



Projekt Internet Technologie

Anwendungsentwicklung für einen intelligenten Raum

Projektarbeit

im Rahmen des Studiengangs Informatik der Universität zu Lübeck

Vorgelegt von

Annika Korz, Dominik Mairhoefer, Michael Olbrich

Contents

1	Aufgabenstellung	1
2	Material und Methoden	3
2.1	Plattformen	3
2.1.1	Raspberry Pi	3
2.1.2	Arduino	4
2.2	Software	4
2.2.1	Semantic Web	6
2.2.2	SSP	7
2.2.3	nCOAP	7
2.2.4	Informationsdienste	8
2.2.5	OpenCV	8
2.3	Bauteile	9
2.3.1	Leuchtdiode	9
2.3.2	Fotowiderstand	10
3	Meilenstein I	11
3.1	LED	11
3.2	LDR Digital	12
3.3	LDR Analog	14
4	Meilenstein II	21
4.1	COAP2SSP	21
4.2	SSP2PI	22
5	Meilenstein III	29
5.1	Smart Mirror	29
5.1.1	Gesichtserkennung	31

Contents

5.1.2	Informationen	33
5.1.3	Darstellung	36
List of Figures		43
Listings		45
List of Tables		47
List of Corrections		49

1 Aufgabenstellung

Das Projekt Internet Technologie umfasst die praktische Anwendung der in der zugehörigen Vorlesung vorgestellten Konzepte und Systeme. Im Speziellen ist das Projekt auf die zunehmende Verbreitung und Integration digitaler Services in die alltägliche Umgebung ausgelegt. Zusammengefasst unter Schlagworten wie Internet of Things oder Ambient Computing werden dabei digitale Rechenwerke, Aktorik und Sensorik miteinander vernetzt um bestehende Funktionalitäten zu erweitern, beispielsweise einen Lichtschalter zu einer Lichtsteuerung zu machen oder gänzlich neue Möglichkeiten zu Interaktion mit der Umgebung zu schaffen.

Vor diesem Hintergrund sollen die teilnehmenden Gruppen in Kollektivarbeit eigene Konzepte entwickeln, welche sich zu einem intelligenten Raum zusammenfassen lassen. Das Projekt nutzt Raspberry Pi und Arduino als Entwicklungsplattform und bedient sich zeitgemäßer Technologien wie CoAP, SPARQL oder Web-Sockets. Die einzelnen Projekte werden mit Maven gruppenübergreifend standardisiert, die Versionsverwaltung erfolgt mit Git.

Ein weiteres Ziel ist die Erstellung einer Bibliothek für den Zugriff auf die verwendeten Hardwarekomponenten.

2 Material und Methoden

Dieses Kapitel gibt einen Überblick der im Projekt verwendeten Hardware und Software.

2.1 Plattformen

Als Minicomputer bestehend aus einer einzelnen Platine wurden Raspberry Pi und Arduino entwickelt um Nutzer an digitale Technik heranzuführen und dabei spielerisch Programmierfähigkeiten zu vermitteln. Die Fülle an Schnittstellen macht die Plattform ideal für das Prototyping interaktiver Anwendungen unter Verwendung von Sensorik und Aktorik.

2.1.1 Raspberry Pi

Der Raspberry Pi ist die vorrangige Entwicklungsplattform des Projekts. Das Institut für Telematik stellt zwei Versionen zur Verfügung, Diese unterscheiden sich in Ausstattung und Leistungsfähigkeit und sind in Bild 2.1 dargestellt. Die Platinen bieten einen HDMI zur direkten Bildschirmausgabe und werden mit dem Betriebssystem Raspbian betrieben, welches auf Debian basiert. Konzipiert für die ARM-Prozessoren und vergleichsweise geringen Ressourcen der Raspberries bietet das System mehr als 35.000 auf Performance ausgelegt Softwarepakete. Die im Projekt verwendete Version Raspbian Jessie Lite verfügt über keine grafische Benutzeroberfläche und ist nur knapp 1,5GB groß. Das Besondere an der Platine sind die General Purpose Input Output Ports (GPIO), welche wahlweise als Eingang oder Ausgang konfiguriert werden können. Der verbaute BCM2835 Mikrocontroller gibt der Nummerierung der Pins den offiziellen Namen BCM-Numbering. In Bild 2.2 ist das BCM-Numbering auf dem T-Cobler dargestellt. Dieser verbindet die Platine mit dem verwendeten Breadboard.



(a) Raspberry Pi 2 B - 4x 900 MHz, 1GB RAM, externes WLAN (b) Raspberry Pi 3 B - 4x 1,2 GHz, 1GB RAM, integriertes WLAN und Bluetooth

Figure 2.1: Versionen der durch das Institut für Telematik bereitgestellten Raspberry Pi

2.1.2 Arduino

Der Arduino Uno basiert auf dem ATmega328P Microcontroller und bietet neben 14 digitalen I/O-Pins auch sechs analoge Pins. Diese sind an mit einem 6Kanal A/D-Wandler verbunden wodurch u.A. das Auslesen analoger Sensorik ermöglicht wird. Das Board kommt mit seiner eigenen Entwicklungsumgebung, der Arduino IDE, diese wird in C programmiert und bietet umfangreiche Bibliotheken zum Ansteuern der einzelnen Funktionalitäten. Das Layout der Platine ist in Bild 2.3 dargestellt.

2.2 Software

Arduino IDE Für die Programmierung des Arduino bereitgestellt Entwicklungsumgebung. Wird in C programmiert und stellt eine umfangreiche Bibliothek zum einfachen Ansteuern der Platine bereit.

PI4J Bibliothek zum Ansteuern der Input/Output-Ports des Raspberry Pi in Java.

RXTX Bibliothek für die Nutzung der seriellen Schnittstelle des Pi in Java.

Maven Java basiertes Build-Management Tool. Ermöglicht die standardisierte Erzeugung von Paketen.

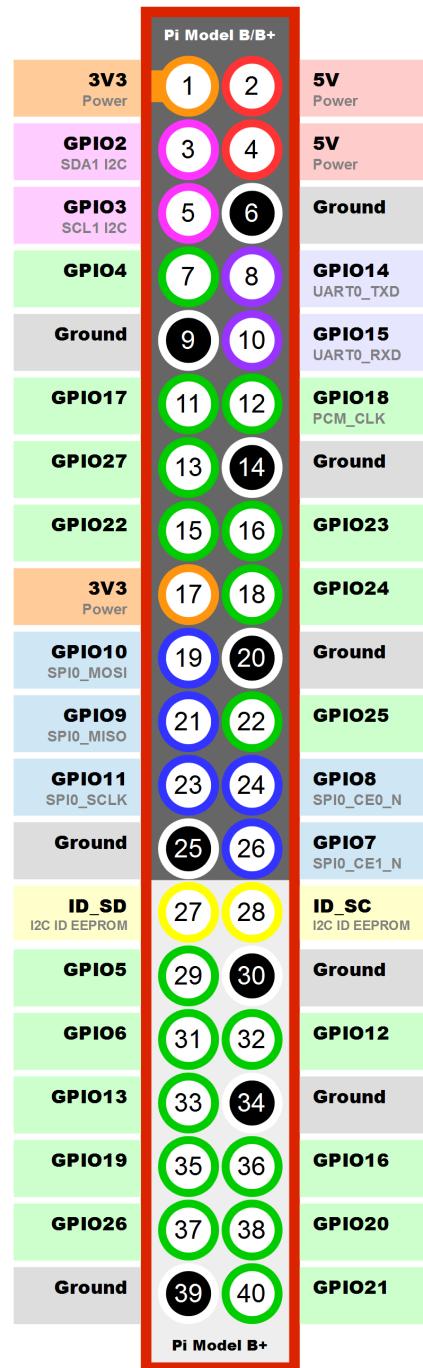


Figure 2.2: BCM-Numbering der GPIO Ports am Raspberry Pi

2 Material und Methoden

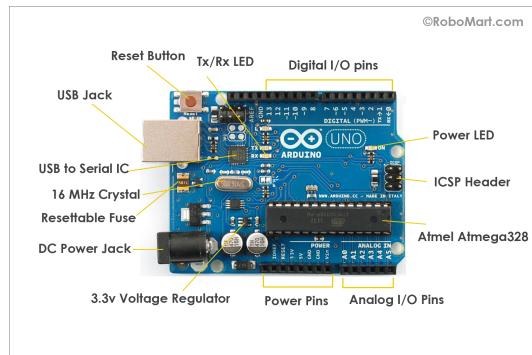


Figure 2.3: Layout Arduino Uno

Git Ein Open Source Programm zur Versionsverwaltung von Dateien. Verwaltet die Versionierung von gemeinsam genutzten Dateien und ermöglicht damit ein koordiniertes Arbeiten an Projekten mit mehreren Entwicklern.

Eclipse Integrierte Entwicklungsumgebung u.a. für Java mit der Möglichkeit eine Vielzahl von Plugins zur Optimierung des Entwicklungsprozesses einzubinden.

