Plus DVD

Eclipse Orion: Das neue IDE-Paradigma » 38

3.11

Deutschland € 12,90
Österreich € 14,80, Schweiz sFr 25,50

S&S

eclipse MAGAZIN

www.eclipse-magazin.de

DVD-INHALT:

JAHRESARCHIV 2009/2010

Über 150 Artikel auf DVD!

Android360-Spezialausgabe

Das komplette Magazin zur Android-Entwicklung auf DVD!

KEYNOTE VON MIK KERSTEN:

Bringing Open Source Collaboration to the Enterprise

WEITERE INHALTE:

- Android SDK for Eclipse
- Reuseware
- EclEmma 1.5.1
- JaCoCo 0.5.0
- Eclipse Orion 0.2M5

u.v.m.

4 196603 212905 03

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

ANDROID

Android-Entwicklung mit Eclipse

OSGi und Vaadin » 8

Dynamische Webapplikationen bauen

Scala » 23

Modern und funktional

TMF meets GMF » 74

Kombination textueller und grafischer Editoren

e4 Tutorial » 31

Services und UI in Einklang gebracht

Home, Smart Home » 58

Heimautomatisierung mit OSGi

EclEmma und JaCoCo » 14

Codeabdeckungsanalyse für jeden

Eclipse Scout » 64

Das neue Enterprise Framework

Heimautomatisierung mit OSGi

Home, Smart Home

Die IT hat fast alle Lebensbereiche erobert, nur in der Hauselektrik fristet sie noch ein Schattendasein. Mit zunehmender Anzahl elektronischer Geräte wird aber auch hier die Vernetzung immer interessanter und eröffnet zahlreiche neue Möglichkeiten. Der Open Home Automation Bus (openHAB) [1] nutzt OSGi, um verschiedene Systeme miteinander zu verbinden und unter einer einheitlichen Oberfläche bedienen zu lassen.

von Kai Kreuzer

Das Smart Home ist schon seit mehr als einem Jahrzehnt ein viel diskutiertes Thema, verspricht es doch neben Komfortgewinn auch eine höhere Sicherheit sowie Energieeinsparungsmöglichkeiten. Dennoch konnte es sich bisher in Privathaushalten noch nicht durchsetzen. Das liegt einerseits sicher an den recht hohen Einstiegskosten und dem notwendigen Installationsaufwand, andererseits aber auch an der starken Marktfragmentierung. So steht der willige Käufer vor einer unübersichtlichen Zahl von Optionen: Soll er ein System mit Bus-Kabel (z. B. KNX [2]) verwenden oder zwecks einfacherer Nachrüstung doch lieber auf Funk (z. B. EnOcean [3]) oder gar Powerline (z. B. DigitalSTROM [4]) setzen? Welche Taster und Aktoren bekommt er von welchem Hersteller für welches System? Welche Geräte gibt es für die Steuerung und Programmierung? Wie greift man übers Internet darauf zu? Und kann er auch seine Waschmaschine daran anschließen? Oder reicht nach dem Blick in die Preislisten doch eine einfache Steckdosenleiste mit Ethernet-Anschluss?

Integrationsplattform

Egal welche Entscheidung man trifft, die Wahrscheinlichkeit, ein System zu wählen, das in ein paar Jahrzehnten kaum noch jemand kennt und für das es keine erschwinglichen Ersatzteile mehr gibt, ist äußerst hoch. Auch wird man kein Angebot finden, das in allen Bereichen überzeugt und sämtliche Wünsche zu einem akzeptablen Preis erfüllen kann. Man wird aber kaum mit mehreren Insellösungen leben wollen, die jeweils ihre eigene iPhone-App brauchen und nicht interagieren können. Es gilt also, eine offene Integrationsplattform zu

finden, die es zum einen erlaubt, flexibel neue Systeme anzuschließen (bzw. bestehende durch eine andere Technologie zu ersetzen) und zum anderen ein über alle Systeme einheitliches User Interface sowie systemübergreifende Automatisierungsregeln bietet. Genau hier setzt der Open Home Automation Bus an.

