Ansteuerung von Sensoren und Aktoren in der Hausautomatisierung

Angelos Drossos und Hermann Lorenz

***Zusammenfassung*—Eine (offene) Hausautomatisierung erfordert Sensoren und Aktoren, die von einer Steuerungseinheit angesteuert werden. Daher behandelt dieses Dokument die Ansteuerung der Sensoren und Aktoren in einem Sensornetz. Ziel ist es CoAP-Anfragen auf einem ATmega128RFA1 mit dem Betriebssystem Contiki über 6LoWPAN zu beantworten.**

***Index Terms*—freie Hausautomatisierung, Drahtloses Sensornetz, Internet der Dinge, 6LoWPAN, CoAP, Contiki-OS, ATmega128RFA1.**

I. Einführung

Die Hausautomatisierung ist ein Teilgebiet der Gebäudeautomatisierung. Ein primäres Ziel der Hausautomatisierung ist die finanzielle Einsparung, z. B. durch intelligente Heizsysteme. Zusätzlich soll sie aber auch die Bequemlichkeit der Bewohner fördern.

Im Bereich der Hausautomatisierung gibt es bereits eine Vielzahl an sowohl offenen als auch proprietären Lösungsansätzen. Jedoch erfüllen sie die Bedingungen an eine freie und erweiterbare Hausautomatisierungslösung nur bedingt.

Offene Varianten sind häufig „Ein-Mann“-Projekte, die sehr spezielle Probleme verfolgen oder nur sehr unzureichend dokumentiert sind. Teilweise stützen sie sich aber auch nur darauf, vorhandene proprietäre Systeme zu verwalten und miteinander zu verbinden.

Diese proprietären Systeme sind für Fremdanbieter häufig aufgrund hoher Lizenzkosten unattraktiv. Des Weiteren setzen sie Netzwerkprotokolle ein, die energieeffizienter als standardisierte Protokolle wie HTTP, TCP und IPv6 arbeiten. Die Protokolle stehen aber durch ihre Geschlossenheit der Interoperabilität entgegen.

Diese Probleme begründen die Notwendigkeit des hier vorangetriebenen Projektes.

A. Projektziele

Im Rahmen des Forschungsprojektes *Sensornetze* an der HTW Dresden, geleitet durch Prof. Dr.-Ing. Jörg Vogt, soll überprüft werden, inwiefern eine offene Heimautomatisierung durch Anwendung bestehender Informationstechnologien in einem Sensornetzwerk realisierbar ist.

Dabei sind für die Konzipierung im Rahmen unseres Forschungsprojektes die folgenden Zielstellungen erarbeitet worden:

- Es sollen nur quelloffene und freie Module verwendet werden. Dabei ist „frei“ im Sinne der Definition der Free Software Foundation zu sehen.
- Das Nutzen existierender, energieeffizienter Standardtechnologien [1, 2] bilden die Grundlage für die *Interoperabilität* (Herstellerunabhängigkeit) sowie die Langlebigkeit batteriebetriebener Sensorknoten.
- Die Erweiterbarkeit des Hausautomatisierungssystems und damit die Integration anderer – auch proprietärer – Sensornetze ist zu untersuchen.
- Es ist wünschenswert, dass neue Geräte sich möglichst automatisch im System anmelden. Der Ausfall einzelner Sensorknoten darf das System nicht gefährden. Somit ist eine Skalierbarkeit des Hausautomatisierungssystems von Nöten.

B. Projektansatz

Die konkret untersuchte Hausautomatisierungslösung besteht aus einem dedizierten Netzknoten (Server), der die Regelung des Systems übernimmt. Auf diesem befindet sich folglich auch die Regelverarbeitung.

Das Sensornetz soll auf dem Funkstandard 6LoWPAN basieren. Als Sensorknoten sollen ATmega128RFA1-Mikrocontroller mit dem Betriebssystem Contiki-OS eingesetzt werden.

Der Informationsaustausch zwischen den einzelnen Sensorknoten und dem Server soll über das Constrained Application Protocol (CoAP) realisiert werden. CoAP ist ein im Vergleich zu HTTP energieeffizientes Protokoll, welches dennoch URIs unterstützt.

Bestehende Sensornetze, wie FS20 oder HomeMatic, werden mit Hilfe von Gateways integriert, welche ein Mapping zwischen dem entsprechenden Sensornetzprotokoll und CoAP durchführen.

C. Projekteinteilung

Zur Konzipierung der offenen Hausautomatisierung wurde das Projekt in folgende Teilprojekte gegliedert:

1) Steuerung des Systems: Es gilt zu untersuchen, ob ein zentraler oder dezentraler Hausautomatisierungsansatz in Hinblick auf die Bedürfnisse der Nutzer des Systems sinnvoll ist. Danach ist die Steuerung des Systems zu entwickeln und ein Anwendungsprotokoll zu finden, das sowohl im Sensornetz als auch auf normalen Computern eingesetzt werden kann.

2) Regelverarbeitung: Sofern die Steuerung des Systems definiert ist, kann mit der Regelverarbeitung begonnen werden. Hierbei sind die Fähigkeiten verschiedener Nutzer besonders zu berücksichtigen und eine einfach konfigurierbare Regelverarbeitung zu finden.

3) Sensornetz-Kommunikation: Neben der Steuerung muss die Kommunikation im Sensornetz analysiert werden und es muss ein Netzwerkstack gefunden werden, der sowohl die Anforderungen an das Hausautomatisierungssystems erfüllt wie auch diejenigen Anforderungen an *Tiny-Networked-Sensors* [3].

4) Sensoransteuerung: Um im Netzwerk Sensordaten auswerten zu können, muss untersucht werden, wie Sensoren und Aktoren auf dem im Ansatz beschriebenen Sensorknoten angesteuert und an die CoAP-Schnittstelle weitergeleitet werden können.

5) Integration bestehender Sensornetze: Wie in den Zielstellungen definiert, sollen grundsätzlich bestehende Sensornetze verwendet werden können. Beispielhaft wurden hierbei FS20- und HomeMatic-Geräte in das Hausautomatisierungssystem eingebunden.

D. Dokumentabgrenzung

Dieses Dokument soll sich lediglich mit dem Teilprojekt der Sensoransteuerung beschäftigen.

