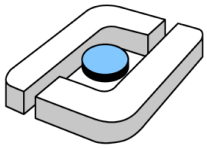


Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

Software Engineering Projekt

über das Thema:

SmartHome From Device to Service

Autor: Tobias Münch (tobias.muench@hs-osnabrueck.de)

Philip Schöppe (philip.schoeppe@hs-osnabrueck.de)

Jana Willmann (jana.willmann@hs-osnabrueck.de)

Fabian Köster (fabian.koester@hs-osnabrueck.de)

Max Zelass (max.zelass@hs-osnabrueck.de)

Alexander Klassen (alexander.klassen@hs-osnabrueck.de)

Julian Dohe (julian.dohe@hs-osnabrueck.de)

Inhaltsverzeichnis

1 Tabellenverzeichnis

2 Abbildungsverzeichnis

3 Literaturverzeichnis

JBOSS. (2013, 08 21). *Drools - The Business Logic integration Platform*. Retrieved from Drools - The Business Logic integration Platform: http://www.jboss.org/drools/

Oracle. (2013, 08 21). *The JAVA EE 6 Tutorial*. Retrieved from Using Alternatives in CDI Applications: http://docs.oracle.com/javaee/6/tutorial/doc/gjsdf.html

1 Einleitung

„From Devices to Services“ ist das Motto dieses Projekts. Dienste (Engl. Services) sind im Internetalltag kaum mehr wegzudenken und sind für uns wichtiger denn je. Sie erleichtern unsere täglichen Aktivitäten, informieren uns überall die Dinge, die uns wichtig sind und manchmal retten sie uns mit einer Erinnerung den Tag. Doch so wirklich sind sie in unserem Alltag noch nicht angekommen. Jedenfalls nicht außerhalb des Internets. Um diesen Zustand zu ändern müssen wir verschiedene Geräte zu Diensten agieren, welche man im täglichen Leben einsetzen kann. Dazu haben sich bereits viele etablierte Hersteller Gedanken gemacht, die genau diesen Missstand auflösen sollen. Die Produktpalette ist mittlerweile sehr breitgefächert. Von Fensterschließern, die bei Regen automatisch das Fenster verriegeln über Wettersensoren, die jegliche Zustände übermitteln und darauf reagieren können. Sogar bei einem Leck der Waschmaschine, könnte der Hauptwasserhahn automatisch geschlossen und der Hausbesitzer per SMS informiert werden. Diese Systeme sind bereits ein enormer Schritt in die richtige Richtung, doch leider haben sie den Nachteil, dass sie untereinander nicht kompatibel sind. Hat man sich einmal für einen Hersteller entschieden, dann ist man wohl oder übel für die Zukunft an ihn gebunden. Es sei denn man freundet sich mit zwei separaten Systemen an. Der „Service“-Gedanke ist dann aber wieder nur in eingeschränkter Form vorhanden.

Wir haben uns als Ziel gesetzt ein System zu entwerfen, das in der Lage ist, all diese Geräte von verschiedenen Herstellern zu verstehen, zu verarbeiten und zu verwalten. Jedes Gerät soll unabhängig vom Protokoll oder vom Hersteller mit einem anderen Gerät kommunizieren und interagieren können, sodass jedes Gerät am Ende einen Dienst bereitstellt. Im Prinzip entwickelte sich mit der Zeit aus diesem Ziel eine eigenständige Hausautomatisierungslösung die ähnlich wie bei anderen Herstellern über eine Oberfläche kontrolliert werden kann. Mit dem großen Vorteil der Heterogenität. Im Rahmen des Projekts sind dabei die drei Hersteller Homematic, EnOcean und Philips Hue implementiert worden. Der Forschungsaufwand, die Umsetzung und die Probleme auf die wir gestoßen sind, werden im Folgenden genauer erläutert.

2 Smarthome – Stand der Technik

Unter Smarthome versteht man die Gesamtheit an technischen Systemen, welche innerhalb des Wohn- und Arbeitsumfelds die Lebensqualität der dort anwesenden Personen erhöht. Dies kann durch eine Vielzahl von verschiedenen Möglichkeiten geschehen. Um ein praktisches Beispiel zu nennen:

Wenn ein Mitarbeiter zwischen 7:00-9:00 seine eigene Wohnung verlässt, wird automatisch über das Internet eine Nachricht an das Smarthome System seines Betriebes versendet. Diese Nachricht beinhaltet, dass der Mitarbeiter auf dem Weg zu einem Arbeitsplatz ist. Durch definierte Regeln, welche der Mitarbeiter selbst festlegen kann, wird z.B. sein Arbeitscomputer gestartet und er muss nicht noch den Startvorgang abwarten, wenn er im Betrieb angekommen ist.

Smarthome ist nicht mehr nur ein Begriff aus der Forschung. Es hat längst Einzug in Wohnungen und Einfamilienhäuser erhalten, da das Preis-Leistungs-Verhältnis auch für die Konsumenten in einem angenehmen Rahmen passt. Auch die Gesellschaft lernt nach und nach die besseren Möglichkeiten der Smarthome Technik kennen und zu nutzen.

Nachdem Smartphone wird demnächst in viele Haushalte die Smarthome Technik Einzug erhalten.

2.1 IST-Zustand

Der Markt der SmartHome-Anbieter ist sehr heterogen aufgebaut. Das bedeutet, dass die System-Lösungen nicht miteinander arbeiten können, sondern nur jeder Anbieter seine eigenen Geräte

unterstützt. Dadurch ist es nicht möglich verschiedene Geräte unterschiedlicher Hersteller über eine dritte unabhängige Plattform zu koppeln.

Das bedeutet für den Konsumenten, dass er für jeden Hersteller ein eigenes Gateway und vermutlich auch einen eigenen Server/Computer benötigt, um die Geräte effektiv anzusteuern. Einige Anbieter können auch über eine handelsübliche Fritzbox[1] angesteuert werden.[2]

Zu den zentralen Anbietern auf den Markt werden RWE[3], Homematic[4] und Enocean[5]. Diese bieten eine große Anzahl von Geräten an, welcher mit ihrem Gateway oder auch CCU (Homematic) gekoppelt werden können. Als kleiner Anbieter auf dem Markt existiert noch Philips mit seiner Smarthome-Licht-Lösung Philips HUE[6]. Es gibt noch weitere Hersteller, jedoch werden sie im Rahmen dieses Projektes nicht weiter betrachtet.

2.2 Rahmenbedingungen

Die Rahmenbedingungen unseres Projektes sind, dass wir nur Geräte nutzen, welche über einen Gateway verfügen. Dieser muss mittels TCP/IP Protokoll erreichbar sein. Das Kommunikations-Protokoll, was dieser Gateway spricht muss nur mit Java bearbeitbar sein. So kann die Kommunikation per Sockets, XML RPC oder auch Webservices hergestellt werden.

Das System „Smarthome – From Device To Service“ wird mit Java Enterprise Edition (JAVA EE) implementiert. Als Entwicklungsumgebung wird die NetBeans IDE benutzt.

Als Applicationserver wird der Glassfish-Server 3.3 verwendet, da es bei einigen JAVA EE 6 Komponenten Problem mit dem Glassfish-Server 4.0 gibt.

3 Konzeption

Die Konzeption basiert auf der Entscheidung, dass drei Smarthome Hersteller als Startmodule bestimmt wurden. Dabei handelt es sich um

- EnOcean
- Homematic
- Philips Hue

Diese drei Hersteller agieren auf völlig unterschiedlichen Protokollen. Dies ist wichtig, um die Flexibilität des entstandenen Systems aufzuzeigen.

3.1 Anforderungen

Die grundlegende Anforderung des gesamten SmartHome Systems ist die Abstrahierung verschiedener Hersteller auf ein einheitliches REST-Protokoll. Dabei soll eine starke Modularisierung der einzelnen Hersteller erfolgen, so dass weitere Projekt Teams dieses Projekt aufgreifen können.

Für die Präsentation auf der Softwareprojekt Messe sollen verschiedene Testfälle mit unterschiedlichen Herstellern abgedeckt werden. Diese Geschäftsfälle sind die folgenden

1. Ventilator bei Temperatur aktivieren

[1] http://www.avm.de/de/Produkte/FRITZBox/index.php
[2] http://www.heise.de/netze/meldung/Neue-Firmware-Fritzboxen-werden-zu-Smarthome-Zentralen-1769896.html
[3] http://www.rwe-smarthome.de/web/cms/de/448330/smarthome/
[4] http://www.homematic.com/
[5] http://www.enocean.com/de/home/
[6] https://www.meethue.com/de-DE

2. Ventilator bei Wetterdaten aktivieren
3. Lampe bei Helligkeitsgrad einschalten
4. Stromverbrauch analysieren
5. Lampen per Mensa-Card aktivieren und

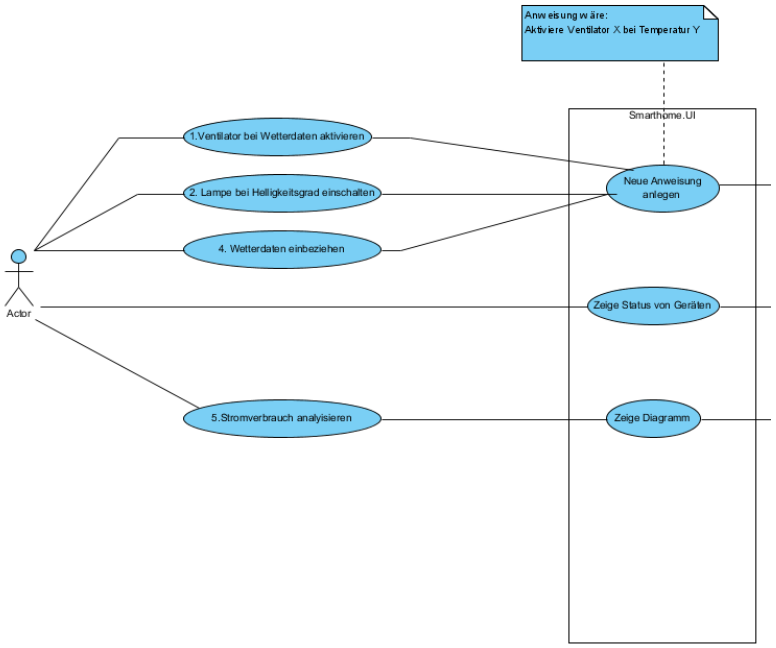


Abbildung 1: Use-Case Diagramm Ausschnitt

Das komplette Use-Case Diagramm befindet sich im Anhang.

3.2 Vorgehensmodell

Das Projektteam ist grob nach dem Vorgehensmodell von OpenUp[7] vorgegangen. Dabei gab es auch in der Abschlussphase Einflüsse von Agilen Vorgehensmodellen.

Die Rollen, welche mittels OpenUp verteilt werden müssen waren die folgenden:

- Projektmanager: Philip Schöppe:
- Analyst: Team
- Architekt: Tobias Münch
- Developer: Team
- Stakeholder: Daniel Kümper, Eike Reetz, Prof. Dr. Tönjes

Bei der Bezeichnung „Team“ waren alle Teammitglieder an diesen Punkt beteiligt. Die Anforderungen wurden meist in den Meetings analysiert. Da die Anforderung seitens des Software Engineering Moduls war, dass jeder Student implementieren soll, waren wir alle Entwickler.

4 Architektur

Die Architektur des Softwareprojektes ist in die Schichten Translation, Middleware und UI aufgeteilt. Diese kommunizieren intern über festdefinierte REST-Schnittstellen. Dabei können die Schichten beliebig verteilt werden. Jede einzelne Schicht ist selbst wieder in verschiedene Schichten aufgeteilt.

[7] http://epf.eclipse.org/wikis/openup/

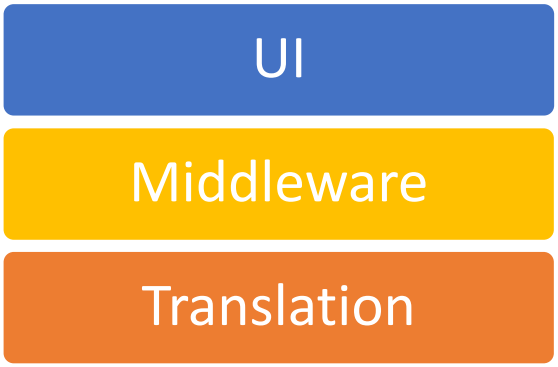


Abbildung 2: Aufbau von SmartHome - From Device to Service

Die Translationsschicht hat die zentrale Aufgabe die verschiedenen Module zu abstrahieren. Dabei ist ein Modul speziell auf einen SmartHome Hersteller zugeschnitten. Die Schicht bietet Schnittstellen zum Auslesen und Setzen von Sensoren bzw. Aktoren, CRUD (Create, Read, Update, Delete) Operationen für Gateways und die Registrierung eines dritten Dienstes. Dieser dritter Dienst ist bei unserer Architektur die Middleware, könnte aber auch ein anderer beliebiger REST-Dienst sein.

