MiFlora Demo

Ziele

Das Produkt "Xiaomi Flora Monitor" bietet zu einem günstigen Preis (unter 15€ bei AliExpress) ein wasserdichtes Pflanzenüberwachungssystem (Bodenfeuchtigkeit, Helligkeit, Temperatur, Batteriezustand, ...) mit Bluetooth LE (low energy) Anbindung. Das System wird mit einer Knopfzelle (CR 2032) versorgt, die ca. ein Jahr halten soll.

Es ist eine Android-App zu entwickeln, die in der Lage ist, mehrere derartige Systeme zu verwalten.

- Übersicht über alle per Bluetooth erreichbare Systeme
  -- Name des Systems
  -- Macadresse
  -- Signalstärke
- Abfrage der Daten eines einzelnen Systems
  -- Firmwareversion
  -- Batteriezustand
  -- Bodenfeuchtigkeit
  -- Temperatur
  -- Helligkeit
  -- Leitfähigkeit (was ist der Unterschied zur Feuchte?)

Android-App MiFloraDemo

Verwendete Ressourcen

Sowohl mit dem Pflanzenmonitor als auch mit Kommunikation über Bluetooth LE unter Android haben sich schon einige Entwickler engagiert.

Mi Flora Pflanzenmonitor

Auf der Website https://www.open-homeautomation.com/2016/08/23/reverse-engineering-the-mi-plant-sensor/ ist ausführlich beschrieben, wie auf den Sensor zugegriffen werden kann. Die Python-Quellen sind unter https://github.com/open-homeautomation/miflora zugreifbar und sind auf den Einsatz von Raspi zugeschnitten. Die Kommunikation mit Bluetooth erfolgt dabei über Shellscripts und entsprechende CLI-Tools (gatttool).

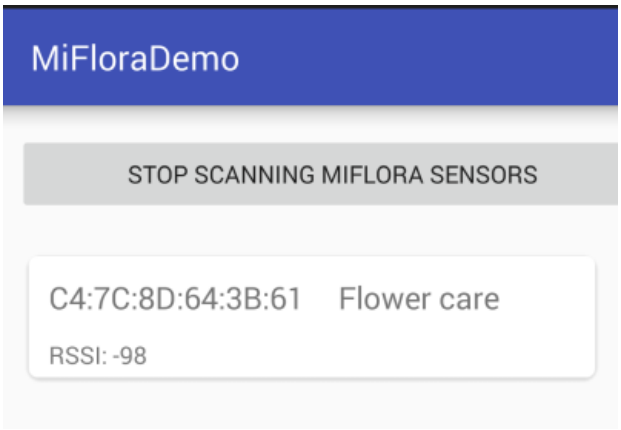
Die Bluetooth-Library RxAndroidBLE

Für den Zugriff auf das MiFlora-System unter Android wird die Bibliothek RxAndroidBLE verwendet (https://www.polidea.com/blog/RxAndroidBLE_the_most_Simple_way_to_code_Bluetooth_Low_Energy_devices/).

Die Bibliothek wird mittels Gradle (compile "com.polidea.rxandroidble:rxandroidble:1.0.1") eingebunden und verlangt als minimale SDK-Version Android 4.3 (API 18).

Übersichtsliste der erreichbaren MiFlora-Systeme

Alle erreichbaren Bluetooth Low Energy Geräte innerhalb der Funkreichweite sind als Liste auszugeben. Neben der MAC-Adresse und dem Namen des Systems wird auch die Feldstärke ausgegeben und ständig aktualisiert.



Observer beobachtet Zustand von BLE-Devices innerhalb der Bluetoothreichweite

Der BLE-Controller verwaltet alle Bluetooth-relevanten Funktionalitäten

```
    /**
     * Der Scanner für BLE-Devices wird gestartet. Über einen
     * Callback (Observer) werden Änderungen gemeldet, die dann
     * im Repository landen (add new device, update device)
     * Das Repository wird von der GUI beobachtet und über einen
     * Handler aktualisiert.
     */
    public void startScanningBleDevices() {
        subscription = rxBleClient.scanBleDevices()
                .subscribe(new Observer<RxBleScanResult>() {
                    @Override
                    public void onCompleted() {
                        Log.d(LOG_TAG, "scanBleDevices, onCompleted ");
                    }

                    @Override
                    public void onError(Throwable e) {
                        Log.d(LOG_TAG, "scanBleDevices, onError: " + e.getMessage());
                    }

                    @Override
                    public void onNext(RxBleScanResult rxBleScanResult) {
                        Log.d(LOG_TAG, "scanBleDevices, onNext: " + rxBleScanResult.toString());
                        MiFloraSensor sensor = new MiFloraSensor(
                                rxBleScanResult.getBleDevice().getMacAddress(),
                                rxBleScanResult.getBleDevice().getName());
                        sensor.setRssi(rxBleScanResult.getRssi());
                        Repository.getInstance().addOrUpdateSensor(sensor);
                    }
                });
    }
```

