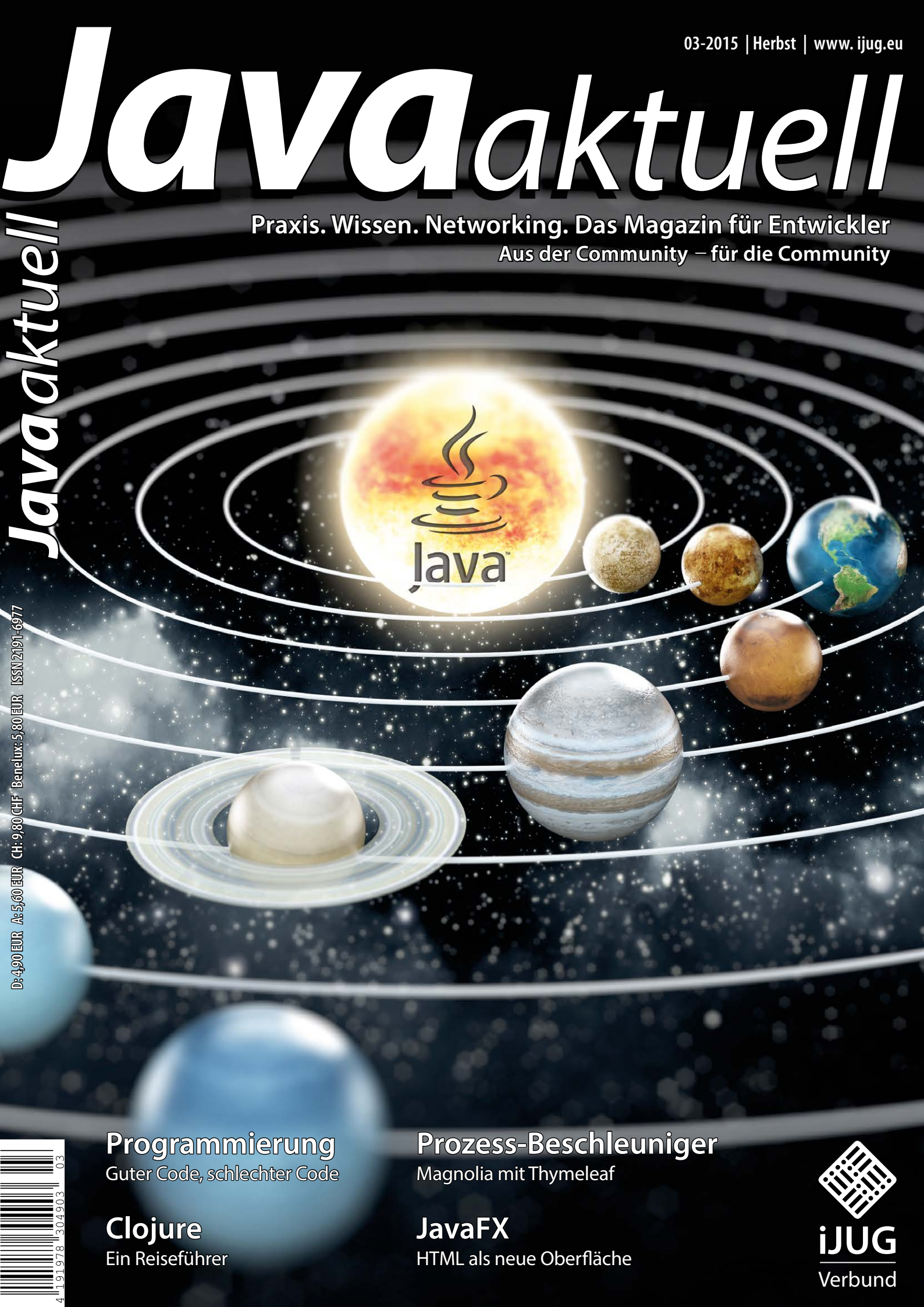
03-2015 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell



D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Programmierung
Guter Code, schlechter Code

Clojure
Ein Reiseführer

Prozess-Beschleuniger
Magnolia mit Thymeleaf

JavaFX
HTML als neue Oberfläche

iJUG Verbund

Kunstprojekt im JavaLand 2015

Seit Java 1.5 erlaubt die Java Virtual Machine die Registrierung sogenannter „Java-Agenten"

Bei Mgnolia arbeiten Web-Entwickler und CMS-Experten mit ein und demselben Quellcode

Ein Heim-Automatisierungs-Projekt

First one home, play some funky tunes!