Die Idee einer solchen Plattform ist natürlich nicht neu, sondern schon so alt wie die Idee des Smart Homes selbst. Doch fast alles, was sich im Open-Source-Umfeld finden lässt, kommt aus der Linux/Skripting-Community – als prominentes Beispiel für ein sehr umfangreiches Projekt sei hier Misterhouse [5] erwähnt. Doch was macht der verwöhnte Java-Entwickler, der den Komfort einer IDE wie Eclipse gewohnt ist – von Syntax Checks über Content Assist und einem leistungsfähigen Debugger? Genau an diesen wendet sich openHAB – mit Equinox als serverseitige OSGi Runtime, Jetty als HTTP-Server, Xtext für Konfigurationen und JBoss Drools für die Automatisierungsregeln. Doch eins nach dem anderen.

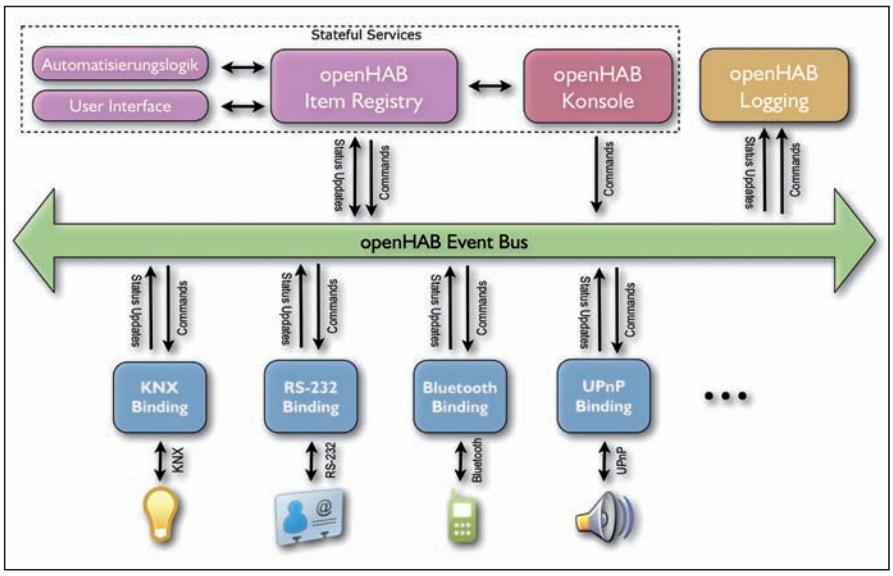

Abb. 1: Die Architektur der openHAB Runtime

openHAB-Architektur

openHAB besteht aus zwei Teilen: der Runtime und dem Designer. Die Runtime läuft serverseitig in einer JVM und nutzt Equinox als OSGi-Umgebung. Sie ist plattformunabhängig und läuft auf Windows, Mac und Linux gleichermaßen. Der Designer dient der Konfiguration – anstelle eines einfachen Texteditors können hier die Konfigurationsdaten und Regeln mit komfortablen Editoren erstellt und bearbeitet werden. Das reduziert das sonst oft nötige Trial-and-Error-Verfahren und reduziert die Gefahr, dass sich Bugs einschleichen – und diese sind in Heimautomatisierungsregeln oft nicht lustig, zumindest nicht für den Rest der Familie.

Die Grundidee der openHAB Runtime basiert auf einem Event Bus nach dem Publish Subscribe Pattern, der technisch durch den OSGi Event Admin Service realisiert ist. Jedes anzusteuernde Gerät bzw. jeder Sensor wird als abstraktes „Item“ definiert, wobei ein „Item“ beispielsweise einen Schalter, einen Temperaturmesswert oder auch den aktuellen Radiosender repräsentieren kann. Erst so genannte Bindings koppeln Items an konkrete Hardware, Schnittstellen oder Protokolle, wie in **Abbildung 1** zu sehen ist. Diese Trennung sorgt dafür, dass auf Basis der Items Benutzerschnittstellen gestaltet sowie Automatisierungsregeln formuliert werden können, ohne dass auf konkrete Hardware Bezug genommen werden muss. So kann sich das System jederzeit um neue Komponenten erweitern, darüber hinaus können bestehende Items durch Änderung der Bindings an neue Hardware gekoppelt werden.

