HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ins institute for networked solutions

Internet of Things
Smart Home

Bachelorarbeit

HSR - Hochschule für Technik Rapperswil

Institute for networked Solutions

Dokumentation

Autoren: Marco Leutenegger, Dominik Freier

Betreuer: Prof. Hansjörg Huser

Gegenleser: <tbd>Prof. TODO

Abstract

<tbd>

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Management Summary

<tbd>

Ausgangslage

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea.

Vorgehen / Technologien

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea.

Ergebnisse

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea.

Ausblick

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea.

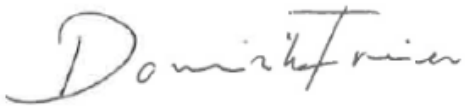
Eigenständigkeitserklärung

Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- das wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum:
Rapperswil, <TBD>



Marco Leutenegger

Dominik Freier

Danksagung

<tbd>

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

Technischer Bericht

1. Ausgangslage

Diese Bachelorarbeit befasst sich mit einem Teilgebiet des «Internet of Things», nämlich der lokalen Vernetzung von Sensoren und Aktoren.

Gemäss der Aufgabenstellung soll eine SmartHome Beispielapplikation erstellt werden, welche wesentliche Aspekte einer IoT-Anwendung demostriert. Das beinhaltet das Steuern von Aktoren, Lesen von Sensoren, Event-Verarbeitung, Überwachung und intelligente Abläufe steuern.
Das System soll auf einer tragbaren, erweiterbaren Architektur aufgebaut werden und Microsoft Azure als Cloud Plattform benutzen.

Die Heimautomation bzw. ein SmartHome grenzt sich von der professionellen Gebäudeautomation in einigen Aspekten ab. Ein SmartHome, wie wir es umsetzen, umfasst insgesamt deutlich weniger Sensoren und Aktoren, stellt dafür aber höhere Ansprüche an die Installierbarkeit, Bedienbarkeit und niedrige Anschaffungskosten. Unsere Arbeit soll zeigen, welche Überlegungen beim Einstieg in die SmartHome-Welt angestellt werden müssen und auf was für Herausforderungen man dabei stösst.

Die Aufgabenstellung schlägt ein System zum Einbruchschutz als Beispielszenario vor.

2. Problembeschreibung

2.1. Motivation

Als SmartHome Beispielszenario soll ein System aufgesetzt werden, das einen grossen Bezug zur Realität hat. Es soll für aussenstehende Personen attraktiv und nachvollziehbar sein und einen Mehrwert mit sich bringen. In diesem Kapitel werden die funktionalen Anforderungen definiert und einige Fachbegriffe, sowie die Martksituation im Bereich SmartHome erklärt. Der Abschnitt mit den funktionalen Anforderungen gliedert sich in ein Basisszenario und einen Lösungsteil. Das Basisszenario befasst sich mit den grundlegenden Konzepten und Anforderungen, im Lösungsteil gehen wir auf die konkreten Anforderungen für unser Beispielszenario «Einbruchschutz» ein.

2.2. Funktionale Anforderungen

2.2.1. Basisszenario

Dieser Abschnitt beschreibt die theoretischen Voraussetzungen, die für das Basiszenario gegeben sein müssen, unabhängig von der späteren Implementierung der konkreten Anwendungsfälle. In der Gebäudeautomation spricht man häufig von sogenannten Sensoren und Aktoren. Beide Begriffe stammen ursprünglich aus der Steuerungs- und Regelungstechnik. Sensoren sind Geräte, die kontinuierlich Messdaten erfassen und die Daten über eine Schnittstelle verfügbar machen. Ob die Daten aktiv vom Sensor an eine Zentrale übermittelt werden, oder ob die Zentrale den Sensor selbstständig abfragen muss ist für die Definition eines Sensors in unserem Rahmen nebensächlich. Ein Aktor kann als Gegenstück zum Sensor betrachtet werden, da er eine aktive Rolle einnimmt. Ein Aktor empfängt Befehle und greift in das Regelungssystem ein. Einige Geräte beinhalten sowohl Sensor- als auch Aktorschnittstellen. Wenn ein Sensor eine Zustandsänderung registriert und diese dem System bekannt gibt, dann sprechen wir von einem *Event*. Als *Command* bezeichnen wir einen Befehl, der an einen Aktor gesendet wird.

Beispiele für Sensoren aus der Gebäudeautomation:

- Bewegungsmelder
- Thermometer
- Fensterkontakte
- Windkraftmesser

Beispiele für Aktoren:

- Lampe
- Heizungsregler
- Rasensprinkler
- Rollläden

Kombinierte Geräte:

- Überwachungskamera (Sensor: Bilder, Aktor: Schwenken)
- Moderner Backofen (Sensor: Temperatur, Aktor: Programmauswahl)

Für das Basisszenario benötigen wir eine Infrastruktur, über die wir alle angeschlossenen Sensoren und Aktoren erreichen können. Konkret ausgedrückt müssen wir in der Lage sein, Events von Sensoren zu Empfangen und Commands an Aktoren zu senden. Darüber hinaus sollen möglichst viele Vorgänge automatisiert werden. Das bedeutet, wir benötigen eine Zentrale, um Events zu verarbeiten und gegebenenfalls mit Befehlen an Aktoren auf diese Events zu reagieren. Neben Aktoren und Sensoren existieren noch Clients. Clients sind Geräte, die von einem Benutzer des Systems verwendet werden, um Sensoren abzufragen und Commands an Aktoren zu senden. Clients greifen in der Regel nicht direkt auf Sensoren und Aktoren zu, sondern benutzen zu diesem Zweck die Zentrale. Das Netz, über das alle Sensoren, Aktoren, Clients und die Zentrale miteinander verbunden sind Bezeichnen wir von nun an als *Bus*.

F01: Sensoren Status
Der Status eines Sensors kann über den Bus abgefragt werden. Sensoren können auch selbstständig Events auf den Bus senden.

F02: Steuern von Aktoren
Jede Komponente, die am Bus angeschlossen ist, kann Commands auf den Bus schicken. Aktoren können die Commands über den Bus empfangen und dadurch gesteuert werden.

F03: Persistieren von Events und Commands in der Cloud
Alle Events und Commands, die auf dem Bus transportiert werden, können in der Cloud gespeichert werden. In der Cloud können anhand dieser Daten umfassende Analysen angestellt werden (nicht Teil der Arbeit).

F04: Regeln
Aufgrund von Events sollen je nach Zustand des Gesamtsystems vordefinierte Aktionen ausgelöst werden. Die Aktionen bestehen in der Regel aus Commands, um wiederum Aktoren zu steuern.

F05: Auf Zentrale zugreifen
Ein Client kann auf die Zentrale zugreifen, um den Status des Systems abzufragen und Commands an Aktoren zu schicken. Der Status des Systems kann sowohl Live-Daten von Sensoren, als auch Werte aus der Vergangenheit beinhalten.

2.2.2. Lösungsteil (Demo-System)

L01: Sicherheits-Status abfragen
Der Client soll den Status des Einbrecherschutzes abgefragt werden können.
Status «OK»:

- Fenster ist geschlossen
- Türe ist geschlossen
- keine Bewegung detektiert

Status «NOK»:

- Fenster ist offen
- Türe ist offen
- Bewegung detektiert

L02: Überwachungskamera
Der Client kann die Überwachungskamera ein- bzw. ausschalten und Livebilder anfordern.

L03: Event Kontaktsensor
Der Kontaktsensor hat permanent einen Status. Der Status ist entweder «offen» oder «geschlossen».

L04: Event Bewegungsmelder
Sobald der Bewegungsmelder eine Bewegung registriert, sendet dieser einen Event. Dieser wird nach interner Logik verarbeitet.

L05: Aktor: Lampe
Der Aktor wird via Command angesteuert. Das Licht kann durch eine Regel (Zeit-Mechanismus zur Prävention) oder durch eine Aktion des Clients ein bzw. abgeschaltet werden.

L06: Aktor: Funksteckdose
Die Funksteckdose kann ebenfalls vielseitig eingesetzt werden. Etwas abstrahiert betrachtet, kann jedes Gerät per Remote ein- bzw. ausgeschaltet werden. An dieser Funktsteckdose kann zum Beispiel eine Musikanlage oder ein Fernsehgerät eingeschaltet werden.

L07: NFC Sticker
Die NFC Stickers können sehr vielfältig eingesetzt werden und dienen in erster Linie zur Entlastung des User Interface. Generell wird durch Auflegen eines NFC-fähigen Smartphones (also ein Client) eine Aktion ausgeführt. Was diese Aktion genau beinhaltet ist offen und könnte genauso gut direkt im User Interface benutzt werden. Beispielsweise könnte ein Command zum «scharf stellen» des Sicherheitssystems auf den Bus gesendet werden.

2.3. Marktsituation

Das Thema IoT, insbesondere SmartHome, erlebt derzeit einen regelrechten Boom. Zwar gibt es schon seit Jahrzehnten Lösungen zur Heimautomatisierung, jedoch standen auch einfache Systeme bisher nur wenigen privilegierten zur Verfügung, denn meist musste man sich bereits beim Hausbau auf einen Anbieter festlegen und dessen Produkt- und Preispolitik akzeptieren. Folglich ist es aufwändig, den Anbieter nachträglich zu wechseln oder das System zu erweitern. Als Vorteile von klassischen Gesamtlösungen sind jedoch das einheitliche Bild und die nahtlose bauliche Integration zu nennen.

KNX
KNX ist ein europäischer Standard für kommerzielle Gebäudeautomation. KNX trennt Gerätesteuerung und Stromversorgung in von einander unabhängige Netze. Mit KNX können Schalter und Steuerungen relativ einfach neu zugewiesen werden, ohne dass erneut bauliche Arbeiten vorgenommen. KNX bringt hohe Anschaffungskosten mit sich und eignet sich eher für Neubauten, da eine nachträgliche Installation noch viel teurer wäre.

digitalSTROM
Ursprünglich ein Projekt der ETH, das die nachträgliche Gebäudeautomation ermöglichen

soll. Durch den Einsatz von mit einander kommunizierenden, speziellen Kabelklemmen lassen sich vorallem Beleuchtungskonzepte und Energiesparlösungen recht schnell und einfach realisieren. Das Absetzen von Befehlen oder auslesen von Messdaten aus unterschiedlichen Protokollen ist kaum möglich. Leider sind die vielen benötigten Komponenten immer noch sehr teuer (ca. 100.- CHF für eine Klemme).

RWE Smarthome
Der Energieversorgerkonzern RWE bietet seit einigen Jahren ein nachrüstbares System für die Heimautomation. Der Fokus liegt vorallem auf der Automatisierung von Beleuchtung, Heizung, Rollläden und Stromversorgung. RWE bietet eine Zentrale, Sensoren, Aktoren sowie Software zur Steuerung und Konfiguration. Die Auswahl an Komponenten ist auf das Angebot von RWE beschränkt.

Qivicon
Qivicon ist eine Allianz verschiedener Industriepartner und wurde von der Deutschen Telekom gegründet. Im Unterschied zum RWE Smarthome können also Komponenten aller beteiligten Hersteller miteinander vernetzt werden. Die Bedienung ist etwas einfacher, dafür sind die Konfigurationsmöglichkeiten weniger umfassend als beim RWE Smarthome.

openHAB
OpenHAB steht für *open Home Automation Bus* und ist ein Open Source Projekt zur lokalen Vernetzung von IoT- und SmartHome Zubehör unterschiedlichster Hersteller. OpenHAB fungiert als Zentrale und kann auf beliebigen Windows, Linux oder Mac OS X Geräten installiert werden. Durch die modulare Architektur können jederzeit neue Technologien integriert werden. Eine eigene Modellierungssprache erlaubt nahezu unbegrenzte Konfigurationsmöglichkeiten anhand von Regeln und Abläufen. Allerdings erfordert dies ein wenig technisches Geschick. Basierend auf openHAB 1.x wurde das Eclipse Smarthome Projekt gegründet, welches ein Framework für SmartHome Software darstellt. OpenHAB 2 wiederum baut auf Eclipse Smarthome auf und soll die Konfiguration, insbesondere für technisch weniger versierte Benutzer, erleichtern.

3. Lösungskonzept

Dieses Kapitel beinhaltet die Beschreibung der Architektur und wichtiger Komponenten. Es werden die theoretischen Grundlagen der eingesetzten Technologien und Systeme erläutert und warum diese verwendet werden. Nicht Teil dieses Kapitels sind Erläuterungen von Konfigurationen und Code. Dies wird separat im Kapitel «Umsetzung» behandelt.

3.1. Evaluation der Plattform

Anhand der Marktsituation und den funkionalen Anforderungen haben wir die Vor- und Nachteile der jeweiligen Plattformen miteinander verglichen und dabei darauf geachtet, welche Kriterien für unsere Lösung von Bedeutung sind.

Wichtige Kriterien:

- Nachträgliche Installation möglich
- Beliebige Szenarien realisierbar
- Herstellerunabhängige Komponenten
- Erfüllung der Anforderungen F01 - F05

Vernachlässigbare Kriterien:

- Optisch ansprechende Integration
- Installation ohne Fachkenntnisse

3.1.1. Ergebnis: openHAB

Mit openHAB haben wir eine Plattform gefunden, die allen wichtigen Kriterien entspricht und zudem kostenlos ist. Da wir openHAB sofort auf unseren Notebooks installieren konnten, war es sehr einfach zu beurteilen, ob die Plattform auch in der Praxis unsere Anforderungen erfüllt. Die mitgelieferte Demo-Konfiguration beinhaltete bereits viele anschauliche Beispiele, die später als Vorlage für unsere eigenen Anwendungsfälle dienen können.

Erfüllung der funktionalen Anforderungen
F01 - F02: Über sogenannte Items können Sensoren und Aktoren virtuell und genügend abstrakt definiert werden. Der OSGi EventBus von openHAB ermöglicht den Transport

von Events und Commands zwischen Items und der Zentrale (OpenHAB Runtime). Bindings mappen die Items auf tatsächliche Sensoren und Aktoren.

F03: OpenHAB kann den Verkehr auf dem EventBus über verschiedene Wege auf externen Systemen protokollieren. Zu unserem Zweck eignet sich das MQTT Persistence Modul.

