Heimautomatisierung für Geeks

openHAB auf dem Raspberry Pi

Phillip Ghadir, Thomas Eichstädt-Engelen

Nachdem openHAB auf der diesjährigen JavaOne einen der begehrten Duke's Choice Awards gewonnen hat, möchten wir die Gelegenheit nutzen, diese quelloffene Lösung für die Do-It-Yourself-Heimautomatisierung vorzustellen. Das Framework openHAB sollte auf praktisch jedem Rechner funktionieren, der eine JVM und OSGI-Runtime ausführen kann. Wir stellen das Open-Source-System im Folgenden an einem kleinen Beispiel vor.

Abb. 1: Smart Clients für openHAB gibt es für die gängigen Geräte und modernen Browser

Einsatzzweck

openHAB steht für „open Home Automatization Bus" und ist ein quelloffenes Framework zur Steuerung und Verwaltung von Geräten. Über openHAB können Sensoren wie Helligkeits- und Feuchtigkeitssensoren, Kontaktschalter oder Temperaturfühler ausgelesen werden, um auf Basis der gelieferten Informationen entsprechend zu reagieren. Es wird dunkel? – Dann schalten wir das Licht ein. Es regnet? – Die Markise wird automatisch reingefahren.

Mit openHAB können wir eigene Steuerregeln programmieren, um Sensoren abzufragen, entsprechend Aktoren zu steuern und so verschiedene Geräte intelligent zu integrieren. Üblicherweise adressiert man mit Heimautomatisierung drei Ziele: den Komfort zu erhöhen, die Sicherheit zu verbessern und zu guter Letzt Energie zu sparen. openHAB befähigt uns Entwickler dazu.

openHAB: Designer und Runtime

openHAB besteht im Wesentlichen aus den Teilsystemen:

- der openHAB-Runtime, einer Laufzeitumgebung zur Steuerung und Integration der Komponenten im smarten Heim,
- dem openHAB-Designer, einer auf Eclipse RCP basierenden Desktop-Anwendung zur Konfiguration der Laufzeitumgebung, und
- den openHAB-Smart-Clients, mit denen eine openHAB-Runtime gesteuert werden kann.

Die openHAB-Runtime basiert unter anderem auf OSGI, dem Eclipse Modelling Framework (EMF) und dem Framework Atmosphere. Letzteres wird dazu verwendet, Daten aus der Runtime asynchron an Webclients weiterzureichen. So können zum Beispiel Temperaturkurven oder Stellungen von Schaltern jederzeit mit dem ins Smartphone integrierten Browser überwacht werden. Jede Änderung wird dabei an den mobilen Browser gesendet, ohne dass der Anwender etwas Spezielles tun muss.

openHAB-Clients

Die openHAB-Runtime erlaubt die Überwachung und Steuerung der integrierten Geräte und Dienste über verschiedene Clients. Menüs, Aktionen und Informationen über die openHAB-Runtime präsentieren die Clients generisch auf Basis einer hierarchischen Beschreibung der Benutzeroberfläche – genannt Sitemap.

Eine Sitemap besteht aus sogenannten Frames, die selbst wiederum Frames, Schalter, Zähler, Graphen, Schieberegler, Auswahlfelder oder Texte enthalten können. Darüber hinaus können auch Multimedia-Inhalte wie Videos oder Webseiten dargestellt werden. Abbildung 1 zeigt beispielsweise die Demo-Sitemap in einem Android-Client.

Abstraktion von Geräten

Wer über die Domäne Heimautomatisierung nachdenkt, wird vermutlich schnell auf Domänenklassen wie „Gerät" mit den Spezialisierungen „Sensor" und „Aktor" kommen. Diese sehr naheliegenden Abstraktionen sind allerdings nicht änderungsresistent genug, um langfristig beliebige Anpassbarkeit für heute noch ungeahnte Anwendungsfälle zu erlauben.

Die Entwickler entschlossen sich daher für eine noch größere Abstraktion. Anstatt in openHAB ein Gerät zu deklarieren, deklariert man die Dinge, die ein Gerät ausmachen. Diese werden in openHAB Item genannt und repräsentieren Werte, die man abfragen oder setzen kann, wie Timer, Regler, Zähler, Schalter und einiges mehr.

openHAB-Konfiguration

Um also Geräte automatisch miteinander zu integrieren oder mit einem openHAB-Client fernsteuern zu können, müssen die deklarierten Items einerseits noch an eine Technologie oder ein Gerät gebunden und andererseits in der Sitemap referenziert werden, damit deklarierte Schalter, Werte oder auch Graphen dargestellt werden können.

