Studienarbeit

Entwurf einer Aktualisierungsmöglichkeit für Single-Purpose-Apps via Bluetooth

Von

Julian Hock

Am Horst 4

63857 Waldaschaff

Matrikelnummer: 2215209

Studiengang: Elektro- und Informationstechnik

Fachbereich: Schaltungs- und Messtechnik

Hochschule Aschaffenburg

University of Applied Sciences

Betreuer: Florian Beck

Prüfer: Prof. Dr.-Ing. Ulrich Bochtler

hochschule aschaffenburg

university of applied sciences

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die nachfolgende Studienarbeit eigenständig verfasst und keine anderen als die hierfür zulässigen Hilfsmittel verwendet habe. Stellen in der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Quellen entnommen wurden, wurden unter Angabe dieser kenntlich gemacht.

Ort, Datum Unterschrift

Sperrklausel

Der Inhalt der vorliegenden Arbeit mit dem Titel:

„Entwurf einer Aktualisierungsmöglichkeit für Single-Purpose-Apps via Bluetooth“

von

Julian Hock

darf Dritten ohne Genehmigung der beteiligten Partner (TEC-Institut für technische Innovationen GmbH & Co. KG bzw. Hochschule Aschaffenburg) **nicht** zugänglich gemacht werden. Der Sperrvermerk gilt für die Dauer von 10 Jahren.

Ort, Datum Unterschrift

Gliederung

1. Einleitung

Seit dem Jahr 2014 beschäftigt sich das TEC-Institut für technische Innovationen GmbH & Co. KG in Waldaschaff mit der Entwicklung eines mobilen Paketaufbewahrungssystems. Dieses System soll es den Bestellern von Versandprodukten ermöglichen, die Sendungen auch zu empfangen wenn sie zum Zustellzeitpunkt nicht zu Hause sind. Der sogenannte PakSafe® Basic (Abb. 1) besitzt einen schnittfesten Gurt, welcher zwischen oberer Kante der Tür und Türrahmen befestigt wird. Der Paketbote legt dann sein Paket in den geöffneten PakSafe® ein und verschließt ihn. Der Empfänger kann dieses, wenn er nach Hause kommt, mittels eines Schlüssels wieder herausnehmen.

Abbildung 1: PakSafe® Basic

Nun in Zeiten der Technifizierung hat sich das TEC-Institut dazu entschlossen intelligentere Versionen ihres Systems zu entwickeln. Hierbei entstand unter anderem die Idee für den PakSafe® Master. Dieser nutzt als Intelligenz ein eingebautes Smartphone mit Android Betriebssystem als Benutzerschnittstelle für Empfänger und Paketbote. Für dieses Smartphone wurde bereits im Rahmen einer Masterarbeit eine App entwickelt, welche die grundlegenden Funktionen wie das Ansteuern des Schlosses und das Verbinden eines Bluetooth Handscanners der DHL beinhaltet. Dieses Projekt beschäftigt sich damit die bestehende App in den sogenannten Single-Purpose-Mode (ugs. Kiosk-Modus) zu versetzen. Dieser Modus sorgt dafür, dass die App nicht verlassen werden kann, um auf andere Funktionen des Smartphones zuzugreifen und macht die Anwendung zu einer Art Betriebssystem des PakSafe®, das PakSafeOS. Hierzu wird eine einfache Beispiel-Anwendung programmiert, welche den Single-Purpose-Mode präsentieren soll. Da die Anwendung jedoch nicht über den Google Play Store aktualisiert werden kann und auch nicht in diesen hochgeladen wird, besteht die Hauptaufgabe dieser Studienarbeit darin, eine weitere neue Android App zu entwickeln, mit der es möglich ist das PakSafeOS via Bluetooth zu aktualisieren.

2. Entwicklungsumgebung

Die beiden beschriebenen Anwendungen sollen für Smartphones mit Android-Betriebssystem realisiert werden. Dies wurde früher in der Entwicklungsumgebung Eclipse programmiert, Google hat jedoch im Mai 2013 mit Android Studio seine eigene Entwicklungsumgebung veröffentlicht. In dieser wird in der objektorientierten Programmiersprache Java und dem von Google bereitgestellten Android SDK (Software Development Kit) gearbeitet. Die grafischen Oberflächen der Anwendung und die Ressource- sowie Manifestdateien werden in XML umgesetzt. [1]

Android Studio verwendet „Gradle“ als Grundlage des Build-Systems. Dieses System übernimmt Aufgaben, wie das Erstellen einer installierbaren APK (Dateiformat für installierbare Android Apps), durch Zusammenfügen der verschiedenen Klassen- und Layoutdateien. [2]

Des Weiteren verfügt Android Studio über einen Debugger, welcher es ermöglicht die einzelnen Zeilen im Quellcode in Echtzeit zu überwachen und so Fehler im Programm genau zu lokalisieren oder den sauberen Ablauf der verschiedenen Vorgänge zu gewährleisten.

Um grundlegende Funktionen von Apps auch dann testen zu können, wenn gerade kein Smartphone zur Hand ist, bietet Googles Entwicklungsumgebung einen Emulator an. Dieser AVD (Android Virtual Device) kann verschiedene Smartphones oder auch Tablets, Smartwatches und Smart-TVs direkt auf dem PC-Monitor simulieren.