Repository verwaltet die Liste der BLE-Devices

Änderungen an den BLE-Devices führen zur Aktualisierung der GUIs (Liste, Detailansicht).

```
public class Repository extends Observable {

  List<MiFloraSensor> miFloraSensors;

 /**
  * Liste der Results zurückgeben
  * @return Liste der Results
     */
 public List<MiFloraSensor> getMiFloraSensors() {
  return miFloraSensors;
 }

 /**
  * Sensor entsprechend Index zurückliefern
  * @param index Position in der Liste
```

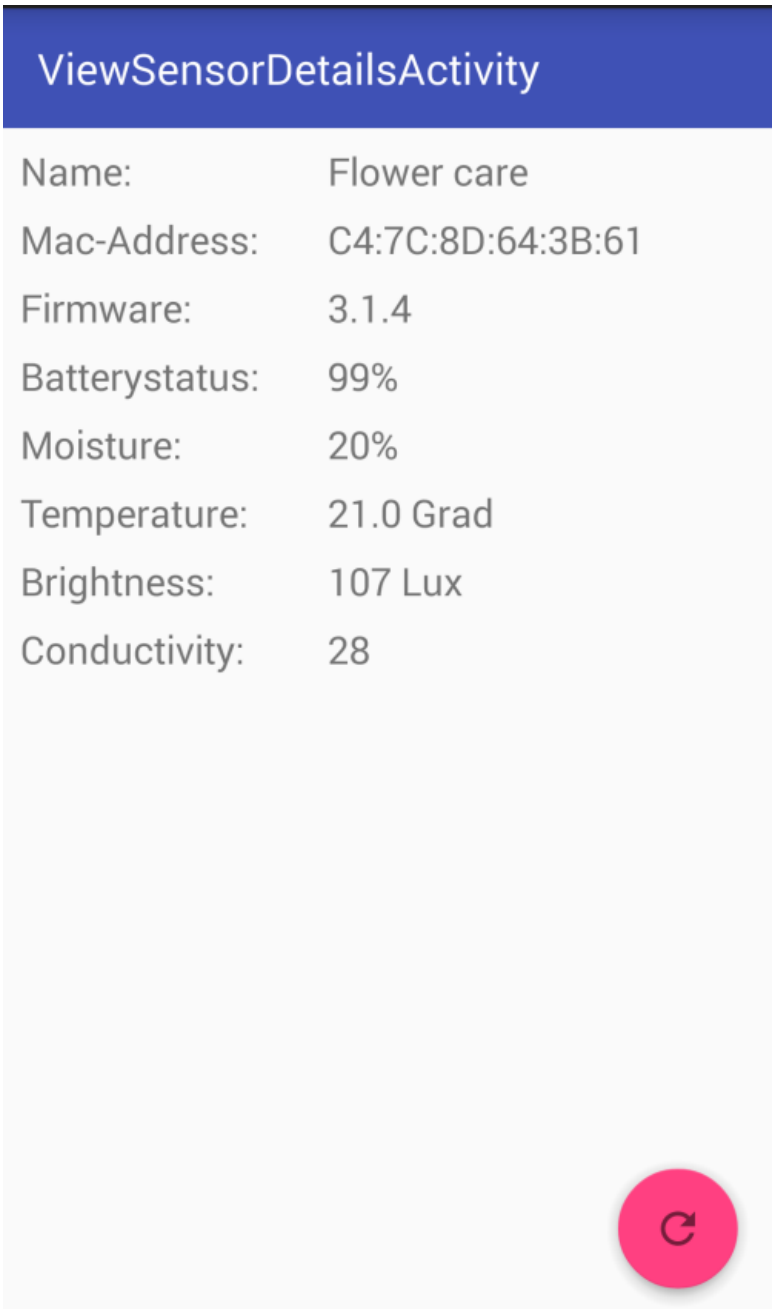
```
  * @return Result
     */
 public MiFloraSensor getMiFloraSensor(int index) {
  if(index >= miFloraSensors.size() || index < 0){
   return null;
  }
  return miFloraSensors.get(index);
 }

 /**
  * Sensordaten haben sich geändert. Ist der Sensor neu (MAC-Adresse unbekannt)
  * wird er angelegt. Sonst werden die änderbaren Daten (RSSI = Feldstärke)
  * übrenommen und die angemeldeten Observer werden verständigt.
  * @param miFloraSensor
  */
 public void addOrUpdateSensor(MiFloraSensor miFloraSensor) {
  for(MiFloraSensor sensor : miFloraSensors){
   if (sensor.getMacAddress().equalsIgnoreCase(miFloraSensor.getMacAddress())){
    sensor.setRssi(miFloraSensor.getRssi());
    setChanged();
    notifyObservers();
    return;
   }
  }
  miFloraSensors.add(miFloraSensor);
  setChanged(); // Observable
  this.notifyObservers();
  Log.d(LOG_TAG, "addOrUpdateSensor(), Observer verstaendigt, Datensaetze: "+ miFloraSensors.size());
 }

 /**
  * Sensordaten wurden direkt geändert. UI muss neu ausgegeben werden.
  */
 public void updateUI() {
  setChanged(); // Observable
  this.notifyObservers();
 }
}
```

