Erweiterbare und bastlerfreundliche Heim-Automatisierung mit openHAB

Philipp Hertweck, Fraunhofer IOSB

Die Vernetzung unterschiedlicher Dinge nimmt immer mehr an Fahrt auf. Auch die eigene Wohnung bleibt davon nicht unberührt, sodass das „Smart Home" an Bedeutung gewinnt. Dafür gibt es bereits Lösungen verschiedener Hersteller, passende App und Cloud-Anbindung inklusive. Die Plattform openHAB liefert die Antwort, damit Geräte unterschiedlicher Anbieter interagieren und Privatsphäre-gerecht, ohne Bekanntgabe sensibler Daten, integriert werden können.

Auf openHAB können unterschiedliche Komponenten (und die damit verbundenen Protokolle) unter einer einheitlichen Benutzeroberfläche dargestellt und gesteuert werden. Neben einer Einführung in openHAB zeigt dieser Artikel Möglichkeiten, wie eigene Sensoren und Aktoren entwickelt und eingebunden werden können, sodass sich mithilfe von Micro-Controllern wie Arduino & Co. (fast) jeder Wunsch im Smart Home erfüllen lässt.

Die Heim-Automatisierung

Wäre es nicht schön, wenn sich die Steuerung von Geräten im Haus automatisieren ließe? Das Licht in der Küche nicht mehr vergessen werden kann, da es beim Verlassen des Raumes automatisch ausgeht? Beim Weggehen eine Warnung erscheint, dass das Fenster im Wohnzimmer noch offen steht? Eine Benachrichtigung kommt, wenn die Waschmaschine im Keller fertig ist? Dank vernetzter Heimgeräte lässt sich dies bereits heute umsetzen.

Unterschiedliche Hersteller bieten Systeme mit verschiedensten Sensoren und Aktoren an. Häufig ist eine App verfügbar, um die einzelnen Komponenten zu verwalten und zu steuern. Wenn die eigenen Wünsche mithilfe dieses Systems vollständig abgedeckt sind, ist dies eine einfache und komfortable Lösung. Anders sieht es aus, wenn Komponenten unterschiedlicher Hersteller kombiniert werden sollen. Bei geschlossenen Systemen ist dies kaum möglich. Abhilfe versuchen (Funk-)Standards wie ZWave, ZigBee, Bluetooth smart oder (kabelgebunden) KNX zu schaffen. Jedoch ist selbst innerhalb dieser Standards eine Interoperabilität verschiedener Hersteller nicht immer sichergestellt, obwohl diese, im Vergleich zu geschlossenen Systemen, häufig teurer sind.

Im Rahmen dieser Diskussion soll auch nicht außer Acht gelassen werden, dass einige proprietäre Systeme auf eine Cloud-Anbindung angewiesen sind; sie funktionieren nur, wenn die Dienste des Herstellers verfügbar und erreichbar sind. Die Vergangenheit hat gezeigt, dass dies nicht immer der Fall ist *(siehe „https://www.heise.de/newsticker/meldung/Serverausfall-bei-Homematic-IP-3903589.html“).* Außerdem stellt sich in diesem Zusammenhang die Frage nach dem Datenschutz: Was passiert mit den Daten aus meinem Smart Home? Wer erhält Zugriff darauf? Wie werden diese ausgewertet *(siehe „https://www.heise.de/newsticker/meldung/Datenschutzpanne-Testgeraete-von-Google-Home-Mini-hoerten-staendig-zu-3856399.html“)?* Ebenfalls sollte man darauf achten, wie lange ein Hersteller seine Cloud-Infrastruktur betreibt. Was passiert, wenn dieser den Betrieb einstellt? Sind dann alle Smart-Home-Komponenten nutzlos *(siehe „https://www.golem.de/news/revolv-google-macht-heimautomatisierung-kaputt-1604-120128.html“)?*

Allen diesen Problemen kann man mit openHAB *(siehe „http://www.openhab.org“)* begegnen, einem in Java geschriebenen Open-Source-Heimautomatisierungs-Framework, das auf dem Eclipse-Smart-Home-Projekt *(siehe „https://www.eclipse.org/smarthome“)* basiert. Es bietet eine hersteller- und protokollunabhängige Unterstützung sehr vieler proprietärer Smart-Home-Komponenten sowie gängiger Standards. Eine Abstraktions- und Integrationsebene erlaubt eine einheitliche Steuerung und das Zusammenspiel über Hersteller- und Protokollgrenzen hinweg. Ein mächtiges Regelsystem bietet die Automatisierung vieler Vorgänge. Die Hardware-Anforderungen von openHAB sind gering, sodass es problemlos auf einem Raspberry Pi ausgeführt werden kann. Dadurch lässt sich die Steuerung des Smart Homes in eigener Verantwortung betreiben und private Daten werden Dritten nicht bekanntgegeben.