2.2.1 Semantic Web

Der Kontext bestimmt die Beziehung zwischen miteinander verbunden Objekten oder Informationen und impliziert zusätzliche Informationen, welche nicht durch die unabhängige Betrachtung der einzelnen Komponenten ersichtlich sind. Während Menschen in der Lage sind den Kontext zu erfassen, daraus intuitiv Verknüpfungen zu bilden und auf weitere Informationen zu schließen, muss dieses Wissen für Maschinen explizit bereitgestellt werden. Das Semantic Web ist die technische Umsetzung dieses Umstandes und zielt darauf ab Datenaustausch und Auswertung durch Computer zu vereinfachen. Unter der Führung des W3C wird seit 2004 die standardisierte Web Ontology Language (OWL) entwickelt um Ontologien in einer formalen Beschreibungssprache konzipieren, veröffentlichen und teilen zu können. Damit können die Eigenschaften einer Domäne in maschinenverständlicher Form beschrieben werden.

```

0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+
|Ver| T | TKL |     Code      |       Message ID    |
+-----+-----+-----+-----+
|   Token (if any, TKL bytes) ...
+-----+-----+-----+-----+
|   Options (if any) ...
+-----+-----+-----+-----+
|1 1 1 1 1 1|   Payload (if any) ...
+-----+-----+-----+-----+

```

Figure 2.4: Coap Nachrichtenformat

2.2.2 SSP

Der Smart Service Proxy wurde durch Oliver Kleine am ITM der Universität zu Lübeck entwickelt und implementiert um nach dem Konzept des Semantic Web ein digitales Abbild der Gegenwart zu ermöglichen. Dazu können Sensorik und Aktorik am SSP registriert und deren Status kontinuierlich aktualisiert werden. Clients können den Zustand aller registrierten, beobachtbaren Ressourcen am SSP abrufen und für ihre Zwecke nutzen. Damit bildet der SSP die Grundlage für die Integration digitaler Services in eine natürliche Umgebung.

2.2.3 nCOAP

Das Constraint Application Protocol ist eine leichtgewichtige Alternative zu HTTP die den Zugriff auf Ressourcen in ressourcen-beschränkten Umgebungen realisiert. Statt TCP wird UDP genutzt um asynchronen Datenaustausch zu ermöglichen. Im Vergleich zu HTTP reduzieren die komprimierten binären Header und Body Abschnitte den Overhead der Kommunikation. Die Gemeinsamkeiten zu HTTP, wie Anfragemethoden *GET*, *PUT*, *POST* und *DELETE* oder ähnliche Fehlercodes ermöglichen die einfache Implementierung von Cross-Protocol-Proxies. Jede Coap Nachricht ist aufgebaut aus einem 4Byte Header, den CoAP-Optionen und dem CoAP-Body, welcher die Payload enthält. Der Aufbau einer Coap-Nachricht wird in Bild 2.4 dargestellt. In diesem Projekt wird die durch Oliver Kleine entwickelte Java Implementierung nCoAP verwendet um mit dem Smart Service Proxy (SSP) zu kommunizieren.

2.2.4 Informationsdienste

iCal

Das Datenformat iCalendar definiert den Internet Media Type *text/calendar* in der RFC 5545. Es wurde durch die Arbeitsgruppe Calsify der Internet Engineering Task Force entwickelt um die Veröffentlichung von Terminen und die gemeinsame Nutzung von Kalendern zu ermöglichen. Das Datenformat selbst ist nicht auf ein spezifisches Netzwerkprotokoll beschränkt. Daher ist in der RFC 5546 das Transport Independent Interoperability Protocol (iTIP) beschrieben, welches den plattformübergreifenden Austausch realisiert.

Weather API

Der Servicedienstleister openweathermap.org bietet neben Anderen die Möglichkeit Datensätze zum aktuellen und prognostizierten Wetter nahezu aller Regionen der Erde abzurufen. So kann beispielsweise mit der Anfrage `api.openweathermap.org/data/2.5/weather?q=London,uk` die momentan Wetterlage in London in Form einer `*.json` abgerufen werden. In der Regel ist die Anzahl der möglichen Anfragen pro Stunde für das kostenlose Angebot begrenzt. Auch weiterführende Funktionen wie 16-Tage Vorhersagen oder detaillierte Wetterkarten sind Premiumkunden vorbehalten.

RSS

Das *Rich Site Summary* ist ein standardisiertes Format für die Zusammenfassung von Informationen und Aktualisierungen auf Webseiten. Daher auch die alternative Bezeichnung *Really Simple Syndication*. Dabei wird ein News-Feed durch eine *URI* identifiziert und durch Attribute wie Titel, Veröffentlichungsdatum oder Sprache beschrieben. Dem Feed zugeordnet sind wiederum Feed-Nachrichten, welche die eigentlichen Informationen in vergleichbaren Attributen enthalten. Diese hierarchische Struktur wird in der Auszeichnungssprache XML beschrieben.

2.2.5 OpenCV

- Bibliothek für Echtzeit Computer Vision

Table 2.1: LED Flussspannungen

Art	Flussspannung
Infrarot	1,2 - 1,8V
Rot	1,6 - 2,2V
Gelb, Grün	1,9 - 2,5V
Blau, Weiß	2,7 - 3,5V
Ultraviolett	3,1 - 4,5V

2.3 Bauteile

Dieser Abschnitt stellt die im Projekt verwendeten Bauteile vor und gibt einen kurzen Überblick über Eigenschaften und Funktionsweise. Mit fortschreitendem Projektverlauf wird die Sektion stetig erweitert werden.

2.3.1 Leuchtdiode

Die Leuchtdiode ist ein Halbleiterbauelement, welches im Aufbau einer pn-Halbleiterdiode entspricht und sich mit diesen die gleichen Grundeigenschaften teilt. In Durchflussrichtung schließt sie einen Stromkreis, während sie entgegen der Durchflussrichtung sperrt. Im Betrieb strahlt der verbaute Halbleiterkristall Licht ab dessen Wellenlänge von der Dotierung und Art des verwendeten Materials abhängt. So können Leuchtdioden im sichtbaren Bereich Licht in verschiedenen Farben erzeugen, aber auch infrarote oder ultraviolette Strahlung abgeben. Die LED wird z.B. in Produkten wie Taschenlampen, Flachbildschirmen oder als Signalgeber in Sensoren eingesetzt.

Die Eigenschaften von Leuchtdioden sind abhängig von der jeweiligen Art, so unterscheiden sich die Spezifikationen für alle verfügbaren Modelle. Um eine Überlastung zu vermeiden und die Lebensdauer der Bauteile zu vermeiden sollten Leuchtdioden nur mit einem Vorwiderstand betrieben werden. Dieser muss auf die gegebene Spannungsquelle und das verwendete Bauteil abgestimmt werden.

2.3.2 Fotowiderstand

Der Fotowiderstand ist ein lichtempfindliches Halbleiterbauelement dessen ohmscher Widerstand von der Stärke des einfallenden Lichts abhängig ist. In elektrischen Schaltungen wird das Bauteil Signalaufnehmer zur Erfassung des Umgebungslichts verwendet. Beispielhafte Anwendungen sind Dämmerungsschalter oder Belichtungsmesser von Kameras. In Reihe mit einem korrekt dimensionierten Widerstand entsteht ein Spannungsteiler über den der Widerstand gemessen werden kann.

3 Meilenstein I

Für den ersten Meilenstein soll ein Programm entwickelt werden, welches auf den Status eines Fotowiderstandes (engl. Light Dependent Resistor, kurz LDR) reagiert. Beim Abdunkeln des LDR wird die LED eingeschaltet, während sie beim Aufhellen deaktiviert wird. Bei der Umsetzung dieser Funktionalität sind die speziellen Charakteristiken der (im vorherigen Kapitel vorgestellten) elektrischen Bauteile zu beachten und in die Planung miteinzubeziehen. Im Folgenden werden die dazu benötigten Vorbetrachtungen in Arbeitsschritten zusammengefasst.

3.1 LED

Im ersten Schritt soll eine LED per GPIO des Raspberry Pi angesteuert werden. Wie in Kapitel 2.3 beschrieben, sollte eine Leuchtdiode in Reihe mit einem Vorwiderstand betrieben werden um die Spannung über diese zu begrenzen.

Die GPIO's des Raspberry Pi stellen 3,3V bei 16mA zur Verfügung. Die Spezifikation der verwendeten LED kann aus der Kennlinie im jeweiligen Datenblatt bestimmt werden. In Bild 3.1(a) ist der Arbeitspunkt im Strom/Spannungsdiagramm einer grünen LED eingetragen. Man kann erkennen, dass die Durchflussspannung bei gegebenen 16mA Strom ca. 2,2V beträgt. Bild 3.1(b) zeigt die zu berechnende Schaltung bei gegebener Spannung $U_0 = 3,3V$ und Strom $I = 16mA$, der gesuchte Vorwiderstand R_V berechnet sich aus dem ohmschen Gesetz 3.1.

