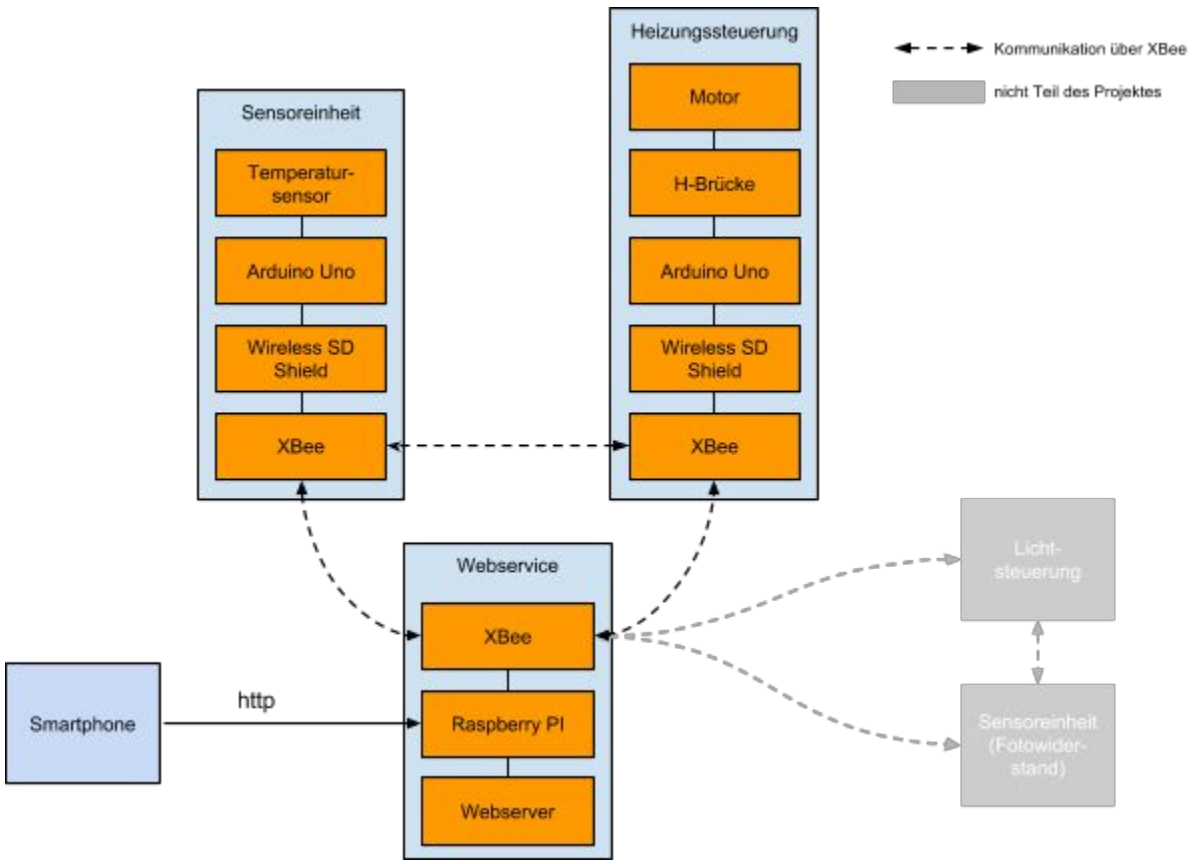
Hausautomatisierung mit ZigBee



Embedded Systems Sommersemester 2015

von Benjamin Chi, Julian Fahrer, Simon Linsner, Marko Radocaj

Inhaltsverzeichnis

1 Aufgabenbeschreibung

Die Aufgabe ist besteht darin, einen Prototypen für die Hausautomatisierung mit ZigBee als Kommunikationsmediun zu entwerfen und zu realisieren. Als konkretes Beispiel soll dabei eine Heizungssteuerung umgesetzt werden. Mittels eines Webservices soll es möglich sein Daten von Sensoren sowie Zustände von Steuereinheiten, wie zum Beispiel von Licht und Heizung, abzufragen. Außerdem soll es über den Webservice möglich sein die Steuereinheiten zu regeln. So, dass diese beispielsweise ein- und ausgeschaltet werden können und im Falle einer Heizung eine Soll-Temperatur gesetzt werden kann.

Es notwendig, die erforderlichen Komponenten zu ermitteln, ein Kommunikationsprotokoll zwischen Webservice, Steuereinheiten und Sensoreinheiten zu definieren, den Webservice zu implementieren sowie die Steuereinheiten und Sensoreinheiten zu bauen und zu programmieren.

2 Komponenten der Hausautomatisierung

In einem Hausnetzwerk sind folgende Komponente vertreten, die miteinander kommunizieren:

- **Steuereinheit:** Steuert einen Aktuator (z.B. Heizung, Licht, Rolladen)
- **Sensoreinheit:** Liefert auf Anfrage Sensormesswerte (z.B. Temperatur, Helligkeit)
- **Webservice:** Stellt alle Daten mittels einer hybriden REST-RPC JSON-API über HTTP für weitere Anwendungen zur Verfügung.