F04: Über die Rule Engine von openHAB können Regeln mit Hilfe einer Java-ähnlichen DSL beschrieben werden. Regeln werden bei gewissen Events auf dem EventBus ausgeführt. Die DLS erlaubt den Zugriff auf den Zustand von Items und kann auch Commands an Items und somit an Aktoren senden.

F05: Ein RESTful API bietet umfassenden Zugriff auf die openHAB Runtime. Über sogenannte Sitemaps können deskriptive User Interfaces automatisch generiert werden.

Nachteile
Ein Nachteil an openHAB ist, dass die Dokumentation grosse Lücken aufweist. Zwar sind die Konzepte leichte verständlich, jedoch fehlen Detailangaben zur DSL und genauen Konfigurationssyntax. Aus diesem Grund müssen oft Beispiele analysiert oder Benutzerforen zu Rate gezogen werden.

3.2. Evaluation der Hardware

Nachdem wir openHAB als Plattform bestimmt haben konnten wir die Hardware für die Sensoren und Aktoren aussuchen. Dafür haben wir uns an den Vorgaben L02 - L06 aus Abschnitt 2.2.2 orientiert. Durch die Vielzahl an Protokollen, die durch openHAB unterstützt werden, hatten wir genügend Auswahl an Hardware von unterschiedlichen Herstellern. OpenHAB selbst läuft auf einem Raspberry Pi B+.

L02: Als Überwachungskamera haben wir die Edimax IC-3115W Netzwerkkamera ausgesucht. Sie ist mit einem Preis von weniger als 50 Euro relativ günstig und über das HTTP Binding von openHAB kompatibel.

L03: Beim Fensterkontaktsensor war uns ein kabelloses Modell wichtig, das unkompliziert montiert werden kann. Aus diesem Grund haben wir uns für den optischen Fensterkontakt HM-Sec-Sco von eQ-3 HomeMatic entschieden. Der Fensterkontakt erfordert jedoch eine Zentrale, die separat bestellt werden musste. Für HomeMatic existiert ein Binding seitens openHAB.

L04: Der Funk Bewegungsmelder HM-Sec-MDIR-2, ebenfalls von eQ-3 HomeMatic, benutzt die gleiche Zentrale wie der Fensterkontaktsensor und ist für den Indoorgebrauch ausgelegt.

L05: Das Philips Hue Lux Starterkit beinhaltet zwei dimmbare LED-Birnen und eine Zentrale, die ans lokale Netzwerk angeschlossen werden muss. Ein openHAB Binding für Philips Hue ist vorhanden.

L06: Da wir durch den Fensterkontakt und den Bewegungsmelder schon eine HomeMatic Zentrale besitzen, liegt es nahe auch die Funksteckdose von diesem Hersteller zu verwenden.

3.3. Einführung openHAB

Das System openHAB wird eingesetzt, um verschiedene Home-Automatisierungssysteme unter einen Hut zu bringen. Um dies zu erreichen müssen Lösungen für die vier Disziplinen Konnektivität, User Interface, Automatisierung und Persistenz gefunden werden. In den nächsten Abschnitten werden diese Disziplinen kurz beschrieben.

3.3.1. Konnektivität

Mit Konnektivität ist gemeint, wie die Sensoren/Aktoren integriert werden können. Es braucht ein Konzept um Protokolle miteinander kompatibel zu machen. Nehmen wir als Beispiel den Use Case *L03: Event Kontaktsensor*. An einem Fenster wird ein Kontaktsensor angebracht, der bei jedem Öffnen oder Schliessen den Status bekannt gibt. OpenHAB muss einerseits das verwendete Protokoll verstehen und zudem die Daten in eine interne, abstrakte Form übersetzen, sodass Herstellerspezifische Details vor dem restlichen System verborgen bleiben. Ein weiters Beispiel ist Use Case *L05: Aktor: Lampe*. Hierbei müssen keine Events gelesen, sondern Commands geschickt werden, da es sich bei der Lampe um einen Aktor handelt. Dazu muss openHAB auch den umgekehrten Fall beherrschen, nämlich aus einer internen Repräsentation des Commands in diejenige des Protokolls der Lampe zu übersetzen und letztlich auch die Lampe erreichen können. Durch Konnektivität ist es also möglich, SmartHome Zubehör von verschiedenen Herstellen in openHAB einzubinden.

3.3.2. User Interface

Nehmen wir an, der Fensterkontakt und die Lampe aus dem vorherigen Abschnitt sind von zwei völlig verschiedenen Herstellern. Der Status des Fensterkontakts soll bei der Verwendung ohne openHAB über eine Website im Browser ausgelesen werden können. Das Steuern der Lampe geschehe mittels einer eigens dafür vorgesehenen App. Dank dem Konzept zur Konnektivität können aber beide Geräte auch über openHAB zugegriffen werden. Einen echten Vorteil hat man dadurch aber nur, wenn es auch ein User Interface dazu gibt. Denn dann hat man alle Geräte in einer Smartphone- oder Web App vereint. OpenHAB benötigt demnach eine Möglichkeit um User Interfaces für verschiedene Clients zu gestalten.

3.3.3. Automatisierung

Durch die Konnektivität und das User Interface kann also SmartHome Zubehör von verschiedenen Herstellern in einer einzigen Anwendung verwendet werden. Das alleine ist schon ein grosser Mehrwert. Doch es fehlt noch etwas der smarte Teil des SmartHomes. Interessant wird es nämlich dann, wenn die verschiedenen Geräte sich gegenseitig beeinflussen sollen. Nehmen wir den Bewegungsmelder aus Use Case L04 hinzu. Sobald er eine Bewegung registriert soll die Lampe eingeschaltet werden. Es sind aber auch wesentlich komplexere Szenarien denkbar. Damit solche automatisierten Vorgänge stattfinden können benötigt openHAB eine Rule Engine. Die Grundlage dazu bildet die interne Repräsentation der Sensoren und Aktoren, die bereits durch die Konzepte zur Konnektivität geschaffen wurde.

3.3.4. Persistenz

Wenn Events gelesen und Commands gesendet werden, dann handelt es sich dabei um Momentaufnahmen. Im User Interface könnte man beobachten, wenn der Status des Fensterkontakts von offen auf zu wechselt. Doch was ist, wenn man wissen möchte, wann das Fenster zuletzt geöffnet wurde? Aus diesem Grund reicht es nicht, Events lediglich zu verarbeiten, sondern sie müssen auch persistiert werden. Zudem können manche Sensoren wie der Fensterkontakt möglicherweise nur immer einen Statuswechsel bekanntgeben, der aktuelle Status kann aber nicht direkt abgefragt werden. Damit trotzdem jederzeit der aktuelle Status bekannt ist, muss openHAB den Status bei jedem Wechsel speichern.

3.4. openHAB Architektur

Im Abschnitt zur Konnektivität haben wir bereits erläutert, welche Anforderungen openHAB erfüllen muss, damit verschiedene Systeme miteinander vernetzt werden können. Die grosse Anzahl an Herstellern und die Vielfalt an Protokollen haben dazu geführt, dass openHAB sehr modular konzipiert wurde. Die Basisinstallation kann zur Laufzeit durch Add-ons erweitert werden. Das hat den Vorteil, dass openHAB selbst recht schlank bleibt und Technologien, die gar nicht eingesetzt werden, nicht im Weg sind. Ausserdem ist dadurch das spätere Einbinden weiterer Plattformen sehr einfach machbar. Technisch wurde diese modulare Architektur mit Hilfe der OSGi-Plattform umgesetzt. Die Implementierung von Protokollen geschieht innerhalb von OSGi Service Bundles, die bei openHAB Bindings genannt werden. Abbildung 4 zeigt einen Überblick der Architektur:

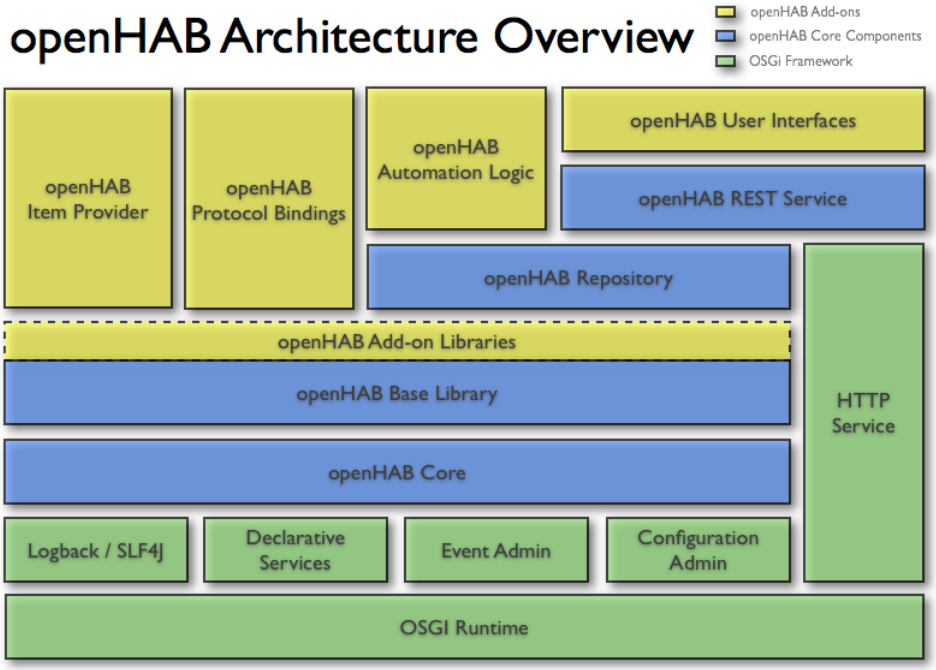


Abbildung 1.: openHAB Architektur

3.4.1. Event Bus

Der Basisservice von openHAB stellt der Event Bus dar. Über diesen Bus werden Events zwischen den verschiedenen OSGi Bundles gesendet. Die Events sind entweder Commands, welche eine Aktion ausführen, oder Status-Updates, welche Zustandsänderungen der

Sensoren/Aktoren beinhalten.
Durch den Einsatz dieses Event Bus wird die Kopplung reduziert und Add-ons können somit einfach ausgetauscht werden.
Intern wurde der Event Bus mit Hilfe des OSGi Event Admin umgesetzt. Durch den Event Admin wird es möglich, dass sich weitere OSGi Bundles (also openHAB Add-ons) im Event Bus einklinken können.

3.4.2. Bindings

Bindings sind Verbindungen zwischen openHAB und den externen Systemen und bilden die Grundlage zur Konnektivität. Dadurch muss für jede Technologie ein eigenes Binding geschrieben werden. Für viele Technologien sind Bindings vorhanden, die einzeln heruntergeladen und als «Add-on» installiert werden können. Falls eine Technologie noch nicht unterstützt wird, kann man das Binding dazu selbst programmieren. Die wesentliche Aufgabe besteht darin, sich einerseits mit dem externen Gerät zu verbinden, auf dem Event Bus zu lauschen und die ausgetauschten Daten miteinander kompatibel zu machen. Alle momentan verfügbare Bindings sind unter folgendem Link zu finden: `https://github.com/openhab/openhab/wiki/Bindings`

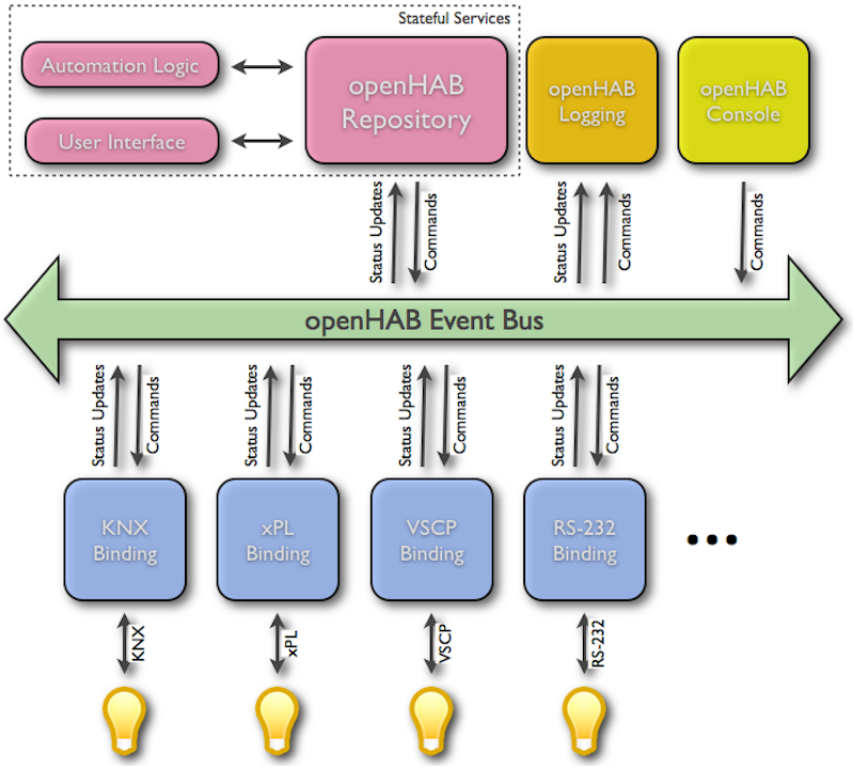


Abbildung 2.: Event Bus als Schnittstelle für Bindings

3.4.3. Items

Wenn so viele unterschiedliche Technologien unterstützt und integriert werden sollen, dann stellt sich die Frage nach dem gemeinsamen Nenner bzw. einer einheitlichen internen Repräsentation. Aus diesem Grund wurden «Items» eingeführt, die zentrale Entität im openHAB Domainmodell. Alle Bindings implementieren ein Mapping zwischen den Daten des Sensors/Aktors und einem zugehörigen Item. Ein Item besteht aus:

- Typ
- Name
- Formatierung
- Icon
- Gruppe
- Bindingparameter

Die Eigenschaften Typ und Name sind zwingend, die Anderen sind optional. Der Typ ist auf eine vorgegebene Auswahl beschränkt, der Name dient als Identifier und muss eindeutig sein.

Je nach gewähltem Typ können Items die unterschiedlichsten Dinge repräsentieren. Ein sehr häufig verwendeter Typ ist das SwitchItem mit den beiden möglichen Werten ON und OFF. Dieses Item eignet sich aufgrund seines Wertebereichs hervorragend als Boolean. Ein SwitchItem kann stellvertretend für etwas reales, wie eine Lampe, oder für etwas abstraktes wie die Anwesenheit eines Bewohners stehen. Es ist wichtig zu verstehen, dass ein Item rein virtuell ist. Die Brücke zur echten Hardware wird erst über Bindingparameter geschlagen, denn dann verknüpft openHAB das Item mit dem entsprechenden Binding. Da die Bindingparameter eines Items aber optional sind macht openHAB keinen Unterschied zwischen virtuellen und realen Items.