In der Middleware befindet sich die zentrale Regelsteuerung mit Drools. Dabei sind Fachlogik und Programmlogik in der Middleware strikt getrennt. Die Fachlogiken können per REST-Interface erstellt werden. Außerdem bietet die Middleware Schnittstellen zum Registrieren von weiteren Diensten, Abrufen von Aktoren und Sensoren und CRUD-Operationen für Gateways. Letztere Optionen werden zusammen mit der Translationschicht bearbeitet.

Das User-Interface ist die Schicht, wo das SmartHome System direkt mit dem User interagiert. Dabei greift das User-Interface auf die Middleware zu, um alle nötigen Information zur Steuerung des Systems zu bekommen. Das User-Interface soll nicht direkt auf die Translationschicht zugreifen.

4.1 Translationschicht

Die Translationschicht ist die grundlegende Schicht des Systems. Sie übersetzt verschiedene Hersteller in unser eigenes Protokoll. Durch eine starke Abstrahierung und Modularisierung mit dem Plug-In Pattern von Adam Bien[8] lassen sich neue Module schnell integrieren.

Die Translationschicht ist in die Schichten Webservice, Facade, Controller, Abstrahierung und die einzelnen Module aufgeteilt.

Die Abstrahierung geschieht dabei vorwiegend in der Klasse *DeviceAbstract*, welche per Reflection sowohl Aktoren als auch Sensoren mit den dazugehörigen Werten ermittelt. Der zweite Teil der Abstrahierung, nämlich der Aufruf von Methoden, wir über den *DeviceController* gesteuert.

Die Controller sind zentrales Steuerungselement der Translationschicht. Im groben gibt es vier primäre Controller. Darunter fallen *MiddlewareController*, *DeviceController*, *GatewayController* und der ConfigurationProviderController. Jeder einzelne Controller hat sein eigenes Aufgabengebiet mit den dazugehörigen Klassen. Die Kommunikation bei allen Controllern bis auf den *MiddlewareController* ist eingehend. Der *MiddlewareController* unterstützt auch die ausgehende Kommunikation in Richtung der Middleware.

[8] http://www.adam-bien.com/roller/abien/entry/building_plug_in_with_java

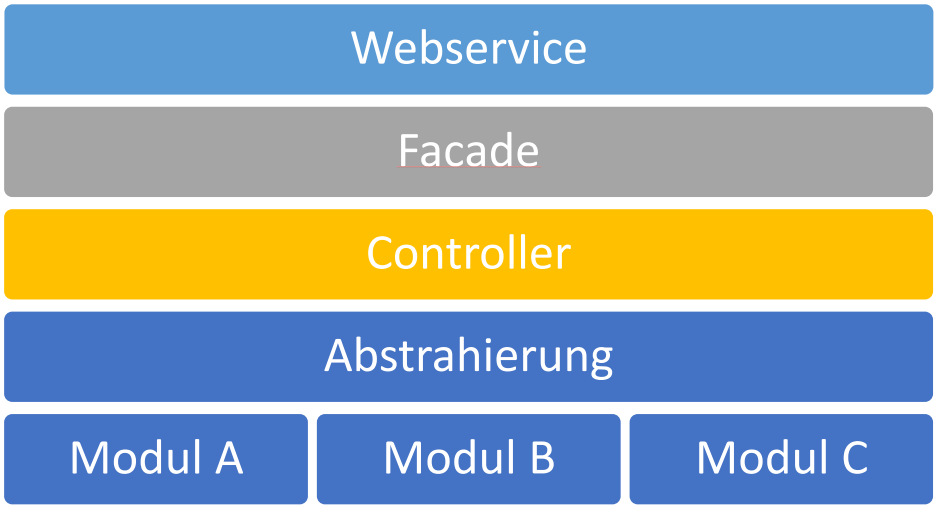


Abbildung 3: Aufbau der Translationschicht

4.1.1 Module

Ein Modul besteht aus den drei grundlegenden Element Configurationprovider, Gateway und Device. Jedes einzelne Element hat seine eigenen Aufgaben.

Der Configurationprovider enthält die Metainformationen zu einem Modul und dient dazu, dass Module sinnvoll in der Translationschicht registriert werden können. Es enthält die Informationen über Bezeichnung, Beschreibung und einen eindeutigen Typ. Der eindeutige Typ wird benötigt, um neue Gateways eines Configurationsproviders hinzuzufügen.

Jedes Modul hat einen Gateway. Dieser Gateway ist per TCP/IP zu erreichen und kann entweder per IP/Port oder per Connectionstring angesteuert werden. Das Gateway stellt dabei die Verbindung zu den einzelnen Geräten her. Die Geräte, im folgenden Devices genannt, haben Aktoren und Sensoren, welche per Annotation gekennzeichnet werden. Die wichtigsten Statusinformationen können persistiert werden. Die Devices werden beim Anfügen eines Gateways identifiziert und mit den aktuellen Statusinformationen persistiert.

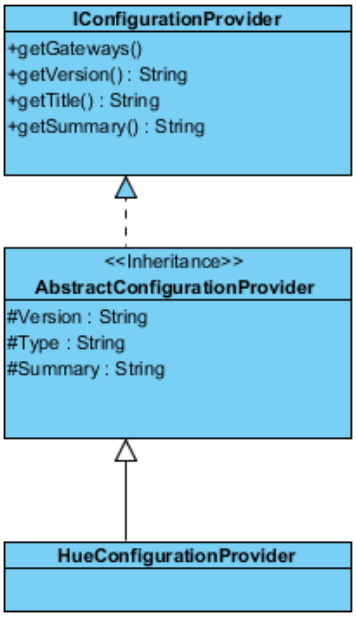


Abbildung 4: Configurationprovider Vererbung

4.1.1.1 Das eigene Modul

Zunächst erstellt man ein Maven EJB-Module und fügt die Abhängigkeit zum Smarthome.Global Projekt hinzu. Dadurch hat man Zugriff auf die wichtigsten schichtenübergreifenden Klassen und Schnittstellen.

Bei ihren primären eigenen Klassen, welche von *ConfigurationproviderAbstract*, *GatewayAbstract* und *DeviceAbstract* erben, muss darauf geachtet werden, dass folgende Annotationen vorhanden sind:

- @Entity
- @DiscriminatorValue(„ihre Kürzel“)
- @XmlRootElement

Damit ein eigenes Modul erfolgreich von der Translationschicht identifiziert werden kann, muss der spezifische Configurationprovider von der abstrakten Klassen *ConfigurationProviderAbstract* erben. Der überladen Default-Konstruktor sollte die grundlegenden Informationen Bezeichnung, Beschreibung, Version und Typ eindeutig setzen. Ferner müssen die Funktionen *addGateway* und *getNewGateway* implementiert werden. Die erste Funktion dient dabei zum Hinzufügen des Gateways zum Provider. Die Letztere dafür einen neuen Gateway zu instanziieren. (vgl. Abbildung 4: Configurationprovider Vererbung)

Zum Erstellen eines eigenen Gateways muss die spezifische Klasse von *GatewayAbstract* erben. Dabei müssen die Methoden *refreshDevices()* und *searchNewDevices()* überschrieben werden(vgl. Abbildung 5: Gateway Vererbung).

Bei den Devices erben sie analog zu den vorhergehenden Klassen von *DeviceAbstract*. Dabei müssen lediglich die *isOnline()* Methode überschreiben werden. Die Klasse selbst kann beliebig angepasst werden, dabei muss dabei auf diese Punkte geachtet werden:

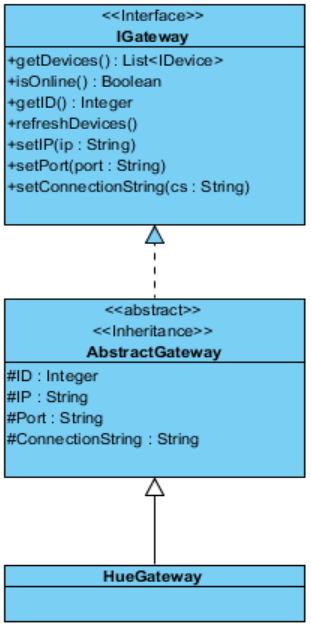


Abbildung 5: Gateway Vererbung

- Aktormethoden mit @Aktor annotieren
- Sensormethoden mit @Sensor annotieren

Bei der Annotation @Aktor können sie die Parameter Status, RegEx, Minimum und Maximum übergeben. Diese werden durch Reflection einem beliebigen Client per Rest mitgeteilt. (vgl. Abbildung 6: Vererbungshirachie des Devices)

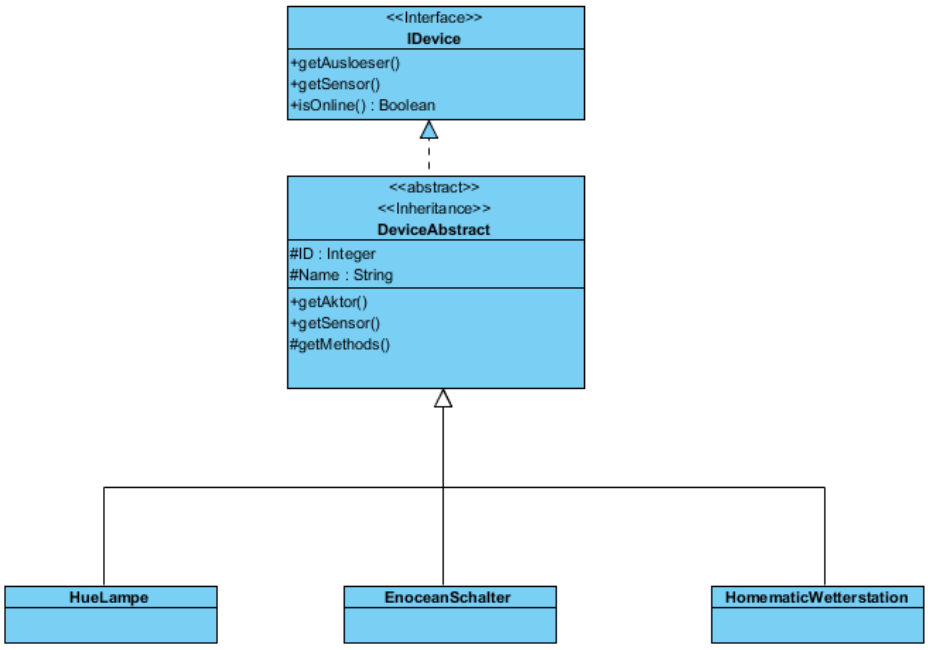


Abbildung 6: Vererbungshirachie des Devices

4.1.2 Modul: Philips HUE

Das Philips Hue System stellt eine Beleuchtungsmöglichkeit dar, die sich mit bis zu 50 einzelne Leuchtmittel individuell steuern lässt. Im Folgenden soll die Funktionsweise diese Systems aufgezeigt und Verwendung innerhalb des Projektes mit Hilfe von UML-Diagrammen erläutert werden. Um

diese Technologie nutzen zu können, wird eine Philips HUE Bridge und mindestens eine Philips HUE Lampe benötigt.[9]

4.1.2.1 Funktionsweise

Das Philips HUE System verfügt über ein Application-Programming-Interface mit dem die einzelnen Leuchten per REST-Befehle im Netzwerk angesprochen und gesteuert werden können. Diese Kommunikation geschieht zunächst nicht direkt mit den Lampen, sondern über die Philips Hue Bridge. Die Verbindung zwischen Leuchten und Bridge wird durch das ZigBee Light Link Protocol[10] realisiert welches auf einer Frequenz von 2,4 Mhz sendet. Der jeweilige Satus einer Lampe wird zudem in der Bridge gespeichert und nicht bei jeder Statusabfrage direkt von der Lampe eingeholt. Die Bridge ist somit das einzige Gerät des Systems, dass eine direkt Verbindung zum Netzwerk des Endanwenders benötigt.

Anstehend soll die Funktionsweise eines REST-Aufrufes anhand eines Beispiels verdeutlicht werden. Request- und Responsewerte werden hierbei als JavaScript-Object-Notation übertragen.