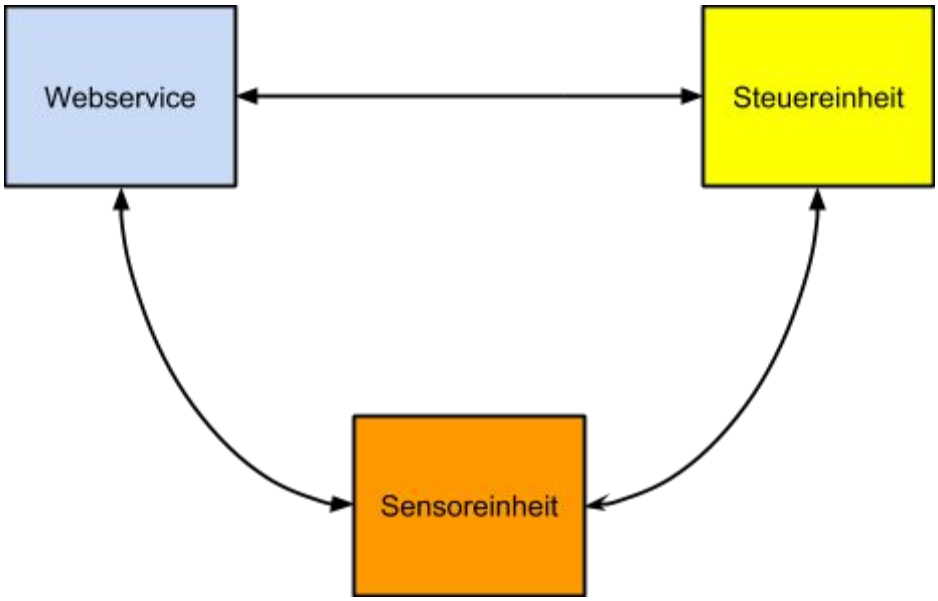


Abbildung 1

3 Kommunikation

Die Kommunikation zwischen den einzelnen Komponenten erfolgt über ZigBee. Jede Komponente hat dazu ein oder mehrere IDs für die Steuerung bzw. Abfrage der Sensoren. Eine ID kennzeichnet dabei eine Schnittstelle der Komponente über die Informationen abgefragt bzw. Befehle ausgeführt werden können. Beispielsweise kann eine Lampe zwei IDs besitzen. Eine für das An- uns Ausschalten und eine für die Vorgabe des Helligkeitswertes zum Dimmen. Es ist außerdem möglich einer Komponente weitere IDs für die gleiche Schnittstelle zuzuweisen, um so eine Gruppe von Geräten einheitlich zu steuern. Die ID *0* ist als Broadcast Adresse definiert. Antworten auf Statusanfragen und Statusänderungen erfolgen in der Regel an ID 0, damit alle Geräte die Statusänderung bei Bedarf verarbeiten können.

Die Kommunikation im ZigBee Netzwerk erfolgt ausschließlich mit Broadcasts[1][2]. Somit erhält jeder Teilnehmer alle Nachrichten. Auf jede gesendete Nachricht sollte der Empfänger dem Sender eine Rückmeldung schicken, die den angefragten bzw. geänderten Status als Wert enthält. Das betrifft z.B. die IST-Temperatur oder den aktuellen Status bei einer Schalthandlung. Sensoren können Ihren aktuellen Status auch ohne Anfrage versenden, um so z.B. Temperaturänderungen oder Störungen direkt zu melden.

[1] https://de.wikipedia.org/wiki/Broadcast
[2] http://knowledge.digi.com/articles/Knowledge_Base_Article/XBee-ZigBee-Addressing

3.1 Aufbau einer Nachricht

Jede Nachricht besitzt folgenden schematischen Aufbau:

Empfänger-ID	Sender-ID	Befehl	Wert (optional)

- **Empfänger-ID:** Die ID des Empfängers oder einer Gruppe
- **Sender-ID:** Die ID des Senders
- **Befehl:** Zahlenwert, der einen bestimmten Befehl kodiert (siehe Tabelle)
- **Wert:** Optional der mit dem Befehl verknüpfte Wert (z.B. Heizung auf Stufe 3 setzen)

Die einzelnen Teile der Nachricht werden als String kodiert und über das Trennzeichen | miteinander verknüpft.

3.1.1 Befehle

Code	**Erklärung**
1	Statusanfrage / einen Wert auslesen.
2	Statusantwort / Wertrückmeldung
3	Einen Wert setzen

3.2 Kommmunikationsbeispiel

Die Folgende Komponenten sind im im Netzwerk enthalten. Der Zusatz "ID" dient nur der Lesbarkeit und entfällt in den tatsächlichen Nachrichten.

Komponente	**Schnittstelle**	**ID**
Webservice	-	ID1
Temperaturfühler	IST-Temperatur	ID2
Heizungsregler	An/Aus	ID3
Heizungsregler	SOLL-Temperatur	ID4
Deckenlicht Wohnzimmer	An/Aus	ID5
Stehleuchte Wohnzimmer	An/Aus	ID6
Stehleuchte Wohnzimmer	Helligkeitswert	ID7

