



Lab on apps development for tablets, smartphones and smartwatches

Week 8: Bluetooth Low Energy

Giovanni Ansaloni

Dimitra Tatli, Riselda Kodra, Yuxuan Wang
Qunyou Liu, Amirhossein Shahbazinia, Christodoulos Kechris

School of Engineering (STI) – Institute of Electrical and Micro Engineering (IEM)

Midterm next week

- We will (briefly) discuss the Midterm solution and grades on **November 25th**
- Also, I will provide more details on the final exam
- ... and also about the *Android meets IoT* meet-up and competition

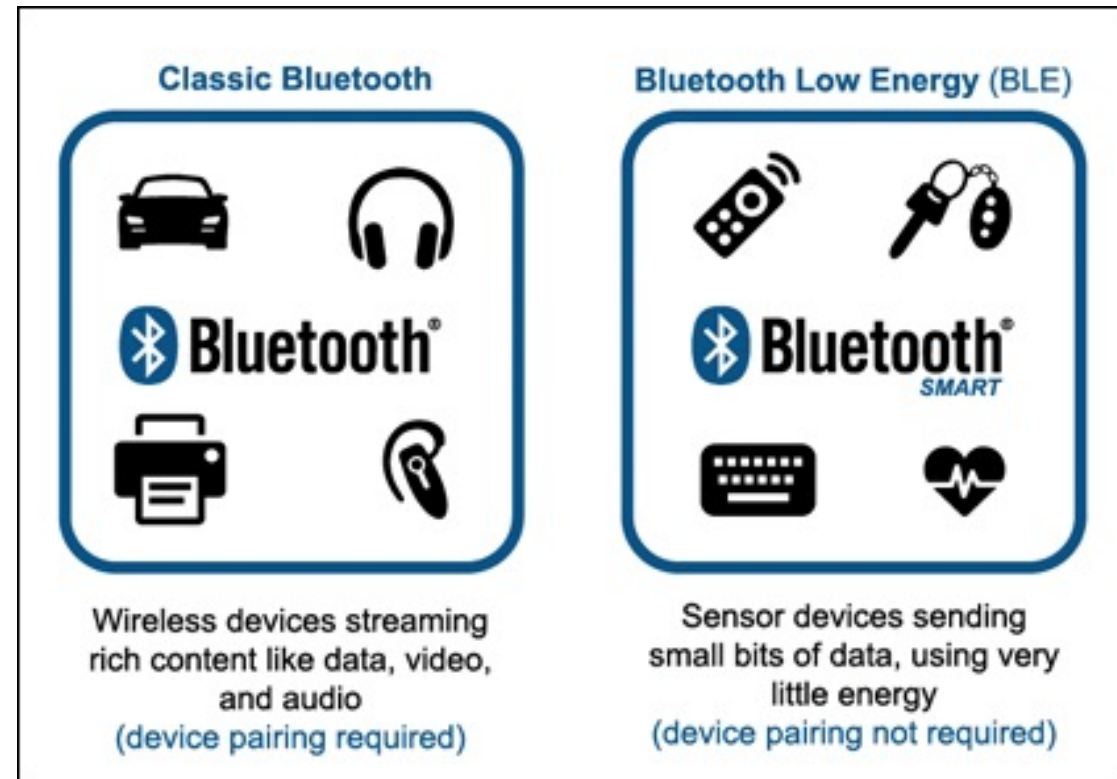


Bluetooth Low Energy (BLE)

- BLE provides lower power consumption with respect to classic Bluetooth
 - small bursts of data exchange

- Apps can communicate with BLE devices that have low power requirement
 - proximity sensors, fitness/medical devices, ...
 - BLE devices can last for weeks/years

- The BLE API allows to:
 - **Discover** devices
 - **Query** for services
 - **Send/receive** data





- Central vs. peripheral
 - Roles in establishing a BLE connections
 - **Central** device scans, looking for advertisement
 - **Peripheral** device makes advertises device capabilities

- Server vs. client
 - How two connected devices talk to each
 - **Client** asks for data
 - **Server** provides data

- In today's lab:
 - Tablet → central device, client
 - HR strap → peripheral device, server

BLE Roles





Generic Attribute Profile (GATT)

- Specification for sending and receiving short pieces of data
 - known as "attributes" over a BLE link

- GATT is built on top of the Attribute Protocol (ATT)
 - ATT is optimized to run on BLE devices
 - each attribute only uses few bytes
 - attributes are identified by a Universally Unique Identifier (UUID)
 - complete documentation: <https://www.bluetooth.com/specifications/specs/>

