Mobile Applikationen

Dokumentation

Bluetooth Chat

Niklas Röhling, Niklas Timme, Nils Dralle
27.6.2016

Inhalt

Einleitung

Im Rahmen des Studiums im Studiengang Informatik an der Fachhochschule Bielefeld stellte sich die Aufgabe eine Mobile Applikation von Grund auf zu entwickeln. Im Folgenden wird das Vorgehen für das Projekt beschrieben und festgehalten.

Ziele

Ziel des Projektes ist die Entwicklung einer nativen mobilen Applikation zum Versenden von Nachrichten zwischen 2 Endgeräten über Bluetooth. Zielplattform ist dabei Android. Wichtig ist dabei, dass die Verbindung über Bluetooth geschieht. Alle Ziele im Detail sind den Anforderungen im folgendem Kapitel zu entnehmen

Haupt-Meilensteine und grober Zeitplan

Es wurden Meilensteine eingesetzt um Pufferzeiten einzukalkulieren und um den Abgabetermin des Projektes einzuhalten.

Meilenstein	Datum	Bezeichnung
	12.04.16	Projektbeginn
M0	12.04.16	Anforderungsanalyse
	19.04.16	Arbeitseinteilung
M1	04.05.16	Grundfunktionalität Bluetooth und Nachrichtenaustausch
M2	15.05.16	Verschlüsselung
M3	29.05.16	Datenbank
M4	06.06.16	GUI
M5	27.06.16	Projektdokumentation
	07.07.16	Projektabgabe

Benötigte Ressourcen

- **Menschliche Ressourcen:**

  Für das Projekt stehen 3 Entwickler zur Verfügung. Es wird davon ausgegangen, dass alle Entwickler durchschnittlich jeweils 12 Stunden die Woche ausschließlich an dem Projekt arbeiten.

- **Hardware:**

  Als Hardware wird ein mobiles Endgerät benötigt. Auf diesem muss Android laufen. Als Entwicklungsgeräte stehen zur Verfügung: HTC Desire S mit Android 4.4.4, Sony Xperia E3 mit Android 6.0.1, Galaxy S7 Edge mit Android 6.0.1 und HTC One M7 mit Android 5.0.2.

- **Räume:**

  Das Projekt wird voraussichtlich weitgehend virtuell organisiert, jedoch wurde sich einmal die Woche getroffen um den aktuellen Stand des Projektes zu besprechen.

Mitarbeiter

Nils Dralle

ndralle@fh-bielefeld.de

Niklas Timme

ntimme@fh-bielefeld.de

Niklas Röhling

nroehling@fh-bielefeld.de

Budget

Abgesehen von den menschlichen Ressourcen steht kein weiteres Budget zur Verfügung.

An Arbeitszeit stehen insgesamt bei einer angenommenen Projektlaufzeit bis zum 7. Juli 2016, schätzungweise 3 * 12 * 12 = 1296 Entwicklerstunden zur Verfügung.

Bluetooth

Bluetooth ist eine Funktechnologie, die den Datenaustausch mit anderen Geräten ermöglicht. Die Reichweite von Bluetooth beträgt ~10m. Eine Basis-Station (wie z.b. bei WLAN oft verwendet) ist nicht nötig, stattdessen stellen zwei Geräte eine direkte Verbindung miteinander her. Die entwickelte Applikation nutzt diese Technologie als Basis.

GUI

Die Grafische Benutzeroberfläche (GUI) ermöglicht dem Benutzer die Interaktion mit dem System. Sie bietet dem Benutzer die Möglichkeit andere Systeme, welche sich in der Reichweite des jeweiligen Bluetooth-Modul befinden und deren Bluetooth System aktiv ist, anzuzeigen und eine Kommunikation zu beginnen. Außerdem ist es möglich dem Benutzer die bereits bekannten Systeme, wie auch bereits geführte Unterhaltungen anzuzeigen. Zusätzlich wird dem Benutzer die Möglichkeit geboten sein Profil zu betrachten, zu verändern sowie andere Einstellungen vorzunehmen. Wie die die GUI im Detail mit dem Benutzer interagiert wird im folgendem Abschnitt erläutert.

Material Design

Um das Design der Applikation ansprechen zu gestalten wurde Material Design von Google Inc. Verwendet. Material Design ist eine Designsprache, welche für mobile Applikationen entwickelt wurde. Es basiert auf einer kachelartigen Darstellungsweise und man erkennt den Minimalismus. Material Design verwendet Animationen und Schatten um Tiefe zu erzeugen und unwichtige Aspekte in den Hintergrund zu versetzen. Somit kann der Benutzer sich auf die wichtigen Abschnitte konzentrieren. Dieses Design wurde mit der LOLIPOP Version von Android bekannt gegeben.