3.4.4. Item Repository

Der Zustand eines Items muss die Ausführungsdauer einer openHAB Instanz überleben können, beispielsweise nach einem Neustart. Der Zustand muss ausserdem jederzeit abfragbar sein, egal ob sich der Zustand des Items gerade erst durch ein Event geändert hat oder schon über mehrere Startvorgänge hinweg genau so exisitiert. Diese Verantwortung übernimmt das Item Repository, welches permanent den Event Bus auf Status-Updates abhört und die Änderungen persistiert.

3.4.5. Rules

Rules sind die Werkzeuge zur Automatisierung. Dank einer Java-ähnlichen DSL können Aktionen ausgelöst werden, wenn gewisse Bedingungen erfüllt wurden. Das können einerseits Zustandsänderungen von Items, aber auch Systemevents oder Events von selbst angelegten Timern sein.

3.4.6. Sitemaps

Sitemaps sind Konfigurationsfiles zur deklarativen Definition von User Interfaces. Sitemaps werden ebenfalls in einer Xtext basierten DSL geschrieben und haben eine baumartige Struktur. OpenHAB liest das Konfigurationsfile und stellt die Sitemap über eine REST API zur Verfügung. Eine mitgelieferte Webapplikation stellt das User Interface

anschliessend im Browser dar. Layouttechnisch orientiert sich die Webapplikation am iOS 6 Design mit Listendarstellung und Drill-Down Prinzip. Die nativen iOS und Android Apps von openHAB verfolgen das selbe Bedienkonzept.
Die verschiedenen Elemente, die in Sitemaps verfügbar sind, können Items direkt über deren Name referenzieren. Einige Elemente sind dynamisch und erlauben eine werteabhängige Darstellung.

3.5. Anbindung Cloud

In den funktionalen Anforderungen zum Basisszenario wurde festgelegt, dass Geschehnisse auf dem lokalen Bus (openHAB) an die Cloud gesendet werden müssen. Als Cloud-Plattform wurde Microsoft Azure bereits zu Beginn der Arbeit definiert. Damit die Daten von openHAB in die Cloud gelangen gibt es im Wesentlichen zwei Möglichkeiten: Entweder man entwickelt selbst ein Plugin für openHAB oder man verwendet das openHAB MQTT Persistence Plugin. MQTT ist ein Publish/Subscribe Protokoll und besonders gut für IOT Systeme mit langsamer oder instabiler Internetverbindung geeignet. Wir haben uns für die Variante mit MQTT entschieden. OpenHAB fungiert als MQTT-Client und kann so konfiguriert werden, dass Events auf dem Bus automatisch an einen MQTT-Broker gesendet werden, der von uns auf Microsoft Azure installiert wurde. Um die Daten zu verarbeiten muss ein MQTT-Client auf Microsoft Azure laufen und die Nachrichten vom Broker entgegennehmen. Danach steht uns frei, wie die Daten verarbeitet werden.

3.5.1. MQTT Broker Evaluation

Für die Evaluation des Brokers haben wir eine List mit unseren Anforderungen erstellt:

1. Open Source/Freeware
2. SSL TLS Verschlüsselung
3. Benutzername & Passwort Authentifizierung
4. High throughput, Low latency
5. Cloud Ready
6. Einfache Installation
7. Qualit of Service Level: Exactly once
8. Last Will unterstützung (Message, die gesendet wird, wenn der Client die Verbindung schliesst)

HiveMQ (`http://www.hivemq.com`)
HiveMQ ist ein proprietärer MQTT-Broker und erfüllt alle unsere Kriterien bis auf das erste Kriterium. Jeder Gebrauch des Brokers muss bezahlt werden, siehe: `http://www.hivemq.com/pricing/`.

Mosquitto (`http://eclipse.org/mosquitto/`)
Mosquitto ist ein Open Source Broker, dessen Projekt von Roger Light im Jahr 2010 auf die Beine gestellt wurde. Seit der Version 1.4 läuft das Projekt unter der Eclipse Foundation.
Die aktuelle Implementation benötigt lediglich 120kB Speicherplatz und 3MB RAM bei 1000 verbundenen Clients. Ein Belastungstest von 100'000 Clients erzielte erfolgreiche und zufriedenstellende Resultate. Alle anderen Kriterien werden ebenfalls erfüllt.

CloudMQTT
CloudMQTT unterscheidet sich von den anderen Brokern, da dieser nicht selbst betrieben werden kann. Das bedeutet, man erstellt bei CloudMQTT eine Instanz und kann denn neu erstellten Broke über verschiedene APIs ansteuern.
Der Anbieter stellt verschiedene Preispläne zur Verfügung (`http://www.cloudmqtt.com/plans.html`). Ab 10 Verbindungen bzw. 10Kbit/s Bandbreite muss für die Leistung bezahlt werden.
Ob TLS SSL Verschlüsselung unterstützt wird, ist in der spärlichen Dokumentation nicht ersichtlich.

Fazit
Aufgrund der gesammelten Daten der drei Produkte wurde entschieden, Mosquitto einzusetzen. Es ist das einzige Produkt, welches alle unsere Kriterien erfüllt. Da dieses Projekt durch die Eclipse Foundation unterstützt wird, besteht die Hoffung, dass die Zusammenarbeit mit openHAB unterstützt bzw. miteinbezogen wurde.

3.5.2. Client-Library

In unserem Systemaufbau wird es zwei MQTT-Clients geben. Einerseits durch das openHAB-Binding, da die Events auf dem EventBus über MQTT in die Azure Cloud gesendet werden soll. Andererseits wird in der Cloud eine Worker Role diese Events konsumieren und persistieren.
Das Binding von openHAB besteht bereits, daher muss nur noch eine Client-Implementation für C# gesuchtwerden. Nach kurzer Rechereche scheint die «M2Mqtt» Library sehr ver-

breitet zu sein.
Durch aufsetzen eines Prototyps konnte die benötigte Funktionalität erfasst werden. Sie erfüllt alle Kriterien, die auch an den Broker gestellt wurden.

3.6. Notification

Damit ein Sicherheitssystem Sinn macht, muss der Besitzer benachrichtigt werden, sobald sich ein Zustand der Sensoren ändert. Genauer gesagt soll der Benutzer informiert werden, wenn der Bewegungsmelder oder der Fensterkontakt im Alarm-Modus eine Veränderung registriert. Ansonsten müsste der Benutzer permanent die Status im Android-App überwachen. Für die Umsetzung bietet sich der Google Cloud Message Dienst (GCM) an, der auch von Microsoft empfohlen wird. Da nur die Android-Platform unterstützt werden muss, wird auf den Einsatz des Azure Notification Hubs verzichtet und die direkte Verbindung zum GCM-Dienst gesucht.

3.7. Sicherheit

In unserem Szenario, Einbruchschutz, ist das Thema Sicherheit sehr wichtig. Daher lohnt es sich dazu Gedanken zu machen. Man kann das System grob in drei Bereiche einteilen, an denen ein Angriff angesetzt werden kann:

- Cloud
- Android App
- Systemaufbau mit Sensoren/Aktoren
- Kommunikation der HomeMatic Komponenten

3.7.1. Cloud

Die Angriffsfläche in der Cloud ist ziemlich eingeschränkt. Sicherheitsrelevante Aspekte sind der Broker und die persistierten Daten im Storage. Zugriff auf diese Ressourcen erhält man über verschiedene Wege. Einerseits über das Management Web-Interface von Microsoft Azure oder über ein Connection-String. Durch die Verwendung eines Connection Strings für den Verbindungsaufbau zum Storage kein Benutzername bzw. Passwort mitgegeben werden.

Zugriff auf diese Ressourcen haben nur authorisierte Personen die ein entsprechendes Microsoft Konto besitzen und auf die Subscription zugelassen werden. Eine vernünftige

Attack kann daher nur durch das Hijacken eines Accounts gefahren werden. Als Gegenmassnahme müssen ausreichend sichere Passwörter auf den persönlichen Konten gesetzt werden. Das liegt in der Verantwortung der einzelnen Personen, die auf diese Subscription Zugriff besitzen.

Auf den Storage kann ausser der Management-Oberfläche auch über ein Connection-String zugegriffen werden. Dieser String kann nur über die Management-Oberfläche eingesehen werden und ist daher sicher versteckt. Weiter könnte der Connection String aus dem Source-Code der Worker Role gelesen werden, da er dort für den Verbindungsaufbau zum Storage verwendet wird. Diese Worker Role und deren Source-Code befindet sich innerhalb der Cloud und ist somit auch gegen ein Eindringen gesichert.

3.7.2. Android App

Über das Android App kann das ganze System manipuliert werden. Daher stellt dies ein grosses Sicherheits-Risiko dar. Gesichtert wird das App durch Eingabe von Benutzerdaten, um auf die RESTful-API des openHAB zuzugreifen.
Auch hier wird die Person, die das App verwendet zur Verantwortung gezogen, um das Smartphone durch ein sicheres Passwort zu schützen.

3.7.3. Systemaufbau mit Sensoren/Aktoren

Wenn sich die angreifende Person bereits im Haus befindet,hat sie die entsprechenden Alarme bereits ausgelöst und wurde durch die Webcam aufgenommen.
Für dieses Szenario ist das grösste Risiko, dass die Person die Videoüberwachung bemerkt und aus diesem Grund das Raspberry Pi zerstört bzw. mitnimmt. Um die aufgezeichneten Daten zu sichern, werden die von der Kamera erzeugten Bilder über MQTT im Cloud-Storage abgelegt. Auch wenn das Raspberry Pi zerstört wird, sind die Bilder nun gesichert und können im Nachhinein vom Storage heruntergeladen werden.

3.7.4. HomeMatic Kommunikation

Da die HomeMatic Komponenten untereinander drahtlos kommunizieren besteht hier ebenfalls die Gefahr einer Attacke. HomeMatic setzt für die Übertragung auf den Funkstandard BidCos. Diese Standard sieht vor, dass das 868 MHz Band verwendet wird. Im

Vergleich zu 2.4 GHz besitzt man so eine grösse Reichweite und die Gefahr von Interferenz durch andere Funkgeräte ist sehr minim.
Bezüglich Sicherheit sieht der Standard lediglich vor, die übertragenen Daten durch XOR-Operationen unleserlich zu machen. HomeMatic hat daher bei der Implentierung dieses Stanards die Verschlüsselung durch AES-128 verstärkt. Der zur Verschlüsselung verwendete Key ist in der Zentrale hinterlegt und kann bei Bedarf geändert werden.

3.7.5. Fazit

Wie aus den Ausführungen zur Sicherheit ersichtlich ist, können alle technischen Sicherheits-Risiken abgedeckt werden. Die grösste Schwachstellt ist jeweils der Mensch, der durch Social-Engineering-Techniken manipuliert werden kann und so Zugriff auf das Android App gewährt.

3.8. Allgemeine Systemsicht

Anhand der Problembeschreibung wurde ein Plan erarbeitet, der das ganze System verständlich beschreibt. Abbildung 3 stellt die wichtigsten Komponenten und Entitäten aus der Problemdomäne in gegenseitiger Beziehung dar.

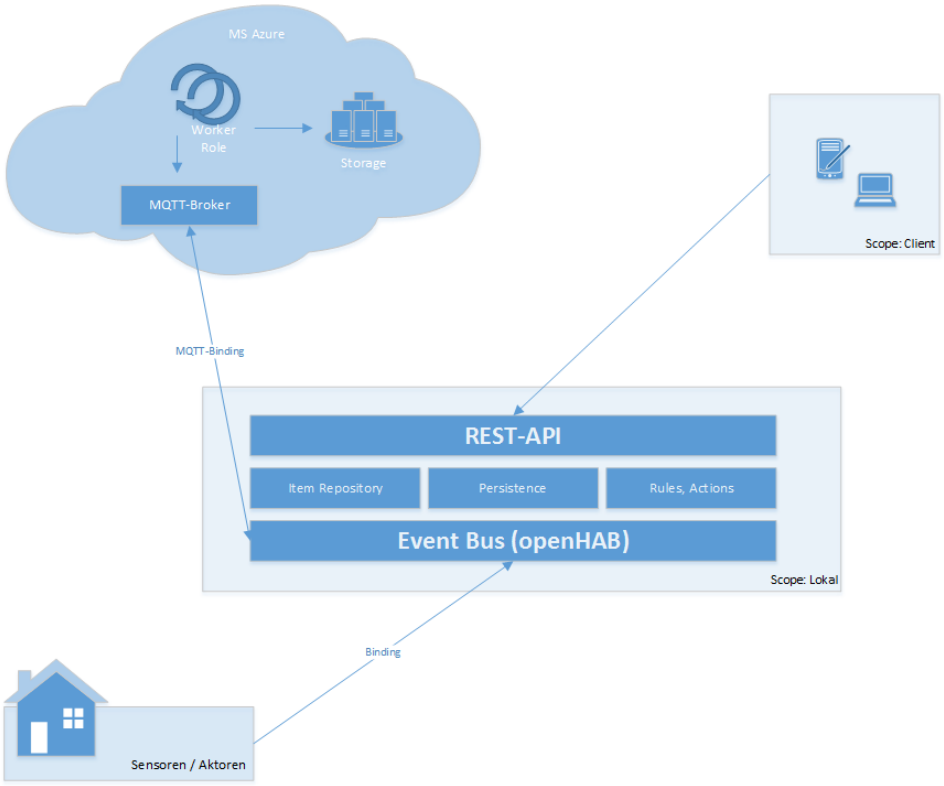


Abbildung 3.: Systemübersicht

4. Umsetzung

4.1. Technologie und Plattform

In der Problembeschreibung zu dieser Arbeit wurden die Anforderungen an eine SmartHome Lösung diskutiert. In der anschliessenden Marktanalyse wurde openHAB als Grundlage zur Umsetzung unseres Projekts evaluiert. OpenHAB erfüllt die geforderten Kritierien, wie Herstellerunabhängigkeit, Installierbarkeit und Flexibilität. Cloudseitig wird MS Azure Cloud zur Persistierung von Events verwendet.

