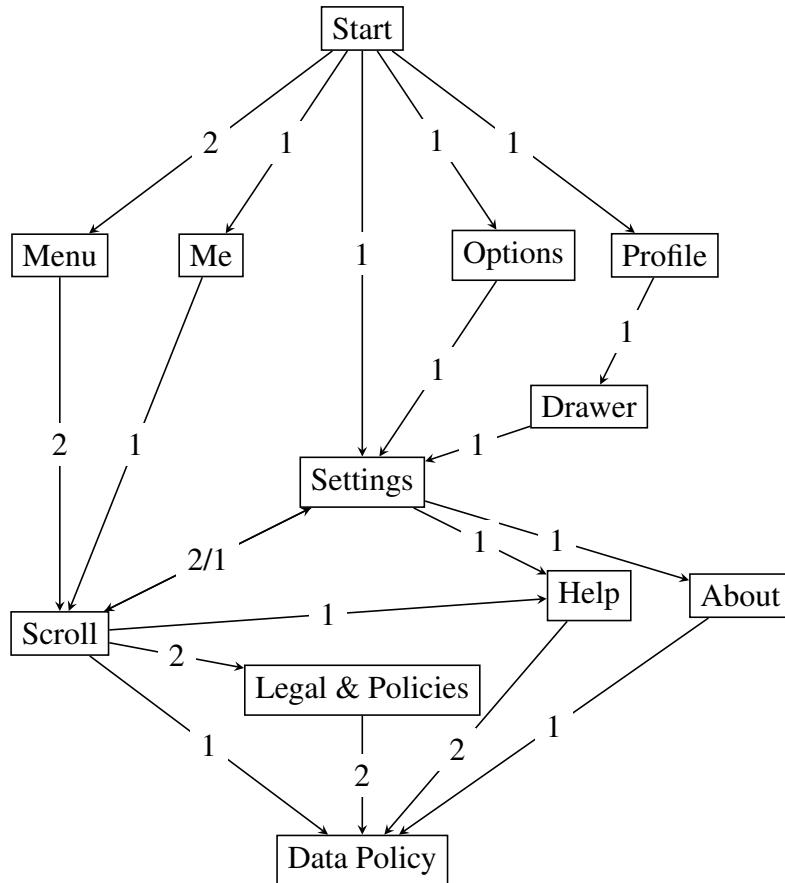


Abbildung 8.7: Ergebnisgraph der Anwendungsanalyse des Unternehmens „Facebook“

beide in ihrer normalen Ausgabe analysiert. Weiterhin wurden die „Lite-Versionen“ dieser Anwendungen getestet. Diese sind so konzipiert, dass sie weniger Netzwerkkapazitäten benötigen [Sto20a]. Während die Einbettung der Datenschutzerklärungen in den Anwendungen des Nachrichtendienstes analog geregelt ist, sind diese in der Anwendung „Facebook Lite“ anders zu erreichen als in der normalen Anwendungsvariante des sozialen Netzwerks.

Abbildung 8.8 veranschaulicht die Ergebnisse aus allen Anwendungen der Gesamtanalyse, die dem Unternehmen „Google“ zugehörig sind. Diese Analyse ist besonders von Bedeutung, da das Betriebssystem Android ebenfalls durch das Unternehmen betrieben wird. Demnach nehmen diese Anwendungen eine Art Vorbildfunktion für andere Entwickler*innen ein. In dem Graph werden analog zum Graph der Gesamtanalyse aus Gründen der Übersichtlichkeit Kanten mit dem Gewicht **1** ungewichtet dargestellt.

Insgesamt wurden für diesen Vergleich 23 Anwendungen des Unternehmens verwendet. Hierbei wurden Systemdienste ausgelassen. Die Ergebnispfade haben Längen von zwei bis sechs Interaktionen. Die Durchschnittslänge beträgt **2.87** und der Median liegt bei **2** Interaktionen. Demnach schneiden die Anwendungen des Softwareentwicklers besser ab als der Durchschnitt der Gesamtanalyse. Besonders der geringe Wert des Medians ist hervorzuheben. In einem Großteil der analysierten Anwendungen können die Datenschutzerklärung mit lediglich zwei Interaktionen erreicht werden.

Die meistverwendete Interaktion ist das Öffnen der Ansicht **Account**. Von dieser führt in zehn von elf Fällen eine Interaktion direkt zu den Datenschutzhinweisen. Weiterhin wird der Knoten **Options** häufig genutzt. Die Anwendungen von „Google“ folgen demnach nicht den typischen,

