Diplomarbeit

Evaluation und Entwicklung einer anonymen und zensurresistenten Lösung für mobilen Datenaustausch

Martin Czernia

19. August 2011

Fakultät Informatik
Hochschule Mannheim
Paul-Wittsack-Straße 10
68163 Mannheim

Betreuer:
M.Sc. Christian Baun
Prof. Dr. Georg Winterstein

Eidesstattliche Erklärung

Hiermit erkläre ich, Martin Czernia, dass ich die vorliegende Diplomarbeit selbstständig erstellt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt.

Mannheim, den 19. August 2011

Martin Czernia

Danksagung

Ich möchte mich an dieser Stelle bei M.Sc. Christian Baun und Prof. Dr. Georg Winterstein für die Betreuung und Unterstützung dieser Diplomarbeit bedanken.

Ein großer Dank geht auch an Luise Breitenstein und Aki Kiatipis für das Korrekturlesen der Arbeit.

Vielen Dank auch an Claudia Hellhorst, die mir bei der Evaluierung der Arbeit behilflich war.

Zusammenfassung

Ausgehend von der Idee des anonymen Datenaustausches über USB-Sticks wird in der vorliegenden Arbeit eine mobile Lösung vorgestellt. Durch die Installation der ersten sogenannten *Dead Drops* durch Aram Bartholl können mit Hilfe von USB-Sticks im öffentlichen Raum Daten anonym und zensurresistent ausgetauscht werden. Mobiltelefone sind mit diesen USB-Sticks jedoch nicht kompatibel.

Die in dieser Arbeit entwickelte Anwendung *BlueDeadDrop* ermöglicht den Austausch von Daten mittels Mobiltelefonen. Die Anwendung wurde für die quelloffene Plattform Android entwickelt und verwendet Bluetooth als Übertragungstechnik.

Die Evaluation der Anwendung hat gezeigt, dass der anonyme Datenaustausch über Bluetooth gelingt. Einschränkungen ergeben sich jedoch aufgrund der begrenzten Sichtbarkeit der Geräte sowie der nötigen Paarung.

Abstract

On the basis of the idea of anonymous data exchange via USB flash drives, this paper presents a mobile solution. Aram Bartholl was first intalling so called *Dead Drops* in public spaces for anonymous and censorship resistent data exchange with the help of USB flash drives. Though mobile phones are not compatible with USB flash drives.

In this thesis an application is developed that allows data exchange using mobile phones. The application is developed for the open source platform Android and uses bluetooth as transmission technology.

The evaluation of the application shows that anonymous data exchange using bluetooth is possible. Although there are a few limitations due to limited visibility and the necessary pairing.

Inhaltsverzeichnis

Abbildungsverzeichnis **VIII**

1 Einleitung **1**

2 Dead Drops **3**
2.1 Ursprung 3
2.2 Moderne Dead Drops 3

3 Bluetooth **5**
3.1 Bluetooth Special Interest Group 5
3.2 Entwicklung 6
3.3 Bluetooth Protokoll Stack 7
3.4 Netzwerk Topologie 9
3.5 Pairing 9

4 Android **11**
4.1 Open Handset Alliance 11
4.2 Software Development Kit 12
4.3 System Architektur 13
4.4 Lebenszyklus einer Anwendung 17

5 Konzept **19**
5.1 Namensgebung 19
5.2 Plattform 19
5.3 Programmiersprache 20
5.4 Entwicklungsumgebung 20
5.5 Anforderungsanalyse 20

6 Implementierung **22**

6.1 Architektur 22
6.2 Datenbank 23
6.3 Grafische Benutzeroberfläche 24
6.4 Dienst 26

7 Evaluation **34**
7.1 Testgeräte 34
7.2 Installation 35
7.3 Erster Start der Anwendung 36
7.4 Erster Kontakt mit anderen Geräten 37
7.5 Datenübertragung 39
7.6 Ergebnis 39

8 Alternativen **41**
8.1 Übertragungstechnik 41
8.2 Plattform 42

9 Fazit **44**

Literaturverzeichnis **IX**

Abbildungsverzeichnis

2.1 Dead Drop in einer Mauer [deab] 4

3.1 Bluetooth-Logo 6
3.2 Bluetooth Stack [Lüd07] 8
3.3 Netzstrukturen [Lüd07] 10

4.1 Android SDK and AVD Manager 13
4.2 System Architektur [andc] 14
4.3 Activity Lifecycle [andb] 18

5.1 BlueDeadDrop Logo 19
5.2 Use Case-Diagramm 21

6.1 MVC-Modell 22
6.2 Datenbank Model 23
6.3 Preference-Screen mit OI File Manager 26
6.4 Aktivitäts Diagramm: AcceptThread 28
6.5 Aktivitäts Diagramm: ConnectThread 29
6.6 Aktivitätsdiagramm: ConnectedThread 30

7.1 Einstellung 'Unbekannte Herkunft' unter Android 2.2 36
7.2 Dialog: Fehlende Anwendung 36
7.3 Gerät sichtbar machen 37
7.4 Pairing bis Bluetooth-Version 2.0 38
7.5 Pairing ab Bluetooth 2.1 38
7.6 Grafische Oberfläche von BlueDeadDrop 39

8.1 Weltweite Verkaufszahlen von Smartphones 43

Kapitel 1

Einleitung

Im Oktober 2010 hat der Künstler Aram Bartholl in New York damit begonnen, sogenannte *Dead Drops*[1] in Mauern zu platzieren. Dabei handelt es sich um eingemauerte USB-Sticks die für jeden frei zugänglich sind. Einige Menschen weltweit haben sich dieser Bewegung angeschlossen. Mittlerweile gibt es 565 *Dead Drops* mit einer Gesamtkapazität von ungefähr 1666 GB[2].

Diese *Dead Drops* dienen zum anonymen, zensurfreien Datenaustausch. Bei Verbindung eines Laptops mit dem USB-Stick können z.B. Dateien übertragen werden, die andere Benutzer kopieren können. Es gibt keine Kontrolle, wer den *Dead Drop* benutzt oder wer Daten auf oder vom *Dead Drop* kopiert. Jeder kann diesen nach belieben für sich nutzen.

Bei der Nutzung von USB-Sticks als *Dead Drops* existieren auch einige Nachteile. Die Standorte der *Dead Drops* sind allgemein bekannt. Diese sind auf der Internetseite [deab] veröffentlicht. Somit könnten diese überwacht werden und die Anonymität ginge verloren. Ein weiteres Problem, das mit dem Standort zusammenhängt, ist die Möglichkeit des Diebstahls oder der mutwilligen Zerstörung. Niemand ist verantwortlich für die Bewachung der USB-Sticks und jeder hat freien Zugang.

Darüber hinaus unterstützt nicht jedes mobile Gerät USB-Speicher. Es gibt zurzeit keine Möglichkeit USB-Sticks mit Mobiltelefonen zu verbinden. Um die *Dead Drops* nutzen zu können, muss der Benutzer ein Netbook oder Laptop dabei haben.

Ziel dieser Diplomarbeit ist es, eine mobile Lösung für die genannten Probleme zu finden. Es soll eine Anwendung für Mobiltelefone entwickelt werden. Der Anwender soll dadurch

[1] deutsch: tote Briefkästen
[2] Stand: 31.07.2011

die Möglichkeit haben, ein Verzeichnis auszuwählen, das er mit anderen teilen will. Wird die Anwendung auf dem Mobiltelefon gestartet, soll das Mobiltelefon automatisch nach anderen Mobiltelefonen suchen und eine Verbindung aufbauen. Darauf hin sollen die freigegebenen Dateien ausgetauscht werden.

Das Projekt für die Idee der Arbeit wird in Kapitel 2 beschrieben. Ein Einblick in die Übertragungstechnik Bluetooth findet in Kapitel 3 statt. In Kapitel 4 ist die Plattform Android, die aus einem Betriebsystem, einer Entwicklungsumgebung und den Anwendungen besteht, beschrieben.

Auf das Konzept der Anwendung wird in Kapitel 5 eingegangen. Dabei wird dargestellt, wie der Name *BlueDeadDrop* zustande kam und welche Plattform, Programmiersprache und Entwicklungsumgebung zum Einsatz kommt. Zudem werden die Anforderungen an das Programm erläutert.

Im Anschluss daran beschäftigt sich Kapitel 6 mit der Implementierung. Darin wird die Architektur der Anwendung sowie die einzelnen Komponenten beschrieben. In Kapitel 7 findet eine Prüfung der Anwendung auf Praxistauglichkeit statt.

Das letzte Kapitel enthält das Fazit dieser Diplomarbeit.

Kapitel 2

Dead Drops

2.1 Ursprung

Ein toter Briefkasten (engl.: Dead Drop) ist nach der Definition auf [wik] ein Versteck, das der Übermittlung von geheimen Nachrichten dient. Dieses Versteck ist meist ein öffentlicher Ort, an dem kein Briefkasten erwartet wird. Er darf in der Öffentlichkeit nicht auffallen. Mithilfe solcher Briefkästen können Nachrichten oder Gegenstände anonym ausgetauscht werden. Weder dem Empfänger noch dem Absender muss die Identität des anderen bekannt sein. Die Übergabe funktioniert mit Signalen an einem nur für die beiden bekannten Ort. Hat der Absender das Paket hinterlegt, gibt er dem Empfänger ein vorher vereinbartes Zeichen in der Öffentlichkeit. Dieses Zeichen darf in der Öffentlichkeit nicht als Zeichen wahrgenommen werden. Der Empfänger kann das Paket abholen und dem Absender ein Zeichen geben, dass er es erhalten hat.

2.2 Moderne Dead Drops

Aram Bartholl hat das Projekt *Dead Drops* im Oktober 2010 ins leben gerufen. Er installierte in New York die ersten fünf *Dead Drops*. *Dead Drops* werden von Aram Bartholl wie folgt auf seiner Homepage [deab] definiert:

> 'Dead Drops' is an anonymous, offline, peer to peer file-sharing network in public space.

Aram Bartholl hatte die Idee, *Dead Drops* in Form von USB-Sticks in der Öffentlichkeit zu platzieren. Jeder kann diese *Dead Drops* nutzen um Dateien auszutauschen. Mittlerweile gibt es 565 *Dead Drops* mit einer Gesamtkapazität von ca. 1666 GB (Stand:

31.07.2011). Jeder kann sich diesem Netzwerk anschließen. Voraussetzung ist lediglich ein Gerät, das mit USB-Sticks kompatibel ist und ein *Dead Drop* in der Nähe. Auf der Homepage ist eine Weltkarte [deaa] mit allen aktuell platzierten *Dead Drops* zu finden. Um die Dateiübertragung zu starten, verbindet man z.B. ein Notebook mit dem USB-Stick. Dabei ist ein kleines USB-Verlängerungskabel hilfreich, um den Stick und/oder das Notebook nicht zu beschädigen. Ist das Notebook mit dem USB-Stick verbunden, kann der Datenaustausch beginnen. Abbildung 2.1 zeigt einen *Dead Drop* in einer Mauer.

Abbildung 2.1: Dead Drop in einer Mauer [deab]

Die wichtigsten Vorteile dieser Art der Datenübertragung sind Anonymität und Zensurfreiheit.

Kapitel 3

Bluetooth

Bluetooth ist ein Funksystem das von der Bluetooth Special Interest Group (BSIG) entwickelt wurde um kurze Kabelverbindungen zwischen verschiedenen Geräten zu ersetzen. Bluetooth[1] ist nach dem dänischen Wikingerkönig Harald Blåtand benannt, der u.a. für seine Kommunikationsfähigkeit bekannt war. Blåtand heißt übersetzt Bluetooth bzw. Blauzahn. Bluetooth wurde von der schwedischen Firma Ericsson ins leben gerufen.

3.1 Bluetooth Special Interest Group

Die BSIG wurde im Jahre 1998 von den Firmen Ericsson, IBM, Intel, Nokia und Toshiba gegründet. Bereits im Gründungsjahr hat die BSIG mehr als 400 Mitglieder gezählt. Inzwischen sind es mehr als 14.000 Mitglieder aus verschiedenen Bereichen. Die BSIG ist eine Interessengemeinschaft, die weder Bluetooth-Geräte herstellt noch verkauft. [bsi] Die Aufgaben der BSIG sind:

- Bluetooth-Spezifikationen veröffentlichen,
- das Qualifizierungsprogramm verwalten,
- die Markenrechte schützen und
- die Bluetooth-Technologie zu verbreiten.

[1] deutsch: Blauzahn

3.2 Entwicklung

Im Jahr 1999 wurde die Bluetooth Spezifikation 1.0 von der BSIG veröffentlicht. In der Zwischenzeit ist Bluetooth ein weltweiter Standard. Mittlerweile haben zahlreiche mobile Telefone diese Übertragungstechnik integriert. Es gibt jedoch noch weitere Geräte, die über Bluetooth kommunizieren können. Einige davon sind:

- Eingabegeräte (z.B. Computermaus)
- Headsets
- Drucker
- Fernbedienungen
- Gamecontroller

In Abbildung 3.1 ist das Bluetooth-Logo zu sehen. Viele Bluetooth-Geräte tragen das blau-weiße Symbol als Leistungsmerkmal.