Dabei soll dargestellt werden, welche Möglichkeiten zur Ansteuerung der Peripherie existieren. Des Weiteren sollen die Möglichkeiten unter Contiki untersucht werden und die Auswirkungen auf das Kommunikationsverhalten des Sensorknotens dargestellt werden.

Der Einsatz von CoAP als Kommunikationsprotokoll wird dabei nicht detailliert untersucht. Es wird jedoch als Rahmenbedingung für den Einsatzzweck herangezogen.

Durch den Einsatz der batteriebetriebenen ATmega128RFA1-Mikrocontroller als Sensorknoten müssen insbesondere die knappen Ressourcen Speicher und Energie beachtet werden.

II. Physikalische Ansteuerung

Die Ansteuerung der Sensoren kann in zwei grundsätzliche Klassen eingeteilt werden:

direkt Der Sensor liefert eine durch den Mikrocontroller messbare elektrische Größe. So wird bei einem Schalter durch einen Flankenwechsel ein Interrupt ausgelöst oder ein regelbarer Widerstand liefert unterschiedliche Spannungswerte, die durch einen Analog-Digital-Wandler abgetastet werden können.

interfacebasiert Die Peripherie wird über eine standardisierte Schnittstelle, wie UART oder I^2C, angesprochen.

Die Ansteuerungen von Aktoren lässt sich analog klassifizieren. Hierbei reagiert der Aktor auf eine anliegende Spannung oder Flanke bzw. ihm wird seine auszuführende Aktion über einen Befehl mitgeteilt.

Diese Klassen sollen im Folgenden näher beleuchtet werden.

A. Direkte Ansteuerung

Die direkte Ansteuerung von Sensoren ist verschiedentlich mit Problemen behaftet.

Tabelle I
Messwerte bei einer Referenzspannung von 1,6 V

$U_{anliegend}$ in V	*erwartet*	*gemessen*
0,050	8	7
0,091	15	13
0,136	22	20
0,169	27	26
0,242	39	37
0,314	50	48
0,346	55	53
0,516	82	80
0,596	95	92
0,799	127	124
0,884	141	137
1,020	163	159
1,170	187	183
1,268	202	198
1,296	207	202
1,353	216	211

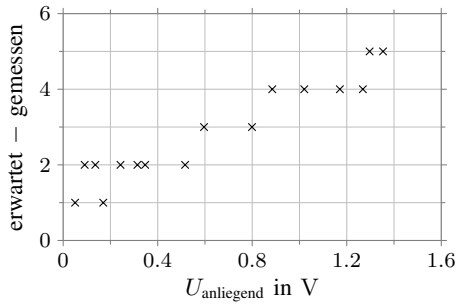

Abbildung 1. Differenz zwischen Erwartungs- und Messwerten nach Tabelle I

1) Spannungswerte: Der Analog-Digital-Wandler leidet unter vorhersehbaren und unvorhersehbaren Messfehlern.

Unvorhersehbare Messfehler ergeben sich durch elektromagnetische Störungen, die durch interne und externe Schaltkreise entstehen. [4, S. 422] Die Unvorhersehbarkeit besagt nicht, dass diese Fehler nur selten auftreten – vielmehr treten sie sogar häufig auf. Lediglich die konkrete Auswirkung des Fehlers ist nicht vorhersehbar.

Diese Fehler können verringert werden, indem nicht benötigte Komponenten abgeschalten werden und der „ADC Noise Reduction Mode“ [4, S. 159] genutzt wird. Üblich ist es auch mehrere (z. B. 16 oder 128) Messungen direkt hintereinander durchzuführen und einen Mittelwert zu bilden. Weitere Möglichkeiten sind hier auch in [4, S. 422] beschrieben.

Dem stehen die vorhersehbaren Fehler entgegen. Hier ist zum einen ein festgestellter systematischer Messfehler des ATmega128RFA1 zu nennen. So weichte der Messwert im unteren Wertebereich wenig und mit steigendem Wertebereich weiter ab. (siehe Abbildung 1 und Tabelle I) Der Fehler wurde noch nicht näher bestimmt und sollte weiter untersucht werden, ob dies von Modell zu Modell oder sogar von Gerät zu Gerät abweicht. Letzteres wäre katastrophal, da sich dann für jedes einzelne Gerät die Notwendigkeit einer Kalibrierung ergeben würde.

Die Konvertierung der Spannung erfolgt durch sukzessive Approximation [4, S. 412] anhand einer Referenzspannung. Ergibt sich die zu messende Spannung aus der Betriebsspannung,

so ist beim Batteriebetrieb des Sensorknotens ein Absinken der Betriebsspannung im Laufe der Betriebszeit zu beachten.

Soll ein *Aktor* mittels analoger Spannungswerte angesteuert werden, so kann dies durch Pulsweitenmodulation erreicht werden. Auf dem ATmega128RFA1 können mit den Timern 0 und 2 Spannungswerte zwischen Ground und Vcc moduliert werden.

2) Interrupts: Mittels Interrupts kann auf sich verändernde anliegende Signale an einem Eingangspin reagiert werden. Der ATmega128RFA1 kann dabei auf steigende, fallend oder beide Flanken reagieren. Zusätzlich kann er auch Interrupts auslösen, solange an dem Pin ein Low-Signal anliegt. [4, S. 220]

Problematisch ist dabei, dass der Programmierer selber die Signale entprellen muss. Außerdem wurde ein eigenartiges Verhalten der Interrupts 0 und 1 festgestellt. Jeder Interrupt funktioniert für sich. Werden jedoch beide gleichzeitig verwendet, so kommt es zu folgender Situation: Wird hardwareseitig der Interrupt 0 ausgelöst, so wird einmal die Interrupt-Service-Routine des Interrupts 0 und zweimal die Interrupt-Service-Routine des Interrupts 1 ausgelöst. Ebenso trat dieses Phänomen andersherum auf – bei einem hardwareseitigen Auslösen des Interrupts 1 wurde einmal die Interrupt-Service-Routine des Interrupts 1 und zweimal die Interrupt-Service-Routine des Interrupts 0 ausgelöst. Die Ursache für dieses Verhalten muss noch weiter untersucht werden. Dabei sollte auch geklärt werden, ob dies für die anderen Interrupts ebenso zutrifft.