Tabs

Um dem Benutzer die Möglichkeit bieten zu können schnell zwischen verschiedenen Ansichten zu wechseln, wurden Tabs für das System verwendet. Der Benutzer kann durch Antippen der jeweiligen Registerkarte des Tabs schnell zu dessen Inhalt gelangen. Indem der Benutzer mit einem Finger von links nach rechts bzw. von rechts nach links wischt kann der er ebenfalls die einzelnen Tabs wechseln. Um die Hauptfunktionen der Anwendung dem Benutzer schnell zu visualisieren wird im ersten Tab eine Liste der in der Umgebung gefundenen Geräte angezeigt (Abbildung 1). Mit dem zweiten Tab werden dem Benutzer die Geräte bzw. Kontakte angezeigt mit denen sich das Gerät vorher bereits verbunden hat. Im dritten Tab werden dem Benutzer die zurzeit offenen Unterhaltungen angezeigt.

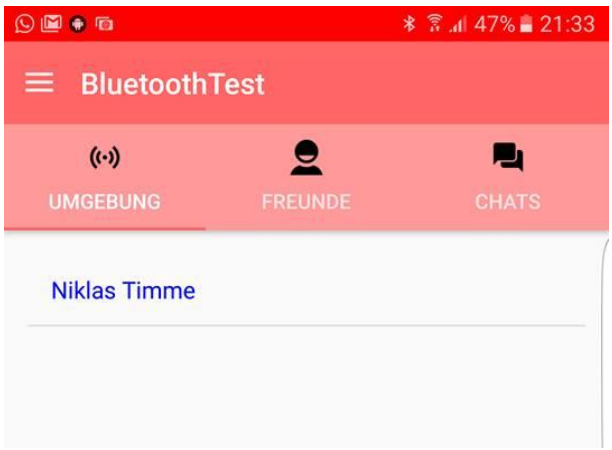

Abbildung 1: Tabs

Um besser verstehen zu können, wie Tabs arbeiten/funktionieren, wird im nächsten Abschnitt auf einzelne Code-Abschnitte eingegangen. Die Abbildung 2 zeigt die Klasse „ViewPagerAdapter“ sowie die Methode „setupViewPager“ aus der Klasse „MainActivity“:

```
private void setupViewPager(ViewPager viewPager) {
    ViewPagerAdapter adapter = new ViewPagerAdapter(getSupportFragmentManager());
    adapter.addFragment(new OneFragment(), "Umgebung"); //TODO: reference strings.xml instead
    adapter.addFragment(new TwoFragment(), "Freunde");
    adapter.addFragment(new ThreeFragment(), "Chats");
    viewPager.setAdapter(adapter);
}

class ViewPagerAdapter extends FragmentPagerAdapter {
    private final List<Fragment> mFragmentList = new ArrayList<>();
    private final List<String> mFragmentTitleList = new ArrayList<>();
    public ViewPagerAdapter(android.support.v4.app.FragmentManager manager) {
        super(manager);
    }
    @Override
    public Fragment getItem(int position) { return mFragmentList.get(position); }
    @Override
    public int getCount() { return mFragmentList.size(); }
    public void addFragment(Fragment fragment, String title) {
        mFragmentList.add(fragment);
        mFragmentTitleList.add(title);
    }
    @Override
    public CharSequence getPageTitle(int position) { return mFragmentTitleList.get(position); }
}
```

Abbildung 2: FragmentPagerAdapter

Die Tabs werden mithilfe eines FragmentPagerAdapter (Abbildung 2) realisiert. Dieser Adapter weißt alle benötigten Methoden auf, welche man benötigt um die Tabs darzustellen. Anhand der Methode setupViewPager(ViewPager viewPager) kann man erkennen wie der FragmentPagerAdapter (ViewPagerAdapter) arbeitet. Jeder Tab der angezeigt wird, wird als Fragment angesehen, welche der ViewPageAdapter in einer

internen Liste speichert.

```
<android.support.design.widget.CoordinatorLayout
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorAccent"
            app:layout_scrollFlags="scroll|enterAlways"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.design.widget.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:tabMode="fixed"
            app:tabGravity="fill"/>
    </android.support.design.widget.AppBarLayout>
```

Abbildung 3: Tabs in activity_main_nt.xml

Anhand der Datei activity_main_nt.xml, welche direkt beim Start des Programmes ausgelesen wird, kann man sehr gut den Grafischen Aufbau der Applikation erkennen (Abbildung 3). Die Definition für die Tabs sind von einem „CoordinatorLayout“-Element umschlossen, was verschiedene Interaktionen zwischen den Unterelementen ermöglicht. Das „AppBarLayout“-Element, welches das direkt Unterelement von „CoordinatorLayout“ ist, ermöglicht das spätere verwenden der Eigenschaft „layout_scrollFlags“ in dem Unterelement Toolbar. Die „Layout_scollFlags“-Eigenschaft bei dem „Toolbar“-Element beschreibt wie die Toolbar sich zu verhalten hat. In unserem Fall reagiert sie auf wischen mit dem Finger sowie das antippen der jeweiligen Überschrift für ein Tab. Das „TabLayout“-Element, welches ein Unterelement von dem „AppBarLayout“-Element ist, beschreibt wie der Inhalt der Tabs visualisiert wird. Durch die Angabe „anddroid:id=“ wird beschrieben wie man die einzelnen Elemente ansteuert.

NavigationView

Mit dem NavigationView Element wird Benutzer durch das Wischen mit einem Finger von links nach rechts oder durch Antippen auf die drei Balken (Abbildung 1) ein

Navigationsmenü angezeigt. Dort kann der Benutzer sein Profilbild erkennen und zwischen 2 Menüpunkten wählen: Profil und Einstellungen (Abbildung 4).