Grundlegende Konzepte in openHAB

Nachfolgend werden einige Begriffe und Konzepte vorgestellt, die in openHAB eine zentrale Rolle spielen. Ein „Thing“ repräsentiert ein physikalisches oder virtuelles Objekt, beispielsweise einen Schalter für eine Lampe, einen Sensor, aber auch einen Webservice. Jede managebare Informationsquelle oder Funktion wird dadurch repräsentiert. „Channels“ bilden die unterschiedlichen Fähigkeiten eines Thing ab. Beispielsweise kann ein Sensor Temperatur und Luftfeuchtigkeit messen; für eine Lampe lassen sich Helligkeit und Lichtfarbe separat regeln. Jede dieser Eigenschaften wird durch einen Channel verfügbar gemacht. Ein „Binding“ ist ein Software-Adapter, der Things in openHAB integriert. Innerhalb eines solchen Add-ons findet die Kommunikation mit Things über unterschiedlichste Protokolle und Technologien statt. Es bietet somit eine Abstraktionsschicht der Smart-Home-Komponenten. Die Fähigkeiten, die in openHAB (durch die GUI oder in Regeln) verwendet werden können, sind durch „Items“ definiert. Jedes Item hat einen Zustand (die Temperatur, die ein Sensor gemessen hat, oder die Angabe, ob ein Schalter an oder aus ist) und kann gegebenenfalls Kommandos empfangen (Ein-/Aus-Schalten, Dimmen etc.). Ein „Link“ verbindet Items und Channels.

Auf dem Weg zum eigenen Smart Home

Nachdem openHAB installiert ist, sind einige Schritte erforderlich, um mit openHAB ein eigenes Smart Home aufzubauen. Anleitungen für verschiedene Systeme finden sich in der sehr guten openHAB-Dokumentation *(siehe „https://docs.openhab.org“).*

Die ersten Schritte können mithilfe des Paper UI ausgeführt werden, einer intuitiven Web-Oberfläche, die nach der Installation zur Verfügung steht. Zu Beginn ist das notwendige Binding für die vorhandenen Smart-Home-Komponenten zu installieren. Davon sind bereits mehr als 150 verfügbar, die die gängigsten auf dem Markt erhältlichen Systeme abdecken. Darüber hinaus sind Bindings verfügbar, mit denen Komponenten generisch integriert werden können. Das Exec-Binding erlaubt die Ausführung beliebiger Programme; mit dem HTTP-Binding lassen sich Komponenten integrieren, die per HTTP erreichbar sind. Das MQTT-Binding ermöglicht die Einbindung von Geräten, die über das gleichnamige Nachrichtenprotokoll angesprochen werden, das nachfolgend zum Einsatz kommt, um selbst gebaute Komponenten zu integrieren.

Sobald das passende Binding installiert ist, sind die Things zu konfigurieren. Werden proprietäre Systeme eingebunden, ist eine Bridge (IP-Gateway für Heimautomatisierungssysteme) häufig die Startkomponente. Sie ermöglicht die Integration weiterer Things, die per Funk mit dem Gateway kommunizieren. Am einfachsten funktioniert dies, wenn die Things per Autodiscovery automatisch gefunden und über Paper UI direkt angelegt werden können *(siehe Abbildung 1)*.

Vor der weiteren Einrichtung zunächst ein Blick auf die openHAB-Konfiguration. Eine komfortable Möglichkeit ist das bereits beschriebene Paper UI. Die Installation von Add-ons, das Anlegen erkannter Things mittels Autodiscovery sowie das Zuordnen von Channels zu Items ist dort möglich. openHAB lässt sich auch über Konfigurationsdateien einrichten. Diese Möglichkeit bietet sich vor allem an für Items, Sitemaps (konfigurierbare Benutzeroberfläche) und Rules (Regeln für die Automatisierung). In der Praxis hat sich eine Kombination beider Möglichkeiten bewährt. Vorlagen für Item- und Sitemap-Konfigurationen lassen sich mithilfe eines Generators (Home Builder, *siehe „https://docs.openhab.org/configuration/homebuilder.html“)* erstellen.