Um *Aktoren* über Flanken oder High-/Low-Pegel zu steuern, können die Pins als Ausgang deklariert werden und dann bei Bedarf an- und ausgeschalten werden.

B. Interfacebasierte Ansteuerung

Im Gegensatz zur in Unterabschnitt II-A beschriebenen direkten Ansteuerung, werden bei der interfacebasierten Ansteuerung die eigentlichen Messwerte von einem externen Schaltkreis bestimmt und dem Mikrocontroller nur noch über ein Interface zur Verfügung gestellt. Die Abstraktion zum physikalischen Problem der Messung ist also höher.

Der ATmega128RFA1 bietet für die Schnittstellen UART, I^2C und SPI eine direkte Hardwareunterstützung. Andere Schnittstellen müssen selber mit ihren Timings programmiert werden. Über UART können nur zwei Geräte miteinander Daten austauschen. I^2C und SPI ermöglichen es, auf einem Anschluss (nacheinander) mit verschiedenen angeschlossenen Sensoren und Aktoren zu kommunizieren.

Bei den Protokollen kann man zwischen Binär- und Textprotokollen unterscheiden.

1) Binärprotokolle: Binärprotokolle sind relativ kompakt. Der Master schickt einen Befehl / eine Anforderung an den Sensor bzw. Aktor, der dann direkt antwortet.

2) Textprotokolle: Textprotokolle sind vielseitiger als Binärprotokolle. Dafür benötigen die Befehle und Antworten mehr Bytes bei der Übertragung. Die Übertragung dauert also länger und verbraucht damit mehr Energie. Zusätzlich muss die Nachricht (umständlich) geparst und Zahlen von ASCII-Code in Integerwerte (und umgekehrt) überführt werden, um sie weiter verarbeiten zu können. Auch dieses Parsen erhöht sowohl den Energiebedarf als auch den Speicherbedarf für Zeichenklassen- und Zustandsübergangstabellen. Wenn außerdem die Länge der einzelnen Nachrichten nicht explizit durch das Protokoll begrenzt ist, muss dynamisch Speicher zum Puffern der eingehenden Nachricht reserviert werden, was sich ungünstig auf speicherarme Systeme wie Mikrocontroller auswirkt.

III. Konzepte der Ansteuerung in Contiki-OS

Zielplattformen von Contiki-OS sind eingebettete, batteriebetriebene Systeme, die über Funk (oder andere Netzwerke) miteinander kommunizieren. [3, 5]

Contiki bietet eine Reihe von Modulen, die es für den Einsatz in einem drahtlosen Sensornetz prädestiniert. Für die Ansteuerung der Sensoren ist das Sensorinterface sehr interessant, weshalb es näher beleuchtet wird.

Da im Sensorinterface die Sensoren physikalisch angesteuert werden müssen und im Forschungsprojekt viele I^2C-Sensoren verwendet wurden, wird ebenfalls ein I^2C-Interface und seine Eigenheiten betrachtet.

Bevor allerdings das Sensor- bzw. I^2C-Interface erläutert wird, soll die Struktur von Contiki-OS [6] kurz vorgestellt werden, um zu verstehen, wo sich welches Interface bzw. welche Implementation befindet.

A. Struktur von Contiki-OS

Da Contiki viele verschiedene Plattformen unterstützen möchte, wird generischer Code von plattformabhängigem Code (strikt) getrennt. Dabei gliedert sich Contiki in folgende Abschnitte:

core Der Kern von Contiki arbeitet *plattformunabhängig*. Der Netzwerk-Stack ist beispielsweise bis auf die Ansteuerung des Funksensors komplett enthalten. Das Interface für die Ansteuerung des Funksensors befindet sich aber wiederum im Kern.

cpu In diesem Abschnitt befindet sich *CPU-abhängiger Code*. Es wird zwischen AVR, ARM und weiteren CPUs unterschieden. Hier ist i. A. die interfacebasierte physikalische Ansteuerung implementiert, da Funktionalitäten des Mikrocontrollers genutzt werden. Zugehörige Schnittstellen können sich im Kern befinden.

platform An dieser Stelle werden *plattformspezifische Eigenschaften* umgesetzt. Vor allem befindet sich die Main-Methoden in diesem Bereich. Eine Plattform kann auch für einen bestimmten Mikrocontroller wie den ATmega128RFA1 stehen.

apps Wenn Funktionalitäten nicht im Kern benötigt werden und sie dennoch plattformunabhängig sind, so kommen diese hier hin. Die CoAP/REST-Implementierung befindet sich beispielsweise hier, da es sich nur um eine Anwendungsschicht im Netzwerk-Stack handelt.

examples / project Der letzte Bereich organisiert ein spezielles Projekt. An dieser Stelle wird festgelegt, welche Plattform verwendet wird und welche Prozesse automatisch gestartet werden. Ein spezielles Projekt könnte ein Sensorknoten sein, welcher Sensoren zum Auslesen der Raumtemperatur sowie der Luftfeuchtigkeit bietet und diese Informationen über Funk (6LoWPAN) verteilen kann.

Listing 1. Das I²C-Interface (Ausschnitt)

```
// The result of a communication
typedef enum i2c_result_t {
        SUCCESSFUL = 0,
        TRANSMISSION_PENDING,
        FAILED_unknown,
        FAILED_ALLOC,
        FAILED_TOO_MUCH_DATA,
        FAILED_BUS_ERROR,
        FAILED_ADDR,
        FAILED_DATA,
        FAILED_NOT_SUPPORTED,
        FAILED_INTERRUPT
} i2c_dev_result_t;

// Send one command to the given device
// (or write null bytes to the register address of the given device)
i2c_dev_result_t
i2c_dev_send_cmd(const uint8_t device_address_rw,
        const uint8_t register_address_command,
        bool is_repeated_start, bool no_stop);

// Write data to a register address field of the given device
i2c_dev_result_t
i2c_dev_write_data(const uint8_t device_address_rw,
        const uint8_t register_address,
        const uint8_t *const register_content_data,
        const size_t register_size,
        bool is_repeated_start, bool no_stop);

// Read data from a register address field of the given device
i2c_dev_result_t
i2c_dev_read_data(const uint8_t device_address_rw,
        const uint8_t register_address,
        uint8_t *const register_content_data,
        const size_t register_size,
        bool is_repeated_start, bool no_stop);
```

B. I2C-Interface

Im Forschungsprojekt wurden vor allem Sensoren verwendet, die eine interfacebasierte Ansteuerung über I²C erlauben.

