Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Elektro- & Informationstechnik
Faculty of Electrical and Information Technology

Evaluierung von Konzepten zur Integration des Nutzerwelten Systems in die Softwareplattform von openHAB

Masterarbeit

Hochschule Düsseldorf

vorgelegt von
Wolfram Gerlach
Matrikelnummer: 524126
Geb. am: 18.05.1983 in: Düsseldorf

Erstprüfer: Prof. Dr. Lux
Zweitprüfer: Prof. Dr. Schaarschmidt

Düsseldorf, den 7. Juni 2017

Anmerkung: Formular Masterarbeit aufgrund datenschutzrelevanter Informationen entfernt.

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit eigenständig und nur unter Verwendung der genannten Literatur und den aufgeführten Hilfsmitteln verfasst habe

Düsseldorf den,

20. Juni 2016

_____

Wolfram Gerlach

Zusammenfassung

In dieser Arbeit geht es um die Implementierung des NutzerWelten Systems in das Open Source Projekt von openHAB. Erst wird openHAB vorgestellt und die Konzepte dahinter, wie das Thing und das Item. Hiernach wird die NutzerWelten Plattform erörtert und die Architektur dahinter präsentiert. Aufbauend darauf wird auf die Konzepte zur Integration des NutzerWelten Systems in openHAB eingegangen. Anschließend wird ein Entwurf des Systems in UML aufgestellt und die Konzepte der Software Architektur vorgestellt, die für die Implementierung hilfreich sind. Zuletzt wird der Entwurf präzisiert und vertieft. Hierbei wird auf die Implementierung der einzelnen Klassen eingegangen und ihre Umsetzung vorgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Kürzelverzeichnis

AAL Ambient Assisted Living

HTTP Hypertext Transfer Protocol

IP Internet Protocol

JSON Java Skript Objekt Notifikation

MQTT Message Queue Telemetry Transport

openHAB open Home Automation Bus

OSGi	Open Services Gateway Initiative
RCP	Rich Client Plattform
REST	Representational State Transfer
UDP	User Datagram Protocol
UI	User Interface
UID	Unique Identifier
UML	Unified Modeling Language
UPnP	Universal Plug and Play
XML	Extensible Markup Language

1. Einleitung

Diese Masterarbeit handelt von der Kozeptionierung einer Integrationslösung zur Implementierung der NutzerWelten Software Architektur in die Plattform von openHAB. Hierdurch werden die beiden Themenfelder Smart Home und Ambient Assisted Living (AAL) miteinander verbunden. Dies ist sinnvoll, da es bei beiden Systemen um durch Technik unterstütztes Leben geht. AAL beschreibt die Erhaltung der Selbstständigkeit im Alter, während Smart Home für den Gewinn von Lebenskomfort steht.

Viele Geräte und Systeme sind damit in beiden Bereichen vorhanden. So soll eine Heizungssteuerung bei AAL Systemen einer Person das Regeln der Heizung abnehmen, damit zum Beispiel bei geöffnetem Fenster nicht unnötig geheizt wird. Im automatisiertem Haus ist dies ein Komfortfaktor, der zudem auch noch Strom und Gaskosten sparen kann. Ein anderes Beispiel wäre eine Präsenz-Erkennung, die detektiert, welche Personen gerade im Haus sind. Im intelligenten Eigenheim dient sie zur Abstimmung der Programme im Haus, ist niemand zuhause, wird die Alarmanlage aktiviert und die restlichen Funktionen in den Stromsparmodus geschaltet. Im technisch betreuten Wohnen dient die Präsenz-Erkennung besonders dazu, um Notfälle zu detektieren und zum Beispiel durch Aktivitätsprofile sicherzustellen, dass mit den Personen im Haus alles in Ordnung ist.

1.1. Ambient Assisted Living

Das Ambient Assisted Living steht für technisch betreutes Wohnen im Alter. Nach Studien des statistischen Bundesamtes und der Bundeszentrale für Politische Bildung werden im Jahr 2060 bis zu 38,2 % der Deutschen Bevölkerung über 60 Jahre alt sein. Schon bis zum Jahr 2020 wird ein Anstieg in der Bevölkerung auf 29,5 % vorhergesagt[3]. Nach Analysen des statistischen Bundesamtes wird damit auch die Anzahl der Pflegebedürftigen steigen[38]. Um Menschen länger ein selbstbestimmtes Leben zu ermöglichen, wird im Bereich des AAL geforscht, welche technischen Helfer den Lebensalltag einer älteren Person erleichtern und sicherer machen, damit sie ihre Selbstständigkeit auch im höheren Alter noch behalten kann.

An der Hochschule Düsseldorf wird zur Zeit das NutzerWelten Projekt durchgeführt, im Zuge dessen diese Arbeit erstellt wird. Dieses Projekt gliedert sich in zwei Bereiche auf; der Studie Sicherheit und der Studie Kommunikation. In der Studie Sicherheit geht es um die Ermittlung, in wie weit ein AAL-System bei Menschen mit demenzieller Erkrankung eingesetzt werden kann, um die Lebensqualität einer Person zu verbessern sowie die Eigenständigkeit zu erhalten und das Sicherheitsempfinden zu erhöhen[18,

Seite 3]. In dieser Studie wurde das AAL-System aus dem vorangegangenen WieDAS Projekt in verschiedenen Haushalten eingesetzt. Es wurden hierbei jeweils 20 Haushalte untersucht, wobei nur zehn Haushalte das System einsetzten und die anderen als Kontrollgruppe dienten.

Die Entwicklung neuer Geräte wird über die Studie Kommunikation durchgeführt. Hierbei geht es darum, die Kommunikation zwischen Demenzerkrankten und ihren Angehörigen, gestützt durch Technik, zu verbessern.

1.2. Smart Home

Das intelligente und vernetzte Haus des Smart Home soll sich an seine Bewohner anpassen und ihnen Arbeiten abnehmen. Hauptgrund für die Vernetzung des Hauses ist der gesteigerte Wohnkomfort, aber ein automatisiertes Haus dient auch zur Energieeinsparung und dem Gewinn von Sicherheit.

Kern ist meist eine Schaltzentrale, ein Server, der im Haus sämtliche Geräte überwacht und durch zusätzlich angebrachte Sensoren erkennt, wann welches Gerät im Hause eingeschaltet wird oder abgeschaltet werden kann. Die Aufgabe dieser Zentrale ist es, dem Nutzer das Schalten von Sachen abzunehmen und ihn über wichtige Gegebenheiten zu informieren. Ein gutes smartes Haus kennt die Orte, an denen sich in ihm Personen befinden, und schaltet das Licht in den entsprechenden Räumen ein. Es erkennt zudem, dass eine Person gerade geschlafen hat und zur Toilette muss und kann entsprechend das Licht dimmen. Zum Teil kann ein Smartes Haus sogar seine Eigentümer erkennen und bemerkt, wenn ein Einbruch durchgeführt wird. Die entfernte Steuerung des Hauses über ein Smartphone ist hierbei nur eine wenig smarte Zusatzfunktion[16, Seite 51].

Das Smart Home ist heute ein stark umworbenes Feld. Viele Hersteller drücken ihre Insellösungen in den Markt. Dies ist aber ein Problem, da diese Lösungen untereinander nicht kompatibel sind. Zudem bilden viele Hersteller nicht das komplette Spektrum an Möglichkeiten mit ihren Systemen ab, sodass Geräte anderer Hersteller zusätzlich nötig werden. Diese Inseln miteinander zu koppeln ist die Aufgabe vom open Home Automation Bus (openHAB).

Dies ist ein Open Source System, welches von Kai Kreuzer entwickelt wurde. Durch den wachsenden Erfolg, wurde die Eclipse Foundation auf das Projekt aufmerksam und entwickelte auf Basis von openHAB die Eclipse SmartHome Plattform, welche wiederum nun das Basissystem zu openHAB 2 darstellt. Die Wahl auf dieses Projekt viel, da es auf den gleichen Technologien basiert wie NutzerWelten (Java und OSGi) und damit bereits Knowhow vorhanden ist.

1.3. Motivation

Schon im WieDAS Projekt war angedacht die Kompatibilität der Software Architektur zu anderen Systemen wie UniversAAL herzustellen. Dies wurde aber nicht realisiert. Die Kompatibilität zu Eclipse SmartHome wurde schon zur Anfangszeit des NutzerWelten Projektes ins Auge gefasst, konnte aber aufgrund der Projekt-Struktur der Studie Sicherheit erst zum aktuellen Zeitpunkt umgesetzt werden. OpenHAB ist neben der QIVICON-Box und anderen Systemen eine der Referenz Implementierungen von Eclipse SmartHome[5].

Ziel dieser Arbeit ist es eine Kompatibilität zwischen der WieDAS-/NutzerWelten-Architecktur und ihren Geräten mit dem System von openHAB herzustellen. Im folgenden sollen dazu die einzelnen Architekturen vorgestellt werden und mögliche Schnittstellen erörtert werden. Des weiteren wird die Realisierung einer möglichen Implementierung vorgestellt und das Vorgehen bei ihrer Umsetzung.

2. Das openHAB System

Der open Home Automation Bus kurz openHAB ist eine Software, welche als Middleware die Hausautomatisierungslösungen verschiedener Hersteller verbindet und für den Anwender über ein Webinterface oder eine Smarthome-App steuerbar macht. Die Entwicklung wurde von Kai Kreuzer als open Source Projekt gestartet. Eine erste Version wurde am 21. Februar 2010 auf Google Code veröffentlicht[30]. Seit dem ist die Community um das Projekt stark gewachsen.

Im September 2013 startete die Entwicklung des Eclipse SmartHome Projektes[26]. Zu diesem diesem Zweck wurde die nicht kommerziell ausgerichtete openHAB UG (haftungsbeschränkt) gegründet, um einen Business Ansprechpartner für Unternehmen zu stellen. Zudem ist die openHAB UG Partner verschiedener Allianzen, wie der EnOcean Allianz, der AllSeen Allianz, sowie der Eclipse Foundation[28]. Eclipse SmartHome wurde aus openHAB abgeleitet, ist ein von der Eclipse Foundation geführtes Projekt und richtet sich an Business Partner. Industrielle Partner des Projektes sind, neben der openHAB UG, Bosch, QIVICON und itemis[6]. Das Eclipse SmartHome-System stellt den Unterbau zu openHAB 2[31].

Am 20. Mai 2016 wurde die openHAB Foundation gegründet. Diese ist der Nachfolger der openHAB UG, welche bis zum Ende des Jahres abgewickelt werden soll. Auch die Foundation arbeitet nicht kommerziell, es soll aber auch für Firmen möglich sein dieser beizutreten[29].

2.1. Eclipse SmartHome

Wie schon erwähnt ist das Eclipse SmartHome Projekt aus openHAB hervorgegangen. Ziel ist es, mit dem System ein Basissystem zu schaffen, welches von Industriepartnern ohne rechtliche Probleme eingesetzt werden kann. Diese steuern im Gegenzug eigenen Quellcode zu Eclipse SmartHome bei und helfen damit das System weiter zu verbessern. Zudem sind die auf Eclipse SmartHome basierenden Systeme zueinander kompatibel. Hierdurch sollen die Abgrenzungen der Insellösungen aufgebrochen werden und eine systemübergreifende Allianz geschaffen werden. Andere Firmen wie Apple mit Homekit verfolgen einen ähnlichen Ansatz[1].

Wie openHAB ist auch das Eclipse SmartHome Projekt ein Open Source Projekt. Im Unterschied hierzu baut Eclipse SmartHome auf der wesentlich restriktiveren Eclipse Public License v1.0 auf. Dies bedeutet zum Beispiel, dass kein fremder Code im Projekt enthalten seien darf und kein Code aus Reverse Engineering in das System mit einfließen

darf. Auch ist die Lizenz inkompatibel zu Lizenzen wie der LGPL[1]. Der Lizensierungsprozess wird zudem durch Urheberrechtsprüfungen gestützt[25].

Zuletzt ist Eclipse SmartHome kein eigenständig lauffähiges System, sondern nur ein Basisframework. Die funktionalen Implementierungen obliegen den einzelnen Herstellern, die das System einsetzen. Die Integration des Eclipse SmartHome Framework in das System von openHAB ist in Abbildung 1 abbgebildet. Zusätzlich zum Framework besitzt der open Home Automation Bus einen Jetty Webserver und die openHAB 2 Core Bundels. Die Bundels basieren auf dem Standard der Open Services Gateway Initiative[2] und werden in einem Apache Karaf[3] -Container gestartet.

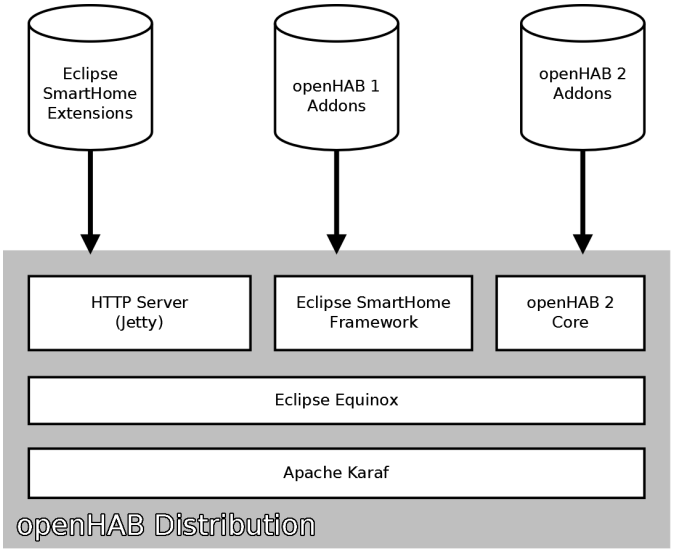

Abbildung 1: Aufbau der openHAB Distribution [27]

2.2. Aufbau von openHAB 2

Der open Home Automation Bus ist ein in Java geschriebenes Programm. Es basiert auf den Eclipse Equinox Spezifikationen. Das Equinox Framework ist ein OSGi-Framework der Eclipse Foundation. Geräte oder Services werden in openHAB als OSGi-Bundels implementiert. Diese Bundels werden Bindings genannt.

[1] LGPL steht für lesser GNU Public License und ist eine abgespeckte Version der GNU Public License (GPL) die vom GNU Projekt herausgegeben wird[13].

[2] Die Open Services Gateway Initiative (OSGi) ist eine Spzifikation der OSGi-Allianz für eine Softwareplattform. Ziel ist es hierbei ein möglichst modulares System zu erschaffen[12, Seite 12].

[3] Apache Karaf ist ein Software Container, der mit verschiedenen Frameworks umgehen kann. Apache Karaf unterstüzt hierfür verschiedene Loggingdienste und Konsolen-Systeme. Der besondere Vorteil von Apache Karaf ist, dass das System dynamisch ist und Änderungen an Einstellungen sowie das Hinzufügen von Softwarekomponenten ohne Neustart möglich sind[11].

Zur besseren Beschreibung der Struktur von openHAB 2 wurde das System in vier Schichten eingeteilt[4]. Diese sind die Benutzerebene, die Konfigurationsschicht, die Abstraktionsschicht und die reale Welt. Auf der Benutzerebene Ebene findet die Interaktion mit dem Anwender statt. Die Konfigurationsschicht ist die Schicht, auf der die Verwaltung und Konfiguration der Nutzerinteraktionen durchgeführt wird. Die Verwaltung der SmartHome Geräte findet auf der Abstraktionsschicht statt. Der Aufbau wird in Abbildung 2 gezeigt. Dieser soll in den folgenden Kapiteln erläutert werden.

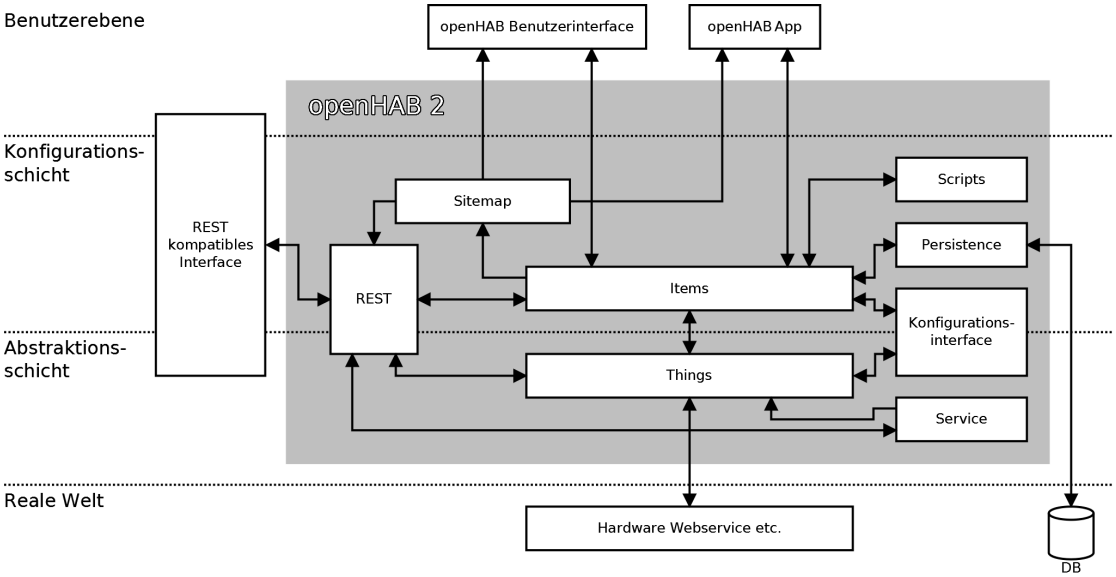

Abbildung 2: Aufbau von openHAB(eigene Abbildung)

2.2.1. Die Benutzerebene

Die Benutzerebene stellt die Dienste für den Anwender zur Verfügung. Der Nutzer kann auf dieser Ebene einzelne Geräte steuern und bekommt Anzeigen über deren Zustand präsentiert. Auf der Ebene befinden sich das openHAB-Webinterface sowie die Smartphone Apps. Zuletzt ist es auch möglich, über ein REST[5] kompatibles Interface ein eigenes Webinterface zu erstellen.

[4] Anmerkung des Autors: Da die Entwickler von openHAB bisher(stand 9. Mai 2016) keine eigne Architektur Beschreibung für openHAB 2 veröffentlicht haben, ist die folgende Beschreibung keine Offizielle. Die internen Strukturen innerhalb der Architektur sind vielschichtiger, eine vollständige Codeanalyse würde den Rahmen der Arbeit sprengen. Der hier dargestellte Aufbau dient zur Visualisierung der Funktionsweise hinter openHAB und geht nicht auf den dahinter liegenden Java Aufbau ein.