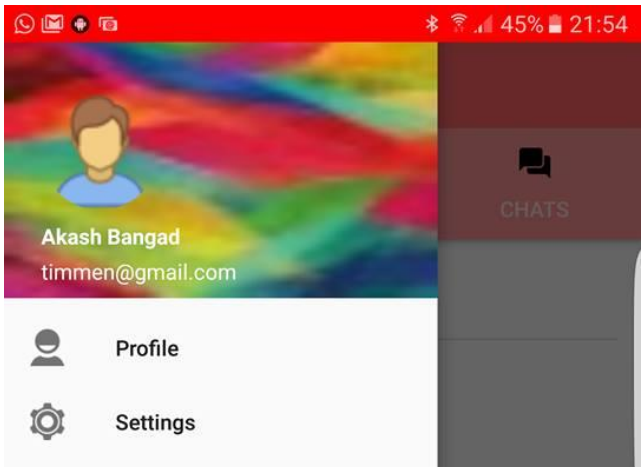

Abbildung 4: Navigationsmenü

Wie das Navigationsmenü implementiert wurde ist zum besseren Verständnis im folgendem Abschnitt erläutert.

```
//Navigation
navigationView = (NavigationView) findViewById(R.id.navigation_view);
navigationView.setNavigationItemSelectedListener(new NavigationView.OnNavigationItemSelectedListener() {
    @Override
    public boolean onNavigationItemSelected(MenuItem menuItem) {
        if(menuItem.isChecked())
            menuItem.setChecked(false);
        else
            menuItem.setChecked(true);

        drawerLayout.closeDrawers();
        switch (menuItem.getItemId()){
            case R.id.profile:
                Toast.makeText(getApplicationContext(),"Profil ausgewählt",Toast.LENGTH_SHORT).show();
                return true;
            case R.id.settings:
                Toast.makeText(getApplicationContext(),"Einstellungen",Toast.LENGTH_SHORT).show();
                return true;
            default:
                Toast.makeText(getApplicationContext(),"Somethings Wrong",Toast.LENGTH_SHORT).show();
                return true;

        }
    }
});
drawerLayout = (DrawerLayout) findViewById(R.id.drawer);
ActionBarDrawerToggle actionBarDrawerToggle = new ActionBarDrawerToggle(this,drawerLayout,toolbar,"Open", "Close"){
    @Override
    public void onDrawerClosed(View drawerView) {
        super.onDrawerClosed(drawerView);
    }
    @Override
    public void onDrawerOpened(View drawerView) {
        super.onDrawerOpened(drawerView);
    }
};
drawerLayout.setDrawerListener(actionBarDrawerToggle);
actionBarDrawerToggle.syncState();
```

Für die Navigation wird die Systemvariable „navigationView“ benutzt (Abbildung 5). Diese Variable wird mit dem „NavigationView“-Element, welche die Eigenschaft „android:id=“@+id/navigation_view““ besitzt, assoziiert. Um die einzelnen Items des Navigationsmenüs anzusprechen ist ein „OnNavigationItemSelectedListener“-Event geschrieben worden. Dieses Event reagiert auf antippen des jeweiligen Items und fragt in der „switch-case“ -Kontrollstruktur ab, welches Item ausgewählt wurde. So wird die jeweilige Aktivität aufgerufen, wo der Benutzer dann entweder Sein Profil bearbeiten oder Einstellungen andern kann.

SwipeRefreshLayout

Das „SwipeRefreshLayout“-Element wurde verwendet um die Wartezeit bis andere Geräte in der Nähe gefunden worden sind mit einer kleinen Simulation zu überbrücken. Wenn der Benutzer sich in dem Tab mit der Überschrift „UMGEBUNG“ befindet, ist es möglich mit einem Finger auf dem Bildschirm von oben nach unten zu wischen. Sobald dies geschehen ist, erscheint eine kleine Simulation in Form eines Striches der sich im Uhrzeigersinn dreht und verschwindet, sobald der „Scan-Vorgang“ abgeschlossen ist. Dies macht die Applikation ansprechender für den Benutzer, da dieser den Eindruck von einem flüssigen Programmfluss vermittelt bekommt.

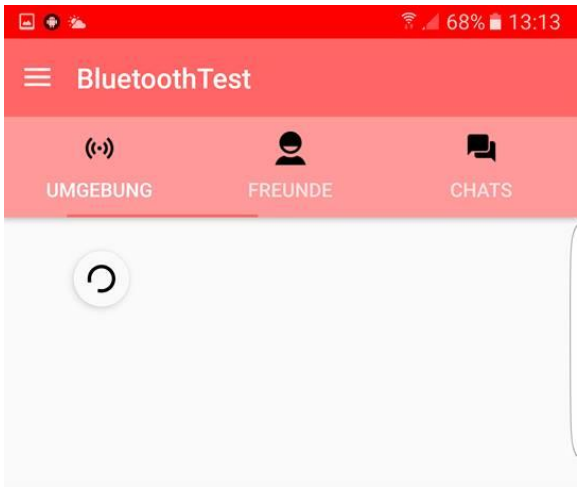

Abbildung 5: SwipeRefreshLayout Visualisierung

Zum besseren Verständnis werden im folgendem Abschnitt einige Highlights im Codeuntersucht.

```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);

    View v= getView();

    mSwipeRefreshLayout = (SwipeRefreshLayout) v.findViewById(R.id.scan_and_refresh);

    mSwipeRefreshLayout.setOnRefreshListener(new SwipeRefreshLayout.OnRefreshListener() {
        @Override
        public void onRefresh() {
            scanAndRefresh();
        }
    });
```