Was verbirgt sich nun technisch hinter einem Item? Es hat im Wesentlichen einen eindeutigen Namen und einen

Was ist ein Smart Home?

Der Begriff „Smart Home“ bezeichnet die Automatisierung und Vernetzung von Hauselektrik (Licht, Rollläden etc.), Elektrogeräten (Waschmaschine, Kühlschrank etc.) und Unterhaltungselektronik (TV, Hifi etc.). Im deutschsprachigen Raum wird allgemein auch der Begriff „Intelligentes Wohnen“ [8] verwendet. Dabei werden je nach Einsatzart verschiedene Ziele verfolgt:

- *Komfort*: das offensichtlichste Merkmal – das Haus denkt mit, man selbst wird entlastet. Eine besondere Form hiervon ist der Bereich Ambient Assisted Living [9], bei dem ältere Menschen von intelligenter Technik unterstützt werden, wobei sich diese fast unsichtbar im Hintergrund hält.
- *Sicherheit*: Eine professionelle Alarmtechnik wird man nur in den wenigsten Smart Homes finden, aber bereits zeitgesteuerte Rollläden, ein Hinweis auf offene Fenster beim Verlassen des Hauses oder die automatische Abschaltung des Bügeleisens können die Sicherheit deutlich erhöhen. Auch SMS-Benachrichtigungen sind eine sinnvolle Ergänzung.
- *Energieeinsparung*: Im Rahmen der Realisierung von Smart Grids versuchen die Energieversorgungsunternehmen im Bereich Smart Home Fuß zu fassen. Für den Verbraucher lässt sich mit einer intelligenten Heizungssteuerung oft eine deutliche Einsparung realisieren, z. B. durch Nachtabsenkung, Präsenzerkennung oder Fernsteuerung (aus dem Urlaub). Für die Energieversorger wird es erst richtig interessant, sobald der Strompreis flexibel gestaltet werden kann, d. h. zu Spitzenzeiten der Strom teuerer ist als beispielsweise in der Nacht. Auch wenn mittlerweile intelligente Stromzähler installiert werden, wird es noch einige Jahre dauern, bis die Netzinfrastruktur und die Abrechnungsverfahren dafür bereitstehen.

Im Allgemeinen wird man in Smart Homes eine Kombination von allen drei Bereichen vorfinden, da sie oft sinnvoll ineinandergreifen. Welche Ausprägung man wählt, hängt vom persönlichen Geschmack und auch vom Budget ab.

Typ (z. B. Switch, Dimmer, Number etc.). Ein Typ definiert mögliche Zustands- bzw. Kommandotypen eines Items. So kann ein Dimmer z. B. Boolean-Werte (*ON/OFF*) und Prozentwerte als Status haben und zusätzlich Kommandos wie *increase* oder *decrease* akzeptieren. Weiterhin kann zu einem Item ein Default-Labeltext und ein Default-Item definiert werden. Eine besondere Stellung haben Items des Typs *Group*. Sie gruppieren, wie der Name schon sagt, mehrere Items entweder logisch (z. B. alle Items eines bestimmten Raums) oder nach Typ (alle Lichter). Über solche Gruppen kann dann sehr leicht der Status von mehreren Items gleichzeitig überwacht („irgendwo ein Fenster offen?“) bzw. ein Kommando an viele Geräte geschickt werden („alle Lichter aus!“).

Items werden der Runtime durch *ItemProvider* als OSGi-Service bereitgestellt. So können dynamisch zur Laufzeit Änderungen vorgenommen und Items hinzugefügt oder entfernt werden. Dennoch ist es in den meisten Fällen sinnvoll, Items statisch zu definieren, denn verbaute Schalter oder Sensoren ändern sich nicht so oft. Hierzu bietet openHAB einen *ItemProvider*, der Dateien mit einer einfachen Syntax einliest. Diese Syntax ist als Xtext-Grammatik definiert, wodurch komfortable Editoren im openHAB-Designer zur Verfügung stehen. Ein einfaches Beispiel für eine solche Datei zeigt Listing 1.