[5] REST steht für Representational State Transfer und ist ein Architektur-Stil für den Entwurf von Netzwerk Applikationen. REST ist meist HTTP-basiert[10].

openHAB Benutzerinterface Der open Home Automation Bus stellt dem Nutzer zwei User Interfaces (UIs) bereit, welche als Webdienst über das lokale Netz im Webbrowser angezeigt werden können, das Basic UI und das Classic UI. Der Unterschied der Interfaces ist dabei rein optisch. Sie besitzen keine direkten funktionalen Unterschiede. Das Classic UI stellt die Geräte in Zeilen dar. Auf der linken Seite ist der Item Name angezeigt auf der rechten sein Zustand. Das Basic UI arbeitet genauso wie das Classic UI erweitert aber die Ansicht bei ausreichend großer Darstellungsfläche um eine weitere Spalte. Bei diesem UI sind die Grafiken zudem als Vektorgrafiken hinterlegt.

openHAB App Der open Home Automation Bus bietet auch für Smartphones eigene Apps an. Diese Arbeiten beim lokalen Zugriff genauso wie das Benutzerinterface.

2.2.2. Die Konfigurationsschicht

Die Konfigurationsschicht ist die Ebene, auf der die Verarbeitung der Benutzerkommandos stattfindet, sowie Verwaltungsdienste für Geräte arbeiten. Auf ihr findet die automatische Verwaltung und die Steuerung des smarten Hauses statt. Ihre Kernkomponenten sind die Sitemap, die Items, die Scripts und die Persistence. Zudem arbeitet das Konfigurationsinterface als auch die REST-Schnittstelle auf dieser Ebene. Beide Elemente haben aber auch Funktionalitäten, die auf die Abstraktionsschicht zurückgreifen. Auf der Konfigurationsschicht wird die Konfiguration über Skript-Dateien angelegt. Die Namenskonvention ist hierbei „Haushaltskonfigurationsname.Skripttyp“ als Beispiel „DemoHaushalt.sitemap“. Nachfolgend sollen die einzelnen Komponenten der Konfigurationsschicht vorgestellt werden.

Items Die Items sind Eigenschaften von Geräten. Dies kann zum Beispiel der Zustand eines Türkontakt oder die verbleibende Ladung in seiner Batterie sein. Auch können Items Schalter bereitstellen, um den Zustand eines Gerätes zu ändern. Eine Übersicht über die in openHAB vorhandenen Items ist in Tabelle 1(Seite 19) zu finden. Die Items werden in den „*.itmes“ Dateien verwaltet. Auf das Konzept der Items und Things wird im Kapitel 2.3 auf Seite 17 noch einmal detailliert eingegangen.

Sitemap In der Sitemap wird der Aufbau des Benutzerinterfaces und der Apps definiert. Sie verwaltet die Darstellung der vorhandenen Items, wie sie in der Benutzerebene dargestellt werden. Die Konfiguration findet in der „*.sitemap“-Datei statt. Im System kann mehr als eine Sitemap existieren. Aus der Benutzerebene heraus kann sie nur gelesen werden.

Scripts Scripts sind in openHAB kleine Programme, um Kommandos an Items zu übermitteln. Es gibt zwei Arten, die Scripts und die Regeln. Letztere werden in den „*.rules“ Dateien gespeichert. Sie sind eine Kernkomponente des smarten Hauses. In ihnen ist hinterlegt, wie sich das Haus in definierten Situationen verhält. Eine Regel könnte zum Beispiel sein, dass das Haus ab einer gewissen Helligkeit in den Nachtmodus geht und sich die Zimmerbeleuchtung nur noch gedimmt einschalten lässt. Regeln können von Items gestartet werden, als auch diese schalten.

Skripts werden in „*.scripts“-Dateien gespeichert. Im Gegensatz zu Regeln wird ihr Ablauf nicht permanent überwacht, sondern aus anderen Programmen wie dem Google Kalender, von XMPP[6] -Befehlen oder auch aus Regeln heraus aufgerufen[17]. Skripte dienen zum Schalten von Items.

Persistence Die Persistenz in openHAB dient zum Speichern von Daten. Der open Home Automation Bus bietet verschiedene Dienste zum Anbinden von Datenbanken an. Hierbei kann nicht nur schreibend, sondern auch lesend auf die Daten zugreifen werden, um beispielsweise einen zeitlichen Verlauf darzustellen. In den DatenBanken wird der Zustand von Items gespeichert.

2.2.3. Abstraktionsschicht

Die Abstraktionsschicht ist die Vermittlungsschicht zwischen dem openHAB-System und den Geräten. Auf dieser Schicht wird die Kommunikation mit den Geräten realisiert. Zudem enthält sie Kernkonfigurationen, die Geräte spezifisch sind. Auf der Ebene befinden sich die Things, die Services sowie das Konfigurationsinterface und das REST-Interface.

Things Die Things sind eines der Kernelemente von openHAB. Ein Thing repräsentiert hierbei ein smartes Gerät oder einen externen Webservice auf den openHAB zugreifen kann. Für die Kommunikation mit der Konfigurationsschicht stellt das Thing Channel zur Verfügung. An diese können die Items gebunden werden. Der open Home Automation Bus kommuniziert über die Items mit dem Thing. Things können über das Konfigurationsinterface ins System integriert werden, oder in Textdateien eingerichtet werden.

Konfigurationsinterface Das Konfigurationsinterface stellt dem Nutzer eine grafische Schnittstelle zur Konfiguration des Systems zur Verfügung. Diese ist das Paper UI, welches über einen Webbrowser aufgerufen werden kann. Dieses registriert die

[6] XMPP steht für Extensible Messanging Presence Protocol und ist Open Source. Es wurde als Alternative zu proprietären Messenging Protokollen entwickelt[41].

Thing im System und verbindet sie mit Items. Auch können über das Interface Bindings geladen werden, um Geräte weiterer Hersteller über openHAB anzusprechen. Zuletzt wird über das Interface die Gerätesuche und die Inbox verwaltet. Letztere ist der Bereich, in dem Geräte abgelegt werden, die von der Gerätesuche gefunden wurden, damit der Nutzer sie als Things registrieren kann.

REST Das REST-Interface ist die vielseitigste Schnittstelle in openHAB. Sie ist sowohl Teil der Abstraktionsschicht als auch Teil der Konfigurationsschicht. Über das Interface lassen sich alle hier beschriebenen Punkte ansteuern. In den meisten Fällen sowohl lesend als auch schreibend.

Service Im Service werden Einstellungen, die ein Binding benötigt, festgelegt. Diese Konfiguration findet über Konfigurationsdateien statt, welche sowohl vom Menschen als auch vom Computer interpretiert werden können. Die Dateien werden mit der Namenskonvention „Bindingname.cfg“ gespeichert. Die Konfigurationen werden vom Thing nur gelesen, können aber vom REST-Interface auch geschrieben werden.

2.2.4. Informationsverarbeitung in openHAB

Nach dem nun die Hauptkomponenten aus openHAB vorgestellt wurden (Abbildung 2), soll nun ihr Zusammenspiel verdeutlicht werden. Der Anwender öffnet über den Browser oder das Smartphone ein Benutzerinterface, welches mit einer Sitemap verknüpft ist. Aus dieser weiß dass System, wie die Items dem Benutzer präsentiert werden. Schaltet der Nutzer nun ein Item, welches beispielsweise mit einer Funksteckdose verknüpft ist, so geht die Info direkt an das Item und nicht über die Sitemap. In den Scripts wird nun geprüft, ob das geschaltete Item eine Regel aktualisiert. Ist dies der Fall so kann die Regel weitere Items schalten. Zudem meldet das Item dem Thing, dass es verändert wurde, welches nun die Steckdose schaltet.

2.3. Das Konzept der Things und Items im Detail

Eines der grundlegenden Konzepte von openHAB ist das Konzept der Things und Items. Das Thing wird in einer XML[7] -Datei beschrieben. Aus der XML-Datei wird eine Instanz erstellt, der die Eigenschaften und Konfigurationen automatisch von openHAB angehängt werden. Der Zugriff auf das Thing ist damit nicht direkt, sondern nur über Handler-Klassen möglich.

[7] XML steht für Extensible Markup Language dies ist ein Textformat um Informationen auszutauschen. Es wird von der XML Working Group entwickelt.[37]

Ihre Eigenschaften stellen die Things in sogenannten Channeln zur Verfügung. Sowohl das Thing, als auch der Channel werden in openHAB über Unique Identifier kurz UIDs angesprochen. In openHAB gibt es zwei Arten von Things; das normale Thing, welches meist ein konkretes Gerät oder einen Dienst repräsentiert und das BridgeThing. Letzteres ist eine Schnittstelle, um Daten zwischen Things und Geräten auszutauschen. Dem BridgeThing kann dabei ein physikalisch vorhandenes Gerät zugeordnet sein, wie es zum Beispiel bei der Philips Hue der Fall ist. Philips verkauft über die Hue Serie Lampen, die sich über den Zigbee Funkstandart fernsteuern lassen. Für die Kommunikation wird eine Brücke (HueBridge) eingesetzt, welche die relevanten Netzwerkdaten an die Lampen weiterleitet. Beim Hue Binding stellt das BridgeThing die Verbindung zur HueBridge her, über diese, werden die Hue-Lampen angesteuert.

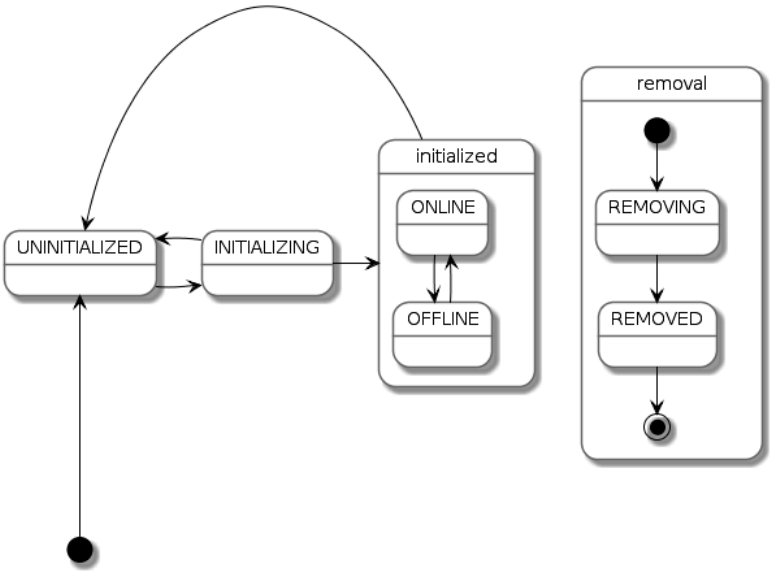

Abbildung 3: Lebenszyklus eines Things aus [8]

Things besitzen wie OSGi-Bundels einen Lebenszyklus (Abbildung 3). Dieser besteht aus drei Kategorien , *uninitalized*, *initializing* und *initialized*. Wobei sich letzterer noch in die Zustände *Online* und *Offline* untergliedert. *Uninitialized* ist ein Thing, wenn es gerade erst erkannt ist. Mit der Initialisierung wechselt es in den Zustand *initalising*. Schlägt die Einrichtung des Things fehl, zum Beispiel weil die Konfigurationsparameter fehlen, kehrt es zurück in den Zustand *uninitialized*. Nach erfolgreicher Initialisierung ist das Thing *Online*. Kommt es im Betrieb zu Fehlern, so wird in den Zustand *Offline* gewechselt. Aus diesen Zuständen kann es nach *uninitalized* zurückkehren, wenn das Gerät zum Beispiel vom Nutzer abgemeldet wird. Bei dieser Abmeldung durchläuft

das Gerät die Zustände *removing* und *removed*. Letzterer ist dabei für den Nutzer nur sichtbar, wenn ein Thing nicht automatisch vom System erkannt, sondern vom Nutzer per Textdatei angelegt wurde[9].

Die Channels werden in openHAB an passende Items gebunden. Diese können über Schnittstellen der Konfigurationsschicht oder der Benutzerebene angesteuert werden. Es müssen nicht alle Channels eines Things mit Items verbunden werden. Über Items können die Eigenschaften der Geräte dann ausgelesen oder verändert werden. Eine Liste der Verschiedenen Item-Typen ist in der Tabelle 1 angegeben.

Item	**Beschreibung**	**Kommandos**
Color	Ein Farbwert	ONOFF, IncreaseDecrease, Percent, HSB
Contact	Ein Binär Wert kann nur gelesen werden	OpenClose
DateTime	Ein Zeitwert mit Datum	-
Dimmer	Ein Prozent Wert	ONOFF, IncreaseDecrease, Percent
Group	Speichert mehrere Items in einer Gruppe	-
Number	Eine Dezimalzahl	Dezimalzahl
Player	Stellt Kontrolloptionen für Medienplayer Bereit	PlayPause, NextPrevious, Rewind-Fastforward
Rollershutter	Ein Prozentwert nicht direkt veränderbar	UpDown, StopMove, Percent
String	Speichert einen Text	String
Switch	Ein Binären Wert	OnOff

Tabelle 1: Items in openHAB aus [7]

Das Thing in OSGi ist eigentlich ein Interface, welches über das Package „thing“ vom Bundle „org.eclipse.smarthome.core.thing“ exportiert wird. Um den Handlerklassen den Zugriff auf dieses zu ermöglichen. Aufgabe des Bundles ist es alle Things im System zu erzeugen und zu kapseln. Implementiert wird das Interface von der Bundle internen Klasse „ThingImpl“. Diese liegt im Package „internal“ und wird nicht exportiert. Das Thing setzt hier die Prinzipien der Kapselung, Abstraktion und Information Hiding um. Inforamtion Hiding bedeutet, dass die konkrete Umsetzungen der Implementierung nicht nach außen hin sichtbar ist. Die Kommunikation erfolgt nur über Interfaces, die das dahinter Liegende abstrahieren (Abstraktion). Kassen sind hierbei eine Kapsel (Kapselung), in der Methoden und Daten zu einem Objekt verbunden werden[14, Seite 6].

Benötigt ein Handler eine neue Instanz von einem Thing, so wird, über das Interface „ThingHandlerFactory“ dies dem Bundle „org.eclipse.smarthome.core.thing“ mitgeteilt, welches die Eröffnung einer neuen Instanz des ThingImpl intern einleitet. Das Thing und sein Handler liegen damit in verschiedenen OSGi-Bundles und können somit nur über exportierte Interfaces miteinander kommunizieren.

2.4. Das openHAB 2.0 Binding im Detail

Bindings sind im open Home Automation Bus die Software Komponenten, die ein Gerät und sein Thing synchronisieren. Für die Programmierung wird Java verwendet. Die vorgegebene Version von Java ist 1.7 oder, für die Verwendung von Java 1.8 Code, das Compact Profile 2[8]. Zudem wird OSGi in der Version R4.2 eingesetzt.

In Abbildung 4 ist das Binding, nach seiner Erstellung mit Maven, im Kontext zum System dargestellt. Dieses wurde um ein Netzwerkinterface zur Kommunikation mit einem möglichen Gerät ergänzt. Es ist hier anzumerken, dass das gesamte Binding auf der Abstraktionsschicht (Abbildung 2) liegt.

Die Kommunikation mit den höheren Schichten findet über das Thing statt und nicht im Handler. Der Handler empfängt Daten über das Interface ThingHandler welches er Implementiert. Dies wird ihm vom Thing-Bundle vorgegeben. Zum übermitteln der Daten an das Thing steht ihm ein Callbackinterface zur Verfügung. Über das er Daten ans Thing senden kann. Der Handler kommuniziert damit mit dem Thing nach dem Verfahren der Inversion of Control[9]. Dies zeigt, dass er mit dem Thing auf einer hierarchischen Stufe steht.

Die „thing-types.xml“ enthält die Definition des Things. In ihr können die Eigenschaften (Channels) eines Things sowie die Konfigurationsdaten eines Things angelegt werden. Hierbei überlässt openHAB dem Programmierer die Wahl, ob die Things des Bindings alle in eine XML-Datei abgelegt werden oder in verschiedene Dateien pro Gerät. Die Channel-Konfiguration lässt sich hierbei auch von der Deklaration der Things trennen um Mehrfachnutzung der Channel zu ermöglichen. Hierfür ist das Philips Hue Binding ein Beispiel, da es für verschiedene Lampen, unterschiedliche XML-Dateien als Thing Beschreibung besitzt[23]. Aus der XML-Datei wird von openHAB das Thing erstellt.

[8] Das Compact Profile ist eine reduzierte Version von Java, welche mit Java 1.8 eingeführt wurde. Es gibt drei Compact Profiles. Hierbei gilt Compact Profile 1 ist das kleinste Profil und vollständiger Bestandteil des Compact Profile 2. Genauso ist Compact Profile 2 Bestandteil des dritten Profils.[35]

[9] Das Prinzip der Inversion of Control dient in der Softwarearchitektur zur Umkehrung der Abhängigkeiten zwischen zwei hierarchisch auf einer Stufe stehenden Klassen. Hierbei übergibt eine Klasse der anderen Klasse ein Callbackinterface um die wechselseitige Kommunikation zu ermöglichen[14, Seite 29].

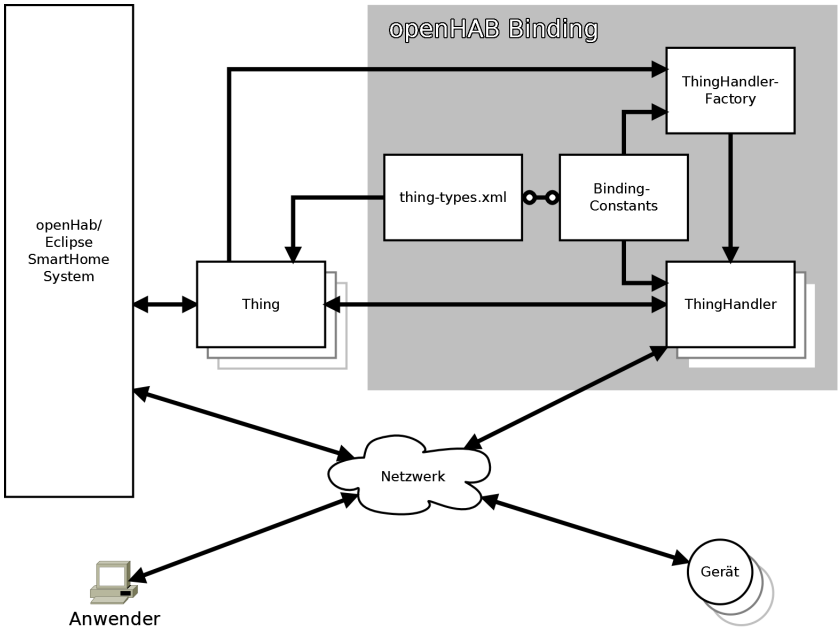

Abbildung 4: Struktur eines openHAB Binding(eigene Abbildung)

In der BindingConstants Datei, werden diese Definitionen für Java lesbar umgesetzt. Zudem werden weitere, für den ThingHandler wichtige Definitionen, über die Things, in dieser Datei gespeichert. Die Daten über das Thing in der XML-Datei und die der BindingConstants müssen identisch sein, damit in Java ein Zugriff auf das Thing möglich ist. Über den ThingHandler wird in Java auf das Thing zugegriffen. Für die Instanz eines Things wird auch eine Instanz von einem Handler aufgemacht. Der Handler ist in diesem Fall auch für die Kommunikation mit dem Gerät verantwortlich.

In Abbildung 5 wird die Struktur des in der Darstellung 4 gezeigten Binding verdeutlicht. Das Binding trägt den Namen **test** dem entsprechend heißt der ThingHandler **testHandler**, die BindingConstants **testBindingConstants** und die ThingHandler-Factory **testHandlerFactory**.