Folgende Gruppen existieren

ID	**Teilnehmer**
ID8	ID5, ID6

3.3 Beispiele

3.3.1 Heizungssteuerung

1. Der Webservice sendet eine Statusanfrage an die Heizung

ID3	ID1	1	

2. Der Heizungsregler antwortet mit seiner aktuellen Status (0 = aus)

ID1	ID0	2	0

3. Der Webservice schaltet die Heizung ein (1 = an)

ID3	ID1	3	1

4. Der Heizungsregler antwortet mit dem neuen Status

ID1	ID0	2	1

5. Der Webservice setzt die SOLL-Temperatur des Heizungsreglers auf 22°C

ID4	ID1	3	22.0

6. Der Heizungsregler antwortet mit dem neuen Status

ID4	ID0	2	22.0

3.3.2 Der Webservice holt sich die aktuelle IST-Temperatur

1. Der Webservice sendet eine Statusanfrage an den Temerpaturfühler

ID1	ID2	1	

2. Der Heizungsregler antwortet mit seiner aktuellen Status (0 = aus)

ID2	ID0	2	22.0

3.3.3 Der Temperatursensor versendet einen neuen Temperaturwert auf Grund einer Temperaturschwankung

1. Der Heizungsregler antwortet mit seiner aktuellen Status (0 = aus)

ID2	ID0	2	22.0

3.3.4 Die Heizungssteuerung fragt die aktuelle IST-Temperatur an

1. Der Heizungsregler antwortet mit seiner aktuellen Status (0 = aus)

ID2	ID3	1	

2. Der Temperaturfühler antwortet mit der aktuellen Temperatur

ID3	ID0	2	22.0

3.3.5 Lichtschalter

1. Der Webservice schaltet das Deckenlicht und die Stehleuchte im Wohnzimmer über die Gruppenadresse ein (1 = ein)

ID1	ID8	3	1

2. Der Stehleuchte antwortet mit ihrem aktuellen Status (1 = an)

ID6	ID0	2	1

3. Die Deckenleuchteantwortet mit ihrem aktuellen Status (1 = an)

ID5	ID0	2	1

4. Der Webservice setzt den Helligkeitswert der Stehleuche auf 50%

ID1	ID7	3	50

5. Die Stehleuchte antwortet mit ihrem neuen Helligkeitswert

ID7	ID0	2	50

4 Webservice API

Zur Steuerung und Statusabfrage Sensor- und Steuereinheiten stellt der Webservice eine API[3] bereit, über die von beliebigen Geräten per HTTP zugegriffen werden kann.

Der Zugriff auf die Haussteuerung kann über die breitgestellt hybride REST-RPC JSON-API[4] erfolgen. Die Befehle werden dabei eins zu eins auf die Haussteuerung umgesetzt. Lediglich der Absender wird in den API Aufrufennicht angegeben, sondern automatisch durch den Webservice festgelegt.

Es gibt zwei API Endpunkt. Das absetzen von Befehlen läuft über einen *POST Request*[5] gegen den API Endpunkt: http://host/api/controll

Die Abfrage der im Webservice zwischengespeicherten Statusinformationen laufen über einen *GET Request*[6] gegen http:///host/api/status?id=:id. :id repräsentiert dabei die ID. Beispielsweise http:///host/api/status?id=1 oder http:///host/api/status?id[]=1&id[]=2&id[]=3

Das Protokoll der Haussteuerung wird wie folgt in JSON kodiert

Protokoll	API	Datentyp
Empfänger-ID	recipient_id	String (entfällt bei Abfragen)
Sender-ID	device_id	String (entfällt bei Befehlen)
Befehl	action	String (entfällt bei Abfragen)
Wert	value	String (optional bei Befehlen)

4.1 Befehle absetzen

Für das absetzen eines Befehls für die Haussteuerung, muss dieser entsprechend in JSON kodiert werden. So kann Beispielsweise das für ID1 der Wert 45 gesetzt werden:

[3] https://de.wikipedia.org/wiki/Programmierschnittstelle
[4] https://cogo.wordpress.com/2008/01/15/restful-vs-rest-rpc-hybrid-vs-rpc-webservices/
[5] https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Argument.C3.BCbertragung
[6] https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Argument.C3.BCbertragung

```
[
  {
    "recipient_id": "1",
    "action": "3",
    "value": "45"
  }
]
```

Bei einem erfolgreichen Request wird der HTTP-Status Code[7] 200 zurückgegeben. Der Erfolg des Requests sagt nichts über den Erfolg des Befehls aus. Der Status muss dazu separat abgefragt werden. Der Response Body ist im Erfolgsfall leer.

Es können mehrere Befehle gleichzeitig gesendet werden:

```
[
  {
    "recipient_id": "1",
    "action": "3",
    "value": "45"
  },
  {
    "recipient_id": "2",
    "action": "3",
    "value": "1"
  },
  {
    "recipient_id": "1",
    "action": "2",
    "value": ""
  }
]
```

In einem Fehlerfall wird ein entsprechender Statuscode zurückgegeben und der Response Body enthält die Fehlerinformation:

[7] https://de.wikipedia.org/wiki/HTTP-Statuscode#2xx_.E2.80.93_Erfolgreiche_Operation

```
[
  {
    "errors": ["unable to send message"]
  }
]
```

Wurden mehrere Befehle gleichzeitig abgesetzt, wird ein Array mit gleicher Länge zurückgegeben. Bei erfolgreiche Befehle wird das Element im Array auf null gesetzt.

```
[
  null,
  {
    "errors": ["invalid id"]
  },
  null
]
```

4.2 Status abfragen

Der Status einzelner Komponentenschnittstellen können über den Webservice ausgelesen werden. Dazu wird ein *GET Request* aus den API Endpunkt http:///host/api/status gesendet. Die gewünschten IDs werden als Parameter angehängt. Beispielsweise http:///host/api/status?id=1 oder http:///host/api/status?id[]=1&id[]=2&id[]=3.

Es wird ein Array mit den Werten der angefragten Schnittstellen zurückgegeben:

```
[
  {
    "device_id": "1",
    "value": "45"
  },
  {
    "device_id": "2",
    "value": "1"
  }
]
```

Die Reihenfolge ist dabei Identisch mit der Reihenfolge der Parameter.