Adresse	http://<bridge-IP>/api/<username>/lights/1/state
Request-Body	{“on”: true}
Methode	PUT
Beispiel-Response	{“success”: {“/lights/1/state/on”: true}}

Tabelle 1:Rest Philip's HUE

Die Adresse unter der der PUT-Befehl erfolgt beinhaltet zum einen die IP-Adresse und zum anderen den Usernamen. Dieser muss zuvor an der Bridge zusammen mit einem Gerätenamen registriert werden. Die Philips HUE Dokumentation spricht hierbei von der Whitelist-Permission. Der darauf folgende Pfad weist an, dass bei der Lampe mit der ID: 1 der Satus verändert werden soll. In diesem Beispiel wird das erfolgreiche Einschalten einer Lampe durch die dargestellte Response bestätigt.

Um den vollen Funktionsumfang zu verstehen, wird an dieser Stelle auf die Philips HUE API Dokumentation verwiesen.[11]

4.1.2.2 Installation und Konfiguration

Um Philips HUE verwenden zu können, ist lediglich sicherzustellen, dass die Bridge im jeweiligen Netzwerk erreichbar ist. Weitere Konfiguration wie die Whiteliste-Permission wird programmierseitig umgesetzt.

4.1.2.3 Einbindung von Philips HUE in die Translationschicht

Wie bereits aus dem Kapitel der Architektur bekannt, müssen die Klassen *AbstractConfiguraionProvider*, *AbstractGateway* und *DeviceAbstract* im speziellen Modul geerbt und deren Methoden umgesetzt werden. Da in diesem Modul nur eine Art von Geräten (Lampen) realisiert werden muss, kann auf eine weitere Generalisierung verzichtet werden. Die Klasse *HueLampe* spiegelt dabei eine Lampe und deren Funktionen wieder.

[9] https://www.meethue.com/de-DE/getstarted
[10] http://www.zigbee.org/
[11] http://developers.meethue.com/

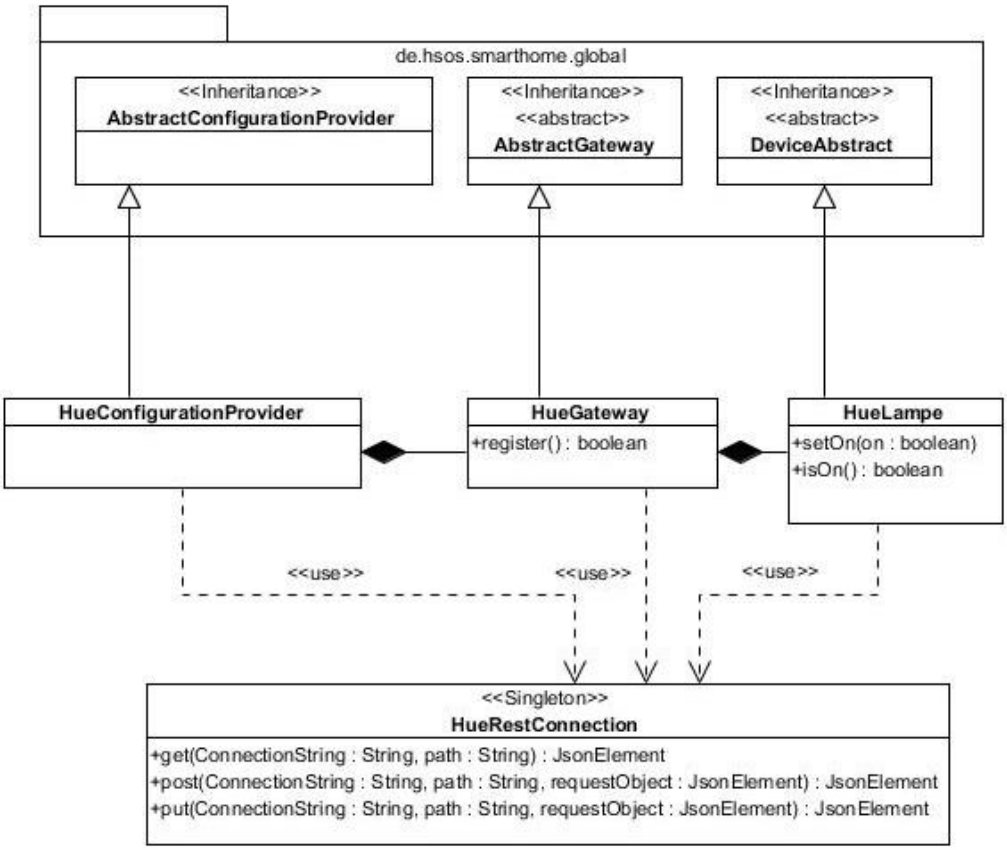


Tabelle 2: Modul Philip's HUE

Wie aus dem Diagramm hervor geht, wird ein REST-Client benötigt um die Kommunikation mit der Bridge sicherzustellen. Dies wird durch das Singleton *HueRestConnection* umgesetzt, welches die HTTP-Aufrufe GET, POST und PUT implementiert. Weiterhin sorgt das HueGateway durch die Methode *register()* für die Whitelist-Permission. Bei einer Erstverwendung einer Bridge im Zusammenhang mit diesem Projekt muss hierzu der Lern-Button an der Bridge gedrückt werden bevor das HueGateway der Translation hinzugefügt wird. Da die Bridge den Status einer Lampe hält, wird zudem eine Synchronisation der realen Lampen mit den persistierten HueLampen vorgenommen.

4.1.2.4 Probleme

Die aktuellen Firmware der Hue Bridge unterstützt noch nicht die Funktion, die prüft, ob eine Lampe noch erreichbar respektive noch vorhanden ist. Diese Tatsache kann zu Problemen führen, wenn eine Lampe nicht mehr an das System gekoppelt sein soll, da sie immer noch der Bridge bekannt ist. Der momentane Ausweg sieht daher nur ein Reset der Bridge vor und ein anschließendes neu Hinzufügen des Gateways.

4.1.3 Modul: Homematic

Die in der Implementierung verwendeten Geräte umfassen eine Wetterstation, eine steuerbare Steckdose und die zum Ansprechen der Geräte benötigte CCU.

Abbildung 7:Homematic Logo

4.1.3.1 Die Grundlagen

Als Grundlage für die Implementierung wurde ein bereits im Internet zu findendes Programm von Oliver Wagner benutzt, welches den Namen "HM Companion" besitzt[12] . Dieses Programm wurde in Java geschrieben und verdeutlicht, wie die CCU mit den Geräten kommuniziert. Es wurden also lediglich die dafür verantwortlichen Codezeilen für unsere Zwecke herausgefiltert und ins Projekt integriert. Diese Klassen befinden sich im Package *de.hsos.smarthome.translation.module.homematic.function.connection*.

4.1.3.2 Struktur der CCU

Um den grundsätzlichen Aufbau zu verstehen, müssen zunächst die Grundlagen der CCU erklärt werden. Die CCU verfügt über drei Schnittstellen, über die Geräte verbunden sein können: Für Funk (BidCos-RF) mit der Portnummer 2001, für drahtgebundene Geräte (BidCos-Wired) mit der Portnummer 2000 und für interne Geräte mit der Portnummer 2002.

Jeder dieser Schnittstellen besitzt wiederum eine XML-RPC Schnittstelle, über welche die Geräte angesteuert und Statusmeldungen abgefragt werden können.

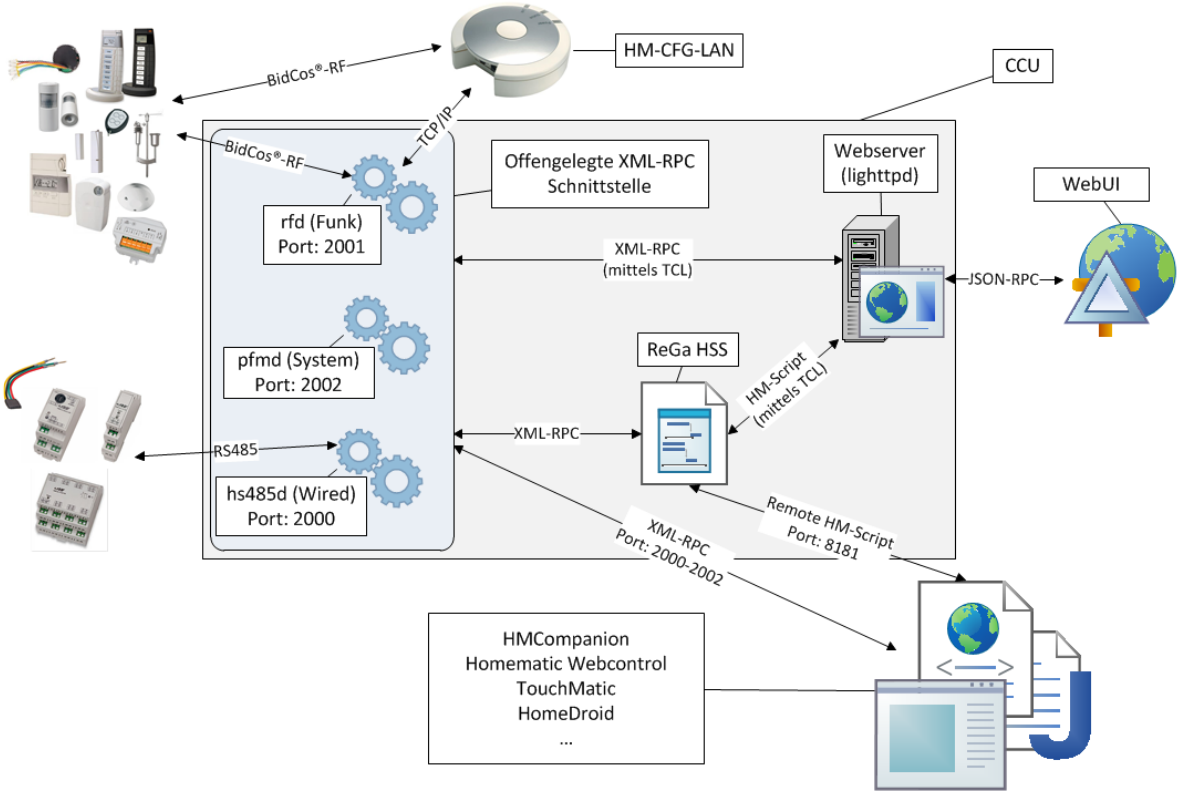


Abbildung 8: Struktur der CCU (http://3.bp.blogspot.com/-quD2x2dFIRM/TxXlDH-F4RI/AAAAAAAAA6I/BOl7FLjs0sw/s1600/Homematic_Aufbau.png)

4.1.3.3 Das Anlernen der Geräte

Damit die CCU mit den Geräten kommunizieren kann, müssen sie ihr zunächst bekannt gemacht werden.

Zum Anlernen der Geräte muss die CCU in den Anlernmodus versetzt werden. Dazu wird die Minus-Taste (oder die Plus-Taste), die sich direkt an der Station befindet, etwas länger gedrückt. Bei den Geräten gibt es Unterschiede, um sie anzulernen. Bei der von uns verwendeten Steckdose muss der kleine Knopf einmal gedrückt werden. Die Wetterstation benötigt eine etwas kompliziertere

[12] https://github.com/owagner/hmcompanion

Anlernphase. Zunächst muss der Windrichtungspfeil nach Norden ausgerichtet werden. Dann muss mit einem spitzen Gegenstand der sich bei dem Batterienfach befindende schwarze Knopf gedrückt werden. Währenddessen müssen die vier Batterien eingelegt werden. Nach 60 Sekunden läuft die Zeit zum Anlernen an der CCU ab und am Bildschirm kann abgelesen werden wie viele neue Geräte die CCU erkannt hat. Einmal angelernt bleiben die Geräte bei der CCU verankert. Nur wenn man die CCU zurücksetzt, werden auch die Geräteinformationen verworfen.

4.1.3.4 XML-RPC-Schnittstelle

Zur Kommunikation mit der CCU wird die in ihr bereitgestellte XML-RPC-Schnittstelle genutzt. Hierzu wird innerhalb der Klasse *HomematicDevice* zuerst eine Verbindung erstellt. Zum Herstellen dieser wird ein Socket angelegt, dem dabei die Adresse der CCU und der Port, über dem mit ihr kommuniziert werden soll, mitgeteilt wird. Da in unserer Software nur kabellose Geräte genutzt werden, trägt dieser Port die Nummer 2001.