Der ThingHandler testHandler wird im Quellcode Beispiel(Listing 1) gezeigt. Die Methoden **initDeviceCommunication( )**, **setDeviceStatus( )** und **getDeviceStatus( )** dienen hier als Platzhalter für die Kommunikation mit dem Gerät. Der ThingHandler erbt von der abstrakten Klasse BaseThingHandler und implementiert deren abstrakte Methoden. In der Methode **initialize( )** besteht die Möglichkeit, eine Kommunikationsschnittstelle einzurichten. Die Methode **handleCommand( )** wird aufgerufen, wenn das System über ein Item eine Nutzereingabe erhält. In dieser kann das Gerät dann geschaltet werden. Die Methode wird aus der abstrakten Klasse „BaseThingHand-

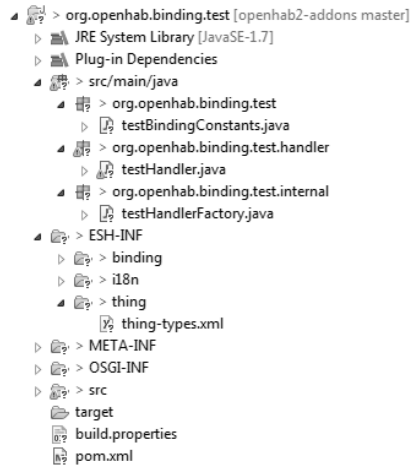

Abbildung 5: Ein openHAB Binding in der Eclipse IDE

ler“ geerbt, welche sie aus dem Interface „ThingHandler“ implementiert. Diese gibt das Thing-Bundle dem Handler vor. Über eine **ScheduledFuture<?>** kann das Thing periodisch ausgelesen werden. Diese greift in openHAB auf einen fest definierten Threadpool zurück. Aus den Programmierrichtlinien von Eclipse SmartHome[4] geht hervor, dass periodische Aufgaben über diese gelöst werden sollen, anstatt über permanent laufende Threads. Dies ist nützlich, um System Ressourcen zu schonen. Der ScheduledFuture wird dabei selbst ein Thread übergeben, der aber nach dem Beenden seiner Tätigkeit terminiert. Das Empfangen der Daten des Things ist hier in der Beispielmethode **getDeviceStatus( )** realisiert, die in der ScheduledFuture aufgerufen wird. Die **dispose()** Methode wird nicht aus der Oberklasse überschrieben, kann aber um ein Thing korrekt zu beenden, nachträglich implementiert werden.

In Zeile 36 wird ein Channel angesprochen. Das dahinter liegende Item ist ein Switch und kann über den OnOffType angesteuert werden. Der Channel des Thing wird über die ChannelUID angesprochen. Da diese nicht in der Klasse hinterlegt ist, wird sie neu generiert. Diese UID setzt sich aus der einzigartigen ThingUID und dem in den Binding-Constants hinterlegten String mit dem Kanalnamen zusammen. Hier entspricht CHANNEL_1 dem String „channel1“.

```
1 /**
2  * The {@link testHandler} is responsible for handling commands, which are
3  * sent to one of the channels.
4  *
5  * @author w_gerlach - Initial contribution
6  */
7 public class testHandler extends BaseThingHandler {
8   private Logger logger = LoggerFactory.getLogger(testHandler.class);
```

```
9   ScheduledFuture<?> updateScheduledFuture;
10
11  public testHandler(Thing thing) {
12    super(thing);
13  }
14
15  @Override
16  public void handleCommand(ChannelUID channelUID, Command command) {
17    if (channelUID.getId().equals(CHANNEL_1)) {
18      boolean onOff = OnOffType.ON.equals(command);
19      setDeviceStatus(onOff);
20    }
21  }
22
23  @Override
24  public void initialize() {
25    initDeviceCommunication();
26    startUpdateScheduler();
27    updateStatus(ThingStatus.ONLINE);
28  }
29
30  private void startUpdateScheduler() {
31    updateScheduledFuture = scheduler.scheduleWithFixedDelay(new Runnable() {
32      @Override
33      public void run() {
34        boolean onOff = getDeviceStatus();
35        if (onOff) {
36          updateState(new ChannelUID(getThing().getUID(), CHANNEL_1),
37                  OnOffType.ON);
38        } else {
39          updateState(new ChannelUID(getThing().getUID(), CHANNEL_1),
40                  OnOffType.OFF);
41        }
42      }
43    }, 5, 5, TimeUnit.SECONDS);
44  }
45
46  @Override
47  public void dispose() {
48    super.dispose();
49    updateScheduledFuture.cancel(true);
50  }
51 }
```

Listing 1: Beispiel eines ThingHandlers

Die Bundles aus openHAB blenden die meisten OSGi-Komponenten aus. Nur das Bundle-Manifest und die XML Komponenten Beschreibung sind im Binding direkt vorhanden. Die openHAB Bindings gehören zu den OSGi Declarative Services. Die Registrierung an der Component Service Runtime befindet sich in der abstrakten Klasse

„BaseThingHandlerFactory.java“,diese wird über das Package „binding“ aus dem Bundle „org.eclipse.smarthome.core.thing“ bereitgestellt. Die Klasse wird von der ThingHandlerFactory des Binding geerbt. Die Registrierung an der Component Service Runtime, sowie die Komponenten Beschreibung sind die Kern Bausteine der Declarative Services[12, Seite 205 & 208]. Declarative Services sind Dienste, die eine Beschreibung ihrer Abhängigkeiten mitbringen, damit das System feststellen kann, ob diese vorhanden beziehungsweise erfüllt sind und der Service ohne Probleme gestartet werden kann[12, Seite 201].

Über openHAB wird ein Thing angelegt. Der Zugriff auf das Thing erfolgt danach aber nur noch über den Handler. Hierzu wird der Handler über Commands(Zeile 16) vom Thing benachrichtigt. Anders herum besitzt der Handler das schon beschriebene Callback Interface. Dies wird über die Methode **updateState( )**(Zeile 36, 39) aufgerufen, welche von der abstrakten Oberklasse des ThingHandlers Implementiert ist. Wird vom Anwender ein neues Thing angelegt, so fragt das Thing Bundle bei der HandlerFactory an ,ob diese ein Thing dieser Art verwaltet. Ist dies der Fall, so teilt diese dem Thing Bundle mit das es die Erstellung starten kann. Das Thing leitet dann die Erstellung seines Handlers ebenfalls über die HandlerFactory ein.

Neben den in Abbildung 4 vorgestellten Klassen existiert noch ein Discovery Service, der verwendet werden kann, um Things automatisch vom System zu erkennen. Mehr zum Discovery Service ist im Kapitel 4.4.1 auf Seite 38 zu finden.

3. Nutzerwelten

Das NutzerWelten System ist aus der Software für das WieDAS-Projekt hervorgegangen. Es basiert wie openHAB auf dem OSGi-Standard und auf Java in der Version 1.7. Entwickelt wurde das System im Labor für Informatik und Embeddedsysteme an der Hochschule Düsseldorf.

3.1. Die Nutzerwelten Architektur

Die Architektur des NutzerWelten Systems besteht aus mehreren Schichten. In Abbildung 6 ist die Architektur, wie ursprünglich im WieDAS-Projekt geplant, zu sehen. In der untersten Schicht liegen die eigentlichen Geräte, welche über die 6LoWPAN-

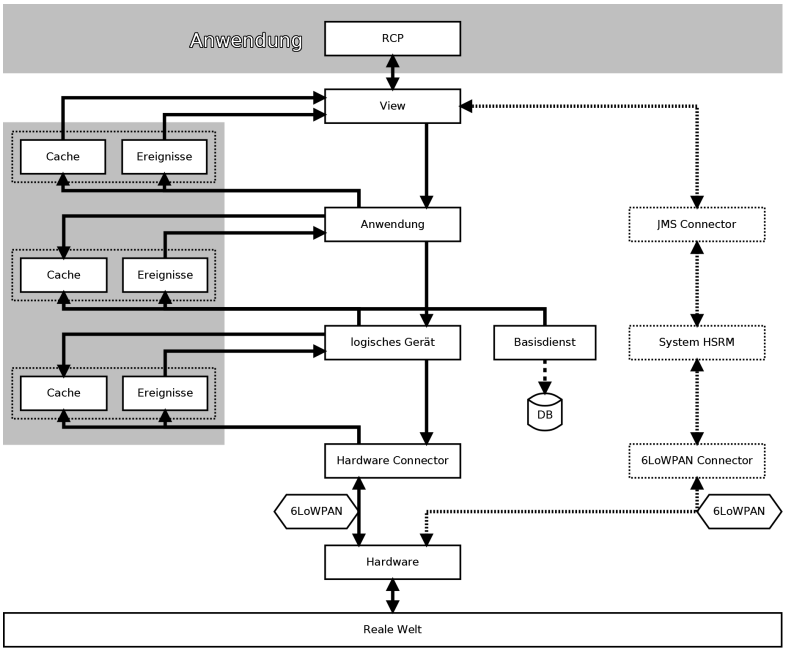

Abbildung 6: Die WieDAS-Architektur aus [20]

Funkschnittstelle mit dem System kommunizieren. Angenommen werden die Daten vom Hardware-Connector, der diese an die verschiedenen Gerätedienste weiterleitet. Für jedes Gerät ist ein eigener Dienst (logisches Gerät) vorhanden, der den Zustand eines Gerätes verwaltet. Hierüber liegen Anwendungsdienste, die die Daten der Geräte auswerten.

In der oberen Schicht liegen die Anzeigedienste für jedes Gerät, die die Ausgabe der Geräteinformationen und Alarme verwalten. Zusammengefasst wird das System in einer Rich Client Plattform (RCP), aus der heraus die einzelnen Dienste gestartet werden. Die

gestrichelten Linien stellen die Verbindungspunkte zum System der Hochschule Rhein Main dar, welche im WieDAS-Projekt mit der Hochschule Düsseldorf kooperierte.

In der aktuellen Umsetzung (Abbildung 7) hat sich die Plattform verändert. Die Alarmierungsdienste sind mit dem Gerätediensten verschmolzen und die Caches nicht umgesetzt worden. Als Basisdienst wurde lediglich die Kommunikation über MQTT realisiert, um die zugehörige Android-App anzusteuern. Persistenz erfolgt nur über Textdateien.

Die Kernkomponenten des NutzerWelten Systems sind die Geräte Views, die logischen Geräte und der Hardware Connector. Die Kommunikation im System verläuft, in der Hierarchie nach unten über Methoden Aufrufe und nach oben über OSGi-Events. Auf die Views, die logischen Geräte und den Hardware Connector soll im folgenden noch einmal detailliert eingegangen werden.

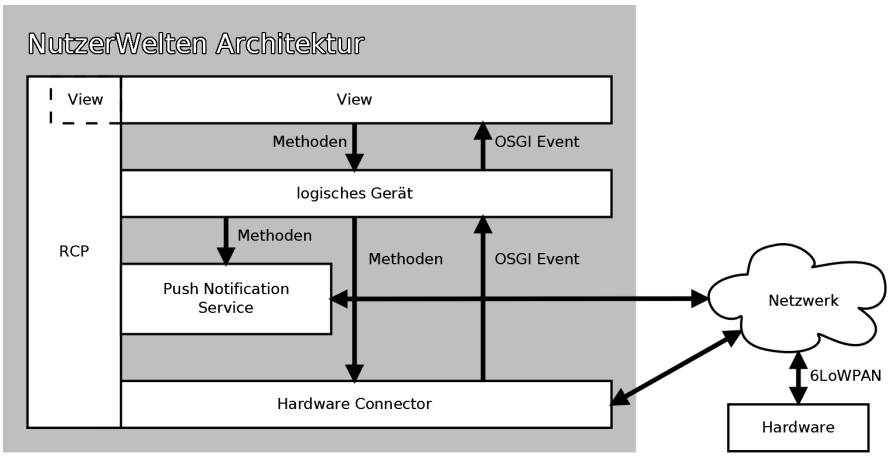

Abbildung 7: Die NutzerWelten-Architektur(eigene Abbildung)

3.1.1. Die View Komponenten

Die View Komponenten sind die grafischen Darstellungen der einzelnen Geräte. Jeder Gerätetyp besitzt ein eigenes View Bundle. Wie bereits erwähnt, wird der Platz für einen Gerätetyp über das RCP-Bundle bereit gestellt und kann nachträglich vom Nutzer verändert werden. Zudem wird in der RCP die LogoView generiert. Über diese werden Projektinformationen für die Testprobanden zur Verfügung gestellt. Die LogoView sorgt für die starke Verknüpfung zwischen View-Komponenten und der RCP. Für jeden Gerätetyp existiert ein eigenes View Bundle und aufgrund der unterschiedlichen Geräte variiert der Aufbau dieser.

In Abbildung 8 ist der Aufbau des Bewegungsmelders gezeigt. Dieses Bundle stellt den kleinsten gemeinsamen Nenner dar und besitzt die Klassen, die allen View Bundels

gemein sind. Das Kürzel BM im Klassenname zeigt an, dass es sich um den Bewegungsmelder handelt.

Das Bewegungsmelderbundle besteht aus zwei Packages dem Package „org.eclipse.wb.swt“ und dem Package „org.wiedas.fhd.views.bewegungsmelder“. Im Package „org.eclipse.wb.swt“ sind SWT[10] spezifische Klassen abgelegt, welche zum Generieren der grafischen Komponenten benötigt werden. Das Package „org.wiedas.fhd.views.bewegungsmelder“ enthält die vom Bewegungsmelder benötigten Klassen. In der Klasse „ConfigDialogBM“ sind Einstellungen für ein angezeigtes Gerät hinterlegt. Der Zustand eines Gerätes wird in der „ItemEintragBM“ gezeigt. Für jedes in der View angezeigte Gerät existiert eine Instanz dieser Klasse. Die Daten Persistenz der View wird in der Klasse „PersistanceBMV“ behandelt. Um eine Verbindung mit einem Gerät herzustellen, wird in der „ServiceBinderBM“ die Verbindung, durch Abonnieren der Geräte Services, des jeweiligen logischen Gerätes, hergestellt. Die Basisdarstellungsklasse ist die „ViewBM“ in dieser wird der eigentliche Aufbau der Ansicht des Gerätetyps festgelegt und die einzelnen Geräteansichten des „ItemEintragBM“ verwaltet.

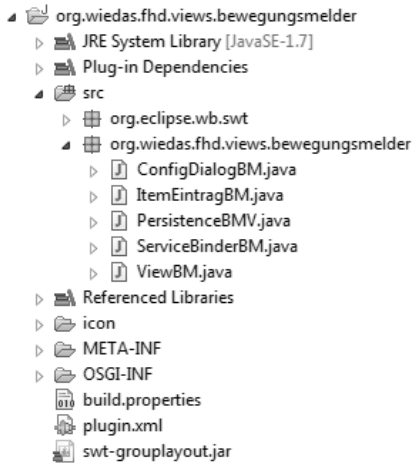

Abbildung 8: Die View in der Eclipse IDE

Zusätzlich zu den hier gezeigten Klassen verfügen einige Bundels noch über die Funktionalität der Text to Speach Engine oder ServiceBinder-Klassen, um Services aus anderen Geräten zu abonnieren. Hierdurch werden Verknüpfungen, wie ein möglicher Alarm bei geöffneter Haustür und eingeschaltetem Herd hergestellt.

[10]SWT steht für Standard Widget Toolkit und ist ein Tool zum Erstellen grafischer Benutzeroberflächen.

3.1.2. Das logische Gerät

Die logischen Geräte sind Verwaltungsklassen. In ihnen wird der Zustand eines realen Gerätes gespeichert. Wie bei den View Komponenten existiert auch bei den logischen Geräten für jeden Gerätetyp ein eigenes Bundle. Auch hier gibt es Unterschiede zwischen dem Aufbau der einzelnen Bundles. In Abbildung 9 ist der Aufbau des Bewegungsmelder Bundles gezeigt. Dieses gliedert sich in zwei Packages auf „org.wiedas.fhd.-bewegungsmelder“ und „org.wiedas.fhd.bewegungsmelder.intf“. Es wird nur das Packet „org.wiedas.fhd.bewegungsmelder.intf“ vom Bundle exportiert, das bedeutet, dass nur über das im Package liegende Interface „IBewegungsmelder“ auf das Bundle zugegriffen werden kann. Im Bundle wird das Interface von der Klasse „DeviceController“ implementiert.

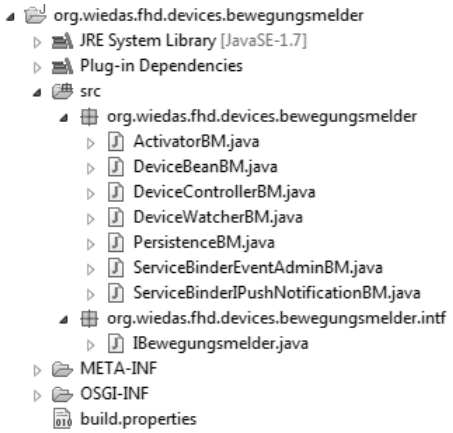

Abbildung 9: Das logische Gerät in der Eclipse IDE

Die Aktivator Klasse „AktivatorBM“ ist eine OSGi Klasse. In ihr befinden sich die Start- und die Stop-Methode des Bundles. Der Zustand eines Gerätes wird in der Speicherklasse „DeviceBeanBM“ abgelegt. Für jedes Gerät wird eine Instanz dieser aufgemacht. Die Beans werden von der Klasse „DeviceControllerBM“ verwaltet. Sie aktualisiert den Zustand des Gerätes. Zur Überwachung der korrekten Funktion dient der „DeviceWatcherBM“. In dieser Klasse wird zum Beispiel geprüft, ob die Geräte in den vorgeschriebenen Abständen Alive-Nachrichten senden. Ist die Komunikation mit einem Gerät gestört, so meldet der DeviceWatcher dies an die View. Für die Datensicherung ist die Klasse „PersitenceBM“ verantwortlich. Die Service Binder Klassen „ServiceBinder-EventAdmin“ und „ServiceBinderPushNotification“ binden den Service für OSGi-Events sowie, den für Push Notifications an. Ersterer ermöglicht es Events im System zu verschicken und Letzterer führt die Kommunikation über MQTT durch.

3.1.3. Der Hardware Connector

Der Hardware Connector ist das Bundle, welches die Kommunikation mit den Geräten übernimmt. Die Geräte des NutzerWelten System verwenden zur Kommunikation den 6LoWPAN-Funkstandart. Dieser lässt sich leicht in ein IPv6-Netzwerk bridgen. Für die Kommunikation zwischen den Geräten und der NutzerWelten-Software werden UDP[11] -Multicasts verwendet. Der Hardware Connector kann damit die Geräte über normale Netzwerkkommunikation erreichen. Die Umsetzung der Kommunikation erfolgt über einen Funkstick, der mit dem Computer oder einem Router verbunden ist.