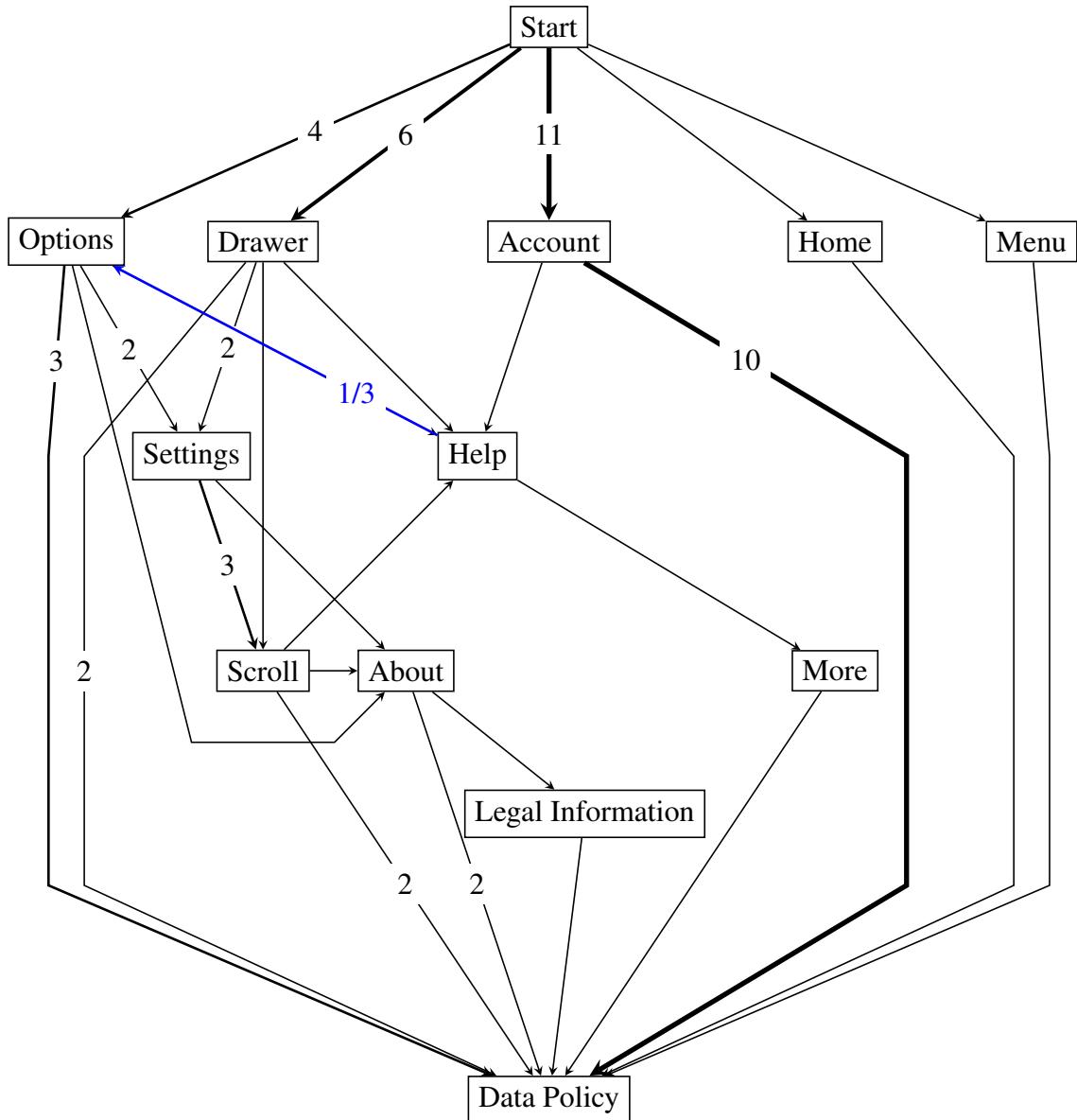


Abbildung 8.8: Ergebnisgraph der Anwendungsanalyse des Unternehmens „Google“

in der Gesamtanalyse vorgestellten Interaktionen. Die Knoten **Settings** und **Scroll** werden lediglich jeweils viermal verwendet.

Zwei Anwendungen des Unternehmens sind negativ hervorzuheben. Eine von diesen ist die zum Play Store gehörige Anwendung „Play Games“ [Sto20c]. Diese ist die einzige Anwendung des Unternehmens, in der weder durch *DataPolicyRipper* noch durch eine darauffolgende handische Analyse Datenschutzhinweise gefunden werden konnten. Die zweite Anwendung ist der Webbrowser „Google Chrome“ [Sto20b]. In dieser sind die Datenschutzhinweise mit sechs Interaktionen zu erreichen. Demnach ist der Webbrowser eine der Anwendungen, die den längsten Interaktionspfad benötigen. Dies könnte darin begründet sein, dass es sich um eine Browseranwendung handelt. Solche Anwendungen funktionieren häufig nach einem ähnlichen Schema und sind nicht wie typische mobile Anwendungen aufgebaut. Um dies zu überprüfen, werden im nächsten Abschnitt *Anwendungen gleicher Art* analysiert.