Abbildung 3.1: Bluetooth-Logo

Einige Spezifikationen mit wichtigen Neuerungen sind hier aufgeführt:

Bluetooth 1.0/1.1 Die erste Version hatte noch mit Kinderkrankheiten zu kämpfen. Die Hersteller hatten bei ihren Produkten Probleme mit der Interoperabilität. Mit der im Jahr 2001 veröffentlichten Version 1.1 wurden Fehler beseitigt. Die maximale Datenrate betrug damals 732,2 kbit/s.

Bluetooth 1.2 Diese Spezifikation wurde 2003 veröffentlicht. Geräte in Reichweite werden schneller gefunden und können nach der Empfangsqualität sortiert werden. Der Verbindungsaufbau wurde beschleunigt. Das adaptive Frequenz-Hopping wurde eingeführt, dieses passt sich an die aktuelle Störsituation an und verbessert damit die Verbindungsqualität.

Bluetooth 2.0 + EDR 2004 wurde die Bluetooth-Version 2.0 veröffentlicht. Neu hinzugekommen ist die Technik Enhanced Data Rate (EDR). Diese Technik steigert die maximale Datenrate auf 2,1 Mbit/s.

Bluetooth 2.1 + EDR Secure Simple Pairing wurde mit dieser Version eingeführt und das Pairing-Verfahren damit vereinfacht. Mehr dazu in Abschnitt 3.5.

Bluetooth 3.0 + HS Die Datenrate wurde mit dieser Version auf 24 Mbit/s erhöht. Dabei wird die Bluetooth-Verbindung dazu verwendet, um eine Ultra-Wide Band (UWB) Verbindung herzustellen. Diese nutzt eine größere Bandbreite im ISM-Band und kann lokale WLAN-Netzwerke stören.

Bluetooth 4.0 Die Spezifikation wurde um ein Low Energy Protokoll erweitert. Sie ist für eingebettete Systeme gedacht.

3.3 Bluetooth Protokoll Stack

Der Bluetooth-Protokoll-Stack besteht aus dem Bluetooth Core, den Bluetooth-Protokollen und den Anwendungsprofilen. Abbildung 3.2 zeigt eine vereinfachte Darstellung des Protokoll-Stacks.

3.3.1 Bluetooth Core

Der Bluetooth Core arbeitet direkt mit der Hardware zusammen. Im Physical Layer sind die Modulationsverfahren, Sender- und Empfängerkenngrößen implementiert. Das Baseband ist für das Bilden der Funkkanäle zuständig. Weiterhin übernimmt dieser die Fehlerüberwachung und Flusskontrolle. Der Link Controller sorgt für den Aufbau, Erhalt und Abbau der Verbindungen und verwaltet diese.

3.3.2 Protokolle

Auf dem Bluetooth Core sitzen die verschiedenen Protokolle. Für die Sprachübertragung wird das Telephony Control Protocol (TCS) benutzt. Um eine IP-Verbindung aufzubauen, wird das Bluetooth Network Encapsulation Protocol (BNEP) verwendet. Es existieren noch weitere Protokolle. Die für das Projekt wichtigen Protokolle SDP und RFCOMM werden im Folgenden näher erläutert.

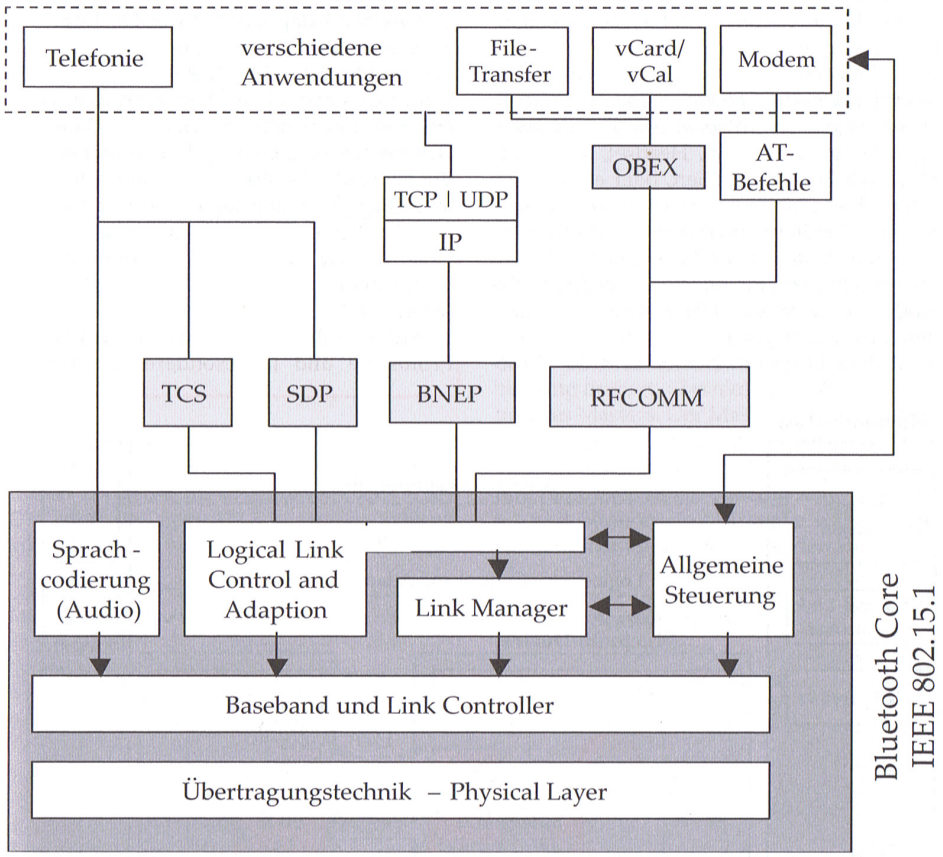


Abbildung 3.2: Bluetooth Stack [Lüd07]

Service Discovery Protocol

Ein Bluetooth-Gerät kann mehrere Dienste anbieten. Dazu gehören z.B. Dial-Up Networking, Dateitransfer, Austausch von Kontakten oder Sprachübertragung. Ein Gerät kann über das Service Discovery Protocol anfragen, welche Dienste ein zweites Gerät anbietet. Bevor eine Verbindung aufgebaut wird, wird über das SDP angefragt, ob der Dienst verfügbar ist und wie man ihn nutzen kann. Aus Anwendersicht sind diese Dienste als Profile bekannt. Dazu mehr in Abschnitt 3.3.3. Jeder Dienst hat eine eindeutige Universally Unique Identifier (UUID), anhand der er identifiziert wird.

Radio Frequency Communications

Das Protokoll Radio Frequency Communications (RFCOMM) ist ein Kabelersatz-Protokoll. Es wird dazu verwendet, um RS232-Schnittstellen zu emulieren. Jeder Dienst, der diese Schnittstelle nutzt, bekommt eine Kanalnummer zugeteilt. Ein zweites Gerät, das einen

Dienst auf der RFCOMM-Schnittstelle nutzen will, bekommt dessen Kanalnummer über das SDP. RFCOMM ist für Entwickler eine einfache Möglichkeit, eine Verbindung zwischen zwei Bluetooth-Geräten herzustellen.

3.3.3 Profile

Die Anwendungsprofile bilden die oberste Schicht. Ob ein Bluetooth-Gerät mit einem zweiten kompatibel ist, hängt davon ab, welche Profile implementiert sind. Ein einfaches Beispiel ist das Drucken über Bluetooth. Ein Drucker mit Bluetooth hat üblicherweise auch das Basic Printing Profile (BPP) implementiert, über das es möglich ist, Dokumente ohne Druckertreiber zu drucken. Mit einem PC oder Smartphone, die das BPP ebenfalls implementiert haben, ist es möglich, über Bluetooth zu drucken.

3.4 Netzwerk Topologie

Bluetooth-Geräte die miteinander kommunizieren, organisieren sich in sogenannten Piconetzen. Ein Piconetz besteht aus maximal acht Teilnehmern. Dabei ist einer der Teilnehmer der 'Master' und die restlichen sieben sind die 'Slaves'. Der komplette Datenverkehr wird vom Master geregelt. Er hat die Aufgabe, das Medium 'Luft' auf die Teilnehmer aufzuteilen. Je höher die Anzahl der Teilnehmer ist, desto weniger Bandbreite hat ein Teilnehmer zur Verfügung. Ist ein Teilnehmer im Empfangsbereich zweier Piconetze, kann er diese zu einem Scatternetz zusammenschließen. In Abbildung 3.3 sieht man die möglichen Netzstrukturen. Für dieses Projekt ist eine Verbindung mit nur einem Master und einem Slave sinnvoll, da so Dateien mit voller Bandbreite übertragen werden können.

3.5 Pairing

Bevor zwei Bluetooth-Geräte miteinander kommunizieren können, müssen diese sich kennen. Bis einschließlich Bluetooth Version 2.0 ist der Vorgang des 'Kennenlernens' aufwendig, denn beide Benutzer müssen eine identische PIN eingeben. Damit wird sichergestellt, dass kein drittes Gerät die Verbindung mithören bzw. einen *Man-in-the-middle-Angriff* (MITM-Angriff) ausüben kann. Das Pairing muss einmalig für zwei Geräte ausgeführt werden. Beide Geräte generieren mit der PIN einen *Link-Key* für die Verschlüsselung und

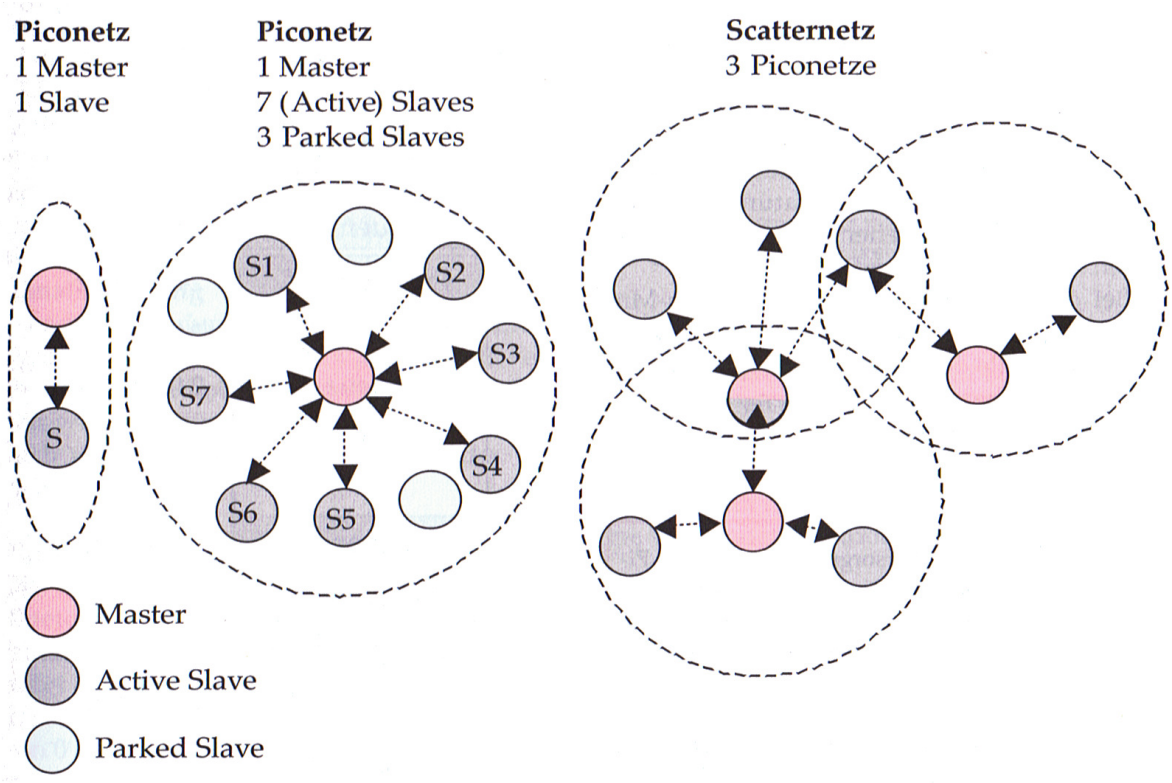


Abbildung 3.3: Netzstrukturen [Lüd07]

Authentifizierung. Dieser Schlüssel kann in Zukunft für beide Geräte verwendet werden und das Pairing entfällt.