Diese Anbindung ist aus zwei Gründen erwähnenswert: UI-Elemente, Items und Regeln sind dadurch entkoppelt von der Hardware. Auch ohne konkrete Hardware kann man mit openHAB bereits Regeln integrieren und testen. In unserem Fall könnten wir in openHAB schon die Regeln und die App vorbereiten, auch wenn noch keine realen Sensoren Werte liefern oder Aktoren Aktionen ausführen.

Zudem ist die Integration vollständig losgelöst von der konkreten technischen Bindung. Die Integrationslogik kann also unverändert genutzt werden, selbst wenn Geräte ausgetauscht und mit ganz anderen Protokollen angesprochen werden müssen.

Eigene Bindings schreiben

Zur Anbindung eines bisher unbekannten Gerätes oder neuen Protokolls muss ein neues Binding erstellt werden. Dabei wird ein Binding in der Regel durch genau ein OSGi-Bundle implementiert.

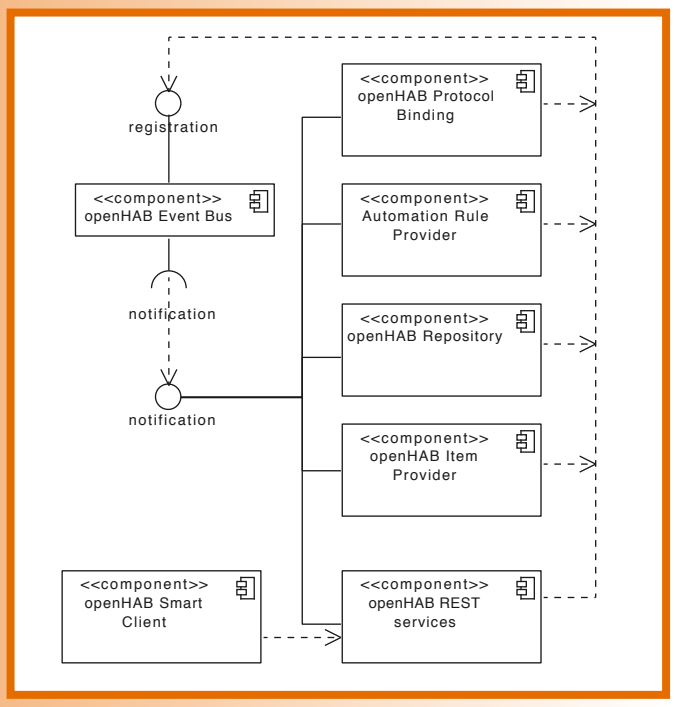

Abb. 2: openHAB-Komponenten hängen am Event-Bus

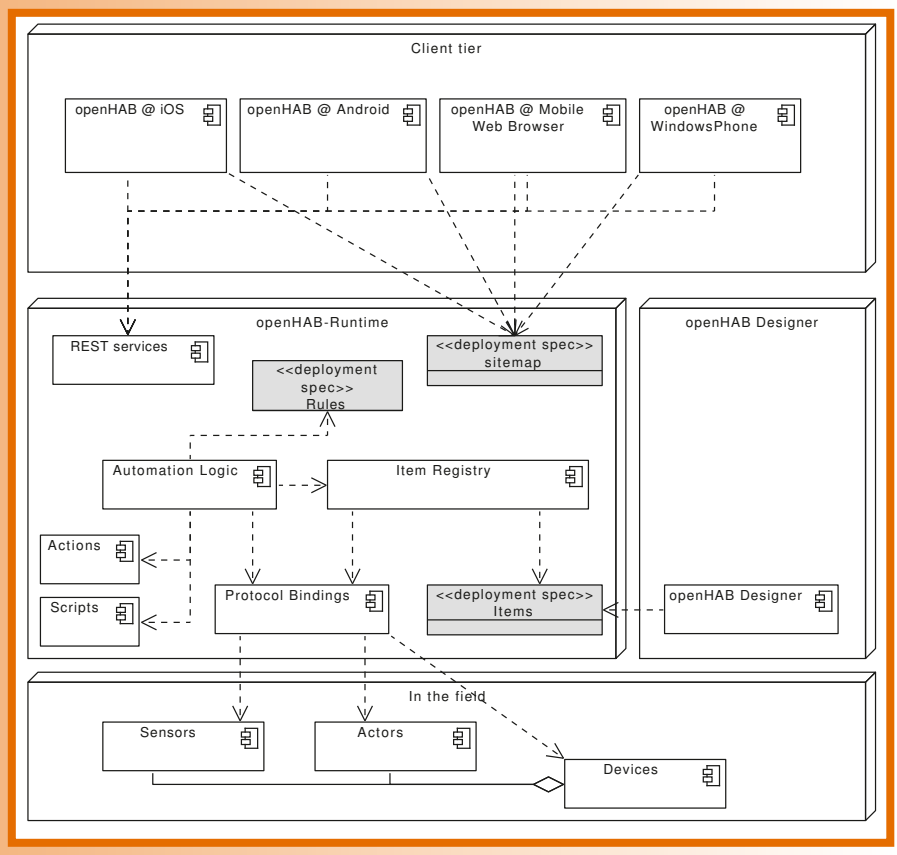

Abb. 3: Struktureller Überblick über openHAB

Das Projekt stellt ein Maven archetype zur Verfügung, mit Hilfe dessen ein lauffähiges Binding-Skeleton ohne Funktionalität erstellt werden kann. Die einzelnen Schritte sind im Entwickler-Wiki beschrieben [binding]

openHAB Architektur

Zentrales Bindeglied zwischen den einzelnen openHAB-Komponenten bildet der openHAB Event Bus, über den sämtliche Bestandteile einer Runtime miteinander kommunizieren. Dies gilt sowohl für protokoll- bzw. technologiespezifische Anbindungen als auch für die Anbindung von Clients, die im Allgemeinen auf den REST-Services von openHAB aufsetzen.