8.3 Anwendungen gleicher Art

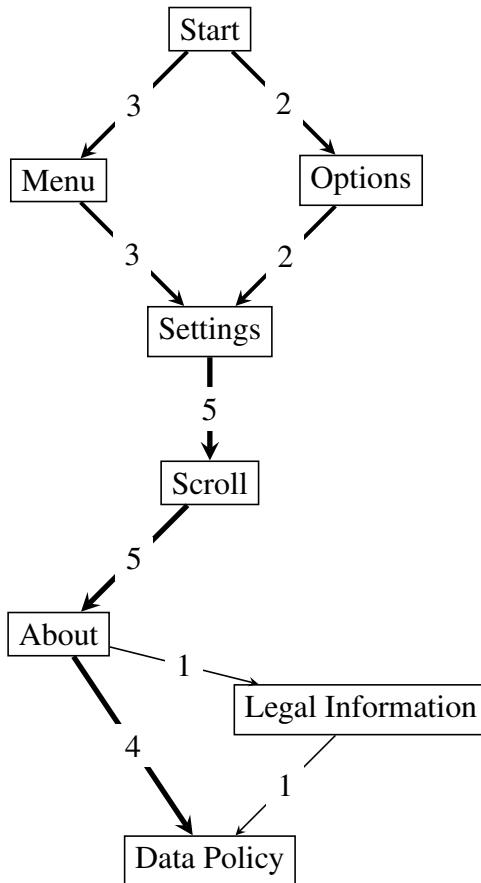


Abbildung 8.9: Ergebnisgraph der Anwendungsanalyse vom Typ Webbrowser

Da sich, wie eben beschrieben, die Bereitstellung der Datenschutzhinweise in der Browseranwendung von „Google“ sehr stark von den anderen Anwendungen des Unternehmens unterscheidet, wird im Folgenden die Erreichbarkeit der Datenschutzhinweise in weiteren Browseranwendungen genauer untersucht. Neben „Google Chrome“ wurden ebenfalls fünf weitere Browseranwendungen betrachtet, die auf dem deutschen Markt vertreten sind. Dies sind die Anwendungen „Samsung Internet“, „Firefox“, „Opera“, „UC Browser“ und „Edge“ [Sta20b]. In Anwendungen für das Browsen im Internet ist die Erreichbarkeit von Datenschutzerklärungen relevant, da in diesen insbesondere Browserverläufe, Suchverläufe und weitere relevante personenbezogene Daten gespeichert und verarbeitet werden. Aus diesem Grund ist interessant zu analysieren, ob die deutlich schwieriger zu erreichenden Datenschutzhinweise in der Anwendung „Google Chrome“ eine spezifische Eigenschaft der Browseranwendungen darstellen.

In Abbildung 8.9 ist der Ergebnisgraph der Analyse zu Browseranwendungen abgebildet. Hier ist ebenfalls wichtig anzumerken, dass in der Anwendung „UC Browser“ [Sto20f] weder durch *DataPolicyRipper* noch durch die nachfolgende händische Analyse eine Datenschutzerklärung gefunden werden konnte. Die Bereitstellung von Datenschutzhinweisen in den anderen fünf Anwendungen ähnelt einander sehr. Die Pfadlängen betragen fünf bis sechs Interaktionen und die durchschnittliche Pfadlänge beträgt **5.2** Interaktionen. Der Median liegt bei **5** Interaktionen. In allen der verschiedenen Browseranwendungen wird zuerst eine menüartige Ansicht geöffnet, über die darauffolgend die Einstellungen geöffnet werden. In diesen wird gescrollt und eine Informationsseite wird aufgerufen. Lediglich in „Google Chrome“ muss von dieser aus noch

eine Ansicht mit rechtlichen Informationen geöffnet werden, bevor die Datenschutzerklärung angezeigt werden kann. In den anderen Anwendungen kann diese von der Informationsseite aus direkt geöffnet werden.