I²C[1] ist eine serielle Schnittstelle, die den Anschluss mehrerer I²C-Devices ermöglicht. So konnte sowohl ein Luftfeuchtigkeits-, ein Druck-, ein Beschleunigungs- sowie ein Lichtsensor an einem Bus angeschlossen werden. Am Mikrocontroller werden für I²C nur zwei Pins (I²C-SCL, I²C-SDA) benötigt. Der Mikrocontroller fungiert hierbei als Master und leitet die Übertragung der Sensordaten ein.

In Contiki wurde kein I²C-Interface gefunden. Da I²C keine generische Implementation aufgrund der Verwendung von Mikrocontroller-Komponenten erlaubt, wurde ein eigenes I²C-Interface in den Kern von Contiki integriert. Eine zugehörige Implementation erfolgte dann im CPU-abhängigen Teil von Contiki.

Dieses I²C-Interface erlaubt die Verwendung des I²C-Busses, ohne detaillierte Kenntnisse zum Ablauf der Kommunikation zu haben. Allerdings ist der Multi-Master-Betrieb nicht berücksichtigt worden, da dieser für die Ansteuerung von Sensoren nicht benötigt wird. Das Interface ist in Listing 1 zu sehen und zeigt die drei benötigten Funktionen zum Ansteuern eines Sensors sowie die möglichen Übertragungsfehler.

Die AVR-Implementation des I²C-Interfaces benutzt den vorgesehenen I²C-Interrupt, um die Übertragung zügig abarbeiten zu können. An welchem Pin sich der I²C-Bus befindet, wird durch die AVR-Libc bestimmt.

[1] I²C wird im AVR Reference Manual [4] auch Two-Wire Interface (TWI) genannt.

Die AVR-Implementation sieht nur eine synchrone Übertragung vor, das heißt, dass das Betriebssystem so lange warten muss, bis die Übertragung abgeschlossen oder ein Fehler aufgetreten ist. Durch die Verwendung des I²C-Interrupts wird die Übertragung sogar bevorzugt abgearbeitet bzw. kann i. d. R. nicht durch andere Interrupts unterbrochen werden.

I²C-Sensoren können in Bezug auf das Übertragungsverhalten in zwei Klassen eingeteilt werden: Die ersten verwenden einen Hold-Mode[2] und die anderen einen No-Hold-Mode.

Hold-Mode Dieses synchrone Übertragungsverhalten hat bei I²C-Sensoren, die im so genannten Hold-Mode arbeiten, den Nachteil, dass während des Messens des Sensorwertes der I²C-Bus blockiert ist und damit auch das Betriebssystem warten muss, bis die Übertragung abgeschlossen ist.
Insgesamt wird mehr Energie verbraucht. Auf der anderen Seite werden keine Mechanismen gebraucht, die anzeigen, dass eine Übertragung bereits stattfindet.

No-Hold-Mode Die zweite Möglichkeit der Übertragung ist der No-Hold-Mode. Bei diesem wird dem Sensor mitgeteilt, dass ein Sensorwert gemessen werden soll. Der Mikrocontroller muss dann beim Sensor nach einer gewissen Messdauer den Sensorwert abfragen (pollen), sofern er bereits vorliegt.
Viele Sensoren bieten für den No-Hold-Mode auch eine Interrupt-Leitung an, die anzeigt, wann der neue Sensorwert ausgemessen ist und somit abgefragt werden kann.
Auch hier ist das synchrone Übertragungsverhalten sinnvoll, da der jeweilige Prozess die Übertragung einleitet und damit auch sofort das Ergebnis der Übertragung vorliegen hat. Nach einer bestimmten Zeit oder beim Eintreten eines externen Interrupts wird dann der Sensorwert abgerufen.

Abschließend ist zu sagen, dass es ratsam ist, zur Ansteuerung der I²C-Sensoren ein Sensorinterface zu benutzen. Mit diesem lassen sich dann alle konfigurierbaren Optionen einstellen, wie den Hold- oder No-Hold-Mode.

C. Das generische Sensorinterface

Contiki besitzt ein *generisches* Sensorinterface, um Sensoren ansteuern zu können. Damit ist eine Abstraktion geschaffen, die die Entwickler der Sensortreiber dazu bewegen soll, ein einheitliches Schema zu nutzen. Entscheidend bei diesem Interface sind drei Funktionen:

- Die erste Funktion (`configure`) konfiguriert den Sensor;
- die zweite (`status`) gibt Informationen zum Sensor zurück, beispielsweise die konfigurierten Einstellungen oder den Zustand des Sensors;
- die dritte (`value`) kann die Sensorwerte auslesen, beispielsweise den gemessenen Temperaturwert oder die Lichtintensität.

Zusammen mit einen Namen für den Sensor bilden diese vier Bestandteile den Sensortreiber.

[2] Der Sensor *hält* den I²C-Bus während der Messung des Sensorwertes, so dass die Kommunikation nicht abgeschlossen oder eine andere nicht begonnen werden kann. Der Feuchtigkeitssensor SHT21 unterstützt sowohl den Hold- wie auch den No-Hold-Mode.

Eine frühe Version des Interfaces erforderte weitere Funktionen vom Sensortreiber: Hierunter war eine Initialisierungs-, Aktivierungs- und Deaktivierungsfunktion. Diese Funktionen können aber durch die Konfiguration des Sensors übernommen werden, weshalb sie aus dem Interface entfernt wurden.

1) Das Konfigurieren: Die Funktion `configure` dient dem Konfigurieren. Sie besitzt zwei Parameter, wovon der erste den Konfigurationstyp und der zweite den Konfigurationswert angibt. Ein Rückgabewert kann anzeigen, ob die Konfiguration erfolgreich war.

Was bei einem Sensor konfiguriert werden kann und was sinnvoll zu konfigurieren ist, muss jeder Sensortreiber für sich entscheiden. Es gibt lediglich die Begrenzung, dass die beiden Parameter sowie der Rückgabewert Integers sind (und damit abhängig vom Mikrocontroller durchaus unterschiedlich groß sein können).