Über die nun bereitstehende Verbindung können RPC-Requests an die CCU gesendet werden. Dies wird über die Klasse *HomematicMessage*, welche ihren Ursprung im "HM Companion" hat, realisiert. Diese stellt den String "methodname" bereit, welcher festlegt welche Art von Anfrage an die CCU gesendet wird. Des Weiteren steht das Array "args" bereit, welches die bei manchen Anfragen nötigen Parameter enthält. Welche Befehle dabei möglich sind wird natürlich von der CCU festgelegt. Zur Übermittlung der so erstellten Message wird nun die Funktion *sendRequest(),* ebenfalls ursprünglich in "HM Companion" zu finden, der Klasse *HomematicDevice* genutzt. Innerhalb dieser wird die Anfrage zuerst an die Hauptsteuerungseinheit gesendet und dann auf eine Antwort gewartet.[13] In unserem Programm werden folgende Arten von Anfragen genutzt:

4.1.3.4.1 getParamsetDescription

Diese Funktion der CCU ist zwar nicht direkt im Code des Projekts zu finden, war aber dennoch während der Entwicklung sehr wichtig, da mit ihm abgefragt werden kann, welche Werte für die jeweilige Device verfügbar sind und somit abgefragt werden können. Die benötigten Parameter sind die Adresse des Gerätes sowie der Name des Paramsets das von Interesse ist. Im Falle der hier verwendeten Devices war meist das "VALUES" Paramset relevant, da dieses die zur Funktion wichtigen Variablen enthält.

4.1.3.4.2 system.listMethods

Auch diese Funktionalität wird nicht direkt im Programm verwendet, gab uns aber die Möglichkeit uns einen Überblick über die Methoden der CCU zu verschaffen. Bei Übergabe des Befehls gibt sie eine Liste der unterstützten Methoden zurück.

4.1.3.4.3 getValue

Über diese Funktion der CCU kann der aktuelle Wert einer Variablen auf einem bestimmten Gerät abgefragt werden. Dabei werden als Übergabeparameter die Adresse der betreffenden Device sowie der Name des anzufragenden Wertes benötigt. Im Falle der verwendeten Wetterstation lautet die Adresse "HM-WDS100-C6-O", die Steckdose hingegen trägt die Bezeichnung "HM-LC-SwX". Die Antwort der CCU wird dann in eine HomematicResponse umgewandelt und kann ausgelesen werden. In unserem Programm wird seitens der Steckdose ausschließlich die Variable "STATE" abgerufen. Diese gibt ob sie an- oder ausgeschaltet ist. Die Wetterstation hingegen besitzt deutlich mehr nutzbare Informationen. So werden unsererseits die Temperatur, Helligkeit, Luftfeuchtigkeit, Regen,

[13] http://www.eq-3.de/Downloads/PDFs/Dokumentation_und_Tutorials/HM_XmlRpc_V1_502__2_.pdf

Regenmenge, Sonnenscheindauer, Windrichtung, Windrichtungsschwankung und Windgeschwindigkeit abgefragt. Außerdem kann über diesen Befehl und den Parameter “UNREACH” überprüft werden, ob die betreffende Device noch erreichbar ist.

4.1.3.4.4 setValue

Dieser Befehl funktioniert umgekehrt zu getValue, da hier ein bestimmter Wert nicht abgerufen, sondern auf der CCU gesetzt wird. Als zusätzlicher Parameter wird hierbei der gewünschte Wert übergeben. Innerhalb der Steckdose wird er beispielsweise dazu genutzt, diese ein- oder auszuschalten.

4.1.3.4.5 listDevices

Bei Empfang dieser Abfrage gibt die CCU eine Liste der ihr bekannten Geräte zurück. Diese können dann aus der HomematikResponse ausgelesen werden.

4.1.3.5 Die Geräte

Um sicherzustellen, dass die versendeten Befehle auch verarbeitet werden können, besitzt jedes Gerät die Funktion *isOnline()*, die überprüft, ob es zum jeweiligen Zeitpunkt über die CCU angesprochen werden kann. So wird auch verhindert, dass stark veraltete Wetterwerte abgefragt werden ohne dass der Nutzer dies bemerken würde.

4.1.3.5.1 Wetterstation

Die verwendete Wetterstation ist in der Lage bestimmte Parameter ihrer Umgebung zu registrieren und zu speichern. Hierbei kann die Temperatur, Helligkeit, Luftfeuchtigkeit, Regen, Regenmenge, Sonnenscheindauer, Windrichtung, Windrichtungsschwankung und Windgeschwindigkeit gemessen werden.

Abbildung 9: Homematic Wetterstation

Die zum Abfragen und Verarbeiten dieser Werte benötigten Funktionen werden in der Klasse *Weatherstation* bereitgestellt. Die hierbei wichtigste Funktionalität stellt die Funktion *setValues()* dar. Diese erstellt und versendet die benötigten RPC-Abfragen um so die auf der CCU zwischengespeicherten Werte abzufragen. Zu beachten ist hierbei, dass die Wetterstation über Batterien betrieben wird und deswegen nicht die gerade auf ihr befindlichen Daten abgefragt werden können. Diese Abfrage findet unbeeinflussbar seitens der CCU statt und wird alle 120-180 Sekunden durchgeführt. Hierdurch kann es gerade während einer Demonstration der Software zu einer Verzögerung beim Erkennen neuer Werte kommen. Auch das Versenden von Events seitens der CCU ist nicht möglich, weshalb wir dazu gezwungen waren eine Art Scheduler einzurichten, der alle 30 Sekunden den aktuellen Wert abfragt, speichert und seinerseits ein Event verschießt falls sich etwas geändert hat. Eine weitere Besonderheit ist, dass die Daten über Regenmenge und Sonnenscheindauer nicht den Wert des aktuellen Tages darstellt, sondern die Gesamtanzahl der Sonnenstunden beziehungsweise der Regenmenge seit dem Aufstellen der Wetterstation wiedergibt. Aufgrund dieser Tatsache wurde ein weiterer Scheduler implementiert, der jede Nacht um 0 Uhr den aktuellen Wetterwert in ein dafür bereit stehendes Array schreibt. Auf diese Weise kann innerhalb der UI der für den aktuellen Tag geltende Wert abgefragt werden, indem

der Wert des Vortags vom aktuellen abgezogen wird. Außerdem bietet sich so die Möglichkeit zur Darstellung von Diagrammen und ähnlichem, die den Wetterverlauf der letzten Tage anzeigen. Des Weiteren bietet diese Klasse selbstverständlich die Möglichkeit all diese Variablen per Get-Funktionen abzufragen.

4.1.3.5.2 Steckdose

Die verwendete Steckdose ist in der Lage die Stromzufuhr zu den an sie angeschlossenen Geräten ein- oder auszuschalten.

Die dazu notwendigen Funktionen werden innerhalb der Klasse Outlet bereitgestellt. So ist es durch die Funktion *setOn()* möglich den Wert "STATE" der Steckdose zu ändern und so auch Einfluss auf die Stromzufuhr der angeschlossenen Geräte zu nehmen. Außerdem kann selbstverständlich der aktuelle Zustand der Steckdose abgefragt werden indem die Funktion *isOn()* genutzt wird. Das alles wurde in die Klasse *Outlet* implementiert.

4.1.3.5.3 Das HomematicGateway

Die Klasse *HomematicGateway* erweitert die Klasse *GatewayAbstract* um die Funktionen die nötig sind, um die einer CCU bekannten Geräte der Liste der in der Software bekannten Devices hinzuzufügen. Die wichtigste dieser Funktionalitäten ist *searchNewDevices().* Diese ruft zunächst die Funktion *giveDevices()* auf, welche an der CCU anfragt welche Geräte dieser bekannt sind und diese dann als Array zurückgibt. Die Einträge dieses aus Strings bestehenden Arrays werden dann nach den Abschnitten durchsucht die die verwendeten Geräte identifizieren. Im Falle der Wetterstation lautet dieser "HM-WDS100-C6-O", die Steckdose trägt die Kennung "HM-LC-SwX". Entsprechend der gefundenen Devices wird dann die jeweilige Add-Funktion aufgerufen, die ein Gerät der jeweiligen Art zu der Deviceliste des Gateways hinzufügt.

4.1.3.5.4 Weitere Geräte implementieren

Um nun ein weiteres Homematic-Geräte in unser System einzuspeisen, muss für dieses eine eigene Klasse erstellt werden und die zu überschreibenden Methoden implementiert werden. Mit Hilfe von "paramsetDescription" kann nun herausgefunden werden, welche Methoden dieses Device bietet. Zum Schluss braucht die Klasse HomematicDevice noch eine Methode, mit der sie auf das neue Device kommt und es muss natürlich auch noch wie in der Beschreibung angelernt werden.

4.1.4 Modul: EnOcean

Die EnOcean GmbH wurde als Spin-off der Siemens AG im Jahr 2001 in München gegründet. Sie ist als Hersteller und Erfinder der patentierten Grundlagentechnologie der batterielosen Funksensorik bekannt. Die Lösungen der EnOcean GmbH basieren auf miniaturisierten Energiewandlern, zuverlässiger Funktechnik und äußerst stromsparender Elektronik. Die Vereinigung dieser Bestandteile ermöglicht es, Sensorlösungen zu vertreiben, die essentiell für energieeffiziente Gebäude und innovative Industrietechnik sind. Funkmodule von EnOcean werden aktuell

Abbildung 10:Enocean Logo

weltweit von „über 100 Produktherstellern (EnOcean Alliance) von Systemlösungen für Gebäude- und Industrietechnik eingesetzt.“ [14]

4.1.4.1 EnOcean Technologie

Der Grundgedanke der von EnOcean genutzten Technologien besteht darin, dass für Senden und Empfangen von kurzen Funksignalen nur geringe Mengen an Energie nötig sein sollen. Um diese Idee zu gewährleisten, nutzen Sender und Empfänger von EnOcean Piezoelektrizität, Energie von Solarzellen, Peltier-Elemente oder die Bewegungsenergie von elektromagnetischen Energiewandlern.

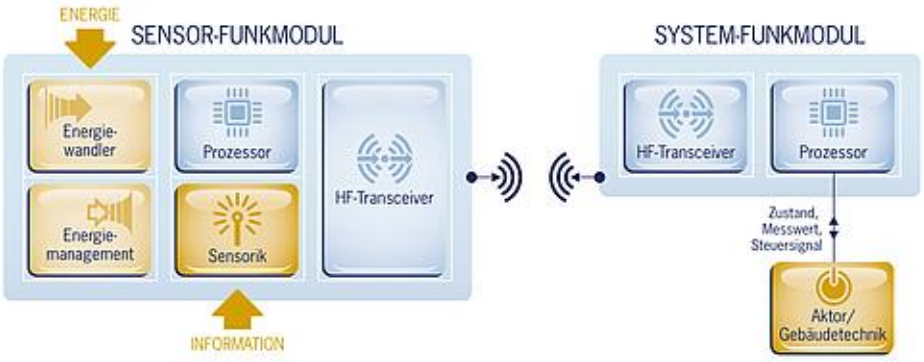


Abbildung 11:EnOcean Modulaufbau

Diese Energiebeschaffungsformen reichen in den meisten Fällen dazu aus, um Sender batterielos und wartungsarm zu betreiben. Falls jedoch keine ausreichenden Lichtverhältnisse oder mechanische Betätigung zu erwarten sind, können die genutzten EnOcean-Module durch Batteriebetrieb angetrieben werden.

Das von EnOcean-Modulen genutzte Funkprotokoll ist ebenfalls darauf ausgelegt, Informationen und Signale möglichst energiearm und mit hoher Zuverlässigkeit zwischen Sender und Empfänger zu transportieren. Hierzu werden Frequenzen von 868 MHz und 315 MHz (für Europa und international) verwendet.

Im März 2012 wurde das EnOcean-Funkprotokoll von der internationalen Electrotechnical Commission (IEC) mit ISO/IEC 14543-3-10 als internationaler Standard eingeführt.

Dieser Standard (ISO/IEC 14543-3-10 Information Technology - Home Electronic Systems (HES) - Part 3-10: Wireless Short-Packet (WSP) Protocol optimized for Energy Harvesting - Architecture and Lower Layer Protocols) arbeitet auf den OSI Schichten 1-3 also den Physical, Data Link und Network Layer.

Die Funkdaten, die zwischen Sender und Empfänger übertragen werden, sind mittels Verschlüsselung und Rolling Code gegen Zugriffe von Dritten geschützt. Des Weiteren wird das Übersenden eines „Datenpaketes von den originären EnOcean-Komponenten nur unter Verwendung festgelegter IDs erlaubt“.[15] Bei Eigenimplementation dieses Funktstandards ist dieser Schutz allerdings nicht gewährleistet.

Zur Kollisionsvermeidung beim Senden und Empfangen von Funksignalen wird versucht möglichst kleine und somit kurze Datenpakete zu versenden, um so Kollisionsbildungen zu umgehen. Um die Kollisionswahrscheinlichkeit weiter zu reduzieren, werden die gesendeten Telegramme innerhalb von 30 Millisekunden zweimal wiederholt.