Das Bundle ist in Abbildung 10 gezeigt. Es besteht aus den Packages „org.wiedas.fhd.-connector.ip“, „org.wiedas.fhd.connector.ip.intf“ und „org.wiedas.fhd.connector.service“. Wie bei den logischen Geräten, wird von dem Bundle nur das Package „org.wiedas.fhd.-connector.ip.intf“ exportiert. Dieses wird von der Klasse „Communication“ implementiert. Das Package „org.wiedas.fhd.connector.ip“ enthält die funktionalen Klassen. Die

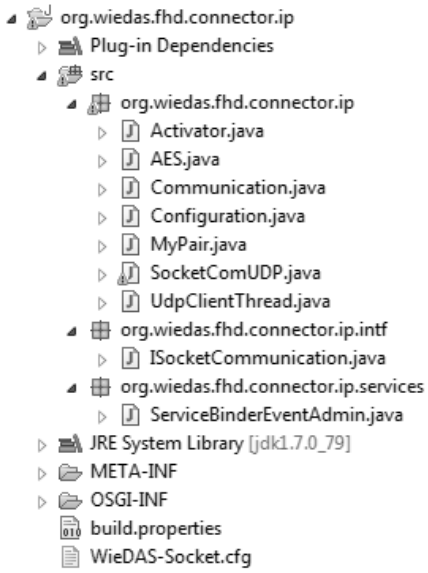

Abbildung 10: Der Hardware Connector in der Eclipse IDE

„Activator“-Klasse ist OSGi-spezifisch und dient zum Starten und Beenden des Bundles. Mit der Klasse „AES“[12] wird die Verschlüsselung über den gleichnamigen Verschlüsselungsstandard realisiert. Das NutzerWelten System ist in der Lage, die Pakete sowohl

[11] Das User Datagram Protocol (UDP) ist ein Protokoll zur Übertragung von Datenpaketen. Es setzt dabei auf keine feste Verbindung zwischen Sender und Empfänger. Für das Ankommen der Pakete besteht damit keine Garantie[19].

[12] AES steht für Advanced Encryption Standard und ist ein leichtgewichtiger Verschlüsselungsstandard der auf 8-Bit Systemen ebenso implementiert werden kann wie auf Großrechnern.[33]

verschlüsselt, als auch unverschlüsselt herauszusenden. Die Klasse „Communication“ codiert die an das Gerät zu übertragenden Daten nach dem NutzerWelten Protokoll und gibt sie an die Klasse „SocketComUDP“ weiter. Über diese werden die Daten dann als UDP-Pakete versendet. Mehr zum NutzerWelten Protokoll ist im folgenden Kapitel zu finden. Die „Configuration“ Klasse dient zur Sicherung und Wiederherstellung der Kommunikationsdaten. Die Klasse „MyPair“ ist eine Speicherklasse für Kommunikationsinterfaces. Die Verwaltung wird nötig, wenn mehr als eine aktive Netzwerkschnittstelle im System vorhanden ist. Wird von der Klasse „SocketComUDP“ ein DatenPacket empfangen, dekodiert es der „UdpClientThread“ und verschickt es als OSGi-Event. Der Service hierzu wird über den „ServiceBinderEventAdmin“ abonniert. Dieser liegt im Package „org.wiedas.fhd.connector.service“.

3.2. Das NutzerWelten Kommunikationsprotokoll

Das NutzerWelten Kommunikationsprotokoll ist das Protokoll, mit dem die Geräte mit der Software kommunizieren. Die Kommunikation findet auf drei Schichten statt. Die oberste Schicht ist das eigentliche NutzerWelten Protokoll. Die unterliegende ist das User Datagram Protocol. Zu unterst liegt der 6LoWPAN Funkstandard oder der IPv6 Standard. Für die Kommunikation zwischen Software und Geräten muss also nur die unterste Schicht ausgetauscht werden. Dies wird im Ravenstick durchgeführt. Dieser ist ein Funkstick für USB, der die eingehenden Daten des 6LoWPAN Funkstandards in ein IPv6 Netz brückt und vice versa.

Die NutzerWelten Geräte kommunizieren hierbei über UDP-Multicasts mit dem System. Dies ist sehr Energie effizient, da die Geräte keine Verbindung halten müssen. Wenn ein Gerät seinen Zustand übertragen hat, kann es sich wieder in den Stromsparmodus versetzten. Die meisten NutzerWelten Geräte kommunizieren dabei nur unidirektional. Lediglich die Steckdose besitzt eine bidirektionale Kommunikation. Für das Protokoll gibt es die Möglichkeiten verschlüsselt und unverschlüsselt zu senden, die Geräte unterstützen zur Zeit aber nur die unverschlüsselte Kommunikation. Diese erfolgt über Multicasts auf dem Port 4321.

Der Aufbau des Protokolls ist simpel gehalten. Die Informationen der Geräte werden im Klartext und für den Menschen lesbar gestaltet. Ein Beispiel für die Kommunikation sieht wie folgt aus:

$$p014 : sd : 0005 : eingeschaltet$$

Der erste Buchstabe ist hier ein ‚p‘ für Plaintext zu deutsch Klartext, alternativ könnte hier auch ein ‚e‘ für encrypted (verschlüsselt) stehen. Die nächsten drei Buchstaben

geben die Länge der Nachricht in Hexadezimalzahl an. Diese wird nach dem ersten Doppelpunkt gezählt. Es folgen die Gerätekennung hier ‚sd' für Steckdose sowie die Nummer des Gerätes 4-Stellig in hexadezimal. Zuletzt folgt die eigentliche Nachricht. Die Kennungen für die Geräte sind in der Tabelle 2 aufgezeigt.

Die Kommunikation ist nicht gerichtet. Die Geräte kennen die Nachrichten, die sie senden, beziehungsweise erhalten müssen und filtern so die Kommunikation für sich heraus. Am Bezeichner „$sd : 0005$" lässt sich nicht erkennen, ob die Kommunikation von der Steckdose kommt oder zu dieser gesendet wird.

Geräte Bezeichner	**Geräte Name**
bm	Bewegungsmelder
fd	Fensterkontakt
hu	Herdüberwachung
nk	Notfallknopf
sd	Steckdose
wm	Wassermelder

Tabelle 2: NutzerWelten Geräte und ihre Bezeichner

3.3. Probleme mit der NutzerWelten Architektur

Das NutzerWelten System zeigt mehrere Probleme. Eines ist der Persistenzdienst. Ursprünglich ist für die Persistenz eine Datenbank vorgesehen worden. Diese wurde allerdings in den laufenden Testsystemen nicht realisiert. Stattdessen wurde die Persistenz in Textdateien ausgelagert. In diesen werden nur die Geräte und ihr letzter Zustand gespeichert. Eine Fehlerprüfung in Form einer Checksumme gibt es hierbei nicht, was dazu führt, dass die Software nach einem Beenden in manchen Fällen nicht mehr korrekt startet.

Wie aus der Architekturabbildung 7 auf Seite 26 hervorgeht, ist das RCP-Bundle an alle Bundles eng gekoppelt. Diese Kopplung ist als ein Import des Bundels über die OSGi Importfunktion des *Require Bundel* realisiert. In OSGi gibt es zwei Möglichkeiten auf ein externes Bundle zu zugreifen, über *Imported Package* oder *Require Bundle*. Beim Import über die *Import Package* Funktion wird nur ein exportiertes Teilpaket des Bundles importiert. Diese Art des Imports ist wesentlich dynamischer, da ein Paket nicht an ein Bundle gebunden seien muss und somit Funktionen durch austauschen der Pakete sehr leicht verändert oder angepasst werden können. Zudem ist der *Package Import* nicht versionsgebunden. Der Import einer Funktionalität über *Require Bundle* bindet diese an

das Bundle. Hierdurch wird nicht nur ein exportiertes Paket importiert, sondern alle exportierten Pakete des Bundles. Zudem werden auch die Abhängigkeiten des Bundles mit importiert. Auch kann der Import mehrerer Bundles, die die gleichen Pakete exportieren, zu Fehlern in der Verarbeitung führen[12, Seite 82 & 83].

Ein weiterer Nachteil des NutzerWelten Systems ist, dass es keine Plattformunabhängigkeit besitzt. Das System ist nur auf das Betriebssystem Windows von Microsoft beschränkt und benötigt eine 32 Bit Java Virtual Machine. Eine hierfür ausgemachte Ursache ist das SWT-Bundle (org.eclipse.swt) in der Version 3.7.1.v3738a. Dieses OSGi-Bundle benötigt für den Betrieb unter Windows zusätzlich das nicht Plattform unabhängige Bundle org.eclipse.swt.win32.win32.x86 mit gleicher Versionsnummer, welches einige der Systemaufrufe direkt auf Windows-Aufrufe umsetzt. Dieses Bundel ist nicht nur von dem Betriebssystem abhängig, sondern auch von der Prozessorarchitektur(32-Bit oder 64-Bit Prozessor). Hier sollte noch angemerkt werden, dass das Bundle noch aus dem damaligen Eclipse Incubator stammt, was bedeutet, dass es zur Einsatzzeit noch in einem unfertigen Entwicklungszustand war. Neuere Versionen dürften dieses Problem nicht mehr besitzen.

4. Integrationsansätze

Es gibt zwei mögliche Ansätze zur Integration des NutzerWelten Systems in openHAB. Ein paralleler Ansatz in dem beide Systeme nebenher laufen und ihre Daten austauschen. Das openHAB System steuert in diesem Fall die NutzerWelten Software. Der andere Ansatz ist der, die Funktionalität des NutzerWelten Systems vollständig in openHAB zu integrieren und Die Geräte dann direkt von openHAB aus zu steuern. Bei einer vollständigen Portierung müsste überprüft werden, ob Bundles aus dem NutzerWelten System in openHAB übernommen werden können.

4.1. Der parallele Ansatz

Für den parallel laufenden Ansatz ergibt sich aus der Abbildung 7 der Push Notification Service als eine mögliche Schnittstelle. Dieser dient dem System bereits zur Kommunikation mit der Android App. Über den Push Notifications Service ist es möglich, MQTT-Nachrichten zu versenden und zu erhalten. Ein Eingriff in die NutzerWelten Architektur ist damit nicht mehr notwendig. Nur die openHAB-Komponenten müssten noch erstellt werden. Ausgehend vom Aufbau des open Home Automation Bus muss für jedes NutzerWelten Gerät ein Thing erstellt werden. Damit werden die Daten in beiden Systemen redundant gespeichert. Dies ist vergleichbar mit dem System der Philips Hue oder dem Hausautomatisierungsdienst von Homematic. Auch hier werden die Daten redundant in den Bridge Geräten (Hue-Bridge oder Homematic CCU) und im Thing gespeichert.

Der Nachteil in diesem Ansatz liegt darin, dass für die Kommunikation über MQTT ein externer Broker gebraucht wird. Dies ist ein Server, der die eingehenden Daten an die Geräte verschickt, die diese abonniert haben. Man spricht hier auch vom Publish and Subscribe Verfahren. Ein System veröffentlicht(publish) seine Daten über einen Broker, der diese an die Abonnenten (subscribe) verteilt[15].

Um der Kommunikation über einen externen Broker aus dem Weg zu gehen, wäre auch die Implementierung eines Kommunikationsbundles in das NutzerWelten System möglich. Dieses könnte dann über das lokale Netzwerk mit dem open Home Automation Bus kommunizieren. Hierfür wären sowohl eine Implementierung im open Home Automation Bus als auch im NutzerWelten System nötig.

Der parallele Ansatz ist ein Ansatz nach dem ursprünglichen Konzept von openHAB, als Vermittler zwischen den einzelnen Plattformen[22, Seite 144]. Während Systeme die vollständig in openHAB realisiert sind und keine verwaltende Bridge besitzen, erst mit zunehmender Entwicklung von Eclipse SmartHome aufkommen dürften.

Zuletzt macht die fehlende Plattformunabhängigkeit das NutzerWelten System für

den parallelen Ansatz unattraktiv. Der open Home Automation Bus ist auf Rechnersysteme wie dem günstigen Raspberry Pi optimiert und damit sehr ressourcenschonend. Vergleichbare System mit Windows sind meist um ein vielfaches teurer. Aufgrund dieser und der im Kapitel 3.3 genannten Probleme wurde von einer weiteren Verfolgung des parallelen Ansatzes abgesehen.

4.2. Der integrative Ansatz

Bei der vollständigen Integration des NutzerWelten Systems in openHAB werden die NutzerWelten Geräte komplett vom open Home Automation Bus gesteuert. Hierbei bietet es sich an, Teile des NutzerWelten Systems zu übernehmen. Zugriffspunkte wären hier die drei Schichten des NutzerWelten Systems, die View, das logische Gerät und der Hardware Connector, sowie der direkte Abgriff der IP-Kommunikation (Abbildung 7).

Die View im NutzerWelten System ist statisch. Für jeden Gerätetyp werden eine festgelegte Anzahl an Geräten gezeigt. Die Änderung der Anzeige erfordert Eingriffe in den Quellcode. Auch ist die Anzeige stark auf die ausgewählten Geräte fixiert. Eine Anzeige von mehr als acht Geräten wird schwierig, da die View Komponenten dann verkleinert oder einzelne Anzeigen vollständig ausgeblendet werden müssen. Damit ist die vollständige Anzeige aller der Geräte eines Types nicht mehr gegeben.

Das openHAB System bietet zur Konfiguration der Geräteansicht, wie schon erwähnt, die Sitemap an. Die Darstellungen der Sitemap erfolgt über die User Interface. Der open Home Automation Bus stellt dabei nicht einzelne Geräte (Things) dar, sondern nur Items also ausgewählte Eigenschaften.

Die Items lassen sich hierbei über die Sitemaps frei gruppieren und Ansichten verschachteln, sodass bei guter Planung der Nutzer nicht überfordert wird. Zuletzt besteht noch die Möglichkeit, über das REST Interface eine Website als Benutzerschnittstelle anzubinden. Mit den verbesserten Konfigurationseigenschaften der View Komponenten in openHAB wird von einer Portierung der NutzerWelten View-Komponenten abgesehen.

Das logische Device ist eine Klasse zur Datenhaltung und Aktualisierung der View. In NutzerWelten existiert zur Speicherung ein Device Bean. Jeder Geräte Typ besitzt sein eigenes Bean mit eigenen Eigenschaften. In openHAB repräsentiert das Thing ein Gerät. Jedes neue im System registrierte Thing wird als Instanz der Klasse „ThingImpl“ aufgemacht. Erst mit dem zuweisen der Handler und Parameter werden dem Thing seine Geräte spezifischen Eigenschaften zugewiesen. Hierdurch vereinfacht sich die Verwaltung des Things vom System. Sowohl das Thing als auch das logische Gerät dienen dem System zur Datenhaltung. Eine Übernahme des logischen Gerätes in openHAB würde zu einer redundanten Datenhaltung führen, da openHAB das Thing intern für die Zugriffe

benötigt. Auf eine Übernahme des logischen Gerätes in openHAB wird damit ebenfalls verzichtet.

Der open Home Automation Bus bietet verschiedene Kommunikationsstandards wie UPnP[13] an, um mit Geräten zu kommunizieren. Die NutzerWelten Geräte verwenden allerdings einen eigenen UDP basierten Standard. Dieser wird vom Hardware Connector verwaltet. Es bietet sich damit an, den Hardware Connector aus dem NutzerWelten System als Bundle zu übernehmen, um die Kommunikation der Geräte mit openHAB durchzuführen.

4.2.1. Integration des Hardware Connector

Der Hardware Connector ist im NutzerWelten System das OSGi-Bundle, welches die Kommunikation mit den Geräten durchführt. Für eine Übernahme des Connectors muss die Bindung an das RCP Bundle entfernt werden. Dies ist nötig, da diese nicht optional ist und somit alle mit der RCP gekoppelten Bundles mit übernommen werden müssen. Wie aus Kapitel 3.3 hervorgeht, müsste somit die gesamte NutzerWelten Plattform übernommen werden. Zudem geht aus den Eclipse SmartHome Programming-Guidlines[4] hervor, dass Bindungen dieser Art in openHAB zu vermeiden sind.

Nach der Entfernung der Abhängingkeiten aus dem Bundle-Manifest, zeigte sich die Abhängigkeit in der Klasse Communication.java durch eine Debug Variable. Mit der Entfernung dieser Variable war der Connector weiterhin ohne Probleme im NutzerWelten System lauffähig. Zur Integration in openHAB wurde das Bundle zuerst als „jar“ exportiert und dann sowohl in den Karaf-Container importiert, als auch in die Eclipse Entwicklungsumgebung von openHAB.

Für die Importierung eines Bundes im Apache Karaf stellt dieser das Verzeichnis ‚Addons‘ zur Verfügung. OSGi Bundles, die in dieses Verzeichnis abgelegt werden, integriert und startet das System automatisch. Abbildung 11 zeigt die Ausgaben der Konsole des Karaf Container. In der oberen Abbildung ist die Ausgabe nach dem einklinken des Connector-Bundles im System gezeigt. Sowohl die gestrichelte Linie als auch die Verzeichnisangabe sind aus der Initialisierung des Bundles. Die Pfadangabe zeigt zusätzlich das der Karaf Container dem Bundle automatisch ein Cache Verzeichnis zuordnet in dem die Daten des Bundels gespeichert werden. Mit der Verwendung eines unterstützten Logging Dienstes treten diese Ausgaben nicht auf und lassen sich über spezielle Kommandos anzeigen. Der open Home Automation Bus verwendet hierfür den SLF4J-Logger. Der un-

[13] Universal Plug and Play kurz UPnP ist eine Schnittstelle welches die Kommunikation zwischen verschiedenen Geräten und Diensten regelt. Es wurde von Microsoft entwickelt und basiert auf Standards wie XML und HTTP[24]

tere Teil der Abbildung zeigt die Ausgabe des Terminals nach der Eingabe des Befehls „Bundle:list“, über diesen Befehl wird der Status der installierten OSGi-Bundels abgerufen. Das Bundle des Hardware Connectors (IP-Connectors) ist mit der Nummer 174 ohne Fehler in das System aufgenommen worden.

Es zeigte sich auch, dass das Bundle zwar startet, aber keine Kommunikation mit Geräten aufbaut. Dies war auch beim Start des Systems aus Eclipse heraus zu beobachten.

Eine Importierung des Projektverzeichnis des Hardware Connectors in den openHAB Workspace Eclipse schlug ebenfalls fehl. Ein Cross Development zwischen verschiedenen Eclipse Plattformen wäre aufwendig. Da die Debugausgaben zusätzlich stören, und ein erstelltes Test-Binding für openHAB zeigte, dass die generelle UDP-Kommunikation funktioniert, wurde von weiteren Arbeiten am Hardware Connector abgesehen.