Das Sensorinterface definiert globale Konfigurationstypen, wovon der wichtigste `SENSORS_ACTIVATE` ist. Dieser sollte das Aktivieren und Deaktivieren des Sensortreibers und sinnvoller Weise auch des Sensors bewirken. Beispielsweise kann der Sensor mit Hilfe dieses Konfigurationsparameters in den Sleep-Mode versetzt werden.

2) Der Auslesen der Status: Die Funktion `status` dient dem Auslesen der Status eines Sensortreibers. Diese Funktion gibt den Status als Rückgabewert zurück. Der gewünschte Status wird wie bei der Konfiguration per Parameter übergeben. Es ist daher nicht undenkbar, dass die beiden Parameter, Konfigurationstyp und Statustyp, dieselben Wertigkeiten behandeln.

Auch hier ist es sehr zu empfehlen, bei Übergabe von `SENSORS_ACTIVATE` den Zustand des Sensortreibers/Sensors zurückzugeben, wobei „0“ bedeutet, dass der Sensor aus ist. Eine „1“ gibt die Aktivität des Sensors an.

3) Das Auslesen von Sensorinformationen: Die Funktion `value` bewirkt, dass Sensorinformationen ausgelesen werden. Der übergebene Parameter kennzeichnet den auszulesenden Sensorwert, beispielsweise einen Temperaturwert.

Da der Rückgabetyp vom Typ Integer[3] ist, ist es nicht möglich jeden beliebigen Sensorwert zurückzugeben. Es gibt Sensorwerte, die einen größeren Wertebereich benötigten. Ein Beispiel sind Temperaturwerte: Da die Typen Float oder Double auf Mikrocontrollern so oft wie möglich vermieden werden sollten, wird die Temperatur $22.53\ ^\circ\mathrm{C}$ typischerweise als ganze Zahl (z. B. 2253) übergeben – was aber den Wertebereich einer Integerzahl (0–255) auf dem ATmega128RFA1 überschreitet.

Hinzu kommt, dass ein fehlerhaftes Verhalten ebenfalls im Rückgabewert eingebunden werden muss. Ein durchaus oft auftretender Fehler bei batteriebetriebenen Netzknoten resultiert aus einer zu niedrigen Betriebsspannung: Der Mikrocontroller ATmega128RFA1 kann noch mit $1.8\ \mathrm{V}$ arbeiten, aber einige Sensoren wie der SHT21 stellen bei $2.1\ \mathrm{V}$ den Betrieb ein, so dass ein Auslesen des Sensorwerts zwangsweise fehlschlagen wird.

Aus diesem Grund empfiehlt es sich, dass der Rückgabetyp kennzeichnet, ob ein Fehler aufgetreten ist oder nicht. Bei bereits vorhandene Sensorimplementationen wurde „0“ für fehlerhaftes sowie „1“ für fehlerfreies Verhalten des Sensors verwendet. Prinzipiell ist dieses (bisher) nicht festgelegt, so dass auch eine „0“ für fehlerfreies Verhalten stehen könnte und ein von Null verschiedener Wert für den Fehlertyp.

Eigene Tests haben gezeigt, dass es sich empfiehlt, den Sensorwert im Sensortreiber zwischenzuspeichern. Dieses kann auch in einem RAW-Format passieren, so dass erst das Auslesen des zwischengespeicherten Sensorwerts diesen in das gewünschte Format konvertiert.

Ein weiterer Vorteil ist es, dass der Sensortreiber alle Sensorwerte intern in einer beliebigen Datenstruktur ablegen kann, so dass auch historische Werte – ggf. mit Zeitstempel – abrufbar sind. Der Sensortreiber stellt dann die benötigten Funktionen zum Auslesen der Sensorwerte bereit.

Zur Zeit ist keine Abstraktion zum Übergeben des Sensorwertes vom Sensorinterface zum Aufrufer vorgesehen. Es wäre also interessant, ob solch eine Abstraktion in Bezug zur Hausautomatisierung existiert und wie das vorhandene, generische Sensorinterface erweitert werden kann, um die Anforderungen der Ansteuerung auf Anwendungsebene zu erfüllen.

IV. Ansteuerung auf Anwendungsebene

In einem Hausautomatisierungssystem sollen Sensorinformationen sowie Steuerbefehle für Aktoren übertragen werden. Daher ist es erforderlich, auf den Netzknoten eine Anwendungsschicht zu besitzen, die diese Informationen bzw. Befehle versenden und empfangen kann.

An dieser Stelle geht es also nicht um die Umsetzung der genauen Befehle an die Sensoren und Aktoren oder die Programmierabstraktion, sondern um eine sinnvolle Abstraktion auf Netzwerkebene, so dass andere Netzknoten alle nötigen Informationen zur Weiterverarbeitung vorliegen haben.

Wichtige Informationen neben dem aktuellen Sensorwert können Folgende sein:

- der Name des Sensors,
- der Typ des Sensors (z. B. Temperatur, Lichtintensität)
- den möglichen Wertebereich,
- Ungenauigkeiten.

Ein Temperatursensor liefert einen Temperaturwert im Bereich von $-40\ ^\circ\mathrm{C}$ bis $+100\ ^\circ\mathrm{C}$. Dem Regelverarbeitungssystem genügt es unter Umständen, dass nur (starke) Temperaturschwankungen aufgezeigt werden, z. B. durch das Öffnen eines Fensters an kalten Tagen. Damit ist es überflüssig, dass jede Sekunde der Sensorwert durch die Regelverarbeitung gepollt[4] wird, da ein Fenster an kalten Tagen nur selten geöffnet wird.

Das Pollen der Sensorwerte, also der Temperaturwerte, bewirkt folglich ein Überfluten des Sensornetzes mit i. A. nicht benötigten Informationen.

Hierfür müssen Mechanismen zur Ansteuerung auf Anwendungsebene bereitgestellt werden, die die Netzlast reduzieren und somit Energie für die Übertragung und Auswertung der Nachrichten eingespart werden kann.

[3] Ein Integer ist auf dem ATmega128RFA1 nur $8\ \mathrm{bit}$ breit. Auf anderen Architekturen kann ein Integer breiter sein.

[4] „Pollen“ bezieht sich in diesem Abschnitt auf die Kommunikation zwischen Regelverarbeitungsserver und Sensorknoten, nicht aber auf ein Pollen zwischen Sensorknoten und Sensor.

