Forschungsseminar Sensornetze SS12-WS12/13

Forschungsseminar Sensornetze
der Fakultät Informatik/Mathematik
Gruppe Heimautomatisierungsserver

Eingereicht von: Martin Dönicke, Ulrich Meckel
Eingereicht am: 03. März 2013

Inhaltsverzeichnis

Abkürzungsverzeichnis

6LoWPAN IPv6 over Low power WPAN

ARM Advanced RISC Machines

CoAP Constrained Application Protocol

CoRE Constrained RESTful Environments

CUL CC1101 USB Lite

FHEM Freundliche Hausautomatisierung und Energie-Messung

FOTA Firmware-Over-The-Air

GPL GNU General Public License

GNU GNU's Not Unix

HTTP Hypertext Transfer Protocol

HTW Hochschule für Technik und Wirtschaft

IETF Internet Engineering Task Force

IP Internet Protocol

IPv6 Internet Protocol Version 6

OTA Over-The-Air

REST Representational State Transfer

RISC Reduced Instruction Set Computer

UDP User Datagram Protocol

URI Uniform Resource Identifier

WPAN Wireless Personal Area Network

1 Einleitung

Im Rahmen unseres Master-Studiums an der Hochschule für Technik und Wirtschaft (HTW) Dresden haben wir am Forschungsseminar Sensornetze teilgenommen. Thema in diesem Modul ist die Erschaffung eines Sensornetzes mit Hilfe von Open-Source Hardware und Open-Source Protokollen und/oder Software. Das folgende Dokument wurde von der Gruppe „Heimautomatisierungsserver“ erarbeitet und soll die Arbeit der letzten zwei Semester in diesem Projekt zusammenfassen.

Zu Beginn bestand die Aufgabe darin, sich mit dem neuen Themengebiet vertraut zu machen, und die vorgeschlagenen Richtlinien des Professors zu analysieren. Drei Gruppen wurden mit unterschiedlichen Themenbereichen beauftragt. Somit entstand eine Gruppe, welche sich mit dem Auslesen von Sensorwerten auf dem Mikrocontroller beschäftigen sollte. Eine weitere Gruppe beschäftigte sich mit Contiki und dem Aufbau des Funknetzwerks. Und schließlich die Gruppe, die sich über alle Protokolle und Software oberhalb der Transportschicht Gedanken machen sollte. In diesem Dokument werden die Aufgaben der dritten Gruppe beleuchtet. Angefangen mit unserer Aufgabe haben wir damit, sich nach ähnlichen Projekten umzuschauen. Eine Analyse ihrer Stärken und Schwächen wurde angefertigt und im Weiteren diskutiert, ob man diese Projekte aufgreifen möchte oder nur versucht, ihre Ideen und Stärken in ein neues Projekt mitzunehmen.

Nachdem beschlossen war ein neues Projekt anzufangen, kam uns die Analyse der anderen Projekte zu gute, da man beispielsweise bereits mit `FS20`-Geräten in Kontakt gekommen ist. Zwar stand von Anfang an fest, dass wir Open-Source-Hardware verwenden wollen, doch ebenso hatten wir das Ziel, wie viele andere Projekte, nicht nur zu unserer Hardware kompatibel zu sein. Es sollte auch möglich gemacht werden, Geräte wie die der `FS20`-Reihe anzusprechen. Die Realisierung dieser Möglichkeit als auch weitere konzeptionelle Überlegungen der Architektur waren vor allem zu Beginn die Hauptaufgabe. Im späteren Verlauf rückte hauptsächlich die Programmierung in den Mittelpunkt.

2 Andere Projekte

2.1 HexaBus

Von dem Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, wird seit ca. 2011 eine sowohl hardware-, als auch softwareseitige Open-Source Lösung für die Heimautomatisierung entwickelt. Leider war zu Beginn unseres Forschungsseminars im März 2012 noch nicht viel von dem Open-Source Projekt zu finden. Außer ein paar Einträge auf ihrer Wiki-Seite[1], welche Hinweise auf eine von Ihnen betriebene Steckdose und einen Stick, mit welchem die Kommunikation über IPv6 over Low power WPAN (6LoWPAN) ermöglicht wird, gab es sonst keine weiteren Informationen.

Doch neben unserem Projekt wuchs auch das Projekt „HexaBus - the Kraken of Home Automation!“ heran. Zwar ist neben der Steckdose derzeit kein weiteres Gerät ansprechbar, doch ist die Dokumentation und die Funktionsfähigkeit gestiegen.

2.1.1 Merkmale

Contiki Auf dem von HexaBus verwendeten Stick und der verwendeten Steckdose wird ein Mikrocontroller betrieben, auf welchem das Betriebssystem Contiki läuft.