Listing 1

```
/* Gruppen */
Group   Licht
Group   Bad   "Bad"

/* Badezimmer */
Switch          Licht_Bad_Decke     "Deckenlicht"                (Bad, Licht)
Switch          Licht_Bad_Spiegel   "Spiegellicht"               (Bad, Licht)
Number          Temperatur_Bad      "Temperatur [%.1f °C]" <temp>  (Bad)
Switch          Heizung_Bad         "Heizung" <heating>          (Bad)
Rollershutter   Rollladen_Bad       "Rollladen"                  (Bad)
Contact         Fenster_Bad         "Fenster [%s]"               (Bad)

/* Wetter */
Number          Aussentemperatur    "Außentemperatur [%.1f °C]"  <temp>
Number          Wind                "Windgeschwindigkeit [%.1f m/s]" <wind>
Number          Helligkeit          "Helligkeit [%.0f Lux]"      <sun>

/* Status */
Switch          Anwesend            <present>
```

Listing 2

```
sitemap mein_haus label="Mein Haus" {
  Frame {
    Group item=Bad icon="bath"
  }
  Frame label="Status" {
    Switch item=Anwesend
  }
  Frame label="Wetter" {
    Text item=Aussentemperatur
    Text item=Wind
    Text item=Helligkeit
  }
}
```

User Interface

Sind die Items dem System bekannt, kann darauf aufbauend ein User Interface erstellt werden. Zur Zeit bietet openHAB hier eine Webapplikation, die durch CSS und JavaScript eine iPhone-App nachbildet [6], aber gegenüber einer nativen App den Vorteil hat, dass sie auch problemlos auf anderen Mobilplattformen wie Android und WebOS oder auch auf PCs mit WebKit-basierten Browsern wie Safari und Chrome funktioniert. Diese Webapplikation registriert sich als OSGi *HttpService* und wird durch den in der openHAB Runtime integrierten Jetty HTTP Server verfügbar gemacht. Hierbei wird die Möglichkeit genutzt, Jetty in Form von OSGi Bundles auf Equinox zu betreiben [7]. Das hat den Vorteil, dass Jetty nicht als getrennter Prozess gestartet werden muss, sondern integraler (aber dennoch optional abschaltbarer) Bestandteil der openHAB Runtime ist. Selbstverständlich lässt er sich entsprechend den eigenen Bedürfnissen frei konfigurieren, also z. B. für sicheren Zugriff über TLS/SSL, Authentifizierung via LDAP etc.

Um den Aufwand für die Erstellung und Pflege des User Interface gering zu halten, setzt openHAB auf einen deklarativen Ansatz: In einer eigenen Xtext-Grammatik werden so genannte Sitemaps erstellt, die den Seiteninhalt sowie eine einfache Navigationsstruktur durch Verschachtelung beschreiben. Eine Seite ist dabei aus einer Liste von Widgets aufgebaut, die mit Items assoziiert werden. Item-Gruppen verlinken automatisch zu einer dynamisch erstellten Unterseite, so muss z. B. für die Items des Bads im Beispiel keine explizite Konfiguration vorgenommen werden, wie man in Listing 2 und am Ergebnis in **Abbildung 2** sieht.