Pascal Brokmeier, OPITZ CONSULTING Deutschland GmbH

Das Internet der Dinge (IoT) ist in aller Munde und bringt der Heim-Automatisierung sowohl im Bastlerumfeld als auch auf kommerzieller Seite frischen Wind in die Segel. Dabei bietet sich die Java-Plattform geradezu an, um die in fast jeder Architektur auffindbaren Gateways zu treiben und die Logik näher an die Grenze der realen Welt zu bringen. Ausgehend von diesen Gegebenheiten entstand ein Heim-Automatisierungs-Projekt auf Basis günstiger Hardware und gängiger Java-Enterprise-Technologien, das zeigt, wie Entwickler bestehendes Backend-Wissen für zukünftige IoT-Projekte einsetzen können.

Wer kennt das nicht: Licht angelassen, Soundsystem über Nacht eingeschaltet oder vergessen, die Heizung herunterzudrehen … Motivationen für die Heim-Automatisierung sind lange bekannt und bereits seit einiger Zeit gibt es Lösungsansätze zur automatisierten Steuerung von Gebäuden. Das Problem dabei: Diese Systeme waren bisher meist unverhältnismäßig teuer, basierten auf kabelgebundenen Bus-Systemen oder nutzten proprietäre und schwer erweiterbare Systeme.

Die Entwicklungen der letzten Jahre lassen jedoch einen Wechsel erhoffen. Geräte wie der Raspberry Pi bieten Bastlern eine günstige Basis, um vollwertige Java-Server-Applikationen zu betreiben. Offene APIs wie RESTful-Webservices führen zu einfacherer Interoperabilität. Mithilfe dieser Services ist es möglich, herstellerübergreifende Architekturen zu entwickeln. Eine einfache Suche auf Google nach „DIY home automation" zeigt, dass es unglaublich viele Projekte gibt. Viele davon setzen auf dem Raspberry Pi als Hardware-Plattform auf.

Raspberry PI und Java

Der Prototyp eines Projekts zur Heim-Automatisierung, das der Artikel näher vorstellt, basiert auf Raspberry Pi und Java-Framework. Ziel ist es, abhängig von Benutzer-Präsenzen und Wetterverhältnissen bestimmte Licht-Einstellungen vorzunehmen oder Musik abzuspielen. Technischer ausgedrückt: Es sollen kontextsensitive Regeln erstellt werden, die an Aktionen gekoppelt sind. Dafür ist von Anfang an eine offene Architektur wichtig. Diese soll folgende Aufgaben erfüllen:

- Die Steuerung weiterer Geräte-Typen ermöglichen
- Andere Informationsquellen integrieren
- Neue Regeln zur Kombination von Event-Erzeugern und Akteuren erfassen

Ein weiterer Aspekt sind die Hardware-Kosten: Hier soll so wenig Geld wie möglich investiert werden. Für einfachste Anwendungsfälle lassen sich die in vielen Haushalten bereits vorhandenen, simplen 433-MHz-Funksteckdosen integrieren. Diese reichen aus, um einfache Leuchten und ältere elektronische Geräte anzuschalten.

Die Architektur des Prototyps

Hersteller wie Oracle und Intel setzen bei ihren IoT-Architekturen auf Gateways, um die Brücke zwischen den Dingen und komplexerer IT wie Serveranwendungen, Big Data etc. zu schlagen. Oft wird auf diesen Gateways bereits beträchtliche Intelligenz ausgeführt, um zum Beispiel Events zu filtern oder lokale Steuerungen vorzunehmen. Dieses Konzept wird im Prototyp aufgegriffen; bereits mit kleinen Gateways ist also einiges möglich.

Die zentrale Einheit des Systems bildet eine Spring-Applikation auf einem Raspberry Pi, gehostet durch einen Jetty-Server. Sie stellt ein REST-API zur Steuerung von Geräten und zur Änderung der Benutzer-Zustände zur Verfügung und kann entweder manuell mit einer Mobile App angesprochen oder durch andere Dinge genutzt werden.