6LoWPAN Wie bereits erwähnt, verwendet HexaBus 6LoWPAN, ein komprimiertes Internet Protocol Version 6 (IPv6) und User Datagram Protocol (UDP). Vorallem der, nun nicht mehr futuristische, Gedanke, jedem Gerät eine eigene IPv6-Adresse geben zu können, ist damit möglich, ohne den gesamten IPv6 Stack übertragen zu müssen. Dieser Vorgang wäre für Mikrocontroller zu speicher- und energieintensiv. Innerhalb des 6LoWPAN-Netzes wird über Funk auf der Frequenz 868 MHz kommuniziert. Weiterhin haben die Mitarbeiter des Projekts „mySmartGrid“, in dem HexaBus entwickelt wird, ein Protokoll entwickelt, um die Geräte internetfähig zu machen. Dem Benutzer ist es also möglich, mit einem Webbrowser oder einer App auf dem Mobiltelefon, sein System zu überwachen oder abzufragen.

Keine Zentrale Kontrollinstanz Informationen des Systems werden nicht zentral überwacht. Stattdessen werden Informationen, laut eines Vortrags[2], über Broadcast an alle verschickt. Da IPv6 keine Broadcast-Funktion anbietet, ist an dieser Stelle sicher ein Multicast gemeint. Nichtsdestotrotz entscheidet bei HexaBus jeder Knoten selbst, ob er mit diesen Informationen etwas anfangen kann oder nicht. Programme für die Knoten können am Computer geschrieben und mittels Funk übertragen werden (sog. Firmware-Over-The-Air (FOTA)-Programmierung) Ebenso ist auch die Rede von einer Webseite, auf der der Benutzer spezielle Namen für seine Geräte vergeben kann. Wo dieser Webserver allerdings läuft, und wozu die Namensgebung dient, ist unklar.

[1] `https://github.com/mysmartgrid/hexabus/wiki`
[2] `http://developer.mysmartgrid.de/lib/exe/fetch.php?media=hexabus_ec2012-05.pdf`

Verschlüsselung Um die Daten zu verschlüsseln, wurde Contiki um eine AES-128 Verschlüsselung erweitert.

2.1.2 Verwertbarkeit

Da zu HexaBus zum Entstehungszeitpunkt unseres Projektes leider noch nicht viele Informationen auffindbar waren, wie bei Ihnen alles funktioniert, ist die Frage der Nutzung von HexaBus leider hinfällig gewesen. Betrachtet man nun Ihr System und das in der Zeit des Forschungsprojektes entstandenen Systems, so könnte man denken, dass wir uns viel von Ihnen abgeschaut haben. Beziehungsweise sind unsere Ziele sehr ähnlich, denn wir verwenden ebenfalls Contiki auf den Knoten. Zusätzlich wollen wir, dass wir unsere Knoten von überall aus ansprechen können.

Weiterhin senden wir auf der gleichen Frequenz und nutzen das gleiche Vermittlungs- und Transportprotokoll. Bei uns leider noch nicht vorhanden, aber geplant, ist ebenso die Verschlüsselung der Kommunikation. Einzig großer Unterschied ist, dass in unserem Projekt eine zentrale Kontrolleinheit geplant ist. Trotzdem wird es bei uns, wie bei HexaBus, auch möglich sein, Sensorknoten direkt mittels einer IPv6-Adresse abzufragen. Schon ein wenig vorweggenommen, aber statt eines selbstentwickelten Protokolls wird, im Gegensatz zu HexaBus, bei uns das Constrained Application Protocol (CoAP) verwendet.

2.2 FHEM

Freundliche Hausautomatisierung und Energie-Messung (FHEM) ist ein unter der GNU General Public License (GPL) veröffentlichte Serversoftware zur Heimautomatisierung. Es wurde hauptsächlich von Rudolf König in Perl geschrieben. Der Quellcode lässt sich von der FHEM-Webseite[3] beziehen. Es werden viele proprietäre Protokolle unterstützt, unter anderem relativ bekannte, wie zB. FS20 und HomeMatic. Leider ist das Projekt für Entwickler sehr spärlich dokumentiert. An Anwenderdokumentationen mangelt es jedoch nicht, weswegen es offensichtlich verbreitet zum Einsatz kommt.

Ursprünglich sollte geprüft werden, ob FHEM eine Grundlage für unser geplantes Vorhaben bilden kann. Diese Idee wurde jedoch relativ schnell verworfen, da der Perl-Quellcode sowohl schwer verständlich, als auch nicht ausreichend dokumentiert ist. Weiterhin werden hauptsächlich proprietäre, unsichere Protokolle verwendet, welche zudem noch andere Netztopologien besitzen, als unser geplantes Sensornetz. Aufgrund dessen wäre eine Anpassung von FHEM an unser Vorhaben wahrscheinlich sehr komplex geworden, womit wir auch von Grund auf neu beginnen konnten, ohne Altlasten mitzuführen. Eine andere Projektgruppe, welche sich mit dem HomeMatic-Protokoll befasst haben, konnten jedoch von Teilen des FHEM-Quellcodes profitieren.