Der Gesamtwiderstand einer Reihenschaltung ist die Summe der Einzelwiderstände wie in Gleichung 3.2, genauso ergibt sich die Gesamtspannung aus der Summe der Spannungen über die einzelnen Bauelemente. In diesem Fall gilt: $3,3V = U_{LED} + U_{R_V}$. Aus der Kennlinie der Diode wurde $U_{LED} = 2,2V$ abgelesen. Umstellen des ohmschen Gesetzes ergibt Gleichung 3.3 zur Berechnung des Vorwiderstandes. Diese Werte ergeben einen Vorwiderstand von $68,75\Omega$. Da i.d.R. nicht

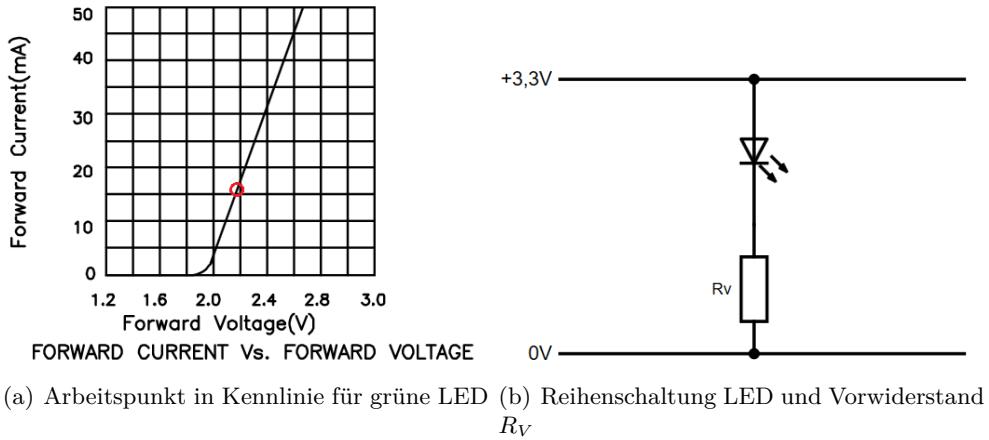


Figure 3.1: Arbeitspunkt und Reihenschaltung einer Leuchtdiode

so spezifisch hergestellt werden, wird der nächstgrößere Widerstand mit 100Ω verwendet. Mit Hilfe der pi4j-Bibliothek wird ein Pin für die Ansteuerung der LED konfiguriert wie in Codebeispiel 3.1 gezeigt.

$$U = R * I \quad (3.1)$$

$$R_{Gesamt} = \sum_i^n R_i \quad (3.2)$$

$$R_V = \frac{U_{R_V}}{I} \quad (3.3)$$

3.2 LDR Digital

Im zweiten Schritt soll der digitale Wert des LDRs mit dem Raspberry Pi ausgelesen werden. Damit kann erkannt werden, ob es gerade hell oder dunkel ist. Bei einer dunklen Umgebung weist der LDR einen Widerstand von $10k\Omega$ und in einer hellen Umgebung einen Widerstand von $1k\Omega$ auf. Da der Raspberry Pi für Spannungen $< 0,8V$ eine logische 0 und für Spannungen $> 2,0V$ eine logische 1 erkennt, nehmen wir einen Spannungsteiler zur Hilfe. Der korrekte Widerstand lässt sich mit Hilfe

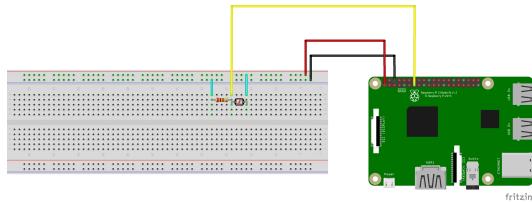


Figure 3.2: Realisierung der Verbindung des LDR mit dem Raspberry Pi

der Formel 3.4 berechnen:

$$\frac{U_{Ges}}{R_{Ges}} = \frac{U_1 + U_2}{R_1 + R_2} = \frac{U_1}{R_1} = \frac{U_2}{R_2} \quad (3.4)$$

So lässt sich nun einmal der benötigte Widerstand berechnen, für den Fall, dass der Raspberry Pi eine logische 0 erhalten soll und der LDR eine dunkle Umgebung erkennt:

$$\frac{U_1}{R_1} = \frac{U_2}{R_2} \quad (3.5)$$

Mit den Werten eingesetzt ergibt sich somit die Formel aus 3.6.

$$\frac{1,3V}{R_1} = \frac{2,0V}{10k\Omega} \quad (3.6)$$

Analog können wir den benötigten Widerstand für die Helligkeit berechnen, so dass die Spannung im Bereich von $> 2,0V$ liegt und der Raspberry Pi eine logische 1 erkennt (3.7).

$$\frac{2,5V}{R_1} = \frac{0,8V}{1k\Omega} \quad (3.7)$$

Löst man die beiden Gleichungen auf, so erhält man das Ergebnis, dass unser Widerstand zwischen $6,5k\Omega$ und $3,125k\Omega$ liegen muss. Wir haben daher einen Widerstand von $4.7k\Omega$ gewählt. Die Schaltung ist in Abbildung 3.2 veranschaulicht.

Den Wert des GPIOs mittels WiringPi lässt sich nun durch den Befehl 3.2 auslesen, wobei der GPIO 2 verwendet wurde.

Listing 3.2: Auslesen des GPIOs mittels WiringPi

```
1 gpio read 2
```

Je nach Beleuchtung wird nun eine 0 für dunkel und eine 1 für hell zurückgeliefert.

3.3 LDR Analog

In diesem Abschnitt des Meilensteins wird der Widerstandswert des Fotowiderstands mit dem im Arduino verbauten A/D-Wandler ausgelesen und in den korrespondierenden Lux-Wert umgerechnet. Wie in Bild 3.3 gezeigt wird ein Spannungsteiler mit Abgriff über den Fotowiderstand konstruiert. Der analoge Eingang am Arduino ist damit in der Lage die über den LDR abfallende Spannung zu bestimmen. Der 10-Bit A/D-Wandler des Arduino gibt Werte zwischen 0 – 1023 aus, daher berechnet sich die über den LDR abfallende Spannung wie in Gleichung 3.8 beschrieben. Der entsprechende Widerstand ergibt sich durch Umstellen der Spannungsteilerformel zu Gleichung 3.9. In Codebeispiel 3.3 wird gezeigt wie der Wert des Fotowiderstandes mittels analogem Pin am Arduino ausgelesen wird. Schließlich soll Wert des Fotowiderstandes noch in den entsprechenden Lux-Wert umgerechnet werden. Zu diesem Zweck wird das nichtlineare Kennlinienverhalten des Fotowiderstandes linearisiert, wie in Bild 3.4 dargestellt ist. Hier werden die zwei Punkte $P1 : 6k\Omega, 50Lux$ und $P2 : 50k\Omega, 2Lux$ gewählt und damit eine Geradengleichung 3.10 zur Approximation des Lux-Wertes konstruiert. Die logarithmische Darstellung des Widerstandsverhaltens erfordert eine Anpassung in der Geradengleichung die in Gleichung 3.11 beschrieben ist. Durch Einsetzen der gewählten Punkte lassen sich Steigung und Offset der Gerade zu 3.12 und 3.13 berechnen und die Gleichung zur Umrechnung in Lux-Wert aufstellen 3.14.

$$U_{LDR} = \frac{analogValue * U_{IN}}{1023} \quad (3.8)$$

$$R_{LDR} = \frac{R_V * U_{LDR}}{U_{IN} - U_{LDR}} \quad (3.9)$$

$$f(x) = m * x + n \quad (3.10)$$

$$\ln(E_{Lux}) = m * \ln(R_{LDR}) + n \quad (3.11)$$

$$m = \frac{\ln(50) - \ln(2)}{\ln(6) - \ln(50)} = -1.5181489 \quad (3.12)$$

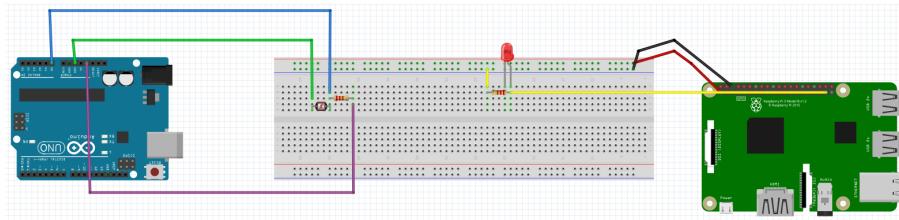


Figure 3.3: Spannungsteiler mit Abgriff am Fotowiderstand

$$n = \ln(50) - m * \ln(6) = 6.6321807 \quad (3.13)$$

$$E_{Lux} = R_{LDR}^{-1.5181489} * 759.1358 \quad (3.14)$$

Die Kommunikation zwischen Raspberry Pi und Arduino erfolgt über die serielle Schnittstelle. Dazu wird der Arduino mittel USB Kabel an den Raspberry Pi angeschlossen. Im Codebeispiel 3.3 ist zu erkennen, dass der Arduino die serielle Kommunikation mit 9600 Baud startet und den Wert des LDR alle 2 Sekunden ausliest und auf den Kanal legt. Auf Seiten des Raspberry Pi wird die serielle Schnittstelle mit Hilfe der RXTX-Bibliothek angesprochen. Die Initialisierung der seriellen Schnittstelle ist in Codebeispiel 3.4 beschrieben. Das Senden eines Wertes erzeugt im Raspberry Pi ein Event auf dem seriellen Port, in Codebeispiel 3.5 ist dargestellt wie der Pi auf solch ein Event reagiert. In diesem Fall wird die LED aktiviert wenn der empfangene Wert größer als ein definierter Grenzwert ist und sonst deaktiviert.