Abbildung 6: OnRefreshListener des SwipeRefreshLayout

Sobald der Benutzer mit einem Finger in dem Tab „Umgebung“ von oben nach unten wischt, soll eine Methode (scanAndRefresh()) aufgerufen werden, welche die geeigneten Geräte findet, darstellt und versucht eine Verbindung aufzubauen. Um auf Die auf diese Geste des Benutzers reagieren zu können bietet „SwipeRefreshLayout“-Klasse einen „OnRefreshListener“-Event an. In dessen „onRefresh()“-Methode wird nun die Methode „scanAndRefresh“ (Abbildung 7) aufgerufen.

```
public void scanAndRefresh(){

    //-------------Scan------------
    serviceConnector.scanForNearbyDevices();

    //-----------devices----------------
    ListView lvDevices = (ListView) getView().findViewById(R.id.listViewDevices);
    lvDevices.setAdapter(displayAdapter);

    lvDevices.setOnItemClickListener((parent, view, position, id) -> {
            Log.i(LOG_TAG,"Clicked device "+id);
            BluetoothDevice btDevice = serviceConnector.getDeviceByIndex((int)id);
            if(btDevice!=null){
                requestNewConnection(btDevice.getAddress());
            }else{
                Log.w(LOG_TAG,"Clicked device is null");
            }
    });
```

Abbildung 7: Methode zum Aktualisieren der gefundenen Geräte

Diese Methode sucht nach Geräten, welche in der Reichweite des Bluetooth-Adapters sich befinden und zeigt diese mithilfe dem „ListView“-Element in der Darstellungsdatei „activity_main.xml“ an. Sobald ein geeignetes Gerät gefunden wurde, wird ein wird um eine neue Verbindung angefragt.

Andere Geräte finden

Um andere Bluetooth-Geräte in der Umgebung zu finden, muss die Umgebung nach anderen Geräten gescannt werden. Dabei sendet das scannende Gerät spezielle Datenpakete aus (Inquiry), die von anderen Bluetooth-Geräten beantwortet werden, vorausgesetzt, das andere Gerät ist „sichtbar“.[Cross und Hoeckle, 2007]

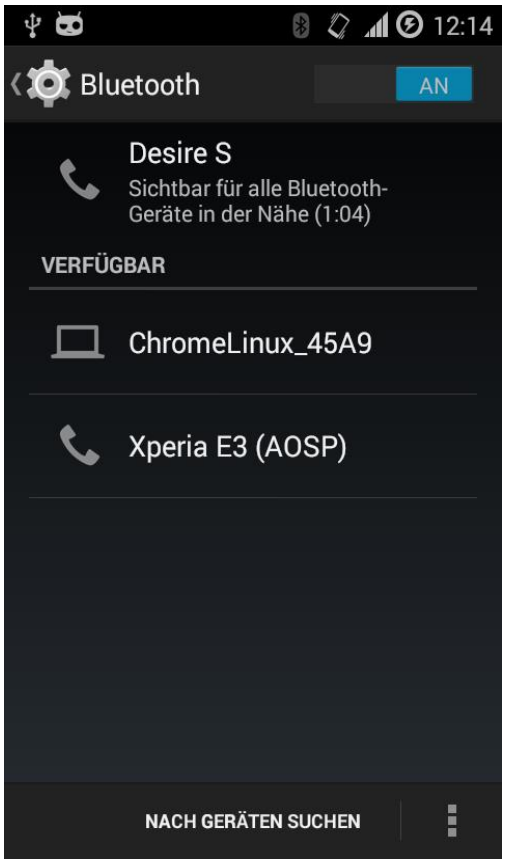

Abbildung 8: Bluetooth-Scan in Android

Zu den Geräten, die ein Bluetooth-Gerät über sich preisgibt, gehören: [Evans und Warren, 2009]

- Der Name des Gerätes. Dieser kann oftmals vom Benutzer festgelegt werden.
- Die MAC-Adresse des Geräts, welche das Gerät eindeutig identifiziert.
- Die „Device major class“ zur Bestimmung des Gerätetyps.
- Die „Device minor class“ zur Bestimmung der Unterklasse eines Gerätetyps.
- Die „Services“, die das Gerät aktuell anbietet.