[3] `http://fhem.de/fhem.html`

3 Entwurf

3.1 Zentrale Kontrolleinheit

Bei der Eigenentwicklung entschieden wir uns für eine zentrale Architektur. Hauptgrund dafür war die Überlegung, dass der spätere Anwender des Netzwerkes nicht selbst mit den einzelnen Sensorknoten direkt kommunizieren soll, sondern eine zentrale Anlaufstelle hat, bei der er alle Informationen des Netzwerkes überblicken kann. Ebenso soll an dieser zentralen Kontrolleinheit die Möglichkeit bestehen, ein Regelwerk zu erstellen. Diese Regeln bilden dann die Logik des Netzwerkes und ermöglichen erst die automatisierte Steuerung im Netz.

Der Vorteil stellt hier gleichzeitig einen Nachteil dar, da zwar die Logik nur an einer Stelle bearbeitet werden muss, sie jedoch komplett weg fällt, wenn die zentrale Einheit einen Fehler hat und dementsprechend versagt. Das Sensornetz ist dann zwar weiterhin über CoAP erreichbar, allerdings ohne automatisierte Vorgänge. Läuft diese Instanz aber stabil, schont es die Batterien der Sensorknoten, da Nachrichten nicht über Broad-/Multicast gesendet werden müssen, sondern direkt an die betreffende Adresse.

Durch die Verwendung von 6LoWPAN und die damit verbundene Möglichkeit, jedem Sensor eine eindeutige Internet Protocol (IP)-Adresse zu vergeben, wollten wir die Möglichkeit der direkten Kommunikation des Benutzers mit einem Sensorknoten nicht ausschließen. Deshalb soll es später, neben der Möglichkeit der Verwaltung über die zentrale Instanz, weiterhin möglich sein, auch direkt über die IP-Adresse eines Geräts, mit diesem über CoAP zu kommunizieren.

3.2 Constrained Application Protocol

CoAP (nach „work in progress“ draft-ietf-core-coap-13[4]) implementiert eine Representational State Transfer (REST)-Architektur, wie sie auch im Internet bei dem allseits bekannten Hypertext Transfer Protocol (HTTP) verwendet wird. Das REST-Prinzip eignet sich aus unserer Sicht hervorragend für die Kommunikation in einem Sensornetz. Denn grundsätzlich stellen wir in unserem Netzwerk folgende vier Anfragen:

Einen Wert abfragen Mit einem GET-Request wird genau dies ermöglicht. Der Client sendet die Anfrage an eine Uniform Resource Identifier (URI) und der Server antwortet mit der Repräsentation der Ressource oder einem Fehlercode.

Einen Wert setzen/ändern Mit einem PUT-Request und einem passenden Wert ist es möglich, die Repräsentation einer Ressource, welche wieder durch eine URI angesprochen wird, zu ändern. Vor allem kann dies zum Schalten von Aktoren genutzt werden.

Neue Sensoren anlegen Kommen neue Sensoren ins Netzwerk, so ist es eventuell nötig, neue Ressourcen anzulegen. Dies kann mittels PUT- und POST-Methoden erledigt werden.

[4] `http://datatracker.ietf.org/doc/draft-ietf-core-coap/`

Sensoren löschen Ressourcen können von einem Server mittels der DELETE-Methode entfernt werden.

Mit diesen vier Anfragemöglichkeiten an Server ist eigentlich alles abgedeckt was man auf den ersten Blick für eine Kommunikation mit den Sensoren benötigt. Für den Server, welcher in unserem Fall ein Batteriebetriebener Mikrocontroller ist, entstehen die Vorteile, das er keine Status für Anfragen pflegen muss. Eine Anfrage enthält alle Informationen die für die Beantwortung notwendig sind. Damit sparen wir zusätzlichen Rechen- und Speicheraufwand.

HTTP, welches ja ein sehr etabliertes Protokoll ist, kommt allerdings für die Verwendung bei uns nicht in Frage, da es einen viel zu großen Overhead, für die meist geringen Informationen, die verschickt werden müssen, hat. Hingegen ist CoAP, welches man als eine abgespeckte Variante von HTTP ansehen könnte, im Gegensatz dazu von der Constrained RESTful Environments (CoRE)-Gruppe, welcher eine Untergruppe der Internet Engineering Task Force (IETF) ist, genau für die Anwendung in „low-power and lossy networks“ konzipiert. Seit 2010 wird an dem Protokoll gearbeitet und nun ist es in einem Zustand, indem es nutzbar ist. Vorallem für den Fehlerfall bei Anfragen stehen wie in HTTP, eine große Anzahl (jedoch im direkten Vergleich vermindert) an Fehlercodes zur Verfügung. Für weitere Informationen bezüglich CoAP verweisen wir auf unsere Ausarbeitung[5] bezüglich CoAP im Rahmen der Lehrveranstaltung „Sensornetze“.

