DAT-Projekt Lichtsteuerung

Marius **Schuller**
Stefan **Thiemann**
Patrick **Wildt**

18. Januar 2016

Inhaltsverzeichnis

1 Einführung

Im Zuge der Internet-of-Things-Kampagne[1] werden immer mehr "dumme" bzw. einfache Geräte miteinander vernetzt und die resultierenden Daten intelligent miteinander verknüpft. Dazu gehören auch Lichter und Glühbirnen. Zur Vernetzung und Steuerung der Lichter existieren bereits mehrere aktuelle Technologien. Mit Hilfe einer der standardisierten Technologien möchten wir einen Controller implementieren, der diese Lichter kontrollieren kann.

2 Lichtsteuerungs-Technologien

Üblicherweise möchten Hersteller ein eigenes Produkt-Ökosystem erstellen, aus dem ein Anwender nicht oder nur schwer entkommen kann. Hierfür werden von den Herstellern eigene, unfreie Protokolle implementiert. Beispielsweise bietet *LimitlessLED*[2] Glühbirnen, welche sich über 2,4 GHz WLAN in das lokale Netzwerk verbinden. Für die eigentliche Steuerung wurde dazu eine eigene API entwickelt. Eine weitere bekannte Technologie zur Steuerung von Geräten ist Bluetooth. Hier ist es derzeit möglich mit Hilfe des Generic Attribute Profile (GATT), ein eigenes Protokoll zu sprechen. Dies wird bei einigen smarten Glühbirnen verwendet um ein proprietäres Lichtsteuerungsprotokoll zu implementieren.

Die Bluetooth Konkurrenten *Z-Wave*[3] sowie *ZigBee* implementieren wieder jeweils eigene Lichtprotokolle. Diese sind jedoch für jeden Client des Funkstandards nutzbar, sodass die Hersteller kein eigenes Protokoll implementieren mussten. Der Funkstandard ZigBee wird von den namhaften Herstellern *Philips* und *Osram* verwendet.

Für das Schwerpunktprojekt würden wir uns auf ZigBee kompatible Geräte konzentrieren. Vor allem die Produkte der *Philips hue*[4] Reihe.

[1] `http://www.nextgenerationmedia.de`
[2] `http://www.limitlessled.com`
[3] `http://www.z-wavealliance.org`
[4] `http://www.philips.de/e/hue/hue.html`

3 Komponenten

Die eigentliche Logik zur Steuerung der Lichter kann auf einem *RPi* implementiert werden. Um den Funkstandard ZigBee sprechen zu können wird ein kompatibles Funkmodul benötigt. Hierfür kann das *RaspBee*-Modul verwendet werden. Dieses gibt es in zwei Varianten, *Basic* und *Premium*. Während man mit der Basic-Variante nur mit 5 Knoten sprechen darf, ist dies bei der Premium-Variante unbegrenzt. Die Lichter würden aus einem *Philips Hue* Starterkit bestehen.

4 Hardware

4.1 RaspBee Premium, Raspberry-Pi Einzeln

Menge	Produkt	Einzelpreis	Gesamtpreis
3	RaspberryPi 2	42 Euro	126 Euro
3	RaspBee Premium	60 Euro	180 Euro
3	Philips Hue LED 1 x 9W A60 E27	59 Euro	177 Euro
Gesamtpreis			**483 Euro**

4.2 RaspBee Premium, Raspberry-Pi Bundle

Menge	Produkt	Einzelpreis	Gesamtpreis
3	RaspberryPi 2 Bundle	70 Euro	210 Euro
3	RaspBee Premium	60 Euro	180 Euro
3	Philips Hue LED 1 x 9W A60 E27	59 Euro	177 Euro
Gesamtpreis			**567 Euro**

4.3 Hinweis

Unter Umständen sind Bestandteile der Liste schon im Vorrat der Hochschule oder der Projektteilnehmer. Je nach Beteiligung der Fachhochschule würden wir für einen Teil der Kosten aufkommen.