Anschließend sollen auf dem Weg zum eigenen Smart Home Items angelegt werden. Diese beschreiben die funktionale Sicht, also sämtliche durch UIs und Regeln steuerbaren Objekte. Um eine konsistente Sicht auf unterschiedliche Arten von Geräten zu erhalten, existieren verschiedene Item-Typen, beispielsweise „Contact“ (Zustand eines Tür-/Fensterkontakts Open/Close), „Dimmer“ (Prozentwert für dimmbare Items On/Off, Increase/Decrease, Percent), „Number“ (Zahlenwert) oder „Switch“ (Schalter On/Off). Die vollständige Liste steht in der openHAB-Dokumentation. *Listing 1* zeigt das Format einer Item-Definition.

Eine weitere praktische Möglichkeit zur Strukturierung ist die Gruppenbildung von Items. Sie ermöglicht die Anzeige (basierend auf einer Aggregats-Funktion) und Steuerung aller Gruppen-Items. Zusätzlich können Gruppen wiederum Gruppen zugeordnet sein, sodass eine hierarchische Gliederung möglich ist. *Listing 2* zeigt eine beispielhafte Items-Definition. Eine Sitemap definiert die in *Abbildung 2* gezeigte Benutzeroberfläche, auf der Items und Gruppen angeordnet werden.

„Rules“ sind ein mächtiges Werkzeug, um Abläufe zu automatisieren. Sie bestehen immer aus einem Namen und einem Skript-Block. *Listing 3* zeigt ein Beispiel, wie täglich um 6:30 Uhr das Item „Wake-Up_Light“ einschaltet wird.

```
itemtype itemname "labeltext [stateformat]" <iconname> (group1, group2, ...) ["tag1", "tag2", ...] {bindingconfig}
```

Listing 1

```
Group    Home                   "Daheim"            <house>
Group    Office                 "Büro"              <office>          (Home)
Group    LivingDining           "Wohn-Esszimmer"    <sofa>            (Home)

Switch   Office_Light           "Licht"             <light>           (Office, gLight)
Number   Office_Humidity        "Luftfeuchtigkeit"  <humidity>        (Office, gHumidity)
Number   Office_Temperature     "Temperatur"        <temperature>     (Office, gTemperature)
Switch   LivingDining_Light     "Licht"             <light>           (LivingDining, gLight)
Number   LivingDining_Humidity  "Luftfeuchtigkeit"  <humidity>        (LivingDining, gHumidity)
Number   LivingDining_Temperature "Temperatur"      <temperature>     (LivingDining, gTemperature)

Group:Switch:OR(ON, OFF) gLight         "Licht"              <light>          (Home)
Group:Number:AVG         gHumidity      "Luftfeuchtigkeit"   <humidity>       (Home)
Group:Number:AVG         gTemperature   "Temperatur"         <temperature>    (Home)
```

Listing 2

Als „Trigger" können ein Item-Event (Zustand eines Items ändert sich), die Uhrzeit, System- oder Thing-Events eingesetzt werden. Eine Java-ähnliche DSL, um den Zustand von Items auszulesen und um Commands an Items zu verschicken, ermöglicht somit eine mächtige Möglichkeit, das Smart Home zu automatisieren.

Selbstbau-Komponenten

Wie bereits angedeutet, soll dieser Artikel auch aufzeigen, wie man eigene Komponenten bauen und in das Gesamtsystem integrieren kann. Dafür kommt mit dem ESP8266 ein Low-Power-32-Bit-Microcontroller von espressif zum Einsatz, der für wenige Euro erhältlich ist. Neben einem integrierten WLAN-Chip ist dieser Mikrocontroller kompatibel zu Arduino. Dadurch lassen sich sämtlichen Tools wie beispielsweise die IDE, aber auch Bibliotheken aus dem Arduino-Umfeld, nutzen. Für Einsteiger ist es auch sehr hilfreich, dass viel Arduino-Hardware (Achtung, der ESP8266 arbeitet im Gegensatz zum Arduino mit 3,3 V statt 5 V) und Anleitungen mit wenig Aufwand angepasst und umgesetzt werden können.

Nachdem die Hard- und Softwareseite geklärt sind, muss im nächsten Schritt eine Integration in openHAB stattfinden. Auf den ersten Blick gibt es mehrere Möglichkeiten: Nutzung der openHAB-REST-Schnittstelle, eigenes Binding mit eigenem Protokoll implementieren oder Nutzen eines MQTT-Message-Bus. In diesem Artikel ist Letzteres umgesetzt, weil dadurch eine lose Kopplung von Komponenten und openHAB möglich ist. Durch die freie Wahl der Topics auf dem Message-Bus kann man schnell den Überblick verlieren. Um dies zu verhindern, hat Marvin Roger mit Homie *(siehe „https://github.com/marvinroger/homie")* eine leichtgewichtige MQTT-Konvention für das Internet der Dinge geschaffen. Dabei werden die Topics „homie/[deviceID]/$[deviceInfo]" und „homie/[deviceID]/[node]/[nodeProperty]" eingesetzt.