Die zugrunde liegende OSGI-Runtime sorgt für eine Entkopplung und Isolation von Bindings. Jede technologische Bindung wird in openHAB in Form eines separaten OSGI-Bundles realisiert. Der Event-Bus integriert die verschiedenen Bausteine der openHAB-Runtime wie in Abbildung 2 dargestellt.

Abbildung 3 bietet einen Gesamtüberblick über openHAB. Es gibt verschiedene Clients. Die Runtime integriert die in der Abbildung dargestellten Komponenten über den Event-Bus. Der openHAB-Designer schreibt die Konfigurationen, die von der Runtime ausgewertet werden: Items, Rules und Sitemap.

openHABs eigene Regelsprache

Dank EMF, Xbase und Xtext können Automasierungsregeln für eine openHAB-Installation in sogenannten Rules-Dateien im openHAB-Designer mit Code-Vervollständigung und Syntax-Highlighting bearbeitet werden.

Regeldateien müssen im Verzeichnis `$(openhab.home)/configuration/rules` mit der Dateiendung `.rules` abgelegt werden, und bestehen aus den Bereichen Imports, Variablendeklarationen und Regeln.

Imports sehen aus wie gewöhnliche Java-import-Anweisungen. Variablendeklarationen sehen aus wie bei Xtend: Veränderliche Variablen werden mit dem Schlüsselwort `var` und unveränderliche mit `val` vereinbart. Auf die explizite Angabe eines Variablentyps kann verzichtet werden, wenn die Variable direkt mit der Deklaration initialisiert wird und daraus der Typ abgeleitet werden kann. Regeln folgen der Form:

```
rule "<REGELNAME>"
when
  <TRIGGER CONDITION>
then
  <EXECUTION_BLOCK>
end
```

openHAB unterstützt drei Arten von Auslösern (oben TRIGGER CONDITION genannt) für Regeln:

- ereignisbasierte reagieren auf Ereignisse, die über den openHAB-Event-Bus empfangen werden,
- zeitbasierte werden zu definierten Zeiten (oder auch in Zeitintervallen) ausgelöst,

