

Projekt Internet-Technologien -

Gruppe 3

Max Golubew, Marco Buchholz, Florian Winzek

Advanced Internet Technologies
Institut für Telematik
Universität zu Lübeck

29. Mai 2017

Inhaltsverzeichnis

1	Erster Meilenstein	4
1.1	Aufgabe 1 - LED	4
1.1.1	Vorbereitung	4
1.1.2	Durchführung	4
1.2	Aufgabe 2 - Digitaler Wert des Fotowiderstands (LDR)	5
1.2.1	Vorbereitung	5
1.2.2	Implementierung	6
1.3	Aufgabe 1.3 - Analoger Wert des Fotowiderstands	7
1.3.1	Vorbereitung	7
2	Zweiter Meilenstein	10
2.1	Aufgabe 2.1 - LDR-Sensorwert mittels CoAP an SSP senden	10
2.1.1	Vorbereitung	10
2.1.2	Java-Programm	11
2.1.3	Java-Anwendung	13
3	Dritter Meilenstein	16
4	Vierter Meilenstein	17

In diesem Projekt geht es darum verteilte Anwendungen mithilfe von aktuellen Internet-Technologien zu entwickeln. Dazu wurden 4 Meilensteine festgelegt, die im Rahmen dieses Dokumentes ausgearbeitet werden. Die notwendigen Komponenten und Technologien zur Bearbeitung der Meilensteine sind in dem Handbuch [1] des Moduls ausführlich beschrieben.

1 Erster Meilenstein

Das Ziel des ersten Abschnittes ist es, den Raspberry Pi einzurichten, Grundlagen der Elektrotechnik aufzufrischen und erste kleine Anwendungen/Schaltungen zu entwickeln, um den GPIO-Umgang kennenzulernen.

1.1 Aufgabe 1 - LED

1.1.1 Vorbereitung

Die erste Aufgabe besteht darin eine LED mithilfe des Raspberry Pi (im Folgenden: Pi) zum Leuchten zu bringen. Dazu benötigt man:

- eine LED
- einen Widerstand
- zwei Jumper-Kabel (male - male)
- ein Breadboard

Damit der Pi nicht zerstört wird, muss der korrekte Widerstandswert berechnet werden. Dazu wird das Ohmsche Gesetz herangezogen: $U = R * I$, mit der elektrischen Spannung U, der Stromstärke I und dem elektrischen Widerstand R. Die Spannung und Stromstärke entnimmt man dem Datenblatt der LED, dann wird die Formel nach R umgestellt: $R = \frac{U}{I}$

$$\text{Dies ergibt: } R = \frac{(3,3V - 2,5V)}{16mA} = 50\Omega$$

Da es keinen 50Ω Widerstand gibt, wird der nächstgrößere verwendet(100Ω).

Nachdem alle Komponenten berechnet wurden, kann nun die Schaltung auf dem Breadboard zusammengesteckt werden. Ein Bild von der Schaltung kann aus der Abbildung 1.1 entnommen werden.

1.1.2 Durchführung

Mithilfe der WiringPi Bibliothek, welche auf dem Pi installiert wurde, kann nun die LED zum Leuchten gebracht werden. Dazu benötigt man folgende Code- Zeilen, die einfach über die Kommandozeile des Pis abgeschickt werden können:

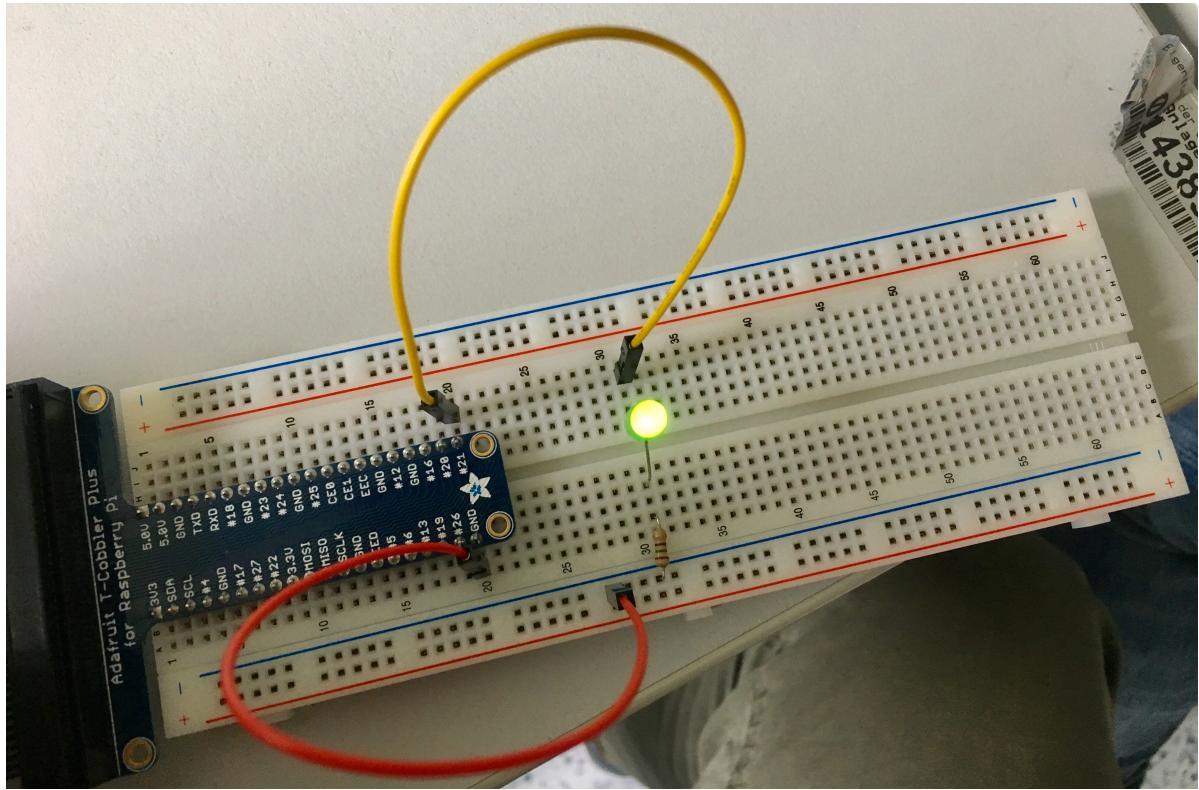


Abbildung 1.1: Schaltung der Aufgabe 1

```
$ gpio mode 26 out
$ gpio write 26 1
$ gpio write 26 0
```

Mit dem mode- Befehl wird der GPIO- Pin als Ausgang gesetzt und kann dann beschrieben werden. Schreibt man auf den Ausgang eine 1, leuchtet die LED, schreibt man eine 0, so geht sie wieder aus.

1.2 Aufgabe 2 - Digitaler Wert des Fotowiderstands (LDR)

1.2.1 Vorbereitung

In dieser Aufgabe soll ein LDR (GL5516) mit einem digitalen GPIO des Pis verbunden und ausgelesen werden. Danach soll die LED aus Abschnitt 1.1 so mit dem LDR geschaltet werden, dass sie leuchtet, sobald der LDR abgedunkelt wird und beim Aufhellen wieder ausgeschaltet wird.

Dazu muss ein Vorwiderstand für den LDR errechnet werden. Anhand des Datenblatts [2] erhält man die Widerstandswerte des LDRs, wenn es dunkel ist (100 kOhm) und wenn

es hell ist(5 kOhm). Nun betrachtet man den Aufbau eines Spannungsteilers und kann mittels der Gleichung 1.1 den Vorwiderstand berechnen.

$$I = \frac{U_{ges}}{R_{ges}} = \frac{U_1 + U_2}{R_1 + R_2} = \frac{U_1}{R_1} = \frac{U_2}{R_2} \quad (1.1)$$

Als Hinweis: Für Spannungen $U < 0.8V$ erkennt der Pi ein Low- Pegel (logische 0) und für $U > 2.0V$ einen High- Pegel (logische 1). Daher ergeben sich nun die folgenden zwei Gleichungssysteme. Dunkel:

$$\frac{3,3V}{R_{ges}} = \frac{1,3V+2,0V}{R_1+10k\Omega} = \frac{1,3V}{R_1} = \frac{2,0V}{10k\Omega}$$