[14] http://www.enocean.com/de/company-profile/
[15] http://www.enocean.com/de/company-profile/

Die Reichweite der EnOcean Funksensoren liegt bei 300 Metern im Freien und bis zu 30 Metern im Gebäudeinneren. Jedes EnOcean-Modul verfügt außerdem über einen universalen 32 Bit Identifikationsschlüssel, welcher die Kenntlichmachung einzelner Schalter gewährleistet.

4.1.4.2 Die Geräte

4.1.4.2.1 Thermokon STC-Ethernet-HS

Zur Kommunikation mit den einzelnen EnOcean-Komponenten wurde das STC-Ethernet-HS der Firma Thermokon verwendet. Das STC-Ethernet-HS ermöglicht hierbei die Wiedergabe und den Empfang von Funktelegrammen, welche von EnOcean Sensoren und Aktoren übermittelt werden und die dem EnOcean-Kommunikationsprotokoll entsprechen. Dazu wird eine Kommunikation über TCP/IP oder UDP genutzt.

Abbildung 12: Thermokon STC-Ethernet HS

4.1.4.2.2 SR-MDS EnOcean Funk-Decken-Multisensor

Der EnOcean SR-MDS Funk-Decken-Multisensor dient zur Helligkeitsmessung in Wohn- oder Büroräumen, zur Bewegungserfassung und zur Temperaturerfassung innerhalb eines Raumes.

Abbildung 13: Enocean Funk-Decken Sensor

Hierbei beträgt die Erfassungsreichweite des Bewegungssensors volle 360 Grad. Außerdem nimmt der integrierte Helligkeitssensor Helligkeitswertänderungen im Bereich von 0-510 Lux wahr. Der Temperatursensor, der im SR-MDS verbaut ist, kann des Weiteren Temperaturen zwischen 0-51 Grad Celsius vermessen. Der Funk-Decken Multisensor arbeitet hierbei nach zwei verschiedenen Messprinzipien: Zum einen kann man per Tastendruck auf die Lerntaste des Gerätes oder bei Bewegungserkennung den Sensor „aufwecken“ und so per Telegramm Messdaten an den jeweiligen Empfänger schicken oder zum anderen den SR-MDS zeitgesteuert in bestimmten Intervallen „aufwecken“ und ihn dann Messdaten zusammentragen und zum Abnehmer schicken lassen. Ein solch bereits erwähntes Funktelegramm besteht aus neun Teilen.

Das Org-byte beschreibt hierbei den jeweiligen Gerätetyp “4BS“. Das Data_byte3 enthält die Batteriespannung des SR-MDS. Der Wertebereich liegt zwischen 0 und 255, wobei Werte, die größer als 120 sind, eine Batteriespannung im guten Bereich darstellen.

Das Data_byte2 beinhaltet die Helligkeitswerte 0-510 Lux, die linear n= 0 bis 255 vom Telegramm erfasst werden. Der Data_byte1 enthält die gemessenen Temperaturwerte zwischen 0 und 51 Grad Celsius. Die restlichen ID_bytes sind zur Bestimmung der Geräte ID vonnöten. [16]

[16] http://greenelectric.eu/content/de/thermokon/sr_mds/produktblatt-sr-mds1306421964.pdf

4.1.4.2.3 EnOcean EasySens Funkschalter 2 Kanal

Der EasySens Funkschalter ermöglicht die Steuerung von auf EnOcean-Technologien basierenden Empfängern und daran angeschlossene Verbrauchern.

Hierbei erfolgt die Energieerzeugung der batterielosen Funkschaltereinsätze durch einen wartungsfreien elektrodynamischen Energiegenerator.

Abbildung 14: EasySens Funkschalter

4.1.4.3 Aufbau des Modules

4.1.4.3.1 Einleitung

Im Folgenden werden die wichtigsten Komponenten des EnOcean Moduls vorgestellt und zum besseren Verständnis näher erläutert. Dazu wird besonders auf die Bestandteile Server, Transiever, Telegramm und das Anlernen von neuen Sensoren und Aktoren eingegangen. Diesbezüglich ist es wichtig, dass als Basis das TCM 120 Modul genutzt wird.[17]

4.1.4.3.2 Der Server

Der Server erstellt einen TCP-Socket, welcher auf Verbindungsanfragen vom STC-Ethernet Klienten reagiert. Hierfür wird auf Port 4000 gehorcht. Um die anderen Komponenten, insbesondere die Middleware, von den entgegengenommen Verbindungsanfragen in Kenntnis zu setzen, werden zwei Events bereitgestellt, welche aktiviert werden, wenn zum einen ein Transiever oder Receiver verbindet oder wenn alle Einstellungen des Transievers initialisiert sind. Hierzu werden ebenfalls zwei Eventhandler eingeführt, welche asynchron die Event-Daten verarbeiten.

4.1.4.3.3 Die Telegramme

Die Telegram-class ist die Basisklasse für alle Arten von Telegrammen. Sie enthält alle wichtigen Eigenschaften und Methoden zum Parsen und Serialisieren von Telegrammen jeglichen Formats, das für das TCM 120 Modul nötig ist. Weiterführende Telegrammtypen erben von dieser Klasse.

4.1.4.3.4 Lernmodus

Um dem Thermokon STC-Ethernet-HS neue Geräte kenntlich zu machen, muss man es in einen Lernmodus versetzen, um die neuen Geräte zu erfassen.

Hierzu wird die Klasse *SetLearnMode* genutzt, welche durch *createMessage()* ein Byte-Array von der Länge 10 erzeugt und hält, welches dann zum STC-Ethernet-HS geschickt wird, um dieses in den Lernmodus zu versetzen (Kenntlich durch eine grüne Lampe auf dem Thermokon STC-Thernent-HS). Während sich der Transiever im Lernmodus befindet, besteht die Möglichkeit die Geräte anzulernen. Bei einem Taster z.B. durch einen simplem Klick. Die Funktionsweise für die Geräte kann den entsprechenden Handbüchern entnommen werden.

4.1.4.3.5 Transiever

Der Transiever spielt als Gateway die zentralste Rolle. Nachdem die Verbindung zum Server aufgebaut wurde, leitet er alle Telegramme die von Sensoren und Aktoren eintreffen weiter. Die

[17] http://enoceannet.codeplex.com/

Kommunikation zwischen Transiever und EnOcean Geräten kann nicht beeinflusst werden. Auf dem TCP-Port werden also nur bereinigte und vollständige Telegramme übertragen. Kommt es vorher zu Fehlern, wird das Telegramm erst gar nicht weitergeleitet.

4.1.4.4 Probleme

Bei der Erstellung des Moduls EnOcean traten einige Probleme auf, welche im folgenden Kapitel näher beschrieben werden. Zu Beginn der Arbeiten an der Komponente EnOcean wurde das EDK 300- the Developer's Kit for EnOcean Dolphin Modules verwendet, welches stationäre Boards mit Sensoren über COM-Port mit unserer Software interagieren lassen sollte. Da diese Boards bei der Konfiguration der besagten COM-Ports aber für uns einige Schwierigkeiten hervorriefen, wurde diese Idee schnell durch die Nutzung des Thermokon STC-Ethernet-HS ersetzt. Hierbei stellt sich die Dokumentation für die Programmiersprache Java als weitere Hürde dar. Aufgrund weniger brauchbarer Quellen wurde schließlich ein C# Projekt gefunden, welches die Interaktion mit den einzelnen EnOcean Komponenten und dem Thermokon STC-Ethernet ermöglichte. Dazu musste allerdings die C# Anwendung in Java portiert werden, was sich durch die signifikanten Unterschiede sowohl in Bereichen wie Eventhandling als auch Threadbehandlung als schwierig darstellte.

Hinzu kam außerdem das Problem, dass man erst nach vollständiger Portierung des gesamten Projektes dieses auf Fehler testen konnte.

Des Weiteren stellte sich die Bytehandhabung der beiden Programmiersprachen C# und Java als unterschiedlich heraus, so nutzt C# zur Darstellung unsigned Bytes mit einem positiven Wertebereich, wohingegen Java ausschließlich signed Bytes unterstützt.

4.1.5 Die Abstrahierung

Wie bereits erwähnt wird die Abstrahierung per Reflection durchgeführt. Dabei wird die mit Java mitgelieferte Reflection Bibliothek genutzt. Durch unsere eigenen Annotationen können wir zentrale Elemente der Klassen identifizieren und weiterverarbeiten. Diese Funktion ist in der Klasse *DeviceAbstract* implementiert und unterteilt sich in zwei Methoden. Die eine Methode ist für die Sensoren zuständig, wohingegen die andere für die Aktoren zuständig ist. Die Funktionsweise soll durch folgenden Code-Snippet verdeutlicht werden:

```
    /**

     * Verarbeitet die Methoden einer bestimmten Annotation
     * @param annotationClass
     * @return
     */
    protected       HashMap<String,DeviceInformation>       getMethods(Class
annotationClass)
    {
        HashMap<String,DeviceInformation> ret = new HashMap();

        for(Method m: this.getClass().getMethods())
        {
            Annotation a = m.getAnnotation(annotationClass);

            if( a != null){
                List<Class> list = new ArrayList<>();
                list.addAll(Arrays.asList(m.getParameterTypes()));
                //Device Information erstellen
                DeviceInformation  n  =  new  DeviceInformation(m.getName(),
list.size(), list);

                 //Werte setzen Min,Max und Status
                setConstraints(annotationClass, a, n);
                ret.put(n.getKey(), n);
            }

        }
        return ret;
    }
```

Listing 1: Reflection der Aktoren & Sensoren

Durch diese Verarbeitung der einzelnen **Aktoren** und **Sensoren**, lässt sich unser spezifisches Device direkt in ein XML-Dokument überführen, welches alle Informationen zum ansteuern des Gerätes enthält.

4.1.6 Controller im Detail

Die Controller sind die zentrale Businesslogik der Translationschicht. Jeder Controller hat seinen eigenen Aufgabenbereich und ist strikt getrennt. Die Controller werden zu den Webservices hin durch eine BusinessFacade getrennt.

Der **DeviceController** hat die Aufgabe sich, um die bereits persistierten Geräte zu kümmern. Dabei wird der Aufruf, welche aus den Webservice Resources stammt, verarbeitet. Die Funktionen des DeviceControllers gliedern sich dabei in

1. Abruf aller Geräte
2. Abruf eines bestimmten Gerätes mittels seiner ID

3. Aufruf eines Aktors mit maximal zwei Parametern
4. Abruf eines Sensors

Beim setzen des Aktors werden die Parameter immer als String entgegengenommen und probiert intern zu marshallen. Falls dies nicht funktionieren sollte, wird eine Exception NoValidParameter geworden.

Der **GatewayController** stellt CRUD Operationen für die Gateways bereit. Dabei wird zunächst geprüft, ob das angegebene Modul existiert. Dabei spielt der Providername eine entscheidende Rolle.

Der **MiddlewareController** bearbeitet Registrierungsanfragen von unterschiedlichen Middlewares und persistiert diese. Man kann eine bereits registrierte Middleware über seinen eindeutigen Namen wieder aus der Datenbank entfernen. Weiterhin kümmert sich dieser Controller um die Kommunikation der bereits registrierten Middlewares. Dabei ruft er alle vorhandenen Middlewares aus der Datenbank ab und sendet die empfangenen Events weiter.

4.1.7 Eventkommunikation

Die Eventkommunikation innerhalb der Translationschicht arbeitet vorwiegend mittels CDI-Events. Der Vorteil an diesen Events ist es, dass sie überall gefeuert und an einer zentralen Stelle eingesammelt werden können. Es kann auch mehrere Stellen geben, an dem diese CDI-Events eingesammelt werden.

Die CDI-Events werden in einem beliebigen Modul folgendermaßen gefeuert:

```
@Inject

private Event<IEvent> events;

private void runSchedular(){
    de.hsos.smarthome.global.communication.data.Event e = new
de.hsos.smarthome.global.communication.data.Event();
    e.setDeviceID(device.getID());
    events.fire(e);
}
```

Listing 2: CDI Events

Diese Events werden dann per Rest-Client und JAX RS an die bereits registrierten Middlewares geschickt. Dabei muss die REST-Ressource seitens der Middleware **communication** heißen.

4.1.8 Webservices

Die Webservices sind wie die Controller in verschiedene Bereiche gegliedert. So gibt es die Device-, Gateway- und Registrationressource.

4.1.8.1 Device Webresource

Die **Device Ressource** kümmert sich um Anfragen, welche sich um die Geräte kümmern. Die URLs, welche seitens des Clients aufgerufen werden, entsprechen einem festen Schema, was im Folgenden verdeutlicht werden soll. Der Basiszugriff auf diese Ressource geschieht über die URL http://localhost:8080/Smarthome.translation/rest/device . Bei diesem Basisaufruf wird eine XML-Liste von Geräten mit den zugehörigen Informationen übermittelt. In Tabelle Tabelle 3: DeviceResource REST-Aufrufe sind die Rest-Aufrufe auf diesem Basispfad beschrieben.