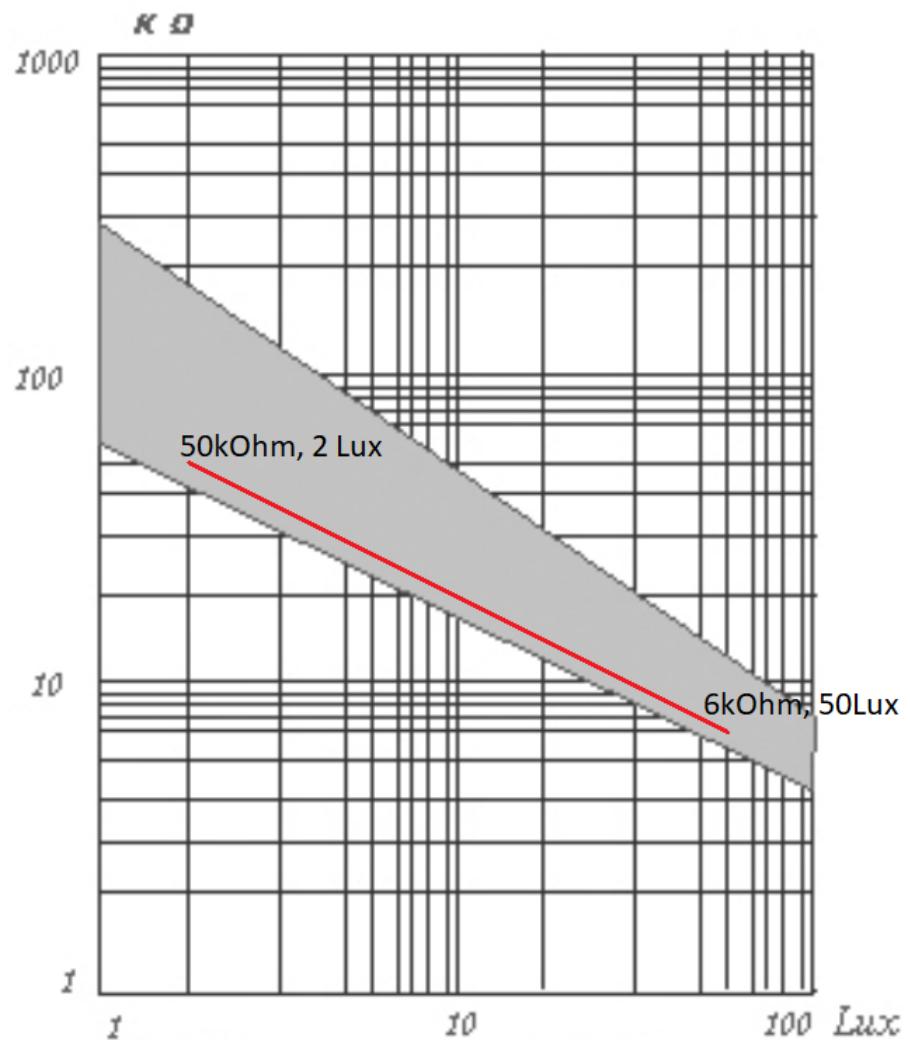


Figure 3.4: Widerstandscharakteristik LDR mit Linearisierungsgeraden

Listing 3.1: Ansteuerung einer LED mittels GPIO

```

1 public class LedControl {
2     public static void main(String[] args) throws
3         InterruptedException {
4
5         // get a handle to the GPIO controller
6         final GpioController gpio = GpioFactory.getInstance();
7
8         // creating the pin with parameter PinState.HIGH
9         // will instantly power up the pin
10        final GpioPinDigitalOutput pin = gpio.
11            provisionDigitalOutputPin(RaspiPin.GPIO_25, "PinLED"
12            , PinState.HIGH);
13        System.out.println("light is: ON");
14
15        // wait 2 seconds
16        Thread.sleep(2000);
17
18        // turn off GPIO 1
19        pin.low();
20        System.out.println("light is: OFF");
21        // wait 1 second
22        Thread.sleep(1000);
23        // turn on GPIO 1 for 1 second and then off
24        System.out.println("light is: ON for 1 second");
25        pin.pulse(1000, true);
26
27        // turn on GPIO 1
28        pin.high();
29        System.out.println("light is: ON");
30
31        // release the GPIO controller resources
32        gpio.shutdown();
33    }
34 }
```

Listing 3.3: Auslesen des analogen LDR Widerstandswertes und Umrechnung in Lux

```
1 #include <Wire.h>
2 #include <math.h>
3
4 int sensorPin = 0;
5 int ledPin = 13;
6 int sensorValue = 0;
7
8 const float u_in = 5.0; // input voltage in volt
9 const float r_v = 4.7; // reference resistor in ohm
10 const float pow_factor = -1.31022; // calculated value
    for lux estimation
11 const float l_factor = 210.9143;
12
13 float u_ldr = 0.0; // voltage over light sensor
14 float r_ldr = 0.0; // resistance of light sensor
15 float e_v = 0.0; // Lux value measured
16
17 void setup() {
18     // start serial communication
19     Serial.begin(9600);
20 }
21
22 // main function repeated in a loop
23 void loop() {
24     // readout sensor Pin
25     sensorValue = analogRead(sensorPin);
26
27     u_ldr = ((sensorValue * u_in) / 1023);
28     r_ldr = ((r_v * u_ldr) / (u_in - u_ldr));
29     e_v = (pow(r_ldr, pow_factor)) * l_factor;
30
31     // send calculated lux-value
32     Serial.println(e_v);
33     delay(2000);
34 }
```

Listing 3.4: Initialisierung der seriellen Schnittstelle im Raspberry Pi, Quelle: <http://playground.arduino.cc/Interfacing/Java>

```

1 public void initialize() {
2     System.setProperty("gnu.io.rxtx.SerialPorts", "/dev/
3         ttyACM0");
4     CommPortIdentifier portId = null;
5     Enumeration portEnum = CommPortIdentifier.
6         getPortIdentifiers();
7
8     //First, Find an instance of serial port as set in
9     //PORT_NAMES.
10    while (portEnum.hasMoreElements()) {
11        CommPortIdentifier currPortId = (CommPortIdentifier)
12            portEnum.nextElement();
13        for (String portName : PORT_NAMES) {
14            if (currPortId.getName().equals(portName)) {
15                portId = currPortId;
16                break;
17            }
18        }
19    }
20    if (portId == null) {
21        System.out.println("Could not find COM port.");
22        return;
23    }
24    try {
25        // open serial port, and use class name for the appName
26        //
27        serialPort = (SerialPort) portId.open(this.getClass().
28            getName(),
29            TIME_OUT);
30        // set port parameters
31        serialPort.setSerialPortParams(DATA_RATE,
32            SerialPort.DATABITS_8,
33            SerialPort.STOPBITS_1,
34            SerialPort.PARITY_NONE);
35        // open the streams
36        input = new BufferedReader(new InputStreamReader(
37            serialPort.getInputStream()));
38        output = serialPort.getOutputStream();
39        // add event listeners
40        serialPort.addEventListener(this);
41        serialPort.notifyOnDataAvailable(true);
42    } catch (Exception e) {
43        System.err.println(e.toString());
44    }
45 }
```

Listing 3.5: Auslesen der seriellen Schnittstelle im Raspberry Pi, Quelle: <http://playground.arduino.cc/Interfacing/Java>

```
1 public synchronized void serialEvent(SerialPortEvent
2   oEvent) {
3   if (oEvent.getEventType() == SerialPortEvent.
4     DATA_AVAILABLE) {
5     try {
6       double inputLine=Double.parseDouble(input.readLine
7         ());
8       if (inputLine > Threshold) pin.high;
9       else pin.low;
10    }
11  }
12 }
```

4 Meilenstein II

Der zweite Meilenstein dient zur Einführung in Werkzeuge und Funktionalitäten des Semantic Web. Obwohl gruppenspezifische Projekte entwickelt werden sollen, sind alle Gruppen in der Lage ihre Aktoren und Sensoren als Ressourcen im Netzwerk zur Verfügung zu stellen. Eine gegebene Ontologie, siehe Bild 4.1, wird gruppenübergreifend erweitert um einheitlichen Zugriff zu gewährleisten.

4.1 COAP2SSP

Um sich mit der Ansteuerung des SSP vertraut zu machen, soll in der ersten Teilaufgabe des zweiten Meilensteins ein einfacher observierbarer Zeitservice implementiert werden. Als Vorlage dient die Beispielanwendung im bereitgestellten nCoAP Paket, wie beschrieben in Abschnitt 2.2.3. Die Klasse *SimpleObservableTimeService* erweitert dazu den vorgegebenen *SimpleObservableService* und besteht aus drei wesentlichen Abschnitten. Das Datenformat des Services wird durch Ontologien vorgegeben und in Listing 4.1 initialisiert. Von besonderem Interesse ist hier der Punkt *ContentFormat.APP_TURTLE* da es sich dabei um die im Projekt zu verwendende Beschreibungsform handelt.

Der Konstruktor der Klasse, im Listing 4.2, erhält als Übergabeparameter zum Einen den eindeutigen Bezeichner der neuen Ressource und zum Anderen das vorge sehene Aktualisierungsintervall.

Mit der Funktion *getSerializedResourceStatus(long contentFormat)* in Listing 4.3 wird die Updatefunktion der Ressource definiert. Als Rückgabewert wird ein String der gewählten Nachrichtenform mit den aktualisierten Informationen ausgegeben.