Ein zweiter Raspberry Pi ermittelt, ob sich Benutzer im Haushalt befinden. Er hostet eine Oracle-Event-Processing-Applikation (OEP). Taucht ein Benutzer im lokalen Netz auf, wird diese Information an die Spring-Applikation weitergegeben. Diese prüft daraufhin ihre Regeln und schaltet entsprechende Gerätegruppen *(siehe Abbildung 1)*.

Struktur der Spring-Applikation

In der Spring-Applikation sind Camunda BPM, ein Prozessautomatisierungs-Framework, sowie Drools, eine Rule Engine, integriert. In

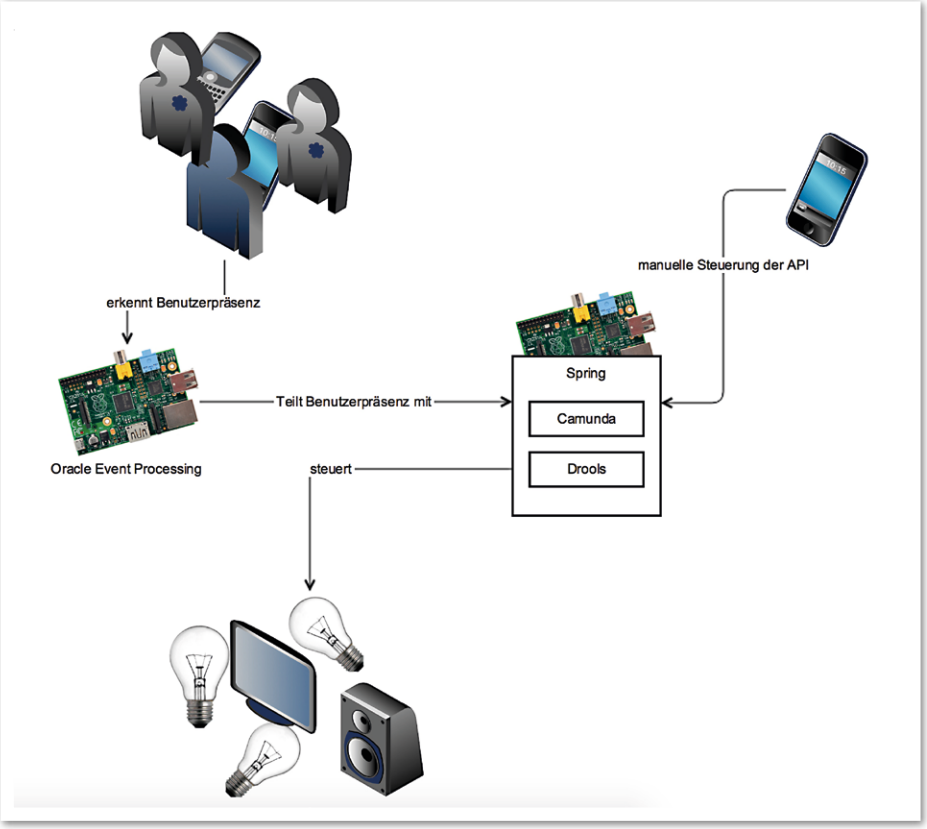


Abbildung 1: System-Übersicht

Drools können diverse Regeln zur Steuerung der Beleuchtung und der Geräte hinterlegt werden, etwa „Alle Bewohner zu Hause + draußen dunkel oder regnerisch = Wohnzimmer-Beleuchtung an". Oder „Nur Person X daheim = Licht im Büro an + leise Radiomusik".

Der Spring-Server ist in Schichten aufgeteilt. Sobald die Logik des Event-Processing eine Änderung bei der Benutzerpräsenz erkennt, wird diese durch einen API-Aufruf an den Spring-Server übermittelt. Die Camunda-Prozesse greifen für ihre Tätigkeiten auf die darunterliegenden Services zu und orchestrieren verschiedene Handlungen *(siehe Abbildung 2)*.