Bei unbekannten Status von Schnittstellen wird `null` als Wert zurückgegeben:

```
[
  {
    "device_id": "99”,
    "value": null
  }
]
```

Der HTTP-Statuscode für Antworten ist 200.

5 Aufbau

Abbildung 2 stellt die Komponenten im ZigBee Netzwerk und deren IDs gemäß der Protokolldefinition dar.

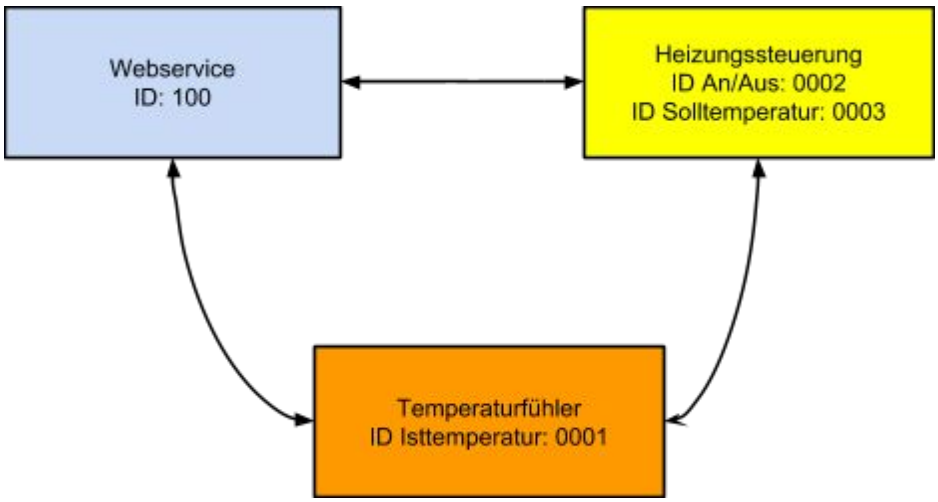


Abbildung 2

6 Verwendete Komponenten

Bezeichnung	Anzahl	Beschreibung
RaspBee premium	1	ZigBee Modul für den Raspberry PI. Nicht im Einsatz. https://shop.dresden-elektronik.de/raspbee.html
NIBO UCOM-XBEE	3	Als Interface für die Konfiguration und Tests der XBee Module sowie als ZigBee Anbindung des Webservices. http://www.reichelt.de/NIBO-UCOM-XBEE/3/index.html?ACTION=3&GROUPID=3644&ARTICLE=

		121924&START=0&OFFSET=16&
Raspberry PI 2 Model B	1	Als Plattform für den Webservice. http://www.reichelt.de/RASPBERRY-PI-2-B/3/index.html?&ACTION=3&LA=446&ARTICLE=152728&artnr=RASPBERRY+PI+2+B&SEARCH=raspberry+b%2B
Maxstream XBee RF Modul 802.15.4 SERIES 2	5	ZigBee Module für die Kommunikation der Teilnehmer. http://www.reichelt.de/ZIGBEE-XBEE/3/index.html?&ACTION=3&LA=446&ARTICLE=74768&artnr=ZIGBEE+XBEE&SEARCH=xbee+series+2
ARDUINO UNO	2	Als Plattform für Sensor- und Steuereinheit des Heizungsreglers http://www.reichelt.de/ARDUINO-UNO/3/index.html?&ACTION=3&LA=446&ARTICLE=119045&artnr=ARDUINO+UNO&SEARCH=arduino+uno
Arduino Shield Wireless SD	2	Für die Anbindung der XBee Module http://www.reichelt.de/?ARTICLE=130163&PROVID=2788&wt_mc=amc141526782519998&&gclid=CKainvqJ5MQCFTDKtAod_CIA4Q
XLIPO C2 2700	1	Stromversorgung http://www.reichelt.de/XLIPO-C2-2700/3/index.html?&ACTION=3&LA=446&ARTICLE=120105&artnr=XLIPO+C2+2700&SEARCH=XLIPO+C2+2700
AD 22100 KT	1	Temperatursensor http://www.reichelt.de/Temperatursensoren/AD-22100-KT/3/index.html?&ACTION=3&LA=2&ARTICLE=39279&GROUPID=6672&artnr=AD+22100+KT
L 293 D :: 4-Ch-Driver mit Diode	1	Ansteuerung und Stromversorgung des Motors http://www.reichelt.de/L-293-D/3/index.html?&ACTION=3&LA=446&ARTICLE=9661&artnr=L+293+D&SEARCH=L+293+D
GS 16 :: IC-Sockel, 16-polig, doppelter Federkontakt	1	Für die Halterung der L 293 D http://www.reichelt.de/GS-16/3/index.html?&ACTION=3&LA=446&ARTICLE=8208&artnr=GS+16&SEARCH=GS+16
SALUS PH60	1	Heizungsregler http://www.amazon.de/Salus-SalusPH60-1-Energiespar-Heizk%C3%B6rperthermostat-SALUS-PH60/dp/B00AKDYEYA