5 Grobarchitektur

In Abbildung 1 wird der grobe, vorraussichtliche Ablauf der Kommunikation aller involvierter Komponenten beschrieben.

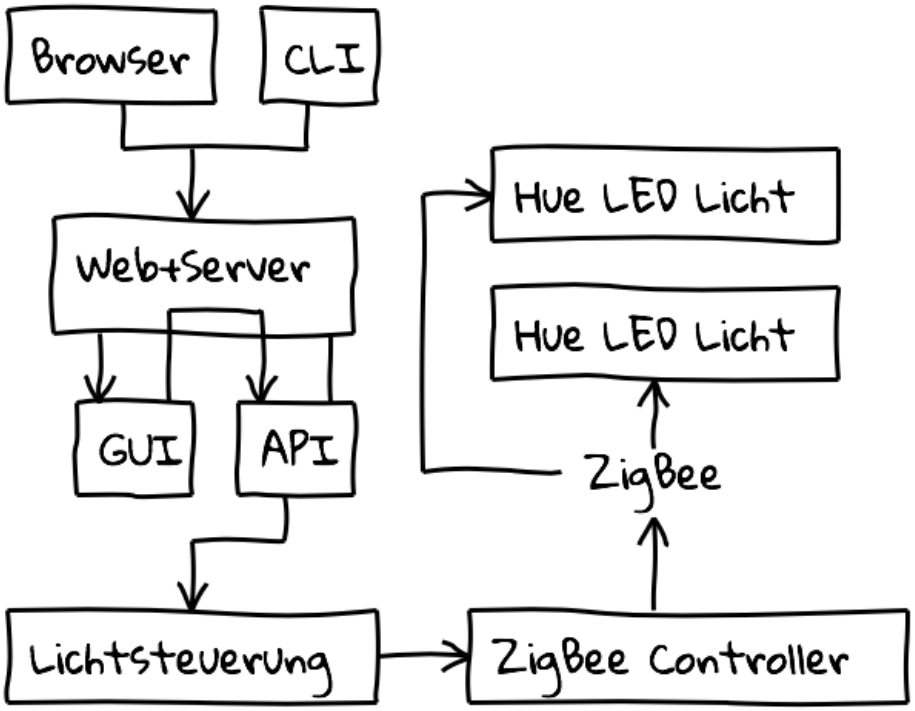

Abbildung 1: Kommunikationsablauf der Komponenten

Die Lichtsteuerung soll aus mehreren miteinander interagierenden Komponenten bestehen, die im Folgenden genauer beschrieben werden.

5.1 Hue LED Licht

Die zu steuernden Lampen werden in eine herkömmliche Fassung geschraubt worüber sie wie normale Leuchtmittel mit Strom versorgt werden. Die Hue LEDs besitzen dazu aber einen ZigBee-Chip, mit dem sie Teil eines ZigBee-Netzwerks werden können. In diesem Netzwerk fungieren sie als *End Device*, das bedeutet, sie nehmen nicht am Routing innerhalb des Netzwerks teil, benötigen dafür aber auch nur einen kleinen Teil der Funktionen des ZigBee-Standards. Über das ZigBee *Light Link*-Protokoll[5] können die Lampen angesprochen werden und bestimmte Einstellungen wie Helligkeit und Farbe eingestellt werden.

5.2 ZigBee Controller

Der ZigBee Controller RaspBee ist die eigentliche Funkeinheit. Sie stellt eine rohe Programmierschnittstelle bereit, um auf das ZigBee-Netzwerk zugreifen zu können.

Abbildung 2: RaspBee-Funkmodul

Der Controller besteht aus zwei Komponenten. Zum einen dem RaspBee, einer aufsteckbaren Erweiterungsplatine mit Funkmodul für den RPi, und

[5] `http://www.zigbee.org/zigbee-for-developers/applicationstandards/zigbee-light-link/`

zum anderen dem RPi selber. Der RPi besitzt eine Reihe an GPIO-Pins am Rand des Boards.