Zur Verdeutlichung: Wird ein Event übermittelt, der besagt, dass ein Benutzer das Haus betreten hat, wird ein entsprechender Prozess gestartet. Hier wird nun geprüft, welche Geräte geschaltet werden sollen. Dazu wird auf die Regeln von Drools zugegriffen. Diese geben vor, welche Geräte für diesen Benutzer geschaltet werden. Während diese Regeln aktuell noch abhängig von der jeweiligen Präsenzsituation schalten, lassen sie sich zukünftig beliebig erweitern. So könnte eine Regel heißen: „Wenn Benutzer X nach Hause kommt, niemand sonst daheim ist, es draußen dunkel ist oder es laut Wetterservice gerade regnet, dann soll das Licht im Wohnzimmer angehen und Musik mit dem Thema funky' abgespielt werden" *(siehe Listing 1)*.

Dieses Regelwerk wird aus der Camunda-Lösung heraus aufgerufen. Nachdem alle Regeln durchlaufen sind, werden die gewünschten Geräte geschaltet und die auszuführenden Befehle ausgeführt. Konkret wird aus der Persistenz-Schicht ein Objekt des Typs „Musicbox" geladen und bei diesem die Methode „searchAndPlay(„funky")" ausgeführt. Der in der Service-Schicht liegende Adapter für die Musicbox schickt diesen Befehl anschließend an den Raspberry Pi, der an das Soundsystem angeschlossen ist. Dieser spielt Musik ab, die via Google Music zur Suche „funky" gefunden wurde *(siehe Abbildung 3)*.

Funksteckdosen aus Java steuern

Um die Funksteckdosen zu steuern, muss man sich als Java-Entwickler ein wenig aus seiner gewohnten Umgebung heraustrauen. Der Raspberry Pi bietet mit seinen GPIO-Pins eine Hardware-Schnittstelle zur Steuerung eines 433-MHz-Funkmoduls. Es existiert eine C++-Bibliothek „wiringpi" *[1]*, um die Steuerung der GPIO-Pins zu vereinfachen. Mit „rcswitch-pi" *[2]* können dann Funksteckdosen in C angesteuert werden. Dazu müssen die 2x5-Bit-Codes für die gängigen ELRO-Chip-Steckdosen übergeben werden. Nachdem „wiringpi" im System installiert wurde und somit unter „/usr/local/lib" liegt, können diese Bibliotheken genutzt werden.

Um diese Funktionalität in Java möglich zu machen, ist ein Java Native Interface (JNI) erforderlich. Dazu erstellt man eine Java-Klasse, etwa „NativeRCSwitchAdapter, und schreibt native Methodenköpfe: „public native void switchOn(String group, String channel);"

Anschließend kompiliert man die Klasse mit „javac" und lässt C-Header-Dateien mittels „javah com.opitz.jni.NativeRCSwitchAdapter" generieren. So entsteht eine C-Header-Datei mit allen Funktions-Deklarationen. Diese Funktionen sind gebunden an die Methoden aus der Java-Klasse. Wird die oben beschriebene Methode aufgerufen, führt die JVM die Funktion im C-Code aus. Tiefer soll jetzt nicht in die Entwicklung in C eingestiegen werden. Die durch „javah" generierten Funktions-Deklarationen sind jedoch erwähnenswert *(siehe Listing 2)*.

Das erste Argument im Skript ist ein Pointer zum JNI-Interface. Das zweite stellt eine Referenz zum Java-Objekt dar, die letzten