Ebenfalls in Android Studio enthalten ist die ADB (Android Debug Bridge), ein Kommandozeilen-Tool, über das man mit einem angeschlossenen Android Gerät oder Emulator kommunizieren kann. Die ADB ermöglicht z.B. das Installieren oder Deinstallieren von Apps mittels Kommandozeilen-Befehl. Außerdem bietet sie Zugriff auf eine Unix-Shell, mit der man zahlreiche Befehle wie das Beenden der aktuell auf dem Gerät laufenden App, oder das Aktivieren oder Deaktivieren verschiedener Gerätefunktionen nutzen kann. [3]

3. Recherche

Um die genannten Punkte umzusetzen, entstehen verschiedene Anforderungen an die Smartphones und das darauf laufende Android Betriebssystem. Hierbei ist der wichtigste Aspekt für das PakSafe-Phone die Möglichkeit, es im Single-Purpose-Mode zu betreiben. Für das Smartphone, das die Aktualisierung übernimmt, ergibt sich die Anforderung die Bluetooth-Funktionalitäten für das klassische Bluetooth und nicht das aktuellere Bluetooth Low Energy programmieren zu können.

Seit Android 5.0 Lollipop hat Google die Funktion Screen Pinning eingeführt, welche es dem Entwickler ermöglicht seine App auf dem Home-Screen zu fixieren. Jedoch lässt sich das Screen Pinning durch das Drücken einer bestimmten Tastenkombination wieder einfach beenden. Dies würde dazu führen, dass der Nutzer das PakSafeOS verlassen und auf sämtliche Funktionen des Smartphones zugreifen kann und wäre nicht im Sinne der Entwickler. Mit Android 6.0 Marshmallow wurde die Möglichkeit hinzugefügt das Android Gerät zu einem sogenannten corporate-owend, single-use Gerät oder kurz COSU Device zu machen. Dieser COSU Device lässt zu, Apps als Geräteadministrator (eng.: Device-Owner) festzulegen. Ist eine App als Device-Owner eingerichtet, kann sie anstelle von Screen Pinning den sogenannten Lock Task Mode verwenden. Befindet sich eine App im Lock Task Mode, kann sie nur über die ADB, das Zurücksetzen des Smartphones über das Recovery Menü oder eine im Code programmierte Funktion, verlassen werden. Abbildung 2 zeigt kurz die Unterschiede zwischen Screen Pinning und Lock Task Mode. [4]

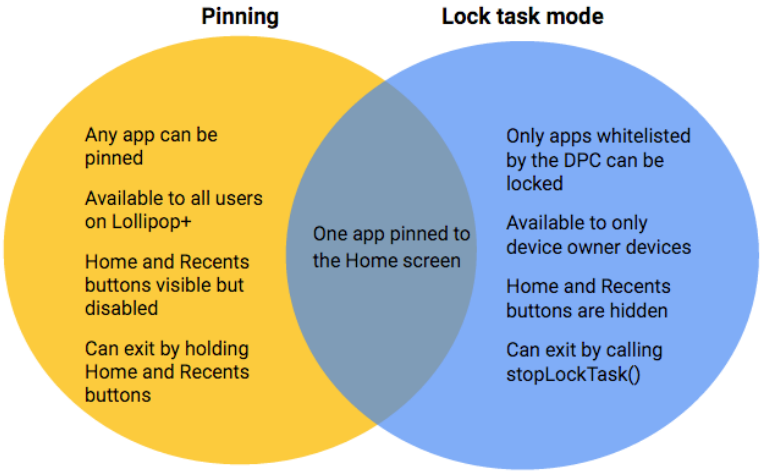

Abbildung 2: Screen Pinning vs. Lock Task Mode

4. Hardwareauswahl

Um den angegebenen Anforderungen gerecht zu werden wurden bei der Auswahl der Hardware verschiedene Smartphones gegenübergestellt. Hierbei war es wichtig für den Single-Purpose-Demonstrator ein Gerät mit physikalischem Button statt einem Touch Button im Display zu verwenden. Damit wird sichergestellt, dass auch physikalische Buttons von ihrer Funktion die App zu verlassen entbunden werden. Die Abbildungen 3 und 4 zeigen die beiden unterschiedlichen Buttonarten.

Abbildung 3: physikalischer Home Button

Abbildung 4: Home Button als Touch Taste

Unter Berücksichtigung dieses und der im letzten Kapitel erwähnten Aspekte wurde das Samsung Galaxy A3 2017 als Simulator für das später im PakSafe® verwendete Smartphone gewählt. Für das Gerät das die Updates überträgt wurde das Huawei P8 Lite gewählt, welches sich noch für das klassische Bluetooth einrichten lässt, während das Samsung sich nur für Bluetooth Low Energy programmieren lässt. Auf beiden Smartphones läuft das benötigte Android 6.0 Marshmallow. In der Realität wird im PakSafe® Master aus Kostengründen kein Smartphone eines namhaften Herstellers wie Samsung sondern einer kleineren aus China stammenden Marke verwendet werden. Diese können für die Massenproduktion in hohen Stückzahlen günstig erworben werden. Da auf solchen Smartphones häufig stark abgewandelte Versionen von Android installiert sind, wurden für dieses Projekt Smartphones eines bekannteren Herstellers verwendet, was die Erst-Entwicklung von Apps deutlich vereinfacht. [5] [6]

5. Android Programmierung

5.1 Allgemeines zur Programmierung

Android verwendet zusätzlich zu seinen Versionsnummern auch Namen und API Level für seine unterschiedlichen Betriebssystemversionen. Da mit jedem Update neue Funktionen eingeführt werden ist es wichtig, ein Minimum-API-Level zu definieren. Dieses legt fest, bis zu welcher minimalen Version die entwickelte Anwendung lauffähig ist. Unterhalb dieser Grenze kann es zu Fehlern oder Problemen kommen.