Stellt man nun die letzten beiden Gleichungen nach R_1 um, erhält man:

$$R_1 = \frac{1,3V}{2,0V} * 10k\Omega = 6.500\Omega$$

Hell:

$$\frac{3,3V}{R_{ges}} = \frac{2,5V+0,8V}{R_1+1k\Omega} = \frac{2,5V}{R_1} = \frac{0,8V}{1k\Omega}$$

Stellt man auch hier die letzten beiden Gleichungen nach R_1 um, so ergibt das:

$$R_1 = \frac{2,5V}{R_1} = \frac{0,8V}{1k\Omega} = 3,125\Omega$$

Der Widerstand liegt zwischen $3,125\Omega < R_1 < 6.500\Omega$.

Für diesen Versuch wählen wir also die logische Mitte, damit ist $R_1 = 4.700\Omega$.

Demnach haben wir die Schaltung auf dem Breadboard zusammengebaut. Diese kann in der nachfolgenden Abbildung 1.2 entnommen werden.

Mit der wiringPi Bibliothek kann nun der Widerstandswert des LDR ausgelesen werden.

1.2.2 Implementierung

Als nächstes soll ein Programm entwickelt werden, welches das automatisierte An- bzw. Ausschalten einer LED in Abhängigkeit des analogen LDR- Wertes übernimmt. Hierfür wird die Java- Bibliothek *Pi4J* [3] eingebunden, um die Kommunikation mit den GPIOs des Pis zu realisieren.

In dem nachfolgenden Listing Listing 1.1 wird das Setzen der LED abhängig vom LDR- Wert dargestellt.

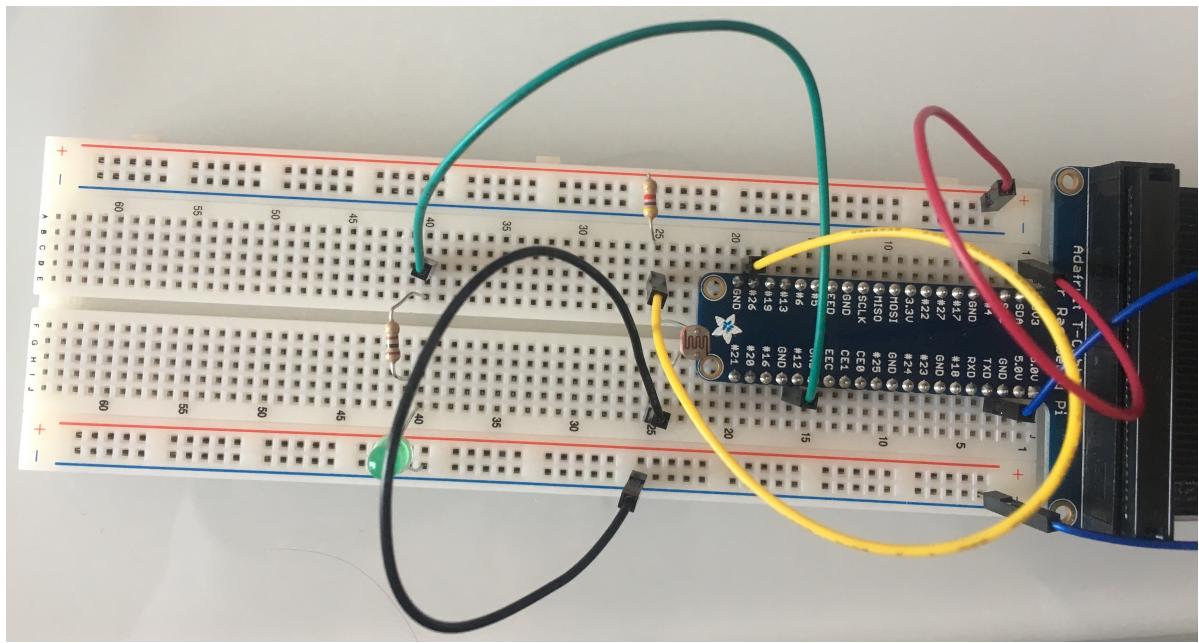


Abbildung 1.2: Schaltung der Aufgabe 2

Listing 1.1: Statusaktualisierung der LED in Abhängigkeit des LDR- Wertes

```
// Final bounding value for the state of the outputpin
private static final int BOUNDING = 70;

// Provision gpio pin #12 as an output pin and turn off
private final GpioPinDigitalOutput outputPin;

/**
 * Update the state of the outputpin
 *
 * @param value The LUX- value of the ldr
 */
public void updateOutputPin(double value) {
    outputPin.setState(value >= BOUNDING);
}
```