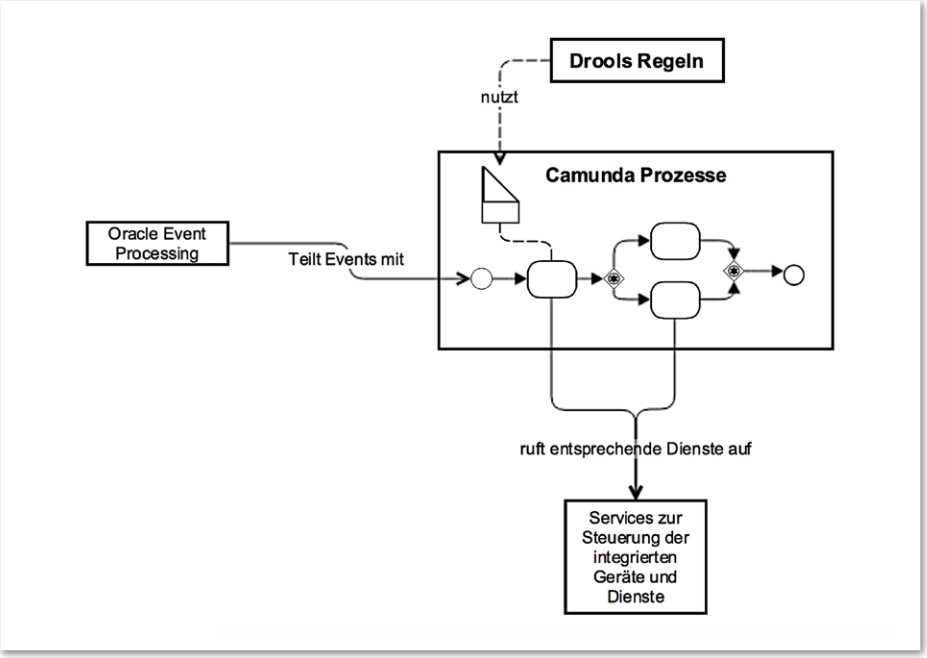


Abbildung 2: Zusammenspiel Camunda BPM, OEP, Services, Drools

```
//Regel schaltet Geräte aus Gruppe "Wohnzimmer-Licht" sowie musicbox an
rule "Apply light theme for Pascal"
when
  u : User( username == "Pascal" )
  u.isAlone()
  WeatherService.isRaining() || WeatherService.isDark()
then
  HashSet<String> devicesToSwitchOn = new HashSet<String>();
  devicesToSwitchOn.add("Wohnzimmer-Licht");
  devicesToSwitchOn.add("Wohnzimmer-musicbox");
  insert(devicesToSwitchOn);
end

//Regel spielt Musik mit Thema funky in Wohnzimmer ab
rule "Apply funky theme for Pascal"
when
  u : User( username == "Pascal" )
  u.isAlone()
  WeatherService.isRaining()
then
  HashSet<Command> commands = new HashSet<Command>();
  ArrayList<String> parameters = new ArrayList();
  parameters.add("funky");
  commands.add(new Command("Musicbox" "Wohnzimmer-musicbox", "searchAnd-
Play", parameters));
  insert(commands);
end
```

Listing 1

beiden sind die eigenen Strings, die der nativen Methode übergeben wurden. Da die String-Objekte nur als Referenz übergeben werden, muss man diese über das „JNIEnv" erfragen. Nach dem Absetzen der Funksignale muss man die String-Objekte noch über „env->ReleaseStringUTFChars(jsGroup, csGroup);" freigeben, damit der Garbage Collector diese aufräumen kann. Andernfalls entstände unter Umständen ein Memory-Leak *[3]*.

Native Bibliotheken in Java laden

Nachdem der C-Code implementiert ist, muss er noch in eine Shared Object Library kompiliert und ebenfalls unter „/usr/local/lib" abgelegt werden. Würde man den Java-Code nun ausführen, bekäme man einen „UnsatisfiedLinkError". Dies liegt daran, dass die JVM noch die entsprechend aufzurufende Bibliothek laden muss. *Listing 3* zeigt, wie das im Projekt gelöst wird.

Dafür bekommt die Datei in der Share-Object-Bibliothek den Namen „libRCSwitchAdapter.so" und wird unter einem der JVM bekannten Bibliotheken-Pfad abgelegt. Nun kann man die native Methode aus Java heraus aufrufen. Diese nutzt anschließend die C++-Funktionen der genannten Bibliotheken, um die Funkwellen der 433-MHz-Signale zu modulieren und damit die Steckdosen zu steuern.

Da die Steckdosen nicht zurückmelden, ob sie wirklich geschaltet haben, empfiehlt es sich, aktiv zu testen, ob die Signale auch überall ankommen. Ein besonders starkes Sendemodul oder eine gute Antenne sowie mehrere Wiederholungen des Sendevorgangs helfen, die erfolgreiche Schaltung sicherzustellen.