Die Architektur von Android baut auf einem Embedded Linux-Kernel auf. Er ist unter anderem für die Speicher- und Prozessverwaltung zuständig und stellt die Schnittstelle der Netzwerkkommunikation dar. Außerdem ist er für das Power Management zuständig und stellt die Gerätetreiber für das System. [7]

Die Android Runtime (ART) ist eine Java-Laufzeitumgebung, die seit Android 5.0 Lollipop verwendet wird und die in früheren Versionen verwendete Dalvik Virtual Machine (DVM) abgelöst hat. Dalvik setzte auf Just-in-time-Kompilierung, was dazu führte, dass der Bytecode bei jedem Aufruf der App umgewandelt werden musste. Dies verursachte eine Verzögerung der Ausführung. ART hingegen wandelt den DEX-Bytecode einmalig und bereits während der Installation der Anwendung um, wodurch sich die App schneller starten lässt. Außerdem macht sich das Wegfallen der Just-in-time-Kompilierung durch eine geringere benötigte Prozessorleistung und einen geringeren Energieverbrauch bemerkbar. [8]

5.2 Komponenten bei der Programmierung

Einen der wichtigsten Bestandteile bei der Android Programmierung stellen die Activities dar. Eine Activity besteht in der Regel aus jeweils einer .java-Klassendatei die den Quellcode mit den Anweisungen enthält und einer .xml-Layoutdatei die die grafische Oberfläche der Activity festlegt. Werden mehrere Activities nacheinander gestartet, werden diese auf einem Stapel (Stack) übereinander gelegt. Wird dann eine Activity verlassen oder beendet, wird die oberste Activity vom Stack wieder entfernt und die darunterliegende angezeigt. Für Activities gelten vorgegebene Lebenszyklen, welche in Abbildung 5 dargestellt werden.

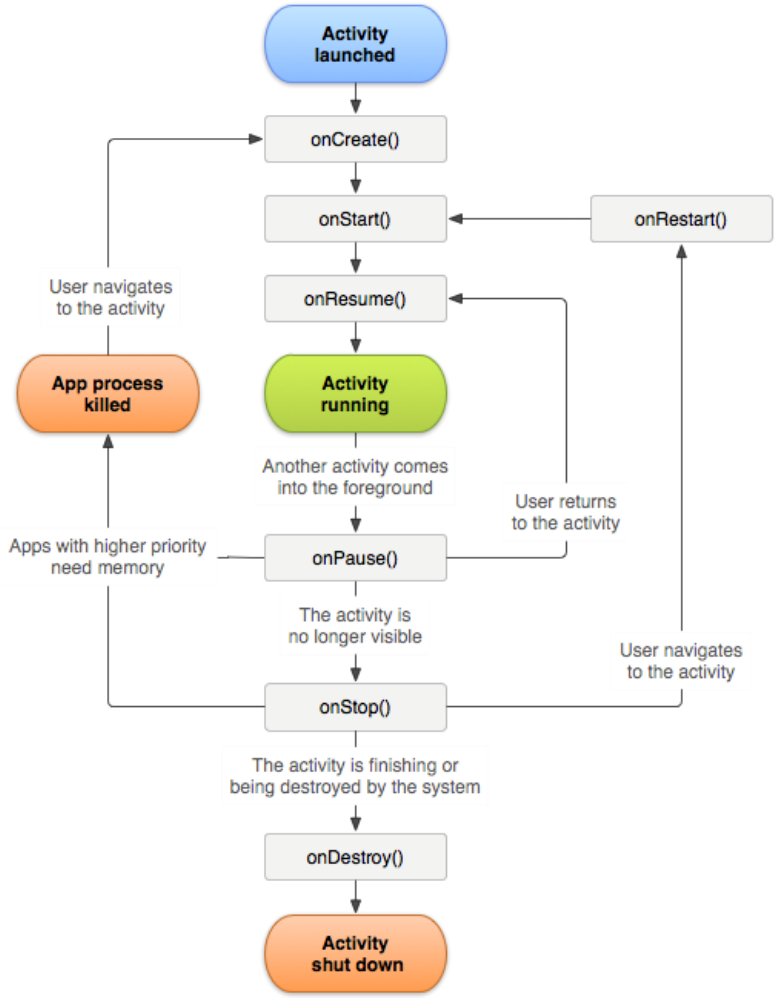

Abbildung 5: Lebenszyklus einer Activity

Beim Start einer Activity werden zuerst die Funktionen onCreate(), onStart() und onResume() ausgeführt. Wird eine neue Activity in den Vordergrund geholt, während die vorhergehende noch sichtbar ist, wird die onPause() Funktion ausgeführt. Sobald man wieder zur ursprünglichen Activity zurückkehrt, wird wieder die Funktion onResume() ausgeführt. Wenn eine Activity nicht mehr sichtbar ist, wird die onStop() Funktion ausgeführt, sobald sie wieder sichtbar ist wird onRestart() ausgeführt und bei onStart() der Zyklus wieder gestartet. Wird eine Activity endgültig beendet, der Prozess also zerstört, wird die onDestroy() Funktion aufgerufen. Um den Zyklus wieder zu starten, muss die Activity erneut gestartet werden, um bei der onCreate() wieder einzusteigen. [9]

Um Dienste wie eine Bluetooth-Verbindung oder Kommunikation mit einer anderen Anwendung zu verwalten, verwendet man in der Regel keine Activity, da die grafische Benutzeroberfläche während der Bearbeitung der entsprechenden Code-Zeilen unterbrochen wäre.