Mit Bluetooth Version 2.1 wurde das *Secure Simple Pairing* eingeführt. Diese Version verwendet ein *Public/Private Key* Verfahren in Kombination mit dem Diffie-Hellmann-Algorithmus anstatt einer PIN. Die beiden Geräte tauschen ihre öffentlichen Schlüssel aus. Anschließend generieren die Geräte eine Zufallszahl und verschlüsseln diese mit dem Öffentlichen Schlüssel der Gegenstelle. Diese Zufallszahlen werden benutzt um die *Link-Keys* zu erzeugen. Die Zufallszahlen können nur mit den privaten Schlüsseln entschlüsselt werden. Dieses Pairing-Verfahren ist jedoch nicht gegen MITM-Angriffe geschützt. Aus diesem Grund müssen beide Anwender einen 6-stelligen Code bestätigen, der von jemanden, der sich zwischen beide Geräte schaltet, nicht zu berechnen ist. Für Geräte, die kein Display zum Anzeigen des 6-stelligen Codes haben, entfällt die Bestätigung. Ein MITM-Angriff hat bei dem Verfahren ohne Bestätigung Erfolg. [Sau08]

Kapitel 4

Android

Google Android ist eine freie und quelloffene Betriebsystem-Plattform. Sie besteht aus den folgenden drei Komponenten:

- Freies und quelloffenes Betriebssystem für mobile Geräte.
- Freie und quelloffene Entwicklungsplattform zum Erstellen von mobilen Anwendungen.
- Geräte, auf denen das Android Betriebssystem sowie deren Anwendungen laufen.

Alle Anwendungen haben über eine Programmierschnittstelle (API) Zugriff auf die Hardware, nutzen die gleichen API's und laufen in der gleichen Laufzeitumgebung. Jede native Anwendung vom Hersteller kann gelöscht und durch eigene ersetzt werden.

4.1 Open Handset Alliance

Im Februar 2007 hat Google mit 33 anderen Firmen die Open Handset Alliance (OHA) gegründet. Die OHA besteht mittlerweile[1] aus 84 Firmen aus unterschiedlichen Bereichen. Vertreten sind Netzbetreiber, Software-Firmen, Marketing-Unternehmen, Halbleiterindustrie und Geräte-Hersteller. Die Geräte-Hersteller z.B. stellen sicher, dass Android mit ihrer Hardware kompatibel ist. Dazu sind Anpassungen des Linux-Kernels nötig. [oha]

[1] Stand: 17.08.11

4.2 Software Development Kit

Android stellt ein Software Development Kit (SDK) bereit, das mit dem SDK von Java vergleichbar ist. Es ist für die drei Plattformen MacOS X, Linux und Windows verfügbar. Beim SDK von Java ist normalerweise nur eine Version installiert. Aktuell[2] ist es JDK 1.6u26. Bei Android ist für jede Version ein eigenes SDK verfügbar.

4.2.1 Android SDK and AVD Manager

Der *Android SDK and AVD Manager* ist ein Werkzeug, das die verschiedenen Versionen der SDKs und die virtuellen Geräte verwaltet.

Die für dieses Projekt verwendete SDK hat die Versionsnummer 2.2 bzw. API Version 8. Einige Hersteller bieten für ihre Geräte auch eigene SDKs an. Diese enthalten spezielle APIs und Emulatoren für bestimmte Geräte. In Abbildung 4.1 ist der *Android SDK and AVD Manager* zu sehen. Dieses Werkzeug hält die installierten SDK's auf dem aktuellen Stand und installiert ggf. neue Versionen.

Mit dem *Android SDK and AVD Manager* ist es möglich, virtuelle Geräte zu erstellen, um die entwickelten Anwendungen zu testen. Das virtuelle Gerät kann dabei jede Android-Version annehmen die installiert wurde. Weiterhin kann die Größe der virtuellen SD-Karte und die Größe des Displays beliebig eingestellt werden. Virtuelle Geräte haben jedoch keine Unterstützung für Bluetooth und eignen sich deshalb nicht als Testgeräte für das vorliegende Projekt.

4.2.2 Android Debug Bridge

Ein wichtiges Werkzeug für die Entwicklung von Android-Anwendungen ist die *Android Debug Bridge* (ADB). Diese stellt eine USB-Verbindung zu angeschlossenen Android Geräten her. Einige Funktionen sind:

- Debug-Ausgabe anzeigen
- .apk Dateien auf dem Android Gerät installieren
- Dateien zwischen Computer und Android Gerät kopieren
- eine *remote shell* auf dem Android Gerät starten
- ...

[2]Stand: 28.07.2011

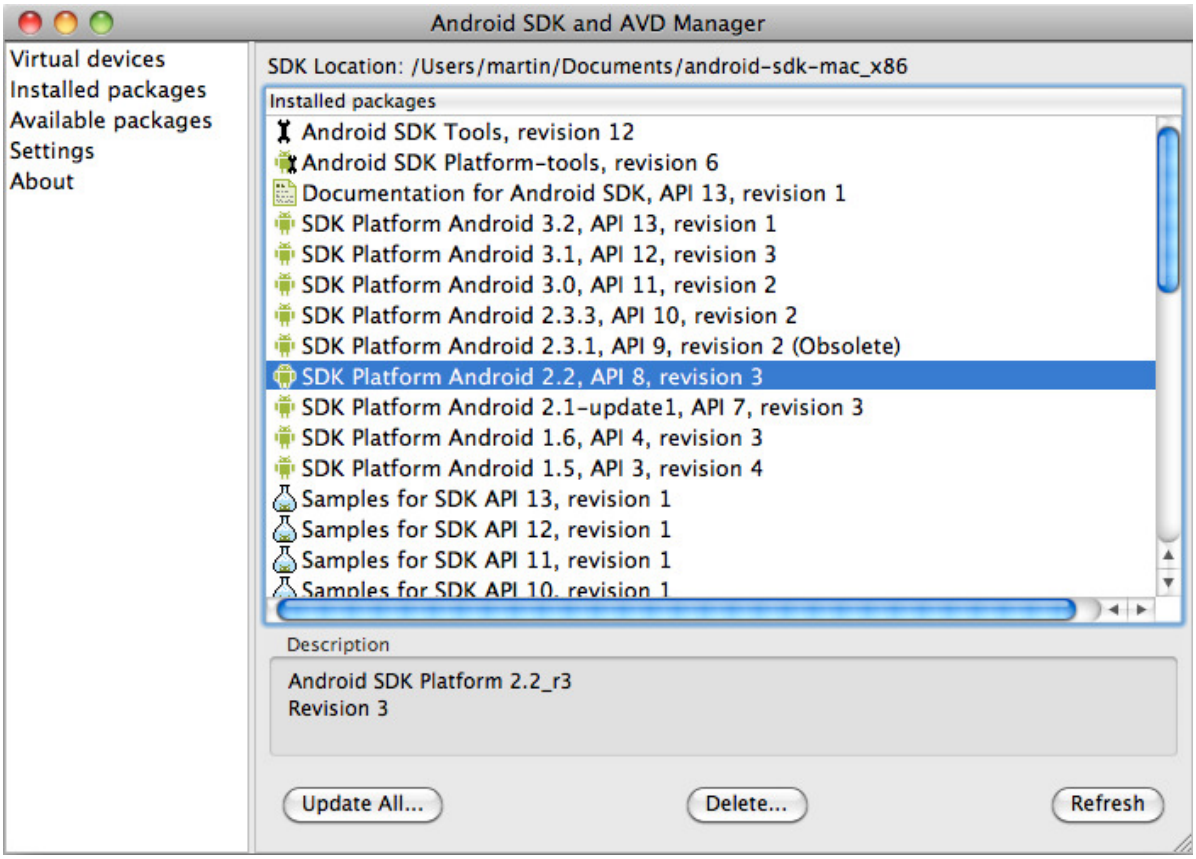


Abbildung 4.1: Android SDK and AVD Manager

4.3 System Architektur

Abbildung 4.2 zeigt die Hauptkomponenten des Android-Betriebsystems. Der angepasste Linux-Kernel befindet sich direkt über der Hardware. Die mittlere Schicht bilden die Programmbibliotheken und die Laufzeitumgebung. Die oberste Schicht besteht aus dem Application Framework und den Programmen selbst. Die einzelnen Schichten der Systemarchitektur von Android werden im Folgenden näher beschrieben.

4.3.1 Linux-Kernel

Die Basis für Android ist ein angepasster Linux-Kernel der Version 2.6. Der Kernel bildet die Schicht zwischen der Hardware und der restlichen Architektur und enthält unter anderem:

- Hardwaretreiber

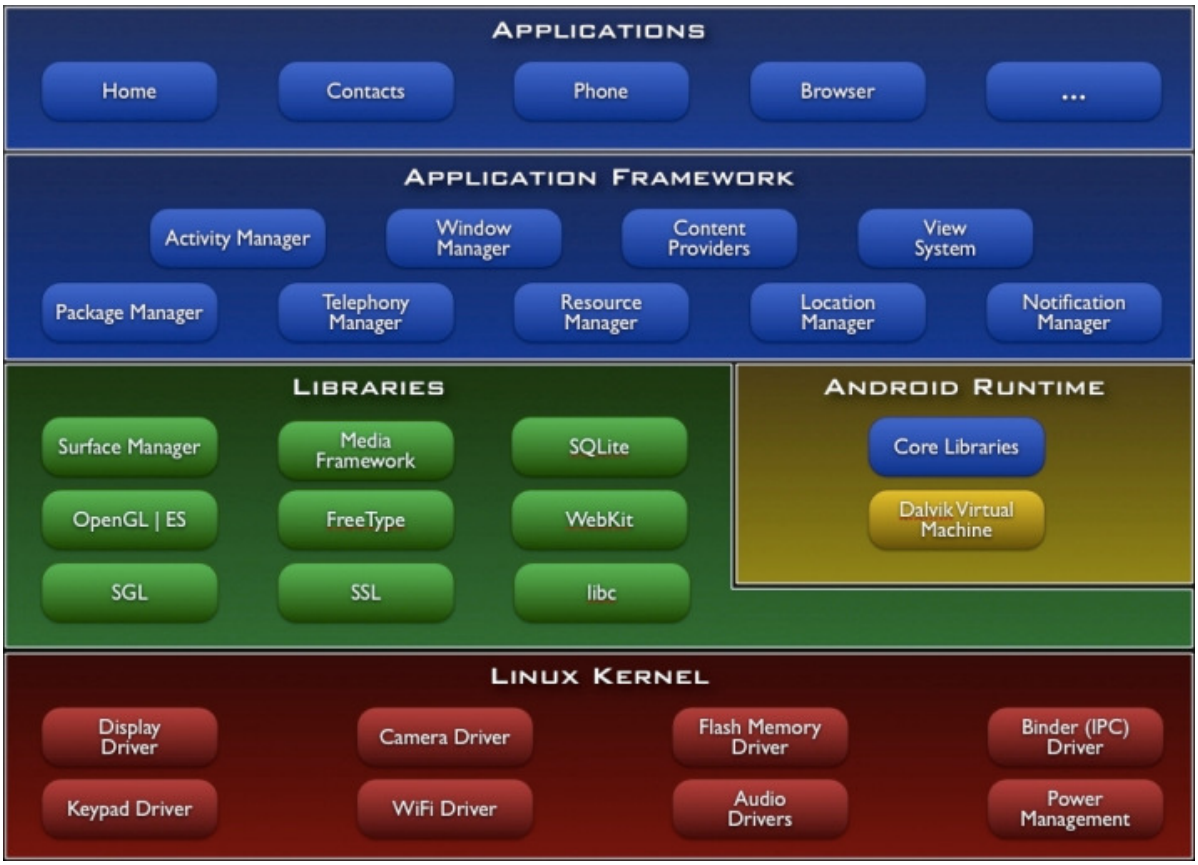


Abbildung 4.2: System Architektur [andc]

- Prozess- und Speicherverwaltung
- Sicherheitsverwaltung

Android läuft auf unterschiedlicher Hardware. Für die unterschiedlichen Hardwarekomponenten entwickeln die jeweiligen Hardwarehersteller die Hardwaretreiber. Es existieren Projekte außerhalb der OHA, die sich damit beschäftigen, Android auf Nicht-Android Geräten zu installieren [non]. Android läuft theoretisch auf jedem Gerät, das der Compatibility Definition [anda] von Android entspricht.

Eine weitere Änderung am Linux-Kernel ist die Prozess und Speicherverwaltung. Preisgünstige Mobiltelefone verfügen über 128 bis 192 MB Arbeitsspeicher. Um jederzeit ausreichend Ressourcen für die laufenden Programme zu haben, hat Android eine aggressive Prozessverwaltung. Die für den Benutzer sichtbare Anwendung hat immer die höchste Priorität. Benötigt diese Anwendung mehr Ressourcen als zur Verfügung stehen, werden andere Programme beendet, um Ressourcen frei zu geben [andd]. Die Tatsache, dass Programme jederzeit beendet werden können, muss man bei der Entwicklung von

Android-Anwendungen berücksichtigen (siehe Abschnitt 4.4).

4.3.2 Libraries

Android enthält eine Sammlung von Programmbibliotheken für C/C++. Diese werden den Entwicklern durch das Android Application Framework zur Verfügung gestellt. Einige davon sind:

System C library wurden von der Berkeley Software Distribution[3] (BSD) Implementierung abgeleitet und für mobile Geräte optimiert. Die optimierte Version ist ca. halb so groß wie das Original. [HKM10]

Media Libraries basieren auf PacketVideo Open Core[4]. Diese sind Zuständig für die Aufnahme und Wiedergabe von Audio- und Video-Formaten.