Dies zeigt einen wichtigen Punkt auf, den man sich als Entwickler zu Herzen nehmen sollte: Im Kontext des IoT kann TCP/IP-basierte Kommunikation ein Luxus sein. Die Ausprägungen der eingesetzten Geräte sind unterschiedlich – angefangen von einem einfachen Drucksensor bis zu einem hochkomplexen PKW gibt es somit auch viele verschiedene Kommunikationstechnologien. Fehleranfällige, langsame oder sehr einfache Kommunikationswege und deren Einschränkungen sollten also bei bestimmten Anwendungsfällen ausführlich bedacht werden. Im gezeigten Prototyp lassen sich durch das wiederholte Senden der Funkbefehle in der Applikationsebene Schwächen der zugrunde liegenden Kommunikationstechnologie kompensieren.

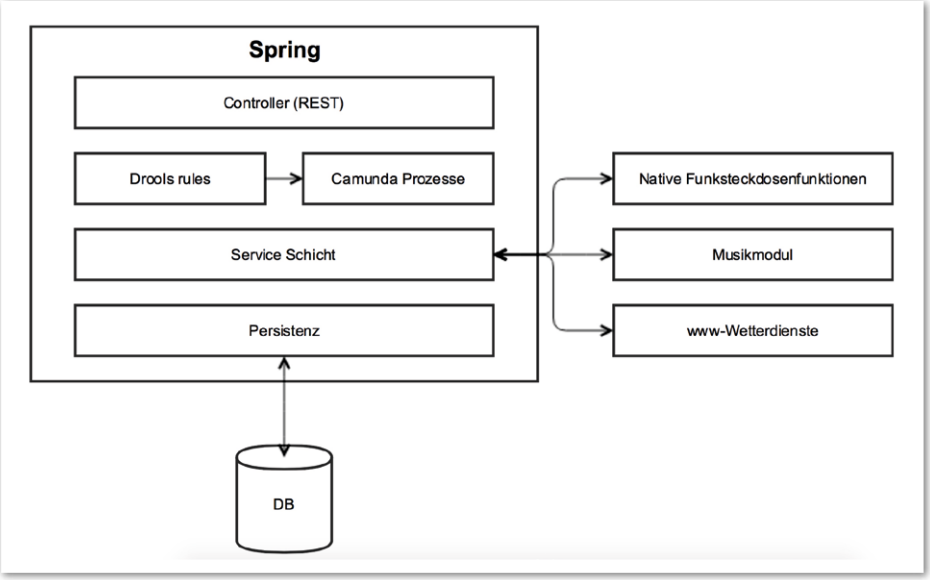


Abbildung 3: Spring-Applikation

Musik-Modul

Um bei der Heimkehr eines Bewohners neben einem warmen Licht auch angenehme Musik abzuspielen, ist ein Musik-Player erforderlich. Hier bietet sich die Pi Musicbox *[4]* an. Dabei handelt es sich um ein Projekt von GitHub, das einen vielseitigen Music-Player auf einen Raspberry Pi bringt und Dienste wie „Spotify", „Google Music" oder „AirPlay" unterstützt. Der Raspberry Pi kann an eine einfache Stereoanlage angeschlossen und mittels Funksteckdosen eingeschaltet werden. Sobald das System hochgefahren ist, startet das Lieblings-Webradio oder es werden über die Schnittstelle des Music Player Daemon (MPD) spezielle Songs oder Playlists ausgewählt *[5]*. Zur Steuerung der MPD-Schnittstelle aus Java heraus gibt es ebenfalls eine „JavaMPD"-Bibliothek. So kann der Spring Server den Musik-Pi wie oben beschrieben auf Basis von Regeln kontextabhängig steuern.

Benutzer-Präsenz und Oracle Event Processing

Die Benutzer-Präsenz-Erkennung ist einfach gehalten. Sie prüft lediglich, ob bekannte, den Benutzern zugeordnete Geräte im Netzwerk gefunden werden. Ein Protokoll namens „ARP" macht dies möglich. Es ist in der Internet Protocol Suite im Link-Layer, also sehr Hardware-nah angesiedelt und überprüft das Netzwerk auf Media-Access-Control-Adressen (MAC-Adressen).

Während sich IP-Adressen oft ändern können und nicht jeder Benutzer eine Client-seitige Applikation auf seinen Geräten installiert haben muss, kann man mithilfe dieses Filters für MAC-Adressen die Präsenz des Client-Geräts unabhängig von der jeweiligen Software feststellen. Hier soll beispielhaft ein Event-Netzwerk vom Oracle Event Processing (OEP) erläutert werden. *Abbildung 4* zeigt das beschriebene Netzwerk.