3.3 Gateway

Das Gateway dient dazu, Sensornetze, welche andere Protokolle als das unsere verwenden, in unser Sensornetz zu integrieren. Diese Protokolle sind zumeist proprietär und Informationen darüber sind nur über Reverse-Engineering zu erlangen. Da FS20 ein recht verbreitetes System ist(siehe Abschnitt 3.3.2, wäre es wünschenswert, dass dieses und ähnliche Protokolle in unserer Heimautomatisierungslösung unterstützt werden würden. Eine Umsetzung davon würde die Kosten für eine Neuinstallation reduzieren, da bereits installierte Systeme optimalerweise ohne Anpassung weiter genutzt werden können.

Somit war es unser Ziel, auch bereits bestehende Sensornetze nahtlos in unser Netz zu integrieren. Das Gateway fungiert somit als CoAP-Cross-Proxy, welcher alle Sensoren des jeweiligen Sensornetzes als seine eigenen CoAP-Ressourcen anbietet. Es muss somit eine Übersetzung von CoAP-Anfragen in das entsprechende andere Protokoll erfolgen. Antworten der Sensorknoten müssen nach CoAP zurückübersetzt werden. Nach außen hin wirkt das Gateway also wie ein einziger Netzknoten, der eine Vielzahl von Sensoren bereitstellt. Mit ihm wird, genau wie mit allen anderen Netzknoten, nur via CoAP kommuniziert, was die Anwendungslogik des Servers kompakt hält.

Während des Semesters wurden Gateways exemplarisch für FS20 und HomeMatic umgesetzt. Die Kommunikation in den Fremdnetzen wird mit Hilfe eines CC1101 USB Lite (CUL) durchgeführt. Auf dieser Hardware in Verbindung mit einer FS20-Funksteckdose

[5] `https://github.com/umeckel/HTWDD-CoAP`

wird im folgenden Abschnitt näher eingegangen. Details zur Implementierung des Gateways sind im entsprechenden Dokument der verantwortlichen Gruppe zu finden, an welche die Aufgabe nach der Entwurfsphase abgegeben wurde.

3.3.1 CUL

Zur Kommunikation mit Geräten, welche lediglich proprietäre Protokolle unterstützen, wurde ein CUL von `busware` eingesetzt. Die Hardware kann aus dem Online-Shop von

Abbildung 1: CUL v3, Quelle: `http://busware.de/tiki-index.php?page=CUL`

`busware`[6] bezogen werden. Mit Hilfe des CUL in Verbindung mit der Firmware `culfw`[7] ist es möglich, auf der Frequenz 868 MHz verschiedene Protokolle zu empfangen/senden. Die `culfw` wurde vom FHEM-Entwickler Rudolf König geschrieben. Sie ist quelloffen und frei zugänglich. Durch die Firmware stellt der CUL unter Linux eine serielle Schnittstelle unter `/dev/ttyACM*` zur Verfügung. Darüber ist eine Kommunikation, mit beispielsweise FS20-Geräten, sehr einfach umzusetzen, wie in Abschnitt 3.3.3 kurz erläutert wird.

3.3.2 FS20

Laut ehomeportal (siehe Webseite[8]) ist FS20 das erfolgreichste Funk-Haussteuerungssystem im Niedrigpreissektor. Es zeichnet sich durch folgende Eigenschaften aus:

+ geringe Anschaffungskosten
+ vielfältiges Angebot an Komponenten
- unsichere Übertragung
- keine Authentifizierung
- keine zuverlässige Übertragung (keine Quittierung von Nachrichten)

[6] `http://shop.busware.de/product_info.php/products_id/29`
[7] `http://culfw.de/culfw.html`
[8] `http://www.ehomeportal.de/Funk-System-FS20.htm?shop=shop&SessionId=&a=catalog&t=1597&c=1597&p=1597`

Viel näher soll in diesem Dokument nicht auf FS20 eingegangen werden, da die Integration von anderen Systemen nur eine untergeordnete Rolle spielt, jedoch trotzdem ermöglicht werden soll.

3.3.3 FS20-Kommunikation via CUL

Um `FS20`-Kommandos vom PC aus abzusetzen, müssen Kommando-Strings auf die serielle Schnittstelle geschrieben werden. Aufbau der Kommando-Strings (nach culfw-commandref[9]):

F Signalisiert CUL, dass ein FS20-Kommando geschickt werden soll.

HHHH FS20 Housecode

AA FS20 Adresscode

CC FS20 Commandcode

EE FS20 Timespec (nur bei komplexen Kommandos benötigt)

Daraus ergibt sich folgende Struktur: `F<HHHH><AA><CC>[<EE>]` → z.B. `F12340111` → Kommando 11 (`On`) an Gerät mit Housecode 1234 und Adresscode 01.

Funksteckdose FS20 ST-3 Die Steckdose ist über den Conrad Online Shop[10] beziehbar. Über eine Hardware-Taste kann sie in den Pairing-Modus versetzt werden. In

Abbildung 2: FS20 ST-3, Quelle: `http://www.conrad.de/ce/de/product/623004/`

diesem Modus kann ihr ein neuer Housecode und ein neuer Adresscode zugewiesen werden. Nachdem dies Prozedur vollzogen ist, lässt sich die Steckdose praktisch einsetzen.