Daneben gibt es auch Situationen, in denen eine Änderung der Sensorinformation zeitkritisch verarbeitet werden muss, das heißt, es kann nicht abgewartet werden, bis die Regelverarbeitung das nächste Mal den Sensorwert pollt. Hierzu muss der Regelverarbeitungsserver selbstständig benachrichtigt werden, damit eine schnelle Abarbeitung erfolgen kann.

A. CoAP

Ein Anwendungsprotokoll, dass die Anforderungen der Ansteuerung auf Anwendungsebene erfüllen kann, ist CoAP. Es ist ein asynchrones Protokoll, weshalb UDP als Transfer Protokoll ausreicht. Da CoAP sich noch in Entwicklung befindet, ist keine Kompatibilität zu vorherigen Draft-Versionen sichergestellt.

Zum Zeitpunkt des Forschungsprojekts befand sich CoAP bei Draft-12 bzw. -13. Die implementierten Draft-Versionen von CoAP waren meist auf unterschiedlichen Ständen. So bot Contiki 2.5 bei Projektbeginn nur Draft-03. Nach Veröffentlichung von Contiki 2.6 war auch Draft-07 implementiert. [7]

Seitens des Hausautomatisierungsservers sah es ähnlich aus: Die CoAP-Implementation auf dem Hausautomatisierungsserver (CoAPy) wurde der Contiki-Version angepasst, um eine Kommunikation herzustellen.

Das Anwendungsprotokoll CoAP bietet verschiedene Vorteile für die Hausautomatisierung im Allgemeinen und für die Ansteuerung der Sensoren und Aktoren im Konkreten. Diese Vorteile werden in den folgenden Abschnitten näher erläutert.

Weitere Informationen zu CoAP sind der Projektgruppe „Steuerung des Systems“ sowie im Speziellen zu Contiki der Gruppe „Sensornetz-Kommunikation“ zu entnehmen.

B. Kommunikationsbeispiele

Mit dem Discovery Mechanismus von CoAP ist es möglich, die Ressourcen des Sensorknotens zu erfahren. Ein Sensorknoten mit dem Sensor SHT21 kann sowohl die Luftfeuchtigkeit wie auch die Temperatur messen. Damit können für diesen Sensorknoten zwei REST-Ressourcen angelegt werden: `sensor/sht21/temperature` und `sensor/sht21/humidity`. Der Sensorknoten könnte aber auch noch andere Ressourcen bieten, wie eine Einstellungsoption, die es erlaubt, den Radio Duty Cycle zu verändern oder Einstellungen am SHT21 vorzunehmen.

Es können zwei grundsätzliche Verfahren unterschieden werden, um auf ein Request, der an den Sensorknoten (CoAP-Server) gerichtet ist, zu antworten:

Piggy-backed Response Bei einem Request wird zuerst der Sensorwert ausgelesen und dann zusammen mit dem ACK als Response übertragen werden. Hierbei wird der Netzwerktraffic reduziert, erfordert aber eine schnelles Auslesen des Sensorwertes.

Separate Response Die andere Möglichkeit ist es, sobald ein Request eingetroffen ist, den Empfang mit einem ACK zu bestätigen. Danach wird der Sensorwert erfasst und als Response zum CoAP-Client geschickt, der dann den Empfang mit einem ACK bestätigt.

Den Zustand einer LED sollte folglich nur mit einem *Piggy-backed Response* beantwortet werden, um den Netzwerktraffic gering zu halten.

Für das Auslesen des Feuchtigkeitssensors SHT21 wird je nach I^2C-Bus-Geschwindigkeit etwa $50\ \mu s$ benötigt. Hier kann man je nach Anwendungsfall entscheiden, welche Response-Methode besser ist.

Für den Lichtsensor TSL2561 eignet sich im Prinzip nur die zweite Variante, denn dieser benötigt durchaus $400\ ms$ zum Messen der Lichtintensität. Es gäbe zwar die Möglichkeit eine kürzere Messdauer einzustellen, allerdings kann dadurch der Messfehler sich erhöhen. Zur Beantwortung des Requests der aktuellen Lichtintensität ist ein separates Response wichtig, denn der CoAP-Client – i. d. R. ist dies der Regelverarbeitungsserver – wird im Falle eines Piggy-backed-Response mehrere Requests losschicken, da er davon ausgehen muss, dass die Nachricht nicht beim CoAP-Server angekommen ist.

Bei zeitkritischen Ereignissen – wie das Öffnen eines Fensters oder der Wohnungstür – möchte der CoAP-Client nicht ständig den Sensorknoten pollen müssen. Daher bietet CoAP auch die Möglichkeit, eine Ressource zu überwachen, das Observe-Feature. Bei diesem Feature teilt ein CoAP-Client dem Sensorknoten mit, dass der Client selbstständig informiert werden soll, sobald sich der Zustand einer Ressource ändert.

In Contiki sind in der REST-Implementierung für dieses Observe-Feature zwei Ressource-Typen zur Verfügung gestellt worden: Der erste Typ ist eine periodische Ressource, die folglich periodisch den Zustand der Ressource prüft und i. d. R. nach Überschreiten eines Schwellwertes eine Response-Nachricht an alle eingetragenen CoAP-Clients verschickt. Damit kann das Sensornetz entlastet werden, da nicht jeder Sensorwert übertragen wird. Der andere Typ ist eine eventbasierte Ressource, die aufgrund des Eintretens eines vorher definierten Events eine Response-Nachricht ausgibt. Das Event kann beispielsweise das Drücken eines Buttons sein oder aber ein PUT-Request an eine andere Ressource.

Bezogen auf den Türkontaktsensor kann nun eine Eventressource angelegt werden: Beim Öffnen der Tür wird dann der Sensorknoten per Interrupt aus seiner Schlafphase erwachen und sofort eine CoAP-Nachricht an alle eingetragenen CoAP-Clients verschicken, um anzuzeigen, dass die Tür geöffnet wurde.

C. CoAP Schnittstelle

Da ein paar Kommunikationsbeispiele genannt wurden, soll nun die CoAP Schnittstelle in Contiki beispielhaft vorgestellt werden.