Surface Manager steuert den Zugriff auf das Display und unterstützt 2D- und 3D-Grafikebenen von mehreren Anwendungen.

LibWebCore ist eine moderne Browser Engine, die auch von Apple Safari und Google Chrome eingesetzt wird.

SGL ist eine 2D-Grafik-Engine.

3D libraries sind eine Implementierung der OpenGL ES[5] 1.0 APIs. Diese benutzen entweder die Hardwarebeschleunigung oder den 3D Software Rasterizer.

FreeType ist für die Schriftartengenerierung verantwortlich.

SQLite ist eine relationale Datenbank die auf dem Gerät verfügbar ist. Es handelt sich um ein quelloffenes Projekt[6], welches nicht speziell für Android entwickelt wurde.

Diese und noch weitere Bibliotheken werden den Entwicklern in Form von APIs zur Verfügung gestellt.

4.3.3 Android Run Time

Die Android Run Time besteht aus den Core Libraries und der Dalvik Virtual Machine (Dalvik VM). Zusammen bilden sie die Basis für das Application Framework.

[3] `http://www.bsd.org`
[4] `http://www.packetvideo.com`
[5] `http://www.khronos.org/opengles/`
[6] `http://www.sqlite.org`

Da die Dalvik VM keine normale Java VM ist, wurden die Core Libraries angepasst. Die Android Libraries implementieren die meisten Funktionalitäten der Java Libraries und darüber hinaus noch spezielle Android Libraries. Die Dalvik VM wurde speziell für Android entwickelt und funktioniert als Mittelschicht zwischen dem Linux-Kernel und den Anwendungen. Über den Linux-Kernel hat die Dalvik VM direkten Zugriff auf das Sicherheits-, Prozess-, und Speichermanagement. Dabei wird die Hardware von der Dalvik VM für den Entwickler abstrahiert. Somit müssen die Entwickler nichts von der genauen Hardwareimplementierung wissen. Sollte es aus Performance-Gründen nötig sein, die Anwendungen direkt in C/C++ zu entwickeln, bietet Android das Native Development Kit (NDK) an. Damit ist es möglich, eigene C/C++ Bibliotheken mit direktem Zugriff auf OpenGL zu erstellen. Die Dalvik VM führt Dalvik executable files (.dex) aus. Diese ausführbaren Dateien sind auf minimalen Speicherverbrauch optimiert. [Mei10]

4.3.4 Application Framework

Das Application Framework von Android stellt dem Entwickler Klassen zur Verfügung, um alle Funktionen der freien und offenen Plattform zu nutzen. Die nativen Anwendungen sowie Anwendungen von Drittherstellern verwenden das gleiche Framework. Das Framework stellt einen abstrahierten Zugriff auf die Hardware bereit. Die Entwickler erhalten dadurch Zugriff auf Bluetooth, GPS, Display, Sensoren und weiteren Hardwarekomponenten. Die wichtigsten Komponenten des Frameworks sind:

Views sind eine Reihe von Bausteinen mit denen man die Oberfläche einer Anwendung erstellen kann. Sie beinhalten z.B. Textfelder, Listen, Knöpfe und einen Browser zum einbetten.

Content Provider geben den Anwendungen die Möglichkeit, Daten zu verwalten, die mit anderen Anwendungen geteilt werden können. Über den Content Provider ist es z.B. möglich auf die Daten des Telefonbuchs zuzugreifen.

Resource Manager verwaltet die externen Ressourcen. Man erhält Zugriff auf Bilder, Übersetzungen, Layouts und weiteren Ressourcen.

Notification Manager ist für Benachrichtigungen in der Statusleiste zuständig. Über diesen kann man eigene Benachrichtigungen erstellen.

Activity Manager ist für den Lebenszyklus von Programmen zuständig. Er übernimmt die Navigation zwischen den Anwendungen.

Das Konzept des Frameworks ist die einfache Wiederverwendbarkeit von Komponenten. Dabei spielt es keine Rolle, ob es die eigenen Komponenten sind oder die eines Dritten. Es besteht die Möglichkeit, Activities, Services und den Speicher einer Anwendung mit anderen Anwendungen zu teilen. Allerdings nur soweit es den Sicherheitseinstellungen entspricht.

4.3.5 Anwendungen

Die oberste und für den Benutzer die einzig sichtbare Schicht sind die Anwendungen selbst. Alle Anwendungen laufen in der Android Run Time und verwenden als Basis das Application Framework. Jede Anwendung kann durch eine andere ersetzt werden.

4.4 Lebenszyklus einer Anwendung

Der Lebenszyklus einer Anwendung in Android unterscheidet sich grundlegend von anderen Plattformen. Anwendungen können normalerweise nur gestartet und beendet werden. Bei Android gibt es weitere Status-Übergänge. Abbildung 4.3 zeigt den Lebenszyklus einer Activity.

Eine Anwendung durchläuft beim Start die Funktionen *onCreate*, *onStart* und *onResume*. Erst dann ist die Anwendung aktiv. Die Funktion *onPause* wird aufgerufen, sobald eine andere Activity in den Vordergrund kommt. Die Activity muss keine Änderungen an der Oberfläche vornehmen, solange sie nicht sichtbar ist. Die Funktion *onPause* gibt der Anwendung die Gelegenheit *Event Handler* zu deaktivieren, um Ressourcen freizugeben.

Sobald die Anwendung wieder in den Vordergrund kommt, führt Android die Methode *onResume* aus. Die Anwendung kann die Oberfläche wieder aktualisieren und die *Event Handler* aktivieren.

Um die Anwendung zu beenden, führt Android die Funktion *onStop* aus. Ist die Anwendung im Status *Pause* oder *Stop* kann diese jederzeit von Android zerstört werden. In diesen Fällen hat die Anwendung keine Zeit mehr, ihren Zustand zu speichern. Der Entwickler hat dafür zu sorgen, dass die Anwendung nach einem Aufruf von *onPause* bzw. *onStop* zerstört werden kann, ohne dass es Konsequenzen für den Benutzer hat (z.B. Datenverlust).

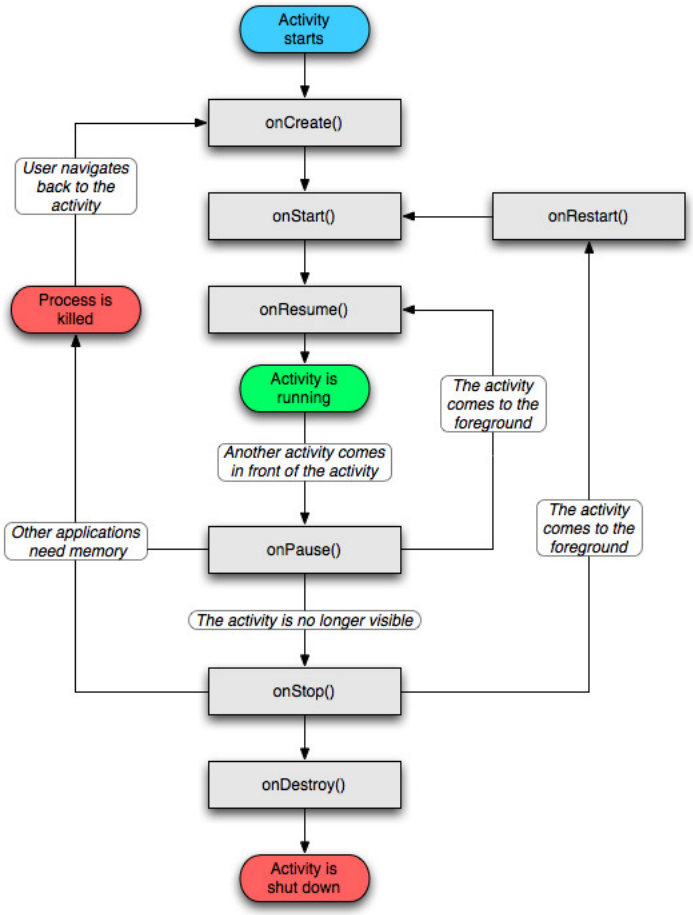


Abbildung 4.3: Activity Lifecycle [andb]

Im Zustand *Stop* kann die Anwendung mit dem Aufruf *onRestart* wieder gestartet werden. Der Aufruf *onDestroy* beendet die Anwendung komplett und der Speicher wird freigegeben.

Kapitel 5

Konzept

5.1 Namensgebung

Das im Rahmen dieser Diplomarbeit entwickelte Projekt beruht auf der Idee der *Dead Drops* von Aram Bartholl. Anders wie bei den *Dead Drops* sollen die Dateien in diesem Projekt nicht über USB-Sticks, sondern über Bluetooth ausgetauscht werden. Die Anwendung hat den Namen durch die Verknüpfung der beiden Wörter *Dead Drops* und *Bluetooth* erhalten: *BlueDeadDrop* ('Blauer Toter Briefkasten'). Das Logo der Anwendung ist in Abbildung 5.1 zu sehen. Dieses Logo stammt von [ico] und steht zur persönlichen Verwendung frei zur Verfügung. Das Logo besteht aus einem Ordner, auf dem ein Bluetooth Zeichen abgebildet ist. Es spiegelt somit den Anwendungszweck optimal wieder.

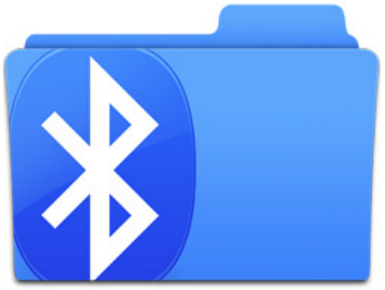


Abbildung 5.1: BlueDeadDrop Logo

5.2 Plattform

Die Wahl der Plattform fällt auf Android. Android ist eine freie und offene Plattform, die dem Entwickler viele Möglichkeiten bietet und eine gut dokumentierte API hat. Die Dateien werden mit Bluetooth übertragen, auch dafür bietet Android eine sehr gut dokumentierte API an. Auf anderen Plattformen ist es hingegen wesentlich schwieriger, die Bluetooth Schnittstelle anzusprechen.

5.3 Programmiersprache

Durch die Wahl von Android als Plattform bietet sich Java als Programmiersprache an. Wie in Kapitel 4 beschrieben, beinhaltet Android eine angepasst Java VM, die Dalvik VM. Es gibt auch die Möglichkeit in C/C++ zu programmieren, doch ist das für das vorliegende Projekt nicht erforderlich. Die API von Android bringt alle nötigen Funktionen, die für dieses Projekt benötigt werden mit.

5.4 Entwicklungsumgebung

Als Entwicklungsumgebung wird *NetBeans 7.0*[1] mit dem Plugin *NBAndroid*[2] verwendet.

NetBeans wurde von Sun Microsystems als quelloffenes Projekt entwickelt. Mittlerweile gehört es, wie Java auch, zu Oracle. Es handelt sich um eine vollständige Entwicklungsumgebung für Java, PHP, C/C++ und weitere Sprachen. Die Funktionalität kann mit Plugins erweitert werden.

NBAndroid ist ein Plugin für NetBeans, womit sich Android-Projekte in NetBeans erstellen lassen. Damit NetBeans die korrekte Auto-Vervollständigung bietet, lässt sich die verwendete Android-Version einstellen. Auf Knopfdruck wird das Projekt kompiliert, gepackt, auf das Gerät übertragen und gestartet. Bei mehreren angeschlossenen Geräten bietet das Plugin einen Dialog, um ein Gerät auszuwählen. Dabei macht es keinen Unterschied, ob es sich um ein echtes oder ein virtuelles Gerät handelt.

5.5 Anforderungsanalyse

Wie bereits in Kapitel 1 beschrieben, gibt es folgende Anforderungen an die Anwendung:

- Die Anwendung soll auf einem mobilen Telefon laufen.
- Das freigegebene Verzeichnis soll frei wählbar sein.
- Das Gerät sucht selbstständig nach anderen Geräten.
- Das Gerät verbindet sich selbstständig mit anderen Geräten.
- Das Gerät tauscht automatisch Dateien aus.

[1] http://www.netbeans.org
[2] http://kenai.com/projects/nbandroid

Daraus ergibt sich das Anwendungsfalldiagramm in Abbildung 5.2. Der Benutzer hat keinen Einfluss auf den Ablauf der Anwendung. Er kann den Dienst starten bzw. stoppen und den Pfad, der freigegeben werden soll, auswählen.