Hierfür benutzt man sogenannte Threads, welche ebenfalls in .java Klassendateien geschrieben werden, jedoch keine zugehörige Layoutdatei haben. Threads können mehrfach parallel zu den Activities laufen und die benötigten Dienste ausführen, ohne dass die grafische Oberfläche in dieser Zeit bedienungsunfähig ist. [10]

Die Datei AndroidManifest.xml ist in jeder Android App vorzufinden. Sie ist die „Schaltzentrale“ der App. In ihr werden z.B. benötigte Genehmigungen, wie die Verwaltung von Bluetooth in der App, oder das Lesen und Schreiben in den externen Speicher festgelegt. Außerdem beinhaltet sie den einzigartigen Packagename, mit dem sich die Anwendung identifizieren lässt. [11]

Ein weiterer wichtiger Bestandteil sind die Broadcast Receiver. Diese werden von der App verwendet um mit dem Betriebssystem zu kommunizieren. Wird beispielsweise in einer Anwendung eine Suche nach umliegenden Bluetooth-Geräten gestartet, passiert die eigentliche Suche nicht in der Anwendung, sondern im Android Betriebssystem. Die gefundenen Geräte können dann mit einem Broadcast Receiver aus dem Betriebssystem „abgefangen“ werden um sie in der App weiterzuverwenden. [12]

In der build.gradle Datei werden wichtige Einstellungen für den Erstellungsprozess der App verwaltet. In ihr befinden sich Angaben für welches Betriebssystem (targetSdkVersion) die Anwendung entwickelt werden soll und bis zu welcher minimalen Android Version (minSdkVersion) diese kompatibel sein soll.

5.3 Lock Task Mode

Für die erste Teilaufgabe dieses Projekts soll eine App entwickelt werden, die vom Endnutzer nicht durch Verwenden der Tasten oder durch Neustart des Smartphones verlassen werden kann. Die Funktionen der in Android-Geräten üblichen Tasten „Aktuelles“, „Home“ und „Zurück“ müssen also gesperrt bzw. eingeschränkt werden. Außerdem wird die App beim Herunterfahren des Systems automatisch in den Hintergrund versetzt, sie muss also beim Neustart direkt wieder in den Vordergrund geholt werden. Beim Wecken des Geräts aus dem Standby-Zustand soll nicht der Sperrbildschirm gezeigt werden, sondern ebenfalls direkt die Anwendung. Für die eben genannten Punkte benötigt die App Geräteadministratorrechte. Bei der Vergabe dieser Rechte ist es wichtig, dass nicht bereits ein anderer Device-Owner gesetzt wurde, da dies immer nur für eine Anwendung möglich ist. Bestenfalls befindet sich das Gerät noch im Werkszustand, da bereits die Einrichtung eines Google Kontos dazu führen kann, dass sich die App nicht mehr als Device-Owner festlegen lässt. Zunächst wird eine sogenannte DeviceAdminReceiver-Klasse benötigt, welche die „Entgegennahme“ der Device-Owner Rechte übernimmt. Dieser Receiver muss zusätzlich in der AndroidManifest.xml angelegt bzw. angemeldet werden.

Der Eintrag in der Manifest-Datei benötigt zusätzlich eine Referenz auf eine XML-Ressource mit einem Device-Admin Tag. Mit diesen 3 Komponenten ist es nun möglich über die ADB eine App als Device-Owner festzulegen. In diesem Falle lautet der Befehl wie folgt:

```
adb shell dpm set-device-owner my.cosuapp/.DeviceAdminReceiver
```

Nach Eingabe dieses Befehls in die Kommandozeile des Android Studio Terminals ist die angegebene Anwendung im System als Geräteadministrator registriert.

Würde man nun versuchen das Gerät in den Lock Task Mode zu versetzen, würde die App jedoch weiterhin nur in den Screen Pinning Modus wechseln. Um in den Lock Task Mode zu gelangen, müssen zusätzlich noch die entsprechenden LockTaskPackages gesetzt werden. Dies passiert durch Ausführung einer Funktion eines DevicePolicyManager-Objekts, mit Übergabe des ComponentName der DeviceAdminReceiver-Klasse und des PackageName der Anwendung.

Von diesem Zeitpunkt an lässt sich die App durch die Funktion „startLockTask()“ oder durch eine Anweisung in der Manifest-Datei in den Lock Task Mode versetzen. Dieser kann dann nur noch über eine Funktion im Code, die Android Debug Bridge oder durch Neustart des Geräts wieder verlassen werden. Da das Smartphone bei einem Neustart jedoch direkt die App anstelle des Startbildschirms anzeigen soll, müssen hier zwei weitere Anweisungen im Manifest festgelegt werden. Diese erlauben die App als Standard-„Launcher“ festzulegen. Somit wird sie beim Start des Geräts direkt geöffnet.