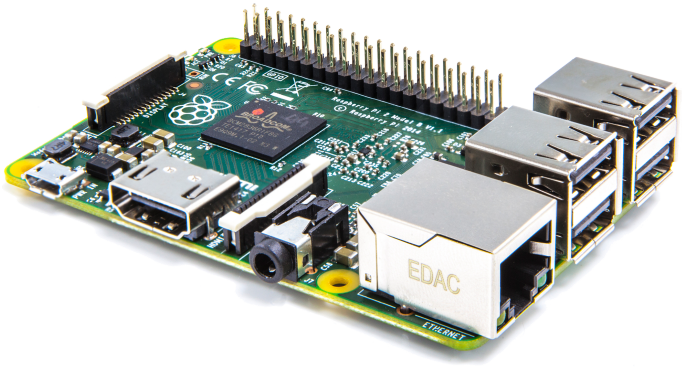

Abbildung 3: Raspberry 2

Das RaspBee ist an diese GPIO-Pins angepasst und wird dadurch mit Strom versorgt. Außerdem werden die UART-Pins zur seriellen Kommunikation mit einem Treiber, der auf dem RPi betrieben wird, verwendet.

5.3 Lichtsteuerung

Die Lichtsteuerung soll als hardwarenahes Backend dienen. Diese Software, die auf dem RPi betrieben wird, kommuniziert über die serielle Schnittstelle mit dem ZigBee Controller. Um auf die Nodes des Netzwerks zugreifen zu können bietet das Backend eine Programmierschnittstelle an.

5.4 API

Die API ist eine weitere Software auf dem RPi. Sie verwendet die Schnittstelle der Lichtsteuerung und bietet eine REST-basierte Webschnittstelle an. Diese Schnittstelle bietet einen vereinfachten Zugriff auf das Funknetz, mit Fokus auf Steuerung und Verwaltung der Lampen. Durch die Trennung der einzelnen Programme bleiben Treiber, API und GUI austauschbar.

5.5 Webserver

Der Webserver dient primär zum Zugriff auf die Weboberfläche, welche die REST-API über den Browser zugänglich macht um bequem die Lichter verwalten und steuern zu können. Weiter wird hier die API etwaigen CLI-Programmen zur Verfügung gestellt.

5.6 GUI

Die GUI, im HTML5 Standard, ist die Schnittstelle zum Benutzer und wird im Browser dargestellt. Über diese kann der Benutzer die Nodes steuern, wie z.B. Helligkeit und Farbe einstellen. Zum bequemen Verwenden der API aus dem Browser wird JavaScript auf dem Client verwendet.

5.7 CLI

Um die Lampensteuerung automatisieren zu können, etwa um aus dem Urlaub oder zeitgesteuert in einem Haus Licht aus- oder anzuschalten, ist ein Kommandozeileninterface wünschenswert. Für viele Sprachen gibt es bereits Module mit denen REST-APIs angesprochen werden können, daher ist die Programmiersprache in der das CLI-Programm umgesetzt werden soll zweitrangig.

6 Umsetzung

Umgesetzt und von uns implementiert werden soll die API sowie die CLI. Eine rudimentäre Umsetzung der Web-GUI ist denkbar, um die prinzipielle Möglichkeit zu verdeutlichen. Die Treibersoftware um das RaspBee-Modul anzusteuern ist leider nicht Open Source. Diese Software bietet eine Programmierschnittstelle, um die eine API herum entwickelt werden kann.

6.1 Einschränkungen

Nach anfänglicher Exploration der mit der RaspBee-Modul zugehörigen Software von *Dresden Elektronik* wurde festgestellt, dass viel der von uns zu implementierenden Funktionalität bereits in der bereitgestellten *deCONZ*-Software vorhanden war. Die Anfrage an den Support von Dresden Elektronik nach Zugang zur Dokumentation der UART-Schnittstelle und dem Funkmodul RaspBee wurde verweigert.

Weiter war das Vorhaben aus Punkt 5 inhaltlich zu groß gefasst als dass es in der gegebenen Zeit vollumfänglich durchgeführt werden hätte können. Aus diesen Gründen beschränkt sich das Projekt auf die Entwicklung eines CLI.