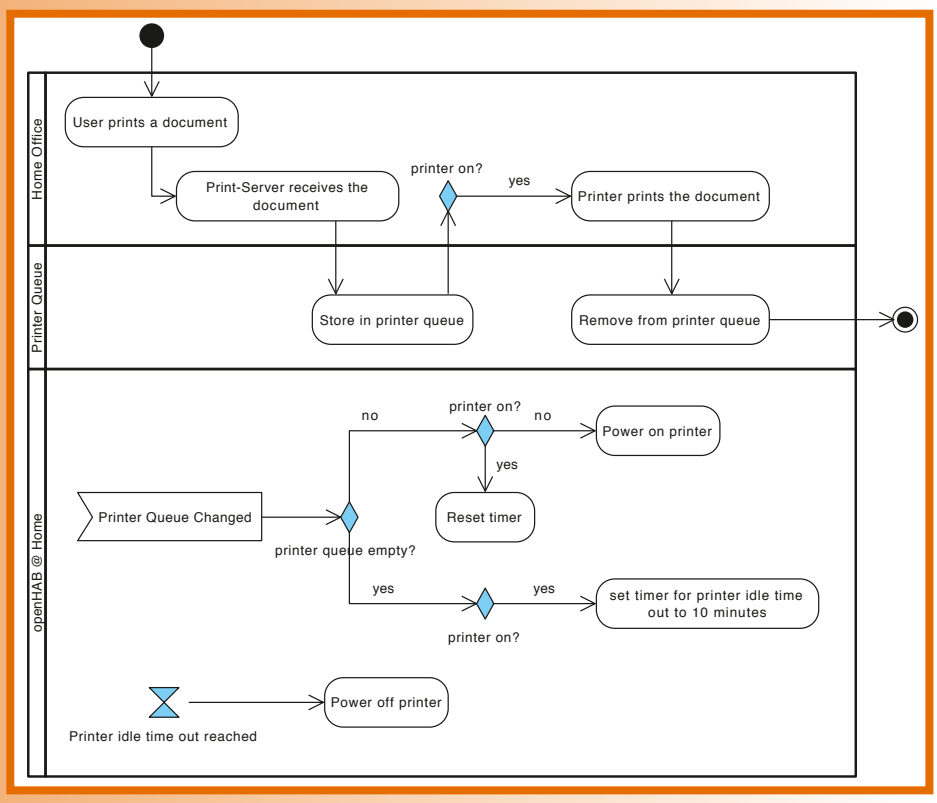

Abb. 4: Plan zur Automatisierung des Drucker-An- und -Abschaltens

▼ systembasierte reagieren wahlweise auf den Systemstart oder das Herunterfahren des Systems.

Regeln sind nicht auf einfache Auslöser beschränkt: Auslöser können bei Bedarf durch logische Oder-Verknüpfungen kombiniert werden.

Anwendungsfall Drucken

Selbst wenn man nicht die eigene Wohnung oder das eigene Haus intelligenter machen möchte, können manche Bindings von openHAB das Leben erleichtern und eventuell sogar Energie sparen.

Angenommen unseren privaten Drucker nutzen wir nur 10 bis 12 mal im Jahr. Die allermeiste Zeit wird der Drucker also nicht gebraucht. Er steht ausgeschaltet im Keller. Ein Dokument zu Hause auszudrucken, ist also lästig: Zum Drucker zu gehen, den Drucker einzuschalten, zurück zum Rechner zu gehen, den Druck anzustoßen. Diese Aufgabe könnte genauso gut openHAB mit dem CUPS-Binding für uns übernehmen und uns das Hin-und-her ersparen.

Bevor wir uns anschauen, wie wir openHAB dazu bewegen könnten, den Drucker ein- oder auszuschalten, planen wir erst einmal, was wir gern hätten. Falls mehrere Dokumente nacheinander gedruckt werden sollten, wäre es schlecht, den Drucker nach einem Druck einfach auszuschalten. Daher setzen wir einfach einen Timeout, nach dessen Ablauf der Drucker ausgeschaltet wird. Das Aktivitätsdiagramm in Abbildung 4 veranschaulicht den Plan.

openHAB und der Drucker

Der Drucker muss in openHAB deklariert werden. Für unseren Anwendungsfall sind zwei Items von Interesse: die Printer-Queue und der Schalter, über den wir den Drucker ein- und ausschalten können. Beides deklarieren wir in einer entsprechenden items-Datei unter `$(openhab.home)/configuration/items`, wie in Listing 1 dargestellt.

```
Number Print_Jobs_Queued "queued jobs"
  (FF_Office)
 { cups=" 1020D#NOT_COMPLETED" }
Switch Printer "1020D"
```