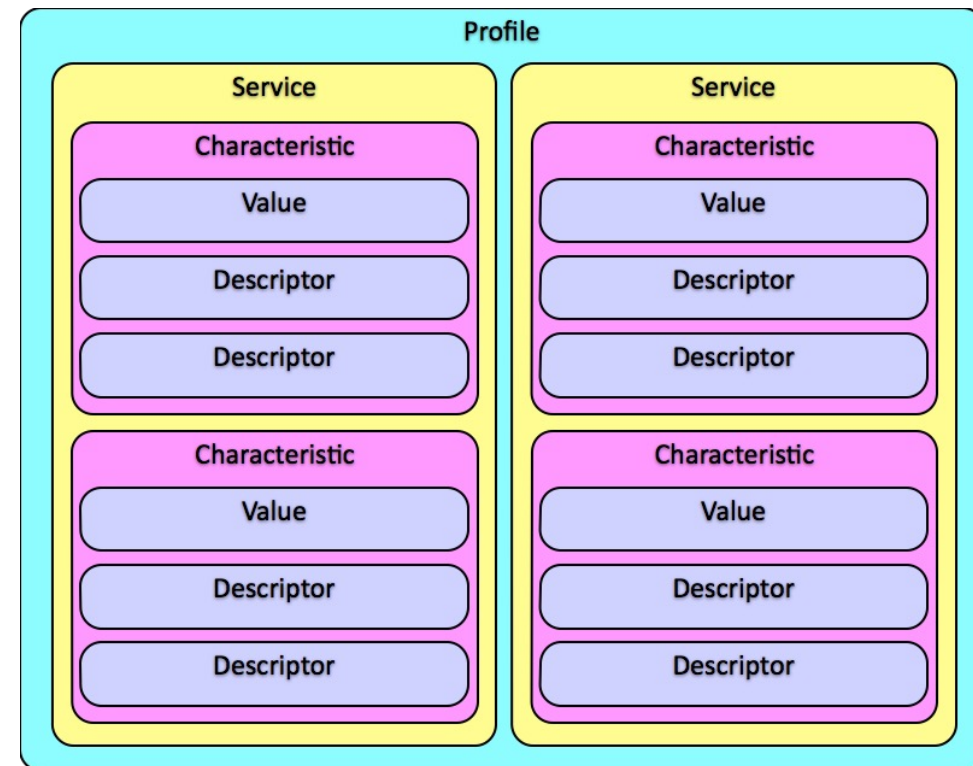
- ATTs
 - services: Battery, Device information, Heart rate, ...
 - characteristics: Battery level, System ID, Heart rate measurement, ...

Characteristics and Services

- Service
 - Collection of characteristics
 - e.g., the “Heart Rate Monitor” service includes the “Heart Rate Measurement” characteristic

- Characteristic
 - Contains a single value and several descriptors that describe the characteristic's value

- Descriptor
 - Describes a characteristic’s value, e.g.:
 - human-readable description
 - acceptable range for a characteristic's value
 - unit of measure of a characteristic's value
 - ...