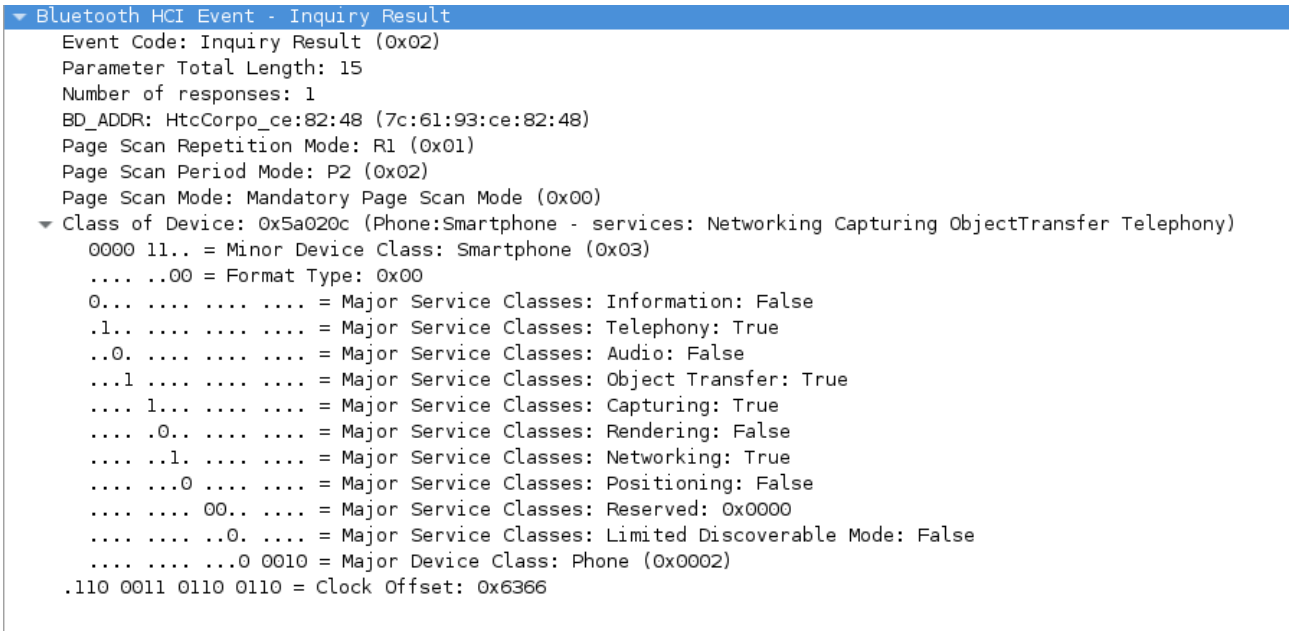

Abbildung 9: Ein Inquiry Result in Wireshark

Die Services, die ein Gerät anbietet, werden über sog. UUIDs bekanntgegeben. UUIDs sind 128 Bit lange, eindeutige Nummern. Jeder Service bekommt eine UUID. Ein Service kann Beispielsweise ein offener RFCOMM-Socket sein.

Bluetooth in Android

Android bietet eine Programmierschnittstelle zur Steuerung der Bluetooth-Hardware eines Geräts. Dafür ist die Berechtigung „android.permission.BLUETOOTH“ nötig, für einige Funktionen allerdings auch „android.permission.BLUETOOTH_ADMIN“. Seit Android 6 ist die Berechtigung „android.permission.ACCESS_COARSE_LOCATION“ notwendig, welche zusätzlich zur Laufzeit eingeholt werden muss.

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

Abbildung 10: Berechtigungen "AndroidManifest.xml"

Der Bluetooth-Scan wird mit der Methode „BluetoothAdapter.startDiscovery“ initiiert.

Um die Ergebnisse zu erhalten, muss ein Broadcast-Receiver für „ACTION_FOUND“ registriert.

```
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(spmaBroadcastReceiver, filter);
```

Abbildung 11: Registrierung des Broadcast-Recievers

Der Broadcast-Receiver muss nach Benutzung am System abgemeldet werden.

Die Ergebnisse werden im Broadcast-Receiver gesammelt und in eine lokale Liste gespeichert. Um die Services zu erhalten, die ein Gerät anbietet, muss auf für jedes Gerät die Methode „fetchUuidsWithSdp“ aufgerufen werden. Dies muss nach dem Ende des Scans geschehen, und die Geräte müssen nacheinander gescannt werden. Um die Ergebnisse des Service-Scans zu erhalten, muss der Broadcast-Receiver zusätzlich für „ACTION_UUID“ registriert werden.

Die erhaltene Liste von UUIDs wird mit den in der App gespeicherten UUIDs verglichen, welche zum Anbieten von Verbindungen genutzt werden.

Services anbieten

Der Aufbau von Bluetooth-Verbindungen in Android funktioniert ähnlich wie der Aufbau von TCP-Verbindungen. Auch für Bluetooth-Verbindungen gibt es einen Server und einen Client, die dafür zuständigen Klassen heißen „BluetoothServerSocket“ und „BluetoothSocket“. Eine Server-Socket wird mit der Methode „listenUsingRfcommWithServiceRecord“ unter Angabe eine 128-Bit UUID erstellt. Ein Client kann sich dann einen Bluetooth-Socket erzeugen und sich mithilfe der Methode „createRfcommSocketToServiceRecord“ unter Angabe der selben UUID verbinden. Auf dem Server wird für die Kommunikation über die neue Verbindung ein neuer Socket erstellt. Zu beachten ist hierbei, das dies nur dann funktioniert, wenn die Geräte vorher gekoppelt wurden.