Listing 1: Fragment der Items-Deklaration für das An- und Abschalten des Druckers

Darüber hinaus müssen wir die Regeln hinterlegen, nach denen openHAB agieren soll, wenn wir einen Druck anstoßen. Dies geschieht in der Regeldatei (s. Listing 2). Die in Listing 2 dargestellte Regel verwendet eine interne Variable `printerTimer`. Diese kann zurückgesetzt werden, wenn ein neuer Druckauftrag eingeht, während bereits ein Timer gesetzt ist. So kann das Ausschalten verhindert werden, solange noch Druckaufträge vorliegen.

```
import org.openhab.model.script.actions.Timer

var Timer printerTimer = null

rule "CUPS-Printer Queue"
when
  Item Print_Jobs_Queued changed
then
  if (Print_Jobs_Queued.state>0) {
    if (printerTimer!=null) {
      printerTimer.cancel
      printerTimer=null
    }
    if (Printer.state==OFF)
      sendCommand(Printer,ON)
  }
  else if (Printer.state==ON) {
    printerTimer = createTimer(now.plusMinutes(10)) [|
      sendCommand(Printer,OFF)
    ]
  }
end
```

Listing 2: Die Regel für openHAB zur Steuerung des Druckers auf Basis der Printer-Queue

Deployment

Zur Realisierung des oben beschriebenen Anwendungsfalls kann eine openHAB-Installation auf einem dedizierten Server durchgeführt werden (s. Kasten „openHAB auf dem Raspberry Pi“). Darüber hinaus wäre es aber genauso gut denkbar, openHAB auf dem Desktop-PC oder Notebook als reines Automatisierungswerkzeug zu installieren.

Aufgrund der Plattformunabhängigkeit könnte man darüber hinaus mit der lokalen Installation beginnen und sie bei Bedarf später leicht auf ein anderes System umziehen. Dazu muss lediglich der Installationsordner gepackt, auf das gewünschte

openHAB auf dem Raspberry Pi

Zum Betrieb der openHAB-Runtime auf einem aktuellen Raspberry Pi – Modell b mit 512 MB RAM – ist eine installierte Java-Laufzeitumgebung erforderlich. Dabei ist der quelloffenen OpenJDK-JVM aus Performancegründen eindeutig Oracles hard-float Java 7 JDK vorzuziehen [performance]. Erfreulicherweise bekräftigte Oracle auf der JavaOne sein Commitment in den Embedded-Bereich mit der Ankündigung, dass zukünftig alle Raspbian Images mit dem Oracle JDK 7 ausgeliefert werden [raspbian]. Alternativ kann das JDK als Paket aus dem Raspberry-Repository nachinstalliert werden.

Nachdem die Laufzeitumgebung zur Verfügung steht, sind die folgenden Schritte zur Installation der openHAB-Runtime (zum Zeitpunkt des Schreibens dieses Artikels ist die Version 1.3.1 aktuell) erforderlich:

- Download von openhab-runtime-x.y.z.zip
- an den gewünschten Speicherort entpacken
- Download von openhab-addons-x.y.z.zip
- an einen beliebigen Speicherort entpacken
- Kopieren von org.openhab.binding.cups-x.y.z.jar in das Verzeichnis $(openhab.home)/addons
- Anlegen der Hauptkonfigurationsdatei openhab.cfg im Verzeichnis $(openhab.home)/
- Konfiguration der CUPS-Server-Adresse über den Parameter „cups:host“
- Erstellen der *.items-, *.sitemap- und *.rules-Dateien mit Hilfe des Designers
- Starten der openHAB-Runtime über das Startskript start.sh

Sobald openHAB gestartet ist, kann man sich nun das Ergebnis seiner Bemühungen ansehen. Am einfachsten geht das mit den nativen Clients, weil sie die Adresse des Servers per autodiscovery ermitteln. Mit einem Webbrowser erreicht man über die URL `https://<host>:8443/openhab.app?sitemap=<sitemapName>` den openHAB-Webclient.