Nachdem die Ressource definiert wurde, braucht es zum Abschluss der ersten Teilaufgabe noch einen CoAP-Endpoint. Dieser Dienst wird auf dem Raspberry Pi ausgeführt und meldet sich am SSP an um dort seine definierten Ressourcen zu registrieren. Zu diesem Zweck müssen Adresse und Port des SSP beschrieben werden,

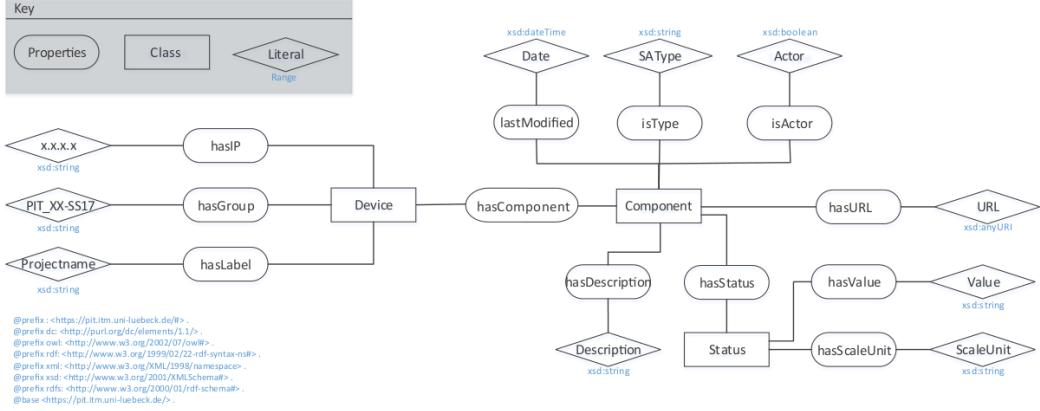


Figure 4.1: Turtle Ontology zur Verwendung und Erweiterung im Rahmen des Projekts

wie in Listing 4.4 dargestellt. Weiterhin benötigt der Endpoint Instanzen der ihm zugeordneten Ressourcen, d.h. Methoden um die entsprechenden Konstruktoren auszuführen. Schließlich erfolgt die Anmeldung am SSP mittels `registerAtSSP()`, welches in Listing 4.5 wiedergegeben wird.

Zum Abschluss der Aufgabe kann, mit Hilfe des Browser-Plugins Copper, direkt am Endpoint getestet werden, ob alle Funktionen ausgeführt werden. Mit dem gegebenen SPARQL-Endpoint `http://141.83.151.196:8080/services/sparql-endpoint` wird sichergestellt, dass die Ressourcen auch am SSP korrekt registriert werden. Dies ist auch der erste Punkt um sich aktiv mit einer SPARQL-Query auseinander zu setzen und so die Vorbereitung für die zweite Teilaufgabe einzuleiten.

4.2 SSP2PI

In der zweiten Teilaufgabe des Meilensteins wird die entwickelte Anwendung um die berechneten Lux-Werte des ersten Meilensteins erweitert. Dazu wird eine weitere beobachtbare Ressource, basierend auf vorgegebener Ontologie 4.1, zu dem Programm hinzugefügt. In Codebeispiel 4.6 ist die Beschreibung der Payload dargestellt. Wieder ist das Turtle-Format von besonderem Interesse, da hier sehr gut nachvollziehbar ist wie die gegebene Ontology praktisch umgesetzt wird. Bei der Implementierung wurden zwei Hürden identifiziert. Zum Einen muss bei der Def-

Listing 4.1: SimpleObservableTimeService Beschreibung der Payload

```

1 static{
2     payloadTemplates.put(
3         ContentFormat.TEXT_PLAIN_UTF8,
4         "The current time is %02d:%02d:%02d"
5     );
6
7     payloadTemplates.put(
8         ContentFormat.APP_TURTLE,
9         "@prefix itm: <http://gruppe04.pit.itm.uni-
10         luebeck.de/>\n" +
11         "@prefix xsd: <http://www.w3.org/2001/XMLSchema
12         #>\n" +
13         "\n" +
14         "itm:time1 itm:hour \"%02d\"^^xsd:integer .\n"
15         +
16         "itm:time1 itm:minute \"%02d\"^^xsd:integer .\n"
17         +
18         "itm:time1 itm:seconds \"%02d\"^^xsd:integer ."
19     );
20 }
```

initiation der Präfixe präzise darauf geachtet werden die korrekte URI zu verwenden. Wird Beispielsweise das Präfix *pit* mit `http://pit.itm...` statt mit `https://pit.itm...` beschrieben so handelt es sich für die Anwendung um eine andere Quelle und die Ressource wird, trotz korrekter Anfrage, nicht gefunden werden. Zum Anderen müssen Daten entsprechend ihrer Beschreibung formatiert werden. Das Attribut *pit:lastModified* ist vom Typ *xsd:dateTime*, welches die Form *YYYY-MM-TT'T'HH:mm:ss* hat. Bei Abweichung von dem Format wird sich die Ressource nicht am SSP registrieren lassen. Die weitere Implementierung dieser Ressource folgt unumwunden dem Ablauf der ersten Teilaufgabe.

Schließlich soll mittel SPARQL-Querry eine Abfrage an den SSP gesendet werden um den Durchschnitt aller registrierten Lux-Werte abzufragen und eine LED entsprechend anzusteuern. Zu diesem Zweck wird ein HTTP-Client benötigt, der die in 4.7 Abfrage an den SSP sendet und die Antwort entsprechend auswertet.

Listing 4.2: Konstruktor mit Ressourcenpfad und Updateintervall

```

1 /**
2  * Creates a new instance of {@link
3  * SimpleObservableTimeService}.
4  *
5  * @param path the path of this {@link
6  * SimpleObservableTimeService} (e.g. /utc-time)
7  * @param updateInterval the interval (in seconds)
8  * for resource status updates (e.g. 5 for every 5
9  * seconds).
 */
10 public SimpleObservableTimeService(String path, int
11     updateInterval, ScheduledExecutorService executor)
12 {
13     super(path, updateInterval, executor);
14 }
```

Listing 4.3: Updatefunktion des Services

```

1 @Override
2 public byte[] getSerializedResourceStatus(long
3     contentFormat) {
4     LOG.debug("Try to create payload (content format:
5         " + contentFormat + ")");
6
7     String template = payloadTemplates.get(
8         contentFormat);
9     if (template == null) {
10         return null;
11     } else {
12         long time = getResourceStatus() % 86400000;
13         long hours = time / 3600000;
14         long remainder = time % 3600000;
15         long minutes = remainder / 60000;
16         long seconds = (remainder % 60000) / 1000;
17         return String.format(template, hours, minutes
18             , seconds).getBytes(CoapMessage.CHARSET);
19     }
20 }
```

Listing 4.4: Definition der SSP Schnittstelle

```

1 @Option(name = "--host", usage = "Host of the SSP (ip or
2   domain)")
3   private String SSP_HOST = "141.83.151.196";
4
5 @Option(name = "--port", usage = "Port of the SSP")
6   private int SSP_PORT = 5683;

```

Listing 4.5: Registrierung am SSP

```

1 public void registerAtSSP() throws URISyntaxException {
2   URI resourceURI = new URI ("coap", null, SSP_HOST
3     , SSP_PORT, "/registry", null, null);
4   CoapRequest coapRequest = new CoapRequest(
5     MessageType.CON, MessageCode.POST, resourceURI
6     );
7   InetSocketAddress remoteSocket = new
8     InetSocketAddress(SSP_HOST, SSP_PORT);
9
10  SimpleCallback callback = new SimpleCallback();
11  this.sendCoapRequest(coapRequest, remoteSocket,
12    callback);
13 }

```

Listing 4.6: SimpleObservableLuxService Beschreibung der Payload

```
1 static{
2     payloadTemplates.put(
3         ContentFormat.TEXT_PLAIN_UTF8,
4         "The current ldr is %a"
5     );
6
7     payloadTemplates.put(
8         ContentFormat.APP_TURTLE,
9         "@prefix itm: <http://gruppe04.pit.itm.uni-
luebeck.de/>\n" +
10        "@prefix pit: <https://pit.itm.uni-luebeck.de
/>>\n" +
11        "@prefix xsd: <http://www.w3.org/2001/XMLSchema
#>\n" +
12        "\n" +
13        "itm:ldr pit:hasDescription \"Sensor for Lux
value\"^^xsd:string ." +
14        "\n" +
15        "itm:ldr pit:hasURL \"http://gruppe04.pit.itm.uni-
luebeck.de/ldr\"^^xsd:anyURI ." +
16        "\n" +
17        "itm:ldr pit:isActor \"false\"^^xsd:boolean ." +
18        "\n" +
19        "itm:ldr pit:isType \"LDR\"^^xsd:string ." +
20        "\n" +
21        "itm:ldr pit:hasStatus itm:ldrStatus ." +
22        "\n" +
23        "itm:ldr pit:lastModified \"%s\"^^xsd:dateTime
." +
24        "\n" +
25        "itm:ldrStatus pit:hasScaleUnit \"Lux\"^^xsd:
string ." +
26        "\n" +
27        "itm:ldrStatus pit:hasValue \"%s\"^^xsd:string
."
28    );
29 }
```

Listing 4.7: SPARQL-Query zur Ermittlung des durchschnittlichen Lux-Wertes

```
1 String sparql = "PREFIX itm: <http://gruppe04.pit.itm.uni-  
-luebeck.de/> \n" +  
2     "PREFIX pit: <https://pit.itm.uni-luebeck.de/>\n" +  
3     "PREFIX xsd: <http://www.w3.org/2001/XMLSchema  
#>\n" +  
4     "\n" +  
5     "Select (AVG(xsd:float(?v)) as ?luxAverage) \n" +  
6     "+\n" +  
7     "Where{ \n" +  
8     "?o pit:isType \"LDR\"^^xsd:string . \n" +  
9     "?o pit:hasStatus ?p . \n" +  
10    "?p pit:hasValue ?v . \n" +  
11    "}" ;
```


5 Meilenstein III

Nachdem die vorangegangenen Teilschritte eine Einleitung in die verwendete Technologie dargestellt haben, konstituiert der dritte Meilenstein den ersten Abschnitt des gruppenspezifischen Projektes. Ziel ist es eine eigene Anwendung zu konzipieren, Hard/Software-Anforderungen zu ermitteln und ein Grundgerüst zu implementieren und zu testen.