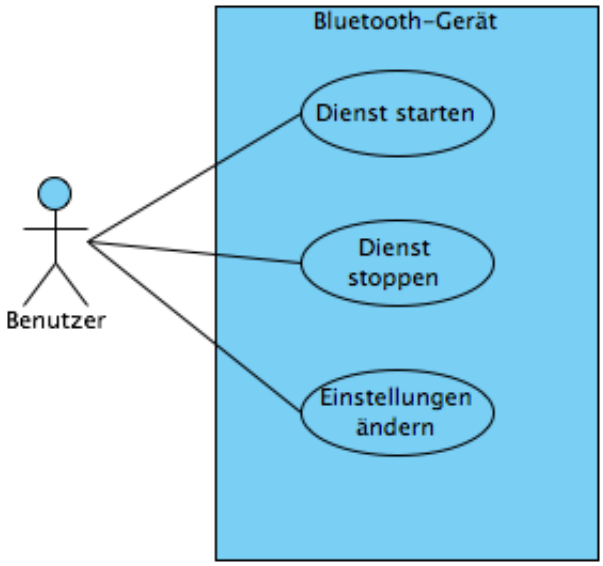


Abbildung 5.2: Use Case-Diagramm

Die Anwendung soll unbemerkt im Hintergrund laufen. Idealerweise läuft die Anwendung den ganzen Tag und tauscht dabei Dateien mit anderen Geräten aus.

Kapitel 6

Implementierung

6.1 Architektur

Die Anwendung ist nach der Architektur Model-View-Controller (MVC) entwickelt. Abbildung 6.1 zeigt die vier Hauptklassen der Anwendung. Dabei ist das Model gelb dargestellt, die View blau und der Controller in rot.

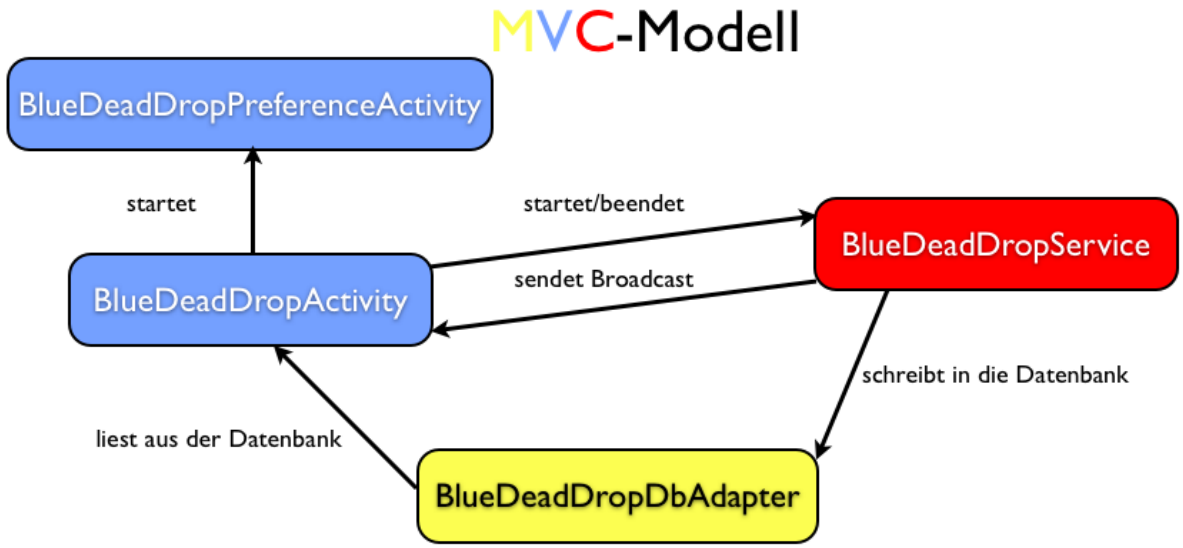


Abbildung 6.1: MVC-Modell

Die Klasse *BlueDeadDropDbAdapter* bildet das Model ab. Die View und der Controller können über Methoden der Klasse auf die Datenbank zugreifen.

Die View besteht aus den zwei Klassen *BlueDeadDropActivity* und *BlueDeadDropPreferenceActivity*. Wobei erstere der Einstiegspunkt für die Anwendung ist. Die *BlueDeadDropActivity* stellt Statistiken dar, die aus der Datenbank gelesen werden. Zudem verfügt

die *Activity* über eine Benutzereingabe, um den *BlueDeadDropService* zu starten bzw. zu beenden. Über das Menü kann der Benutzer die *BlueDeadDropPreferenceActivity* starten und Einstellungen für das Programm vornehmen.

Der Hauptteil der Anwendung sitzt im Controller und wird durch die Klasse *BlueDeadDropService* implementiert. Über mehrere Threads werden Geräte gesucht, gefunden und Dateien ausgetauscht. Die Statistiken werden in die Datenbank geschrieben.

6.2 Datenbank

Die folgenden Abschnitte beschreiben das Model der Datenbank und wie die Anwendung auf diese zugreift kann.

6.2.1 Model

Die Datenbank besteht aus zwei Tabellen. Das Diagramm der Datenbank ist in Abbildung 6.2 zu sehen. Die Tabelle *Device* wird genutzt, um die gefundenen Geräte zu speichern. Die *Statistics* Tabelle speichert statistische Werte, wie z.B. die Anzahl der gesendeten Dateien.

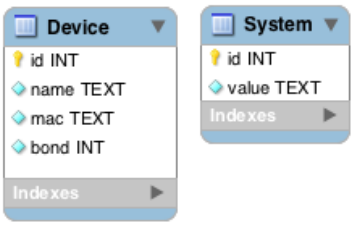


Abbildung 6.2: Datenbank Model

6.2.2 BlueDeadDropDbAdapter

Die Klasse *BlueDeadDropDbAdapter* dient der View und dem Controller als Schnittstelle zur Datenbank. Die Android API stellt die abstrakte Klasse *SQLiteOpenHelper* bereit. Diese ist als statische Member-Variable im *BlueDeadDropAdapter* implementiert.

SQLiteOpenHelper

Es müssen die folgenden beiden Methoden implementiert sein:

- onCreate(SQLiteDatabase db)
- onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

Die Methode *onCreate* wird aufgerufen, sobald die Anwendung das erste mal auf die Datenbank zugreift und diese nicht existiert. Die Methode *onUpgrade* wird ausgeführt, sobald die Datenbank nicht mehr aktuell ist. Neben der Übergabe der Datenbank findet auch die Vermittlung der alten und der neuen Versionsnummer an die Methode *onUpgrade* statt. Somit lässt sich die Datenbank mit passenden Scripts in die aktuelle Version überführen.

Schnittstellen

Die View und der Controller haben Zugriff auf folgende Funktionen:

open() Öffnet eine Datenbankverbindung.

close() Schließt eine Datenbankverbindung.

insertDevice(BluetoothDevice device) Speichert das übergebene Gerät in der Datenbank ab.

removeAllDevices() Löscht alle Geräte aus der Datenbank.

getAllDevicesCursor() Liefert einen Cursor, mit dem man auf die Einträge in der Datenbank zugreifen kann.

getStatisticsValue(int id) Liefert den statistischen Wert für die angegebene ID.

setStatisticsValue(int id, String value) Setzt den statistischen Wert für die angegebene ID.

Diese Funktionen sind für die Anwendung vollkommen ausreichend.

6.3 Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche dient zur Interaktion mit dem Benutzer. Dieser kann über die Oberfläche Statistiken ansehen, den Dienst starten bzw. stoppen und die Einstellungen ändern.

6.3.1 BlueDeadDropActivity

Das Layout der Oberfläche ist in einer XML-Datei definiert. Es besteht aus einer Tabelle für die Statistiken, zwei Buttons um den Dienst zu starten bzw. zu beenden und einer Liste für die Auflistung der gefundenen Geräte.

Die Activity bindet sich bei der Initialisierung an den Dienst. Dadurch erhält die Activity Zugriff auf die *public methods* der Dienst-Klasse. Über diese Methoden liest die Activity den aktuellen Zustand vom Dienst aus und zeigt ihn auf der Oberfläche an. Startet oder beendet der Benutzer den Dienst über die Knöpfe auf der Oberfläche, ruft die Activity die entsprechende Funktion vom Dienst auf.

Die Activity öffnet die Datenbankverbindung ebenfalls bei der Initialisierung. Dies geschieht über den *BlueDeadDropDbAdapter*. Die angezeigten Statistiken und gefundenen Geräte werden aus der Datenbank gelesen.

Der Dienst verwendet *Intents*[1], um die Activity über Statusänderungen zu informieren. Die Activity wiederum verwendet *BroadcastReceiver*, um die Intents zu empfangen und auf Änderungen zu reagieren.

6.3.2 OpenIntents File Manager

Wie bereits in Abschnitt 4.3.4 erwähnt, ist Android für die Wiederverwendbarkeit aller Komponenten konzipiert. Aus diesem Grund verwenden wir fertige Komponenten aus anderen Anwendungen. OpenIntents[2] bietet offene Anwendungen an, die leicht wieder zu verwenden sind. Der OpenIntents File Manager bietet die Action

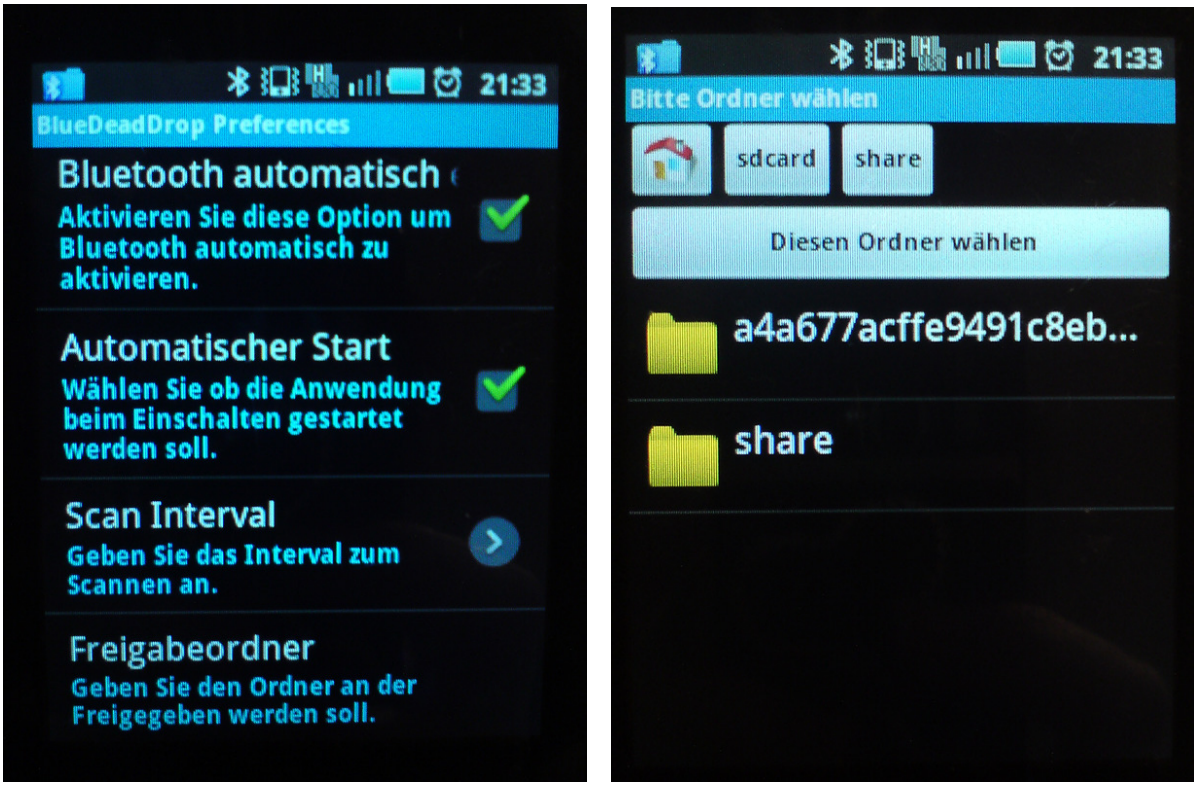
```
org.openintents.action.PICK_DIRECTORY
```

an, die den Benutzer auffordert, einen Ordner auszuwählen. Startet man die Action über ein Intent aus der eigenen Anwendung, gibt die Action den gewählten Ordner als Ergebnis zurück.

6.3.3 BlueDeadDropPreferenceActivity

Android bietet für die Einstellungen eine eigene abgeleitete Activity-Klasse an. Durch die Verwendung von XML als Layout, ist es bei Android möglich, mit einem Code aus nur wenigen Zeilen ein komplexes Menü für die Einstellungen zu erstellen. Dabei wird nicht nur das Layout erstellt. Das Menü ist voll funktionsfähig. Der Aufruf *addPreferencesFromResource(R.xml.preferences);* erstellt anhand der XML Datei die kompletten Einstellungen. Auf Abbildung 6.3 (a) sind die Einstellungen der Anwendung zu sehen.