Aktuell wird nach einem kurzzeitigen Ausschalten des Bildschirms durch die „Power“-Taste oder bei einem Neustart noch der Sperrbildschirm angezeigt. Um das zu verhindern, werden in der onCreate()-Funktion der Main Activity drei sogenannte „Flags“ gesetzt. Nun wird der Sperrbildschirm vollständig umgangen. Beim Wecken des Geräts aus dem Standby-Modus oder bei einem Neustart wird unmittelbar die Single-Purpose-App angezeigt.

Versucht man nun die Anwendung zu verlassen wird das verhindert, jedoch erhält man noch eine sogenannte „Toast-Meldung“ (Abb. 6).

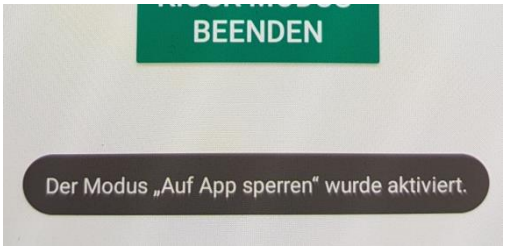

Abbildung 6: Toast-Meldung

Dies könnte den Nutzer verwirren und sieht optisch nicht professionell aus. Durch folgenden ADB-Befehl lässt sich die Anzeige, solcher vom System verursachten Toast-Meldungen, deaktivieren:

```
adb shell appops set android TOAST_WINDOW deny
```

Nachdem die App sich nun im Lock Task Mode befindet, wird im Folgenden auf das Bluetooth-Setup für die Aktualisierung dieser eingegangen.

5.4 Bluetooth-Service

Die Übertragung und Installation der APK soll über Bluetooth realisiert werden. Hierbei ist es wichtig, dass für das Senden keine Pairing- und Verbindungsanfrage am Single-Purpose-Smartphone gestellt wird. Diese Anfragen kommen vom System, würden im Lock Task Mode nur im Hintergrund angezeigt werden und wären somit nicht bestätigbar. Die fehlende Bestätigung hätte zur Folge, dass die Übertragung fehlschlägt. Um dies zu verhindern, wird in der App ein eigener Bluetooth-Service eingerichtet, anstatt die vorhandene API zum Versenden von Dateien via Bluetooth vom Android System zu verwenden. In diesem Bluetooth-Service wird anstelle einer „Secure Rfcomm“-Verbindung eine „Insecure Rfcomm“-Verbindung genutzt. Der hier programmierte Bluetooth-Service basiert auf einer für die Zwecke des Projekts abgewandelten Version des Bluetooth Chat Example von Android. [13] Dadurch wird beim Aufbau der Verbindung keine Verbindungsanfrage gestellt.

Um die besagten Funktionen nutzen zu können, benötigt die Anwendung folgende Genehmigungen in der Manifest-Datei, um auf Bluetooth und den externen Speicher zugreifen zu dürfen:

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

Zusätzlich muss die Berechtigung für den Speicherzugriff der Anwendung in den Einstellungen des Betriebssystems freigegeben werden.

Das Prinzip des Bluetooth-Service wird in Abbildung 7 mit einem Flussdiagramm wiedergegeben. Das Flussdiagramm setzt voraus, dass Bluetooth auf beiden Geräten bereits eingeschaltet ist. Ist dies der Fall muss zunächst auf dem Single-Purpose-Smartphone die Bluetooth Sichtbarkeit und der Listening-Mode aktiviert werden. Mit der Aktivierung des Listening-Mode wird der Accept Thread der Bluetooth-Service Klasse gestartet und somit ein neuer Bluetooth Socket mit vorgegebener UUID (Universally Unique Identifier) geöffnet. Diese UUID besteht aus einer 16-Byte-Hexadezimalzahl und wird vorher im Programmcode festgelegt.