5.1 Smart Mirror

Wie in Abbildung 5.1 dargestellt wird ist der intelligente Spiegel eine Form von erweiterter Realität. Die grundlegende Funktionalität wird ergänzt durch die Darstellung digitaler Informationen wie Kalendereinträgen, Wetterdaten oder aktueller Nachrichten. Aufbauend auf diesem Konzept haben wir uns entschieden eine Nutzeridentifikation durch Gesichtserkennung hinzuzufügen. Die Idee dahinter ist, die dargestellten Daten den Bedürfnissen und Vorlieben des jeweiligen Anwenders anzupassen.

Für die Umsetzung des Projektes haben wir uns entschieden Gesichtserkennung und Informationsservices auf separaten Plattformen zu implementieren. Es wird ein Raspberry Pi verwendet um die Gesichtserkennung durchzuführen und darauf folgend die Nutzeridentifikation an den SSP zu senden. Ein weiterer Pi wird zur Informationsbeschaffung sowie Darstellung auf dem display genutzt. Dazu werden die Nutzerdaten am SSP abgefragt und die zugeordneten Datensätze heruntergeladen.

Tabelle 5.1 zeigt die Gliederung des Smart Mirror in Teilsysteme deren Komponenten bereits in Kapitel 2 vorgestellt wurden. Im Folgenden wird die Implementierung der einzelnen Teilsysteme vorgestellt.

5 Meilenstein III



Figure 5.1: Darstellung Smart Mirror

Table 5.1: Struktur Smart Mirror

Teilsystem	Komponente
Gesichtserkennung	OpenCV
	OpenFace
	Camera Module v2
Informationen	Calendar API
	Wetter API
	RSS Feed
Darstellung	JavaFX

5.1.1 Gesichtserkennung

Um unterschiedliche Benutzer mithilfe einer Kamera zu erkennen wird eine Gesichtserkennung benötigt. Dazu wird das Python Paket `face_recognition`¹² genutzt, welches selbst wiederum auf der Bibliothek `dlib` aufbaut, um die Gesichtserkennung zu ermöglichen. Mithilfe des Pakets `face_recognition` ist es mit wenigen Codezeilen möglich eine sehr robuste Gesichtserkennung zur Verfügung zu stellen. Ein solches Python Skript findet sich in 5.1 . Um das Paket `face_recognition` auf einem RaspberryPi verwenden zu können müssen mehrere weitere Pakete, sowie die `dlib` Bibliothek installiert werden. Im Git des `face_recognition` Projekts findet sich dazu eine ausführliche Anleitung³. Zusätzlich wurde Bibliothek OpenCV verwendet. Dies ist nicht zwingend für die Gesichtserkennung notwendig, vereinfacht jedoch die Handhabung der Bilder. Eine Anleitung zur Installation von OpenCV 3 findet sich unter ⁴.

Anschließend an die Installation kann die Gesichtserkennung mit dem gegebenen Python Skript aus 5.1 erfolgen. Um die vom Skript erkannten und ausgegebenen Gesichter in den SSP eintragen zu können, wird das Skript von einem Java Programm gestartet. Ein Code Beispiel dazu ist in 5.2 zu finden. Die eingelesenen Namen können auf die gleiche Art wie ursprünglich die Lux werte in den SSP eingetragen werden.

Die Qualität der Gesichtserkennung ist bei dem von uns verwendeten Verfahren extrem robust. Es reicht ein einziges Referenzbild von jeder zu erkennenden Person um diese aus fast allen Winkeln zu erkennen. Auch die Geschwindigkeit ist ausreichend. Es dauert höchstens 1-2 Sekunden um eine sich vor der Kamera befindende Person zu erkennen. Die Geschwindigkeit sinkt dabei mit steigender Anzahl von Personen vor der Kamera.

Listing 5.1: Python Skript für Gesichtserkennung mithilfe von `face_recognition`

```
1 import face_recognition
2 import cv2
3 import time
```

¹https://pypi.python.org/pypi/face_recognition

²https://github.com/ageitgey/face_recognition

³<https://gist.github.com/ageitgey/1ac8dbe8572f3f533df6269dab35df65>

⁴<http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/>

```
4 from picamera.array import PiRGBArray
5 from picamera import PiCamera
6
7 # initialize the camera and grab a reference to the raw
8 # camera capture
9 camera = PiCamera()
10 camera.resolution = (1280, 960)
11 camera.framerate = 32
12 rawCapture = PiRGBArray(camera, size=(1280, 960))
13 time.sleep(1)
14
15 # Load a sample picture and learn how to recognize it.
16 person1_image = face_recognition.load_image_file("/home/
17     pi/face_recognition/examples/person1.jpg")
18 person2_image = face_recognition.load_image_file("/home/
19     pi/face_recognition/examples/person2.jpg")
20 person1_face_encoding = face_recognition.face_encodings(
21     dominik_image)[0]
22 person2_face_encoding = face_recognition.face_encodings(
23     biden_image)[0]
24
25 # Initialize some variables
26 face_locations = []
27 face_encodings = []
28 face_names = []
29 scale = 8
30
31 for frame in camera.capture_continuous(rawCapture, format
32     ="bgr", use_video_port=True):
33     # Grab a single frame of video
34     frame = frame.array
35
36     # Resize frame of video to 1/4 size for faster face
```

```

    recognition processing
31 small_frame = cv2.resize(frame, (0, 0), fx=(1/scale),
                           fy=(1/scale))
32
33 # Find all the faces and face encodings in the current
   frame of video
34 face_locations = face_recognition.face_locations(
   small_frame)
35 face_encodings = face_recognition.face_encodings(
   small_frame, face_locations)
36
37 face_names = []
38 for face_encoding in face_encodings:
    # See if the face is a match for the known face(s)
39     match = face_recognition.compare_faces([
        person1_face_encoding, person2_face_encoding],
        face_encoding)
40     name = "Unknown"
41
42     if match[0]:
43         name = "Person1"
44     if match[1]:
45         name = "Person2"
46     print(name)
47     face_names.append(name)
48
49 # Release handle to the webcam
50 video_capture.release()
51 cv2.destroyAllWindows()

```

5.1.2 Informationen

Die auf dem Spiegel dargestellten Informationen umfassen aktuelles Wetter sowie eine Vorhersage für die nächsten Tage und benutzerspezifische Informationen in

5 Meilenstein III

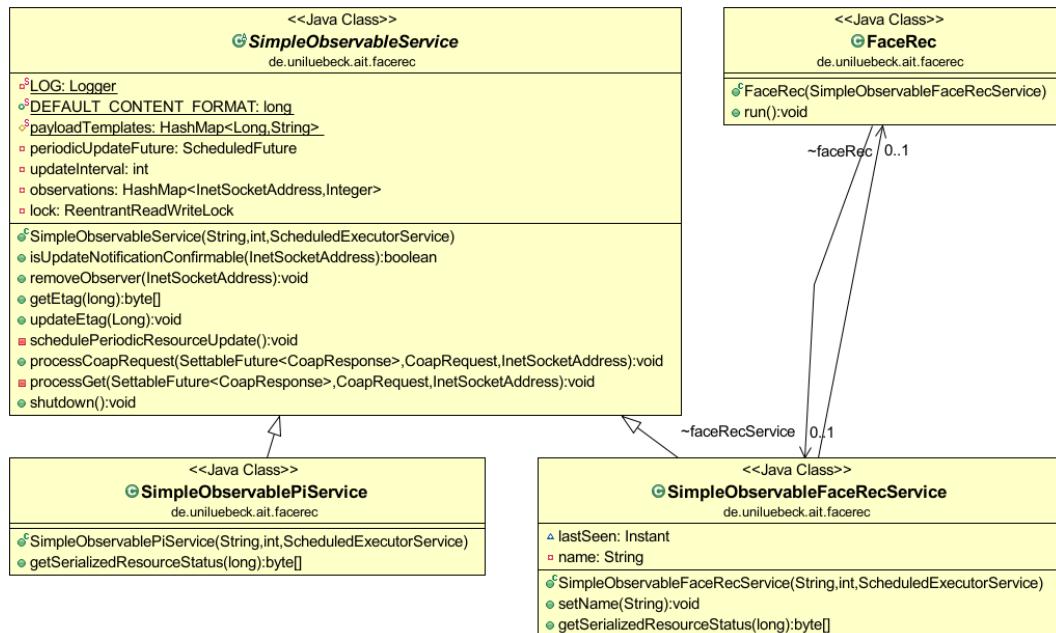


Figure 5.2: Klassendiagramm des Observable Service für die Gesichtserkennung

form von Kalendereinträgen und präferierten Nachrichtenfeeds. In Abbildung 5.3 werden die entsprechenden Services und deren Aggregation in der für die Darstellung zuständigen Klassen beschrieben.