Ausgabe der Liste in der MainActivity

Die MainActivity beobachtet das Repository und startet über einen Handler die Aktualisierung der GUI.
Über einen ToggleButton kann der Scanvorgang ein- und ausgeschaltet werden.
Zur Ausgabe wird eine RecyclerView mit eigenem Layout für die Row verwendet.
Bei Auswahl eines Sensors wird die ViewSensorDetailsActivity aufgerufen.

Anzeige der Detaildaten eines Devices

Die ViewDetailsActivity gibt alle Daten des Sensors aus. Über einen FAB kann das Einlesen der Daten über Bluetooth neu gestartet werden.



Lesen der Detaildaten eines Sensors über Bluetooth Low Energy (BLE)

Durch die Verwendung der Bluetooth-Library RxAndroidBLE vereinfacht sich das Einlesen von BLE-Devices enorm. Wie das Rx schon andeutet verwendet die Library intensiv "rective" Technologien mit sehr vielen asynchronen "Subscribern". Die Library setzt auch masiv auf LambdaExpressions und damit auf Java 8. Da das gesamte Homeautomationprojekt derzeit noch auf Java 7 basiert, entsteht etwas unübersichtlicher Code mit mehreren verschachtelten anonymen Objekten.

```
    /**
     * Daten vom MiFlora-Sensor laden.
     * Bei BLE werden einzelne Kanäle (Characteristics) verwendet, die für Read, Write oder/und
     * Notify eingesetzt werden können. Die Charakteristiken werden über UUIDs adressiert.
     * MiFlora sendet bei Abfrage der Charakteristik 0x38 Stammdaten (Firmwareversion, Batteriezustand)
     * Die eigentlichen Daten werden über Charakteristik 0x33 geliefert. Dazu muss aber zuerst auf
     * die Charakteristik 0x35 ein Steuercode geschrieben werden.
     * Zuerst wird asynchron eine Connection erstellt. Auf die Rückmeldung der Connection wird die
     * Firmwareversion und der Batteriezustand abgefragt. Ist diese Kommunikation abgeschlossen, wird
     * das Einlesen der eigentlichen Sensorwerte gestartet.
     *
     * @param miFloraSensor
     */
    public void loadDataFromSensor(final MiFloraSensor miFloraSensor) {
        final RxBleDevice rxBleDevice = rxBleClient.getBleDevice(miFloraSensor.getMacAddress());
        subscription = rxBleDevice.establishConnection(MyApplication.getInstance(), false) // <-- autoConnect flag
                .subscribe(new Observer<RxBleConnection>() {
                               @Override
```

```
                        public void onCompleted() {
                            Log.d(LOG_TAG, "loadDataFromSensor, subscription completed");
                        }

                        @Override
                        public void onError(Throwable e) {
                            Log.d(LOG_TAG, "loadDataFromSensor, subscription error: "+e.getMessage());
                        }

                        @Override
                        public void onNext(final RxBleConnection rxBleConnection) {
                            Log.d(LOG_TAG, "loadDataFromSensor, subscription onNext: "+rxBleConnection.toString());
                            rxBleConnection.readCharacteristic(UUID_FIRMWARE_BATTERY_LEVEL)
                                    .subscribe(new Observer<Object>() {
                                        @Override
                                        public void onNext(Object o) {
                                            Log.d(LOG_TAG, "loadDataFromSensor, load firmware next: "+o.toString());
                                            byte[] bytes = (byte[]) o;
                                            miFloraSensor.setBatteryLevel(bytes[0]);
                                            StringBuilder text = new StringBuilder();
                                            for(int i=2; i<bytes.length;i++){
                                                text.append((char) bytes[i]);
                                            }