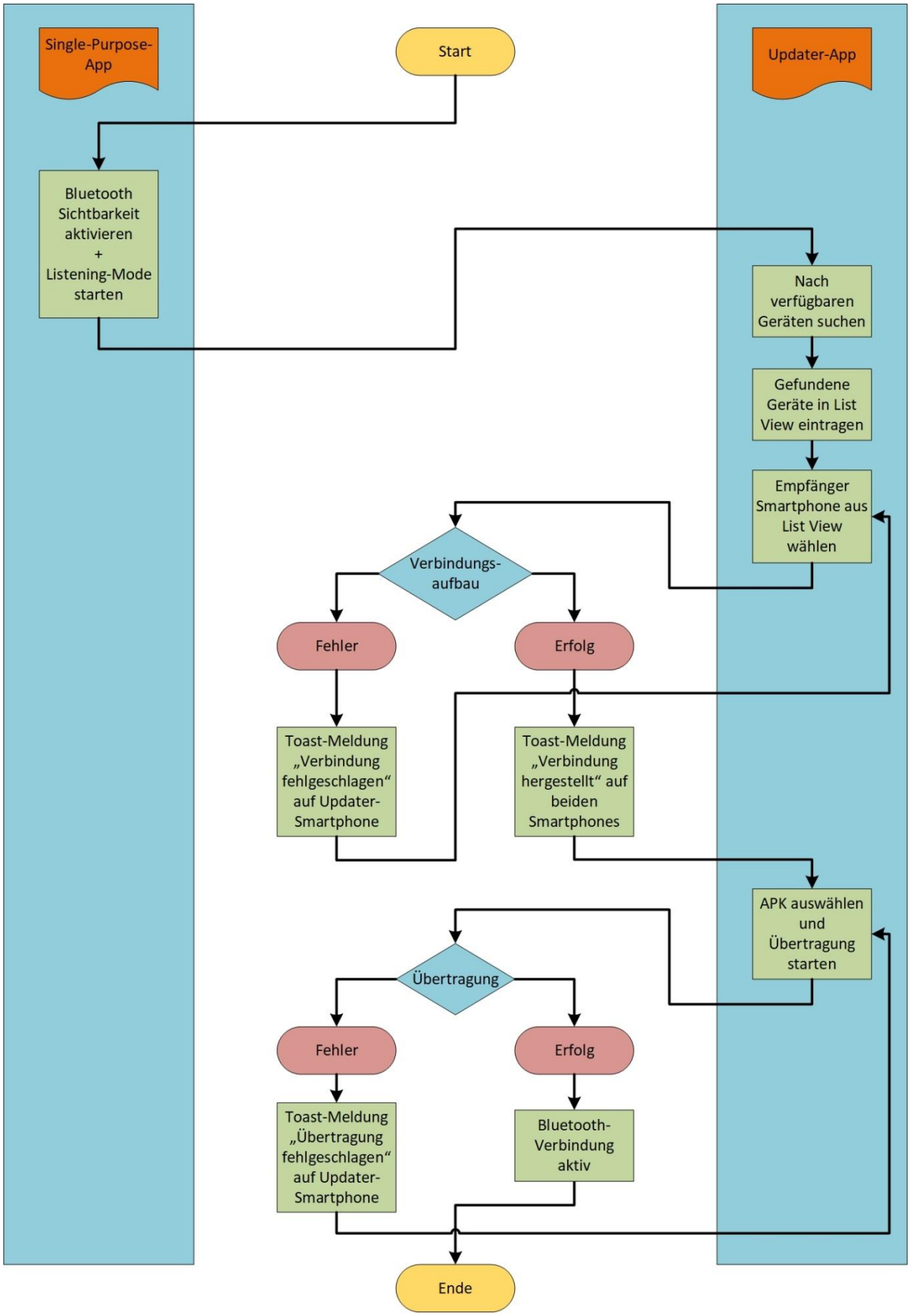

Abbildung 7: Flussdiagramm Bluetooth-Service

Der Verbindungsaufbau kann nur dann erfolgen, wenn beide Geräte die gleiche UUID verwenden. Im nächsten Schritt wird mit dem Updater-Smartphone nach verfügbaren Geräten gesucht. Taucht das gesuchte Smartphone in der Liste der gefundenen Geräte auf, kann durch Klicken auf den Gerätenamen ein Verbindungsaufbau gestartet werden. Dies geschieht im Connect Thread, indem für den gewählten Bluetooth Device ein neuer Insecure Rfcomm Socket geöffnet und die festgelegte UUID an ihn übergeben wird. Ist der Verbindungsversuch gescheitert, erfolgt die Anzeige einer Fehlermeldung auf dem Updater-Smartphone. Wurde die Verbindung erfolgreich aufgebaut, erhält man auf beiden Geräten ebenfalls eine optische Bestätigung. Ab diesem Punkt haben die beiden Geräte eine dauerhafte aktive Verbindung, über die Daten übertragen werden können. Die aktive Verbindung wird im Connected Thread verwaltet und aufrecht gehalten. Es ist nun möglich am Updater-Smartphone die APK-Datei für die Aktualisierung zu wählen. Daten werden bei Bluetooth in Byte Arrays übertragen. Hierfür müssen Funktionen programmiert werden, die die Datei auf dem Sender-Smartphone in ein Byte Array umwandelt und sie so für das Versenden bereit macht. Auf dem Empfänger-Smartphone wird das Byte Array wieder in eine Datei geschrieben und unter einem im Programmcode festgelegten Pfad abgespeichert.

5.5 Installation des Updates

Nachdem nun die APK, die die Aktualisierung beinhaltet, in den Speicher geschrieben wurde, ist sie bereit zur Installation. Da die Datei immer unter dem gleichen Pfad und Dateinamen gespeichert wird, ist für das Anwählen derselben kein Explorer nötig. Es wird ein einfacher Button programmiert, mit dem der festgelegte Pfad als Intent geöffnet wird. Installationen von Apps können jedoch generell nur vom System und nicht von einer eigenen App ausgeführt werden. Wie bereits erwähnt werden aber andere Anwendungen während des Lock Task Mode blockiert, also ließe sich der Installer nicht öffnen. An dieser Stelle muss vor Öffnen des Intents der Lock Task Mode mit dem Befehl „stopLockTask()“ beendet und nach Installation und Wiederaufruf der App wieder gestartet werden.