Seit API-Level 10 stellt Android die Methoden „listenUsingInsecureRfcommToServiceRecord“ und „createInsecureRfcommSocketToServiceRecord“ bereit, die eine Verbindung auch zwischen nicht gekoppelten Geräten ermöglichen. In der App werden je nach Kopplungsstatus der Geräte beide Varianten verwendet.

Der Socket stellt einen Input- und einen Outputstream bereit, über die Daten ausgetauscht werden können. Es ist zu empfehlen, diese Streams in ihren jeweiligen Buffered- Versionen zu kapseln, da gesendete und empfangene Nachrichten sonst oftmals nur in Teilen gesendet werden.

Daten senden und empfangen

Um abzufragen, wie viele Daten an einem Inputstream bereitstehen, kann die Methode „Inputstream.available()“ genutzt werden. Diese Methode gibt allerdings je

nach Gerätehersteller und Android-Version nicht die tatsächliche Anzahl an Bytes zurück, die bereit steht, sondern nur, ob Daten bereitstehen.

Um die bereitstehenden Daten zu erhalten, wird die Methode „Inputstream.read()“ verwendet. Mithilfe dieser Methode werden die Daten in ein Byte-Array geschrieben und können anschließend in einen String umgewandelt werden.

Der umgekehrte Weg, das Senden von Daten, geht etwas einfacher: Der zu sendende String wird in ein Byte-Array umgewandelt und mithilfe von „OutputStream.write()“ in den Stream geschrieben. „OutputStream.flush()“ sorgt dann dafür, dass die Daten gesendet werden.

Interne und externe Nachrichten

Die App nutzt Broadcasts, um intern zwischen der GUI und dem Background-Service zu kommunizieren. Um Broadcasts zu versenden, wird die Methode sendBroadcast();

verwendet. Die Methode steht in den Klassen Activity und Service zur Verfügung. Diese Methode bekommt einen Intent als Parameter, welcher eine Action und die eigentliche Nachricht enthält. Die Action bestimmt, wer den Broadcast empfangen kann. Alle Klassen, die diesen Broadcast empfangen wollen, müssen für die Action einen BroadcastReceiver implementieren.

Die im Intent enthaltende Nachricht ist ein mit JSON formatierter String. Interne Nachrichten können daran erkannt werden, dass sie das Attribut Extern : false haben. Alle internen Nachrichten haben das Attribut Action (nicht zu verwechseln mit der im Intent enthaltenen Action), welches eine Anweisung für den Empfänger beinhaltet sowie weitere, von der Action abhängige, optionale Attribute.

Im Gegensatz dazu gibt es auch externe Nachrichten. Dies sind die Nachrichten, die über Bluetooth mit anderen Geräten ausgetauscht werden. Auch diese sind mit JSON formatiert. Alle externen Nachrichten haben das Attribut Extern : true sowie weitere, vom Inhalt der Nachricht abhängige Attribute.

Datenverarbeitung

Um die Verarbeitung der empfangenen Daten zu vereinfachen, werden alle übertragenen Nachrichten mit JSON formatiert. Android stellt dazu im Package „org.json“ einige Klassen zur Verarbeitung von JSON bereit.

{"ReceiverAddress":"00:00:00:00:5A:AD","Sender":"Desire S","Extern":true,"Message":"UZvgZljeKpBjuAlM1lgeIw==\n","SenderAddress":"7C:61:9

Abbildung 12: Beispiel einer mit JSON formatierten Nachricht

Verschlüsselung

Da das Projekt um das Chatten mit anderen Benutzern geht, muss die Privatsphäre gewahrt werden. Dies wird erreicht durch eine RSA gepaart mit einer AES Verschlüsselung.

AES Verschlüsselung

Der Advanced Encryption Standard (AES) wurde von Joan Daemen und Vincent Rijmen entwickelt und vom National Institute of Standards and Technology als Standard bekanntgegeben. Er ist eines der meistgenutzten und sichersten Verschlüsselungsverfahren. Seine Funktionsweise beruht auf einer Reihe von Byteersetzungen, Verwürfelungen und linearen Transformationen, die auf 16 Byte Datenblöcken ausgeführt werden. Diese Operationen werden mehrmals wiederholt bis man letztendlich den AES Schlüssel generiert hat. Die Blocklänge beschränkt sich auf 128 Bit und und die Schlüssellänge auf 128, 192 oder 256 Bit. AES-192 und AES-256 sind in den USA für staatliche Dokumente mit höchster Geheimhaltungsstufe zugelassen.

RSA Verschlüsselung

RSA, benannt nach den Entwicklern Rivest, Shamir und Adleman, ist im Gegensatz zu AES ein asymmetrisches Verschlüsselungsverfahren. Es kann zum Verschlüsseln aber auch zum digitalen Signieren verwendet werden. Der Algorithmus benutzt hierbei einen privaten Schlüssel und einen öffentlichen Schlüssel. Der private Schlüssel wird zum Entschlüsseln benutzt und der öffentliche dient zum Verschlüsseln. Es ist wichtig, dass der private Schlüssel geheimgehalten wird.