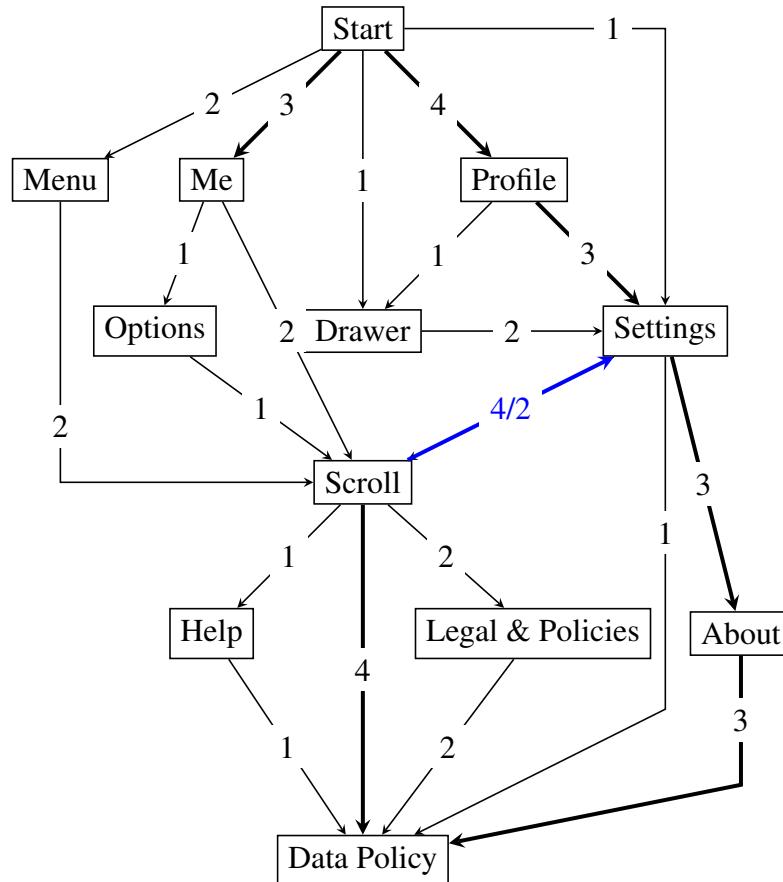


Abbildung 8.10: Ergebnisgraph der Anwendungsanalyse vom Typ Soziales Netzwerk

Hieraus lässt sich schließen, dass die Bereitstellung der Datenschutzhinweise in Browseranwendungen zwar unterdurchschnittlich gehandhabt ist, jedoch ist diese in den auf dem Markt dominierenden Anwendungen sehr ähnlich geregelt. Ist Nutzer*innen dieses Schema demnach bekannt, können diese die Hinweise in den Browseranwendungen gut erreichen.

Die zweite relevante Art an Anwendungen auf mobilen Geräten sind Anwendungen für soziale Netzwerke. In diesen werden viele personenbezogene Daten verwaltet und Datenschutz ist hier ein kritischer zu betrachtender Aspekt. Daher wurden elf Anwendungen von sozialen Netzwerken, die unter den 100 meistinstallierten Anwendungen zu finden sind, analysiert. Die Ergebnisse dieser Analyse sind in Abbildung 8.10 veranschaulicht.

Die Pfadlängen zu den Datenschutzhinweisungen der Anwendungen liegen zwischen drei und fünf Interaktionen. Der Median beträgt **4** Interaktionen. Der Durchschnitt der elf Anwendungen hat eine Pfadlänge von **4.18**. Somit sind die Datenschutzhinweise in den Anwendungen der sozialen Netzwerke schwieriger zu erreichen als die durchschnittliche Anwendung aus der Gesamtanalyse.

In dem Graphen aus Abbildung 8.10 ist zu erkennen, dass es relativ wenig Gemeinsamkeiten in der Bereitstellung der Datenschutzhinweise gibt. Mit neun und acht Interaktionen sind **Scroll** und **Settings** am häufigsten vertreten. Demnach folgt dies der Tendenz, die ebenfalls in der

Gesamtanalyse beobachtet werden konnte. Mit den Interaktionen **Profile** und **Me** werden in sieben der Anwendungen (63.6 %) als erstes Ansichten zu den Benutzerkonten geöffnet. Hierauf sollten Nutzer*innen folglich achten.

Diese Erkenntnisse legen nahe, dass es Anwendungsarten gibt, in denen sich die Bereitstellung der Datenschutzhinweise ähnelt, wie die Analyse der Browseranwendungen ergeben hat. Jedoch gibt es auch Kategorien von Anwendungen, in denen Entwickler*innen die Datenschutzerklärungen auf sehr unterschiedlichen Wegen einbinden. Demnach ist die Art einer Anwendung im Normalfall kein Indikator dafür, wie die Datenschutzhinweise in dieser zu erreichen sind.

9 Fazit

Um diese Arbeit abzuschließen, wird anfänglich der Blick auf mögliche, erweiternde *Zukünftige Arbeiten* in Bezug auf diese Arbeit gerichtet. Im Anschluss werden die Ergebnisse dieser Arbeit zusammenfassend bewertet und Best Practices für sowohl Anwendungsentwickler*innen als auch Nutzer*innen in Bezug auf die Erreichbarkeit von Datenschutzhinweisen in mobilen Anwendungen des Betriebssystems Android vorgeschlagen.

9.1 Zukünftige Arbeiten

Zukünftige wissenschaftliche Arbeiten, die das Thema dieser Arbeit aufgreifen und erweitern könnten, können in zwei Bereiche gegliedert werden: dem Anpassen des Verfahrens *DataPolicyRipper* sowie das Erweitern der Analyse der Datenschutzhinweise.

Zukünftige Arbeiten, welche das Verfahren anpassen, könnten die in dieser Arbeit thematisierten Limitationen betrachten und Lösungen für diese entwickeln. Durch *DataPolicyRipper* können hauptsächlich native und hybride Anwendungen analysiert werden, die den Richtlinien des „Material Design“ [Goo] folgen und demnach beispielsweise Beschreibungen für Piktogramme implementieren. Demnach könnte das Verfahren zum Beispiel für webbasierte Anwendungen oder Spiele erweitert werden. Hierfür müsste womöglich ein anderes als in dieser Arbeit verwendetes Framework genutzt werden.