Zielsystem kopiert und dort ausgepackt werden. Die Runtime kann dort (eine Java-Laufzeitumgebung vorausgesetzt) ohne weitere Änderungen sofort gestartet werden. Eine Vorgehensweise, die gerade bei schwächeren Embedded-Systemen häufig Anwendung findet.

Eclipse-SmartHome-Projekt

Um die Weiterentwicklung des Projektes auch in Zukunft sicherzustellen, haben sich die Autoren des Projektes entschieden, große Teile des Kerns an die Eclipse Foundation zu spenden. Dafür wurde das Eclipse-Projekt SmartHome [smarthome] vorgeschlagen, das voraussichtlich Anfang 2014 sein erstes Release vorstellen wird. Weitere Details zur Entscheidung sind über die Google-Group zu erfahren [smarthome-discussion].

Zusammenfassung

openHAB ist ein System zur Automatisierung und regelbasierten Integration verschiedenster Komponenten, wie Temperaturfühler, Schalter oder Motoren. Die openHAB-Runtime läuft praktisch überall, wo eine moderne JVM funktioniert. Ein Raspberry Pi reicht als Steuerzentrale bereits aus. Die Konfiguration erfolgt über spezifische Textkonfigurationen in dafür vorgesehenen Verzeichnissen. Wer statt eines einfachen Texteditors gern Code-Vervollständigung und Syntax-Highlighting hat, erhält mit dem openHAB-Designer ein geeignetes Konfigurationswerkzeug, das nicht nur das Konfigurieren erlaubt, sondern zudem auch die Preview der Steuerungs-App bietet.

Für openHAB gibt es Steuerungs-Apps für die modernen mobilen Plattformen Android, iOS und Windows sowie eine generische Webapplikation, die auf allen modernen Browsern funktionieren sollte.

Über eine Vielzahl an Bindings ist es möglich, verschiedenste Dinge über HTTP, UDP, Bluetooth uvm. anzusprechen. Es gibt bereits Anbindungen für Fritz!Box, CUPS (einem freien Print-Server und Framework) und etwa vierzig Bindings mehr – darunter auch viele professionelle Bus-Systeme.

Der Slogan „Heimautomatisierung für Geeks“ ist Programm und brachte openHAB dieses Jahr den Duke’s Choice Award [duke] ein.

Links

[atmosphere] `https://github.com/Atmosphere/atmosphere`
[binding] `http://code.google.com/p/openhab/wiki/HowToImplementABinding`
[bindings] `https://code.google.com/p/openhab/wiki/Bindings`
[cups] `http://www.cups4j.org/`
[duke] `https://www.java.net/dukeschoice`
[eclipseRCP] `http://wiki.eclipse.org/index.php/Rich_Client_Platform`
[emf] `http://www.eclipse.org/modeling/emf/`
[openhab] `http://www.openhab.org/`, `http://code.google.com/p/openhab/`, `https://github.com/openhab/openhab`
[performance] `https://groups.google.com/forum/#!searchin/openhab/jdk/openhab/BGpzkg4ZOCo/qZG-q8bq084J`
[raspbian] `http://www.raspberrypi.org/archives/4920`
[smarthome] `http://eclipse.org/proposals/technology.smarthphone`
[smarthome-discussion] `https://groups.google.com/forum/#!searchin/-openhab/eclipse$20smart$20home/openhab/_Js71WUI1os/rYCiPL2cXMkJ`
[startscript] `http://code.google.com/p/openhab-samples/wiki/Tricks`
[xtend] `http://www.eclipse.org/xtend/documentation.html`

Phillip Ghadir baut am liebsten tragfähige, langlebige Softwaresysteme. Er ist Mitglied der Geschäftsleitung bei innoQ und hat sich früh auf Architekturen für verteilte, unternehmenskritische Systeme spezialisiert. Darüber hinaus ist er Mitbegründer und aktives Mitglied des iSAQB, des International Software Architecture Qualification Board.
E-Mail: phillip.ghadir@innoq.com.

Thomas Eichstädt-Engelen arbeitet als Senior Consultant bei innoQ, wo er sich mit der Entwicklung und dem Training im Umfeld von Eclipse, RCP und OSGi beschäftigt. Seine Freizeit bestimmt die Weiterentwicklung der Integrationsplattform openHAB, wo er seit 2010 aktiver Committer und Project-Owner ist.
E-Mail: Thomas.Eichstaedt-Engelen@innoq.com