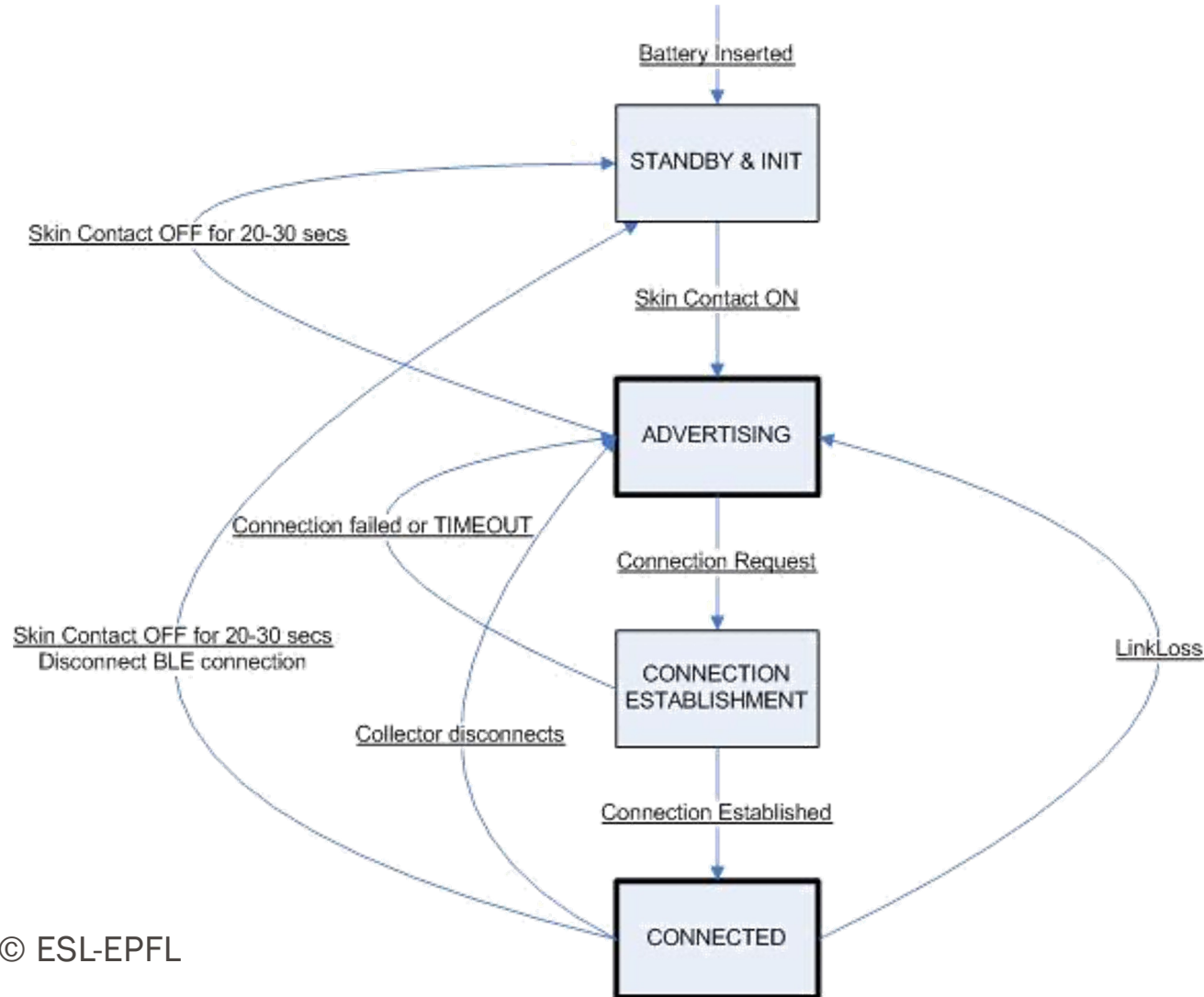
Geonaute HRM – Services

- Heart rate service
 - Heart rate measurement characteristic
 - Value: “Heart rate measurement: 61bpm, Contact is Detected, RR Interval 983.04ms”
 - Descriptor: “Notifications enabled”
 - Body Sensor Location characteristic
 - Value: “Chest”

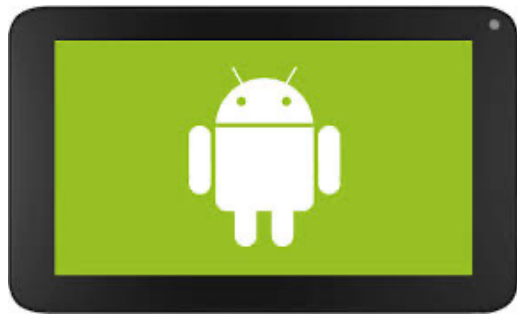
- Device information service
 - Device information characteristic
 - System ID, Model Number, Serial Number, Firmware Revision, Hardware Revision, Software Revision, Manufacturer Name



Geonaute HRM- BLE HR state machine



Ok, but in practice?



- Scans
- Discovers
- Connects
- Reads/writes characteristics

- Advertises services&characteristics
- Accepts connections



Setting up BLE: app manifest

- Bluetooth permissions
- Bluetooth Admin permissions
 - to scan for BLE devices

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

- Location permission
 - Because discoverable devices might reveal information on the user location,

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```



Setting up BLE: app manifest

- App is available to BLE-capable devices only:

```
<uses-feature android:name="android.hardware.bluetooth_le"  
             android:required="true"/>
```

- To make BLE optional, set `required="false"` in the manifest, and check if BLE availability is supported in the host device

```
if(packageManager.hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE)) {  
    ...  
}
```



Setting Up BLE

- Grab the **BluetoothAdapter**

- bridge towards the BLE API

```
val bluetoothAdapter: BluetoothAdapter? = BluetoothAdapter.getDefaultAdapter()
```

- Enable Bluetooth:

- ask Android to ask user → implicit Intent
 - Activity.RESULT_OK if the user turned Bluetooth on, Activity.RESULT_CANCELED otherwise

```
private fun promptEnableBluetooth() {  
    val enableBtIntent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)  
    resultLauncher.launch(enableBtIntent)  
}
```

```
val resultLauncher =  
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->  
        if (result.resultCode == AppCompatActivity.RESULT_OK) {  
            ...  
        }  
    }  
}
```

Finding BLE Devices

- Use a BluetoothLeScanner
 - startScan() method
 - callback of class ScanCallback invoked when new BLE device found