Weitere Ansätze wären die Portierung der Methodik für die Nutzung in einem weiteren Betriebssystem. Beispielsweise würde sich iOS, das nach Android am zweitmeisten genutzte Betriebssystem für mobile Geräte [Sta20a], anbieten. Ebenfalls ist die Anpassung des Verfahrens *DataPolicyRipper* für Anwendungen anderer Sprachen denkbar. Durch solche Erweiterungen könnte analysiert werden, ob es Unterschiede zu den in dieser Arbeit vorgestellten Ergebnissen in der Erreichbarkeit von Datenschutzhinweisen in Anwendungen anderer Sprachen gibt. Hier würde sich auch der deutsche Anwendungsmarkt anbieten. Dies würde eine Kombination aus einer Anpassung des Verfahrens und einer erweiterten Analyse darstellen.

Andere Analyseerweiterungen könnten durch die Kombination dieser Arbeitsergebnisse und Verfahren aus der bereits anfänglich genannten Literatur zum Stichwort „Usable Privacy“ eine tiefergehende Betrachtung der ganzheitlichen Bereitstellung von Datenschutzerklärungen realisieren. Durch eine Methodik dieser Art könnte sowohl die Erreichbarkeit von Datenschutzerklärungen betrachtet als auch die natürlichsprachliche Verständlichkeit der rechtlichen Erklärungen analysiert werden. Somit könnte die Bereitstellung der Datenschutzhinweise kombinatorisch als Formel aus Erreichbarkeit und Verständlichkeit bewertet werden.

Eine mögliche ergänzende Art der Erweiterung der Analyse ist die Betrachtung weiterer Anwendungen und Anwendungsgruppen des Betriebssystems Android. Der Fokus dieser Arbeit liegt auf der Entwicklung und Vorstellung des Verfahrens. Weitere Arbeiten hätten die Möglichkeit dieses zu nutzen, um tiefergehende Erkenntnisse zu erhalten. Eine interessante Betrachtung könnte die Gruppierung von Anwendungen nach den Berechtigungen, die diese für die Nutzung

benötigen, darstellen. Auch die Analyse der Erreichbarkeit von Datenschutzhinweisen in Anwendungen, die sich derzeit in der Entwicklungsphase befinden (Open Beta), könnte relevant sein.

9.2 Bewertung der Analyseergebnisse

In dieser Arbeit wurde die Entstehung und die Funktionsweise des Verfahrens zum Messen der Erreichbarkeit von Datenschutzhinweisen mit dem Namen *DataPolicyRipper* ausführlich behandelt. Die Analyse mehrerer Anwendungen durch das Programm *DataPolicyRipper* hat ergeben, dass die Einbettung der Datenschutzerklärungen in mobilen Anwendungen des Betriebssystems Android nicht einheitlich geregelt ist. Um die Bewertung der Bereitstellung dieser relevanten Informationen durch Entwickler*innen zu ermöglichen, kann die Durchschnittslänge der Interaktionspfade der analysierten Anwendungen verwendet werden. Zusammenfassend lässt sich feststellen, dass eine Interaktionsfolge von drei oder weniger Schritten als positiv anzusehen ist, da dies über dem Durchschnitt der analysierten Anwendungen liegt. Anwendungen, in denen Datenschutzhinweise mit fünf oder sechs Interaktionen erreicht werden, könnten die Einbettung dieser Hinweise verbessern.

Mögliche Best Practices für Anwendungsentwickler*innen werden vom Bundesministerium der Justiz und für Verbraucherschutz vorgestellt. Ein Beispiel für die Handhabung der datenschutzrechtlichen Informationen und Einstellungen ist die Darbietung dieser an einer gut erreichbaren Stelle in der mobilen Anwendung, wie unter einem Menüpunkt „Datenschutz“ [Jus17]. Gemäß dem Ministerium ist neben dieser „zentrale[n] leicht aufzufindene[n]“ [Jus17, S.5] Position für eine gute Implementation in mobilen Anwendungen eine Seite mit Kurzinformationen wünschenswert, die dem Betroffenen die wichtigsten Hinweise anschaulich darstellt und auf die ausführlicheren Informationsseiten verlinkt.

Dieser Vorgabe ist sich anzuschließen. Jedoch haben die Ergebnisse der Analyse gezeigt, dass in der durchschnittlichen Anwendung solche Best Practices nicht befolgt werden. Besonders zusammenfassende Ansichten mit Kurzinformationen sind eine Seltenheit. Die Interaktion zu solchen Informationsseiten ist mit 3.2 Prozent die am wenigsten verwendete Interaktion (siehe Tabelle 8.3). Wesentlich häufiger werden Ansichten mit einer einfachen Verlinkung zu allen rechtlichen Dokumenten verwendet (Interaktion **Legal**). Diese sind jedoch wenig informativ und dienen nicht der Veranschaulichung der wichtigsten Hinweise.