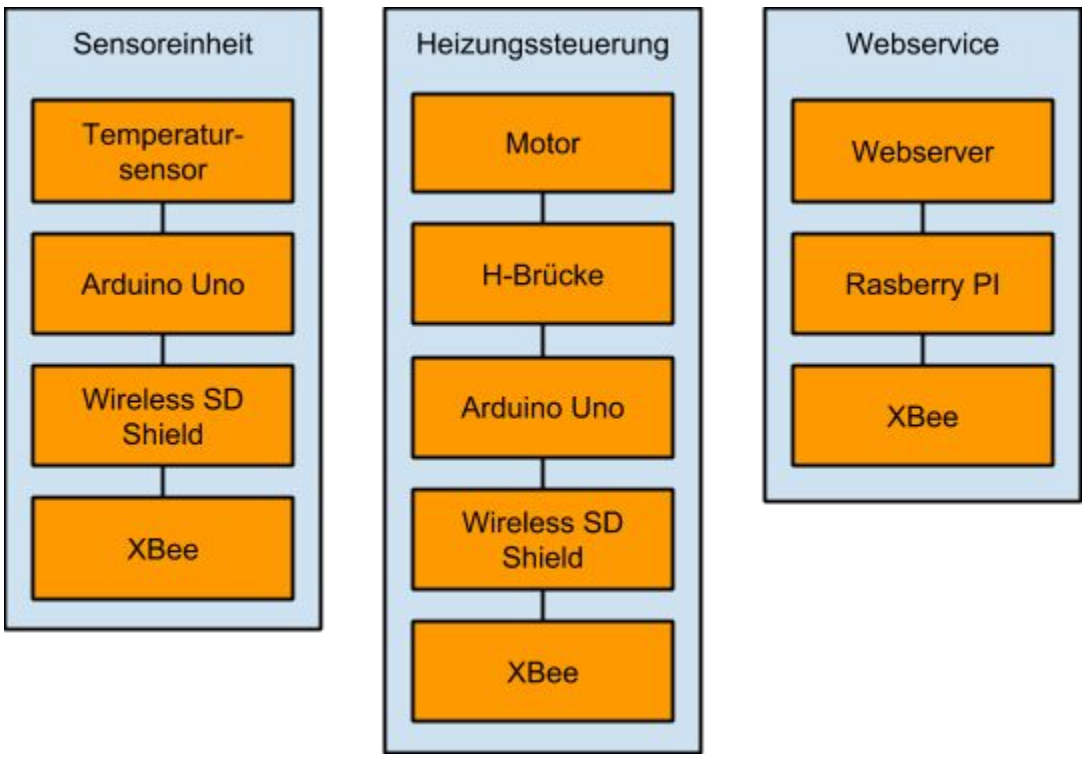
6.1 Nicht genutzte Komponenten

Der ursprünglich für die ZigBee Kommunikation des Webservice geplante RaspBee premium wurde nicht verwendet. Um über den RaspBee mit den anderen Teilnehmern zu kommunizieren, muss man zwangsweise ein deCONZ C++ Plugin geschrieben werden.

7 Vorgehensweise

Die Sensoreinheiten und Steuereinheiten werden auf Basis des Arduino Uno, des Wireless SD Shield und dem XBee-Modul realisiert. Die Kommunikation zwischen den Einheiten erfolgt unter Verwendung unseres eigenen Protokolls (siehe oben). Abbildung 3 zeigt den schematischen Zusammenhang zwischen den Komponenten.

Während der Umsetzung der Komponenten, wurden der Serial Monitor der Arduinos verwendet, um die Kommunikation zu Testen. Das Simulieren der Nachrichten und Auswerten der Rückmeldungen wurde mit Hilfe der *NIBO UCOM-XBEE* Module und der Applikation *XTCU*[8] von Digi realisiert. So konnten alle Teile einzeln entwickelt und später zusammengeführt werden.



[8] http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu

Abbildung 3

8 Realisierung Temperaturfühler

Der Temperatursensor AD22100 KT wird auf ein Steckbrett gesetzt und mit dem Arduino Shield Wireless SD über 3 Pins verbunden. Die Pins des Temperatursensors sind folgendermaßen definiert:

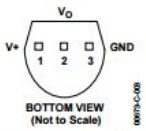


Figure 3. 3-Lead TO-92

Table 4. 3-Lead TO-92 Pin Function Descriptions

Pin No.	Mnemonic	Description
1	V+	Power Supply Input.
2	V_O	Device Output.
3	GND	Ground Pin Must Be Connected to 0 V.

Abbildung 4

Pin 1 wird mit 5V, Pin 2 wird mit A0 und Pin 3 wird mit GND verbunden. Um die Temperatur auslesen zu können wird der Arduino so programmiert, dass der Wert bei Pin A0 gelesen wird. Dieser Wert wird mithilfe der Transferfunktion aus dem Datenblatt des Temperatursensors nun umgewandelt, so dass man die Temperatur in Grad Celsius erhält. Die errechnete Temperatur wird mittels eines Broadcasts in einem festgelegten Intervall an alle Beteiligten Teilnehmer des ZigBee-Netzwerks gesendet. Alle Teilnehmer erhalten die aktuelle Temperatur und das Arduino Board beim Heizungsregler merkt sich den aktuellsten Wert um die Temperatur regeln zu können. Die Temperatur kann neben dem regelmäßigen Broadcast auch zu jedem Zeitpunkt selbst erfragen werden. Die Anfrage wird zu der ID-Adresse(„0001“) des Sensors gesendet. Dadurch erhalten ebenfalls alle Teilnehmer den Status über die aktuelle Temperatur.