[9] `http://culfw.de/commandref.html`
[10] `http://www.conrad.de`

Dieser, hier ersichtliche, geringe Konfigurationsaufwand ist unter anderem ein Grund dafür, dass FS20 relativ erfolgreich geworden ist. Listing 1 zeigt beispielhaft, wie ein FS20-Kommando in Python über den CUL abgesendet werden kann.

```
houseCode = 1234
deviceCode = 01
commandCode = 11
msg = ''.join(('F',houseCode,deviceCode,commandCode,'\n'))
ser = Serial('/dev/ttyACM0', writeTimeout=1)
ser.write(msg)
```

Listing 1: FS20

Die verwendeten Kommandos zur Steuerung der FS20 ST-3 zeigt Tabelle 1.

Kommando	Code	Aktion
On	11	Schaltet die Steckdose ein.
Off	00	Schaltet die Steckdose aus.
Toggle	15	Lässt Steckdose ihren Zustand wechseln.

Tabelle 1: Kommandos

3.4 Datenbankentwurf

Um alle Informationen des Systems und seiner Struktur zu speichern, soll eine Datenbank verwendet werden. Im Verlauf des Forschungsseminars hat sich der Datenbankentwurf dazu kontinuierlich verändert, bis er schließlich zu dem Stand gekommen ist, wie er jetzt in Abbildung 3 zu sehen ist. Mit diesem Entwurf ist es möglich, sogenannte CoAP-Endpoints in Form eines Nodes einzutragen. Erhält man im Weiteren dazu noch die an diese Mikrocontroller angeschlossenen Sensoren, so können diese in der Tabelle „Sensor“ gespeichert werden. Mit Hilfe des Fremdschlüsselattributs „NodeID“ ist es möglich, den Sensor genau einem Node zuzuordnen. Analog funktioniert die Speicherung von Sensorwerten, die in der der Tabelle „Readings“ gespeichert werden und über den Fremdschlüssel zugeordnet werden.

Aus verschiedenen Gründen wurde die Tabelle „Group“ mit in die Datenbank aufgenommen. Vor allem wegen ihr und anderen Alternativen hat sich der Datenbankentwurf oft geändert. In ihr können sowohl Sensoren, als auch Nodes in verschiedene Teilmengen gruppiert werden. Denkbar ist hier eine räumliche Unterteilung, wie zum Beispiel in die Gruppen „Wohnzimmer“, „Schlafzimmer“ usw., aber auch in Gruppierung hinsichtlich ihrer Funktion, wie zum Beispiel „Sensor“ und „Aktor“ sind denkbar. Insbesondere für Aktoren könnte es eine Gruppierung bezüglich ihrer Verwendung geben. Das bedeutet, dass Aktoren, die mit den Befehlen wie „An“ oder „Aus“ gesteuert werden können, sich ebenfalls zusammenfassen lassen. Ebenso könnte es möglich sein, dass manche Sensoren