Nr.	URL-Erweiterung	Aufruf	Beschreibung
1	{DeviceID}	GET	Gibt ein spezielles Gerät zurück
2	{DeviceID}/{Sensor}	GET	Spricht einen Sensor auf einem Gerät an
3	{DeviceID}/{Aktor}/?p1={p1}&p2={p2}	PUT	Führt einen Aktor mit einem oder zwei Parameter aus

Tabelle 3: DeviceResource REST-Aufrufe

Zur Verdeutlichung was von einem Gerät als XML übertragen wird, dient das X. In diesem Beispiel wird ein TestDevice übertragen, welches die Methoden *setHello()* und *getHello()* anbietet.

```
<testDevice>

 <aktors>
  <entry>
    <key>setHello-1</key>
    <value>
     <anzahlParameter>1</anzahlParameter>
     <key>setHello-1</key>
     <maxValue>0.0</maxValue>
     <methodName>setHello</methodName>
     <minValue>0.0</minValue>
     <parameterTypes>java.lang.String</parameterTypes>
     <regex></regex>
    </value>
  </entry>
 </aktors>
 <bezeichnung>HelloSensor</bezeichnung>
 <gateway>...</gateway>
 <ID>2</ID>
 <sensors>
  <entry>
   <key>getHello-0</key>
   <value>
    <anzahlParameter>0</anzahlParameter>
    <key>getHello-0</key>
    <maxValue>0.0</maxValue>
    <methodName>getHello</methodName>
    <minValue>0.0</minValue>
   </value>
  </entry>
 </sensors>
</testDevice>
```

Listing 3: Device als XML

4.1.8.2 Gateway Webresource

Die **Gateway Ressource** stellt die CRUD Operationen des Gatewaycontrollers für REST-Aufrufe bereit. Die Basis URL wäre dabei http://localhost:8080/Smarthome.translation/rest/gateway . Bei der BasisURL wird eine Liste von Gateways zurückgegeben.

Nr.	URL-Erweiterung	Aufruf	Beschreibung
1	{ID}	GET	Gibt einen Gatway aus
2	{ID}/AddIpPort/?ip={ip}&port={port}	PUT	Fügt einen Gateway hinzu
3	{ID}/AddConnectionString/?connectionstring={cs}	PUT	Fügt einen Gateway hinzu
4	{ID}/UpdateIpPort/?ip={ip}&port={port}	PUT	Updatet einen Gateway
5	{ID}/UpdateConnectionString/?connectionstring={cs}	PUT	Updatet einen Gateway
6	{ID}	DELETE	Löscht einen Gateway

Tabelle 4: Gateway Restschnittstelle

4.1.8.3 Registration Webresource

Die **Registration Ressource** dient zur Registrierung von neuen Middlewares, welche Events seitens der Translationschicht empfangen soll. Weiterhin kann man bereits registrierte Middlewares deregistrieren. Die Basis URL dieser Ressource lautet: http://localhost:8080/Smarthome.translation/rest/registration

Nr.	URL-Erweiterung	Aufruf	Beschreibung
1	/?name={name}&connectionstring={cs}	PUT	Fügt eine Middleware hinzu
2	/?name={name}	DELETE	Löscht eine Middleware

Tabelle 5: Registration Restschnittstelle

4.2 Middleware

Dies ist die zweite Schicht unserer dreistufigen Smarthome Architektur. Sie dient dazu die Businesslogik umzusetzen. Im groben ist sie auch wieder in Schichten aufgeteilt, um eine bessere Struktur zu garantieren.

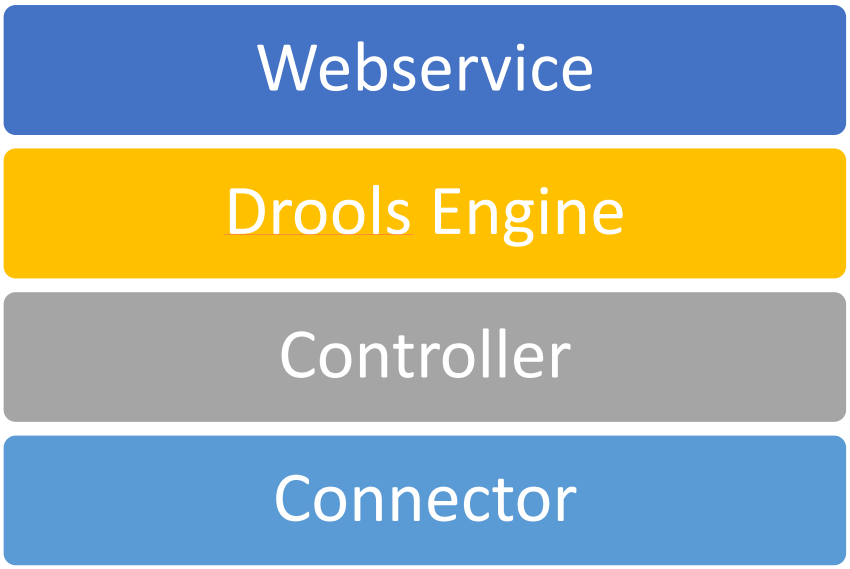


Abbildung 15: Middleware Architektur

In der Abbildung 15: Middleware Architektur sieht man die Unterteilung der Middleware.

Der Connector ist dazu bestimmt, die REST-Schnittstelle der Translationschicht anzusprechen. Dabei wird ein Marshalling der XML-Dokumente und Exceptions durchgeführt.

Die Controller innerhalb der Middleware kümmern sich darum, dass die internen Events richtig verwaltet und gefeuert werden.

Die Drools Engine ist unser zentrales Element, welches sich um die Fachlogik kümmert und Ergebnisse dieser Logik über den Connector an die Translationschicht weiterleitet.

Die Webservices bieten für die UI eine zentrale Stelle, um sich anzumelden. Dabei werden alle notwendigen Anforderungen für die UI erfüllt. So kann man über die REST-Schnittstelle Devices bearbeiten, Gateways bearbeiten und neue Regeln erstellen.

4.2.1 Connector

Um eine effektive Verbindung zwischen den beiden Schnittstellen zu schaffen, gibt es die Connector-Schicht. Sie gliedert sich in Client, Controller und Facade. Dieses wurde dreistufig gewählt, um eventuellen Erweiterungen bei den Controllern einen Platz zu verschaffen. Denn dafür muss der REST-Client nicht geändert werden, sondern man kann z.B. das Caching getrennt davon durchführen.

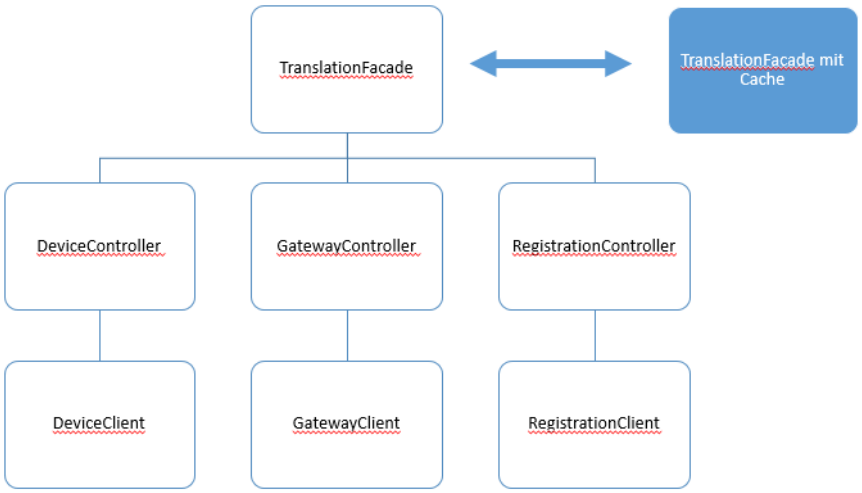


Abbildung 16: Middleware Connector

Das Caching von Device ID's wird momentan in der Translationfacade durchgeführt. Dabei kann die Facade über die Konfiguration ausgetauscht werden. Dies funktioniert mit den Java Alternatives[18].

Innerhalb dieses Connectors wird von dem Global Paket das Marshalling benutzt. Es handelt sich dabei um das Paket *de.hsos.smarthome.global.communication.marshalling*.

4.2.2 Drools

Drools [19]ist ein Java EE Framework, welche von JBOSS entwickelt wird. Es dient dazu die Fachlogik von der programmierten Businesslogik loszulösen.

Unsere Regeln sind nachvollziehbar und logisch aufgebaut. Wir können mit einer Gruppe aus Bedingungen einen Aktor setzen. Die Gruppe von Bedingungen besteht aus verschiedenen Geräten dessen angegebener Sensor einen bestimmten Status hat. Die Gruppe könnte im einfachsten Fall ein Sensor auf einem Gerät sein, welcher einen bestimmten Status hat.

Der Status oder auch Wert muss genauer beleuchtet werden, da wir über die Abstrahierung drei mögliche Datentypen haben. Die Werte seitens der Translationschicht können die Datentypen String, Boolean oder Double annehmen. Dieses Problem wurde mit Hilfe einer Vererbungshierachie der Stateconditions gelöst. So gibt es analog zu den Datentypen die StateConditions *StringStateCondition, BooleanStateCondition* und *DoubleStateCondition*.

Bei den Regeln, welche erstellt werden, werden die Restschnittstellen der Geräte hinterlegt. Das bedeutet im Detail, dass eine Regel über die ID und die Methode auf einen bestimmten Aktor und

[18] (Oracle, 2013)
[19] (JBOSS, 2013)

Sensor zugreifen kann. Durch den Zugriff über die Translationfacade wird der Zugriff auf die Translationschicht bearbeitet.

Da durch das Abarbeiten der Regeln eine gewisse Mehrstufigkeit auftritt, wurden Agendas eingeführt. Dadurch wird ein circulärer Verweis nach einer bestimmten Ausführungsanzahl beendet. Jedoch werden Regeln, welche weitere Sensoren beeinflussen per Drools ausgelöst. So werden auch die nachfolgenden Regeln für die angesprochenen Aktoren ausgelöst. Dies wird durch die Abbildung 17: Drools Funktionsweise illustriert.

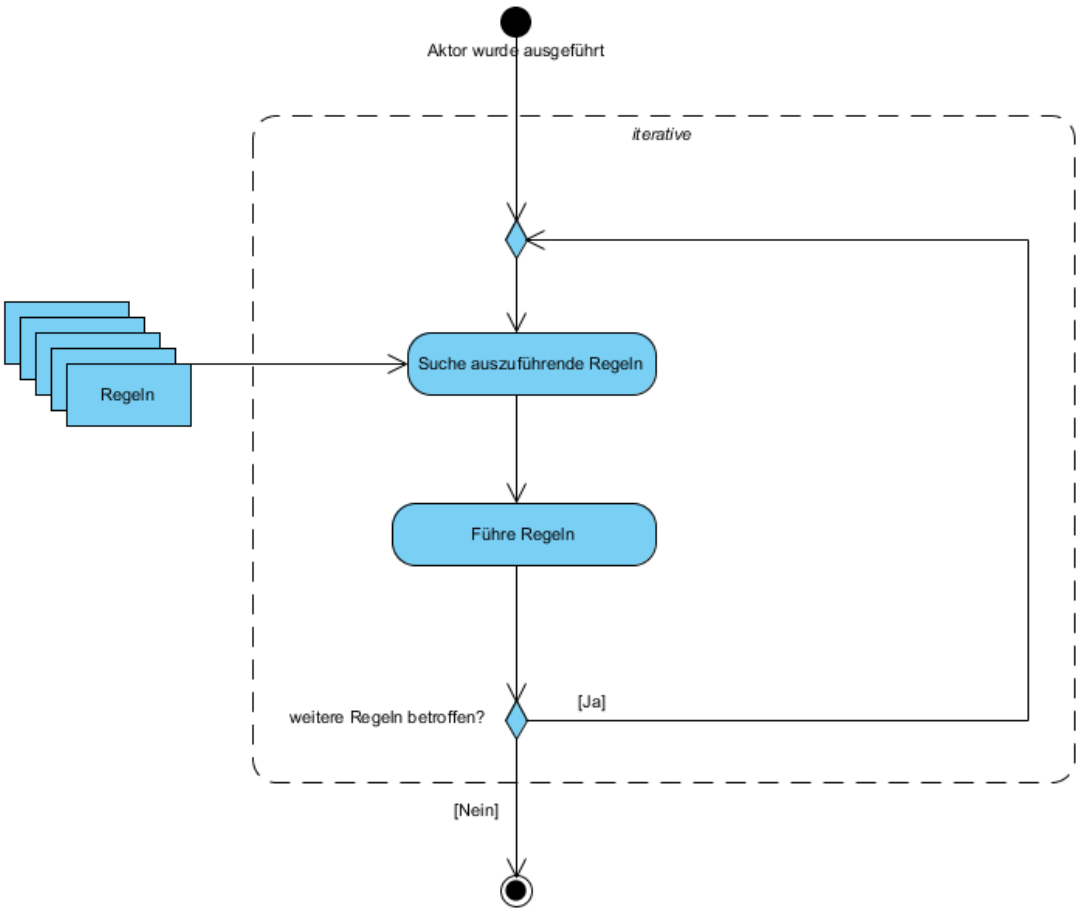


Abbildung 17: Drools Funktionsweise

Da wir nur einen kleinen Teil von Drools nutzen, welchen wir aber sehr individuell auf unsere Bedürfnisse angepasst haben, ist das Projekt in diesem Bereich sehr stark erweiterbar.