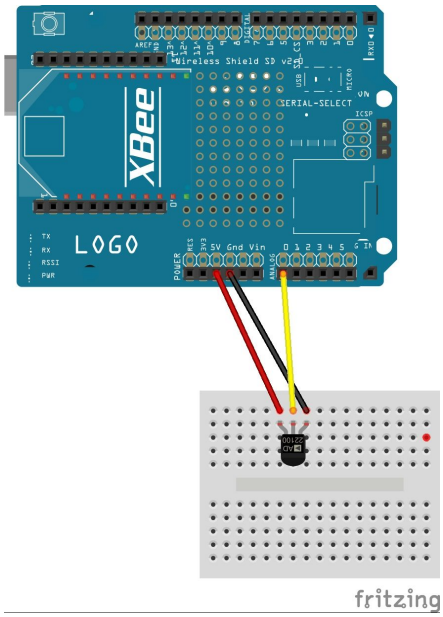


Abbildung 5

9 Realisierung Heizungsteuerung

Der elektronische Heizungsregler PH-60-1 von Salus wird über 4 Pins an ein weiteres Arduino Shield Wireless SD verbunden. Pin 13 ist für das Ein- und Ausschalten des Motors und Pin 9 und 12 sind für die Drehrichtung des Motors verantwortlich.

Das Arduino Board ist so programmiert, dass es die aktuellste Temperatur vom Sensor speichert. Je nachdem auf welchen Wert die Temperatur gesetzt werden soll, wird durch Abgleich der aktuell gespeicherten Temperatur der Motor nach rechts oder links gedreht. Der Status des Heizungsreglers kann jederzeit erfragt werden. Dabei kann man kann man die aktuelle Soll-Temperatur oder die Information bekommen, ob der Heizungsregler angeschaltet ist bzw. es ergibt sich die Möglichkeit den Regler ein- oder auszuschalten.

Dafür verfügt das Arduino Board über zwei ID-Adressen. Eine Adresse („0002“) wird dazu verwendet den aktuellen Status des Reglers auszulesen und um den Regler einzuschalten. Die andere Adresse („0003“) dient dazu eine Soll-Temperatur zu setzen, falls der Motor eingeschaltet ist.

In der untenstehenden Abbildung kann man die Zusammensetzung des Heizungsreglers und Temperaturfühlers mit den beiden Arduinos in einem Steckbrett sehen.

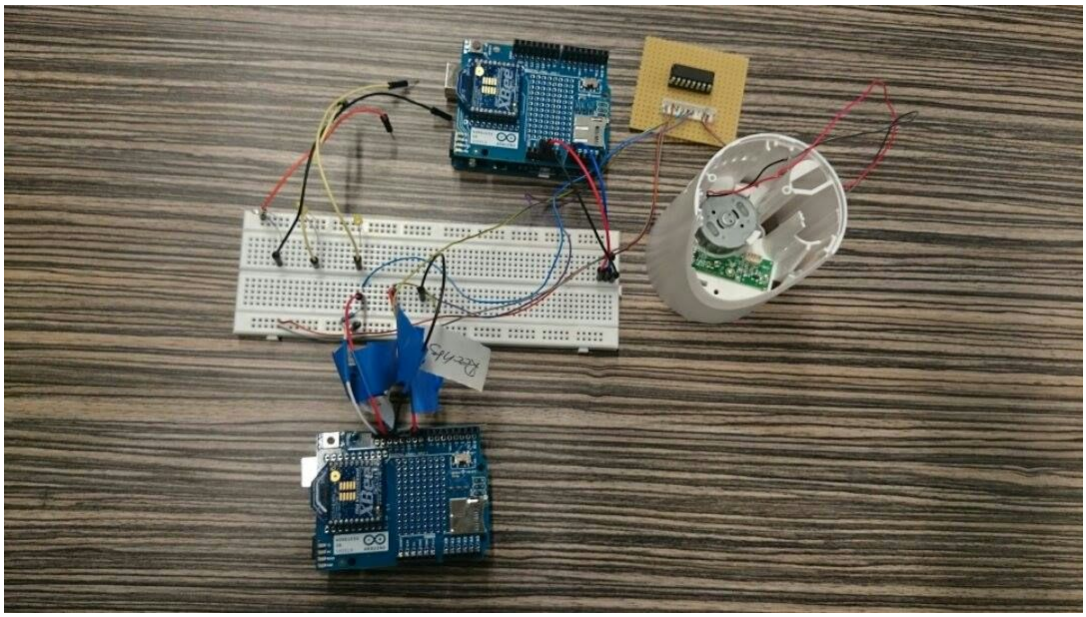


Abbildung 6

10 Realisierung Heizungsregler

Der L293D IC beinhaltet 4 Halb-H-Brücken Treiber. Damit lassen sich 2 DC-Motoren bidirektional betreiben (als 2-fach H-Brücke), oder 1 Schrittmotor birektional. Die Belastbarkeit pro Halbbrücke beträgt 600mA. Diesen schützen den IC vor den, vom Motor ausgehenden, Spannungsspitzen.

Wir haben eine 2 Halb-H-Brücken benutzt weil wir nur einen Motor angeschlossen haben und nur einen Motor benötigt.

Das größte Problem was passiert ist, war wir haben eine Bauplan nachgemacht von einer 4 Halb-H-Brücken so hat bei uns nicht alles geklappt gleich. Dannach haben wir das Problem beseitigt und dann funktionierte alles.

10.1 Drehrichtung

Die beiden Eingangssignale PC6 und PC7 bestimmen die Richtung, in die die Motoren laufen.

Das einzelne Richtungs-Signal vom Mikrocontroller wird dazu über ein invertierende Gatter dem L293D zugeführt.

10.2 Geschwindigkeit

Die beiden Eingangssignale PWM1A und PWM1B bestimmen die Geschwindigkeit, mit der die Motoren laufen.