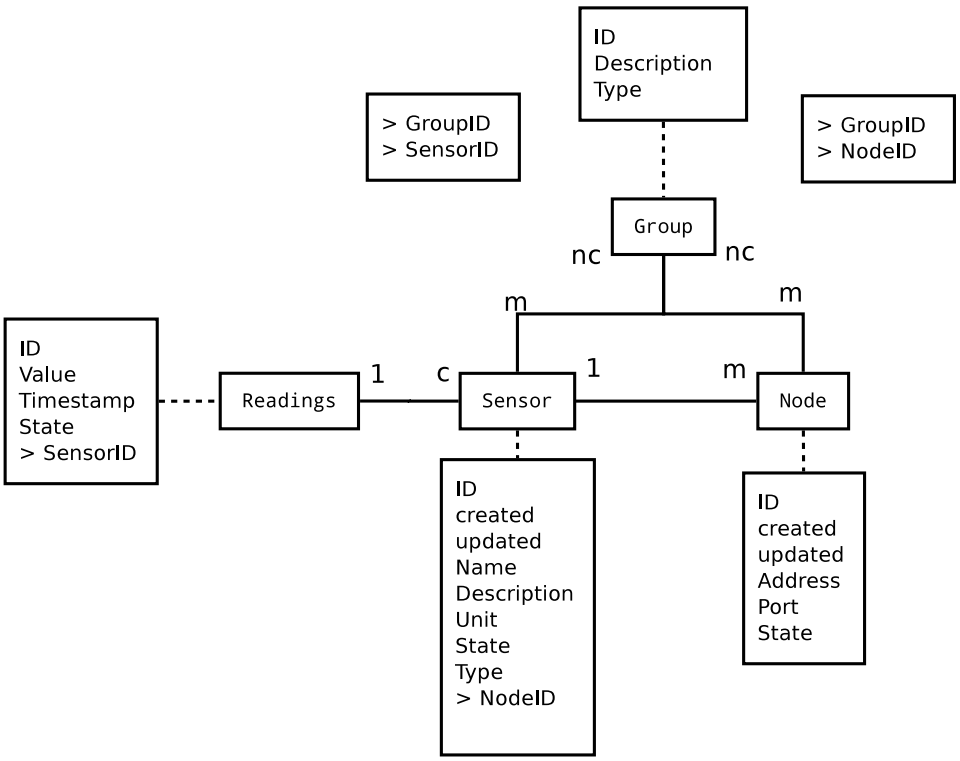

Abbildung 3: Datenbankentwurf

oder Aktoren eine gewisse Zeit lang nicht automatisch gesteuert werden sollen. Anstatt die Logik zu löschen, welche für ihre Steuerung verantwortlich ist, könnte hier eine Gruppe definiert werden, welche sie temporär deaktiviert.

4 Implementierung

Bei der Implementierung wurde die Programmiersprache Python in der Version 2.7 gewählt. Python verfolgt eine sogenannte „batteries included“-Philosophie, diese besagt das Mitliefern einer umfangreichen Bibliothek. Somit wird eine große Menge an häufig benötigten Funktionalitäten durch die mitgelieferten Bibliotheken bereitgestellt, welche dem Entwickler zur freien Verfügung bereit stehen.

Im späteren Praxiseinsatz ist es sinnvoll, die Server-Software auf einem Gerät in Betrieb zu nehmen, welches möglichst wenig Strom verbraucht und gegebenenfalls sowieso dauerhaft an ist. Denkbar hierfür wäre ein Router mit einem zugänglichen Unix. Projekte, wie beispielsweise OpenWRT[11] oder Freetz[12], ermöglichen einen freien Zugriff auf das System von entsprechenden Routern. Dort muss dann der Python-Interpreter mitsamt der benötigten Bibliotheken lauffähig gemacht werden, um den Einsatz der Server-Software zu ermöglichen.

Für den Einsatz in der Praxis könnte man sich auch die Inbetriebnahme der Server-Software auf einem Raspberry Pi[13] vorstellen. Hierbei handelt es sich um einen ARM-basierten Minicomputer, auf welchem GNU's Not Unix (GNU)/Linux läuft. Hier ist die Installation von Python über die Paketverwaltung schnell erledigt, und dem Einsatz dieses Geräts mit der entsprechenden Software steht nichts mehr im Weg. Tiefgehendere Überlegungen wurden hier jedoch nicht angestellt, da diese im betrachteten Entwicklungszeitraum eine sehr niedrige Priorität hatten. Es wurde lediglich darauf geachtet, dass ein vielfältiger Einsatz auf verschiedensten Systemen prinzipiell möglich ist.

Für die Nutzung einer Datenbank bietet Python unter anderem die `sqlite3`-Bibliothek[14] an. Mit Hilfe dieser Bibliothek wurde die Datenbankschnittstelle für den Heimautomatisierungsserver realisiert. Ebenso wäre es auch möglich gewesen eine `MySQL`-Datenbank zu verwenden, doch wurde sich wegen verschiedenen Gründen dagegen entschieden. Zum einen spielt die Wahl der Datenbank für die Entwicklung des Prototypen keine große Rolle, da man später mit nicht sehr viel Aufwand die Datenbank austauschen kann. Dafür müssen lediglich die Datenbankfunktionen des Datenbankmoduls neu implementiert werden. Da `sqlite` aber keine beziehungsweise wenig Konfiguration benötigt, hielt es so weniger vom wesentlichem Implementierungsaufwand ab. Der große Vorteil der Benutzerverwaltung von `MySQL` wurde hier auch nicht benötigt. Wächst die Datenmenge, also die Anzahl an Sensoren im Netzwerk so stark an, dass es doch zu Performanceproblemen kommen sollte, könnte man immer noch auf `MySQL` umsteigen. Doch im derzeitigen Zustand, und auch für die nächste Zeit, ist dies aus unserer Sicht nicht nötig.

Weiterhin wurde für das Loggen von Meldungen des Servers die `Logging`-Bibliothek[15] verwendet. Konfiguriert wird das Loggingsystem durch eine Konfigurationsdatei. In die-