Eine weitere mögliche zukünftige Lösung für die leichtere Erreichbarkeit der Datenschutzhinweise wäre eine zentrale Stelle für die Bereitstellung der Datenschutzhinweise aller installierten Anwendungen im Betriebssystem. Dies könnte beispielsweise über einen Punkt in den Einstellungen des Systems geregelt sein, unter dem die Datenschutzerklärungen der installierten Anwendungen einsehbar wären. Dies würde es Nutzer*innen erleichtern, die relevanten Informationen ohne Probleme erreichen zu können, da diese an einer gewohnten Stelle auffindbar wären.

In der derzeitigen Situation sollten Nutzer*innen an erster Stelle nach einer menüartigen Ansicht Ausschau halten, um von dieser zu den Einstellungen der Anwendung zu gelangen. Von hier aus können die Datenschutzerklärungen sowohl direkt erreichbar sein oder hinter Hilfeseiten, Ansichten zu Anwendungsinformationen sowie rechtlichen Informationsseiten verfügbar sein.

Anzumerken ist weiterhin, dass das Wissen über die Erreichbarkeit von Datenschutzhinweisen in einer Anwendung eines Herstellers nicht unbedingt bedeutet, dass dies zur erleichterten Erreichbarkeit dieser in anderen Anwendungen dieses Herstellers führt. Jedoch gibt es hier ebenfalls ein Positivbeispiel. In den meisten Anwendungen des Unternehmens „Google“ sind die Datenschutzhinweise mit lediglich zwei Interaktionen zu erreichen und das Schema für diese Erreichbarkeit ähnelt sich in den Anwendungen stark. Doch könnte das Unternehmen als Betreiber des Betriebssystems weiterhin die Anwendungsentwickler*innen für die Plattform dazu anhalten, bestimmten Best Practices zu folgen und diese beispielsweise auch in das „Material Design“ [Goo] einbinden.

Um auf den Ausgangspunkt dieser Arbeit einzugehen, kann ebenfalls die EU für die schwammige Formulierung in der DSGVO kritisiert werden. Indem die geforderte „präzise[], transparente[], verständliche[] und leicht zugängliche[] Form“ [Med19, S.146] nicht weiter spezifiziert wird, ist eine Chance auf eine Harmonisierung der Einbindung der Datenschutzhinweise, nicht nur in mobilen Anwendungen, sondern ebenfalls in allen Softwarelösungen, verpasst worden. Dies könnte es Nutzer*innen und Entwickler*innen zugleich vereinfachen, Datenschutzhinweise zu handhaben.

Literatur

- [Aca14] Software Testing Academy. *Software Testing Methoden*. 2014. URL: <https://www.software-testing.academy/software-testing-methoden.html#dynamisch> (besucht am 08.07.2020).
- [Ama+12a] Domenico Amalfitano u. a. *A toolset for GUI testing of Android applications*. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE. 2012, S. 650–653.
- [Ama+12b] Domenico Amalfitano u. a. *Using GUI ripping for automated testing of Android applications*. In: *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE. 2012, S. 258–261.
- [And19a] Google APIs for Android. *Overview of Google Play Services*. 2019. URL: <https://developers.google.com/android/guides/overview> (besucht am 02.07.2020).
- [And19b] AndroidRank. *List of Android Most Popular Google Play Apps*. 2019. URL: <https://www.androidrank.org/android-most-popular-google-play-apps?category=all&sort=4&price=free> (besucht am 28.09.2020).
- [Ber19] Daniel Berger. *Vor fünf Jahren kaufte Facebook WhatsApp - Wie geht es weiter?* heise online. Feb. 2019. URL: <https://www.heise.de/newsticker/meldung/Vor-fuenf-Jahren-kaufte-Facebook-WhatsApp-wie-geht-es-weiter-4312349.html> (besucht am 31.05.2020).
- [Deu20] Samsung Deutschland. *Samsung Galaxy S7 und S7 edge*. 2020. URL: <https://www.samsung.com/de/smartphones/galaxy-s7/overview/> (besucht am 27.09.2020).
- [Doc19a] Android Developers Documentation. *App Crawler*. 2019. URL: <https://developer.android.com/training/testing/crawler> (besucht am 16.07.2020).
- [Doc19b] Android Developers Documentation. *Application Fundamentals*. 2019. URL: <https://developer.android.com/guide/components/fundamentals> (besucht am 09.08.2020).
- [Doc19c] Android Developers Documentation. *Introduction to Activities*. 2019. URL: <https://developer.android.com/guide/components/activities/intro-activities> (besucht am 24.06.2020).
- [Doc19d] Android Developers Documentation. *Make apps more accessible*. 2019. URL: <https://developer.android.com/guide/topics/ui/accessibility/apps> (besucht am 12.10.2020).
- [Doc19e] Android Developers Documentation. *Picture-in-picture Support*. 2019. URL: <https://developer.android.com/guide/topics/ui/picture-in-picture> (besucht am 25.09.2020).
- [Doc19f] Android Developers Documentation. *UI Automator*. 2019. URL: <https://developer.android.com/training/testing/ui-automator> (besucht am 06.07.2020).