Hardware-Bindings

Wir haben gesehen, wie wir alle Geräte modellieren und für diese ein User Interface erstellen können. Interessant wird es aber erst, wenn wir auch reale Hardware daran anschließen. Für diesen Zweck gibt es die so genannten Bindings. Ein Binding ist üblicherweise ein optional verfügbares OSGi Bundle für eine konkrete Hardware bzw. für ein Interface oder Protokoll. Ein solches Binding verbindet sich mit dem Event Bus und „übersetzt“ die Kommandos und Statusupdates zwischen diesem Bus und der Hardware. Eine spezielle Konfiguration sagt dem Binding, für welche Items es diese Übersetzung vornehmen soll. Beispiele für existierende Bindings sind folgende:

- *KNX* (vormals *EIB*): Über ein KNX-IP-Gateway (oder das Softwareäquivalent eibd [10]) werden Bus Events

als KNX-Telegramme weitergegeben und umgekehrt. Das ermöglicht insbesondere die Anbindung von Hauselektrik, sofern diese mit einem KNX-Bus ausgeführt ist.

- *1-Wire*: 1-Wire [11] ist ein beliebtes System, um kostengünstig Messwerte (z. B. Temperatur, Feuchtigkeit etc.) zu erfassen.
- *RS-232*: Dieses Binding ermöglicht es ganz allgemein, Hardware anzubinden, die über eine serielle Schnittstelle kommuniziert. Dabei werden sowohl binäre Schalt-Items (Signalisierungen auf RS-232) als auch String Items (übertragene Daten) unterstützt. Anwendungsfälle sind z. B. RFID-Leser, steuerbare AV-Receiver u. Ä.
- *Bluetooth*: Sind Bluetooth-Geräte in der Nähe, können Schaltvorgänge ausgelöst werden (z. B. als Präsenzerkennung). Auch die Liste aller Bluetooth-Geräte in Reichweite ist auswertbar.
- *Wake-on-LAN*: Hiermit können Geräte im lokalen Netzwerk gestartet werden, z. B. ein NAS beim Betreten des Hauses.
- *HTTP*: Dieses Binding bietet zweierlei Funktionalität: Zum einen können openHAB-Kommandos Zugriffe auf URLs auslösen, um z. B. Geräte mit Webinterface anzusteuern. Zum anderen kann per Web-Scraping Information aus dem Web für ein openHAB Item genutzt werden. So können beispielsweise Wetterdaten von einer Website ausgewertet werden.

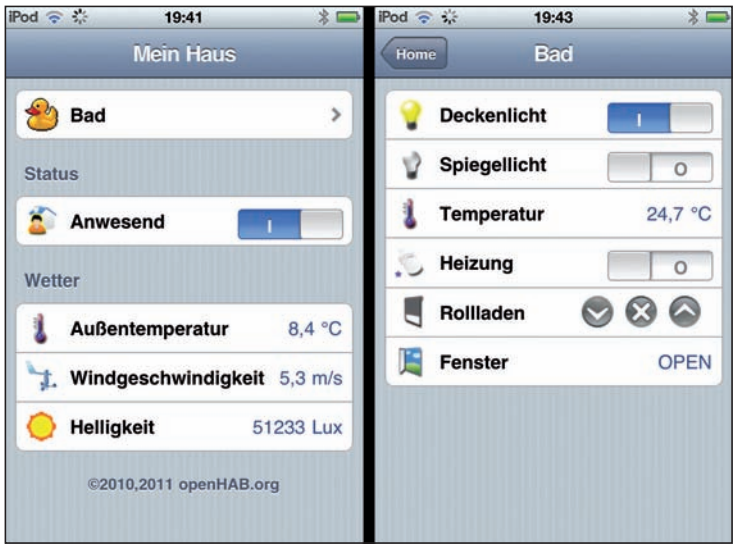

Abb. 2: Das User Interface

- *Kommandozeile*: Das Schweizer-Messer-Binding ermöglicht es, beliebige Kommandos auf der Kommandozeile des Host-Rechners abzusetzen; zu beachten ist jedoch, dass mit der Nutzung dieses Bindings die Plattformunabhängigkeit aufgegeben wird.