1.3 Aufgabe 1.3 - Analoger Wert des Fotowiderstands

1.3.1 Vorbereitung

In dem letzten Aufgabenteil des 1. Meilensteins soll der analoge Wert des LDR mittels eines Arduinos eingelesen werden. Um den analogen Wert des LDR zu ermitteln, muss eine Leitung an einer der analogen Eingänge (bei uns A0) des Arduinos angeschlossen

werden. Für diese Schaltung ist zu beachten, dass die Schaltungen der LED und des LDR getrennt sind. Ansonsten werden vom Arduino verfälschte Werte gemessen. Der komplette Aufbau ist in Abbildung 1.3 zu sehen.

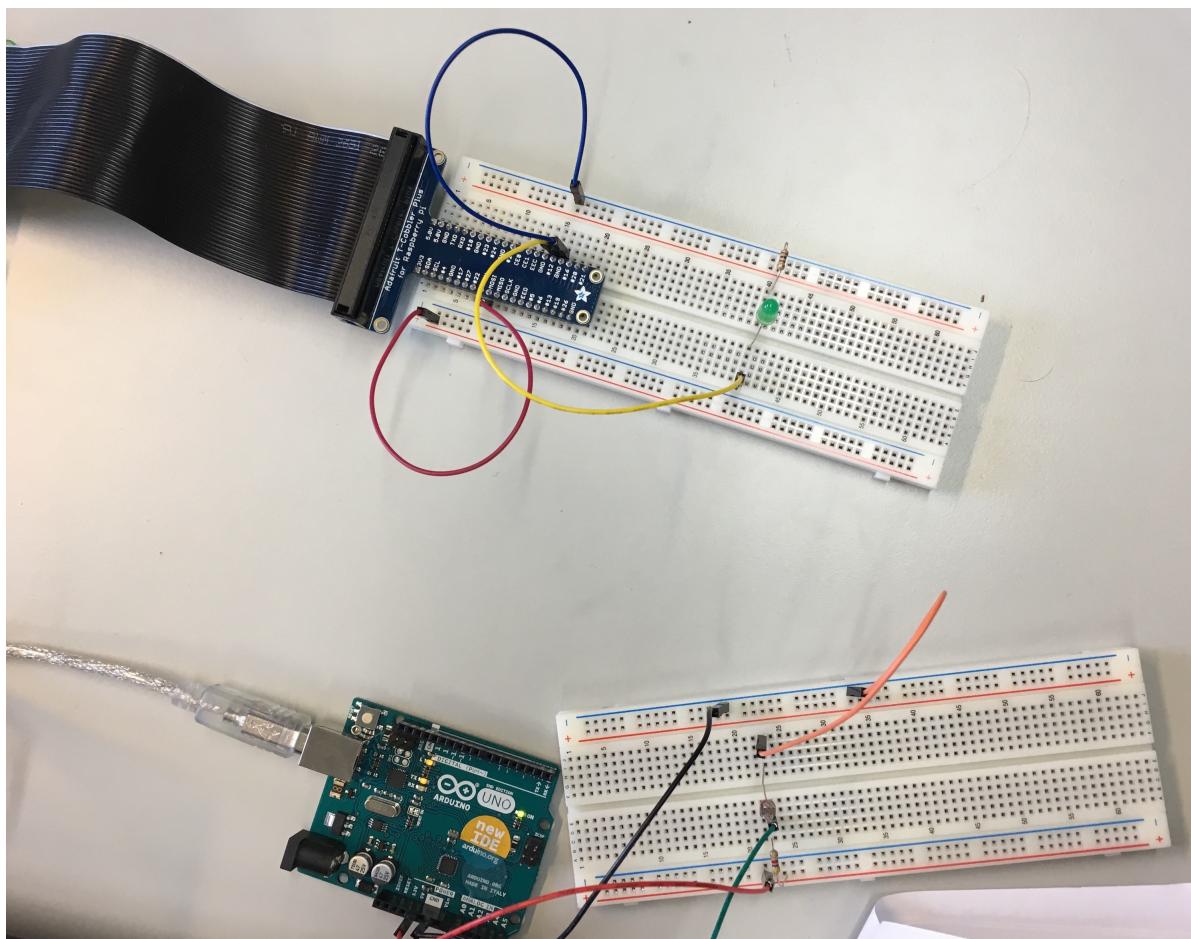


Abbildung 1.3: Schaltung der Aufgabe 3

Nun ist es das Ziel mit dem Arduino einen Wert einzulesen und diesen über die Serielle Schnittstelle an der Pi zu senden. Um den analogen Wert interpretieren zu können, nutzen wir den integrierten A/D-Wandler des Arduinos. Dieser besitzt eine Auflösung von 10 Bit und liefert somit Werte zwischen 0 und 1023 ($2^{10} - 1$). Dieser Wert kann nun an den Pi geschickt werden. Allerdings hängen wir noch ein Trennzeichen (#) an, um mit dem Pi das Ende einer Nachricht erkennen zu können.

Im Anschluss werden mit diesen Werten der korrespondierende Lux- Wert errechnet, welcher für die weiteren Meilensteine benutzt werden soll. Die Berechnung wird von einem Java- Programm durchgeführt, welches auf dem Pi ausgeführt wird und zusätzlich eine LED anschaltet, sobald der LDR abgedunkelt wird und beim Aufhellen wieder ausschaltet.

Für die Lux Umrechnung werden folgende drei Formeln benötigt:

$$U_{LDR} = \frac{analogValue * U_{IN}}{1023}$$