Nachrichten

Wie in Abschnitt 2.2.4 beschrieben ist es möglich mittels RSS-Feeds automatisiert Aktualisierungen auf Websites zu verfolgen. In dieser Anwendung wird die Technologie benutzt um eine dem Nutzer zugeordnete Nachrichtenquelle abzurufen und die Informationen auf dem Display darzustellen. Die gewünschten Informationen werden in einer XML-Datei heruntergeladen und, wie in Abbildung 5.3 dargestellt, unter Benutzung des *javax.xml package* in die für die Darstellung relevanten Blöcke zerlegt.

5.1 Smart Mirror

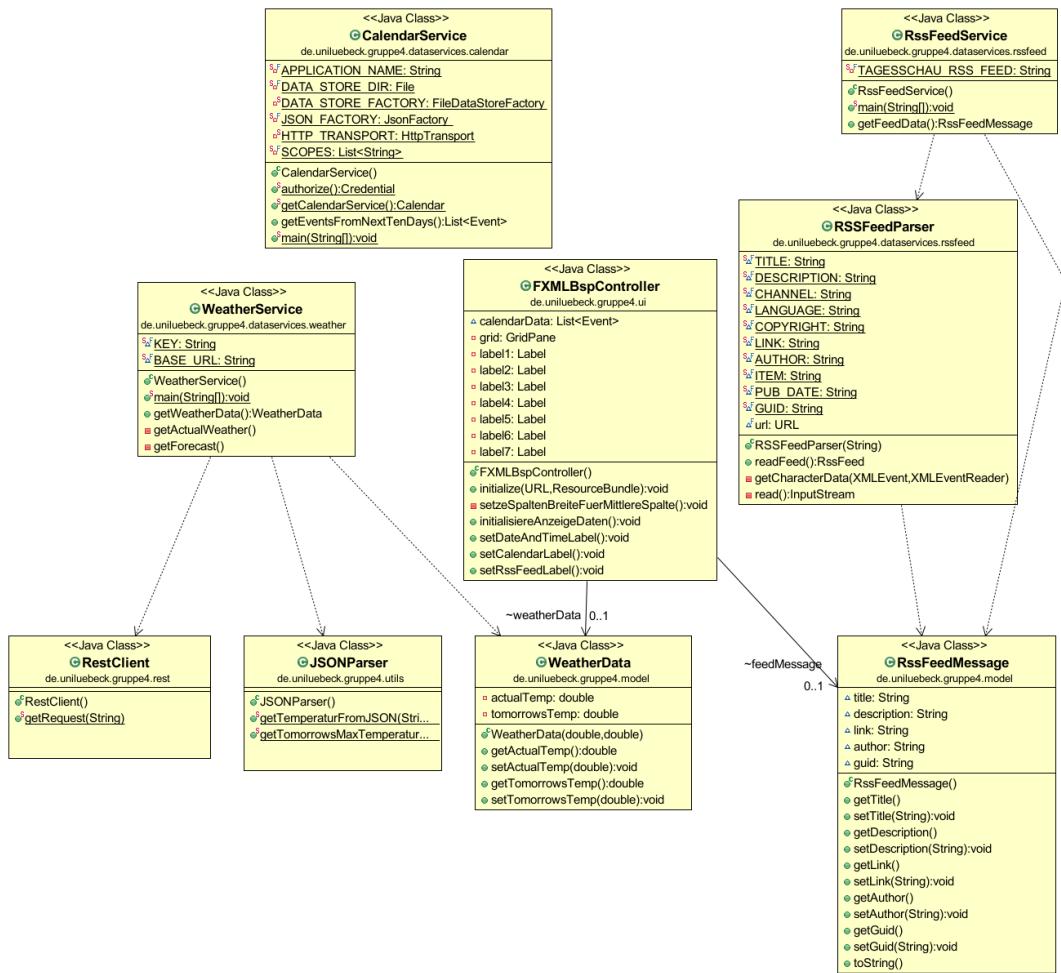


Figure 5.3: Klassendiagramm der Informationsservices mit Bezug auf die Darstellung im Display

Wetter

Nach erfolgreicher Registrierung bei dem Dienstleister Openweathermap⁵ wird ein Schlüssel bereitgestellt, welcher einen Zugriff auf deren API ermöglicht. Darüber können die benötigten lokalen Wetterdaten in Form einer *.json kostenfrei bezogen werden. In Codeabschnitt 5.4 werden sowohl der Request als auch die Zerlegung in relevante Blöcke, mittels *JSONParser* beschrieben.

Kalender

Um die Darstellung der Kalenderinformationen zu implementieren haben wir uns entschieden die entsprechende Google-API⁶ zu nutzen. In diesem Rahmen wurde ein Benutzerkonto für *Max Mustermann* angelegt. Der Zugriff auf die entsprechenden daten erfordert die Erstellung eines *Credential Objects*, der Vorgang ist in Codeabschnitt 5.5 dargestellt.

5.1.3 Darstellung

Die UI befindet sich noch in der Entwicklung. Sie wird mittels JavaFx realisiert. Den aktuellen Stand zeigt Figure ??, anhand derer man sich eine grobe Vorstellung holen kann, wie das Ganze aufgebaut ist. Aktuell sind 3 Anzeige-Flächen für Informationen geplant: Die linke Spalte, für das aktuelle Datum, die Uhrzeit und anstehende Termine aus dem Google-Kalender, die rechte Spalte für Wetter-Informationen und eine Fußzeile mit einem Auszug der aktuellsten Nachrichten. In der Mitte wird dann platz gelassen für das Spiegelbild. Hierzu wurde als Layout-Element von JavaFx der BorderPane gewählt, welcher bereits in diese Left, Right, Top, Bottom und Center Regionen aufgeteilt ist (siehe Figure ??).

⁵openweathermap.org, Abschnitt 5.1.2

⁶<https://developers.google.com/google-apps/calendar/>

5.1 Smart Mirror



Figure 5.4: Bisheriger Stand der UI

Listing 5.2: Java Code zum starten und auslesen des Python Scripts

```
1 public void run() {
2     try {
3         System.out.println("Start FaceRec");
4         ProcessBuilder pb = new ProcessBuilder("python", "-u", "/
5             home/pi/face_recognition/facerec.py");
6         pb.redirectOutput(Redirect.PIPE);
7         Process p = pb.start();
8         BufferedReader reader = new BufferedReader(new
9             InputStreamReader(p.getInputStream()));
10        String line = null;
11
12        while (true) {
13            try {
14                p.exitValue();
15                break;
16            } catch (Exception e) {
17            }
18            if ((line = reader.readLine()) != null) {
19                faceRecService.setName(line);
20            }
21        }
22        System.out.println("Stop FaceRec");
23    } catch (IOException e) {
24        e.printStackTrace();
25    }
26}
```

Listing 5.3: RRSFeedParser Klasse - Auslesen der Informationen aus empfangenem XML Dokument

```

1 // First create a new XMLInputFactory
2     XMLInputFactory inputFactory = XMLInputFactory.
3         newInstance();
4     // Setup a new eventReader
5     InputStream in = read();
6     XMLEventReader eventReader = inputFactory.
7         createXMLEventReader(in);
8     // read the XML document
9     while (eventReader.hasNext()) {
10         XMLEvent event = eventReader.nextEvent();
11         if (event.isStartElement()) {
12             String localPart = event.asStartElement().getName
13                 ().getLocalPart();
14             switch (localPart) {
15                 case ITEM:
16                     if (isFeedHeader) {
17                         isFeedHeader = false;
18                         feed = new RssFeed(title, link,
19                             description, language,
20                             copyright, pubdate);
21                     }
22                     event = eventReader.nextEvent();
23                     break;
24                     ...
25             }
26         } else if (event.isEndElement()) {
27             if (event.asEndElement().getName().
28                 getLocalPart() == (ITEM)) {
29                 RssFeedMessage message = new
30                     RssFeedMessage();
31                 message.setAuthor(author);
32                 message.setDescription(description);
33                 message.setGuid(guid);
34                 message.setLink(link);
35                 message.setTitle(title);
36                 feed.getMessages().add(message);
37                 event = eventReader.nextEvent();
38                 continue;
39             }
40         }
41     }
42 } catch (XMLStreamException e) {
43     throw new RuntimeException(e);
44 }
45 return feed;
46 }
```

Listing 5.4: WeatherService Klasse zum Abrufen der aktuellen Wetterlage sowie Vorhersage für eine Region

```
1 package de.uniluebeck.gruppe4.dataservices.weather;
2
3 import de.uniluebeck.gruppe4.model.WeatherData;
4 import de.uniluebeck.gruppe4.rest.RestClient;
5 import de.uniluebeck.gruppe4.utils.JSONParser;
6
7 public class WeatherService {
8
9     static final String KEY = "appid=
10         da7c4d55907d8cff81b1e5a02bae88e6";
11     static final String BASE_URL = "http://api.
12         openweathermap.org/data/2.5/";
13
14     public static void main(String[] args) {
15         WeatherService service = new WeatherService();
16         WeatherData data = service.getWeatherData();
17     }
18
19     public WeatherData getWeatherData(){
20         String actualWeather = getActualWeather();
21         String forecast = getForecast();
22
23         double actualTemp = JSONParser.getTemperaturFromJSON(
24             actualWeather);
25         double tomorrowTemp = JSONParser.
26             getTomorrowsMaxTemperaturFromJSON(forecast);
27
28         WeatherData weatherData = new WeatherData(actualTemp,
29             tomorrowTemp);
30
31         return weatherData;
32     }
33
34     private String getActualWeather() {
35         return RestClient.getRequest(BASE_URL + "weather?q=
36             London,uk&" + KEY);
37     }
38
39     private String getForecast(){
40         return RestClient.getRequest(BASE_URL + "forecast?q=
41             London,uk&" + KEY);
42     }
43 }
```