Am Enable Eingang des L293D liegt ein so genanntes Pulsweitenmoduliertes (PWM) Signal. Dieses PWM Signal kann der Mikrocontroller mithilfe seiner Timer selbsttätig erzeugen.

Vom Programm her muss nur noch das Verhältnis zwischen Signal und Signalpause geändert werden. Damit wird die Geschwindigkeit des Motors geregelt.

10.3 Features

- Großer Spannungsbereich von 4,5..36V
- separate Eingangs-Logik Spannungsversorgung (z.B. TTL Pegel kompatibel)
- Interner ESD Schutz
- automatische Abschaltung bei Übertemperatur
- Ausgangsstrom 500mA pro Kanal
- kurzzeitige Spitzen Strom bis zu 1,2A je Kanal
- integrierte Freilaufdioden

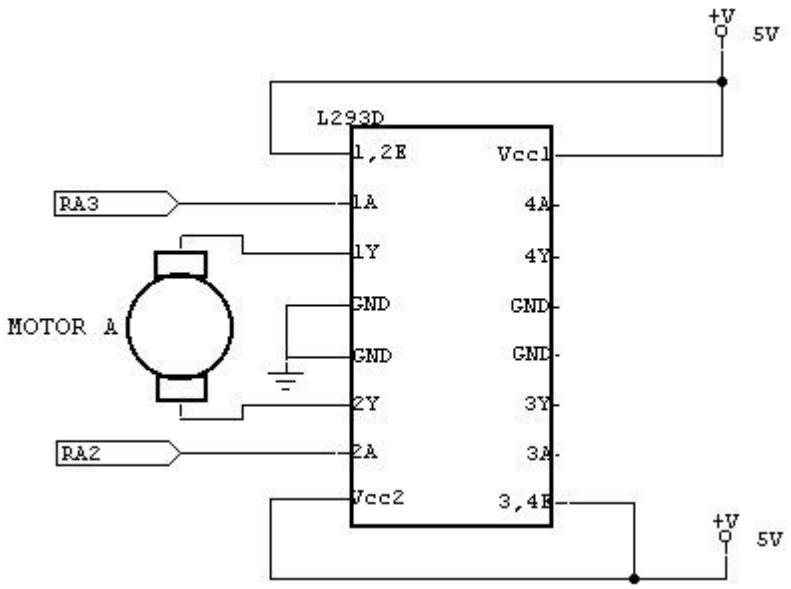


Abbildung 7

11 Realisierung Webservice

Der Webservice ist mit Ruby realisiert. Es handelt sich um eine *Sinatra*[9] Applikation. Als Hardwareplattform kommt ein *Raspberry PI 2 Model B* mit *Rasbian* als Betriebssystem zum Einsatz. Die Teilnahme am ZigBee Netzwerk erfolgt über das per USB angeschlossene *NIBO UCOM-XBEE* Modul. Das Modul stellt eine Serielle Schnittstelle bereit. Da der AT-Modus für die Kommunikation genutzt wird, kann von der Seriellen Schnittstelle einfach gelesen, bzw. geschrieben werden.

Alle verwendeten Klassen und befinden sich in der beiliegenden Datei *webservice.rb*. Folgende Bestandteile werden genutzt:

Message: Eine *Message* ist eine Nachricht die zwischen den Komponenten der Haussteuerung ausgetauscht werden. Sie abstrahiert den im Protokoll definierten Aufbau von Nachrichten.

PeerStates: Die *PeerStates* repräsentieren den Status der einzelnen Komponenten in der Hausteuerung. Jede Wertrückmeldung und Statusantwort wird gespeichert. Die gespeicherten Werte werden dann bei einer Statusanfrage an den Webservice ausgeliefert. So muss keine Statusanfrage an die Komponente gestellt und auf Rückmeldung gewartet werden. Da im Protokoll definiert ist, dass Komponenten bei einer Statusänderung auch eine Wertrückmeldung generieren, sollten die Status der Komponenten immer aktuell sein.

Communicator: Der *Communicator* ist für die Kommunikation im ZigBee Netzwerk verantwortlich. Es werden dazu Nachrichten von der Seriellen Schnittstelle gelesen bzw. auf diese geschrieben.

Handler: Der *Handler* initialisiert die *PeerStates* und den *Communicator*. In einem separaten Thread, wird die Verarbeitung von eingehenden Nachrichten durch den *Communicator* angestoßen. Hier ist ein separater Thread notwendig, da Schreib- und Lesevorgänge gleichzeitig stattfinden müssen.

Es werden außerdem zwei Routen für Sinatra definiert:

- `post '/controll'`
- `get '/status'`.

[9] http://www.sinatrarb.com/

Details zur Nutzung des Webservices können dem Kapitel *Webservice API* entnommen werden.

Zur Nutzung des Webservices auf dem Raspberry PI muss zunächst Rasbian installiert werden[10]. Danach kann Ruby 2.2.1 über die *Single User Installation* von RVM installiert werden[11]. Zum Schluss muss noch diverse gems installiert werden:

```
gem install sinatra sinatra-param sinatra-contrib --no-ri
--no-rdoc
```

Der Webservice kann dann über `ruby /home/pi/webservice.rb` gestartet werden. Er hört in der Standardkonfiguration auf TCP Port 4567. Um den Webservice automatisch beim Hochfahren des Raspberry PIs zu starten, kann folgender Befehl in `/etc/rc.local` angehängt werden:

```
su - pi -c /home/pi/webserver.sh
```

In der Datei webserver.sh muss der Webservice wie folgt gestartet werden:

```
#!/bin/bash
/home/pi/.rvm/rubies/ruby-2.2.1/bin/ruby /home/pi/webservice.rb
```

Für den vorbereiteten Raspberry PI wurde der Benutzername *pi* und das Passwort *test1234* verwendet.