4.2. Einführung openHAB

Das System openHAB wird eingesetzt, um verschiedene Home-Automatisierungssysteme unter einen Hut zu bringen. Um dies zu erreichen müssen Lösungen für die vier Disziplinen Konnektivität, User Interface, Automatisierung und Persistenz gefunden werden. In den nächsten Abschnitten werden diese Disziplinen kurz beschrieben.

4.2.1. Konnektivität

Mit Konnektivität ist gemeint, wie die Sensoren/Aktoren integriert werden können. Es braucht ein Konzept um Protokolle miteinander kompatibel zu machen. Nehmen wir als Beispiel den Use Case *L03: Event Kontaktsensor*. An einem Fenster wird ein Kontaktsensor angebracht, der bei jedem Öffnen oder Schliessen den Status bekannt gibt. OpenHAB muss einerseits das verwendete Protokoll verstehen und zudem die Daten in eine interne, abstrakte Form übersetzen, sodass Herstellerspezifische Details vor dem restlichen System verborgen bleiben. Ein weiters Beispiel ist Use Case *L05: Aktor: Lampe*. Hierbei müssen keine Events gelesen, sondern Commands geschickt werden, da es sich bei der Lampe um einen Aktor handelt. Dazu muss openHAB auch den umgekehrten Fall beherrschen, nämlich aus einer internen Repräsentation des Commands in diejenige des Protokolls der Lampe zu übersetzen und letztlich auch die Lampe erreichen können. Durch Konnektivität ist es also möglich, SmartHome Zubehör von verschiedenen Herstellen in openHAB einzubinden.

4.2.2. User Interface

Nehmen wir an, der Fensterkontakt und die Lampe aus dem vorherigen Abschnitt sind von zwei völlig verschiedenen Herstellern. Der Status des Fensterkontakts soll bei der Verwendung ohne openHAB über eine Website im Browser ausgelesen werden können. Das Steuern der Lampe geschehe mittels einer eigens dafür vorgesehenen App. Dank dem Konzept zur Konnektivität können aber beide Geräte auch über openHAB zugegriffen werden. Einen echten Vorteil hat man dadurch aber nur, wenn es auch ein User Interface dazu gibt. Denn dann hat man alle Geräte in einer Smartphone- oder Web App vereint. OpenHAB benötigt demnach eine Möglichkeit um User Interfaces für verschiedene Clients zu gestalten.

4.2.3. Automatisierung

Durch die Konnektivität und das User Interface kann also SmartHome Zubehör von verschiedenen Herstellern in einer einzigen Anwendung verwendet werden. Das alleine ist schon ein grosser Mehrwert. Doch es fehlt noch etwas der smarte Teil des SmartHomes. Interessant wird es nämlich dann, wenn die verschiedenen Geräte sich gegenseitig beeinflussen sollen. Nehmen wir den Bewegungsmelder aus Use Case L04 hinzu. Sobald er eine Bewegung registriert soll die Lampe eingeschaltet werden. Es sind aber

auch wesentlich komplexere Szenarien denkbar. Damit solche automatisierten Vorgänge stattfinden können benötigt openHAB eine Rule Engine. Die Grundlage dazu bildet die interne Repräsentation der Sensoren und Aktoren, die bereits durch die Konzepte zur Konnektivität geschaffen wurde.

4.2.4. Persistenz

Wenn Events gelesen und Commands gesendet werden, dann handelt es sich dabei um Momentaufnahmen. Im User Interface könnte man beobachten, wenn der Status des Fensterkontakts von offen auf zu wechselt. Doch was ist, wenn man wissen möchte, wann das Fenster zuletzt geöffnet wurde? Aus diesem Grund reicht es nicht, Events lediglich zu verarbeiten, sondern sie müssen auch persistiert werden. Zudem können manche Sensoren wie der Fensterkontakt möglicherweise nur immer einen Statuswechsel bekanntgeben, der aktuelle Status kann aber nicht direkt abgefragt werden. Damit trotzdem jederzeit der aktuelle Status bekannt ist, muss openHAB den Status bei jedem Wechsel speichern.

4.3. openHAB Architektur

Im Abschnitt zur Konnektivität haben wir bereits erläutert, welche Anforderungen openHAB erfüllen muss, damit verschiedene Systeme miteinander vernetzt werden können. Die grosse Anzahl an Herstellern und die Vielfalt an Protokollen haben dazu geführt, dass openHAB sehr modular konzipiert wurde. Die Basisinstallation kann zur Laufzeit durch Add-ons erweitert werden. Das hat den Vorteil, dass openHAB selbst recht schlank bleibt und Technologien, die gar nicht eingesetzt werden, nicht im Weg sind. Ausserdem ist dadurch das spätere Einbinden weiterer Plattformen sehr einfach machbar. Technisch wurde diese modulare Architektur mit Hilfe der OSGi-Plattform umgesetzt. Die Implementierung von Protokollen geschieht innerhalb von OSGi Service Bundles, die bei openHAB Bindings genannt werden. Abbildung 4 zeigt einen Überblick der Architektur:

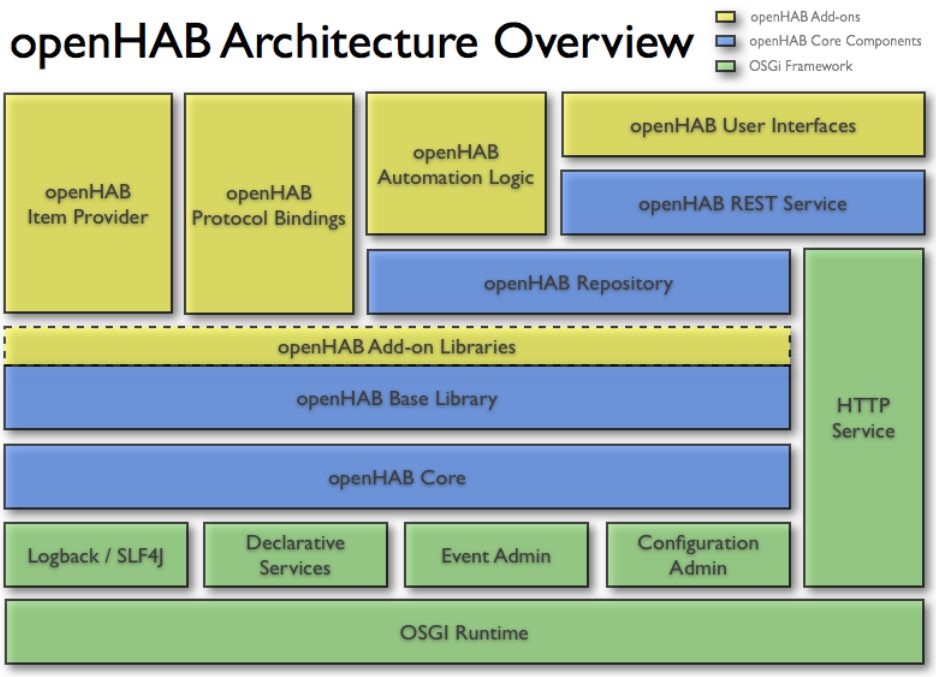


Abbildung 4.: openHAB Architektur

4.3.1. Event Bus

Der Basisservice von openHAB stellt der Event Bus dar. Über diesen Bus werden Events zwischen den verschiedenen OSGi Bundles gesendet. Die Events sind entweder Commands, welche eine Aktion ausführen, oder Status-Updates, welche Zustandsänderungen der Sensoren/Aktoren beinhalten.
Durch den Einsatz dieses Event Bus wird die Kopplung reduziert und Add-ons können somit einfach ausgetauscht werden.
Intern wurde der Event Bus mit Hilfe des OSGi Event Admin umgesetzt. Durch den Event Admin wird es möglich, dass sich weitere OSGi Bundles (also openHAB Add-ons) im Event Bus einklinken können.

4.3.2. Bindings

Bindings sind Verbindungen zwischen openHAB und den externen Systemen und bilden die Grundlage zur Konnektivität. Dadurch muss für jede Technologie ein eigenes Binding geschrieben werden. Für viele Technologien sind Bindings vorhanden, die einzeln herun-

tergeladen und als «Add-on» installiert werden können. Falls eine Technologie noch nicht unterstützt wird, kann man das Binding dazu selbst programmieren. Die wesentliche Aufgabe besteht darin, sich einerseits mit dem externen Gerät zu verbinden, auf dem Event Bus zu lauschen und die ausgetauschten Daten miteinander kompatibel zu machen. Alle momentan verfügbare Bindings sind unter folgendem Link zu finden: `https://github.com/openhab/openhab/wiki/Bindings`

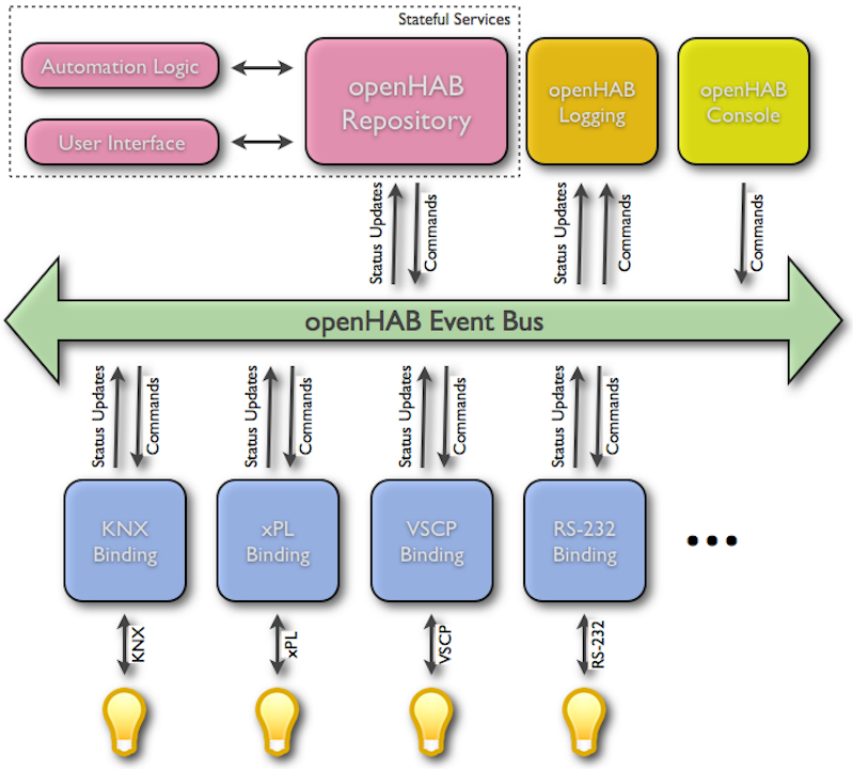


Abbildung 5.: Event Bus als Schnittstelle für Bindings

4.3.3. Items

Wenn so viele unterschiedliche Technologien unterstützt und integriert werden sollen, dann stellt sich die Frage nach dem gemeinsamen Nenner bzw. einer einheitlichen internen Repräsentation. Aus diesem Grund wurden «Items» eingeführt, die zentrale Entität im openHAB Domainmodell. Alle Bindings implementieren ein Mapping zwischen den Daten des Sensors/Aktors und einem zugehörigen Item. Ein Item besteht aus:

- Typ

- Name
- Formatierung
- Icon
- Gruppe
- Bindingparameter

Die Eigenschaften Typ und Name sind zwingend, die Anderen sind optional. Der Typ ist auf eine vorgegebene Auswahl beschränkt, der Name dient als Identifier und muss eindeutig sein.

Je nach gewähltem Typ können Items die unterschiedlichsten Dinge repräsentieren. Ein sehr häufig verwendeter Typ ist das SwitchItem mit den beiden möglichen Werten ON und OFF. Dieses Item eignet sich aufgrund seines Wertebereichs hervorragend als Boolean. Ein SwitchItem kann stellvertretend für etwas reales, wie eine Lampe, oder für etwas abstraktes wie die Anwesenheit eines Bewohners stehen. Es ist wichtig zu verstehen, dass ein Item rein virtuell ist. Die Brücke zur echten Hardware wird erst über Bindingparameter geschlagen, denn dann verknüpft openHAB das Item mit dem entsprechenden Binding. Da die Bindingparameter eines Items aber optional sind macht openHAB keinen Unterschied zwischen virtuellen und realen Items.

4.3.4. Item Repository

Der Zustand eines Items muss die Ausführungsdauer einer openHAB Instanz überleben können, beispielsweise nach einem Neustart. Der Zustand muss ausserdem jederzeit abfragbar sein, egal ob sich der Zustand des Items gerade erst durch ein Event geändert hat oder schon über mehrere Startvorgänge hinweg genau so exisitiert. Diese Verantwortung übernimmt das Item Repository, welches permanent den Event Bus auf Status-Updates abhört und die Änderungen persistiert.

4.3.5. Rules

Rules sind die Werkzeuge zur Automatisierung. Dank einer Java-ähnlichen DSL können Aktionen ausgelöst werden, wenn gewisse Bedingungen erfüllt wurden. Das können einerseits Zustandsänderungen von Items, aber auch Systemevents oder Events von selbst angelegten Timern sein.

4.3.6. Sitemaps

Sitemaps sind Konfigurationsfiles zur deklarativen Definition von User Interfaces. Sitemaps werden ebenfalls in einer Xtext basierten DSL geschrieben und haben eine baumartige Struktur. OpenHAB liest das Konfigurationsfile und stellt die Sitemap über eine REST API zur Verfügung. Eine mitgelieferte Webapplikation stellt das User Interface anschliessend im Browser dar. Layouttechnisch orientiert sich die Webapplikation am iOS 6 Design mit Listendarstellung und Drill-Down Prinzip. Die nativen iOS und Android Apps von openHAB verfolgen das selbe Bedienkonzept.
Die verschiedenen Elemente, die in Sitemaps verfügbar sind, können Items direkt über deren Name referenzieren. Einige Elemente sind dynamisch und erlauben eine werteabhängige Darstellung.