Die Binding-Konfiguration kann flexibel per OSGi-Service vorgenommen werden. In den meisten Fällen lässt sich das

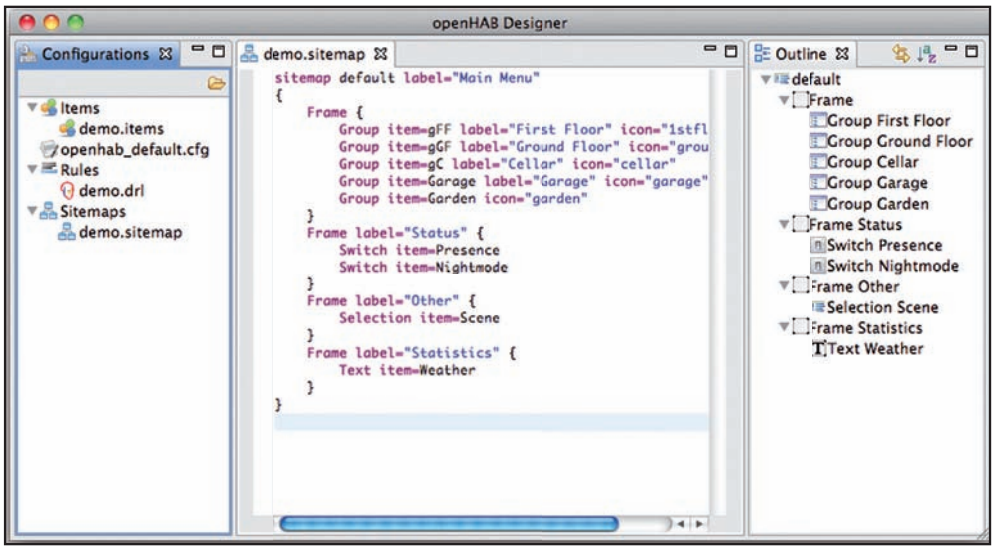

Abb. 3: Der openHAB-Designer in Aktion

aber auch komfortabler über die Items-Definitionsdatei erledigen. Hierfür ergänzt man die Item-Definition um die notwendigen Informationen. Zum Beispiel koppelt man den Schalter ANWESEND an sein Handy mit *Switch Anwesend { bluetooth="EC9B5BC453E6" }* oder an einen Taster mit Statusrückmeldung am KNX-Bus mit *Switch Anwesend { knx="2/4/31+0/4/31" }*. Man hat auf diese Weise jederzeit die Möglichkeit, angeschlossene Hardware auszutauschen oder umzukonfigurieren, ohne dass es eine Auswirkung auf das definierte User Interface oder die Automatisierungsregel hat. Für Hardware, die bisher noch nicht von openHAB unterstützt wird, lässt sich leicht ein eigenes Binding Bundle implementieren und der Runtime hinzufügen.

Regeln

Für die Formulierung der Regeln steht das mächtige Werkzeug Drools Expert [12] von der JBoss-Community bereit. Es handelt sich hierbei um eine vollwertige Rule Engine, die auf dem Rete-Algorithmus basiert. Als Expertensystem lassen sich mit Drools deutlich komplexere Regelwerke abbilden, als es mit einer einfachen sequenziellen Abarbeitung möglich wäre. Regeln können gewichtet, gruppiert und dynamisch aktiviert oder deaktiviert werden. Durch diese Flexibilität lassen sich auch komplexe Anforderungen elegant umsetzen. Zusätzlich unterstützt Drools Timer und Kalender, was insbesondere für den Heimautomatisierungsbereich ein wichtiges Feature ist. Als Syntax für die Regeln stehen MVEL [13] und Java zur Verfügung, sodass eine Regel zum Beispiel folgendermaßen aussehen kann:

```
rule "Haustürklingel"
when
  Item(name="Klingel", hasChanged==true, state==OnOffType.ON)
  Item(name="Anwesend", state==OnOffType.OFF)
then
  XMPP.send("user@jabber.org", "Es hat an der Haustür geklingelt!");
end
```

Auch für die Regeln enthält der openHAB-Designer komfortable Editoren mit den üblichen Features wie Content Assist und Syntaxüberprüfung. Zukünftig wird openHAB auch eine eigene Regel-DSL unterstützen, um die Formulierung von Regeln noch weiter zu vereinfachen.