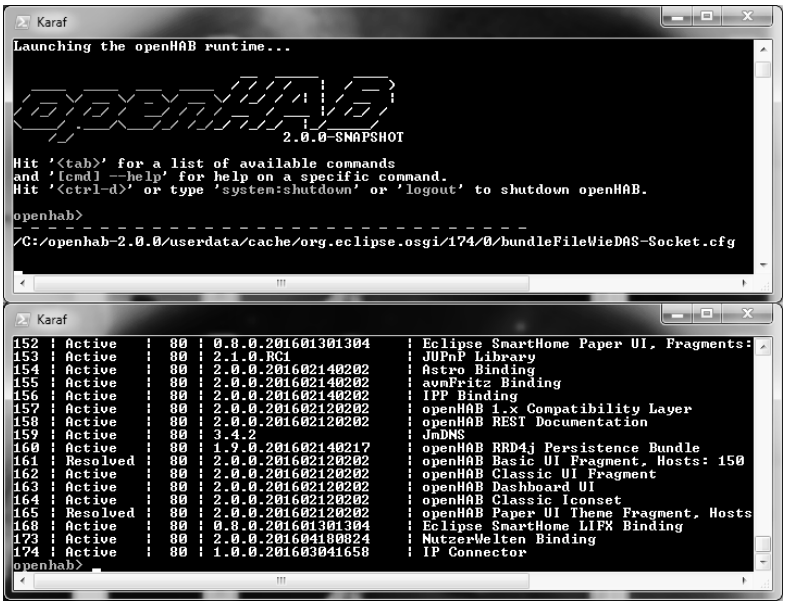

Abbildung 11: Der Hardware Connector im Karaf-Container

4.3. Anforderungen an die neue Architektur

Nach der Untersuchung der Architekturen soll nun darauf eingegangen werden, welche funktionalen Eigenschaften von NutzerWelten auf openHAB portiert werden können, und bei welchen Eigenschaften es zu Problemen kommen kann. Für die Übernahme gilt generell, dass jedes NutzerWelten Gerät, welches in den Demonstratoren zum Einsatz gekommen ist, auch auf das neue System portiert wird. Zu diesen Geräten gehört:

- Schaltbare Funksteckdose

- Herdüberwachung
- Notfallknopf mit Sturzerkennung
- Bewegungsmelder
- Tür-/Fensterkontakt
- Wassermelder

Da für die übrigen Geräte keine NutzerWelten Bundles existieren, oder diese nicht mehr Teil der aktuellen Version sind, werden diese Geräte nicht mit in das neue System übernommen, können aber in späteren Arbeiten portiert werden.

Nicht übernommen werden die Nutzerverwaltung und die Konfigurationstools der Herdüberwachung. Die Benutzerverwaltung ist in openHAB nicht auf der Abstraktionsschicht (Abbildung 2) möglich, ein Bundle mit einer Benutzerverwaltung müsste aber auf der Benutzerebene arbeiten, beziehungsweise den Verkehr auf der Konfigurationsschicht zur Benutzerebene überprüfen. Zudem ist ein Zugriffskontrolldienst zum Schutz vor Fremdzugriffen Teil von openHAB 1, sodass anzunehmen ist, dass dieses in näherer Zukunft auch in openHAB 2 wieder eingeführt wird. Ein Umweg über anwenderbezogene Sitemaps kann hier begrenzt Abhilfe schaffen, dies würde aber keine echte Benutzerverwaltung ermöglichen.

Die Herdüberwachung wird zur Zeit dieses Projektes ebenfalls überarbeitet, zum aktuellen Zeitpunkt ist es damit nicht möglich, den neuen Funktionsumfang zu beschreiben. Für die alte Version wird nur ein Binding mit der Basisfunktionalität erstellt. Der Email Benachrichtigungsdienst, die Text to Speach Engine und der Push Notification Service werden ebenfalls nicht übernommen, da openHAB hierfür vergleichbare Dienste zur Verfügung stellt.

Auch wird für das System keine Nutzeroberfläche entwickelt, da dies in den Arbeiten von Kim Meuter „Implementierung der Funktionalität des NutzerWelten Systems in openHAB“[32], sowie Ngan youe Juilenne Gaelle „Konzeption einer Benutzerschnittstelle für NutzerWelten in HTML5“[21] im Labor durchgeführt wird. Letztere stellt ein Abbild der Benutzeroberfläche des alten NutzerWelten Systems als Webinterface dar.

4.4. Aufstellung einer neuen Architektur

Im Kapitel 2.4 wurde bereits der Aufbau eines einfachen Ein-Geräte-Bundels diskutiert. Für die Portierung des NutzerWelten Systems wird eine komplexere Struktur benötigt. Auch ist es wünschenswert, ein erweiterbares System zu erstellen, in dem neue Hardware

sehr einfach angebunden werden kann. Es bietet sich an, eine zentrale Kommunikationsverwaltung einzusetzen, ähnlich des Hardware Connectors im NutzerWelten System. Der open Home Automation Bus stellt hierfür das Bridge Thing zur Verfügung. Zudem ist es wünschenswert, den Kommunikations-Standard austauschbar zu halten, damit in Zukunft eventuell auch andere Funkstandards eingesetzt werden können.

Um die Konfiguration zu vereinfachen ist es angedacht, den Discovery Service von openHAB einzusetzen, um Geräte vom System automatisch erkennen zu lassen. Im Folgenden soll auf diese Dienste noch einmal eingegangen werden.

4.4.1. Der Discovery Service

Der Discovery Service ist ein Dienst zur Suche von Geräten, deren Binding bereits im System installiert ist. Findet der Discovery Service ein Gerät, so teilt er dies openHAB mit, welches das Gerät in der Inbox ablegt. Über das Paper UI kann ein Gerät dann als Thing im System hinzugefügt und initialisiert werden.

Es gibt hierbei zwei Möglichkeiten, Geräte über den Discovery Service zu suchen. Die manuelle Suche, die über die Geräte Inbox des Paper UI vom Nutzer ausgeführt wird und die Hintergrundsuche, die automatisch vom System durchgeführt wird. Das Binding muss hierbei den Service unterstützen, ansonsten bleibt nur das manuelle Einrichten eines Gerätes. Der Discovery Service wird in der ThingHandlerFactory erzeugt, welche im Kapitel 4.4.3 vorgestellt wird.

4.4.2. Das Bridge Thing

Die meisten Smart Home Geräte besitzen eigene Interfaces zur Kommunikation. Der Nachrichtenaustausch erfolgt über lizenzlich geschützte Bussysteme oder Funkprotokolle. Durch den heutigen Drang Daten in die Cloud oder auf das Smartphone zu bringen, benötigen die Hersteller eine Kommunikation, die in der Lage ist, die Systeme über das Netzwerk oder Internet zu steuern. Solche Geräte werden Brücken oder Bridges genannt. Damit ist die Brücke die einzige Möglichkeit mit einem Gerät über standardisierte Protokolle zu kommunizieren. Hier legt openHAB das BridgeThing an. Dieses soll mit den Brücken kommunizieren und über diese eine Kommunikation zwischen dem ThingHandler und dem zu steuernden Gerät aufbauen. Es ist dann die zentrale Kommunikationseinheit eines Binding.

4.4.3. Die ThingHandlerFactory

Die ThingHandlerFactory ist eine Klasse, die nach dem Erzeugungsmuster der Fabrikmethode arbeitet. Dies ist ein Konzept, dass dazu dient, eine Klasse zu erzeugen, von dem nur die Oberklasse zur Kompilierzeit bekannt ist. Da Konstruktoren nicht polymorph sind, wird die Erzeugung eines Objektes in eine polymorphe Methode gepackt, diese gibt ein konkretes erzeugtes Produkt zurück. Die Methode wird Fabrikmethode genannt. In einer abstrakten Erzeugerklasse wird die Methode formal definiert und von einer konkreten Erzeuger Klasse überschrieben. Zur Laufzeit erzeugt der konkrete Erzeuger dann die Klasse[14, Seite 243]. In Abbildung 12 ist das Verfahren dargestellt.

Der abstrakte Erzeuger ist in openHAB die Klasse „BaseThingHandlerFactory“ und das zu erzeugende Produkt ist die ebenfalls abstrakte Klasse BaseThingHandler. Beide kommen aus dem Package „binding“ des Bundles „org.eclipse.smarthome.core.thing“. Implementiert werden diese in openHAB im Binding durch die ThingHandler und die ThingHandlerFactory.

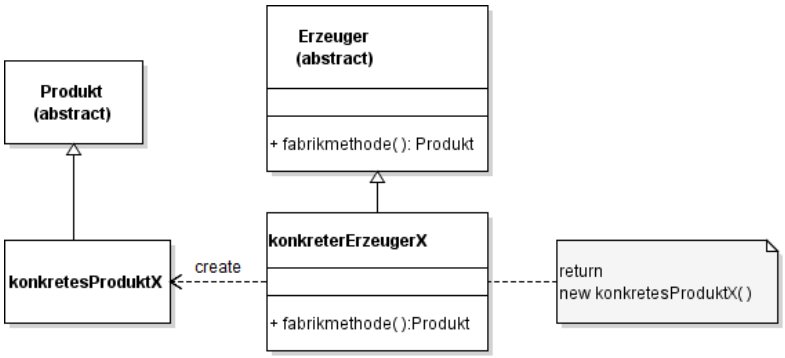

Abbildung 12: Das Verfahren der Fabrikmethode aus [14, Seite 245]

4.5. Die neue NutzerWelten Architektur

Unter der Berücksichtigung der oben genannten Dienste, erweitert sich das System aus Abbildung 4 zu dem in Abbildung 13 dargestellten Bild. Hierbei wurden die NutzerWelten Geräte mit in das System übernommen. Es ist zu erwähnen, dass die NutzerWelten-Things auch alle über einen gemeinsamen ThingHandler gesteuert werden könnten, dies dürfte aber bei einer Erweiterung des Systems unübersichtlich werden. Somit wurde für jedes Gerät ein eigener Händler gewählt. Da das NutzerWelten System nicht als Brücke dienen wird, ist es angedacht ein rein virtuelles Bridge-Thing zur Kommunikation mit den Geräten einzusetzen.

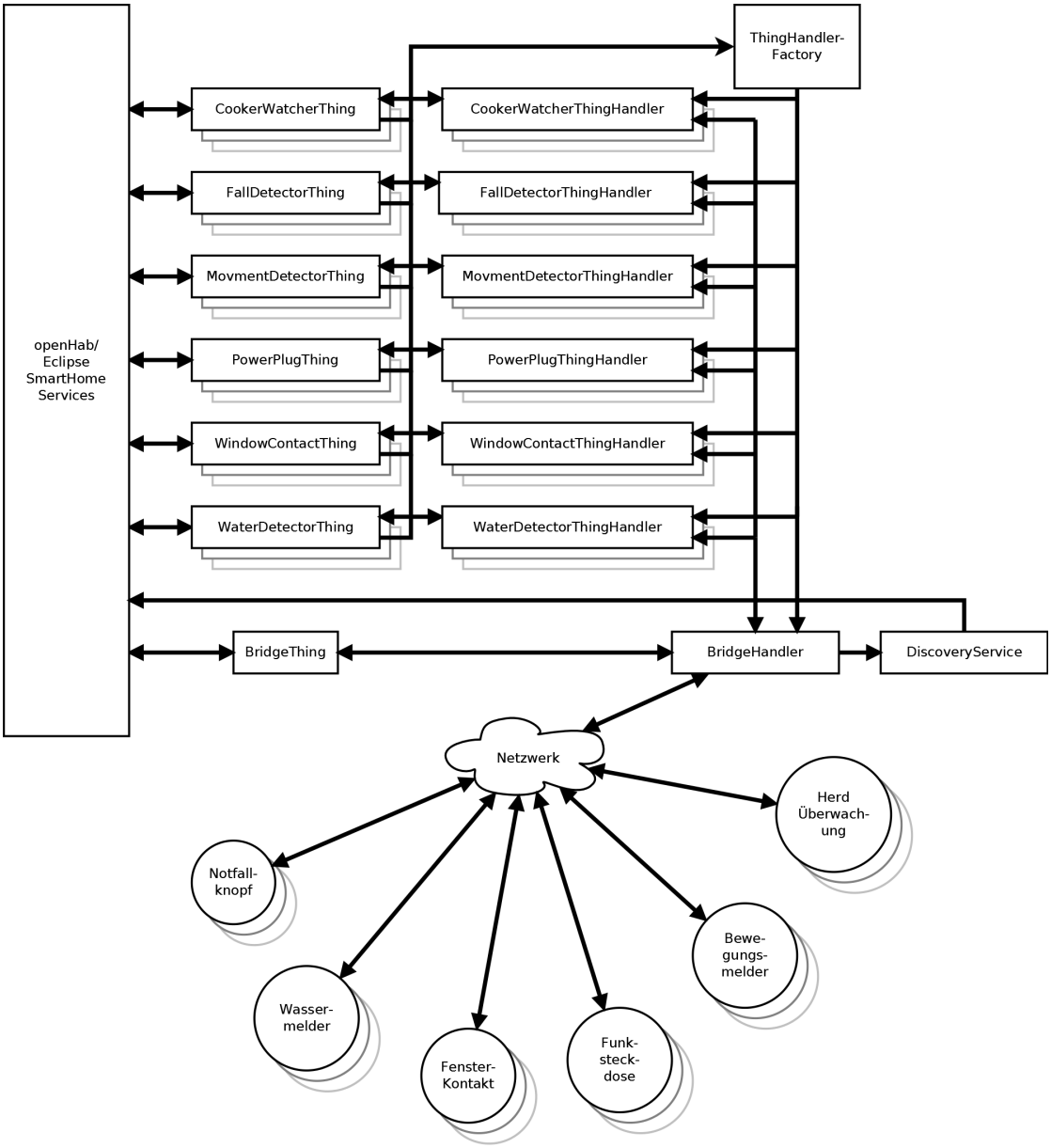

Abbildung 13: Der Rohbau der neuen NutzerWelten Architektur in openHAB (eigene Abbildung)

Die ThingHandlerFactory erzeugt die Handler im System, wenn sie von openHAB dazu aufgefordert wird. Der Discovery Service wird vom BridgeHandler über die Kommunikation informiert und teilt neue Geräte dem System mit. Der Handler der Bridge leitet seine Nachrichten an die ThingHandler weiter, diese wiederum senden ihre Updates an die Brücke zurück.

Hierbei zeigt sich, dass das Bridge-Thing ein Knotenpunkt in dem System darstellt. Die gesamte Netzwerkkommunikation des Binding läuft über die Brücke. Hierdurch wird dieser eine Vermittlerrolle zuteil, die die Kommunikation zwischen Thing und Gerät durchführt. Dies wird in der Abbildung 14 noch besser verdeutlicht. Die Things sowie die übrigen Klassen wurden hier weggelassen.

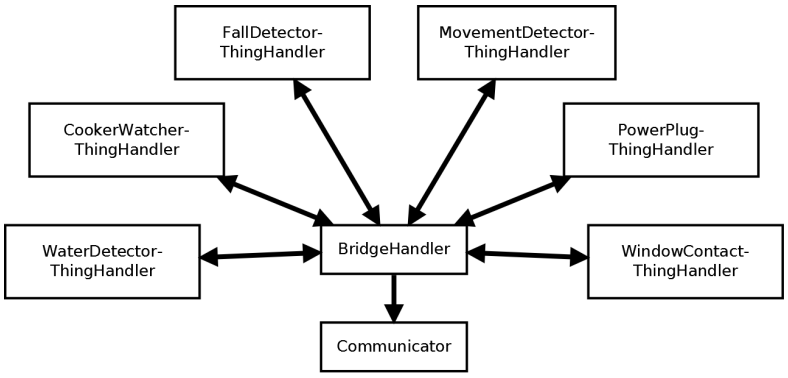

Abbildung 14: Die NutzerWelten Architektur als Vermittler (eigene Abbildung)

Da Informationen sowohl zu den ThingHandlern geführt werden müssen, als auch von diesen weg, empfiehlt sich für die Kommunikation die Brücke zum Vermittler (Mediator) zu machen. Das Verhaltensmuster des Vermittlers ist in der Software Architektur ein Muster, zur Kommunikation zwischen verschiedenen Klassen, die bei diesem Muster Kollegen genannt werden. Ein Kollege teilt einem oder mehreren anderen Kollegen seine Informationen nicht direkt mit, sondern nur dem Vermittler. Dieser kennt die anderen Kollegen und stellt die Nachricht entsprechend zu. Durch dieses Muster müssen sich die Kollegen untereinander nicht mehr kennen. Der Vermittler muss aber jedem Kollegen bekannt sein[14, Seite 181].

Um die Kommunikation mit den Geräten in der realen Welt zu übernehmen, wurde dem BridgeHandler eine Kommunikationsschicht unterstellt. Diese Schicht wurde Communicator genannt. Sie bindet die Geräte an den Vermittler an.

Der Communicator wird der Brücke untergeordnet. Das heißt, dass er keine Methodenaufrufe innerhalb der Brücke durchführen darf. Um Abhängigkeiten im System zu reduzieren dürfen in hierarchisch aufgebauten Systemen, nur die übergeordneten Schich-

ten auf untergeordnete zugreifen. Eine Zugriffsvariante, die es der übergeordneten Schicht ermöglicht, Daten aus der unteren Schicht zu erhalten, ist das Prinzip der Dependency Inversion. Hierbei werden die Zugriffe nur über ein Interface, welches die übergeordnete Schicht der untergeordneten vorgibt, durchgeführt. Das Interface ist damit Teil der oberen Schicht, wird aber von der unteren implementiert. Erfunden wurde das Verfahren von Robert C Martin[14, Seite 25].

Als Kommunikationsknoten verwaltet die Brücke den Discovery Service und teilt diesem mit, dass unbekannte Geräte gefunden wurden. Dies ist möglich, da die Brücke die im System registrierten Geräte als Vermittler kennt. Erhält die Brücke über das NutzerWelten Kommunikationsprotokoll eine Nachricht von einem nicht registrierten Gerät, so kann sie diese, an den Erkennungsservice weiterleiten. Die Discovery gibt die Daten des gefundenen Gerätes an die Services von openHAB weiter, welche die Erstellung des Things und des Handler einleiten.

Unter Berücksichtigung der oben genannten Verfahren, lässt sich ein erstes UML-Kassendiagramm(Abbildung 15) aufstellen. Hierbei wurden zur Übersicht nur zwei NutzerWelten Geräte Handler abgebildet, der ‚NutzerWeltenPlugHandler'(Steckdose) und der ‚NutzerWeltenFallDetector'(Notfallknopf). Zudem wurde die Klasse ‚BaseNutzerWeltenHandler' eingeführt, die als Vorlage für weitere Geräte dient.

Der Vermittler ist der BridgeHandler, dieser implementiert das Interface ‚INutzerWeltenMediator', in dem die Zugriffsmethode auf den Vermittler vorgegeben wird. Die Kollegen sind die einzelnen Geräte, die das Kollegen-Interface ‚INutzerWeltenColleague' über die abstrakte Klasse ‚BaseNutzerWeltenHandler' übergeben bekommen. Die Klasse ‚INutzerWeltenColleague' assoziiert hierbei das Interface ‚INutzerWeltenMediator', das bedeutet, dass allen konkreten Implementierungen der Vermittler bekannt seien muss. In den konkreten Klassen zwischen dem BridgeHandler und den ThingHandlern ist es anders herum, da die konkrete Brücke alle Handler kennen muss. Dies ist durch das Vermittler Verhaltensmuster vorgegeben[14, Seite 182].