[1] Intents sind abstrakte Beschreibungen von Aktionen die in Android ausgeführt werden können.
[2] `http://www.openintents.org`



(a) Preference Screen (b) OI File Manager

Abbildung 6.3: Preference-Screen mit OI File Manager

Android bietet aber keine fertige *Preference* an, um Verzeichnisse auszuwählen. Aus diesem Grund wurde die *PreferenceActivity* erweitert. Über einen *OnPreferenceClick-Listener* wird abgefragt, ob der Benutzer das entsprechende Menü für das Verzeichnis ausgewählt hat. Ist dies der Fall, startet eine Activity vom *OpenIntents File Manager*, was in Abbildung 6.3 (b) zu sehen ist. Die Texte 'Bitte Ordner wählen' und 'Diesen Ordner wählen' werden der Activity über ein Intent übergeben. Hat der Benutzer ein Verzeichnis ausgewählt, erhält die *PreferenceActivity* den Pfad ebenfalls innerhalb eines Intents.

6.4 Dienst

Der Dienst verrichtet die eigentliche Arbeit. Dieser läuft unbemerkt im Hintergrund. Er wird über die Activity vom Benutzer gestartet und beendet. Zu Beginn werden mehrere Threads gestartet. Ein Thread namens *AcceptThread* lauscht auf einkommende Verbin-

dungen. Die Suche nach anderen Geräten wird über einen Timer ins Laufen gebracht. Ist die Suche beendet, wird der *ConnectThread* gestartet, und versucht sich mit den gefundenen Geräten zu verbinden. Sobald eine Verbindung aufgebaut ist, kommt der *ConnectedThread* zum Einsatz.

6.4.1 AcceptThread

Sobald der Dienst gestartet ist, wartet die Anwendung auf eingehende Verbindungen. Das wird durch die Aufrufe

```
BluetoothServerSocket serverSocket =
    mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME,
        MY_UUID);
socket = serverSocket.accept();
```

realisiert. Die beiden Parameter NAME und MY_UUID werden in die Datenbank des SDP[3] eingetragen, damit andere Geräte den Dienst finden können. Der zweite Aufruf ist eine blockierende Funktion und wartet auf eine eingehende Verbindung. Damit das Programm auf weitere Eingaben reagieren kann, sind diese Aufrufe in einem separaten Thread.

Sobald eine Verbindung von außen eingegangen ist, wird ein Socket erstellt. Bei Android sind beide Enden eines Sockets gleichwertig. Es gibt keine explizite Client- bzw. Server-Seite. Damit die Anwendung als Server oder Client agiert, wird die entsprechende Variable auf *true* gesetzt. In diesem Fall agiert das Gerät als Server.

Ist die Verbindung hergestellt übernimmt der *ConnectedThread* die Datenübertragung. Vor dem Start wird der Timer für die Suche, sowie ein eventuell laufender *ConnectThread* beendet, da eine Suche nach anderen Geräten oder ein Verbindungsaufbau die Datenübertragung negativ beeinflussen. Vor dem Ende des *AcceptThread* wird ein *ConnectedThread* erstellt und diesem der Socket übergeben. An dieser Stelle endet der *AcceptThread*. Abbildung 6.4 zeigt den Ablauf in einem Aktivitätsdiagramm.

6.4.2 Suche nach Geräten

Über die Einstellungen der Anwendung kann der Abstand zwischen zwei Suchvorgängen festgelegt werden. Der Dienst erstellt beim Starten einen Timer, der die Suche in

[3] siehe Abschnitt 3.3.2

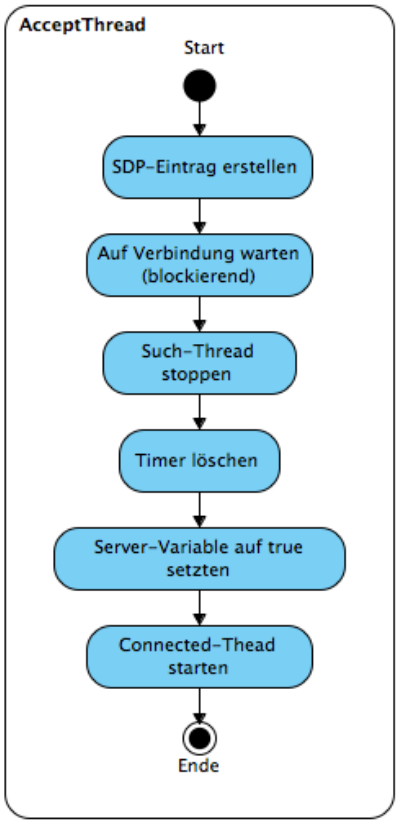


Abbildung 6.4: Aktivitäts Diagramm: AcceptThread

den eingestellten Abständen startet. Über einen *OnPreferenceChangeListener* wird der Dienst über Änderungen informiert. Wurde der Abstand vom Benutzer geändert, startet der Dienst den Timer neu.

In der Dienst-Klasse ist ein *BroadcastReceiver* implementiert. Dieser reagiert auf folgende Broadcasts:

BluetoothAdapter.ACTION_DISCOVERY_STARTED Die Suche nach Geräten wurde gestartet.

BluetoothAdapter.ACTION_DISCOVERY_FINISHED Die Suche nach Geräten wurde beendet.

BluetoothDevice.ACTION_FOUND Ein Gerät wurde gefunden.

Der Benutzer erhält bei den genannten Ereignissen eine kurze Benachrichtigung in Form einer Toast Nachricht[4]. Beim Start der Suche bleibt es bei einer Benachrichtigung. Wurde

[4]Dabei handelt es sich um eine Nachricht, die dem Benutzer auf dem Bildschirm dargestellt wird, jedoch die Bedienung der aktuellen Anwendung nicht behindert.

ein Gerät gefunden, schreibt der Dienst dieses in die Datenbank. Danach informiert er die Activity anhand eines Broadcast über die Änderung. Dies geschieht über folgenden Aufruf:

```
sendBroadcast(new Intent(ACTION_NEW_DEVICE_FOUND));
```

Sobald die Suche beendet wurde, kann davon ausgegangen werden, dass alle sichtbaren Geräte in Reichweite gefunden wurden. Die Suche nach Geräten ist damit beendet und der Dienst startet den *ConnectThread*.

6.4.3 ConnectThread

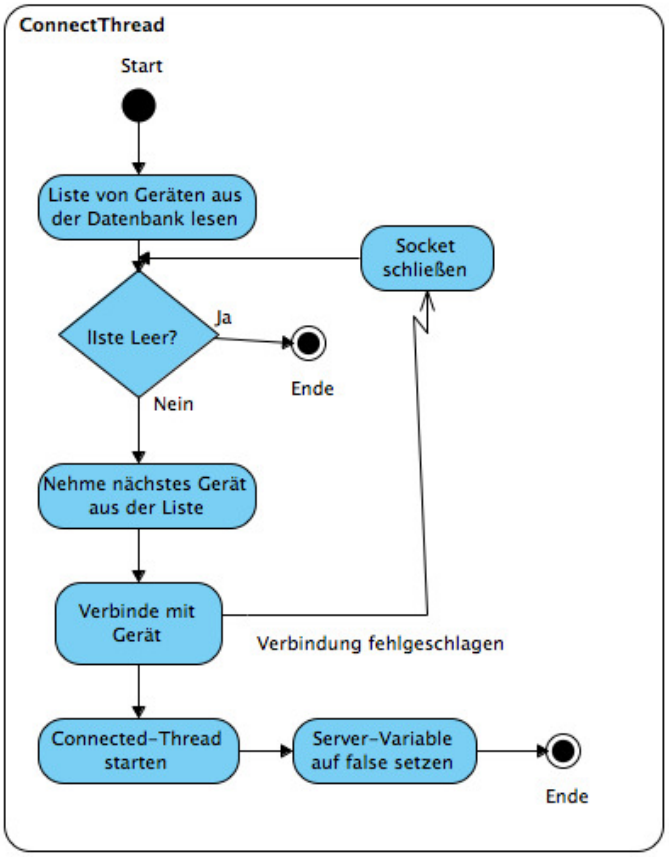


Abbildung 6.5: Aktivitäts Diagramm: ConnectThread

Dieser Thread versucht eine Verbindung mit einem Gerät herzustellen. Dazu verwendet der Thread die Liste der gefundenen Geräte aus der Datenbank. Eine Schleife iteriert über die Liste aller Geräte. Hat die Liste keine Einträge, ist der Thread am Ende.

Nach und nach wird mit jedem Gerät ein Verbindungsversuch gestartet. Ein Exception-Handler fängt Verbinungsfehler ab und sorgt dafür, dass mit dem nächsten Gerät fortgefahren werden kann. Ist ein Verbindungsversuch geglückt, startet der *ConnectedThread*. Vorher wird noch die Server-Variable auf *false* gesetzt. Das Vorgehen ist in Abbildung 6.5 in Form eines Aktivitätsdiagramms dargestellt.

6.4.4 ConnectedThread

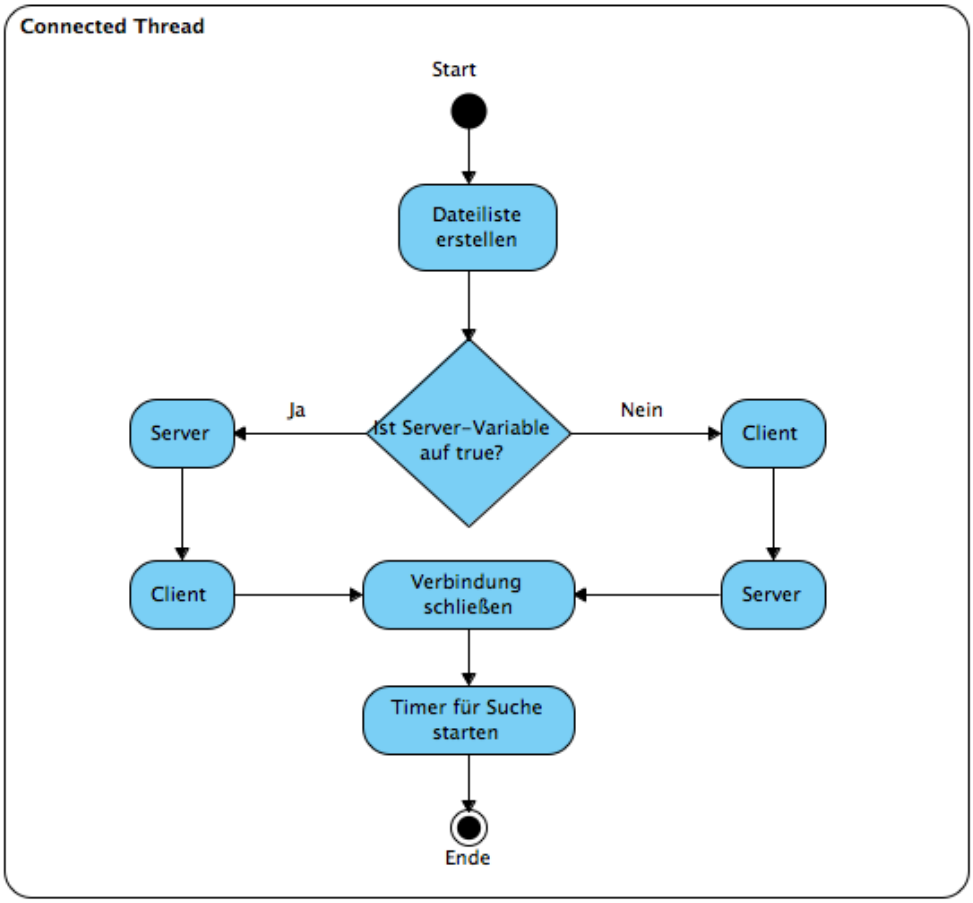


Abbildung 6.6: Aktivitätsdiagramm: ConnectedThread

Ist der Such- und Verbindungsprozess beendet und eine Verbindung mit einem anderen Gerät aufgebaut, kann die Datenübertragung beginnen.

Zunächst wird eine Instanz der *BlueDeadDropFileList* erstellt. Der Konstruktor erhält den Pfad des freigegebenen Verzeichnisses und sucht diesen rekursiv nach Dateien und Verzeichnissen ab. Eine Liste mit *BlueDeadDropFiles* wird erstellt.

Ein *BlueDeadDropFile* repräsentiert eine Datei oder Verzeichnis. Diese Klasse kann Dateien aus einem Pfad oder einem InputStream erstellen. In beiden Fällen generiert die Klasse den zugehörigen MD5 Hash der Datei. Zusätzlich ist es möglich, die repräsentierte Datei in einen OutputStream zu schreiben.

Wie in Abbildung 6.6 zu sehen ist, folgt eine Verzweigung. Agiert das Gerät als Server, startet zuerst die Server-Funktion und danach die Client-Funktion. Auf dem entfernten Gerät ist es umgekehrt. Die Server- und Client-Funktionen werden im Folgenden näher erläutert.

Server

Die Server-Funktion besteht aus einer Schleife, die über alle Dateien in der *BlueDeadDropFileList* iteriert. Die *toString();* Funktion der *BlueDeadDropFile* sieht wie folgt aus:

```
@Override
public String toString() {
    if(isDirectory()) {
        return "DIR " + getAbsolutePath().replaceFirst(
           sharePath, "");
    }

    return "FILE " + length() + " " + getMd5() + " " +
       getAbsolutePath().replaceFirst(sharePath, "");
}
```

Ist die Datei ein Verzeichnis, gibt die Funktion 'DIR ' und den relativen Pfad zum Freigabeverzeichnis zurück.