Die eventbasierte sowie die periodische Ressource werden hierbei nicht als Quellcode vorgestellt, da die Möglichkeiten hierbei sehr vielfältig sein können und dieses Observe-Feature noch nicht vollständig im Projekt untersucht werden konnte.

Der REST Server kann in Contiki als Prozess[5] gestartet werden. Die Initialisierungsphase des Servers, welche in Listing 2 gezeigt ist, erfolgt in drei Schritten:

1) Zuerst müssen alle Komponenten in den Ressourcen initialisiert werden, sofern dies noch nicht geschehen ist.

[5]Prozesse sind in Contiki Protothreads [8–10]

Listing 2. CoAP REST Serverprozess

```
PROCESS(rest_server, "CoAP REST Server");
PROCESS_THREAD(rest_server, ev, data)
{
        PROCESS_BEGIN();
        // init hardware components, needed in resources
        // init rest engine
        rest_init_engine();
        // activate rest resources
        rest_activate_resource(&restext);
        rest_activate_resource(&ressensor_sht21Temp);
        // endless loop
        while (1) PROCESS_WAIT_EVENT();
        PROCESS_END();
}
```

Listing 3. Text Ressource

```
RESOURCE(restext, METHOD_GET, "uri/to/text",
                "title=\"HelloWorld: ?len=0..\" rt=\"Text\"");
restext_handler(void *request, void *response,
        uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{
        const char *const message = "HelloWorld";
        int length = 0;
        const char *length_string;

        // query variables
        if (REST.get_query_variable(request, "len", &length_string))
        {
                // length query variable is available
                length = atoi(length_string);
                if (length < 0) length = 0;
                else if (length > REST_MAX_CHUNK_SIZE)
                        length = REST_MAX_CHUNK_SIZE;
        }

        // payload preparation
        memcpy(buffer, message, length);

        // REST: piggy-backed response
        REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
        REST.set_header_etag(response, (uint8_t *) &length, 1);
        REST.set_response_payload(response, buffer, length);
}
```

2) Danach wird die REST Engine initialisiert und
3) zuletzt werden alle benötigten Ressourcen aktiviert.

Damit Ressourcen aktiviert werden können, müssen diese zuvor auch angelegt werden. In Contiki geschieht dies mit dem Makro `RESOURCE`. Der erste Parameter definiert den Namen der Ressource, welcher zugleich auch im Handler vorkommt. Der zweite Parameter kennzeichnet, welche *REST-Method-Types* von der Ressource unterstützt werden. Aus HTTP bekannte Method-Types sind `METHOD_GET`, `METHOD_POST`, `METHOD_PUT`, `METHOD_DELETE`. Auf eine Beschreibung wird an dieser Stelle verzichtet und auf die Dokumentation von CoAP [7] verwiesen. Der dritte Parameter gibt den URI-Pfad an und der letzte die Attribute. Es besteht die Möglichkeit, eine Ressource auch als Subressource zu definieren.

In Listing 3 ist eine Text-Ressource gezeigt. Dieser Ressource kann ein CoAP-Client eine Wunschlänge per Query-Variable übergeben.

Bei einer einfachen Sensor-Ressource genügt die GET-Methode, so dass ein Client die Ressource lediglich auslesen kann. Die Piggy-backed Variante ist in Listing 4 gezeigt.

Eine Erweiterung des in Listing 4 gezeigten Beispiels könnte die Einführung einer Query-Variable sein, welche anzeigt, ob der Temperaturwert in Grad Celsius oder Grad Fahrenheit übertragen werden soll. Hierzu sind keine zwei Ressourcen

Listing 4. Sensor Ressource

```
RESOURCE(ressensor_sht21Temp, METHOD_GET, "uri/to/sensor/sht21",
        "rt=\"Sensor\"");
ressensor_sht21Temp_handler(void *request, void *response,
        uint8_t *buffer, uint16_t preferred_size, int32_t *offset)
{
        // read sensor value
        int length = 10; // <= REST_MAX_CHUNK_SIZE !
        char message[10];
        sht21_sensor.value(SHT21_SENSOR_TEMP);
        sht21_sensor_getLastTemperature(message, length);

        // REST: piggy-backed response
        REST.set_header_content_type(response, REST.type.TEXT_PLAIN);
        REST.set_header_etag(response, (uint8_t *) &length, 1);
        REST.set_response_payload(response, message, length);
}
```

notwendig.

Soll die Observe Funktion von CoAP genutzt werden, dann stehen hierzu zwei weitere Ressource-Typen bereit, die die normale Ressource erweitern: Die periodische sowie die eventbasierte Ressource. Beide Ressourcetypen haben einen zusätzlichen Handler. Ein CoAP-Client kann dieser Ressource überwachen und wird damit in eine Liste aufgenommen.

In diesem Abschnitt wurde lediglich ein Einblick in die CoAP-Schnittstelle gegeben. Die Implementation der CoAP-Funktionalität ist in Contiki zweigeteilt: Die Ressourcen und die REST Engine werden von einem *Abstraction Layer* bereitgestellt, der sich in Contiki *Erbium* nennt und sich im Apps-Ordner befindet. Dieser stellt die *RESTful Web services* zur Verfügung.

Zur Kompilierzeit wird bestimmt, welche CoAP-Implementation genutzt wird. Die Implementationen sind ebenfalls im Apps-Ordner von Contiki zu finden: Es ist die Draft-Version 03 (er-coap-03) und 07 (er-coap-07) implementiert.

Sowohl Abstraction Layer wie auch die beiden Draft-Implementationen stammen vom selben Autor, Matthias Kovatsch. Ziel der Implementation ist ein Low-Power-CoAP [7] für Contiki bereitzustellen.

D. Zusammenhang zwischen CoAP und RDC

Mit dem im vorherigen Absatz genannten Beispiel des Türkontaktsensors kann man einen Zusammenhang zwischen CoAP und dem Radio Duty Cycle (RDC) erkennen:

Angenommen, die Tür wird im Normalfall nur alle drei Monate einmal geöffnet und der Sensorknoten arbeitet zuverlässig, so dass ein Ausfall ausgeschlossen werden kann. Dann ist es nicht von Vorteil, dass der RDC besagt, dass der Knoten alle 8 Hz aufwacht, um empfangsbereit zu sein. Die hierfür benötigte Energie könnte durchaus eingespart werden, so dass sich die Lebenszeit des Sensorknotens drastisch erhöht. Der Nachteil ist natürlich, dass der Sensorknoten während einer langen Zeit nicht erreichbar ist. Wenn man nur die Zustände „offen“ und „geschlossen“ überträgt, könnte der Fall eintreten, dass aufgrund eines Ausfalls oder anderer Gründe der letzte Zustand nicht bekannt ist. Das nächste Update erfolgt allerdings nicht in nächster Zeit.