7 Installation

7.1 Betriebssystem

Das im RPi Bundle aus Punkt 4 auf der Micro-SD-Karte vorinstallierte System heißt NOOBS (New Out Of the Box Software). Dieses System dient als Platform um das eigentliche System zu installieren. Es stellt eine grafische Oberfläche auf der HDMI-Schnittstelle bereit auf der das über das Internet zu installierende Betriebssystem ausgewählt werden kann.

Da die deCONZ-Software nur in Binärform vorliegt muss ein daran angepasstes Betriebssystem installiert werden. *Raspbian*[6], eine für das RPi an-

[6] `https://www.raspbian.org/`

gepasste *Debian*[7] Version, ist in der Version Wheezy zur deCONZ-Software kompatibel. Weiterhin ist Raspbian eines der verbreitetsten Betriebssysteme für den RPi. Wahlweise kann Raspbian auch manuell installiert werden. Dafür muss ein Image[8] der Version Wheezy heruntergeladen werden. Das Image ist komprimiert und muss nach dem Entpacken auf die SD-Karte gespielt werden. Das kann mit folgenden Kommandos ausgeführt werden:

```
# unzip 2015-05-05-raspbian-wheezy.zip
# dd if=2015-05-05-raspbian-wheezy.img of=/dev/disk
```

Listing 1: Raspbian manuell installieren

Der RPi bekommt automatisch von einem im Netzwerk verbundenen DHCP-Server eine IP-Adresse. Diese kann nun entweder über die grafische Oberfläche abgefragt oder aus dem DHCP-Server ausgelesen werden. Mit Hilfe dieser Information ist es möglich sich auf den automatisch gestarteten SSH-Server einzuloggen.

7.2 deCONZ

Die aktuelle deCONZ-Version[9] liegt als Debian Paket auf der Website des Herstellers bereit. Nach dem Download kann die Software mit dpkg installieren. Danach ist die Software betriebsbereit.

```
# dpkg -i deconz-2.02.05.deb
```

Listing 2: deCONZ installieren

7.3 Vorbereitung für die Nutzung der CLI auf RPi

Um möglichst wenige externe Abhängigkeiten zu haben, wird auf die über Advanced Packaging Tool (APT) verfügbare, alte Ruby-Version 1.9.3 zurückgegriffen. Diese ist in APT fehlerhafterweise mit der Versionsnummer 1.9.1 gekennzeichnet. Letztlich sollten aber beide Versionen gleich gut funktionieren. Um wichtige Funktionalität nicht selbst neu entwickeln zu müssen wird

[7] https://www.debian.org/
[8] https://www.raspberrypi.org/downloads/raspbian/
[9] http://www.dresden-elektronik.de/rpi/deconz/deconz-2.02.05.deb

für die REST-Anfragen das Ruby-Modul *rest-client*[10], in der Ruby-Welt als *Gem* bezeichnet, dafür verwendet. Für einfaches Handling der Kommandozeilenparameter nutzen wir *clamp*[11].

```
# aptitude install ruby ruby-dev rubygems
# gem install rest-client clamp --no-ri --no-rdoc
```

Listing 3: Ruby und Ruby-Modulmanager installieren

8 deCONZ Bedienung

Bei der Verwendung von deCONZ werden zwei Bedienoberflächen bereitgestellt. Eine davon wird als Desktop-Applikation angezeigt. Dort kann sich auf einer hardwarenahen Ebene verfügbare ZigBee-Nodes lassen. Weiter wird eine Weboberfläche angeboten, über die die eigentliche Lampen-Steuerung erfolgt. Hiermit können Gruppen — Verbünde aus mehreren Lampen — und Szenen — eigene, gespeicherte Konfigurationen für Lichter und Gruppen — definiert und angewählt werden. Im Hintergrund wird eine bereitgestellte REST-API verwendet, über die auch unser zu programmierendes CLI auf die Hardware und Daten zugreifen soll.