Zusammenspiel RSA und AES

In dem Projekt werden beide Verfahren angewandt um die höchstmögliche Sicherheit zu erreichen. Dies bedeutet, dass die eigentliche Nachricht mit AES verschlüsselt wird und der AES-Schlüssel für den Schlüsseltausch mit RSA verschlüsselt wird.

Schlüsseltausch

Nachdem eine Verbindung hergestellt wurde, müssen die Schlüssel ausgetauscht werden. Zu beachten ist hierbei, das beide beteiligten Geräte drei Schlüssel haben: einen AES-Schlüssel, einen RSAPublic-Schlüssel und einen RSAPrivate-Schlüssel. AES- und RSA-Public-Schlüssel werden ausgetauscht. Beide Geräte senden zunächst einen "DataRequest" mit dem Type "RSAPublic". Daruafhin sendet das jeweils andere Gerät eine "DataResponse"-Nachricht mit demselben Type und dem RSAPublic-Schlüssel. Im nächsten Schritt senden beide Geräte eine erneute "DataRequest"-Nachricht mit dem Type "AESEncrypted". Das empfangende Gerät verschlüsselt daraufhin seinen AES-Schlüssel mit dem vorher

empfangenen RSAPublic-Schlüssel und sendet diesen dann an zurück.

Implementierung

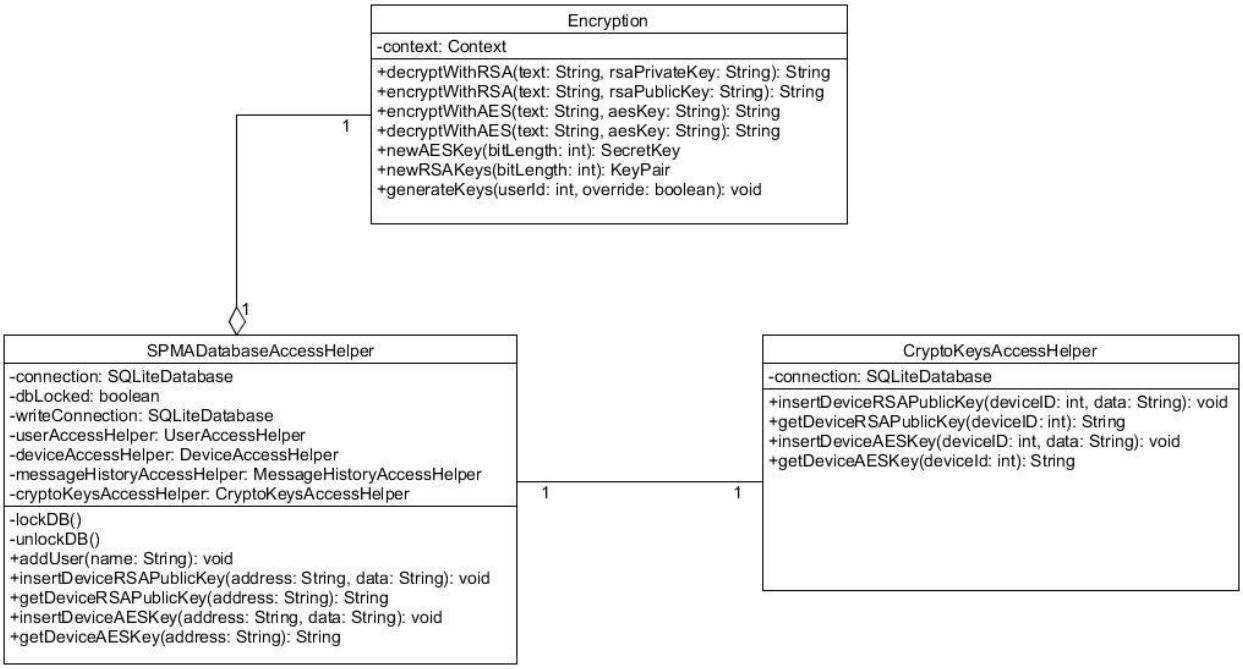

Abbildung 13: UML-Klassendiagramm

In dem UML-Klassendiagramm (Abbildung 13) ist dargestellt, dass in diesem Projekt 3 Klassen für die Verschlüsselung benötigt werden. Encryption ist die eigentliche Verschlüsselungsklasse, SPMADatabaseAccessHelper stellt die Datenbankoperationen über CryptoKeysAccessHelper zur Verfügung.

Encryption Klasse

Die Encryption Klasse greift auf das java.security und javax.crypto package zurück. Sie verfügt über Methoden zum Ver- und Entschlüsseln und zum Generieren neuer Schlüssel. Dabei unterscheiden sich die Methoden von der AES-Verschlüsselung und RSA-Verschlüsselung.

Lokale Speicherung von Daten

Lokal gespeicherte Daten, wie das Profil des Nutzers und die Nachrichtenhistorie, werden in einer Datenbank abgelegt. Als Datenbanklösung kommt SQLite zum Einsatz. SQLite ist ein lokales Datenbanksystem für einen Nutzer. Alle Datenbanktabellen und sonstige Inhalte der Datenbank werden in einer Datei auf dem Gerät des Nutzers abgelegt, eine Netzwerkverbindung ist dementsprechend nicht notwendig.