4.4. openHAB Konfiguration

Items, Rules, Sitemaps und Persitence Strategies werden in openHAB in einer eigens dafür entwickelten DSL beschrieben. Für alle diese Bereiche werden Konfigurationsdateien im entsprechenden Unterverzeichnis des openHAB Konfigurationsordner abgelegt. Änderungen an den Konfigurationsdateien werden von openHAB zur Laufzeit sofort erkannt und berücksichtigt. Nebst den genannten Bereichen existiert noch eine globale Konfigurationsdatei um beispielsweise IP-Adressen für Bindings einzutragen. Folgende Dateien sind für diese Arbeit relevant:

- /configurations/*.cfg
- /configurations/items/*.items
- /configurations/rules/*.rules
- /configurations/sitemaps/*.sitemap
- /configurations/persistence/*.persist
- /configurations/transform/*.map

Anhand diesen Konfigurationen werden die User Cases modelliert.

4.4.1. Konfiguration Überwachungskamera

Für die Überwachungskamera existiert kein entsprechendes Item. Stattdessen wird in der Sitemap ein Image-Element angezeigt, dessen URL auf das HTTP Snapshot Interface der Netzwerkkamera verweist.

Das Bild wird dank dem Attribut `visibility` immer dann angezeigt, wenn das Item `Alarm_activated` den Wert `ON` hat. Das `refresh` Interval bewirkt einen Reload des Bildes alle 200 Millisekunden.

```
1 [...]
2 Image url="http://192.168.1.15/snapshot.jpg" refresh=200
3 visibility=[Alarm_activated==ON]
4 [...]
```

Listing 1: demo.sitemap - Webcam Bild

4.4.2. Konfiguration Kontaktsensor

Der Kontaktsensor ist ein Use Case der in openHAB sehr oft vorkommt. Deshalb existiert ein Item Type «Contact» mit den beiden möglichen Werten OPEN und CLOSED. Das Homematic Binding liefert aber die Werte **`true`** oder **`false`** und kann deshalb nicht an ein Contact Item gebunden werden. Als Alternative haben wir ein String Item verwendet und mit Hilfe einer Transformation-Map **`false`** mit «geschlossen» und **`true`** mit «offen» übersetzt. Das Icon <contact> kann trotzdem noch verwendet werden. Allerdings wird nun nicht mehr automatisch das Icon für das offene bzw. geschlossene Fenster angezeigt, da openHAB den Basisnamen des Icons mit dem State konkateniert. Wir haben dementsprechend die Icons kopiert und ein File contact-false.png und contact-true.png im Ressourcen-Ordner hinterlegt.

Auf der letzten Zeile werden die Bindingparameter für Homematic angegeben. `address=LEQ1469091` bezieht sich auf die Seriennummer des Kontaktsensors und `parameter=STATE` gibt an, welche Eigenschaft gebunden werden soll. `channel=1` ist ein Defaultwert.

```
1 [...]
2 String Window_Bedroom "Schlafzimmerfenster [MAP(window.map):%s]"
3 <contact> (OV_Windows, Windows)
4 {homematic="address=LEQ1469091,channel=1,parameter=STATE"}
5 [...]
```

Listing 2: demo.items - Kontaktsensor

In der Sitemap wird das Item in einem Text-Element referenziert. Weitere Attribute sind nicht notwendig.

```
1 [...]
2 Text item=Window_Bedroom
3 [...]
```

Listing 3: demo.sitemap - Kontaktsensor

4.4.3. Konfiguration Bewegungsmelder

Der Bewegungsmelder ist ebenfalls von Homematic und wird in den Bindingparametern gleich referenziert wie beim Kontaktsensor. Der Bindingparameter `MOTION` erhält einen boolschen Wert und muss mit Hilfe der Map `motion.map` in einen benutzerfreundlichen String transformiert werden. Der Parameter `ERROR` dient dem Sabotageschutz.

Leider muss bei Homematic jeder Parameter einem eigenen Item zugeordnet werden. Deshalb haben wir ein drittes String-Item `Motion_Summary` definiert, indem die Werte der beiden anderen Items zusammengefasst werden.

```
1 [...]
2 String Motion_Livingroom "Bewegungsmelder [MAP(motion.map):%s]"
3 {homematic="address=LEQ0797607, channel=1, parameter=MOTION"}
4
5 String Motion_Livingroom_Sabotage "Sabotage"
6 {homematic="address=LEQ0797607, channel=1, parameter=ERROR"}
7
8 String Motion_Summary "Bewegungsmelder [%s]"
9 [...]
```

Listing 4: demo.items - Bewegungsmelder Items

Die Ermittlung und Zuweisung des korrekten Wertes an das Item `Motion_Summary` geschieht mit Hilfe der Rule `Motion Aggregator`.

```
1 rule "Motion Aggregator"
```

```
2     when Item Motion_Livingroom received update or
3          Item Motion_Livingroom_Sabotage received update
4     then
5     if(Motion_Livingroom.state == "true" &&
6        Motion_Livingroom_Sabotage.state == "NO_ERROR") {
7        postUpdate(Motion_Summary, "Bewegung erkannt")
8     } else if(Motion_Livingroom.state == "false" &&
9              Motion_Livingroom_Sabotage.state == "NO_ERROR") {
10       postUpdate(Motion_Summary, "ruhig")
11    } else if(Motion_Livingroom.state == "true" &&
12             Motion_Livingroom_Sabotage.state == "SABOTAGE") {
13       postUpdate(Motion_Summary, "Bewegung und Sabotage!")
14    } else if(Motion_Livingroom.state == "false" &&
15             Motion_Livingroom_Sabotage.state == "SABOTAGE")  {
16       postUpdate(Motion_Summary, "Sabotage")
17    } else {
18       postUpdate(Motion_Summary, "---")
19    }
20 end
```

Listing 5: demo.rules - Rule «Motion Aggregator»

4.5. Deploymentübersicht

4.5.1. Binding Azure

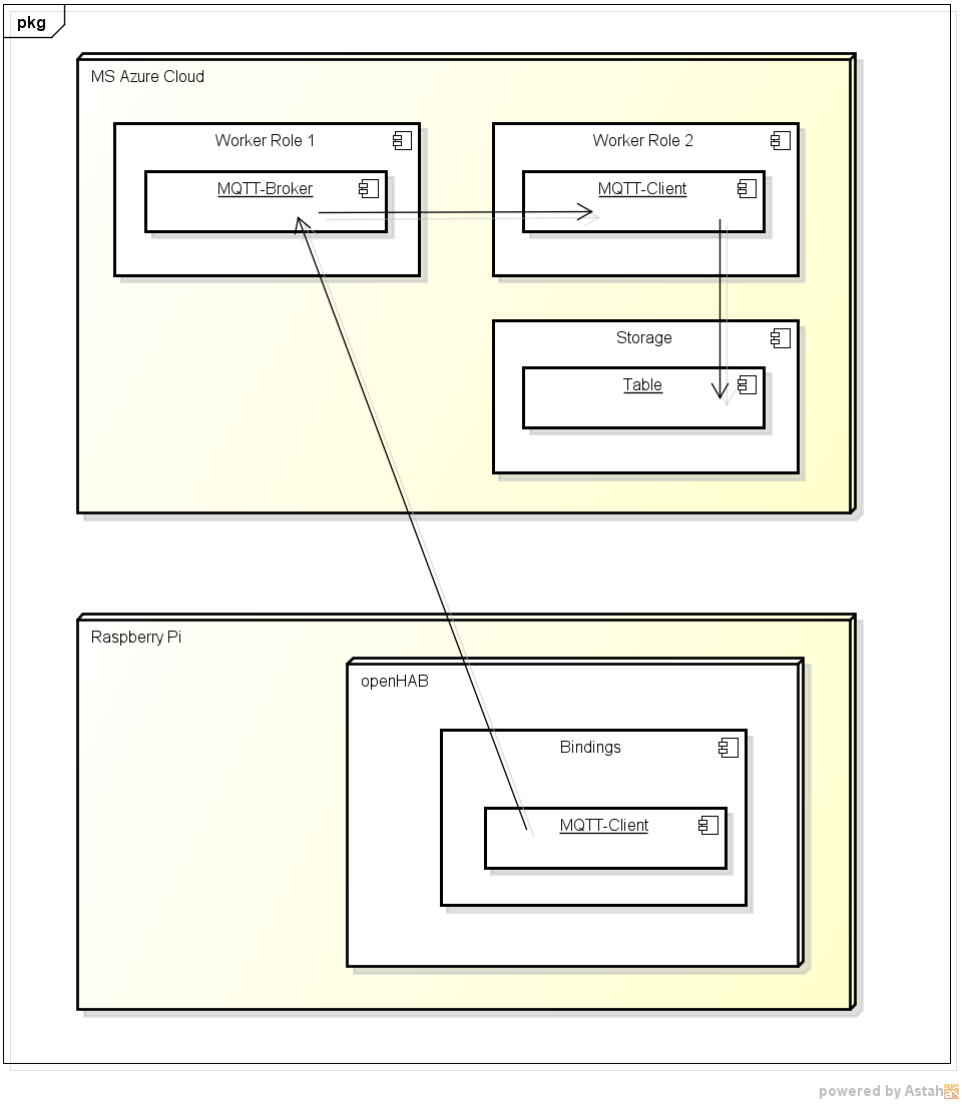


Abbildung 6.: Binding Azure Cloud

4.6. MQTT

MQTT steht für «Message Queue Telemetry Transport» und ist ein Nachrichten-Protokoll, das speziell für IOT-Anwendungen konzipiert wurde. Es setzt auf dem TCP/IP Stack

auf und wird für den Nachrichtenaustausch zwischen verschiedenen, verteilten Maschinen verwendet.
Das Protokoll wurde speziell für Systeme designt, die über wenig Speicherplatz und kleiner Netzwerk-Bandbreite verfügen, was bei IOT-Anwendungen meist der Fall ist.

4.6.1. Funktionsweise

MQTT folgt dem Prinzip «Publish/Subscribe», sprich Clients können bestimmte Topics abonnieren. Wenn Messages auf dieses Topic gesendet werden, leitet der Broker diese an alle interessierten Clients weiter.
In Bezug auf erstellen von Topics agiert der Broker passiv. Das bedeutet, Clients können sich auf beliebigen, selber defnierte, Topics registrieren. Wenn aber niemand auf dieses Topic publiziert, wird der Client nie eine Message erhalten.

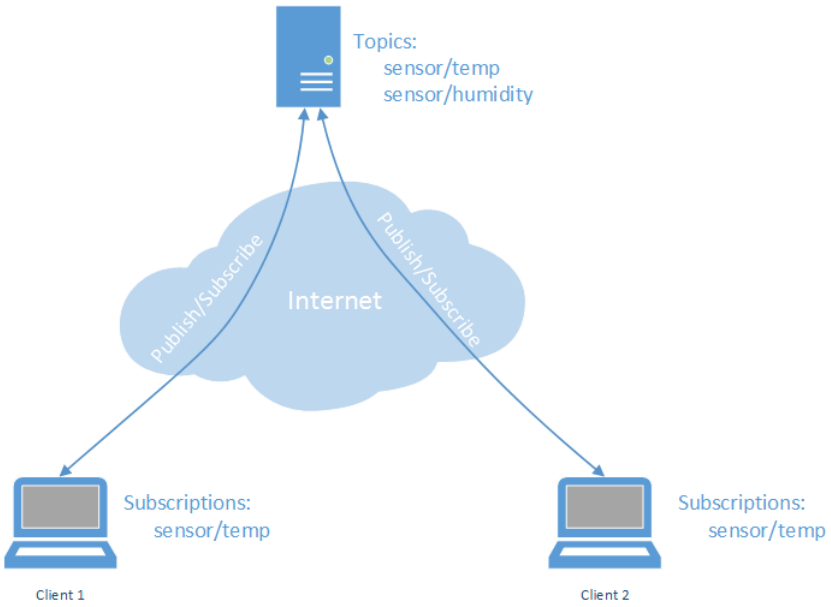


Abbildung 7.: Funktionsweise MQTT

4.6.2. Broker

Zertifizierungsstelle/Server-Zertifikat
Da die MQTT-Verbindung verschlüsselt werden soll, müssen verschiedene Zertifikate erstellt werden. Das erstellte Server-Zertifikat muss von einer CA (Certification Authority)

signiert werden. Da ein gültiges Zertifikat nicht entgeltlich erworben werden möchte, wird eine eigene Zertifizierungsstelle erstellt. OpenSSl bringt da alle nötigen Mittel für die Erzeugung eines CAs mit. Dies bringt den Nachteil mit, dass Computersysteme diesem Zertifikat nicht automatisch trauen, daher muss dann das Zertifkat von Hand dem Certificate Store als «Trusted Root Certification Authority» hinzugefügt werden.

Nachdem die Zertifizierungsstelle erfolgreich generiert wurde, kann das Server Zertifikat erstellt werden. Anschliessend muss dieses Server Zertifikat von der eben erstellten Zertifizierungsstelle signiert werden.

Da sowohl die Zertifizierungsstelle, als auch das Server-Zertifikat auf dem gleichen Computer erstellt werden, muss darauf geachtet werden, dass bei der Erzeugung unterschiedliche Parameter gesetzt werden. Die betroffenen Parameter sind zum Beispiel «Locality Name», «Organizational Name», «Organizational Unit» etc. Falls hier dieselben Werte eingetragen werden, schlägt die Signierung des Serverzertifikates fehl.
Weiter muss beachtet werden, dass im Server-Zertifikat der «Common Name» dem FQDN (Fully Qualified Domain Name) des Servers entspricht, auf dem der MQTT-Broker laufen soll. Wird hier beispielsweise nur der Hostname eingetragen, schlägt die Überprüfung des Zertifikates fehl, da sich der CN vom FQDN des Servers unterscheidet.

Installation und Konfiguration des Brokers
Wie bereits im Lösugnskonzept erarbeitet, wird als MQTT-Broker «Mosquitto» eingesetzt. Der Broker kann als Binary installiert und über die Commandline gestartet werden. Nach der Standard-Installation muss der Broker Konfiguriert werden. Dazu wird das File «mosquitto.conf» bearbeitet.
Folgende Parameter müssen editiert werden:

Parameter	**Erklärung**
bind_address	
mqttbrokerba.cloudapp.net	IP-Adresse, an den der Default-Listener gebunden wird.

port 8883	Port, auf den der Default-Listener hören soll. Wenn er nicht speziell definiert wird, hört der Listener per Default auf den Port 1883. Da aber mit TLS verschlüsselt wird, muss dieser von Hand auf den dafür vorgesehenen Port 8883 gesetzt werden.
cafile `C:\OpenSSL-Win64\bin\m2mqtt_ca.crt`	Hier wird der Pfad eingetragen für das zuvor erstellte CA-Zertifikat.
certfile `C:\OpenSSL-Win64\bin\m2mqtt_srv.crt\path`	Hier wird der Pfad für das PEM-Encodete Server Zertifikat eingetragen.
keyfile `C:\OpenSSL-Win64\bin\m2mqtt_srv.key\path`	Hier wird der Pfad für das PEM-Encodete Keyfile eingetragen.
tls_version tlsv1	Diese Option definiert die zu verwendende TLS-Version. Für Openssl (Version 1.0.2) wird tlsv1 verwendet.
password_file `C:\ProgramFiles(x86)\mosquitto\passwords`	Das Passwords-File beinhaltet die definierten Benutzernamen und Passwörter, um sich am Broker anzumelden. Durch setzen dieses Pfades wird dies automatisch vom Broker berücksichtigt.