Stellt man die Begrifflichkeiten in Zusammenhang mit den Konzepten von openHAB, so entspricht ein Device in Homie einem Thing und jeder Node eines Device entspricht einem Channel. Für den ESP8266-Microcontroller existiert bereits eine Implementierung dieser Konvention *(siehe „https://github.com/marvinroger/homie-esp8266").* Diese übernimmt die vollständige WLAN- und MQTT-

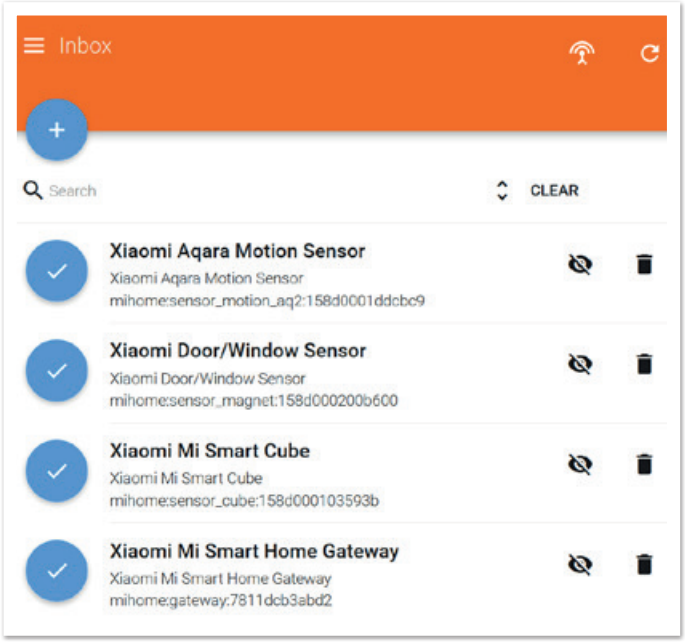

Abbildung 1: Per Autodiscovery erkannte Things

```
rule "WakeUp Light"
when
  Time cron "0 30 6 ? * *"
then
  sendCommand(WakeUp_Light, ON)
end
```

Listing 3

Verbindungsverwaltung (etwa erneutes Verbinden nach Abbruch) sowie das Bereitstellen von Meta-Informationen per MQTT. Ebenfalls müssen die WLAN- und MQTT-Konfiguration nicht im Quelltext hart codiert, sondern können per App über einen automatisch aufgespannten Access-Point eingestellt werden, wie man es auch von kommerziellen Produkten her kennt. Somit kann man sich bei der Programmierung auf die eigentliche Anbindung der Sensoren/Aktoren konzentrieren. Zuletzt muss in openHAB noch ein Item kon-

```
Number Living_Temperature "Temperatur"<temperature> {mqtt="<[local:homie/a020a616d1a3/temperature/
temperature:state:default]"}
```

Listing 4

figuriert werden, das die Temperatur vom Message-Bus liest und bereitstellt *(siehe Listing 4)*.

Sicherheit im Smart Home

Aufgrund des tiefen Eingriffs eines Smart Homes in den persönlichen Lebensbereich darf das Thema „Sicherheit" nicht außer Acht gelassen werden. Zu Beginn dieser Diskussion ist zu erwähnen, dass openHAB selbst keine Authentifizierung und keinen Zugriffsschutz bietet. Dadurch darf die openHAB-Instanz keinesfalls aus dem Internet erreichbar sein.

Wünscht man dennoch eine Steuerung/Überwachung aus der Ferne (beispielsweise mit der openHAB-App), gibt es mehrere Möglichkeiten der Absicherung. Zum einen kann mittels VPN eine gesicherte Verbindung ins Heimnetz aufgebaut werden, sodass ein Zugriff von außerhalb nicht notwendig wird. Alternativ existiert mit dem myopenHAB-Cloud-Service *(siehe „https://myopenhab.org")* ein Online-Dienst, der bestimmte Items nach Eingabe von Benutzername und Passwort erreichbar macht.

Es ist ebenfalls möglich, einen Reverse-Proxy vor openHAB zu konfigurieren, der die Authentifizierung und Autorisierung regelt. Dieser kann zusätzlich durch den Einsatz von SSL sicherstellen, dass der Zugriff vertraulich stattfindet.