$$R_{LDR} = \frac{R_V * U_{LDR}}{U_{IN} - U_{LDR}}$$

$$E_v = R_{LDR}^{-1.31022} * 210.91430$$

analogValue ist der empfangene Wert, welcher vom Arduino gemessen wird. Da der Arduino eine Betriebsspannung von 5V besitzt, muss für $U_{IN} = 5V$ gewählt werden. Angenommen *analogValue* = 300, dann ist

$$U_{LDR} = \frac{300 * 5V}{1023} \approx 1.466V$$

Für R_V muss nun der berechnete Widerstand ($4,7k\Omega$) aus Abschnitt 1.2 genommen werden. Daraus ergibt sich nun die Gleichung

$$R_{LDR} = \frac{4.7k\Omega * 1.466V}{5V - 1.466V} \approx 1.950k\Omega$$

Somit lässt sich nur der Wert in Lux berechnen:

$$E_v = 1.950k\Omega^{-1.31022} * 210.91430 \approx 87.91$$

2 Zweiter Meilenstein

Das Ziel des 2. Meilensteins ist es, den in Meilenstein 1 errechneten Lux-Wert des LDR an einen Smart-Service Proxy(SSP) zu senden. Dies soll mittels des CoAP Protokolls realisiert werden unter Zuhilfenahme einer vorhanden Ontologie, sowie das RDF-Schema. Des weiteren soll eine Java-Anwendung sämtliche gespeicherten Lux-Werte vom SSP einlesen und eine LED schalten abhängig der erhaltenen Werte.

2.1 Aufgabe 2.1 - LDR-Sensorwert mittels CoAP an SSP senden

2.1.1 Vorbereitung

In der vorbereitenden Aufgabe sollte das nCoAP-Repository installiert werden, sowie die ssp-example-Application, damit ein erster Testversuch, Daten von einem Client an den SSP zu schicken, realisiert werden kann. Dazu wurde das Firefox-Plugin *Copper* installiert, mit welchem man HTTP-Requests verfolgen kann. Über eine SPARQL-UI konnten ein paar Beispiel Requests verschickt werden.