4.6.3. Client (Azure Worker Role)

Wie die Abbildung 3 (Systemübersicht) zeigt, befindet sich nebst dem Broker auch ein Client, in form einer Worker Role, in der Cloud. Die Worker Role abonniert alle Topics und persistiert die Messages im Table bzw. Blob-Storage.

In der `OnStart()`-Methode der Worke Role werden zu erst die Referenzen zum Table- und Blob-Storage erzeugt.
Danach wird die Verbindung zum MQTT-Broker hergestellt, die Topics definiert, die er abonnieren möchte und der QoS-Level gesetzt.
Damit der Client benachrichtigt wird, wenn eine Message eintrifft, wird der Eventhandler `client_MqttMsgPublishReceived()` definiert. Die Methode muss die gleichen Parameter entgegennehmen, wie die Delegate-Methode. Das ist einerseits der Sender (Object) und die Event-Argumente. Damit der Eventhandler beim Eintreffen einer Message aufgerufen wird, muss dieser auf dem Event registriert werden:
`client.MqttMsgPublishReceived += client_MqttMsgPublishReceived;`

```
1  public override bool OnStart()
2  {
3    setupStorageConnections();
4    MqttClient client = new MqttClient(
5                          "mqttbrokerba.cloudapp.net",
6                             8883, true, null
7                            );
8    client.MqttMsgPublishReceived += client_MqttMsgPublishReceived;
9    client.Connect(Guid.NewGuid().ToString(),
10                "username",
11                "password"
12                );
13   string[] topics = { "openhab/+" };
14   byte[] qos = { MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE };
15   client.Subscribe(topics, qos);
16
17   return result;
18 }
```

Listing 6: WorkerRole.cs - MQTT Topic Subscribe

Im Eventhandler wird dann schlussendlich die Logik zur Persistierung eingefügt. Wenn eine Message über das Topic «openhab/blob» empfangen wird, handelt es sich um eine Fotografie der Webcam. Dieses JPG-File wird als Byte-Array übermittelt und wird so auch im Blob-Storage abgelegt.
Bei allen anderen Messages muss es sich um Text handeln, daher werden sie im Table-Storage abgelegt.

```
void client_MqttMsgPublishReceived(object sender,
                                   MqttMsgPublishEventArgs e)
{
    if (e.Topic.Equals("openhab/blob"))
    {
        CloudBlockBlob blockBlob = container.
                       GetBlockBlobReference(Guid.NewGuid()
                                                 .ToString());
        blockBlob.UploadFromByteArray(e.Message,
                                0, e.Message.Length);
     }
    else
    {
        var message = System.Text.Encoding.
                              Default.GetString(e.Message);
        Entity entity = new Entity(message);
        TableOperation insertOperation = TableOperation.
                                   Insert(entity);
        table.Execute(insertOperation);
    }
}
```

Listing 7: WorkerRole.cs - EventHandler

Zertifikat

Da die Verbindung durch SSL/TLS mit einem Self-signed Zertifikat verschlüsselt wird, muss dieses Zertifikat dem Certificate Store des virtuellen Hosts als «Trusted Root Certification Authority» hinzugefügt werden. Dies muss vorgenommen werden, bevor die Worker Role versucht eine Verbindung zum Broker herzustellen. Ansonsten terminiert die Worker Role mit einer Exception, da sie das Zertifikat nicht akzeptiert.
Da man zwischen Erzeugung des virtuellen Hosts und dem Start der Worker Role nicht auf die Maschine zugreifen kann, muss das zertifikat programmatisch als Startup-Skript dem Store hinzugefügt werden. Für solche Aufgaben bietet Microsoft Azure die Möglichkeit Startup Tasks zu definieren. Dies wird in der ServiceDefinition vorgenommen, durch einfügen folgender Anweisung:

```
<Startup>
    <
```

```
3       Task commandLine="startup.cmd" executionContext="elevated"
4       taskType="simple"
5    />
6 </Startup>
```

Listing 8: ServiceDefinition.csdef - Startup Task

Dieser Startup Task führt das Batchfile «startup.cmd» aus, welches das Zertifikat in den Store hinzufügt. Dazu muss das Skript und auch das Zertifikat dem Visual-Studio-Projekt als Element hinzugefügt werden. Für beide Elemente muss in den Eigenschaften der Buildvorgang als «Inhalt» deklariert werden.

```
1 certutil -addstore root m2mqtt_ca.cer
```

Listing 9: startup.cmd - Zertifikat hinzufügen

Im Sequenzdiagram der Abbildung 8 wird aufgezeigt, wie die MQTT-Nachrichten nachdem QoS 2 versendet werden.

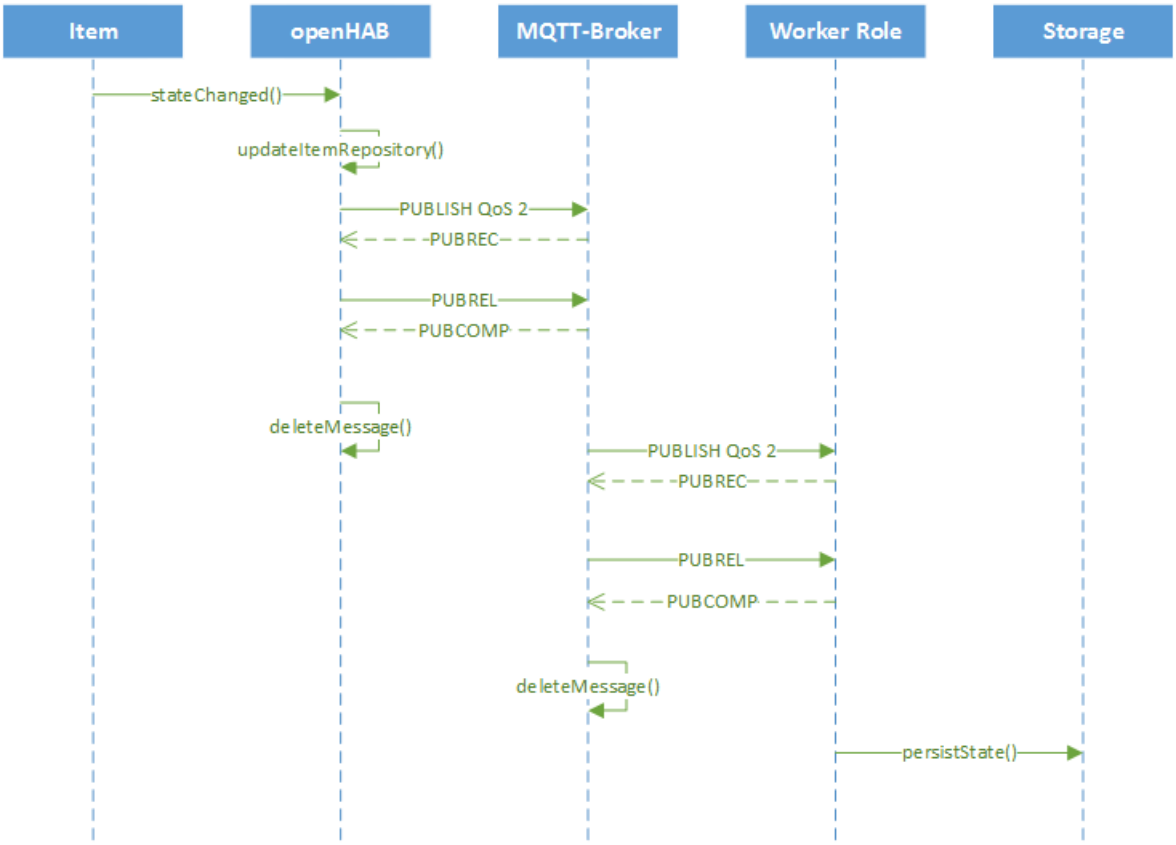


Abbildung 8.: Sequenzdiagramm MQTT

4.7. Notification

Der Mobile-Client wird durch Notifications benachrichtig, sobald der Alarm ausgelöst wird. Umgesetzt wird dies mithilfe des Google Cloud Messaging Dienstes.

4.7.1. Cloud

Um den GCM-Dienst nutzen zu können, muss zu erst in der Google Developer Console (`https://console.developers.google.com`) ein Projekt erstellt werden mit einer Cloud-Messaging-API. Für diese API kann ein Schlüssel erzeugt werden, der für das Versenden der Notification über die neu erstellte API benötigt wird.

Ausgelöst wird der ganze Vorgang durch die MQTT-Nachricht «alarm_activated». Die Worker Role nimmt diese Nachricht entgegen und erzeugt eine Notification-Message:

```
{
  "registration_ids" : ["APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx..."],
  "data" : {
   "message" : "Alarm wurde ausgeloest!"
  }
}
```

Listing 10: Notification.cs - Notification Message

Die Registration-Id bezeichnet den Client, der die Message erhalten soll. Dazu müssen sich die Clients zuvor bei GCM registrieren und erhalten dann die Id. Diese Message muss also für jeden Client einzeln über einen HTTP-Post gesendet werden. GCM weiss anhand des Id-Strings an welche Android-Clients er die Message weiterleiten soll.

Folgende Grafik stellt den Registrations-Vorgang dar:

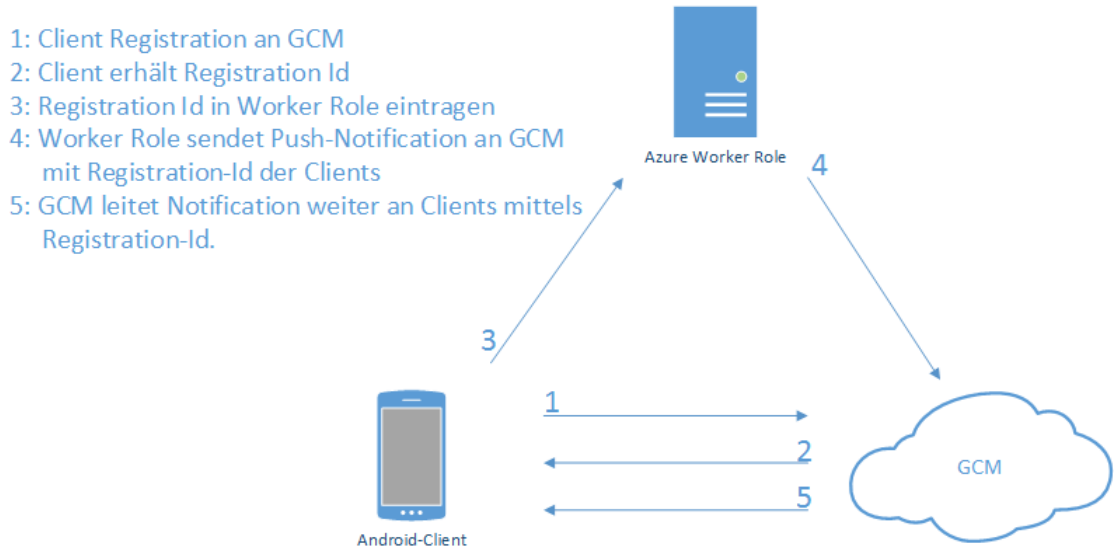


Abbildung 9.: Vorgang Push-Notification

Die nachfolgende Grafik (Abbildung 10) zeigt auf, wie anhand der MQTT-Message die Notifikation ausgelöst wird.

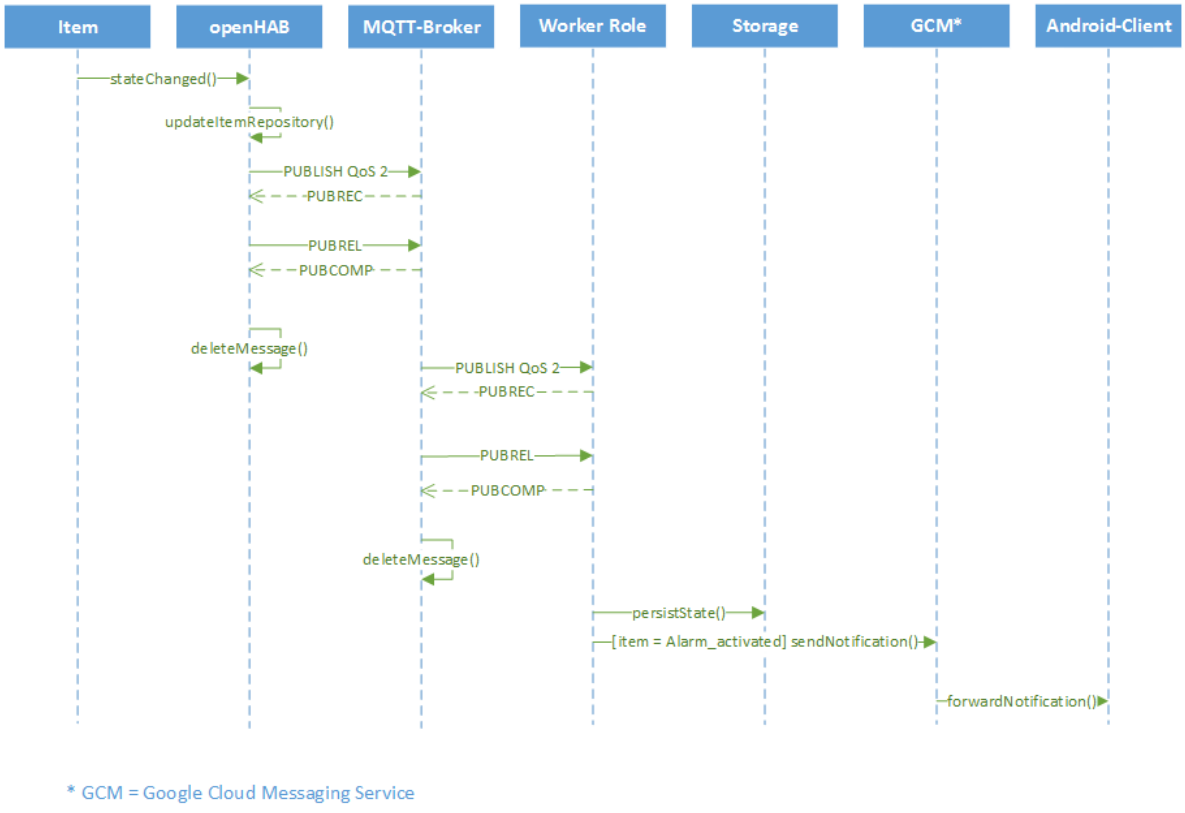


Abbildung 10.: Sequenzdiagramm MQTT-Notification

4.7.2. Android-Client

4.8. Sicherheit

Aus technischer Sicht können folgende Massnahmen getroffen werden, um die im Lösungskonzept beschriebenen Schwachstellen zu sichern:

- Verschlüsselung der MQTT-Verbindung.
- Anmeldung am MQTT-Broker durch Benutzername & Passwort.
- Verschlüsselung des WLANs.