4.2.3 Webservice

Die Webservices der Middleware sind ähnlich, wie die Webservices der Translationschicht aufgebaut. Die Ressourcen *Devices* und *Gateways* sind auch über die Middleware zu benutzen, wie auch auf der Translationschicht selbst (siehe Webservices).

4.2.3.1 Conditionresource

Weiterhin gibt es die Ressource Condition. Diese Ressource dient dazu Regeln anzulegen und zu definieren. Die Schnittstelle ist nach X definiert. Die Basis-URL lautet http://localhost:8080/Smarthome.middleware/rest/condition .

Nr.	URL-Erweiterung	Aufruf	Beschreibung
1	/{DeviceID}/{Aktor}/{Wert}	PUT	Als XML muss eine Conditionlist übergeben werden. Erstellt eine Condition in der Middleware
2	/{DeviceID}	DELETE	Löscht eine Condition
3	/	GET	Ruft eine Liste von Regeln ab
4	/{RuleID}/{DeviceID}/{Aktor}/{value}	PUT	Updatet eine Regel

Tabelle 6: Conditionresource

4.2.3.2 UI-Registration-Resource

Die UI-Ressource dient zum Registrieren einer oder mehrere User Interfaces mit Rest-Client. Die Schnittstelle ist nach X definiert. Die Basis-URL lautet http://localhost:8080/Smarthome.middleware/rest/ui .

Nr.	URL-Erweiterung	Aufruf	Beschreibung
1	/{name}/&connectionstring={cs}	PUT	Fügt eine UI hinzu
2	/{name}	DELETE	Löscht eine UI

Tabelle 7: UI-Registration Resource

4.3 Exceptionhandling

Das Exceptionhandling ist sowohl innerhalb einer Schicht als auch schichtübergreifend möglich. Die Exceptions werden dabei im globalen Paket für alle Schichten einheitlich geführt. Dabei kann die Exception direkt über Java Jersey weitergeleitet werden. Dies funktioniert durch eine geschickte Wahl der Vererbung. Denn die Smarthomeexception, von der alle weiteren Exceptions erben, erweitert die Klasse *WebApplicationException*.

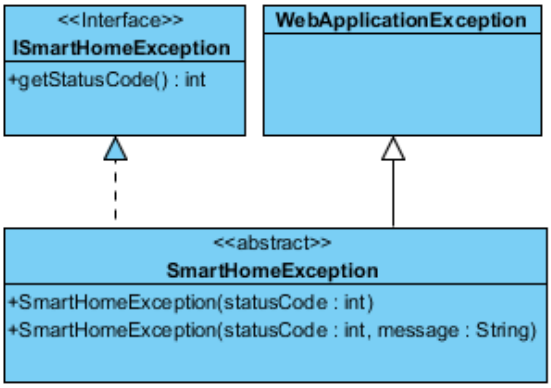
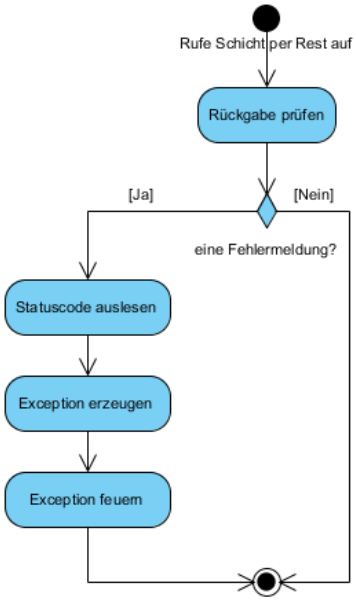


Abbildung 18:Exception Hierachie

Wenn nun ein REST-Client einen Fehler produziert, bekommt dieser den Statuscode 500 zurück. Weiterhin erhält er den eindeutig definierten Status-Code und, sofern sie definiert wurde, eine Nachricht.

Sofern der Rest-Client einen Fehler seitens der gegenüberliegenden Schicht bekommt, wird das Exceptionmarshalling ausgelöst. Der auslösende Fehler ist in Java die UniformInterfaceException. Der Marshaller prüft nun ob ein Fehler vorhanden ist. Wenn dieser vorhanden ist, wird er übersetzt und eine neue Java Exception generiert. Falls keine geeignete Exception gefunden wird, wird eine UnknownException gefeuert.

Diese Exceptions können nun von einem Controller bearbeitet werden. Spätestens müssen sie an der Oberfläche gefangen und in einen sinnvollen Kontext gebracht werden.



4.4 Smarthome - GUI

Die Oberfläche beruht auf Primefaces in der Version 3.5. Diese stellt Komponenten für JSF bereit und ermöglicht so eine einfache und schnelle Entwicklung der Oberfläche.

4.4.1 Die Startseite

Das Layout teilt die Seite in drei Bereiche. Im oberen Teil der Seite befindet sie Die Toolbar für die Navigation, das Datum und die Uhrzeit.

Auf der linken Seite sehen wir die vorhandenen Geräte die uns die Provider zur Verfügung stellen. Die rechte Seite der Oberfläche stellt das Datencenter da, hier werden die jeweiligen Aktoren zu dem selektierten Gerät von der linken Seite Dargestellt. Zunächst werden keine Geräte angezeigt, da ein Provider erst noch registriert werden muss. Um eine Anmeldung eines Providers vorzunehmen, geht man auf den Button „Steuerung“ und wählt „System“ aus.

Abbildung 19:UI Startseite

4.4.2 System

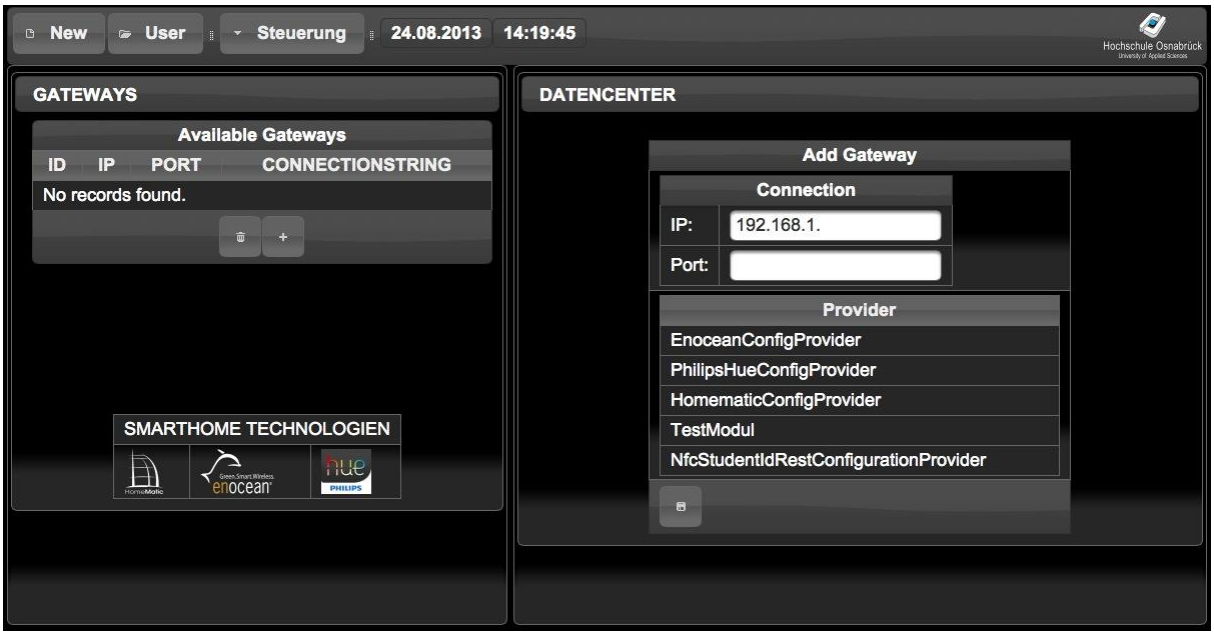


Abbildung 20: UI Gateway hinzufügen

Auf der linken Seite sieht man eine Liste mit den bereits angemeldeten Providern.

Die rechte Seite stellt ein Eingabeformular bereit, um einen neuen Provider anzumelden.

4.4.3 Eingabeformular

Hier sind zwei Eingabe Felder für die IP und dem Port des Gateways und eine Liste mit zu dem Zeitpunkt unterstützten Providern.

Nach Eingabe der IP und Port muss man den Provider auswählen den man registrieren möchte. Sobald die Anmeldung erfolgreich verlaufen ist wird der neue hinzugefügte Provider in der Liste auf der linken Seite angezeigt.

Nun stehen die Geräte des neuen Providers auch auf der Startseite zur Verfügung.

Abbildung 21: Startseite mit Geräten

4.4.4 Aktoren Interface

Wenn ein Gerät ausgewählt wird, öffnet sich auf der rechten Seite das Aktoren Interface. Dieses stellt alle möglichen Aktoren bereit und rendert, je nach Gerät, die benötigten Aktoren. Daraufhin kann man die gewünschten Einstellungen vornehmen und durch den Button „Set“ bestätigen.

4.4.5 Neue Aktoren hinzufügen

Jeder Aktor benötigt einen User-Interface und eine Controller-Klasse, welche von AbstraktAktorController erbt. Der Controller benötigt eine MiddlewareFacade, welche wir uns über @Inject holen, eine Variable methode, welches den Namen des Aktoren als String Datentypen trägt, eine Funktion setAktor, um die Werte über REST an die Middleware weiter zuleiten, so wie die jeweiligen getter und setter.

```
@Named("On")
@SessionScoped
public class OnOffController extends AbstraktAktorController {

    @Inject
    private MiddlewareFacade mwFacade;
    private String methode;
    private boolean onOff;

    public OnOffController() {
        this.onOff = false;
        this.methode = "setOn";
    }

@Override
    public void setAktor(String ID) {
        int id = Integer.parseInt(ID);
        mwFacade.setDeviceValue(id,methode,String.valueOf(onOff));
}
}
```

Listing 4:Beispiel: OnOffController für Philips Hue Lampe und Steckdose

Es wird noch ein User-Interface für den Controller gebraucht welches wir in die aktors.xhtml einfügen.

```
<h:outputText value="Status: "
rendered='#{Aktors.render(On.methode)}'/>
<p:selectBooleanButton value="#{On.onOff}"
onLabel="On" offLabel="Off" rendered='#{Aktors.render(On.methode)}' />
     <p:commandButton icon="ui-icon-disk" value="Set"
action="#{On.setAktor(MenuDevices.selectedDevice.ID)"
rendered='#{Aktors.render(On.methode)}'/>
```

Listing 5:OnOffController

Dieser Controller muss noch angemeldet werden. In den Konstruktor der Klasse AktorsController.java fügt man noch folgende Zeile ein. this.b.put("setOn", new KeyPair(false));

4.4.6 Regeln

Im Untermenü des „Steuerung“ Button findet man „Rules“, das zu einer Oberfläche führt, in der man Regeln erstellt.

Zunächst hat man eine Liste von Regeln die bereits vorhanden sind. Um neue Regeln zu erstellen, drückt man den Plus Button im unteren Teil der Tabelle.

Hier werden alle Geräte angezeigt die Aktoren besitzen, von denen ein Gerät selektiert werden kann. Im nächsten Fenster befindet sich eine Liste mit den Vorhandenen Aktoren des selektierten Gerätes in dem erneut eines selektiert werden muss. Das nächste Feld enthält ein Eingabe Fenster für den zu übergebenen Wert, dabei ist es wichtig mögliche Eingabewerte zu kennen. Weiter findet man eine Liste mit Conditions, die erstellt werden müssen. Durch den Plus Button geht es weiter. Man wählt

das Gerät, den zugehörigen Sensor, in das Eingabefenster legt man den Parameter fest und wählt noch einen Operator, wann diese Condition greifen soll. Wenn man dann wieder zur Condition-Liste gelangt kann man weitere Condition hinzufügen oder durch den Button „Rules“ zu den Regeln springen. Wichtig: Vorher muss die erstellte Regel bestätigt werden, dies Geschieht über die Diskette.