Die Brücke kommuniziert mit dem Communicator mittels Dependency Inversion über das Interface ‚ICommunicator'. Dieses muss damit Methoden sowohl für das Senden als auch den Empfang vorgeben. Das Interface ‚ICommunicator' aggregiert die Brücke, da es zu ihr gehört. Die ‚NutzerWeltenHandlerFactory' ist die Klasse, welche die Handler erstellt. Ihre Abhängigkeiten liegen außerhalb des Bundle und sind hier nicht eingezeichnet.

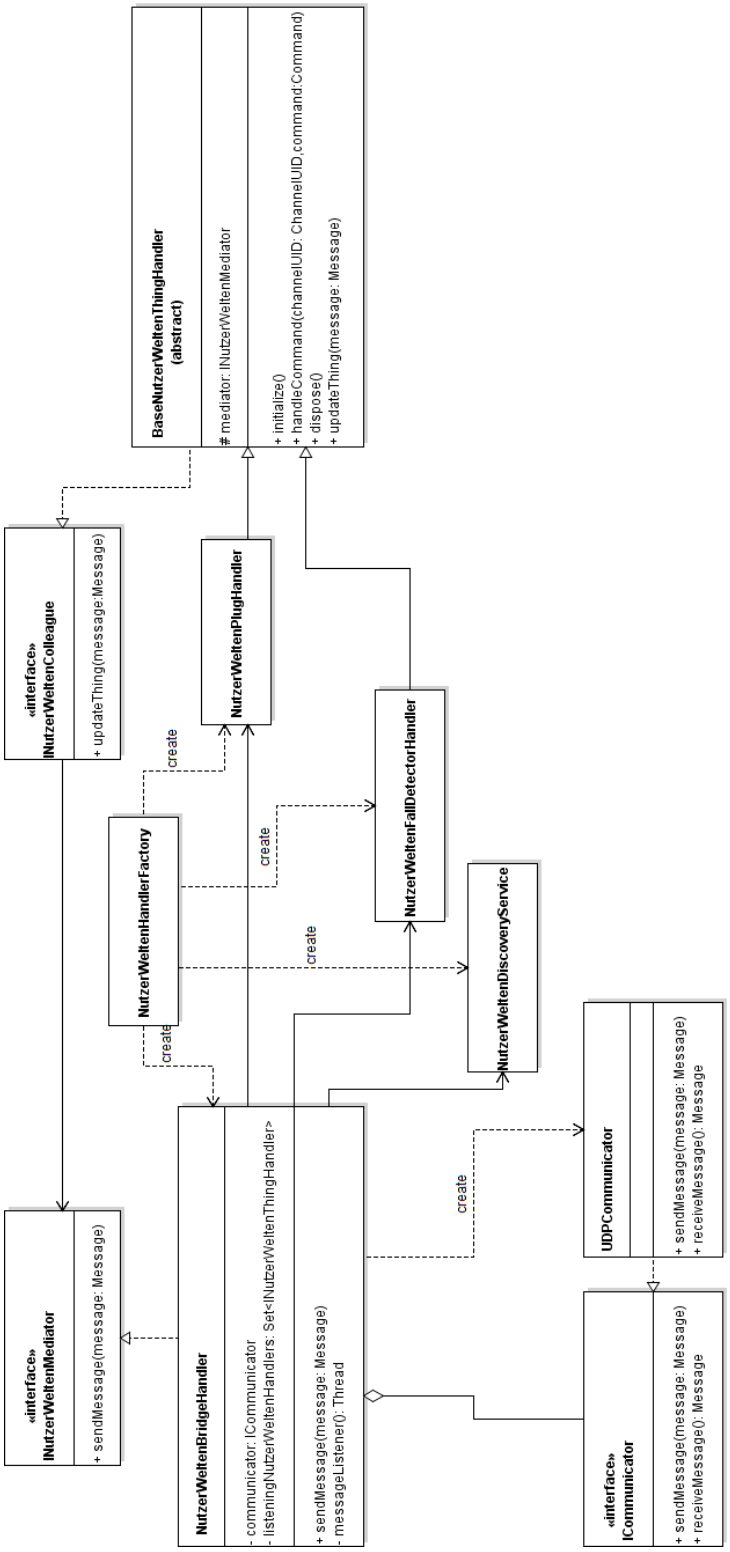

Abbildung 15: Entwurf der NutzerWelten Architektur in openHAB(eigene Abbildung)

5. Umsetzung

Mit dem Aufstellen der neuen Architektur soll hier, der Entwurf der Klassen beschrieben werden. Hierbei soll auch tiefer auf die Entwurfsmuster eingegangen werden, welche für die Trennung zwischen den Schichten und Komponenten verwendet wurden. Der Quellcode ist im Anhang A der Arbeit zu finden.

5.1. Die NutzerWelten Bridge

Die NutzerWelten Bridge besteht, wie die anderen Geräte Klassen, aus dem Thing und seinem Handler. Das BridgeThing kann wie normale Geräte eigene Parameter und Kanäle besitzen. Im NutzerWelten System hat das Thing weder Channel noch Konfigurationsparameter. Das Thing selbst erfüllt damit nur die Aufgabe, zu signalisieren, dass die Bridge im System existiert. Der BridgeHandler ist das zentrale Element der Architektur. Ihm fällt als Vermittler die Kommunikation mit den Geräten und den ThingHandlern zu. Um diese Aufgaben besser von einander abzugrenzen, wurde der Bridge Handler in zwei Komponenten zerlegt dem Communicator, der Kommunikationseinheit mit den Geräten und dem NutzerWeltenBridgeHandler, als Vermittler zwischen den ThingHandlern. Da der Communicator Teil des BridgeHandlers ist, ist er diesem hierarchisch untergeordnet.

5.1.1. Der NutzerWeltenBridgeHandler

Der „NutzerWeltenBridgeHandler“ ist die Klasse, die die Vermittlerrolle übernimmt. Sie erbt von der Klasse BaseBridgeHandler aus dem Package „binding“ des Bundles „org.eclipse.smarthome.core.thing“ und implementiert das Interface „INutzerWeltenMediator“, welches dem Vermittler die Kommunikationsmethode „sendMessage( )“ vorgibt. Zusätzlich wurden dem Interface Methoden hinzugefügt, über die sich die ThingHandler am Mediator anmelden beziehungsweise abmelden. Das UML Diagramm, mit den Vermittler Abhängigkeiten, wird in Abbildung 16 gezeigt. Auf die Methoden wird im folgenden nach Interfaces beziehungsweise Oberklassen sortiert noch einmal eingegangen. Aufgabe des Vermittlers ist es Abhängigkeiten zwischen den Klassen zu reduzieren.

Für die Bearbeitung der Kommunikation zu den einzelnen ThingHandlern, wurde eine Registrierung geschaffen, an der sich die Handler über das Interface „INutzerWeltenColleague“ registrieren. Für die Zustellung einer Nachricht, kann der Vermittler einen Bezeichner aus den Kollegen auslesen. Dieser ist auch als Ziel in einer Nachricht hinterlegt, damit weiß der Vermittler, an welchen Handler er die Nachricht zustellen muss, wenn sie vom Communicator kommt. Anders herum sind die Geräte für die Erkennung ob eine Nachricht an sie gerichtet ist selbst verantwortlich.

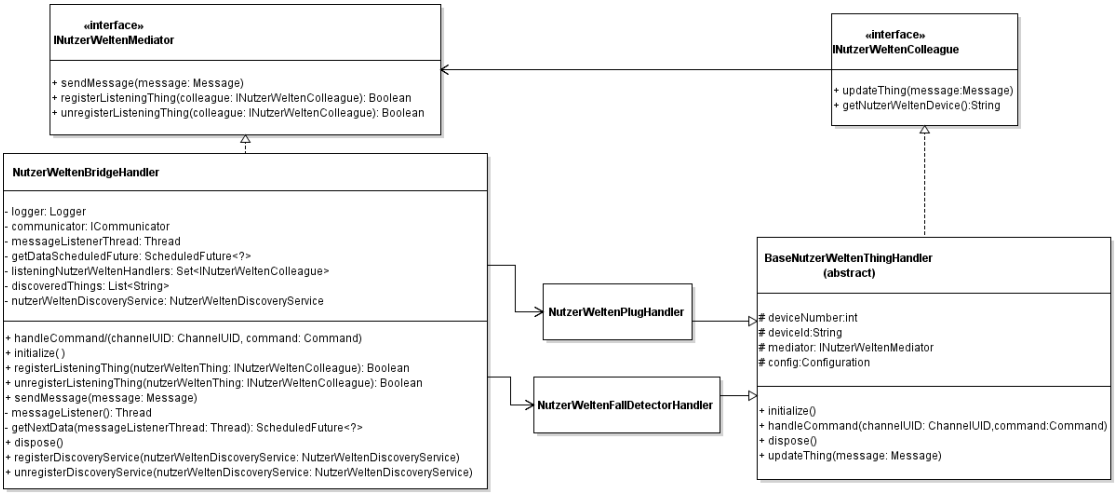

Abbildung 16: Die NutzerWeltenBridge als Mediator in UML

BaseBridgeHandler Ist die abstrakte Basisklasse, die aus openHAB vorgegeben wird.

handleCommand( ) Die Methode ist eine der Basismethoden eines Things in openHAB. Im BridgeThing wird sie allerdings nicht weiter benötigt und hat einen leeren Methodenrumpf.

initalize( ), dispose( ) Die Methode „initialize( )“ dient zur Initialisierung der Klasse, in ihr können zum Beispiel auch die Konfiguration aus dem Thing ausgelesen werden. Im NutzerWelten Binding wird hier der Communicator initialisiert, sowie eine ScheduledFuture gestartet, welche die eingehenden Daten aus dem Communicator entgegen nimmt. Die ScheduledFuture ist eine Java Klasse, über die sich zeitgesteuert Threads starten lassen. In openHAB werden meist terminierende Threads über eine ScheduledFuture gestartet. Der Vorteil hierbei ist der, dass die ScheduledFuture aus dem Java Executor Framework bereitgestellt wird und sich damit, der Threadpool besser verwalten lässt[39]. Die Methode „dispose( )“ wird aufgerufen, wenn ein Thing beendet wird. In ihr wird die ScheduledFuture abgebrochen und der Communicator gestoppt.

INutzerWeltenMediator Ist das Vermittler Interface über das die ThingHandler Messages an den Vermittler senden.

sendMessage( ) Die Methode wird von einem Kollegen(ThingHandler) aufgerufen um Nachrichten an den Vermittler zu senden. Der Vermittler weiß, dass wenn ein Thinghandler die Methode aufruft, er die Nachricht an den Communicator

weiterreichen muss. Da dieser in der Hierarchie unter dem BridgeHandler steht, leitet er die Nachricht über das Interface ICommunicator(Abbildung 17) an diesen weiter.

registerListeningThing( ), unregisterListeningThing( ) Mit der Methode „registerListeningThing( )“ wird ein ThingHandler in die Registrierung des Vermittlers eingetragen. Dieser speichert die hierbei mit übergebene Instanz in dem Set „listeningNutzerWeltenHandlers“. Der Mediator weiß damit, dass das Thing vorhanden ist und wird Nachrichten an den Handler weiterleiten. Sendet ein Gerät nun eine Message, wird aus dieser das Ziel ausgelesen und der passende Handler in der Registrierung gesucht. Von diesem wird die Methode updateThing( ) aufgerufen, welche der Handler vom Interface INutzerWeltenColleague implementiert. Wird ein Gerät aus dem System entfernt, so wird es über die Methode „unregisterListeningThing( )“aus der Liste genommen.

Sonstige Methoden aus keiner Oberklasse

messageListener( ), getNextData( ) Über die private Methode „messageListener( )“ wird ein terminierenden Thread zurück gegeben. Dieser fragt beim Communicator an, ob Daten vorhanden sind. Ist dies der Fall, so holt er alle Daten vom Communicator ab und terminiert, wenn dieser keine mehr besitzt. Über die Methode „getNextData( )“ wird der Thread dem Sceduler von openHAB übergeben und mit einer festen Verzögerung nach seiner Terminierung, erneut wieder aufgerufen.

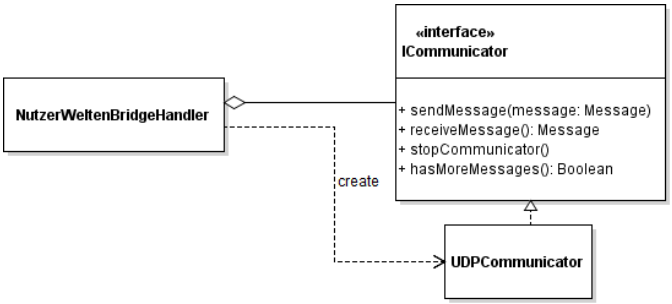

Abbildung 17: Dependency Inversion im NutzerWeltenSystem nach [14, Seite 27]

Die Zugriffe auf den Communicator erfolgen hierbei nach dem Verfahren der Dependency Inversion welches, schon im Kapitel 4.5 auf Seite 39 beschrieben wurde. In dem Fall des NutzerWelten Bindings, pollt der BridgeHandler die Daten aus dem Communicator im Abstand von einer halben Sekunde. In Ab-

bildung 17 ist die Abhängigkeit des Communicators zur Bridge aufgezeigt. Die Klasse, die das Interface ICommunicator implementiert, ist der UDPCommunicator. Dieser stellt die Kommunikation über das UDP-Protokoll bereit.

registerDiscoveryService( ), unregisterDiscoveryService( ) Die Methoden „registerDiscoveryService( )“ und „unregisterDiscoveryService( )“ dienen zur Registrierung und Deregistrierung des NutzerWelten DiscoveryServices am BridgeHandler. Dies ist nötig, da der Handler die Kommunikation mit den Geräten verwaltet und somit auch Nachrichten unbekannter NutzerWelten Geräte erhält. Der BridgeHandler kann damit diese Geräte über den DiscoveryService im System bekannt machen.

5.1.2. Der Communicator

Der Communicator ist der Softwareteil, der die Kommunikation mit den Geräten durchführt. Dieser soll das System in Bezug auf Kommunikationsprotokolle erweiterbar machen. Er ist so entworfen, dass es möglich ist weitere Communicatoren in das System zu implementieren. Um zum Beispiel Brückengeräte mit neuen Funkprotokollen anzubinden. Die Vorgabe für den Aufbau eines Communicators bietet das Interface ICommunicator, welches von einem Communicator implementiert werden muss. Zur Zeit existiert nur der UDPCommunicator welcher, das NutzerWelten Protokoll verwaltet.

5.1.3. Der UDPCommunicator und das Interface ICommunicator

Der UDPCommunicator ist mit dem Interface ICommunicator im Klassendiagramm 18 gezeigt. Das Interface gibt dem Communicator die Methoden vor, über die der BridgeHandler auf diesen zugreifen kann. Die Implementierung ist im UDPCommunicator umgesetzt, der zusätzlich private Methoden zur Verwaltung der eigentlichen Netzwerk Kommunikation besitzt. Da der BridgeHandler die Daten aus dem UDPCommunicator pollt, speichert dieser, die Daten über einen FIFO[14]-Puffer zwischen. Die Implementierten Methoden werden im folgenden beschrieben.

sendMessage( ) Sendet eine von einem BridgeHandler kommende Message an das Ziel Gerät, welches von der Messageklasse vorgegeben wird. Die Kommunikaiton nach außen wird über einen DatagramSocket durchgeführt. Eine Beschreibung der Message Klasse ist in dem folgenden Kapiteln zu finden. Da die Kommunikation über

[14] FIFO steht für First In First Out und ist ein Pufferspeicher in dem das als erstes gespeicherte Objekt auch als erstes wieder entnommen wird[40].

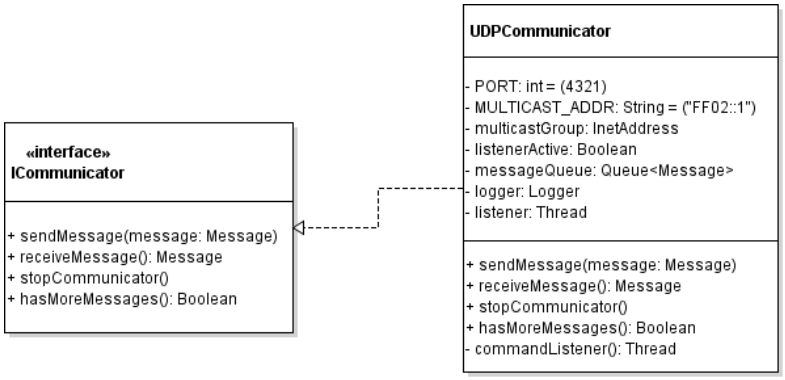

Abbildung 18: Der UDP-Communicator und das Interface ICommunicator in UML

das NutzerWelten Protokoll fest vorgegeben ist, werden der Kommunikationsport und die Broadcast Adresse als Konstanten festgelegt.

receiveMessage( ) Nimmt vom internen Message Puffer ein Element weg. Ist der Puffer leer, wird „Null“ zurück gegeben. Als interner Puffer wird eine Queue (messageQueue) aus der Java.util verwendet. Diese arbeitet nach dem FIFO Prinzip.

stopCommunicator( ) Die Methode dient zum Beenden des Communicators. In ihr wird der Empfangsthread gestoppt und die Queue geleert.

hasMoreMessages( ) Die Methode dient zum überprüfen, ob Nachrichten auf der Queue vorhanden sind. Ist die Queue leer, so wird ‚false‘ zurück gegeben. Sind noch Messages in der Queue, so gibt die Methode ‚true‘ zurück.

commandListener( ) Die private Methode gibt den Thread zurück, der auf Pakete von Geräten im Netzwerk wartet. Der Thread wandelt die eingehenden Datagram-Packets in Messages um und befüllt die „messageQueue“ mit diesen. Problematisch ist der Thread, da er nicht terminiert, sondern durch eine while-Schleife permanent durchlaufen wird. Dies ist nach den Guidlines für Eclipse SmartHome[4], welche auch für openHAB gelten, nicht erlaubt. Während der Durchführung dieser Arbeit konnte keine zufriedenstellende Lösung für das Problem gefunden werden. Problematisch ist hierbei die Tatsache, dass der Java MulticastSocket nur einen Puffer für ein UDP-Datagram Paket besitzt und es keine Methoden gibt, um zu Überprüfen, ob dieser Daten enthält oder leer ist[34]. Wird ein leerer Puffer von einem Thread gelesen, so blockiert dieser bis Daten ankommen. Eine Lösung über eine ScheduledFuture wäre zwar denkbar, ist aber unbefriedigend, da diese erstens sehr schnell pollen müsste, um Datenverlust vorzubeugen und zweitens permanent

am Socket blockiert und damit eine openHAB Systemressource dauerhaft bindet. Auch das Binding von Homematic, sowie das Bundle für UPnP wenden für ihre Kommunikation die Lösung über einen dauerhaften Thread an[36], [2].

5.1.4. Die Kommunikationsklasse Message

Die Klasse Message ist eine Klasse, die zur Kommunikation zwischen dem Bridge Handler, dem Communicator und dem Thing Handler eingesetzt wird. Sie ist an das NutzerWelten Protokoll angelehnt und lässt sich dadurch sehr leicht auf dieses umsetzten. In der Klasse werden die nötigen Parameter zum Zustellen der passenden Geräte oder Handler gespeichert. In Abbildung 19 ist das Klassendiagramm der Message-Klasse aufgezeigt. Im String „device“ wird der Gerätebezeichner aus Tabelle 2 auf Seite 31 abgelegt. Aus dem Integer „number“ ist die Gerätenummer auszulesen. Der String „data“ speichert die zu übertragenen Daten. Die Methoden sind unter der Abbildung aufgeführt.