4.8.1. Verschlüsselung MQTT-Verbindung

Die Verbindung zum MQTT-Broker wird durch SSL/TLS Verschlüsselt. Dazu wird ein Self-signed Zertifikat eingesetzt. Wie dieses Zertifikat erzeugt wird, ist aus dem Anhang

A zu entnehmen.
Die jeweilige Implementierung der TLS-Verbindung wird im Kapitel 4.6 erläutert.

4.8.2. Konfiguration MQTT-Broker

Am Broker sollen sich nur authentifizierte und autorisierte Clients anmelden können. Dazu bietet der MQTT-Standard die Möglichkeit ein Password-File zu erzeugen. In dieses File wird der Benutzername und das zugehörige Passwort eingetragen, mit dem sich die Clients anmelden können. Nur wenn diese Angaben übereinstimmen, kann eine Verbindung aufgebaut werden.
Mosquitto bringt ein Tool mit, zur Erzeugung und Verschlüsselung dieses Files. Über die Konsole wird das Skript aufgerufen und die Benutzerangaben können als Parameter übergeben werden:

```
1 > mosquitto_passwd -c passwordFile
2 > mosquitto_passwd -b passwordFile username password
```

Listing 11: mosquitto_passwd.exe - generate password-file

Wie sich die Clients gegenüber dem Broker authentifizieren, ist im Abschnitt 4.6.3 und <tbd>(MQTT Binding) ersichtlich.

4.8.3. Verschlüsselung WLAN

Über das WLAN kann man ohne Authentifizierung auf das openHAB zugreifen. Um unautorisierten Zugriff zu verhindern, wird das WLAN mit dem Sicherheitsstandard WPA2 verschlüsselt. Die Kommunikation wird also über den symmetrischen AES (Advanced Encryption Standard) Algorithmus verschlüsselt. Da es sich um eine symmetrische Verschlüsselung handelt, muss jeder Client ein PSK (Pre Shared Key) besitzen, der jeweils beim Verbindungsaufbau eingegeben werden muss.

Literaturverzeichnis

[1] openHAB, “openHAB documentation/wiki.” [Online]. Available: https://github.com/openhab/openhab/wiki

[2] “Mosquitto documentation.” [Online]. Available: http://mosquitto.org/documentation/

[3] “M2Mqtt documentation.” [Online]. Available: https://m2mqtt.wordpress.com/m2mqtt_doc/

Glossar

AES Advanced Encryption Standard

API Application Programming Interface

BLOB Binary Large Objects

CA Certificate Authority

GCM Google Cloud Messaging

HTTP Hypertext Transfer Protocol

IoT Internet of Things

IP Internet Protocol

MQTT Message Queue Telemetry Transport

MS Microsoft

NFC Near Field Communication

OSGi Open Services Gateway Initiative

PSK Pre Shared Key

QoS Quality of Service

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

URL Uniform Resource Locator

A. Installationsanleitungen

Zertifikat Erzeugung

Die Zertifikate wurden mit OpenSSL (`http://slproweb.com/products/Win32OpenSSL.html`) erzeugt.

CA Zertifikat erstellen

```
openssl req -new -x509 -days 3650 -keyout m2mqtt_ca.key -out m2mqtt_ca.crt
```
Dieser Command erzeugt ein CA (Certificate Authority) Zertifikat mit einem privatem Schlüssel. Das X.509-Zertifikat ist für 3650 Tage gültig.

Server Zertifikat erstellen

```
openssl genrsa -des3 -out m2mqtt_srv.key 1024
```
Erzeugt einen 1024 Bit grossen 3DES, privaten Schlüssel.

Signierung des Server Zertifikates

```
openssl req -out m2mqtt_srv.csr -key m2mqtt_srv.key -new
```
Dieses Kommando erzeugt ein Zertifikat-Request vom Server zur Signierung an die CA. Durch dieses Kommando wird lediglich der Request erzeugt. Das bedeutet, dass die Signierung noch nicht durchgeführt wurde.

```
openssl x509 -req -in m2mqtt_srv.csr -CA m2mqtt_ca.crt -CAkey m2mqtt_ca.key
-CAcreateserial -out m2mqtt_srv.crt -days 3650
```

Mit diesem Kommando wird der Server-Request signiert. Als Produkt entsteht das finale Zertifikat für den Broker.

Für die Client-Library (M2Mqtt) muss das Zertifikat noch ins DER-Format konvertiert werden:

```
openssl x509 -outform der -in m2mqtt_ca.crt -out m2mqtt_ca.der
```

B. Projektplan

Thema	Aufbau einer Smart-Home Beispielapplikation
Studenten	Dominik Freier, Marco Leutenegger
Betreuer	Prof. Hansjörg Huser

Änderungsgeschichte

Datum	Version	Änderung	Autor
25.02.2015	0.0.1	Dokument erstellen	M. Leutenegger
27.02.2015	0.0.2	Meilensteine erfasst	D. Freier, M. Leutenegger
04.03.2015	0.1.0	Risikomanagement angepasst	D. Freier, M.Leutenegger

Einführung

Zweck

Dieses Dokument dient als Projektplan für die Bachelorarbeit von Dominik Freier und Marco Leutenegger und definiert alle organisatorischen Rahmenbedingungen.

Gültigkeitsbereich

Die Gültigkeit des Projektplans beschränkt sich auf die Bachelorarbeit von Dominik Freier und Marco Leutenegger im Frühjahrssemester 2015.

Referenzen

Bezeichnung	**Referenz**
Risikomanagement	Siehe separates Dokument
Security Infos	`https://github.com/openhab/openhab/wiki/Security`

Projekt und Übersicht

Zweck und Ziel

Diese Bachelorarbeit hat as Ziel, eine Smart-Home Beispielapplikation aufzubauen, welche wesentliche Aspekte einer Internet-of-Things-Anwendung demonstriert, wie Steuern von Devices, Lesen von Sensoren, Event-Verarbeitung, Überwachung und intelligente Abläufe steuern, Streaming von Sensordaten und Online-Analyse der Daten usw.
Das System soll auf einer tragfähigen und erweiterbaren Architektur aufgebaut werden und Microsoft Azure als Cloud Plattform benutzen.

Lieferumfang

Die abzuliefernden Dokumente und Software-Artefakte des Projekts richten sich im Wesentlichen nach den Vorgaben aus den Dokumentationsanleitungen der HSR. Eine davon abweichender Lieferumfang wurde mit dem Betreuer besprochen und genehmigt.

Referenzen

Nr.	Art	Bezeichnung	Form	Empfänger
1	Publikation	Poster	PDF	H.Huser
2	Publikation	Kurzfassung	PDF	H.Huser
3	Dokument	Bericht	PDF/Ausdruck	H.Huser
4	Dokument	Projektplan	PDF/Ausdruck	H.Huser
5	Dokument	Sitzungsprotokolle	PDF/Ausdruck	H.Huser
6	Dokument	Eigenständigkeitserklärung	PDF/Ausdruck	H.Huser
7	Dokument	Erfahrungsbericht D.Freier	PDF/Ausdruck	H.Huser
8	Dokument	Erfahrungsbericht M.Leutenegger	PDF/Ausdruck	H.Huser
9	Source	Code-Abgabe	ZIP	H.Huser
10	Archiv	2x Deliverables 1-9	DVD	H.Huser

Projektorganisation

Die Dokumentation des Projekts gliedert sich in diesen Projektplan und einen Bericht. Im Projektplan werden alle organisatorischen Aspekte festgehalten, wie etwa die Planung der Meilensteine, Aufgaben der Teammitglieder oder Abmachungen zum Dokumentemanagement. Im Bericht werden technische Beschreibungen der Ausgangslage, Diskussionen für Lösungsansätze, Requirements und Details zur Umsetzung dokumentiert.

Damit die Teammitglieder möglichst parallel und effizient arbeiten können, werden alle Dokumente mit LaTeX geschrieben und auf einem Git-Repository verwaltet. Daruch wird das Risiko von Versionskonflikten reduziert und der Zugriff insbesondere für den Betreuer vereinfacht.

Die Verwaltung der Aufgaben und agilen Vorgänge erfolgt durch Jira. Wir erhielten zu diesem Zweck eine Classroom Lizenz vom Hersteller Atlassian. Jira wurde auf einem virtuellen Server der HSR installiert.

Organisationsstruktur

Verantwortung	**Teammitglied**
Verwaltung und Bereinigung de Dokumente	D. Freier, M. Leutenegger
Pflege virtueller Server, Jira inkl. Backups	D. Freier, M. Leutenegger
Sitzungsprotokolle verfassen	D. Freier, M. Leutenegger
Iterationsplanung	D. Freier, M. Leutenegger
Risikomanagement	D. Freier, M. Leutenegger
Architekturdesign	D. Freier, M. Leutenegger

Externe Schnittstellen

Betreuer der Bachelorarbeit ist Prof. Hansjörg Huser. Experte ist Herr Stefan Zettel. Gegenleser ist <tbd>.

Management Abläufe

Zeitliche Planung

Das Projekt wird während des Frühjahrssemester 2015 durchgeführt. Der Start der Arbeit war am Montag, den 16. Februar 2015. Die Abgabe der Vollständigen Dokumentation an den Betreuer erfolgt am Freitag, den 12. Juni 2015. Als Zeitbudget sollen in den 17 Wochen insgesamt 720 Stunden, bzw. rund 21 Stunden pro Woche und Student eingeplant werden.

Vorgehensmodell

Als Vorgehensmodell wurde der Rational Unified Process ausgewählt, da das Projektteam mit diesem Modell aus früheren Arbeiten (inkl. Semesterarbeit) vertraut ist und damit gute Erfahrungen gemacht hat. Die Phasen wurden nach dem Schema «eins, drei, drei, eins» in insgesamt acht Iterationen à zwei Wochen aufgeteilt.

Meilensteine

MS	Iter.	Beschreibung	Datum
MS1	I1	Der Projektauftrag wurde zusammen mit dem Betreuer besprochen und ist akzeptiert. Den Teammitgliedern ist klar, welches die Ziele des Projekts sind und haben eine gemeinsame Vision. Die organisatorischen Aspekte wurden so weit wie möglich abgeklärt und die benötigte Infrastruktur steht allen Beteiligten zur Verfügung.	04.03.2015
MS2	E1	Die Analyse der funktionalen und nicht-funktionalen Anforderungen ist abgeschlossen und die Use Cases definiert. Die technische Umsetzung der Use Cases wurde analysiert und mit Umsetzung kann begonnen werden. Die Hardware wurde bestellt und für das Mobile-App wurden erste Mockups gezeichnet.	18.03.2015
MS3	E2	Ein Architekturprototyp (Installation und Konfiguration openHAB) existiert. Ein Prototyp für die Use Cases mit existierenden Bindings wurde entwickelt.	01.04.2015

MS4	E3	Prototyp mit eigenen Bindings wurde entwickelt, parallel dazu wird die Cloud mit den benötigten Komponenten aufgesetzt.	15.04.2015
MS5	C1	Die Use Cases mit den eigenen Bindings sind fertig implementiert.	29.04.2015
MS6	C2	Das Android-App ist gemäss den, in der Analyse (E1) gezeichneten, Mockups entwicklt und die geplanten Funktionen sind implementiert.	13.05.2015
MS7	C3	Der geschriebene Code wurde überarbeitet und optimiert. Die nötigen Komponenten sind gemäss FR und NFR getestet.	27.05.2015
MS8	T1	Die Dokumentation wurde nachgeführt, und finalisiert. Die Deliverables werden am darauf folgenden Freitag den entsprechenden Personen übergeben. Dieser Meilenstein definiert den Abschluss des Projektes.	10.06.20115

Iterationsplanung

It.	**Arbeitspakete**	**Ziele**	**SW**
I1	1. Besprechung Projektauftrag 2. Einarbeitung Thematik 3. Aufsetzen LaTeX-Dokument	□ MS1: Projektauftrag erhalten □ Gemeinsame Vision des Projekts	1-2
E1	4. Definition der Use Cases 5. Aufbau/Setup/Anordnung 6. Hardware Evaluation 7. Abklären technische Machbarkeit 8. Android Mock-Up 9. Meilensteine und Iterationsplan	□ MS2: Review Projektplan □ Hardware bestellt □ Mockups für App gezeichnet	3-4

E2	10. Installation openHAB auf Raspberry Pi 11. Einrichten WLAN und Router 12. Use Cases mit DSL umsetzen 13. Integration HomeMatic 14. Integration Philips Hue 15. Integration Webcam 16. Dokumentation nachführen	□ MS3: Erster Prototyp existiert □ Erste Version der Architektur-Dokumentation	5-6
E3	17. Aufsetzen und Anpassen der Azure Cloud 18. Implementation MQTT 19. Integration Azure Cloud 20. Dokumentation nachführen	□ MS4: Prototyp mit Bindings fertig □ Cloud aufgesetzt	7-8
C1	21. Alle Komponenten vollständig integrieren 22. Vernetzung der Hardware 23. Dokumentation nachführen	□ MS5: Binding für Cloud erstellt	9-10
C2	24. Android App Model portieren (HABDroid) 25. Anbindung Android an Cloud 26. User Interface 27. Dokumentation nachführen	□ MS6: Android-App entwickelt	11-12
C3	28. Refactoring und Unit-Testing 29. Systemtests 30. Überprüfung NFR und FR 31. Dokumentation nachführen	□ MS7: Refactoring und Testing durchgeführt	13-14
T1	32. Dokumentation abschliessen 33. Poster erstellen 34. Dokumentation drucken & binden 35. CD erstellen 36. BA abgeben	□ MS8: Abschluss des Projektes □ Dokumentation abgeschlossen □ Deliverables übergeben	15-16

Besprechungen

Wöchentliche Besprechungen:

Bezeichnung	**Ziel**	**Wochentag**	**Uhrzeit**	**Ort**
Teambesprechung	Projektarbeiten im Plenum erledigen	Donnerstag	08:10-08:40	HSR (Labor)
Fortschrittsbesprechung	Fortschritte bzw. Probleme besprechen	Mittwoch	10:10-10:50	HSR (6.010)

Risikomanagement

Risiken

Nachstehend wird auf die projektbezogenen Risiken eingegangen. Eine Übersicht in Form einer Tabelle ist auf der nächsten Seite zu finden. Die Tabelle wird während des ganzen Projektes angepasst und aktualisiert, falls notwendig.