Listing 2.1: Beispiel Request

```
PREFIX itm: <http://gruppe03.pit.itm.uni-luebeck.de/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT * WHERE {
  itm:time1 itm:hour ?hour .
  itm:time1 itm:minute ?minute .
  itm:time1 itm:seconds ?seconds .
}
```

Listing 2.2: Beispiel Response

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
<head>
<variable name="hour"/>
<variable name="minute"/>
<variable name="seconds"/>
</head>
<results>
<result>
<binding name="hour">
<literal datatype="http://www.w3.org/2001/XMLSchema#integer">9</literal>
</binding>
<binding name="minute">
<literal datatype="http://www.w3.org/2001/XMLSchema#integer">6</literal>
</binding>
<binding name="seconds">
<literal datatype="http://www.w3.org/2001/XMLSchema#integer">21</literal>
</binding>
</result>
</results>
</sparql>
```

2.1.2 Java-Programm

Als weiterführende Aufgabe soll jetzt ein Programm entwickelt werden, welches auf Basis der ssp-example-Application die LUX-Werte des LDR an den SSP versendet. Dazu wurden elementar wichtige Klassen übernommen bzw. erweitert (z.B ObservableBase-Service), um dann einen eigenen CoAP Service für den LDR zu entwickeln.

Das folgende Klassendiagramm zeigt die Übersicht aller wichtigen Komponenten:

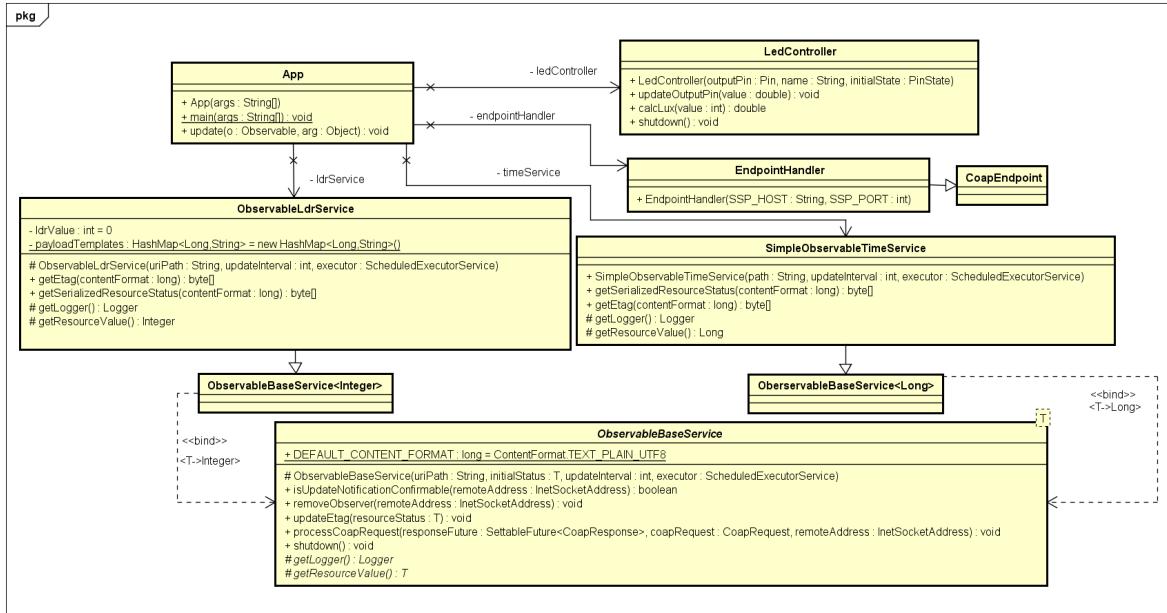


Abbildung 2.1: Klassendiagramm Meilenstein 2, Aufgabe 1

In der Klasse *ObservableLdrService*, welche den BaseService erweitert, liegt sämtliche Semantik, um den SSP mit Lux-Werten zu versorgen. Zunächst wird das *Content-Format* so angepasst, dass es der standardisierten Ontologie des ITM entspricht. Um das einheitliche Schema fortzuführen müssen Änderungen in der Ontologie gruppenübergreifend abgesprochen werden.

Listing 2.3: Aufbau des Content-Formats als Triple Muster