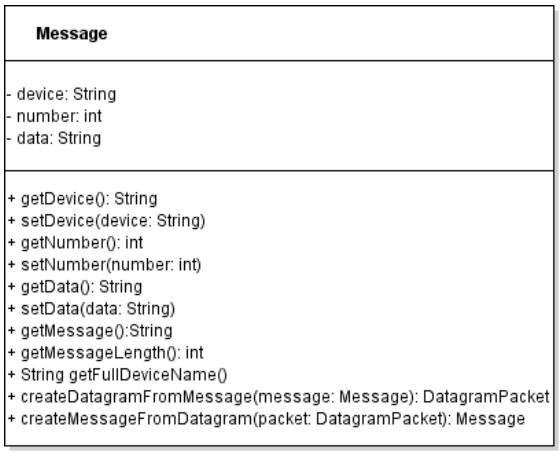

Abbildung 19: Die Message Klasse in UML

getDevice( ), setDevice( ), getNumber( ), setNumber( ), getData( ), setData( ) Die Methoden sind die Getter und Setter für die einzelnen Parameter.

getMessage( ), getMessageLength( ) Über diese Methoden lässt sich die Nachricht für die Konvertierung ins NutzerWelten Protokoll ausgeben. Die Methode „getMessage( )“ gibt das Nachrichtenpaket in der folgenden Form zurück:

$$device : number : data$$

Die Länge der Nachricht wird über die Methode „getMessageLength( )“ bestimmt. Diese wird ebenfalls zur Konvertierung benötigt.

getFullDeviceName( ) Die Methode gibt eine Kennung zurück, die zur Identifizierung des konkreten Gerätes verwendet werden kann. Der Gerätename dient nur eingeschränkt zur Erkennung des Ziels der Nachricht. Das Ziel ist entweder der jeweilige Handler oder das Gerät. Die Zustellung wurde schon im Kapitel 5.1.1 auf Seite 44 beschrieben. Wird die Nachricht vom Communicator empfangen, so dient die Adresse zur Zustellung an den Handler.

Zudem wird der Identifizierer verwendet um das Gerät an openHAB anzumelden. Der Gerätename fließt in die Zuweisung der ThingUID mit ein. Diese UID ist eine einzigartige Kennung, die dazu dient, ein konkretes Thing anzusprechen. Die Form der Rückgabe der Methode „getFullDeviceName( )“ sieht wie folgt aus:

$$device + number$$

Die Komponenten werden durch ein ‚+‘ separiert, da openHAB den Doppelpunkt intern selbst als Trenner in der ThingUID verwendet.

createDatagramFromMessage( ), createMessageFromDatagram( ) Diese statischen Methoden dienen zur Umwandlung von UDP Datagram Paketen in Messages und vice versa. Die Methode „createMessageFromDatagram( )“ überprüft hierbei, ob die eingehenden Datagramme auf dem NutzerWelten Protokoll basieren. Ist dies nicht der Fall, so wird das Paket verworfen.

5.2. Die NutzerWelten Things

Die Geräte des NutzerWelten Systems müssen, damit openHAB mit ihnen umgehen kann, als Things definiert werden. Der open Home Automation Bus generiert diese Things aus XML-Dateien, denen ihre Eigenschaften in Channeln zugewiesen werden. In Tabelle 3 sind die NutzerWelten Geräte mit ihren Channeln aufgelistet. Die XML-Dateien sind nach den Gerätebezeichnern benannt. Um die Wiederverwendung von Channeln zu ermöglichen, wurden diese in die „channel.xml“ ausgelagert. Zur Zeit wird aber lediglich der „battery“ Channel mehrfach verwendet. Auch die rein in Software existierende Brücke hat ein eigenes Thing. Ihr wurde aber kein Channel zugewiesen.

Geräte Name	**Datei**	**Channel**	**Item Typ**
Bewegungsmelder	bm.xml	movmentDetector	Contact
NutzerWelten Brücke	bridge.xml	—	—
Fensterkontakt	fd.xml	windowContact battery	Contact Contact
Herdüberwachung	hu.xml	cookerState	Number
Notfallknopf	nk.xml	fallAlert alive buttonAlert alertImpuls alertReset	Number Contact Contact Contact Switch
Steckdose	sd.xml	onOff	Switch
Wassermelder	wm.xml	water battery	Contact Contact

Tabelle 3: NutzerWelten Things und ihre Channel

5.2.1. Der Aufbau der Things XML-Datei

Jedes Thing in openHAB wird in seiner XML Datei beschrieben. Aus dieser Datei wird dann eine Javaklasse generiert. Dies ist möglich, da die Eigenschaften eines Things durch die Channel von openHAB vorgegeben werden. Diese wurden schon in Tabelle 1 auf Seite 19 vorgestellt. Der Aufbau der XML-Datei soll hier am Beispiel der Nutzerwelten Funksteckdose(Listing 2) erörtert werden.

Im Header der Datei ist die Version und die Zeichencodierung vereinbart. Das XML-Tag <thing:thing-descriptions /> eröffnet, die Beschreibung, in ihm wird gleichzeitig die Binding Zugehörigkeit über die „bindingId“ festgelegt. Zudem wird in dem Tag der Namensraum der XML-Datei angegeben. über das Tag <thing-type /> (Zeile 10) werden dem Thing die Parameter und Channel übergeben. Mit dem Tag wird auch die Type ID (hier gleich „sd“) festgelegt. Über die Binding ID und die Type ID lässt sich die „ThingTypeUID“ der Steckdose bestimmen, welche von openHAB intern verwendet werden kann, um alle Steckdosen des NutzerWelten Bindings zu identifizieren. Mit dem XML-Tag <supported-bridge-type-refs /> werden die Brückentypen spezifiziert die eine Instanz des Things aufmachen können. Auch hier gilt Binding ID und Brücken Type ID ergeben die ThingTypeUID der Brücke.

Die Kanäle der Steckdose werden über das <channels /> Tag (Zeile 17) dem Thing zugewiesen. Ein Kanal verfügt in der Beschreibung über zwei Parameter, der „typeId“ und der „id“. Die ID ist der Parameter, der vom System verwendet wird um die Steckdose

anzusteuern. Über die Kanal Type ID wird dem Kanal, in der „channel.xml“ einem Item Typen zugewiesen.

Das Tag <config-description />(Zeile 20) dient zum Festlegen von Parametern. Diese können bei der Einrichtung des Gerätes abgefragt werden, um für den Betrieb benötigte Einstellungen an das Thing zu übergeben. Im Thing sind die Parameter „device-Tag“ (Zeile 21)und „deviceNumber“ (Zeile 28) festgelegt. Das Tag ist hierbei der Geräte Bezeichner nach Tabelle 2, über den das Gerät angesteuert werden kann. Die Nummer ist entsprechend die Gerätenummer. Über das XML-Tag <required /> wird definiert, dass ein Thing ohne diesen Parameter nicht erstellt werden kann. Die XML-Tags <label /> und <description /> dienen zur Angabe von einem Bezeichner oder einer Beschreibung.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <thing:thing-descriptions bindingId="nutzerwelten"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xmlns:thing="http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
5    xsi:schemaLocation=
6      "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
7      http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">
8
9    <!-- Power Plug Thing Type -->
10   <thing-type id="sd">
11     <supported-bridge-type-refs>
12       <bridge-type-ref id="nutzerweltenbridge" />
13     </supported-bridge-type-refs>
14
15     <label>NutzerWelten remote power-plug</label>
16     <description>NutzerWelten power plug thing </description>
17     <channels>
18       <channel typeId="onOff" id="onOff"></channel>
19     </channels>
20     <config-description>
21       <parameter name="deviceTag" type="text">
22         <label>Name of the device type</label>
23         <description>
24           Name of the device type. Equal for all devices of the same type.
25         </description>
26         <required>true</required>
27       </parameter>
28       <parameter name="deviceNumber" type="text">
29         <label>Number of the device type</label>
30         <description>Uniqe Number of the device.</description>
31         <required>true</required>
32       </parameter>
33     </config-description>
34   </thing-type>
35 </thing:thing-descriptions>
```

Listing 2: Die NutzerWelten Steckdose aus der sd.xml

5.2.2. Die Channel

Um die Wiederverwendbarkeit von Kanälen zu gewährleisten werden die Kanäle in der „channels.xml“ gespeichert. Der Aufbau der „channels.xml“ entspricht dem der XML-Datei eines Things. anstatt des <thing-type /> Tags werden hier die Kanäle definiert. Die Definition ist im folgenden für einen Schalter(Switch) gezeigt.

```
1  <channel-type id="onOff">
2    <item-type>Switch</item-type>
3    <label>Switch</label>
4  </channel-type>
```

Listing 3: Definition des Steckdosen Kanal

Im <channel-type /> Tag wird der Kanal definiert. Die „id“ ist die typeId des Kanales welche in der XML-Datei des Thing festgelegt ist(Listing 2 Zeile 18). Über den <item-type /> wird der Typ des Kanales definiert. Dies bestimmt das Item was später an den Kanal gebunden werden kann. In diesem Fall wird ein Switch definiert um die Steckdose später schalten zu können.

5.2.3. Der ThingHandler

Der ThingHandler in openHAB dient zur Steuerung des Things. Da in dem NutzerWelten System mehrere Things vorhanden sind, wird eine abstrakte Klasse definiert, welche als Vorlage verwendet werden kann. Diese kann auch zur Erweiterung des Systems mit neuen Geräte Handlern verwendet werden. Die abstrakte Klasse heißt „BaseNutzerWeltenHandler“ und erbt von der ebenfalls abstrakten Klasse „BaseThingHandler“ aus dem Package „binding“ des Bundle „org.eclipse.smarthome.core.thing“. Zudem implementiert die Klasse das Interface „INutzerWeltenColleague“ für die Kommunikation mit dem Vermittler. Diesen bekommt die Klasse von der HandlerFactory als BaseBridgeHandler übergeben. Durch eine Instanz Überprüfung und einer Typenkonvertierung wird hieraus der INutzerWeltenMediator gewonnen. Das Klassendiagramm des BaseNutzerWeltenHandler mit einer Beispielimplementierung und dem Interface ist in Abbildung 20 gezeigt.

Die einzelnen NutzerWelten Geräte ThingHandler implementieren nur die Methoden die vom BaseThingHandler aus openHAB vorgegeben werden, mit der Ausnahme der Kommunikationsmethode „updateThing( )“. Über diese Methode werden Nachrichten, die vom Vermittler kommen, ausgewertet. Die Methode „initialize( )“ dient zur Initialisierung des Gerätes. Da die Methode sowohl in der Klasse des Gerätes als auch in der Oberklasse „BaseNutzerWeltenHandler„ implementiert ist, wird in dieser Klasse die Oberklassen Methode aufgerufen und das Thing in den Status Online versetzt. Näheres

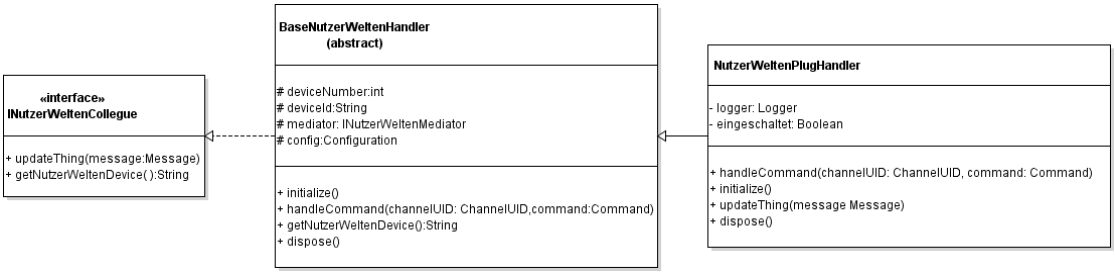

Abbildung 20: Der BaseNutzerWeltenHandler

zu dem Status wurde schon im Kapitel 2.3 auf Seite 17 beschrieben. In der Oberklassen Methode wird das Gerät an dem Vermittler angemeldet.

Die Methode „handleCommand( )“ dient zum Behandeln von Befehlen, aus openHAB. Ein Befehl wird an einen Handler abgeschickt, wenn sich der Zustand eines Kanals ändern soll, meist dann, wenn sich das zugehörige Item geändert hat. Über die „ChannelUID“ wird der zu ändernde Kanal an gegeben und über das „Command“ kann der neue Zustand des Kanals ausgelesen werden.

Die Methode „updateThing( )“ dient zum Aktualisieren des Items, wenn das reale Gerät eine Nachricht versendet hat. Sie wird über den Vermittler aufgerufen. In der übergebenen „Message“ ist die Nachricht des Gerätes gespeichert.

In der abstrakten Klasse BaseNutzerWeltenHandler wird zudem noch die Methode „getNutzerWeltenDevice( )“ überschrieben. Die Rückgabe entspricht hier der Form, des Rückgabewertes der Methode „getFullDeviceName( )“ der Message Klasse. Hier überprüft der Vermittler, an welches Gerät eine Message zugestellt werden muss. Zudem wird die Methode benötigt, um in der Brücke das Gerät, wenn es aus openHAB abgemeldet wird, aus der Gerätesuche zu entfernen, damit es später neu vom Discovery Service gefunden werden kann.

5.3. Der Discovery Service im NutzerWelten System

Wie schon erwähnt kann vom Discovery Service ein Gerät gefunden werden. Hierfür stellt der Service zwei Möglichkeiten zur Verfügung die Background Discovery, und den direkten Scan. Ist die Background Discovery aktiviert, so sucht openHAB permanent im Hintergrund nach neuen Geräten. Wird ein direkter Scan gestartet, so sucht das System nur für eine vorgegebene Zeit nach Geräten und legt nur die an, die während der Suche gefunden wurden.

NutzerWelten Geräte verfügen über keine eigene Taste zum Triggern eines Sendeimpulses zur Registrierung. Noch sind im Protokoll Befehle hinterlegt, die alle Geräte zum

Senden einer Bestätigung auffordern. Dies führt dazu, dass die NutzerWelten Geräte sehr sporadisch senden und eine Nachricht im Idealfall nur alle 15 Minuten vorkommt. Für eine Erkennung der Geräte ist damit ein zielgerichteter Scan problematisch, da er zu lange dauert. Auf die Möglichkeit, über das System einen direkten Scan zu starten, wurde damit verzichtet. Stattdessen wurde die automatische Hintergrund Erkennung eingerichtet, damit die Geräte im Fall eines Sendeimpulses direkt ins System integriert werden.

Das Klassendiagramm ist in Abbildung 21 gezeigt. Für die Entwicklung des Discovery Services wurde der Service der Philips Hue als Vorbild genommen. Der Service wird nicht mehr an der Service Registrierung von openHAB registriert, sondern am BridgeHandler. Das bedeutet, dass die Methoden „activate( )“ und „deactivate( )“ im „NutzerWeltenDiscoveryService“ überschrieben werden müssen. Erstellt wird der Discovery Service in der NutzerWeltenHandlerFactory. Für die Registrierung eines neuen Gerätes besitzt der Service die Methode „addNutzerWeltenDevice( )“. Dieser wird der String übergeben der aus der Messageklasse über die Methode „getFullDeviceName( ) (Seite 49) gewonnen wird. Dieser enthält die Gerätenummer und den Geräte Typ. Der Service besitzt damit fast alle Daten, die er zur Registrierung des Gerätes benötigt. Nur die BridgeUID entnimmt er der in ihm registrierten Brücke. Da der Service keinen manuellen Scan unterstützen soll, wurde auf die Implementierung der Methoden „startScan( )“ und „stopScan( )“verzichtet.

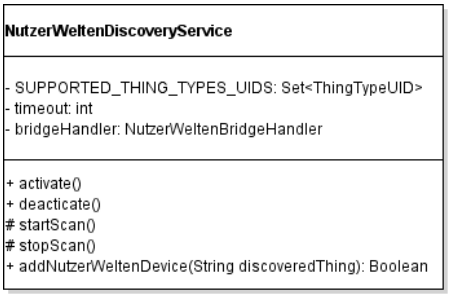

Abbildung 21: Der NutzerWeltenDiscoveryService in UML

Neben der generellen Discovery bietet openHAB auch Support für UPnP und mDNS an. Hiermit lassen sich Geräte, die diese Standards unterstützen, ohne eine aufwendige Implementierung erkennen.

5.4. Die ThingHandlerFactory im NutzerWelten System

Die ThingHandlerFactory ist die Klasse, welche die konkreten Fabrikmethoden für das Binding implementiert. Sie dient zum Erstellen der Handler. Die abstrakte Klasse, die die Methoden vorgibt, ist die BaseThingHandlerFactory. In der NutzerWeltenHandler-Factory werden nicht nur die Handler erstellt, sondern auch die Initialisierung der Things eingeleitet. Dies ist nötig, da für das BridgeThing eine etwas andere Erstellung benötigt wird. Im Gegensatz zum NutzerWelten Thing besitzt das BridgeThing hier keine Parameter, die dem Thing übergeben werden müssen. Da das Thing seinen Geräte Bezeichner und seine Nummer als Parameter übergeben bekommt, müssen die Daten aus den Parametern ausgelesen werden und dem Thing als Konfiguration übergeben werden. Da jeder Thingtyp einen eigenen Handler besitzt, muss auch die Erstellung dieser so angepasst werden, dass zum jedem Thing der passende Handler aufgemacht wird.

Das UML-Diagramm der NutzerWeltenHandlerFactory ist in Abbildung 22 aufgezeigt. Die Methode „supportedThingType( )“ gibt zurück, ob das Bundle ein Thing unterstützt. Mit „createThing( )“, wird die Initialisierung eines Things eingeleitet. Sie wird aus der Oberklasse „BaseThingHandlerFactory“ überschrieben. Die Oberklassenmethode Ruft die Thingfactory aus dem Bundle „.org.eclipse.smarthome.core.thing“ auf und leitet damit die Erstellung ein. Die erbende Methode legt die Parameter fest und ruft mit diesen die Methode aus der Oberklassen auf, damit das Thing erstellt wird. Über die Methode „createHandler( )“ wird der ThingHandler erzeugt. Hierbei aus dem übergebenen Thing der Typ ausgelesen und der passende Handler erstellt. Zuletzt dienen die Methoden „registerDiscoveryService( )“ zum Erstellen und Registrieren eines Discovery-Service am BridgeHandler, sowie „removeHandler( )“ zum Deregistrieren des Discovery Service, wenn die Bridge beendet wird.

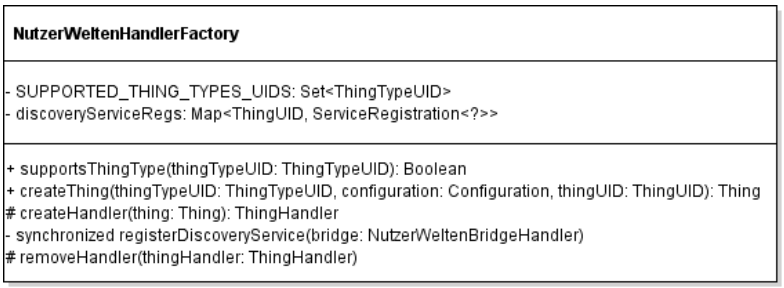

Abbildung 22: Der NutzerWeltenHandlerfactory in UML

6. Fazit

Mit dieser Arbeit wurde das NutzerWelten System in das System von openHAB integriert. Die Geräte des Systems sind über das erstellte Binding in openHAB nutzbar und arbeiten mit den Geräten anderer Plattformen zusammen. Dies bedeutet, dass die Geräte nun von dem besseren Regelsystem des open Home Automation Bus profitieren. Im Nuterwelten System waren nur fest in Software vorgegebene Regeln nutzbar. Das openHAB Regelsystem ist frei konfigurierbar und bietet auch die Möglichkeit komplexe Regeln zu erstellen und dynamisch zu verschalten.

Zudem lässt sich das System mit openHAB nun nicht nur über verschiedene Rechner steuern, sondern, mit dem Dienst my.openHAB, auch per Fernzugriff aus dem Internet. Auch ist das System nun nicht mehr an einen Prozessor oder ein Betriebssystem gebunden.

Für die Portierung wurde ein Konzept entwickelt und umgesetzt. Die Geräte der NutzerWelten Plattform werden an das Binding über einen Communicator angebunden, der die Nachrichten der Geräte aufnimmt, beziehungsweise an die Geräte aussendet. Über diesen arbeitet das BridgeThing als Vermittler der Nachrichten. Es stellt die Nachrichten im System zu und verringert, da es der zentrale Kommunikationsknoten ist, die Abhängigkeiten der ThingHandler. Der Vorteil des Konzeptes ist die Dynamik, die es ermöglicht, das System einfach zu erweitern. Ein neuer ThingHandler muss sich nur beim Vermittler registrieren und erhält automatisch seine Nachrichten zugestellt. Hierdurch wird das System erweiterbar. Neue Geräte erben vom BaseNutzerWeltenHandler, der die Registrierung übernimmt, und die Implementierung neuer Handler vereinfacht. Zudem ist der Communicator austauschbar gehalten, damit wird es möglich, auch neue Funkprotokolle an das System anzubinden.

6.1. Probleme mit der Architektur

Das Konzept hinter der neuen NutzerWelten Architektur zeigt allerdings auch Schwächen. Der Communicator ist hierarchisch an den Vermittler angebunden. Dies stammt noch von der Idee, den NutzerWelten Hardware Connector in das System von openHAB zu implementieren. Im Nachhinein wäre es besser gewesen, die Schichtung nicht mit zu übernehmen und stattdessen den Communicator, wie die ThingHandler als Kollegen an den Vermittler anzumelden. Dies hätte die interne Struktur weiter vereinfacht, da die Brücke dann keine Daten vom Communicator mehr pollt, sondern dieser seine Nachrichten an den Vermittler leitet. Ein Problem hierbei wäre dann, dass die Messages zielgerichtet versendet werden müssen, da der Vermittler sonst nicht weiß, ob die Nach-

richt an ein Gerät oder an einen Handler geleitet werden muss. Weiter gedacht wäre es dann auch sinnvoll, den Vermittler aus der Brücke zu entfernen und ihn, als eigenständige Klasse im System die Kommunikation verwalten zu lassen. Die Brücke würde dann die Aufgabe des Communicator vollständig übernehmen und wäre damit wesentlich besser austauschbar. Dies würde die Entwicklung weiter vereinfachen und wesentlich flexibler machen.

6.2. Ausblick

Mit dem neuen System wurde nur die Basis an Geräten implementiert, die auch in den Demonstratoren zum Einsatz kam. Im Labor sind zudem weitere Geräte wie der Fensteröffner, die Türkamera mit Funköffner oder die Heizungssteuerung vorhanden, für die kein eigener Handler existiert. Diese können in weiteren Arbeiten noch in das neue System implementiert werden. Auf der Kommunikationsseite wäre es denkbar, das System um einen anderen Funkstandard wie Zigbee zu erweitern. Hierfür müsste ein Brücken Gerät entwickelt werden und ein neuer Communicator ins System integriert werden.

A. Quellcode

A.1. Konstanten

Listing 4: NutzerWeltenBindingConstants.java