                                            miFloraSensor.setFirmwareVersion(text.toString());
                                        }

                                        @Override
                                        public void onError(Throwable e) {
                                            Log.d(LOG_TAG, "loadDataFromSensor, load firmware error: "+e.getMessage());
                                        }

                                        @Override
                                        public void onCompleted() {
                                            Log.d(LOG_TAG, "loadDataFromSensor, load firmware completed");
                                            loadSensorValues(miFloraSensor, rxBleConnection);
                                        }
                                    });
                        }
                    }

            );
    // When done... unsubscribe and forget about connection teardown :)
    //subscription.unsubscribe();
}

public static int toUnsignedInt(byte x) {
    return ((int) x) & 0xff;
}

/***
 * Das Einlesen der eigentlichen Sensorwerte erfolgt wieder asynchron, indem zuerst
 * die Steuerbytes auf Charakteristik 0x35 geschrieben werden und wenn das abgeschlossen ist,
 * die Daten über Charakteristik 0x33 ausgelesen werden.
 * @param miFloraSensor
 * @param connection
 */
private void loadSensorValues(final MiFloraSensor miFloraSensor, final RxBleConnection connection) {
    Log.d(LOG_TAG, "loadSensorValues, start");
    String ctrlText = "A01F";
    byte[] ctlBytes = HexString.hexToBytes(ctrlText);
    connection.writeCharacteristic(UUID_WRITE_CTRL, ctlBytes).subscribe(new Observer<byte[]>() {
        @Override
        public void onCompleted() {
            Log.d(LOG_TAG, "writeCharacteristic, onCompleted: ");
            connection.readCharacteristic(UUID_MEASUREMENTS)
                    .subscribe(new Observer<Object>() {
                        @Override
                        public void onNext(Object o) {
                            Log.d(LOG_TAG, "loadSensorValues, next: "+o.toString());
                            byte[] dataBytes = (byte[]) o;
                            miFloraSensor.setTemperature((toUnsignedInt(dataBytes[1])*256.0+toUnsignedInt(dataBytes[0]))/1
```

```
                            miFloraSensor.setMoisture(toUnsignedInt(dataBytes[7]));
                            miFloraSensor.setBrightness(toUnsignedInt(dataBytes[4])*256+toUnsignedInt(dataBytes[3]));
                            miFloraSensor.setConductivity(toUnsignedInt(dataBytes[9])*256+dataBytes[8]);
                            Repository.getInstance().updateUI();
                            subscription.unsubscribe();
                        }

                        @Override
                        public void onError(Throwable e) {
                            Log.d(LOG_TAG, "loadSensorValues, error: "+e.getMessage());
                        }
                        @Override
                        public void onCompleted() {
                            Log.d(LOG_TAG, "loadSensorValues, load completed");
                        }
                    });
        }

        @Override
        public void onError(Throwable e) {
            Log.d(LOG_TAG, "writeCharacteristic, error: "+e.getMessage());
        }

        @Override
        public void onNext(byte[] bytes) {
            Log.d(LOG_TAG, "writeCharacteristic, onNext: "+bytes.toString());
        }
    });
}
```

Verwalten der Detaildaten über das Repository

Die Bluetoothroutinen ändern die Daten des ausgewählten Sensors direkt. Nach Abschluss der Änderungen werden die Observer (DetailsView) verständigt, die die Anzeige aktualisieren.