8.1 deCONZ starten

Um deCONZ zu starten muss zwangsweise ein X-Server aktiv sein. Allerdings wird diese Oberfläche hauptsächlich zur Fehlersuche verwendet und wird von uns nicht direkt benötigt. Aus diesem Grund starten wir den X-Server nur im Hintergrund und starten die GUI-Applikation dort. deCONZ lauscht mit der, zusammen mit der GUI gestarteten, Weboberfläche auf Port 80. Optional kann ein anderer Port angegeben werden.

```
$ startx &
$ DISPLAY=:0 deCONZ --auto-connect=1 &
```

Listing 4: deCONZ starten

[10] https://rubygems.org/gems/rest-client
[11] https://rubygems.org/gems/clamp

8.2 Web-GUI

Das deCONZ Programm stellt neben der Schnittstelle zur Hardware auch eine Webseite zur Verfügung, über welche die API angesprochen werden kann. Standardmäßig ist diese unter `http://<IP>:80` zu erreichen, wobei <IP> ein Platzhalter für die private IP-Adresse des RPi steht. Über die Web-GUI können Lichter hinzugefügt, Gruppen und Szenen angelegt, verwaltet sowie gelöscht werden. Eine Automatisierungsmöglichkeit über die Web-GUI ist ebenfalls vorhanden.

8.3 API

Nach dem Starten der deCONZ Software wird neben einem Webserver (vgl. Kapitel 8.2) auch eine REST-API zur Verfügung gestellt, die unter der Adresse `http://<IP>:80/api` erreichbar ist.[12]

Über diese Schnittstelle agiert `lolo` mit der deCONZ-Software, um nahezu dieselbe Funktionalität der Web-GUI auf der Kommandozeile bereitzustellen.

[12] API-Doku unter `http://dresden-elektronik.github.io/deconz-rest-doc/`

9 lolo - Lights on, Lights off

9.1 Beispielszenario

In den folgenden Beispielen wird mit folgendem Datenbestand gearbeitet:
Lichter `L1` und `L2`
Gruppen `G1` und `G2`
Szene `S1`

9.2 Ausgabe der Hilfe

```
$ lolo
Usage:
    lolo.rb [OPTIONS] SUBCOMMAND [ARG] ...

Parameters:
    SUBCOMMAND      subcommand
    [ARG] ...       subcommand arguments

Subcommands:
    list            List all lights and groups.
    update          Update cache of lights, groups and scenes.
    light           Change a specific light.
    group           Change a specific group.
    scene           Change a specific scene.
    add             Add a group or a light to a group.
    delete          Delete a group or scene.

Options:
    -d              Enable debug output
    -h, --help      print help
```

Listing 5: Hilfe von `lolo`

9.3 Auflisten des Datenbestands

```
$ lolo list
Lights:
  1    L1
Groups:
  1    G1   Lights: []
Scenes:
  S1  G1
```

Listing 6: Abfrage aller Lichter, Gruppen und Szenen

9.4 Lichtsteuerung

```
# Licht L1 einschalten und Farbe setzen
$ lolo light L1 on
$ lolo light L1 red
$ lolo light L1 cold
$ lolo light L1 off
```

Listing 7: lolo Beispielaufrufe

9.5 Gruppen und Szenen

```
# Erstellen einer neuen Gruppe
$ lolo add group G1

# Licht L2 wird angelegt und zur Gruppe G1
# hinzugefuegt
$ lolo add light L2 <UUID> G1

# Cache updaten
$ lolo update

# Gruppe G1 ausschalten
$ lolo group G1 off

# Gruppe G1 und Szene S1 loeschen
$ lolo delete group G1
$ lolo delete scene S1

# Neue Szene aus aktueller Einstellung von G1,
# L3 und L4, die nicht in G1 sind
$ lolo add scene S1 from G1 L1 L2

# Frauenbesuch aktivieren
$ lolo scene S1 on

# Frauenbesuch deaktivieren
$ lolo scene S1 off
```

Listing 8: Bedienung von Gruppen und Szenen

10 Ausblick