Listing 5.5: CalendarService Klasse - Authorisierung des jeweiligen Anwenders

```

1 /**
2  * Creates an authorized Credential object.
3  * @return an authorized Credential object.
4  * @throws IOException
5 */
6 public static Credential authorize() throws
7 IOException {
8     // Load client secrets.
9     InputStream in =
10        CalendarService.class.getResourceAsStream("/" +
11            "client_secret.json");
12     GoogleClientSecrets clientSecrets =
13         GoogleClientSecrets.load(JSON_FACTORY, new
14             InputStreamReader(in));
15
16     // Build flow and trigger user authorization
17     // request.
18     GoogleAuthorizationCodeFlow flow =
19         new GoogleAuthorizationCodeFlow.Builder(
20             HTTP_TRANSPORT, JSON_FACTORY,
21                 clientSecrets, SCOPES)
22             .setDataStoreFactory(DATA_STORE_FACTORY)
23             .setAccessType("offline")
24             .build();
25
26     // Credential credential = new
27     //     AuthorizationCodeInstalledApp(
28     //         flow, new LocalServerReceiver()).authorize("user");
29
30     Credential credential = new
31         AuthorizationCodeInstalledApp(
32             flow, new LocalServerReceiver()).
33                 authorize("Max_Mustermann");
34
35     System.out.println(
36         "Credentials saved to " + DATA_STORE_DIR.
37             getAbsolutePath());
38
39     return credential;
40 }
```

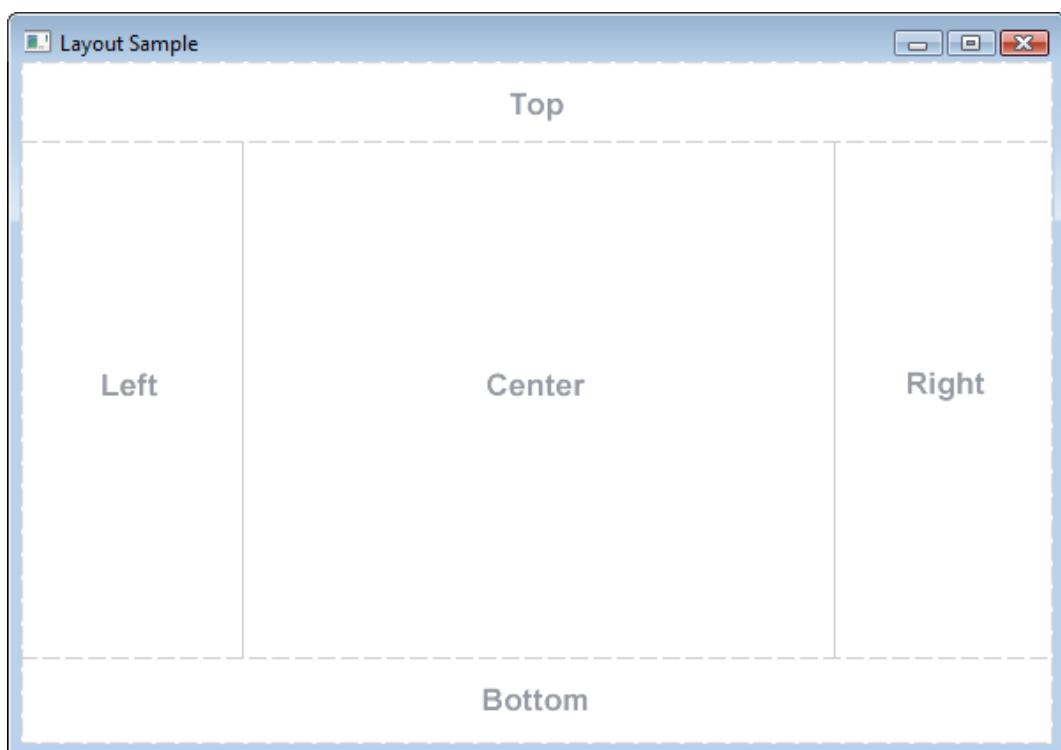


Figure 5.5: BorderPane aus JavaFx

Listing 5.6: FXMLBspController Klasse - Definiert das Layout auf dem Spiegel und nutzt alle Service-Klassen um die relevanten Informationen abzurufen

```

1  public void setDateAndTimeLabel() {
2      DateFormat formatter = new SimpleDateFormat("EEEE, dd
3          MMMM yyyy");
4      String dateString = formatter.format(new Date());
5      label1.setText(dateString);
6
7      DateFormat timeFormatter = new SimpleDateFormat("hh:
8          mm");
9      String timeString = timeFormatter.format(new Date());
10     label2.setText(timeString);
11 }
12
13 public void setCalendarLabel(){
14     StringBuilder calendarString = new StringBuilder();
15     if(CollectionUtils.isNotEmpty(calendarData)){
16         for (Event event : calendarData) {
17             DateFormat formatter = new SimpleDateFormat("dd.
18                 MM hh:mm");
19             DateTime start = event.getStart().getDateTime();
20
21             if (event.getStart().getDateTime() == null) {
22                 start = event.getStart().getDate();
23                 formatter = new SimpleDateFormat("dd.MM");
24             }
25             Date date = new Date(start.getValue());
26
27             calendarString.append(formatter.format(date) + " "
28                 + event.getSummary() + " " + System.
29                 getProperty("line.separator"));
30         }
31     }else{
32         calendarString.append("Keine anstehenden Termine");
33     }
34     label3.setText(calendarString.toString());
35 }
36
37 public void setRssFeedLabel(){
38     RssFeedService rssFeedService = new RssFeedService()
39         ;
40     feedMessage = rssFeedService.getFeedData();
41
42     label4.setText(feedMessage.getTitle());
43     label5.setText(feedMessage.getDescription());
44 }

```


List of Figures

2.1	Versionen der durch das Institut für Telematik bereitgestellten Raspberry Pi	4
	https://www.arduino.cc/en/Main/arduinoBoardUno	
2.2	BCM-Numbering der GPIO Ports am Raspberry Pi	5
	www.raspberrypi-spy.co.uk	
2.3	Layout Arduino Uno	6
	www.robomart.com/blog/wp-content/uploads/2015/07/Arduino_Components_layout-1024x675.png	
2.4	Coap Nachrichtenformat	7
	https://tools.ietf.org/html/rfc7252	
3.1	Arbeitspunkt und Reihenschaltung einer Leuchtdiode	12
	https://kingbright-europe.de/	
3.2	Realisierung der Verbindung des LDR mit dem Raspberry Pi	13
3.3	Spannungsteiler mit Abgriff am Fotowiderstand	15
3.4	Widerstandscharakteristik LDR mit Linearisierungsgeraden	16
	https://kingbright-europe.de/	
4.1	Turtle Ontology zur Verwendung und Erweiterung im Rahmen des Projekts	22
	https://moodle.uni-luebeck.de/pluginfile.php/112836/mod_resource/content/7/handbuch.pdf	
5.1	Darstellung Smart Mirror	30
	https://community.openhab.org/t/smooth-mirror-and-openhab/3611/3	
5.2	Klassendiagramm des Observable Service für die Gesichtserkennung	34

List of Figures

5.3 Klassendiagramm der Informationsservices mit Bezug auf die Darstellung im Display	35
-------------------------------------------------------------------------------------------------	----

Listings

3.2	Auslesen des GPIOs mittels WiringPi	13
3.1	Ansteuerung einer LED mittels GPIO	17
3.3	Auslesen des analogen LDR Widerstandswertes und Umrechnung in Lux	18
3.4	Initialisierung der seriellen Schnittstelle im Raspberry Pi, Quelle: http://playground.arduino.cc/Interfacing/Java	19
3.5	Auslesen der seriellen Schnittstelle im Raspberry Pi, Quelle: http://playground.arduino.cc/Interfacing/Java	20
4.1	SimpleObservableTimeService Beschreibung der Payload	23
4.2	Konstruktor mit Ressourcenpfad und Updateintervall	24
4.3	Updatefunktion des Services	24
4.4	Definition der SSP Schnittstelle	25
4.5	Registrierung am SSP	25
4.6	SimpleObservableLuxService Beschreibung der Payload	26
4.7	SPARQL-Query zur Ermittlung des durchschnittlichen Lux-Wertes .	27
5.1	Python Skript für Gesichtserkennung mithilfe von face_recognition .	31
5.2	Java Code zum starten und auslesen des Python Scripts	37
5.3	RRSFeedParser Klasse - Auslesen der Informationen aus empfangenem XML Dokument	38
5.4	WeatherService Klasse zum Abrufen der aktuellen Wetterlage sowie Vorhersage für eine Region	39
5.5	CalendarService Klasse - Authorisierung des jeweiligen Anwenders .	40
5.6	FXMLBspController Klasse - Definiert das Layout auf dem Spiegel und nutzt alle Service-Klassen um die relevanten Informationen abzurufen	41

List of Tables

2.1	LED Flussspannungen	9
5.1	Struktur Smart Mirror	30

List of Corrections