Konnektivität

Wie in obigem Beispiel gezeigt, sind Regeln oft nützlich, um bestimmte Zustände und Ereignisse zu protokollieren oder zu melden. Für diesen Zweck bietet openHAB diverse Kommunikationswege. So können Nachrichten als Instant Message über das XMPP-Protokoll verschickt werden; alternativ kann das genauso einfach über E-Mails oder Prowl [14] Push Notifications geschehen. Für lokale Benachrichtigungen kann auch einfach auf die Soundausgabe zurückgegriffen werden; neben Kontrolle der Ausgabelautstärke können Sounddateien oder auch Internetstreams abgespielt werden. Für die Zukunft ist ebenfalls eine Sprachausgabe geplant.

Für den Remote-Zugriff kann nicht nur das Web-User-Interface genutzt werden, sondern auch XMPP. Hierbei wird per Instant Messenger eine Konsole angeboten, über die der Status von Items direkt abgefragt oder Kommandos gesendet werden können.

Praxisszenario

An einem Beispiel soll nun gezeigt werden, wie sich ein Smart Home gestalten und openHAB in der Praxis einsetzen lässt. Da zum Betrieb im Wesentlichen nur eine JVM vorausgesetzt wird, läuft die openHAB Runtime im Beispielhaus auf einem günstigen Netbook. Der Großteil der Aktorik und Sensorik ist über ein KNX-Interface am KNX-Bus angeschlossen. Das sind zum einen alle Lampen, Rollläden und elektrischen Heizelemente sowie ausgewählte Steckdosen. Zum anderen ist jeder Raum mit einem Temperatur- und einem Helligkeitssensor, einem Bewegungs- und einem Rauchmelder sowie einem Lautsprecher ausgestattet. Alle Außentüren und Fenster verfügen über Magnetkontakte, die Haupteingangstüren zusätzlich über Motorschlösser. Außen gibt es Sensoren für Temperatur, Helligkeit, Windgeschwindigkeit, Niederschlag und Wasserdruck in der Zisterne zur Füllstandsbestimmung. Über Magnetventile können Wasserleitungen im Garten geschaltet werden. Weiterhin befindet sich in Haustürnähe ein Bluetooth-Empfänger, der per USB direkt ans Netbook angeschlossen ist.

Alles zusammen ergibt das mehrere hundert Geräte und Sensoren, die an openHAB gekoppelt sind. Allein diese Menge macht deutlich, dass man sich bei der Logikerstellung schnell in komplexen Szenarien wiederfindet.

Um die Anzahl der Regeln in openHAB klein zu halten, sind alle einfachen Fälle („Schalter x schaltet Licht y“) direkt in die KNX-Geräte programmiert. Das hat zusätzlich den Vorteil, dass bei einem eventuellen Ausfall von openHAB die Grundfunktionen der Haussteuerung weiterhin verfügbar bleiben. Komplexe Anwendungsfälle sind hingegen die Domäne von openHAB, hier ein paar Beispiele:

- Klingelt es an der Tür, so wird die Musiklautstärke reduziert und ein „Gong“ abgespielt – ist hingegen niemand zu Hause, meldet eine Regel den Besucher per Prowl-Push-Nachricht auf das Smartphone; zusätzlich wird auf Bluetooth-Geräte in Reichweite geprüft und gegebenenfalls der Eigentümer über eine Lookup-Tabelle ermittelt und ebenfalls gemeldet. Per E-Mail erhält der Hausherr weiterhin ein Bild der in die Türstation integrierten IP-Cam.
- Wenn es im Sommer mehr als drei Tage nicht geregnet hat, wird am Abend die Gartenbewässerung automatisch über Magnetventile gestartet, wenn kein Hausbewohner mehr auf der Terasse ist (festgestellt durch Bewegungsmelder und Türkontakt).
- Um eine durchgängige Bewässerung gewährleisten und ein Trockenlaufen der Gartenpumpe verhindern zu können, wird automatisch Leitungswasser in die Zisterne gefüllt, wenn deren Füllstand unter fünf Prozent sinkt. So ist sogar während der Urlaubszeit der Garten gut versorgt.
- Wird während Abwesenheit ein (Innen-)Bewegungsmelder ausgelöst oder ein Fenster geöffnet, so wird das ebenfalls per Push Notification gemeldet. Optional können zur Abschreckung die Signaltöne der Rauchmelder aktiviert, Rollläden hochgefahren und Lichter angeschaltet werden.