Umgang mit Risiken

Reserven/Rückstellungen

Das grösste Risiko stellt R1 (ungeplante Machbarkeiten) dar. Aus diesem Grund werden in diesem Projekt Rückstellungen von 20 Stunden eingeplant.

Überprüfung von Risiken

Weitere Risiken werden im Laufe des Entwicklungsprozesses erkennbar. Hierfür aktualisieren wir dieses Dokument, welches als zentrale Stelle dient, um Entscheidungen und Risiken zu Dokumentieren und auch eine zentrale Anlaufstelle bei Fragen darstellt. Des weiteren wird in der Beschreibung des betroffenen Vorgangs auf mögliche Risiken hingewiesen und dokumentiert.

Nr	Titel	Beschreibung	Schaden[h]	Eintrittswahrsch.	Gew. Schad.	Vorbeugung.	Verhalten beim Eintreten.
R1	Ungeplante Machbarkeit	Nicht alle Arbeitspakete in Iteration oder Meilensteine abgedeckt.	20	40%	8	Laufende Kontrolle des Zeitplans	Überstunden in Kauf nehmen, um folgende Iteration nicht in Gefahr zu bringen.
R2	Absturz Jira-Server und Datenverlust	Der virtuelle Server der HSR stürzt ab, und die Daten des Jira gehen verloren.	2	10%	0.2	Backup pro Woche erstellen.	Letztes Backup einspielen und die Differenz von Hand erneut eintragen.
R3	Verlust von Code	Das persönliche Notebook stürzt ab und die Daten sind verloren.	2	10%	0.2	Code wird ständig auf GitHub gepusht.	Lab-PC oder sonstige Computer verwenden und GIT Repository Klonen.
R4	Fabrikationsfehler Sensoren	Die Sensoren kommen mit einem Fabrikationsfehler an.	20	10%	2		Sensor zurücksenden und mit anderem weiterarbeiten.
R5	Schnittstellen Sensoren	Schnittstellen zu anderen Systemen bereitet Probleme	16	5%	0.8	Dokumentation gut prüfen.	Community durchforsten, Workaround suchen.

Arbeitspakete

Die Arbeitspakete wurden im Projektmanagementtool Jira als Vorgänge definiert. Einige Vorgänge beinhalten weitere Untertätigkeiten, die wir ebenfalls als einzelne Arbeitspakete betrachten.

Eine Übersicht mit allen Arbeitspaketen und dem zeitlichen Ablauf nach Iterationen befindet sich unter: `http://sinv-56046.edu.hsr.ch:8080` > Agile > Zeige alle Boards > baIOTBoard > Plan

Infrastruktur

Software

Wie in jedem Projekt kommt verschiedene Software zum Einsatz.

Software	**Version (Major)**	**Beschreibung/Einsatzbereich**
GitHub	v3	Source Code Verwaltung inkl. Branchmanagement, Web Interface für Git-Verwaltung.
Atlassian: Jira	6.4	Projektmanagement
Windows Server	2012 R2 (64Bit)	Virtueller Server für Jira
<tbd>	<tbd>	<tbd>

Qualitätsmassnahmen

Massnahme	Zeitraum	Ziel
Einsetzen eines Projekt-Management-Tools	ganzes Projekt	Alle auf dem aktuellsten Stand halten
Versionierungssystem (git) ganzes Projekt	Sicherung des Codes/Doku, keine Blockaden	
Koordinationsmeetings	ganzes Projekt	Ressourcen optimal zuteilen: Wer benötigt wo Hilfe, wer ist schon fertig?
Vier-Augen-Prinzip	ganzes Projekt	Dokumentation/Programmcode wird jeweils von beiden Partnern kontrolliert. Bei einem Ausfall einer Person, ist das andere Mitglied informiert.

Dokumentation

Ablage

Alle Dokumente können auf dem GitHub Repository gefunden werden. Die Vorgänge werden mit Jira auf einem virtuellen Server der HSR verwaltet.

- Dokumentation: `https://github.com/greekins/baIOT_TeX`
- Vorgänge: `http://sinv-56046.edu.hsr.ch:8080`

Der Source-Code wird mit Git verwaltet: <tbd>

Qualität

- Commits verlangen eine Beschreibung
- Benutzerfreundliche Commit-Übersicht dank Github

- Für die Qualität des Codes wird in jeder Iteration (ab Elaboration E2) Codereviews durchgeführt (siehe Managementabläufe)

Projektmanagement

Es wird die von Atlassian zur Verfügung gestellte Umgebung eingesetzt:
`http://sinv-56046.edu.hsr.ch:8080`
Gast Login: hhuser

Entwicklung

Code Reviews

Die Commits sind für alle Projektmitglieder ersichtlich und werden in einem Activity Stream auf dem Repository unter «Graph» angezeigt. Diese werden sporadisch von den anderen Mitgliedern geöffnet und kurz überprüft.

Bei einem wöchentlichen Meeting werden getätigte Implementierungen im Plenum angeschaut und besprochen. Auch lautet usere Regel, dass bei Unsicherheiten bei laufender Entwicklung Rat vom anderen Teammitglied eingeholt wird.

Code Style Guidelines

Es wird sich an die gängigen Style Guidelines gehalten, die im Laufe des Studiums eingeführt wurden.

C. Sitzungsprotokolle

Sitzung 1 - Kick-Off

Datum: 17. Februar 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Projektdefinition
Aufbauen einer Demoanwendung für «Smart Home». Wie genau die aussehen wird, steht noch nicht fest. Ist Bestandteil der Analyse und Evaluation. Als Resultat der Arbeit soll ein Showcase entstehen mit ein paar Anwendungsfällen.
Mögliche Bestandteile:

- Sensoren, Raspberry-Pi
- Cloud (simple gehalten, Service Bus)
- UI (Mobile/Tablet)

Anstehende Arbeiten

- Evaluation HW-Platform
- Evaluation Framework
- erste Version des Projektplans

Organisatorisches

- Virtueller Server beantragt
- Wöchentliche Besprechungen: Mittwoch, 10.10 Uhr

Sitzung 2

Datum: 25. Februar 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- System-Architektur von Herrn Huser zur Kenntnis genommen und akzeptiert.
- Anwendungsszenarien sollen richtung Einbrecherschutz gehen (Türkontakte, Bewegungssensoren etc.

Anstehende Arbeiten:

- Bestellliste mit Sensoren und Aktoren erstellen.
- Anwendungsszenarien Anpassen.
- Erste Version des Projektplans erstellen.

Sitzung 3

Datum: 04. März 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Wunderbar wird vernachlässigt, die Antwort abgewartet. Als Ersatz wird Tinkerforge gewählt.

Anstehende Arbeiten:

- Bestellliste anpassen.
- Einige Änderungen am Projektplan.
- Risikoliste anpassen.
- Detailplanung erstellen.

Sitzung 4

Datum: 11. März 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Iterationsplanung wurde besprochen und gutgeheissen.
- Use Cases wurden besprochen und gutgeheissen.
- Projektplan wurde als Ganzes angenommen.
- Tinkerforge wird nach Absprache nicht weiter verfolgt. Falls genügend Zeit vorhanden ist, werden, aus der bestehenden Hardware, die Use Cases erweitert.
- **Meilenstein 1 erreicht und abgenommen!** - Phase E1 abgeschlossen.

Anstehende Arbeiten:

- Beginn der Phase E2.
- Nach erhalt der Hardware mit Prototyp beginnen.

Sitzung 5

Datum: 17. März 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- **Beginn der Phase E2**
- Fassung für Glühbirne kaufen die Studenten in einem Elektro-Fachgeschäft. Die Kosten werden vom Institut übernommen.
- Router wird von Herrn Huser organisiert.

Anstehende Arbeiten:

- Installation openHAB auf Raspberry Pi.
- Einrichten des Netzwerkes (Router).
- Use Cases umsetzen mit HomeMatic, Philips Hue und Webcam.
- Erste Version der Architekturdokumentation erstellen.

Sitzung 6

Datum: 25. März 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Prototyp wurde vorgeführt und für gut befunden.
- Man befindet sich im Zeitplan und beginnt mit den nächsten Arbeitspaketen.

Anstehende Arbeiten:

- Termin für Zwischenpräsentation mit Prof. Dr. Rinkel in zwei Wochen (8. April 2015)
- Integration Webcam abschliessen.
- Mit Aufsetzen von Azure Cloud und Binding beginnen.

Sitzung 7

Datum: 01. April 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Cloud: Anstatt ein eigenes Binding zu schreiben, wird das MQTT-Binding für die Kommunikation mit der Cloud eingesetzt.
- In der Dokumentation soll mehr auf Übersicht geachtet werden (Big Picture). Am Anfang abstrakt beginnen und immer detailierter werden.

Anstehende Arbeiten:

- Zwischenpräsentation vorbereiten.
- Cloud Aufsetzen und anpassen.
- Dokumentation nachführen.
- MQTT-Broker evaluieren.

Sitzung 8 (Zwischenpräsentation)

Datum: 08. April 2015
Teilnehmer: Prof. H. Huser, Prof. Dr. A. Rinkel, Dominik Freier, Marco Leutenegger

Diese Sitzung wurde durch die Zwischenpräsentation für Herr Rinkel ersetzt. Gezeigt wurden alle Aspekte der Bachelorarbeit und der bestehende Prototyp.

Anforderungen betreffend Dokumentation:

- Event Driven Design: Differenzierung/Abgrenzung.
- Möglichkeiten/Grenzen von openHAB aufzeigen.
- Einsatz von Cloud begründen, Mehrwert aufzeigen.
- Ersetzen von Mobile App begründen, Mehrwert aufzeigen.

Anstehende Arbeiten:

- Cloud aufsetzen und anpassen.
- MQTT-Broker evaluieren.
- Integration openHAB in Azure Cloud.
- Dokumentation nachführen.

Sitzung 9

Datum: 15. April 2015

Diese Sitzung wurde aufgrund der HSR Stellenbörse und der fehlenden Notwendigkeit einer Standortbestimmung abgesagt.

Sitzung 10

Datum: 24. April 2015

Diese Sitzung wurde aufgrund von Terminen von Herrn Huser abgesagt. Anstelle des Meetings wurde der aktuelle Stand per E-Mail festgehalten:

> Folgende Arbeitsschritte wurden gemäss unserem Projektplan verrichtet:

- Einarbeitung MQTT
- Evaluation MQTT Broker & .NET Library
- Aufsetzen des MQTT Brokers auf einer VM in der Azure-Cloud (inkl. End-to-end Verschlüsselung)
- Erstellen eines Cloudservices mit einer Worker Role, die sich auf Topics subscribed und die empfangenen Messages im Table- bzw. Blob-Storage persistiert.
- Erstellen einer Action in openHAB, um die Bilder der Webcam per MQTT an den Broker zu senden.

Momentan sind wir im Plan eine Woche im Voraus und können ab nächster Woche bereits mit der Mobile-App beginnen.

Sitzung 11

Datum: 29. April 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Die Implementierung des MQTT-Protokolls wurde gezeigt und erklärt.
- Der Stand der Arbeit, im Zusammenhang mit dem Projektplan wurde definiert.

Anstehende Arbeiten:

- Dokumentation nachführen.
- Beginn mit Android-App.

Sitzung 12

Datum: 06. Mai 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- In der Dokumentation fehlt ein Beschrieb bezüglich Sicherheit.

- Interaktion Commands, Events etc. Die wichtigsten Abläufe durch ein Sequenzdiagramm aufzeigen.

Anstehende Arbeiten:

- Sicherheit beschreiben und Sequenzdiagramm erstellen.
- Mit dem Android-App beginnen (Umfang reduzieren).

Sitzung 13

Datum: 13. Mai 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Notification wurde in die Anforderungen übernommen und als höchstes Priorisiert.
- Die Notifikation soll ins bestehende, reduzierte, App integriert werden.
- Falls Zeit vorhanden ist, soll das eigene App umgesetzt werden. Dazu wird das ReactiveX-Framework eingesetzt (`http://reactivex.io/`).

Anstehende Arbeiten:

- Notifikation umsetzen
- Android-App erstellen mit ReactiveX-Framework

Sitzung 14

Datum: 20. Mai 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Prototyp Android App wurde gezeigt.
- Notification wurde demonstriert.

Anstehende Arbeiten:

- Android App weiterentwickeln.
- Dokumentation nachführen.

Sitzung 15

Datum: 27. Mai 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

- Stand Android App wurde gezeigt.
- Zeitpunkt für mündliche Prüfung wurde auf 23.06.2015 gesetzt.

Anstehende Arbeiten:

- Android App fertigstellen.
- Dokumentation nachführen.

Sitzung 16

Datum: 03. Juni 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

Anstehende Arbeiten:

Sitzung 16

Datum: 03. Juni 2015
Teilnehmer: Prof. Hansjörg Huser, Dominik Freier, Marco Leutenegger

Organisatorisches:

Anstehende Arbeiten:

D. Persönliche Reflektion

Marco Leutenegger

<tbd>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Dominik Freier

<tbd>Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.