6. Benutzeroberfläche

Für die grafische Benutzeroberfläche beider Anwendungen wurden von Android Studio bereitgestellte Layouts und Designs verwendet. In Abbildung 8 und 9 kann man die beiden Menüs der Single-Purpose-App sehen. Sie verfügt über einen Hauptbildschirm und ein Updatemenü, welches mit dem „Update“-Button betreten werden kann. Auf dem Hauptbildschirm lässt sich Bluetooth ein- und ausschalten. Außerdem wurde, um die Entwicklung der Anwendung zu vereinfachen, ein Button programmiert mit dem sich der Lock Task Mode beenden lässt. Im Updatemenü tauchen die im letzten Kapitel besprochenen Buttons auf, mit denen die Bluetooth-Verbindung hergestellt und das Update installiert werden kann. Um den Erfolg des Updates zu visualisieren wurde ein einfacher Versionstext implementiert der mit der Aktualisierung umgeschrieben wird.

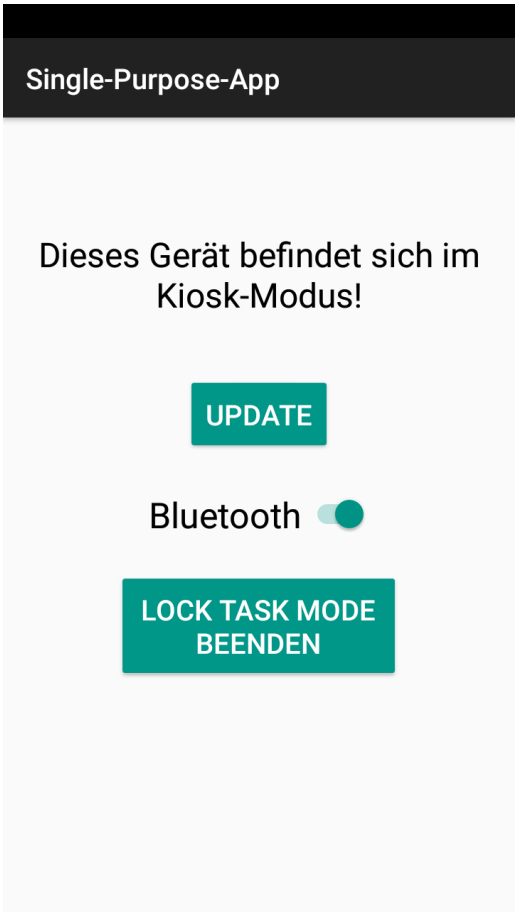

Abbildung 8: Single-Purpose-App Hauptmenü

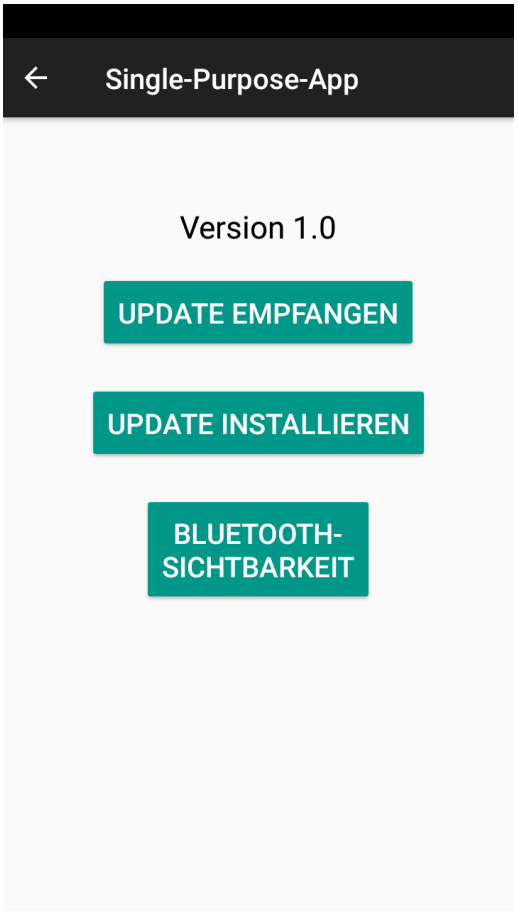

Abbildung 9: Single-Purpose-App Updatemenü

In Abbildung 10 findet sich die grafische Oberfläche der Updater-App. Hier lässt sich Bluetooth ein- und ausschalten und eine Suche nach neuen Geräten starten. Die gefundenen Geräte und ihre zugehörige MAC-Adresse werden in einer Liste im unteren Teil des Bildschirms eingetragen, so wie hier das „PakSafe Phone“. Die beiden Buttons „Verbindung beenden“ und „APK senden“ werden erst bei einer aktiven Verbindung sichtbar.