Für Dateien fängt der String mit 'FILE ' an. Es folgt die Länge der Datei in Bytes, der MD5-Hash als Hexdezimale und zum Schluss wieder der relative Pfad zum Freigabeverzeichnis.

Diesen String sendet der Server an den Client. Der Server wartet auf eine Antwort vom Client. Es gibt zwei mögliche Antworten 'ACCEPT' oder 'SKIP'. 'ACCEPT' gibt an, das der Client die Datei akzeptiert. Dem aktuellen *BlueDeadDropFile* wird der OutputStream vom Socket übergeben. Die Datei schreibt sich dann selbständig in den Stream. Empfängt der Server ein 'SKIP' vom Client, überspringt er die Datei.

Am Ende der Schleife sendet der Server ein 'CLOSE' zum Client und beendet damit die Server-Funktion.

Client

Der Client beginnt damit den Empfang der Dateien vom Server vorzubereiten. Für jedes entfernte Gerät wird ein Verzeichnis im Freigabeverzeichnis erstellt. Mithilfe einer SHA-Prüfsumme (Hash) wird ein eindeutiger und anonymer Name für dieses Verzeichnis erstellt. Der Hash wird aus folgender Zeichenkette erstellt:

```
MY_UUID.toString() + mmSocket.getRemoteDevice().getName() +
    mmSocket.getRemoteDevice().getAddress()
```

Dieses Verzeichnis ist für ein Gerät immer gleich. So können bei einer erneuten Begegnung mit dem entfernten Gerät die bereits übertragenen Dateien dem Gerät zugeordnet werden. Der Name bleibt durch die SHA-Prüfsumme anonym.

Analog zur Server-Funktion beginnt eine Schleife. Diese hat kein Abbruchkriterium und muss mittels eines *break* beendet werden. Die Funktion wartet auf eine Nachricht vom Server, um darauf reagieren zu können. Es gibt drei verschiedene Nachrichten, die erwartet werden:

1. Mit 'CLOSE' wird die Schleife abgebrochen und die Client-Funktion beendet.
2. Eine Nachricht, die mit 'DIR ' beginnt, veranlasst die Funktion das angegebene Verzeichnis zu erstellen, falls es nicht bereits existiert.
3. Beginnt die Nachricht mit 'FILE ', muss überprüft werden, ob die Datei bereits existiert. Die empfangene Nachricht wird dazu in vier Teile getrennt. Der erste Teil ist die Zeichenkette 'FILE'. Der zweite Teil die Länge der Datei in Bytes. Die MD5-Prüfsumme der Datei ist im dritten Teil zu finden. Der letzte Teil ist der relative Pfad der Datei. Die Funktion überprüft, ob die Datei bereits vorhanden ist und ob die Länge und die MD5-Prüfsumme übereinstimmen. Die Nachricht 'SKIP' teilt dem entfernten Gerät mit, dass die Datei übersprungen wird. Ein 'ACCEPT' veranlasst den Server, die Datei zu übertragen. Die Klasse *BlueDeadDropFile* erstellt die Datei mit den Parametern:
   - Stream
   - Länge

- MD5
- Pfad

Die Klasse weiß durch die Längenangabe, wie viele Bytes aus dem Stream gelesen werden sollen. Mit der MD5-Prüfsumme kann die korrekte Übertragung überprüft werden. Bei einer unvollständigen Übertragung wird die Datei gelöscht.

Kapitel 7

Evaluation

7.1 Testgeräte

Die beiden Testgeräte sind das Samsung GT-I5500 und das T-Mobile G1. Das T-Mobile G1 war eines der ersten Mobiltelefone mit Android Betriebssystem. Es wurde von der Firma HTC entwickelt und ist auch als HTC Dream bekannt. Die technischen Details zu den beiden Geräten sind in Tabelle 7.1.

Gerät	T-Mobile G1	Samsung GT-I5500
Android	2.2.1	2.2
Prozessor	ARM11 528 MHz	ARM11 600 Mhz
RAM	192 MB	184 MB
ROM	256 MB	184 MB
Display	320x480 Pixel 3,2 Zoll	240x320 Pixel 2,8 Zoll
Bluetooth	2.0 + EDR	2.1 + EDR
WLAN	802.11b/g	802.11b/g
Größe	117,7mm × 55,7mm × 17,1mm	108mm x 56mm x 12.3mm
Gewicht	158g	102g

Tabelle 7.1: Technische Daten der untersuchten Geräte

Beide Geräte verwenden Android Version 2.2. Das Gerät von Samsung verwendet die Firmware des Herstellers. T-Mobile bietet für das G1 eine Aktualisierung auf die Version 1.6. Die Systempartition ist nur ca. 70 MB groß. Aus diesem Grund ist es auf normalem Weg nicht möglich, eine neuere Version zu installieren. Mittlerweile gibt es allerdings Portierungen für das T-Mobile G1. Dabei wird die SD-Karte partitioniert und ein Teil

dem Betriebssystem zur Verfügung gestellt. Die SD-Karte ist zum Betrieb des Mobiltelefons zwingend notwendig, da hier ein Teil des Betriebssystems liegt. Für das G1 wird die Portierung *MagicOTA-GalisMod*[1] verwendet.

Die Leistung der beiden Geräte ist vergleichbar. Beide haben einen ARM11 Prozessor mit bis zu 600 Mhz und einen vergleichbaren Arbeitsspeicher.

Der für dieses Projekt bedeutendste Unterschied ist die Bluetooth-Version. Das GT-I5500 hat bereits Bluetooth-Version 2.1+EDR, während das G1 die Version 2.0+EDR nutzt. Wie im Abschnitt 3.5 beschrieben, wurde mit der Version 2.1 das Pairing vereinfacht.

7.2 Installation

Android-Anwendungen liegen in Form von Dateien im Dateiformat '.apk' vor. Für die Installation muss zunächst die Datei auf das Telefon übertragen werden. Es gibt verschiedene Möglichkeiten dies zu tun:

- Download über eine Webseite.
- Upload via Bluetooth mit Object-Push.
- Kommandozeilen Programm *adb* vom Android SDK
- ...

Bevor die Datei installiert werden darf, muss eine Einstellung geändert werden. In den Einstellungen von Android findet man das Menü 'Anwendungen'. Dort ist der Hacken bei 'Unbekannte Herkunft' zu setzen. Abbildung 7.1 zeigt diese Einstellung.

Im nächsten Schritt ist ein *File Manager* notwendig. Im Normalfall ist bereits einer installiert. Sollte keiner verfügbar sein, ist der *OI File Manager* über den Android Market zu finden. Dieser wird empfohlen, da das Programm diese Anwendung benötigt.

Über einen *File Manager* lässt sich die Datei mit der Endung '.apk' auswählen und installieren.

[1] `http://www.android-hilfe.de/root-hacking-modding-fuer-t-mobile-g1/68203-rom-froyo-magicota-galismod.html`

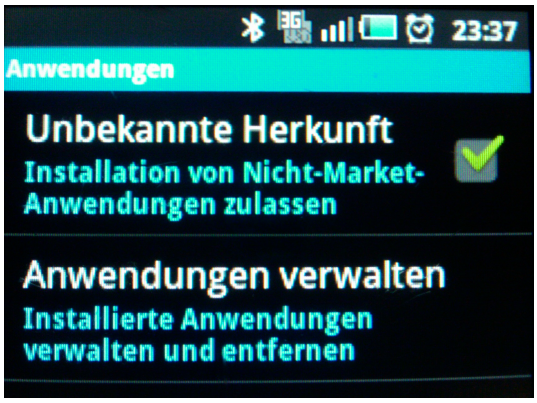


Abbildung 7.1: Einstellung 'Unbekannte Herkunft' unter Android 2.2

7.3 Erster Start der Anwendung

Während des Startvorgangs überprüft die Anwendung, ob der *OI File Manager* installiert ist. Ist dies nicht der Fall, erscheint ein Dialog, wie er in Abbildung 7.2 zu sehen ist. Beide Auswahlmöglichkeiten beenden die Anwendung. Wählt der Anwender 'Installieren', startet die Anwendung eine Suche im Android Market. In den Suchergebnissen ist die Anwendung *OI File Manager* zu sehen und der Anwender kann diese installieren.

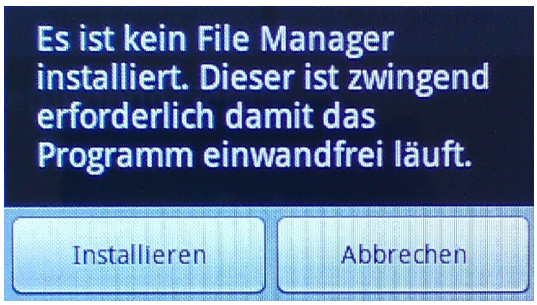


Abbildung 7.2: Dialog: Fehlende Anwendung

Ist der *OI File Manager* installiert und Bluetooth nicht aktiviert, wird der Benutzer aufgefordert Bluetooth zu aktivieren.

Sobald beide Anforderungen erfüllt sind, startet die Anwendung. Der Benutzer kann über die Schaltfläche 'Service starten' den *BlueDeadDropService* starten. Allerdings erscheint beim ersten Start eine Aufforderung, den Pfad in den Einstellungen zu setzen. Über die Taste *Menü* gelangt der Anwender zu den Einstellungen. Diese bietet folgende Optionen:

Bluetooth automatisch einschalten Ist diese Option aktiviert, aktiviert die Anwendung, wenn nötig, Bluetooth von selbst.

Automatischer Start Diese Option startet den *BlueDeadDropService* beim Einschalten des Gerätes.

Scan Interval Gibt an, in welchen Abständen die Suche nach anderen Geräten startet.

Freigabeordner Diese Einstellung startet den *OI File Manager* und der Anwender kann den Ordner auswählen, den er freigeben möchte.

Sobald der Anwender ein Verzeichnis ausgewählt hat, erstellt die Anwendung darin einen Ordner 'share'. In diesen kann der Anwender seine Dateien kopieren, die er teilen will. Im selben Verzeichnis erstellt die Anwendung die Ordner mit den Dateien der entfernten Geräte.

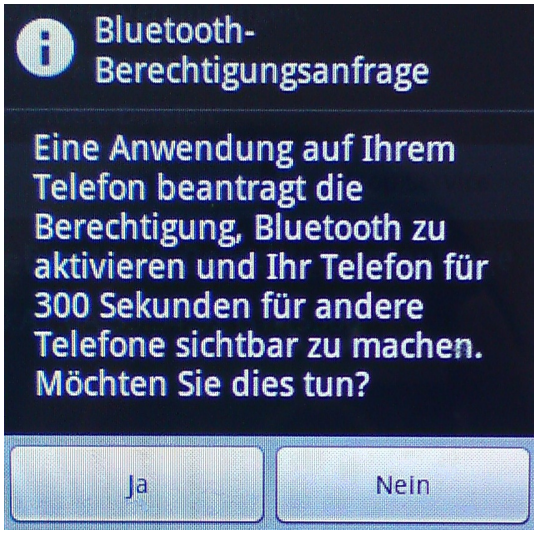


Abbildung 7.3: Gerät sichtbar machen

Wurde ein Pfad vom Anwender ausgewählt, kann er den Dienst starten. In der Benachrichtigungszeile ist das *BlueDeadDrop* Icon zu sehen, solange der Dienst aktiv ist. Der Benutzer kann diese Benachrichtigung auswählen und gelangt so zur Anwendung. Ist das Gerät nicht sichtbar, erscheint ein Dialog (siehe Abbildung 7.3).

7.4 Erster Kontakt mit anderen Geräten

Sobald mindestens zwei Geräte in Reichweite sind und mindestens eines davon sichtbar ist, kommt es zu einer Verbindung. Vor der ersten Verbindung müssen die Geräte gepaart werden. Geräte mit einer Bluetooth-Version bis einschließlich 2.0 haben noch das alte *Pairing-Verfahren* implementiert. Ein Benutzer wird aufgefordert einen PIN einzugeben.

Wie in Abbildung 7.4 zu sehen ist, schlägt Android die PIN ’0000’ und ’1234’ vor. An diesem Beispiel sieht man, das das alte *Pairing-Verfahren* nicht sicher ist, da die PIN sich leicht erraten lässt. Bei diesem Projekt ist es allerdings erwünscht, dass man die PIN erraten kann.