Android bringt alle für SQLite nötigen Klassen im Package „android.database.sqlite“ bereits mit. Die wichtigste Klasse dabei ist „SQLiteOpenHelper“. Diese Klasse stellt die Verbindung zur Datenbank zur Verfügung. Dazu muss „SQLiteOpenHelper“ überschrieben werden, insbesondere die Methode „onCreate()“, denn diese Methode erstellt die Datenbank.

Die Datenbank wird nach folgendem Schema erstellt:

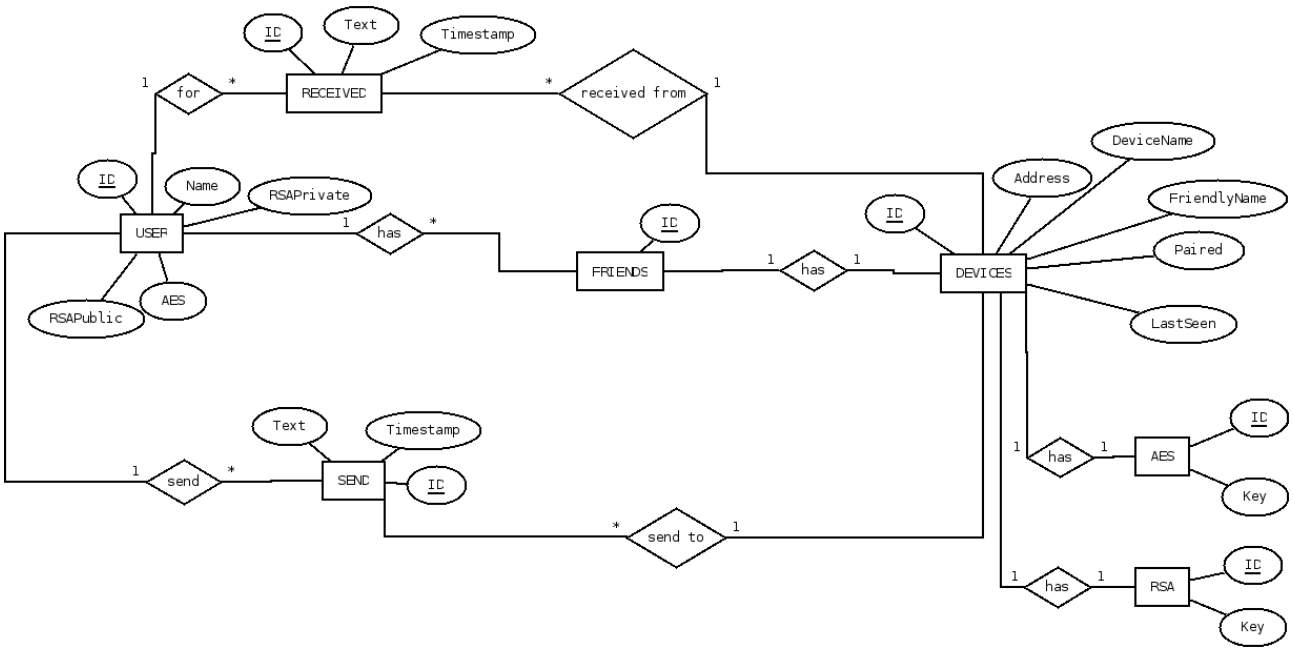

Abbildung 14: Entity-Relationship Diagramm der Datenbank

Die Tabelle „User“ enthält Daten zum lokalen Benutzer sowie dessen zur Verschlüsselung genutzte Schlüssel. Die Tabellen „Send“ und „Received“ werden genutzt, um die Nachrichtenhistorie zu speichern. „Devices“ enthält Daten aller bekannten, kompatiblen Geräte, „AES“ und „RSA“ enthalten die zur Kommunikation mit diesen Geräten benötigten Schlüssel. Die Tabelle „Friends“ enthält eine Freundesliste, die in der App angelegt werden kann.

Alle zum anlegen bzw. löschen der Tabellen benötigten Statements sind in der Datei „strings.xml“ zu finden.

Da die App in einigen Fällen, wie z.b. beim Austausch der Schlüssel nach Verbindungsaufbau, mit mehreren Threads auf die Datenbank zugreift, muss die Datenbank selbstständig vor gleichzeitigen Zugriffen. Hierzu wurde die Datenbank-Managment-Klasse (in der alle Zugriffe auf die Datenbank gesammelt sind) in einem ersten Schritt zu einem Singleton gemacht. In einem zweiten Schritt wurde ein Mechanismus implementiert, um die Datenbank bei einem Zugriff für konkurrierende

Abbildungsverzeichnis

Literaturverzeichnis

Cross und Hoeckle, 2007: Daniel Cross, Justin Hoeckle, Michael Lavine, Jason Rubin and Kevin Snow, DETECTING NON-DISCOVERABLEBLUETOOTH DEVICES, 2007
Evans und Warren, 2009: David Evans, Robert H. Warren, Anonymity properties of stored or transmitted datataken from Bluetooth scans, 2009