Hierfür bietet CoAP eine Lösung an: Man kann CoAP-Proxies verwenden, die die Sensorwerte cachen und somit im Namen der schlafenden Sensorknoten antworten können.

Wenn diese Idee weiterverfolgt wird, so erscheint es naheliegend, auch einen unregelmäßigen RDC zu verwenden, wie im Falle des Türkontaktsensors. Normalerweise wird in einem Sensornetz davon ausgegangen, dass alle Netzknoten den gleichen RDC besitzen. Eine vernünftige Zusammenarbeit von CoAP, dem Routing Protokoll sowie dem RDC Protokoll kann bewirken, dass verschiedene RDCs im gleichen Sensornetz zum Einsatz kommen. Diese Theorie gilt es noch zu prüfen.

V. Ausblick

Die durchgeführten exemplarischen Implementationen haben geholfen, einen Überblick über die Möglichkeiten zur Anbindung von Sensoren bzw. Aktoren unter Contiki zu entwickeln. Dabei sind allerdings verschiedene Probleme aufgetreten, die in nachfolgenden Arbeiten untersucht und gelöst werden müssen:

- Die Messungenauigkeiten des Analog-Digital-Wandlers des ATmega128RFA1 sowie das ungewöhnliche Verhalten der Interrupts 0 und 1, beides in Unterabschnitt II-A beschrieben, müssen untersucht und gelöst werden.
- Die CoAP-Schnittstelle Contikis muss dahingehend untersucht werden, wie die in Abschnitt IV angesprochenen eventbasierten und periodischen Ressourcen konkret umgesetzt werden können. Auch die Verwendung des Separate Response unter Contiki muss getestet werden.
- Die Stromsparmechanismen von Contiki [11, 12] müssen genutzt werden. Dazu sind unter anderem die Schlafmodi des ATmega128RFA1 zu implementieren und ein Konzept zu entwickeln, wie die Prozesse dem Betriebssystem mitteilen können, ob ein Schlafmodus zulässig ist oder nicht.

Anhang
Vorlesung Sensornetze

In der Vorlesung *Sensornetze* wurden Themen behandelt, die gewinnbringend in das Forschungsprojekt einfließen konnten. Folgende Themen waren für das Forschungsprojekt besonders interessant:

- Contiki-OS: Implementationsaspekte und Netzwerk-Stack
- RPL (Routing Protocol Layer)
- CoAP
- HomeMatic
- KNX

Zu diesen Themen wurden Vorträge gehalten und es wurden auch Papers geschrieben, die über Prof. Vogt mit Zustimmung der jeweiligen Authoren eingesehen werden können.

Danksagung

Die Autoren wollen Prof. Dr.-Ing. Jörg Vogt für das Anbieten und Betreuen des Forschungsprojektes *Sensornetze* sowie den anderen Teilnehmern für ihre Mitarbeit danken. Es ist ein sehr interessantes Forschungsgebiet und wir wünschen den noch verbleibenden sowie den neu hinzukommenden Teilnehmern viel Spaß bei der Fortführung dieses Forschungsprojektes. Für Fragen stehen wir gerne zur Verfügung.

Literatur

[1] A. Dunkels, T. Voigt und J. Alonso, *Connecting wireless sensor networks with the internet*, ERCIM News, Apr. 2004. url: http://www.ercim.org/publication/Ercim_News/enw57/dunkels.html.

[2] A. Dunkels und J.-P. Vasseur, *IP for Smart Objects*, IPSO Alliance White Paper #1, Sep. 2008. url: http://dunkels.com/adam/dunkels08ipso.pdf.

[3] A. Dunkels, B. Grönvall und T. Voigt, „Contiki - a lightweight and flexible operating system for tiny networked sensors“, in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004. url: http://dunkels.com/adam/dunkels04contiki.pdf.

[4] *ATmega128RFA1*, Rev. 8266C-MCU Wireless-08/11, Atmel Corporation, Aug. 2011.

[5] the Contiki project and its contributors. (2012). Contiki, The Open Source OS for the Internet of Things, url: http://www.contiki-os.org/ (besucht am 10. 02. 2013).

[6] ——, *Contiki Documentation*, Version 2.6, 2012. url: http://contiki.sourceforge.net/docs/2.6/ (besucht am 09. 02. 2013).

[7] M. Kovatsch, S. Duquennoy und A. Dunkels, „A Low-power CoAP for Contiki“, in *Proceedings of the IEEE Workshop on Internet of Things Technology and Architectures*, Valencia, Spain, Okt. 2011. url: http://dunkels.com/adam/kovatsch11low-power.pdf.

[8] A. Dunkels, O. Schmidt und T. Voigt, „Using Protothreads for Sensor Node Programming“, in *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, Juni 2005. url: http://dunkels.com/adam/dunkels05using.pdf.

[9] A. Dunkels, O. Schmidt, T. Voigt und M. Ali, „Protothreads: simplifying event-driven programming of memory-constrained embedded systems“, in *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, Nov. 2006. url: http://dunkels.com/adam/dunkels06protothreads.pdf.

[10] ——, „Simplifying memory-constrained event-driven programming with contiki's protothreads“, in *Proceedings of Real-Time in Sweden 2007*, Västerås, Sweden, Aug. 2007.

[11] H. Ritter, J. Schiller, T. Voigt, A. Dunkels und J. Alonso, „Experimental Evaluation of Lifetime Bounds for Wireless Sensor Networks“, in *Proceedings of the Second European Workshop on Sensor Networks (EWSN2005)*, Istanbul, Turkey, Jan. 2005. url: http://dunkels.com/adam/ewsn2005.pdf.

[12] A. Dunkels, F. Österlind, N. Tsiftes und Z. He, „Software-based on-line energy estimation for sensor nodes“, in *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, Juni 2007. url: http://dunkels.com/adam/dunkels07softwarebased.pdf.