An dieser Stelle ist jedoch jede einzelne Komponente des Smart Homes zu betrachten: Ist diese hinsichtlich Authentizität und Vertraulichkeit sicher integriert? Diese Frage ist nicht immer einfach zu beantworten, da oft proprietäre Komponenten und Protokolle zum Einsatz kommen. Wie sieht es aber mit den selbst gebauten Komponenten aus? Wurde die Standard-Einstellung des MQTT-Message-Brokers angepasst und dieser abgesichert?

Auch die Verfügbarkeit eines Smart Homes ist wichtig. Es wäre sehr ärgerlich, wenn das Licht nicht mehr eingeschaltet werden kann, weil das Netzwerk streikt. openHAB selbst läuft sehr stabil, auch auf einem Raspberry Pi. Dennoch kann ein Ausfall nicht ausgeschlossen werden. Daher ist beim Kauf der Komponenten darauf zu achten, dass die Basis-Funktionalität ohne openHAB realisiert werden kann und lediglich Komfort-Funktionen darauf angewiesen sind. Dadurch ist sichergestellt, dass man bei Problemen nicht zwingend im Dunklen oder Kalten sitzt. Ebenfalls sollten kritische Komponenten, wie beispielsweise eine Alarmanlage, nicht auf openHAB angewiesen sein. Hier sollte man auf zertifizierte Hardware-Systeme setzen, die durch optionale Funktionen (wie Benachrichtigung) ergänzt werden.

Weitere Funktionen

Bis hierhin wurden lediglich die grundlegenden Funktionen von openHAB vorgestellt. Durch dessen modularen Aufbau besteht eine Vielzahl an Erweiterungen. Beispielsweise existiert mit HABPanel ein leichtgewichtiges und konfigurierbares UI, das problemlos für Wand-Displays eingesetzt werden kann. Ebenfalls sind Benachrichtigungen, etwa per E-Mail oder Smartphone-Push-Meldung, beim Eintritt eines bestimmten Ereignisses möglich.

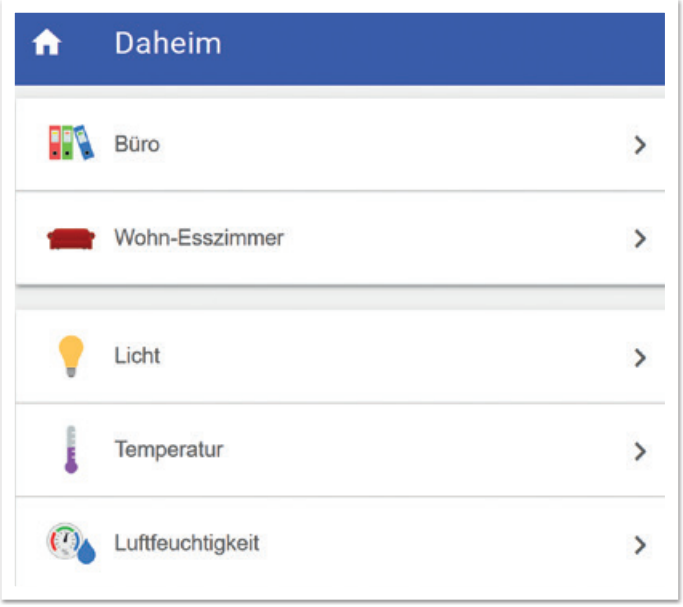

Abbildung 2: Konfigurierbare Benutzeroberfläche

Schnittstellen für Sprach-Ein- und -Ausgabe, wie sie beispielsweise durch Alexa und Google Home bekannt sind, sind ebenfalls vorhanden. Sollten selbst diese Funktionen die eigenen Wünsche nicht vollständig abdecken, gibt es für openHAB selbst ein Alexa Skill sowie eine Google-Home-Anbindung. Eine Integration mit IFTTT ist ebenfalls problemlos möglich. Dank einer großen Community mit einem aktiven Forum existieren sehr viele Projekte, Ideen und fertige Add-ons.

Philipp Hertweck
philipp.hertweck@iosb.fraunhofer.de

Nach dem Abschluss seines Informatik-Studiums am Karlsruher Institut für Technologie arbeitet Philipp Hertweck als wissenschaftlicher Mitarbeiter am Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung IOSB in Karlsruhe. Dort beschäftigt er sich unter anderem mit Software-Architekturen sowie der semantischen Modellierung, insbesondere im Zusammenhang mit Entscheidungsunterstützungssystemen.