4.4.7 Fassade

Die Oberfläche kommuniziert mit der Middleware über die Middlewarefacade, diese wird über @Inject injiziert. Alle Informationen die für diese Oberfläche gebraucht werden, müssen über die Facade geholt und verarbeitet werden. Die Oberfläche hält keine Informationen.

4.4.8 Fazit

Primefaces ist ein sehr nützliches Framework um Oberflächen zu erstellen. Doch führen die Beispiele auf der Demoseite nicht immer zu einer Lösung, da manche Beispiele so wie sie dort beschrieben werden nicht funktionieren und wichtige Informationen nicht erwähnt werden.

In Zukunft kann man diese Oberfläche intuitiver gestalten. Das klicken bei der Regelerstellung ist unvorteilhaft und es ist notwendig die zu übergebenen Parameter genau zu kennen. Schöner wäre es, diese dem Nutzer direkt über ein Interface anbieten zu können. Weiter hin bekommt der Nutzer keine Rückmeldung über das Hinzufügen von Providern.

5 Aussichten für weitere Projekte

Weitere Projekte könnten auf unser Smarthome Projekt aufbauen. Diese Projekte könnten verschiedene Ziele beinhalten:

1. Optimierung der Performance durch intensiveres Caching der Status in der Middleware
2. Tests & Qualitätssicherung massiv ausweiten und eine Test Abdeckung von ca. 85% erreichen. Dies könnte man in ein Projekt „Java Unit Tests on Legacy Code“ bearbeiten.
3. Neue Module für weitere Hersteller entwickeln. Dies sollte durch unsere Dokumentation sehr schnell möglich sein.
4. Oberfläche erweitern und neue Funktionen implementieren.
5. Verteilung der einzelnen Schichten in der Hochschule Live durchführen und größere Performancetests durchführen.

6 Fazit

Um ein geeignetes Fazit zu ziehen, müssen wir das Projekt ganzheitlich betrachten. Das heißt vom Projektmanagement an bis hin zu einem Ausblick wo ein weiteres Projekt anschließen könnte.

6.1 Projektmanagement

Das Projektmanagement wurde von Anfang an stringent durchgeführt. Die wöchentlichen Meetings machten das Projekt überschaubarer und es leichter das ganze Team zu synchronisieren. An manchen Ecken wäre das Projektmanagement effektiver verlaufen, hätte man mit agilen Vorgehensmethoden gearbeitet. Dies war aber Aufgrund des Studiums und der Belastung von anderen Fächern nur eingeschränkt möglich.

6.1.1 Aufgabenverteilung

Um einen konkreten Ansprechpartner für Module zu haben, ist hier eine Aufstellung, wer sich mit welchem Modul und Aufgabe innerhalb des SmartHome Projektes befasst hat.

Nr.	Aufgabe	Personen
1	Recherche	Dohe, Klassen, Köster, Münch, Schöppe, Willmann, Zelass
2	Anforderungsanalyse	Dohe, Klassen, Köster, Münch, Schöppe, Willmann, Zelass
3	Architektur Konzept und Implementierung	Münch, Schöppe
4	Translationschicht	Münch
5	Modul Philips Hue	Schöppe
6	Modul Homematic	Dohe, Willmann
7	Modul Enocean	Köster, Zelass
8	Middleware	Münch, Schöppe
9	UI	Klassen, Köster

6.2 Eingesetzte Werkzeuge

In dem Projekt wurden vor allem Tools genutzt, welche dem Team geholfen haben effizient zu arbeiten und die Ergebnisse sinnvoll aufzubereiten. Zur Dokumentenablage wurde Feng Office benutzt, ein Open Source Dokumentenmanagementsystem.

GIT wurde als Versionsverwaltung gezielt genutzt und half bei der Kollaboration der Teammitglieder, so dass alle auf dem gleichen Stand waren. Als Anbieter wurde github.com ausgewählt. Das Projekt ist dort als privates Projekt deklariert.

Bei der Entwicklung von Java EE 6 haben wir auf die Netbeans IDE gesetzt, welche mit dem Glassfish Server sehr gut integriert ist. Außerdem haben wir die Integration von GIT in Netbeans genutzt, um unseren Quellcode zu teilen.

Zum Darstellen der Softwarearchitektur mit UML haben wir das Werkzeug Visual Paradigm genutzt.

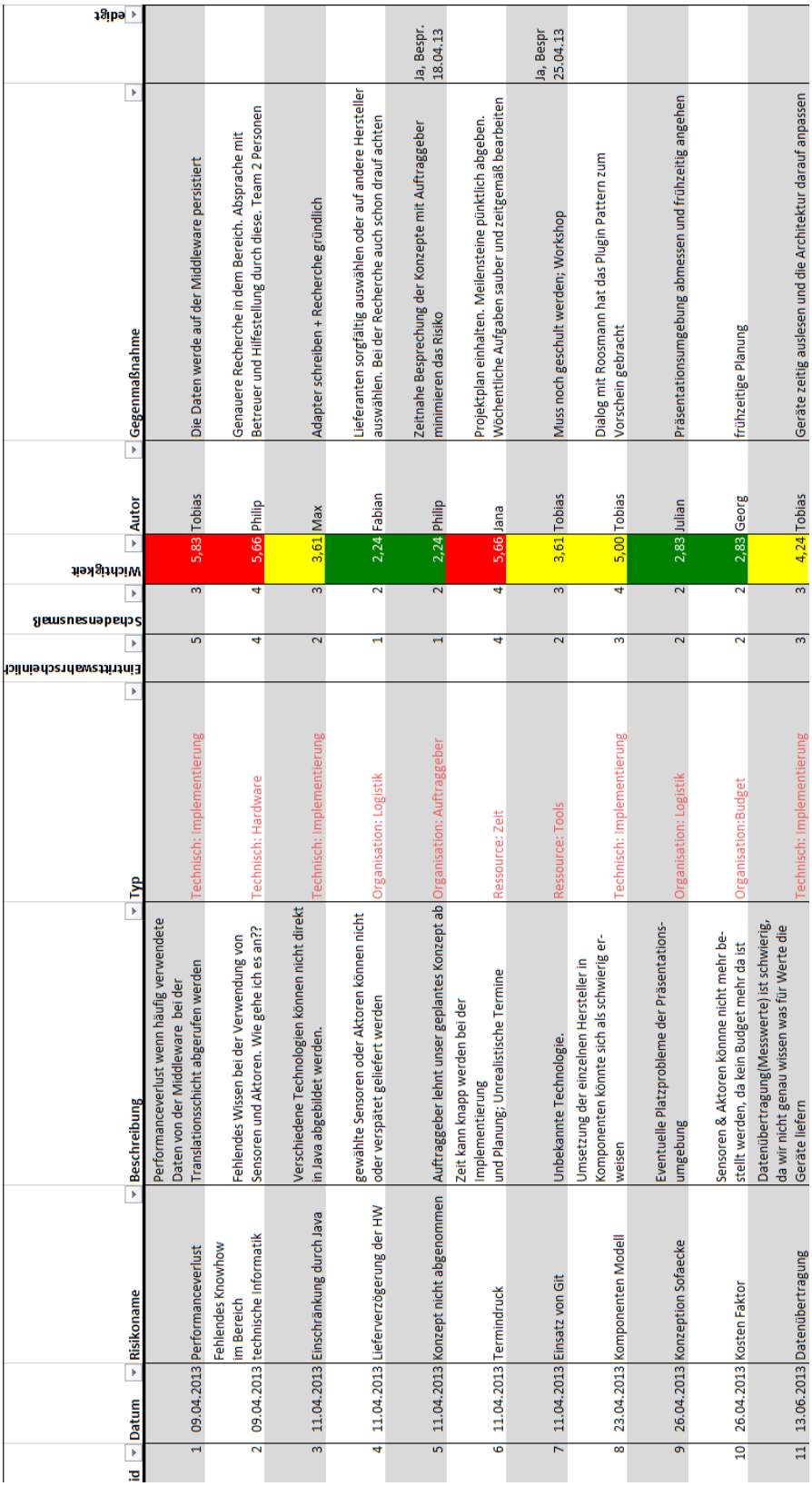
6.3 Lessons Learned

Aus dem Projekt können wir als Team folgendes ziehen:

- Ein Projekt was übersichtlich beginnt, birgt oft unbekannte Tiefen. So haben wir uns bei den Meilensteinen für Enocean und Homematic verkalkuliert, da das übersetzen von dritten Protokollen ziemlich zeitaufwändig ist.
- Die Eskalation eines Meilensteins hätte sauber und professioneller geführt werden müssen, so dass für alle ersichtlich ist, dass wir hinter dem Zeitplan hängen.
- Es wäre von Vorteil gewesen alle Anforderungen von Anfang an zu kennen, um so diese effektiv in den Projektplan zu verwalten.
- Bei den Meetings ist es sinnvoll vorher eine Agenda an alle Beteiligten zu schicken, um genau aufzuzeigen um was es geht. Dadurch können sich alle Teammitglieder sinnvoll vorbereiten und das Meeting besteht nicht nur aus dem Verteilen von Rechercheaufgaben.

7 Anhang

7.1 Risikoliste



id	Datum	Risikoname	Beschreibung	Typ	Eintrittswahrscheinlich	Schadensumfa	Wichtigkeit	Autor	Gegenmaßnahme	edigt
1	09.04.2013	Performanceverlust	Performanceverlust wenn häufig verwendete Daten von der Middleware bei der Translationsschicht abgerufen werden	Technisch: Implementierung	5	3	5,83	Tobias	Die Daten werde auf der Middleware persistiert	
2	09.04.2013	Fehlendes Knowhow im Bereich technische Informatik	Fehlendes Wissen bei der Verwendung von Sensoren und Aktoren. Wie gehe ich es an??	Technisch: Hardware	4	4	5,66	Philip	Genauere Recherche in dem Bereich. Absprache mit Betreuer und Hilfestellung durch diese. Team 2 Personen	
3	11.04.2013	Einschränkung durch Java	Verschiedene Technologien können nicht direkt in Java abgebildet werden.	Technisch: Implementierung	2	3	3,61	Max	Adapter schreiben + Recherche gründlich	
4	11.04.2013	Lieferverzögerung der HW	gewählte Sensoren oder Aktoren können nicht oder verspätet geliefert werden	Organisation: Logistik	1	2	2,24	Fabian	Lieferanten sorgfältig auswählen oder auf andere Hersteller auswählen. Bei der Recherche auch schon drauf achten	
5	11.04.2013	Konzept nicht abgenommen	Auftraggeber lehnt unser geplantes Konzept ab	Organisation: Auftraggeber	1	2	2,24	Philip	Zeitnahe Besprechung der Konzepte mit Auftraggeber minimieren das Risiko	Ja, Bespr. 18.04.13
6	11.04.2013	Termindruck	Zeit kann knapp werden bei der Implementierung und Planung; Unrealistische Termine	Ressource: Zeit	4	4	5,66	Jana	Projektplan einhalten. Meilensteine pünktlich abgeben. Wöchentliche Aufgaben sauber und zeitgemäß bearbeiten	
7	11.04.2013	Einsatz von Git	Unbekannte Technologie.	Ressource: Tools	2	3	3,61	Tobias	Muss noch geschult werden; Workshop	Ja, Bespr 25.04.13
8	23.04.2013	Komponenten Modell	Umsetzung der einzelnen Hersteller in Komponenten könnte sich als schwierig erweisen	Technisch: Implementierung	3	4	5,00	Tobias	Dialog mit Roosmann hat das Plugin Pattern zum Vorschein gebracht	
9	26.04.2013	Konzeption Sofaecke	Eventuelle Platzprobleme der Präsentationsumgebung	Organisation: Logistik	2	2	2,83	Julian	Präsentationsumgebung abmessen und frühzeitig angehen	
10	26.04.2013	Kosten Faktor	Sensoren & Aktoren könnne nicht mehr bestellt werden, da kein Budget mehr da ist	Organisation:Budget	2	2	2,83	Georg	frühzeitige Planung	
11	13.06.2013	Datenübertragung	Datenübertragung(Messwerte) ist schwierig, da wir nicht genau wissen was für Werte die Geräte liefern	Technisch: Implementierung	3	3	4,24	Tobias	Geräte zeitig auslesen und die Architektur darauf anpassen	

7.2 Use-Case Diagramm