Der OEP-Pi pingt regelmäßig alle Geräte im Netzwerk an („NetworkNodeAdapter") und vergleicht anschließend die MAC-Adressen der Geräte (ausgelesen aus dem ARP-Cache des Betriebssystems) mit den ihm bekannten Geräten („UserDeviceProcessingBean"). Der Zustand wird für alle relevanten Geräte geprüft („StateCalculatingBean"). Hat sich dieser innerhalb der letzten

```
JNIEXPORT void JNICALL Java_com_opitz_jni_NativeRCSwitchAdapter_switchOn
  (JNIEnv *, jobject, jstring, jstring);
```

Listing 2

```
static{
  System.loadLibrary("RCSwitchAdapter");
}
```

Listing 3

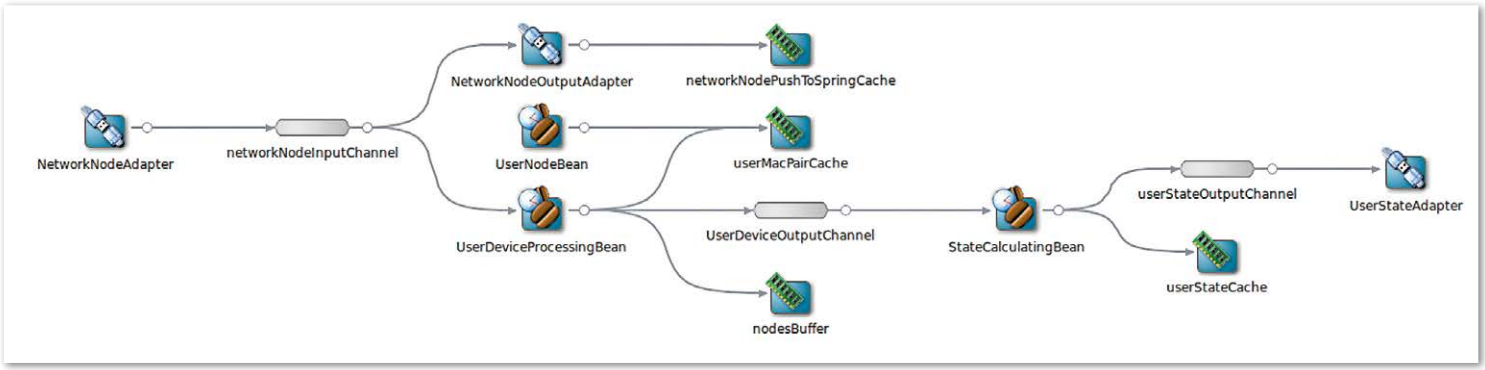


Abbildung 4: Event-Netzwerk

Minuten geändert? Wenn dies der Fall ist, wird ein Event an den Spring Pi geschickt („UserStateAdapter"). Ist der Zustand gleich geblieben, wird kein Event weitergegeben.

Nun ist klar, dass eine Benutzer-Präsenz-Prüfung auf Basis von IP-fähigen Geräten im Netzwerk heute meist noch Zukunftsmusik ist. Während die Präsenz der meisten 14-Jährigen wohl auf diese Weise ermittelt werden könnte, ist dies bei anderen Personengruppen nicht der Fall. Außerdem kann es vorkommen, dass ein Gerät im Flugmodus ist oder keine WLAN-Verbindung besitzt. Das System würde in diesen Fällen also irrtümlich unterstellen, der Benutzer sei nicht mehr anwesend, und dementsprechend Geräte ausschalten. Um diese Schwachstellen zu korrigieren, wird eine Mischung aus Geo-Fencing, Bewegungsmelder oder iBeacons eingesetzt. So kann man verschiedene Event-Typen kombinieren und mit höherer Wahrscheinlichkeit bestimmen, ob sich jemand im Gebäude befindet, um welche Person es sich handelt – und bis zu einem gewissen Grad an Genauigkeit sogar, an welchem Ort sie sich im Gebäude aufhält.