- [Doc19g] Android Developers Documentation. *User Interface & Navigation*. 2019. URL: <https://developer.android.com/guide/topics/ui> (besucht am 18.06.2020).
- [Doc20a] Android Developers Documentation. *AlertDialog*. 2020. URL: <https://developer.android.com/reference/android/app/AlertDialog> (besucht am 25.09.2020).
- [Doc20b] Android Developers Documentation. *Build instrumented unit tests*. 2020. URL: <https://developer.android.com/training/testing/unit-testing/instrumented-unit-tests> (besucht am 09.08.2020).
- [Doc20c] Android Developers Documentation. *Dialog*. 2020. URL: <https://developer.android.com/reference/android/app/Dialog> (besucht am 25.09.2020).
- [Doc20d] Android Developers Documentation. *HorizontalScrollView*. 2020. URL: <https://developer.android.com/reference/android/widget/HorizontalScrollView> (besucht am 06.07.2020).
- [Doc20e] Android Developers Documentation. *Layouts*. 2020. URL: <https://developer.android.com/guide/topics/ui/declaring-layout> (besucht am 18.06.2020).
- [Doc20f] Android Developers Documentation. *Meet Android Studio*. 2020. URL: <https://developer.android.com/studio/intro> (besucht am 13.07.2020).
- [Doc20g] Android Developers Documentation. *ScrollView*. 2020. URL: <https://developer.android.com/reference/android/widget/ScrollView> (besucht am 06.07.2020).
- [Doc20h] Android Developers Documentation. *Test UI for multiple apps*. 2020. URL: <https://developer.android.com/training/testing/ui-testing/uiautomator-testing> (besucht am 09.09.2020).
- [Doc20i] Android Developers Documentation. *Test UI for single app*. 2020. URL: <https://developer.android.com/training/testing/ui-testing/espresso-testing> (besucht am 16.07.2020).
- [Doc20j] Android Developers Documentation. *View*. 2020. URL: <https://developer.android.com/reference/android/view/View> (besucht am 06.07.2020).
- [Eth20] Ethnologue. *What are the top 200 most spoken languages?* 2020. URL: <https://www.ethnologue.com/guides/ethnologue200> (besucht am 27.09.2020).
- [Fed20] Prof. Dr.-Ing. Hannes Federrath. *Datenschutz in der Informationsgesellschaft*. Folien der Vorlesung DIG im Wintersemester 2019/20. 2020. URL: <https://svs.informatik.uni-hamburg.de/teaching/DIG.pdf> (besucht am 21.03.2020).
- [Git20] GitHub. *jnsdmn/Bachelorthesis_JannisDammann_ErreichbarkeitDatenschutzhinweiseAndroid*. Okt. 2020. URL: https://github.com/jnsdmn/bachelorthesis_JannisDammann_ErreichbarkeitDatenschutzhinweiseAndroid (besucht am 16.10.2020).
- [Goo] Google. *Homepage - Material Design*. URL: <https://material.io/> (besucht am 24.06.2020).
- [Gue16] Andre Da Cruz Guerreiro. *GUI ripping of iOS mobile applications*. In: *Hamburg University of Technology* (2016).
- [Inn13] oose Innovative Informatik GmbH. *Notationsübersicht UML 2.5*. 2013. URL: <https://www.oose.de/wp-content/uploads/2012/05/UML-Notations%C3%BCbersicht-2.5.pdf> (besucht am 14.08.2020).