12 Verbesserungen

Die folgenden Kapitel zeigen Verbesserungsmöglichkeiten des Projektes aus.

12.1 ZigBee Kommunikation

Die ZigBee Kommunikation findet momentan im AT Modus statt. Alle Pakete werden dabei an die Broadcast Adresse gesendet. In den Tests mit mehreren Modulen kam es dabei teilweise zu Schleifenbildung. Hier wäre der Einsatz des des *API Modes*[12] sinnvoller. Dies würde auch den Einsatz des RaspBee premium Moduls für den Webservice erlauben.

[10] https://www.raspberrypi.org/documentation/installation/installing-images/README.md
[11] https://rvm.io/rvm/install
[12] http://knowledge.digi.com/articles/Knowledge_Base_Article/What-is-API-Application-Programming-Interface-Mode-and-how-does-it-work

Die Kommunikation findet außerdem ohne Sicherheitsmaßnahmen statt. Die Einführung einer Authentifizierung und Autorisierung sowie einer Verschlüsselung ist durchaus sinnvoll. In der momentanen Implementierung kann dem Netzwerk einfach beigetreten und somit auch Schalthandlungen an beliebigen Komponenten vorgenommen werden.

12.2 Protokoll

Das Protokoll für die Kommunikation ist in ASCII kodiert. Eine Umstellung auf ein Binäres Format könnte die Netzwerkauslastung deutlich verringern.

Das Einführen von Datentypen würde das Setzen und Auslesen von Werten und erleichtern. So wäre klar, wann eine Fließkommazahl für Temperaturwerte oder eine positive Ganzzahl für Prozentangaben notwendig ist.

12.3 Webservice

Um aktuelle Statusinformationen zu Komponenten und deren Schnittstellen in Erfahrung zu bringen, muss der Webservice regelmäßig angefragt werden. Hier wäre der Einsatz von Websockets oder Push Notifications bei Statusänderungen

Es fehlt die Möglichkeit zur Authentifizierung am Webservice. Jeder, der über HTTP mit dem Webservice kommunizieren kann, hat die Möglichkeit beliebige Schalthandlungen auszulösen bzw. Statusabfragen abzusetzen.

12.4 Sensoren / Heizungsregler

Die Logik des Heizungseglers wird bei jedem Empfang einer Nachrichtig angestoßen. Hier wäre der Einsatz eines Timers[13], der diese Aufgabe im Hintergrund übernimmt besser geeignet. Auch findet im Heizungsegler keine Auswertung des momentanen Öffnungzustandes des Ventils statt. Ebenso fehlt eine Regelung, die auf Basis des Temperaturunterschiedes das Ventil zu einem bestimmten Prozentsatz öffnet.

12.5 Allgemein

Die Adressen der Komponentenschnittstellen sind momentan fest im Quellcode verankert. Eine Konfiguration der Adressen über DIP-Schalter oder das Setzen der Adresse über den Webservice sind für einen Produktiveinsatz mit mehreren Komponenten durchaus empfehlenswert.

[13] http://playground.arduino.cc/Code/Timer

Eine Möglichkeit zum Auflisten aller Komponenten und deren Schnittstellen würde es ermöglichen eine generische App zu entwickeln, mit der sich beliebige Teilnehmer steuern lassen.

13 Fehlende Hardware

Folgende Komponenten fehlen für den Betrieb des Prototypen:

USB Kabel A auf B	2 Stk.	Für die Konfiguration und Stromversorgung der Arduinos.
Micro SD-Karte, min CL10	1 Stk.	Als Speicher für den Raspberry PI
Micro USB Netzteil, 5V, 2A	1 Stk.	Stromversorgung Raspberry PI

Fazit

Das Projekt zeigt, dass sich ZigBee prinzipiell zur Datenübertragung im Bereich dem Heimautomation nutzen lässt. Das entworfene Protokoll lässt sich leicht realisieren und skaliert auch bei mehreren Geräten. Die Entscheidung eine hybride REST-RPC API zur Steuerung zu nutzen, erlaubt das direkte senden von Paketen gemäß der Protokoll Definition. Auch können mehrere Befehle mit nur einem Request ausgeführt werden. So lassen sich beispielsweise auch Logikmodule und andere Anwendungen durch die Nutzung des Webservices realisieren ohne für jede Statusabfrage oder Schalthandlung einen eigenen Request ausführen zu müssen.

Trotz der grundsätzlichen Nutzbarkeit sind weitere Verbesserungen und Anpassungen, insbesondere im Bereich der Kommunikation, notwendig.

Quellen

Katja and Guido Socher (1. Juli 2003). *Bau eines autonomen Lichtfinder-Robots*. Abgerufen am 9. Juli 2015 von http://www.linuxfocus.org/Deutsch/July2003/article297.shtml

Akiba (18. Mai 2009). *Zigbee Network Layer Tutorial - Part 3: Broadcasts and Neighbors*. Abgerufen am 9. Juli 2015 von http://www.freaklabs.org/index.php/Blog/Zigbee/Zigbee-Network-Layer-Tutorial-Part-3-Broadcasts-and-Neighbors.html

SGS-THOMSON Microelectronics (2009). *PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES*. Abgerufen am 9. Juli 2015 von https://www1.elfa.se/data1/wwwroot/assets/datasheets/ocL293D_647107_dat_en.pdf