```

"@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> . \n" +
"@prefix xsd: <http://www.w3.org/2001/XMLSchema#> . \n" +
"@prefix pit: <https://pit.itm.uni-luebeck.de/OWLPATH#> . \n" +
"@prefix itm: <http://gruppe03.pit.itm.uni-luebeck.de/> . \n" +
"itm:Pi rdf:type pit:Device ; \n" +
"\t pit:hasIp \"141.83.175.234\" ; \n" +
"\t pit:hasGroup \"PIT_03-SS17\" ; \n" +
"\t pit:hasComponent itm:Ldr . \n" +
"itm:Ldr rdf:type pit:Component ; \n" +
"\t pit:isType \"LDR\" ; \n" +
"\t pit:lastModified \"%s\"^^xsd:dateTime ; \n" +
"\t pit:isActor \"false\"^^xsd:boolean ; \n" +
"\t pit:hasURL \"coap://141.83.175.234:5683/ldr\"^^xsd:anyURI ; \n" +
"\t pit:hasDescription \"Light dependant resistor\" ; \n" +
"\t pit:hasStatus itm:Lux . \n" +
"itm:Lux rdf:type pit:Status ; \n" +
"\t pit:hasScaleUnit \"Lux\" ; \n" +
"\t pit:hasValue \"%d\"^^xsd:integer . \n"

```

Um einen Timestamp mitzusenden, wurde die Klasse *SimpleObservableTimeService* für dieses Projekt angepasst, sodass nun der LUX-Wert, sowie der Timestamp im richtigen Format an den SSP gesendet werden. Wir haben uns weitergehend auf ein Sendeintervall von vier Sekunden geeinigt, da bei vier Gruppen theoretisch dann jede Sekunde einen Wert gesendet wird und der SSP somit nicht überlastet wird.

Listing 2.4: Aufbau des TimeStamps als Triple Muster

```
"@prefix itm: <http://gruppe03.pit.itm.uni-luebeck.de/>\n" +
"@prefix xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
"\n" +
"itm:time1 itm:hour \"%02d\"^^xsd:integer .\n" +
"itm:time1 itm:minute \"%02d\"^^xsd:integer .\n" +
"itm:time1 itm:seconds \"%02d\"^^xsd:integer ."
```

Der *EndpointHandler* ist dafür da, um einen neuen Serviceendpunkt am SSP zu registrieren und die Verbindung zum Server Proxy herzustellen. In dieser Anwendung werden folglich der TimeService sowie der LDRService am SSP registriert.

Die Komponente *LedController* berechnet den LUX-Wert anhand des eingegebenen Werts vom LDR und stellt eine Funktion zur Verfügung, welche die LED an- bzw. ausschaltet. Hierfür wird weiterhin die Pi4J-Bibliothek verwendet.

In der Main-Klasse *App* werden sämtliche Variablen für den Verbindungsauflauf zum SSP gesetzt, sowie alle wichtigen Komponenten für den Ablauf des Programms initialisiert, unter anderem der EndpointHandler.

Listing 2.5: Registrierung der Services

```
// initializes a new EndpointHandler
handler = new EndpointHandler(SSP_HOST, SSP_PORT);