- [Jus17] Bundesministerium der Justiz und für Verbraucherschutz. *Verbraucherfreundliche Best-Practice bei Apps*. Feb. 2017. URL: https://www.bmjjv.de/SharedDocs/Downloads/DE/Service/StudienUntersuchungenFachbuecher/Apps_Best_Practise_StiWa_DE.pdf?__blob=publicationFile&v=1 (besucht am 22.04.2020).
- [Liu+14] Fei Liu u. a. *A step towards usable privacy policy: Automatic alignment of privacy statements*. In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. 2014, S. 884–894.
- [MBN03] Atif Memon, Ishan Banerjee und Adithya Nagarajan. *GUI ripping: Reverse engineering of graphical user interfaces for testing*. In: *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings*. IEEE. 2003, S. 260–269.
- [Med19] Appel Klinger Druck und Medien GmbH. *INFO 01*. Hrsg. von Der Bundesbeauftragte für den Datenschutz und die Informationsfreiheit. Juni 2019.
- [Mem+13] Atif Memon u. a. *The first decade of GUI ripping: Extensions, applications, and broader impacts*. In: *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE. 2013, S. 11–20.
- [Mic20a] Microsoft. *Microsoft Office*. 2020. URL: <https://www.microsoft.com/de-de/microsoft-365/microsoft-office?rtc=1> (besucht am 08.10.2020).
- [Mic20b] Microsoft. *OneDrive*. 2020. URL: <https://www.microsoft.com/de-de/microsoft-365/onedrive/online-cloud-storage> (besucht am 08.10.2020).
- [Mic20c] Microsoft. *OneNote, die App für digitale Notizen*. 2020. URL: <https://www.microsoft.com/de-de/microsoft-365/onenote/digital-note-taking-app?ms.url=ononetocom&rtc=1> (besucht am 09.10.2020).
- [Mic20d] Microsoft. *Skype*. 2020. URL: <https://www.microsoft.com/de-de/microsoft-365/microsoft-office?rtc=1> (besucht am 08.10.2020).
- [Pla19] Google Play. *How Google Play Works - 2019 Google Play Public Policy Report*. 2019. URL: <https://kstatic.googleusercontent.com/files/de5640816a4d4099f246b64864c038fee1eac9a9e944b3f31e993e9a315d9f49aa813f27b92be0fe1070f52975476b8fa15529cc2ec314bebcde73f91331f77e> (besucht am 27.05.2020).
- [PM10] David L Poole und Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.
- [Pro20] Extreme Programming. *Test First*. 2020. URL: <http://www.extremeprogramming.org/rules/testfirst.html> (besucht am 16.10.2020).
- [Rad16a] Renas Rada. *Questions Answers*. 2016. URL: <https://github.com/RobotiumTech/robotium/wiki/Questions-&-Answers> (besucht am 09.08.2020).
- [Rad16b] Renas Rada. *Robotium*. 2016. URL: <https://github.com/RobotiumTech/robotium> (besucht am 09.08.2020).
- [Rus12] Evelyn M. Rusli. *Facebook Buys Instagram for \$1 Billion*. The New York Times. Apr. 2012. URL: <https://dealbook.nytimes.com/2012/04/09/facebook-buys-instagram-for-1-billion/> (besucht am 10.06.2020).
- [Sad+13] Norman Sadeh u. a. *The usable privacy policy project*. In: *Technical report, Technical Report, CMU-ISR-13-119*. Carnegie Mellon University, 2013.

- [Sta20a] StatCounter. *Mobile Operating System Market Share Worldwide*. 2020. URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide> (besucht am 14. 10. 2020).
- [Sta20b] Statista. *Mobile Browser - Marktanteile in Deutschland bis Juli 2020*. 2020. URL: <https://de.statista.com/statistik/daten/studie/184297/umfrage/marktanteile-mobiler-browser-bei-der-internetnutzung-in-deutschland-seit-2009/#:~:text=Der%5C%20Firefox%5C%2DBrowser%5C%20landete%5C%20mit,Browser%5C%20mit%5C%204%5C%2C3%5C%20Prozent>. (besucht am 13. 10. 2020).
- [Sta20c] Statista. *Mobile operating systems' market share worldwide from January 2012 to July 2020*. 2020. URL: <https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/> (besucht am 29. 09. 2020).
- [Sto20a] Google Play Store. *Facebook Lite*. 2020. URL: <https://play.google.com/store/apps/details?id=com.facebook.lite&hl=en&gl=US> (besucht am 11. 10. 2020).
- [Sto20b] Google Play Store. *Google Chrome: Fast & Secure*. 2020. URL: <https://play.google.com/store/apps/details?id=com.android.chrome&hl=en&gl=US> (besucht am 11. 10. 2020).
- [Sto20c] Google Play Store. *Google Play Games*. 2020. URL: <https://play.google.com/store/apps/details?id=com.google.android.play.games&hl=en&gl=US> (besucht am 11. 10. 2020).
- [Sto20d] Google Play Store. *Microsoft SwiftKey-Tastatur*. 2020. URL: <https://play.google.com/store/apps/details?id=com.touchtype.swiftkey&> (besucht am 08. 10. 2020).
- [Sto20e] Google Play Store. *Nova Launcher*. 2020. URL: <https://play.google.com/store/apps/details?id=com.teslacoilsw.launcher&referrer> (besucht am 12. 10. 2020).
- [Sto20f] Google Play Store. *UC Browser- Free & Fast Video Downloader, News App*. 2020. URL: <https://play.google.com/store/apps/details?id=com.UCMobile.intl&hl=en&gl=US> (besucht am 13. 10. 2020).
- [Tho16] Jessica Thornsby. *Android UI Design*. Packt Publishing Ltd, 2016.
- [Wir20] Bundesministerium für Wirtschaft und Energie. *Europäische Datenschutz-Grundverordnung*. 2020. URL: <https://www.bmwi.de/Redaktion/DE/Artikel/Digitale-Welt/europaeische-datenschutzgrundverordnung.html> (besucht am 21. 03. 2020).
- [Zim+19] Sebastian Zimmeck u. a. *MAPS: Scaling privacy compliance analysis to a million apps*. In: *Proceedings on Privacy Enhancing Technologies* 2019.3 (2019), S. 66–86.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Köln, den 17. Oktober 2020

Jannis Dammann