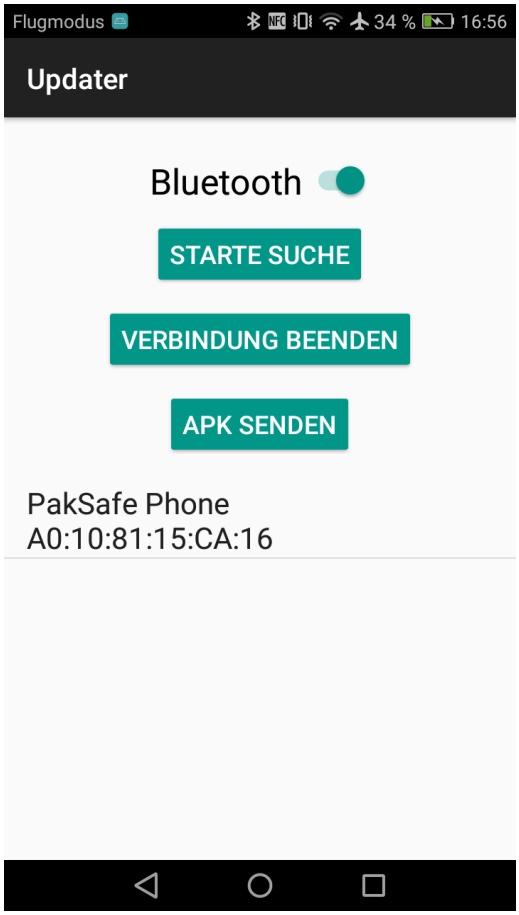

Abbildung 10: Updater-App Hauptmenü

7. Fazit

Abschließend lässt sich sagen, dass alle für die Studienarbeit vorgesehenen Anforderungen erfüllt werden und sich mit den beiden Anwendungen die Aktualisierung des PakSafeOS erfolgreich demonstrieren lässt.

Die Schritte für den Geräteadministrator und den Lock Task Mode der Single-Purpose-App lassen sich sehr leicht in das bestehende Projekt des PakSafeOS implementieren. Die Updater-App lässt sich funktionell direkt verwenden, das Prinzip kann bei entsprechender Programmierung aber auch auf andere Bluetooth-Geräte wie z. B. Notebooks übertragen werden.

Beim Installieren der Aktualisierung treten in der Single-Purpose-App noch zwei kleinere Probleme auf. Zum einen kommt es gelegentlich zu einem Absturz wenn im aufgerufenen Installer anstelle von „Installieren“ die Option „Abbrechen“ gewählt wird. Die Ursache dieses Fehlers liegt daran, dass die Anwendung direkt versucht, wieder den Lock Task Mode zu betreten. Jedoch passiert dies teilweise bevor sich die Anwendung wieder im Vordergrund befindet, was dann den Absturz verursacht. Das zweite Problem liegt darin, dass der Lock Task Mode während der Installation verlassen werden muss. Wird nach der Installation „Öffnen“ gedrückt, startet die App wieder in den Lock Task Mode, wird jedoch „Fertig“ gedrückt, gelangt man auf den Hauptbildschirm des Geräts. Dies würde im späteren Gebrauch zu einer massiven Sicherheitslücke führen, da der Benutzer Zugriff auf das gesamte Betriebssystem des Smartphone erhält.

8. Ausblick für die Zukunft

Zum aktuellen Zeitpunkt benötigt man für die Freigabe der Device-Owner Rechte noch Android Studio und die ADB, außerdem muss die APK für die Anwendung manuell installiert werden. Für eine Massenproduktion des PakSafe® Master ist dieses Verfahren jedoch ungeeignet, da es viel Zeit in Anspruch nimmt und somit erhöhte Kosten verursachen würde. Um dem entgegenzuwirken wurden im Rahmen dieser Studienarbeit einige Möglichkeiten untersucht, um dieses Verfahren zu beschleunigen. Im Zuge dessen entstand die Idee einen einfachen NFC-Tag mit Informationen zu beschreiben der SSID und Passwort eines W-LAN Routers, Serverpfad zur APK des PakSafeOS und die benötigten Daten zum Freigeben des Geräteadministrators enthält. Der NFC-Tag kann dann an das im Werkszustand befindliche Smartphone gehalten werden und dieses installiert auf eine kurze Bestätigung hin die gewünschte Anwendung. Zudem werden die benötigten Rechte festgelegt und die App direkt im Lock Task Mode gestartet. Dies würde die Einrichtung der PakSafe® Smartphones deutlich beschleunigen und vereinfachen und somit auch die Kosten hierfür verringern.

Quellenverzeichnis

		Aufgerufen am:
[1]:	https://de.wikipedia.org/wiki/Android_Studio	18.09.2017
[2]:	https://developer.android.com/studio/intro/index.html#build-system	18.09.2017
[3]:	https://developer.android.com/studio/command-line/adb.html	19.09.2017
[4]:	https://developer.android.com/work/cosu.html	20.09.2017
[5]:	https://www.inside-handy.de/handys/huawei-p8-lite	20.09.2017
[6]:	https://www.inside-handy.de/handys/samsung-galaxy-a3-2017	20.09.2017
[7]:	https://source.android.com/devices/architecture/kernel/	28.09.2017
[8]:	https://source.android.com/devices/tech/dalvik/	28.09.2017
[9]:	https://developer.android.com/reference/android/app/Activity.html	28.09.2017
[10]:	https://developer.android.com/reference/java/lang/Thread.html	29.09.2017
[11]:	https://developer.android.com/guide/topics/manifest/manifest-intro.html	29.09.2017
[12]:	https://de.wikibooks.org/wiki/Googles_Android/_BroadcastReceiver	29.09.2017
[13]:	https://developer.android.com/samples/BluetoothChat/index.html	05.10.2017

Abbildungsverzeichnis

Inhalt der CD

Die zur Studienarbeit gehörige CD beinhaltet:

- diese Dokumentation im .pdf-Format
- den kommentierten Quellcode der Single-Purpose-App im Android Studio Projektformat
- den kommentierten Quellcode der Updater-App im Android Studio Projektformat
- die installierbare .apk-Datei der Single-Purpose-App
- die installierbare .apk-Datei der Updater-App