Weniger spannend, aber Bestandteil des Pflichtprogramms für ein Smart Home sind außerdem Funktionen wie Beschattungsautomatik nach Sonnenstand, Szenenabruf (d. h. mit einem Tastendruck vieles gleichzeitig schalten, z. B. Szene „Dinner“: Licht gedimmt, TV aus, sanfte Musik), All-off-Funktion (beim Verlassen des Hauses) usw. Die Liste möglicher Einsatzzwecke ließe sich beinahe unbegrenzt fortsetzen.

Ausblick

Generell sind für openHAB noch viele Erweiterungen denkbar – angefangen von weiteren Bussystemen wie X10 oder Insteon über die Anbindung von Telefoniesystemen wie Asterisk oder VoIP-Routern. Ein weiteres spannendes Thema ist das Visualisieren von Daten in Form von Grafiken. Hierfür wird an einer Integration von rrd4j [15] gearbeitet, das dann z. B. Temperaturverläufe als Chart darstellen kann. Auch wird es nicht bei einem User Interface bleiben – das bisherige bietet zwar für Smartphones ein übersichtliches und leicht zu bedienendes Layout, nutzt aber auf Tablets oder PCs die verfügbare Displayfläche nicht effektiv. Für Tablets ist daher ein alternatives UI auf Basis des HTML5-basierten SenchaTouch-Frameworks [16] geplant. Darüber hinaus wäre die Integration des eigenen Google-Kalenders in Regeln denkbar, sodass bestimmte Abläufe, z. B. eine Weckfunktion oder die Erinnerung an Mülltonnenleerungen, komfortabel über den Kalender definiert werden können, ohne Regeln in openHAB verändern zu müssen. Nähere Informationen zur Roadmap von openHAB finden Sie unter [17].

Fazit

Wer in der Java-Welt zu Hause ist und eine Heimautomatisierungssoftware sucht, bei der die Möglichkeit besteht, selbst Erweiterungen zu entwickeln oder Veränderungen vorzunehmen, der findet in openHAB eine ideale Plattform. Die modulare Architektur macht es nicht nur leicht, Funktionen als neue Bundles zu entwickeln, die verwendeten Technologien von OSGi/Equinox über Xtext, Drools und Maven Tycho sorgen auch dafür, dass diese Entwicklung sogar viel Spaß macht.

Kai Kreuzer arbeitet als Softwarearchitekt mit den Schwerpunkten MDD und Eclipse RCP bei Odyssey Financial Technologies in Lausanne (Schweiz). In seiner Freizeit beschäftigt er sich mit dem Thema Heimautomatisierung und ist Gründer des Open-Source-Projekts openHAB.

Links & Literatur

[1] http://www.openHAB.org
[2] http://www.knx.org
[3] http://www.enocean.com/de
[4] http://www.digitalstrom.org
[5] http://misterhouse.sourceforge.net
[6] http://webapp-net.com
[7] http://wiki.eclipse.org/Jetty/Tutorial/Jetty-OSGi_SDK
[8] http://www.intelligenteswohnen.com
[9] http://de.wikipedia.org/wiki/Ambient_Assisted_Living
[10] http://www.auto.tuwien.ac.at/~mkoegler/index.php/eibd
[11] http://de.wikipedia.org/wiki/1-Wire
[12] http://www.jboss.org/drools/drools-expert.html
[13] http://mvel.codehaus.org
[14] http://prowl.weks.net
[15] http://code.google.com/p/rrd4j
[16] http://www.sencha.com/products/touch
[17] http://code.google.com/p/openhab/wiki/Roadmap