- Scanning is battery-intensive:
 - Set a time limit on your scan

- Callback
 - do something when device found
 - overrides onScanResult()

```

private val bluetoothLeScanner: BluetoothLeScanner? =
    bluetoothAdapter?.bluetoothLeScanner
private val handler = Handler()

// Stops scanning after 10 seconds.
private val SCAN_PERIOD: Long = 10000.

private fun scanLeDevice() {
    if (bluetoothLeScanner != null) {
        bluetoothLeScanner.startScan(leScanCallback)
    }

    handler.postDelayed({
        if (bluetoothLeScanner != null) {
            bluetoothLeScanner.stopScan(leScanCallback)
        } }, SCAN_PERIOD
    )
}

var myBluetoothDevice: BluetoothDevice? = null

private val leScanCallback: ScanCallback =
    object : ScanCallback() {
        override fun onScanResult(callbackType: Int,
                                   result: ScanResult) {
            super.onScanResult(callbackType, result)
            myBluetoothDevice = result.device
        }
    }

```





Connecting to a BLE device

- Connect to the device Gatt Server

```
myBluetoothGatt =  
    myBluetoothDevice.connectGatt(  
        requireContext(),  
        false,  
        myGattCallback)
```

- the callback defines responses to BLE events
 - device connected
 - device service discovered
 - characteristic read/changed

```
private val myGattCallback = object : BluetoothGattCallback() {  
    override fun onConnectionStateChange(gatt: BluetoothGatt,  
        ...) {  
        if (newState == BluetoothProfile.STATE_CONNECTED) {  
            gatt.discoverServices()  
            ...  
        }  
    }  
  
    override fun onServicesDiscovered(gatt: BluetoothGatt,  
        ...) {  
        ...  
    }  
  
    override fun onCharacteristicChanged(gatt: BluetoothGatt,  
        characteristic: BluetoothGattCharacteristic) {  
        ...  
    }  
  
    override fun onCharacteristicRead(gatt: BluetoothGatt,  
        characteristic: BluetoothGattCharacteristic,  
        ...) {  
        ...  
    }  
}
```

Reading BLE attributes

- Requests a read of a characteristic value to the BLE device GATT server

```
val hrCharacteristic: BluetoothGattCharacteristic =  
myBluetoothGatt.getService(HEART_RATE_SERVICE_UUID)  
    .getCharacteristic(HEART_RATE_MEASUREMENT_CHAR_UUID)
```

```
myBluetoothGatt.readCharacteristic(hrCharacteristic)
```

- The device response triggers the onCharacteristicRead() callback



Enabling BLE data notifications

- If we want to be notified each time a characteristic is updated
 - e.g: heart rate acquired from BLE sensor
 - usually, set in onServiceDiscovered()

1. Enable notifications for the characteristic on client (tablet)

```
gatt.setCharacteristicNotification(characteristic, enabled)
```

2. Enable stream of data from server (HR sensor)

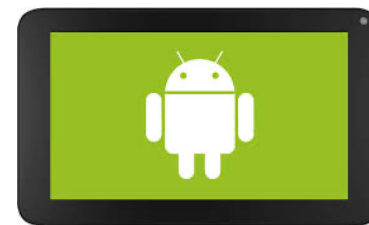
```
val descriptor : BluetoothGattDescriptor =
    characteristic.getDescriptor(HEART_RATE_MEASUREMENT_CHAR_UUID)
    descriptor.value = BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE
    gatt.writeDescriptor(descriptor)
```

- Characteristic updates trigger onCharacteristicChanged()



Today's Lab

- Interfacing with the HR belt using BLE
 - scanning
 - service discovery
 - reading/writing characteristics
- Letting user choose the source of HR data
 - smartwatch or belt





Where are we?

CLOUD

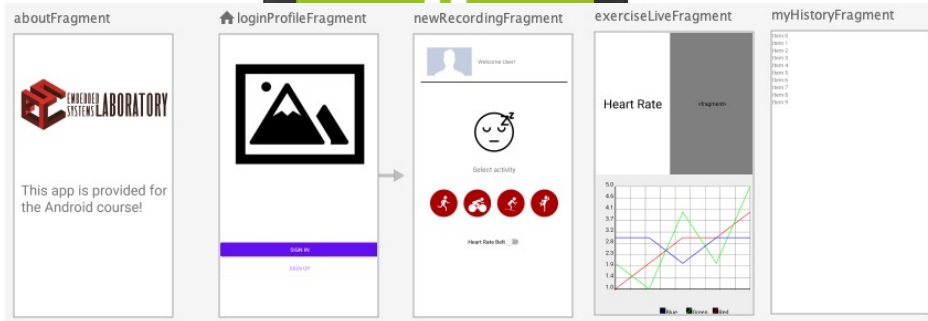


Firestore

```
project-sports-tracker
├── profiles
│   └── -LPRawc-6u20SsS06zNe
│       ├── height: 173
│       ├── password: "YouMustNotStorePlainTextPasswords"
│       ├── username: "Rose"
│       └── weight: 61.29999923706055
```



TABLET



WEARABLES



Questions?