[11] `https://openwrt.org/`
[12] `http://freetz.org/`
[13] `http://www.raspberrypi.org/`
[14] `http://docs.python.org/2/library/sqlite3.html`
[15] `http://docs.python.org/2/library/logging.html`

ser Datei lassen sich verschiedene Submodule definieren, wie zum Beispiel `Datenbank` oder `CoAPHandler`. Des Weiteren werden in der Datei auch verschiedene Logginghandler konfiguriert und zugewiesen. Die Bibliothek stellt eine Reihe solcher Handler zur Verfügung[16], wie zum Beispiel einen für das Schreiben in eine Datei oder in einen Stream. Ebenso ist es aber möglich, sich Meldungen über Sockets oder via E-Mail senden zu lassen. Des Weiteren lassen sich auch noch verschiedene Nachrichtenformate definieren, welche die Handler nutzen sollen. So kann der File-Handler beispielsweise zusätzlich Datum und Uhrzeit, der E-Mail-Handler hingegen nur das Datum zusätzlich loggen. Hat man diese Dinge erstellt und konfiguriert, lassen sich, wie in Listing 2 zu sehen ist, Loggingmodule anfordern. Über die Funktionen info, warning, error, critical und debug lassen sich dann die Nachrichten in unterschiedlichen Logginglevels generieren und werden je nach eingestelltem Level im Handler gesendet oder geschrieben.

```
Log = logging.getLogger('Datenbank')
Log.log(Log.level,"Logging enabled on level {0}".format(
    logging.getLevelName(Log.level)))
```

Listing 2: Anforderung eines Loggingmoduls

4.1 CoAPy

Für die Kommunikation mit dem Sensornetz ist es notwendig CoAP zu unterstützen. Nach anfänglicher Freude über eine bereits vorhandene CoAPy-Bibliothek[17] von der Firma „People Power Company“[18], kam die schnelle Ernüchterung, als festgestellt wurde, dass diese Implementierung auf dem Stand von CoAP-03 war. Der aktuelle Stand, zu dem damaligen Zeitpunkt, war allerdings bereits CoAP-12. Zum Zeitpunkt der Erstellung dieses Dokuments bereits CoAP-13. Nach dieser Erkenntnis wurde sich dazu beraten, wie man mit dem Problem umgeht.

Es musste einerseits geklärt werden, ob die Version 03 für unsere Bedürfnisse ausreiche und vorallem auch welche CoAP-Version von Seiten Contiki her schon vorhanden ist. Nach Diskussion in der Gruppe stellte sich heraus, dass von Contiki bereits eine CoAP-08 Implementierung vorliegt. Im folgenden war also nur noch die Entscheidung offen, ob man doch auf eine andere Programmiersprache wechselt, in der es eine aktuellere Version von CoAP bereits gibt, oder ob man selbst das stillgelegte CoAPy-Projekt weiterführt.

Nach Untersuchungen der Changelogs von CoAP und der Dokumentation von CoAPy wurde entschieden CoAPy auf die Version 08 im Rahmen des Projektes zu aktualisieren. Denn vorrangig wurden in CoAP nur weitere Optionen hinzugefügt und Mechanismen klarer formuliert und strukturiert. Es gab nur wenige Änderungen, die grundsätzliche Dinge des Protokolls betrafen oder änderten. Zusätzlich ist der Quellcode von CoAPy sehr übersichtlich strukturiert, sodass es nicht sehr lange dauerte, die Bibliothek zu verstehen und zu erweitern beziehungsweise aktualisieren zu können.

[16] `http://docs.python.org/2/library/logging.handlers.html`
[17] `http://coapy.sourceforge.net/`
[18] `http://www.peoplepowerco.com//`

Die Version 08 von CoAP brachte auch den Vorteil, dass die Contiki-Gruppe, welche ebenfalls auf ihren Sensorknoten CoAP implementieren musste, zum Testen das Firefox-Plugin Copper[19] verwenden konnte, welches ebenfalls CoAP-08 unterstützt. Sie waren somit nicht abhängig von unserer Aktualisierung und wurden dadurch nicht blockiert. Für uns bot sich so der Vorteil, dass wir vergleichen konnten, ob unsere Aktualisierungen der CoAPy-Bibliothek korrekt waren und später kein Unterschied festzustellen ist, ob via Abfragen CoAPy oder Copper gesendet wurden. Die Aktualisierungen zu CoAPy[20] sind, wie auch die ursprünglichen Quelldateien von CoAPy, öffentlich einsehbar.

4.2 Schnittstelle zum Sensornetzwerk

Da es geplant war, über eine Webseite Werte der Sensoren abzufragen oder ändern zu können, wird eine Schnittstelle benötigt, um CoAP-Anfragen in das Sensornetzwerk zu senden. Ebenso wird diese Schnittstelle von dem Regelwerk benötigt, welches für Aktionen ebenfalls CoAP-Nachrichten versenden muss. Diese Schnittstelle implementiert der Server, um weiterhin als zentrale Einheit die Kontrolle über das Netzwerk zu behalten. Zu diesem Zweck lauscht der Server auf einem konfigurierbaren Port (Standard: `4123`) auf eingehende Nachrichten. Diese Nachrichten müssen einer speziellen Syntax(`ID\nTyp\nZusatz`) entsprechen, um akzeptiert zu werden. Details hierzu sind in Tabelle 2 zu finden.