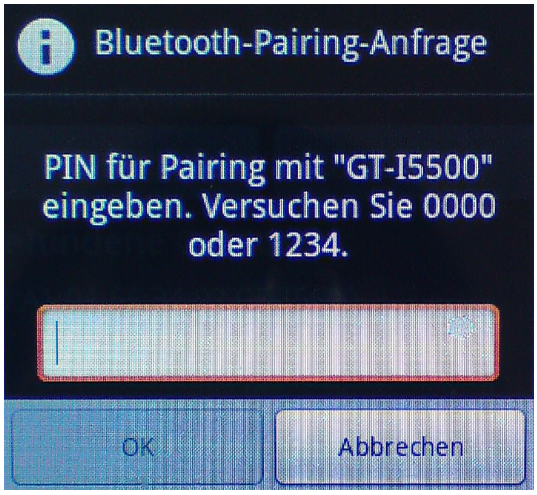


Abbildung 7.4: Pairing bis Bluetooth-Version 2.0

Ab der Bluetooth-Version 2.1 kommt das neue *Pairing-Verfahren* zum Einsatz. Dieses ist sicherer und schneller. Der Benutzer musst sich keine PIN ausdenken und der andere Teilnehmer muss diese nicht erraten. Abbildung 7.5 zeigt eine Pairing-Anforderung nach dem neuen Verfahren.

Abbildung 7.5: Pairing ab Bluetooth 2.1

7.5 Datenübertragung

Die Datenübertragung beginnt sobald zwei Geräte miteinander verbunden sind. In der *Activity* wird dies oben rechts angezeigt. Bis auf diese Anzeige in der Activity bekommt der Anwender nichts von der Verbindung mit. Abbildung 7.6 zeigt die Anwendung mit einer Statistik nach einigen Verbindungen.

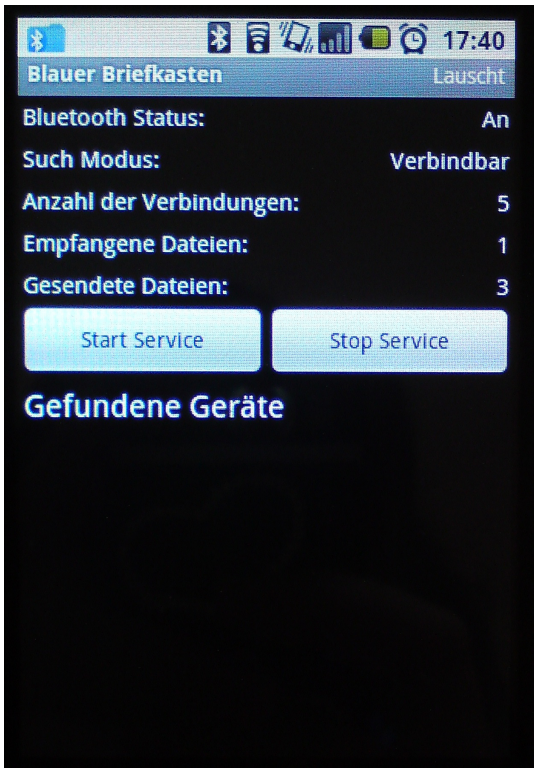


Abbildung 7.6: Grafische Oberfläche von BlueDeadDrop

7.6 Ergebnis

Die Sichtbarkeit der Android-Geräte und das *Pairing-Verfahren* sind problematisch für dieses Projekt. Von einer Anwendung die sich selbstständig mit anderen verbindet und Dateien austauscht, ist diese Weit entfernt.

Von den Anwendern kann nicht erwartet werden, häufig einen Dialog auf ihrem Mobiltelefon zu beantworten, damit das Gerät sichtbar bleibt. Android ist ein offenes Betriebssystem. Aus diesem Grund lässt sich die Sichtbarkeit dauerhaft aktivieren. Dazu

ist aber ein *rooten* des Mobiltelefons notwendig. Nicht jeder Anwender verfügt über die dafür erforderlichen Kenntnisse. Hinzu kommt ein eventueller Garantieverlust.

Das alte *Pairing-Verfahren* stellt sich als Glücksspiel heraus. Der PIN muss erraten werden um erfolgreich eine Verbindung herzustellen. Dieses ist aber nicht das einzige Problem. Bei beiden *Pairing-Verfahren* muss der Benutzer aktiv werden. Geschieht dies mit einer Verzögerung, oder überhaupt nicht, schlägt die Verbindung fehl.

Die Anwendung eignet sich für Benutzer die bereit sind häufiger Dialoge auf ihrem Mobiltelefon zu beantworten. Alle *BlueDeadDrop* Benutzer können sich z.B. auf den PIN '0000' einigen. Sind diese beiden Voraussetzungen erfüllt, steht dem anonymen und zensurresistenten Datenaustausch nichts mehr im Wege.

Kapitel 8

Alternativen

8.1 Übertragungstechnik

Viele Mobiltelefone unterstützen, neben Bluetooth, die Übertragungstechniken WLAN und Infrarot. Auf diese Beiden wird im folgenden näher eingegangen.

8.1.1 Wireless Local Area Network (WLAN)

WLAN ist ein Standard für kabellose lokale Netzwerke. Die aktuelle Spezifikation IEEE 802.11n verwendet die Frequenzen 2,4 GHz und 5 GHz. Ebenso wie bei Bluetooth liegen diese im ISM-Band[1] und können frei verwendet werden.

WLAN verwendet eine höhere Signalbreite und erreicht damit höhere Datenraten als Bluetooth. Damit steigt auch die Leistungsaufnahme. Im mobilen Bereich wirkt sich dies stark auf die Laufzeit aus.

Um über WLAN eine direkte Verbindung zwischen zwei Geräten aufzubauen, muss ein Ad-Hoc-Netzwerk aufgebaut werden. Ein Ad-Hoc-Netzwerk ist ein dezentrales Netzwerk bei dem es keinen zentralen Zugangspunkt (Access Point) gibt. Alle Teilnehmer müssen die gleichen Parameter für folgende Einstellungen haben:

- Service Set ID (SSID)
- Kanalnummer
- Verschlüsselungsverfahren

[1] Industrial Scientific and Medical Band

- Schlüssel

Zusätzlich dürfen keine zwei Teilnehmer dieselbe IP Adresse verwenden. Mit steigender Anzahl an Teilnehmern in einem Netz, steigt auch die Anzahl der Kollisionen und die Übertragungsrate verringert sich.

Ein Gerät kann immer nur in einem Netz sein. Normalerweise werden offene WLANs benutzt, um das Datenvolumen, das über das Mobilfunknetz geht, zu verringern. Teilnehmer eines Ad-Hoc-Netzwerks müssen auf diese Möglichkeit verzichten.

WLAN ist nicht die richtige Technik um kurzfristig ein anonymes Netzwerk aufzubauen um wenige Dateien zu übertragen.

8.1.2 Infrarot

Für dieses Projekt ist Infrarot nicht geeignet. Denn bei Infrarot ist ständiger Sichtkontakt nötig, um die Verbindung aufrecht zu halten. Eine automatische Verbindung ist somit unmöglich.

8.2 Plattform

Android ist aktuell[2] die einzige freie und offene mobile Plattform. Die Alternativen Plattformen sind properitär und unterliegen den Einschränkungen der Hersteller. Die Abbildung 8.1 zeigt die weltweiten Verkaufszahlen von Smartphones.

Als Beispiel einer alternativen Plattform wird im Rahmen dieser Arbeit auf iOS von Apple eingegangen. iOS ist ein Betriebssystem das auf MacOS X basiert und für mobile Geräte angepasst wurde. iOS läuft auf folgenden Geräten:

- iPhone
- iPod
- iPad
- Apple TV

Apple bietet für seine Plattformen eine umfangreiche Entwicklungsumgebung namens Xcode an. Xcode beinhaltet SDKs für die Entwicklung von Anwendungen für MacOS X und iOS in der Sprache Objective C.

[2]Stand: August 2011

Worldwide smart phone market
Market shares Q4 2010, Q4 2009

OS vendor	Q4 2010 shipments (millions)	% share	Q4 2009 shipments (millions)	% share	Growth Q4'10/Q4'09
Total	101.2	100.0%	53.7	100.0%	88.6%
Google*	33.3	32.9%	4.7	8.7%	615.1%
Nokia	31.0	30.6%	23.9	44.4%	30.0%
Apple	16.2	16.0%	8.7	16.3%	85.9%
RIM	14.6	14.4%	10.7	20.0%	36.0%
Microsoft	3.1	3.1%	3.9	7.2%	-20.3%
Others	3.0	2.9%	1.8	3.4%	64.8%

*Note: The Google numbers in this table relate to Android, as well as the OMS and Tapas platform variants
Source: Canalys estimates, © Canalys 2011

Abbildung 8.1: Weltweite Verkaufszahlen von Smartphones

Wie bereits erwähnt, ist die Plattform properitär und man ist auf die Hilfsmittel von Apple angewiesen. Es gibt zwei Wege bei Apple auf die Bluetooth-Hardware zuzugreifen.

Mit einem kostenpflichtigen iOS Developer Benutzerkonto, kann man am Programm 'Made For iPod/iPhone/iPad' teilnehmen. Dieses Programm ist für Hersteller von Zubehör gedacht, das die Übertragung von Audio unterstützt.

Über das Framework Gamekit ist es möglich, Daten zu versenden. Mit diesem Framework kann man Bluetooth-Verbindungen zu anderen iOS Geräten aufbauen. Verbindungen zu Geräten anderer Hersteller sind nicht möglich.

Diese Einschränkungen machen iOS als Plattform unattraktiv für die Entwicklung von Bluetooth-Anwendungen.

Kapitel 9

Fazit

Die Evaluation hat gezeigt, dass die Verwendung von *BlueDeadDrop* nur bedingt praxistauglich ist. Android-Geräte sind maximal fünf Minuten sichtbar. Der Anwender wird nach dieser Zeit aufgefordert, die Sichtbarkeit wieder zu aktivieren. Damit ein Gerät über eine längere Zeit sichtbar bleibt, muss der Anwender sein Bluetooth-Gerät ständig kontrollieren.

Sind zwei Geräte in Reichweite und es kommt zu einer Verbindung, müssen diese Geräte gepaart werden. Der Benutzer wird dann erneut zu einer Eingabe aufgefordert. Bemerkt er dies nicht, schlägt die Verbindung fehl. Bei den Bluetooth-Versionen bis einschließlich 2.0 müssen sich beide Benutzer auf einen PIN einigen. Die Tatsache, dass sich beide Anwender nicht kennen, erschwert das Pairing.

Bluetooth Geräte sind häufig mit dem PC verbunden. Die getauschten Dateien können Viren und Schadsoftware enthalten. Dies macht die Benutzung von *Dead Drops* allgemein (egal ob per USB-Stick oder per Bluetooth) gefährlich. Dessen muss sich der Anwender bewusst sein und entsprechende Sicherheitsvorkehrungen treffen.

Eine Lösung für die ersten beiden Probleme ist eine Übertragungstechnik, die ohne Konfiguration auskommt. Dies erhöht jedoch die Gefahr von Viren und Schadsoftware. Eine optimale Lösung ist aus den genannten Gründen nicht bekannt.

Literaturverzeichnis

[anda] *Android 2.3 Compability Definition*. http://static.googleusercontent.com/external_content/untrusted_dlcp/source.android.com/de//compatibility/android-2.3-cdd.pdf

[andb] *Android Activities*. http://developer.android.com/guide/topics/fundamentals/activities.html

[andc] *Android Developers: What is Android?* http://developer.android.com/guide/basics/what-is-android.html

[andd] *Android Fundamentals*. http://developer.android.com/guide/topics/fundamentals.html

[bsi] *BSIG Homepage*. https://www.bluetooth.org/About/bluetooth_sig.htm

[deaa] *Dead Drops Database*. http://deaddrops.com/dead-drops/db-map/

[deab] *DeadDrops Homepage*. http://deaddrops.com

[HKM10] Hashimi, Sayed Y. ; Komatineni, Satya ; MacLean, Dave: *Pro Android 2*. Berkeley, CA : Apress, 2010 (Safari Tech Books Online). – ISBN 978–1–4302–2659–8

[ico] *Bluetooth Icon*. http://www.iconfinder.com/icondetails/19520/128/bluetooth_icon

[Lüd07] Lüders, Christian-Friedrich: *Lokale Funknetze: Wireless-LANs (IEEE 802.11), Bluetooth, DECT*. 1. Würzburg : Vogel, 2007 (Vogel-Fachbuch). – ISBN 978–3–8343–3018–5

[Mei10] Meier, Reto: *Professional Android 2 application development*. Indianapolis, Ind : Wiley, 2010 (Wrox programmer to programmer). – ISBN 978–0–470–56552–0

[non] *Android on Non-Android Devices*. http://android.stackexchange.com/questions/6849/can-i-install-android-on-my-non-android-device

[oha] *Open Handset Alliance*. http://www.openhandsetalliance.com/oha_faq.html

[Sau08] Sauter, Martin: *Grundkurs mobile Kommunikationssysteme: Von UMTS und HSDPA, GSM und GPRS zu Wireless LAN und Bluetooth Piconetzen*. 3., erweiterte Auflage. Wiesbaden : Friedr. Vieweg & Sohn Verlag | GWV Fachverlage GmbH Wiesbaden, 2008. – ISBN 978–3–8348–0397–9

[wik] *Wikipedia: Toter Briefkasten*. http://de.wikipedia.org/wiki/Toter_Briefkasten

Letzte Einsicht jeweils 31.07.2011