Diese Schwachstelle beschreibt ein noch häufig anzutreffendes Problem im IoT-Kontext: Physische Nähe-Relationen sind schwer in Erfahrung zu bringen. Während die Wissenschaft im Bereich „Mobile Wireless Sensor Network Localization" bereits einige Algorithmen zur Lokalisation von Netzwerk-Knoten ohne Nutzung von Energie-intensivem GPS bietet *[6]*, ist die Praxis noch weit von standardisierten Lokalisierungsmethoden entfernt.

Lessons Learned

Java-Entwickler können mit ihrem bestehenden Skillset auch im Embedded-Bereich bereits einiges leisten. Mit kleinen Abstechern in nativen Code sind auch mit Java direkte Hardware-Interaktionen möglich. Plattformen wie der Raspberry Pi bieten außerdem eine mächtige Grundlage, um komplexere Intelligenz ohne teure Hardware und mit geringem Energie-Aufwand näher an die Grenze zwischen IT und Realität zu bringen. Mithilfe des Event Processing konnte eine architektonische Taktik verfolgt werden, mit der sich aus den massiven Datenflüssen in Richtung „Backend" die wichtigsten Informationen herausfiltern und somit die Last für das Backend reduzieren lassen.

Neben diesen positiven Entwicklungen zeigen sich aber auch noch einige Probleme: Geräte im IoT haben häufig Leistungs-, Kommunikations-, oder Energie-Restriktionen. Diese Faktoren muss der Entwickler mit bedenken und durch entsprechende Maßnahmen kompensieren.

Auch die Kommunikations-Protokolle und -Technologien sind sehr verschieden und sollten für jedes System gründlich evaluiert und verglichen werden. Allein der genannte, noch recht simple Prototyp enthält verschiedene Kommunikationstechnologien: RESTful APIs via HTTP-Requests und WiFi, 433-MHz-Funkkommunikation, MPD, ein spezielles auf TCP aufbauendes Protokoll und ein proprietäres Protokoll, von Philips übertragen über „Zigbee". Viele dieser Technologien bieten aber auch Vorteile gegenüber schwergewichtigen Alternativen. Ein Blick beispielsweise auf CoAP 6LoWPAN und IEEE 802.15.4 zeigt dies.

Fazit

Die Kostenfrage der Heim-Automatisierung ist im Grunde schon beantwortet. Die notwendige Hardware ist in ihrer elementaren Form günstig und leistungsstark. Fertige, bereits am Markt erhältliche Anwendungen sind hingegen meist proprietär und teuer. Es fehlt noch an zufriedenstellenden Software-Systemen, um die vielen verschiedenen Geräte-Typen zu integrieren und für Nicht-Software-Entwickler benutzbar zu machen.

Open-Source-Projekte können dabei helfen, einige gute Projekte herauszustellen, die diverse Plattformen integrieren und so eine einheitliche Schnittstelle bieten. Die dafür hilfreichen Frameworks und Engines sind bereits in Form von Rule Engines und Prozess-Frameworks verfügbar. Die Kabel der verbreiteten BUS-Systeme sind jedoch definitiv Funkverbindungen gewichen. Hier gibt es allerdings wieder viele Standards und Protokolle, die es zu integrieren gilt.

Weitere Informationen

[1] https://projects.drogon.net/raspberry-pi/wiringpi
[2] https://github.com/r10r/rcswitch-pi
[3] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/design.html#wp16696
[4] http://www.woutervanwijk.nl/pimusicbox
[5] http://mpd.wikia.com/wiki/Music_Player_Daemon_Wiki
[6] http://doi.acm.org/10.1145/2677855.2677858 und http://dx.doi.org/10.1007/s00779-013-0692-9

Pascal Brokmeier
pascal.brokmeier@opitz-consulting.com

Pascal Brokmeier arbeitet seit vier Jahren als Software-Entwickler bei der OPITZ CONSULTING Deutschland GmbH, daneben studiert er Wirtschaftsinformatik an der Universität zu Köln. Über mobile Applikationen und Java-EE-Anwendungen hat er sich in den letzten zwei Jahren auf das Internet der Dinge spezialisiert und analysiert hierzu aufkommende Architekturen sowie technische und wirtschaftliche Entwicklungen. Über seine Erkenntnisse hält er regelmäßig Vorträge auf Konferenzen.

http://ja.ijug.eu/15/3/15