Typ	ID	Zusatz	Beschreibung	Beispiel
GET	SensorID	-	Schickt eine GET-Anfrage an eine Ressource mit der SensorID aus der Datenbank	`25\nGET\n`
PUT	SensorID	Payload	Schickt eine PUT-Anfrage an eine Ressource mit der SensorID aus der Datenbank um den Wert im Payload zu setzen	`25\nPUT\noff`
DISC	NodeID	-	Sendet eine Discovery-Anfrage an die NodeID aus der Datenbank	`5\nDISC\n`
DISC	-1	IP:Port	Sendet eine Discovery-Anfrage an die spezielle Adresse	`-1\nDISC\n ::1:5683`

Tabelle 2: Details zur Nachrichten-Syntax

[19] `https://addons.mozilla.org/de/firefox/addon/copper-270430/`
[20] `https://github.com/umeckel/FS_coapy`

5 Zusammenfassung und Ausblick

Resultat der Projektarbeit ist ein Prototyp, bestehend aus einem Testnetz mit zwei Netzknoten und deren Sensoren, eine Anbindung einer FS20- und einer HomeMatic-Steckdose via Gateways und ein Webfrontend zur Anzeige von Sensorwerten und Steuerungsmöglichkeit von Aktoren. Zentraler Bestandteil dieses Systems ist der hier entstandene Heimautomatisierungsserver mit folgenden Funktionen:

- Empfang und Versand von CoAP-Nachrichten
- Eintragung von Sensorwerten in eine Datenbank
- Bereitstellung einer Web-Schnittstelle

Besonders ins Gewicht fällt hier die Handhabung der CoAP-Nachrichten, da die Bibliothek zwar die Möglichkeit bietet, Nachrichten zu versenden, doch nicht viel mehr Funktionalität bereitstellt. Das beudetet, dass Features, wie die Observe-Funktion oder der Blockmodus, noch selbst zu implementieren sind. Ebenso muss die gesamte Fehlerbehandlung selbst implementiert werden. CoAPy übernimmt dabei nur die Unterstützung der Syntaxkontrolle. Wie mit dem Fehler verfahren wird, was im CoAP-Draft bereits festgeschrieben ist, muss noch selbst implementiert werden.

Mit der Webschnittstelle, welche erst in den letzten Wochen nur prototypisch umgesetzt wurde, kann derzeit nur unidirektional kommuniziert werden. Das bedeutet, dass über die Schnittstelle zwar eine CoAP-Nachricht gesendet werden kann, aber das anfordernde Programm eine Rückmeldung nur über die Datenbank erhalten kann.

Für den weiteren Verlauf des Forschungsseminars nach Entstehung dieses Dokuments bleiben hauptsächlich noch folgende Fragestellungen offen.

Laufzeit Wie lange lassen sich Sensorknoten unter gegebenen Bedingungen betreiben, ohne dass die Batterien gewechselt werden müssen?

Verschlüsselung Welche Art der Verschlüsselung ist praktikabel und umsetzbar?

Authentifizierung Wie wird sichergestellt, dass lediglich authorisierte Sensorknoten unserem Netz beitreten können?

In unserem Aufgabenbereich überwiegt großteils Implementierungsaufwand, offene Punkte sind:

Observe-Funktionalität Implementierung in CoAPy und Handling im Server muss umgesetzt werden. Dieses Feature von CoAP sollte aber dringend noch integriert werden, um nicht periodisch Polling-Anfragen über das Netzwerk zu senden. Mit dieser Funktionalität ist es möglich, dass Sensorknoten selbstständig ihren aktuellen Wert senden, wenn dieser sich verändert hat, oder eine gewisse Zeitspanne abgelaufen ist. Stark verknüpft mit diesem Thema ist auch die Integration von Reverse Proxys im Netzwerk, welche Sensorwerte im Cache halten könnten, um so den Funkverkehr auf batteriebetriebenen Knoten weiter zu minimieren. Auch dafür findet man im CoAP-Draft bereits ein „Freshness Model“.

CoAP-Versionsanpassung je nach aktueller Contiki-Bibliothek für CoAP gegebenenfalls Anpassung nötig.

Spezifizierung Link-Formate aktuell lediglich prototypisch umgesetzt. Drei Attribut-Definitionen sind bereits im Wiki vorhanden, doch muss noch weiter überlegt werden, welche Informationen von den Sensoren bei einer Discover Anfrage versendet werden müssen. Denn dies ist die einzige Gelegenheit, alles über die Ressource zu senden. Wichtige Informationen sind dabei beispielsweise in welcher Einheit ihre Daten vorliegen, oder wie Werte bei einem Aktor gesetzt werden dürfen. Gleichzeitig sollte diese Information über jede einzelne Ressource aber auch nicht zu groß werden, da sonst die gesamte Discover-Antwort zu groß wird.

Node- und Sensorgruppierungen die Gruppierung wurde lediglich in der Entwurfsphase beleuchtet. Derzeit wird lediglich zwischen Aktor und Sensor unterschieden. Weiterhin wird bei Aktoren über Gruppen geregelt, welche Werte zu ihnen gesendet werden dürfen.

Integration Regelwerk wegen Zeitmangel nicht umgesetzt, Nutzung der Webschnittstelle denkbar