// register all services at the ssp
handler.registerWebresource(new SimpleObservableTimeService("/utc-time", 1,
    handler.getExecutor()));
handler.registerWebresource(ldrService);
```

2.1.3 Java-Anwendung

Abschließend soll ein Java- Programm implementiert werden, welches die LED aus 1 in Abhängigkeit aller im SSP gespeicherten LUX- Werte schaltet. Die LED soll angehen, sofern der durchschnittliche LUX- Wert aller Gruppen unter unserem Schwellwert liegt. Ansonsten wird sie wieder ausgeschaltet.

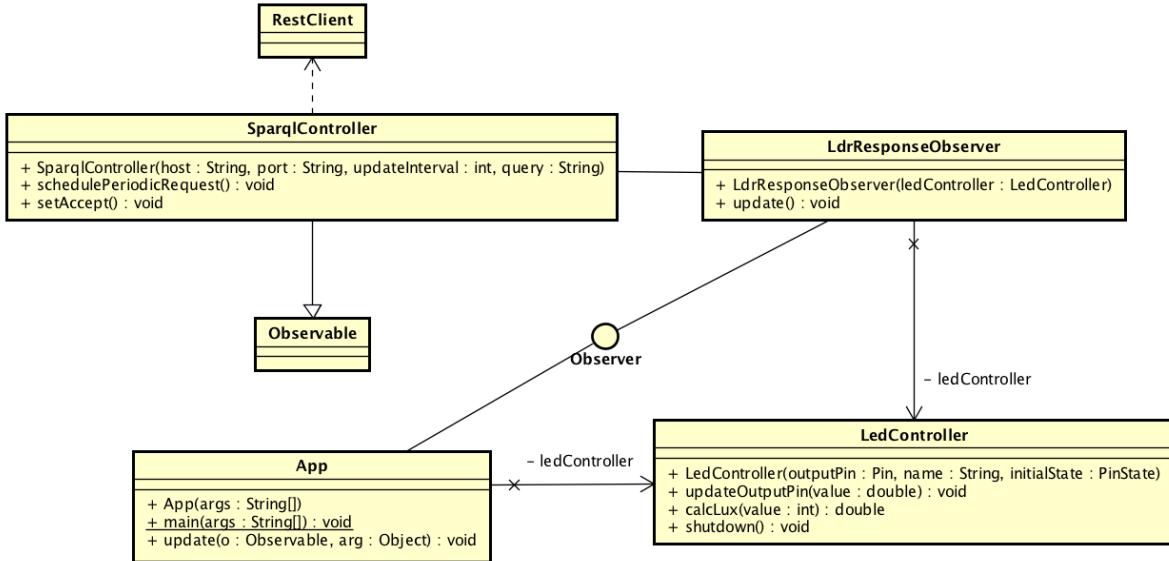


Abbildung 2.2: Klassendiagramm Meilenstein 2, Aufgabe 2

Für die Berechnung des durchschnittlichen LUX- Wertes, haben wir eine neue Klasse *LdrResponseObserver* erstellt. Diese implementiert die *Observer-Schnittstelle* und wird über die *Update-Methode* immer über neue LUX- Werte der Gruppen informiert. Um die in Listing 2.6 gezeigte Anfrage an den SSP zu senden, wird der *SparqlController* verwendet. Intern benutzt dieser den *RestClient* für die Kommunikation. Der *SparqlController* erbt von der Klasse *Observable* und leitet die erhaltene Antwort an die Observer.

Listing 2.6: Anfrage der LUX-Werte an den SSP

PREFIX pit: <<https://pit.itm.uni-luebeck.de/OWLPATH#>>

```

SELECT * WHERE {
    ?component pit:hasStatus ?status .
    ?status pit:hasValue ?value .
    ?status pit:hasScaleUnit "Lux" .
}
  
```

In dem nachfolgenden Listing 2.7 wird ein Abschnitt der Methode abgebildet. Nachdem sämtliche Werte vom SSP gesammelt wurden, konnte das im Response enthaltene XML geparsert und der Durchschnittswert aller LUX-Werte ermittelt werden. In Abhängigkeit von diesem Wert wird die LED ein oder ausgeschaltet. Das Schalten der LED ist in dem Listing 1.1 abgebildet.

Listing 2.7: Berechnung des LUX-Durchschnittes aller Gruppem

```
// Encode the response into a byte array
InputStream stream = new ByteArrayInputStream(xml.getBytes("UTF-8"));
// Create a instance of an xml document
Document doc = new SAXBuilder().build( stream );
// Get the root element and the namespace
Element root = doc.getRootElement();
Namespace namespace = root.getNamespace();

// Get the lux-results of the xml
Element results = root.getChild("results", namespace);
List<Element> resultList = results.getChildren();
int sum = 0;

// Loop through all results and calculate the sum
for (Element result : resultList) {
    for (Element binding : result.getChildren("binding", namespace)) {

        if (binding.getAttribute("name").getValue().equals("value")) {
            Element literal = binding.getChild("literal", namespace);
            sum += Integer.parseInt(literal.getText());
        }
    }
}
// Calculate the average value
int avg = sum / resultList.size();
// Update the outputpin
ledController.updateOutputPin(avg);
```

3 Dritter Meilenstein

4 Vierter Meilenstein

Literaturverzeichnis

- [1] D. B. u. F. L. Raphael Allner, *Projekt Internet-Technologien (zu CS4509) - Handbuch*. Universitaet zu Luebeck, 2017.
- [2] P. E. GmbH, *PFW55 Series Photoresistor*. Pollin Electronic GmbH.
- [3] “The Pi4J Project - Java I/O library for the Raspberry Pi.” <http://pi4j.com>, 2016. [Online; letzter Zugriff 22-Mai-2017].