```java
/**
 * Copyright (c) 2014-2015 openHAB UG (haftungsbeschraenkt) and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package org.openhab.binding.nutzerwelten;

import java.util.Collections;
import java.util.Set;

import org.eclipse.smarthome.core.thing.ThingTypeUID;

import com.google.common.collect.Sets;

/**
 * The {@link NutzerWeltenBinding} class defines common constants, which are
 * used across the whole binding.
 *
 * @author w_gerlach - Initial contribution
 */
public class NutzerWeltenBindingConstants {

    public static final String BINDING_ID = "nutzerwelten";

    // List of all Thing Type UIDs
    public final static ThingTypeUID THING_TYPE_BRIDGE =
            new ThingTypeUID(BINDING_ID, "nutzerweltenbridge");
    public final static ThingTypeUID THING_TYPE_PLUG =
            new ThingTypeUID(BINDING_ID, "sd");
    public final static ThingTypeUID THING_TYPE_WATER_DETECTOR =
            new ThingTypeUID(BINDING_ID, "wm");
    public final static ThingTypeUID THING_TYPE_WINDOW_CONTACT =
            new ThingTypeUID(BINDING_ID, "fd");
    public final static ThingTypeUID THING_TYPE_MOVEMENT_DETECTOR =
            new ThingTypeUID(BINDING_ID, "bm");
    public final static ThingTypeUID THING_TYPE_FALL_DETECTOR =
            new ThingTypeUID(BINDING_ID, "nk");
    public final static ThingTypeUID THING_TYPE_COOKER_WATCHER =
            new ThingTypeUID(BINDING_ID, "hu");

    // List of all Channel IDs
    public final static String CHANNEL_ONOFF = "onOff";
    public final static String CHANNEL_WATER = "water";
```

```
46      public final static String CHANNEL_BATTERY = "battery";
47      public final static String CHANNEL_WINDOW_CONTACT = "windowContact";
48      public final static String CHANNEL_MOVEMENT_DETECTOR = "movementDetector";
49      public final static String CHANNEL_FALL_ALERT = "fallAlert";
50      public final static String CHANNEL_ALIVE = "alive";
51      public final static String CHANNEL_BUTTON_ALERT = "buttonAlert";
52      public final static String CHANNEL_ALERT_RESET = "alertReset";
53      public final static String CHANNEL_FALL_ALERT_STRING = "fallAlertString";
54      public final static String CHANNEL_ALERT_IMPULS = "alertImpuls";
55      public final static String CHANNEL_COOKER_STATE = "cookerState";
56      public final static String CHANNEL_COOKER_STATE_STRING =
57              "cookerStateString";
58
59      // device config properties for all devices
60      public final static String CONFIG_DEVICE_NUMBER = "deviceNumber";
61      public final static String CONFIG_DEVICE_TAG = "deviceTag";
62
63      // Bridge config properties
64      public final static String CONFIG_BRIDGE_NETWORKINTERFACE =
65              "networkinterface";
66
67      // List of Bridges
68      public final static Set<ThingTypeUID> SUPPORTED_BRIDGE_TYPES =
69              Collections.singleton(THING_TYPE_BRIDGE);
70
71      // List of UDP/6LoWPAN Things Classic NutzerWelten Things
72      public final static Set<ThingTypeUID> SUPPORTED_UDP_THING_TYPES =
73              Sets.newHashSet(THING_TYPE_PLUG, THING_TYPE_WATER_DETECTOR,
74              THING_TYPE_WINDOW_CONTACT, THING_TYPE_MOVEMENT_DETECTOR,
75              THING_TYPE_FALL_DETECTOR, THING_TYPE_COOKER_WATCHER);
76
77  }
```

A.2. Mediator und Colleague Interfaces

Listing 5: INutzerWeltenColleague.java

```
1   package org.openhab.binding.nutzerwelten.handler;
2
3   import org.openhab.binding.nutzerwelten.tools.Message;
4
5   /**
6    * The interface for the colleague classes in the NutzerWelten System
7    *
8    * @author w_gerlach
9    */
10  public interface INutzerWeltenColleague {
11
12      /**
13       * Send a Message to the Thing Handler
14       *
```

```
15      * @param message
16      */
17     public void updateThing(Message message);
18
19     /**
20      * returns the NutzerWelten device Name of the Thing for the Message
21      * distribution
22      *
23      * @return
24      */
25     public String getNutzerWeltenDeviceName();
26
27 }
```

Listing 6: INutzerWeltenMediator.java

```
1  package org.openhab.binding.nutzerwelten.handler;
2
3  import org.openhab.binding.nutzerwelten.tools.Message;
4
5  /**
6   * Interface for the Mediator Class in the NutzerWelten System
7   *
8   * @author w_Gerlach
9   *
10  */
11 public interface INutzerWeltenMediator {
12
13     /**
14      * Send a Message to the physical device
15      *
16      * @param message to be send
17      */
18     public void sendMessage(Message message);
19
20     /**
21      * Registers Nutzerwelten Things for the Message distribution inside the
22      * NutzerWelten Bundle
23      *
24      * @param thing Interface for a generic NutzerWeltenThing
25      * @return true if no error occurs
26      */
27     public boolean registerListeningThing(INutzerWeltenColleague thing);
28
29     /**
30      * Unregisters a NutzerWelten Thing from the internal Message distribution
31      *
32      * @param thing
33      * @return true if no error occurs
34      */
35     public boolean unregisterListeningThing(INutzerWeltenColleague thing);
36
```

```
37 }
```

A.3. Handler Klassen

Listing 7: BaseNutzerWeltenHandler.java

```
1  /**
2   * Copyright (c) 2014-2015 openHAB UG (haftungsbeschraenkt) and others.
3   * All rights reserved. This program and the accompanying materials
4   * are made available under the terms of the Eclipse Public License v1.0
5   * which accompanies this distribution, and is available at
6   * http://www.eclipse.org/legal/epl-v10.html
7   */
8  package org.openhab.binding.nutzerwelten.handler;
9
10 import static org.openhab.binding.nutzerwelten.NutzerWeltenBindingConstants.*;
11
12 import org.eclipse.smarthome.config.core.Configuration;
13 import org.eclipse.smarthome.core.thing.Bridge;
14 import org.eclipse.smarthome.core.thing.ChannelUID;
15 import org.eclipse.smarthome.core.thing.Thing;
16 import org.eclipse.smarthome.core.thing.ThingStatus;
17 import org.eclipse.smarthome.core.thing.binding.BaseThingHandler;
18 import org.eclipse.smarthome.core.thing.binding.ThingHandler;
19 import org.eclipse.smarthome.core.types.Command;
20
21 /**
22  * The {@link BaseNutzerWeltenHandler} is an abstract class that is inherited
23  * by the NutzerWelten Device Handlers. The class is manages the
24  * communication between the Bridge and the Handlers, while the Handlers
25  * managing the Channels and Items.
26  *
27  * @author w_gerlach - Initial contribution
28  */
29 public abstract class BaseNutzerWeltenHandler extends BaseThingHandler
30                                         implements INutzerWeltenColleague {
31
32     protected int deviceNumber;
33     protected String deviceId;
34     protected INutzerWeltenMediator nutzerWeltenbridge;
35     protected Configuration config = null;
36
37     public BaseNutzerWeltenHandler(Thing thing) {
38         super(thing);
39     }
40
41     @Override
42     public void initialize()
43                 throws NullPointerException, IllegalArgumentException {
44         config = getThing().getConfiguration();
45         deviceNumber = Integer.parseInt(
```

```
46                  (String) config.get(CONFIG_DEVICE_NUMBER), 16);
47          deviceId = (String) config.get(CONFIG_DEVICE_TAG);
48
49          nutzerWeltenbridge = getBridgeHandler();
50          nutzerWeltenbridge.registerListeningThing(this);
51          updateStatus(ThingStatus.ONLINE);
52      }
53
54      private INutzerWeltenMediator getBridgeHandler()
55                  throws NullPointerException, IllegalArgumentException {
56          if (nutzerWeltenbridge == null) {
57              Bridge bridge = getBridge();
58              if (bridge == null) {
59                  throw new NullPointerException(
60                          "NutzerWelten Bridge not found please create"+
61                          " a Bridge first");
62              } else {
63                  waitforBridgeInitialisation(bridge);
64                  nutzerWeltenbridge =
65                          getHandlerfromBridge(nutzerWeltenbridge, bridge);
66              }
67          }
68          return nutzerWeltenbridge;
69      }
70
71      private INutzerWeltenMediator getHandlerfromBridge
72                      (INutzerWeltenMediator bridgeHandler, Bridge bridge)
73                      throws IllegalArgumentException {
74          ThingHandler handler = bridge.getHandler();
75          if (handler instanceof INutzerWeltenMediator) {
76              bridgeHandler = (INutzerWeltenMediator) handler;
77          } else {
78              throw new IllegalArgumentException("bridge is not a "+
79                      "NutzerWelten bridge");
80          }
81          return bridgeHandler;
82      }
83
84      private void waitforBridgeInitialisation(Bridge bridge) {
85          while (bridge.getStatus() != ThingStatus.ONLINE) {
86              // give the system time to start the Bridge Handler on restart
87              // otherwise concurrency problems may occur
88              try {
89                  Thread.sleep(1000);
90              } catch (InterruptedException e) {
91
92              }
93          }
94      }
95
96      @Override
```

```
97      public abstract void handleCommand
98                              (ChannelUID channelUID, Command command);
99
100     @Override
101     public String getNutzerWeltenDeviceName() {
102         return deviceId + "+" + String.format("%04x", deviceNumber);
103     }
104
105     @Override
106     public void dispose() {
107         nutzerWeltenbridge.unregisterListeningThing(this);
108     }
109 }
```

Listing 8: NutzerWeltenBridgeHandler.java

```
1  package org.openhab.binding.nutzerwelten.handler;
2
3  import static org.openhab.binding.nutzerwelten
4          .NutzerWeltenBindingConstants.CONFIG_BRIDGE_NETWORKINTERFACE;
5
6  import java.io.IOException;
7  import java.net.SocketException;
8  import java.net.UnknownHostException;
9  import java.util.HashSet;
10 import java.util.LinkedList;
11 import java.util.List;
12 import java.util.Set;
13 import java.util.concurrent.ScheduledFuture;
14 import java.util.concurrent.TimeUnit;
15
16 import org.eclipse.smarthome.config.core.Configuration;
17 import org.eclipse.smarthome.core.thing.Bridge;
18 import org.eclipse.smarthome.core.thing.ChannelUID;
19 import org.eclipse.smarthome.core.thing.ThingStatus;
20 import org.eclipse.smarthome.core.thing.binding.BaseBridgeHandler;
21 import org.eclipse.smarthome.core.types.Command;
22 import org.openhab.binding.nutzerwelten.communicators.ICommunicator;
23 import org.openhab.binding.nutzerwelten.communicators.UDPCommunicator;
24 import org.openhab.binding.nutzerwelten.internal.discovery
25         .NutzerWeltenDiscoveryService;
26 import org.openhab.binding.nutzerwelten.tools.Message;
27 import org.slf4j.Logger;
28 import org.slf4j.LoggerFactory;
29
30 /**
31  * The BridgeHandler in the NutzerWelten system is only a virtual Handler
32  * he distributes the Messages as Mediator to the ThingHandlers and the
33  * Communicator
34  *
35  * @author w_Gerlach
36  *
```

```
37  */
38 public class NutzerWeltenBridgeHandler extends BaseBridgeHandler
39                                      implements INutzerWeltenMediator {
40
41     private Logger logger =
42             LoggerFactory.getLogger(NutzerWeltenBridgeHandler.class);
43
44     private ICommunicator communicator = null;
45     private Thread messageListenerThread;
46     ScheduledFuture<?> getDataScheduledFuture;
47     private Set<INutzerWeltenColleague> listeningNutzerWeltenHandlers;
48     private List<String> discoveredThings;
49
50     private NutzerWeltenDiscoveryService nutzerWeltenDiscoveryService;
51     private Configuration config = null;
52
53     public NutzerWeltenBridgeHandler(Bridge bridge) {
54         super(bridge);
55         discoveredThings = new LinkedList<String>();
56
57     }
58
59     @Override
60     public void handleCommand(ChannelUID channelUID, Command command) {
61         // Nothing to do
62
63     }
64
65     @Override
66     public void initialize() {
67
68         logger.info("initalising Nutzerwelten Bridge Handler");
69         String netconf = getNetworkConfig();
70         boolean initializationFaild = false;
71
72         initializationFaild = initCommunicatorWithConfAndReportError
73                 (netconf, initializationFaild);
74         listeningNutzerWeltenHandlers =
75                 new HashSet<INutzerWeltenColleague>();
76         getDataScheduledFuture = startDataScheduler();
77         if (initializationFaild) {
78             updateStatus(ThingStatus.UNINITIALIZED);
79         } else {
80             updateStatus(ThingStatus.ONLINE);
81         }
82
83     }
84
85     private String getNetworkConfig() {
86         config = getThing().getConfiguration();
87         String netconf = (String) config.get(CONFIG_BRIDGE_NETWORKINTERFACE);
```

```
88          return netconf;
89      }
90
91      private boolean initCommunicatorWithConfAndReportError(
92                              String netconf, boolean initializationFaild) {
93          try {
94              communicator = new UDPCommunicator(netconf);
95          } catch (SocketException e) {
96              logger.info("could not get NetworkInterfaces " + e.getMessage());
97              initializationFaild = true;
98          } catch (UnknownHostException e) {
99              logger.info("could not create Multicast Group", e.getMessage());
100             initializationFaild = true;
101         } catch (IOException e) {
102             logger.info("could not Initialize Socket", e.getMessage());
103             initializationFaild = true;
104         }
105         return initializationFaild;
106     }
107
108     /**
109      * Method to start the scheduling for more Data
110      *
111      */
112     private ScheduledFuture<?> startDataScheduler() {
113         return scheduler.scheduleWithFixedDelay
114                 (messageListener(), 100, 100, TimeUnit.MILLISECONDS);
115     }
116
117     /**
118      * Listener Thread that distributes the Messages from the communicator
119      * to the devices
120      *
121      * @return Listener Thread
122      */
123     private Thread messageListener() {
124         return new Thread(new Runnable() {
125             @Override
126             public void run() {
127                 if (communicator != null) {
128                     while (communicator.hasMoreMessages()) {
129                         Message message = communicator.receiveMessage();
130                         logger.info(message.getMessage());
131                         String deviceName = message.getFullDeviceName();
132                         logger.debug(deviceName);
133                         if (discoveredThings.contains(deviceName)) {
134                         } else {
135                             addUnknownDevice(deviceName);
136                         }
137                         distributeMessageToDevice(message, deviceName);
138                     }
```

```
139                 } else {
140                     logger.info("Communicator dosen't seem to run");
141                 }
142             }
143 
144         });
145     }
146 
147     private void distributeMessageToDevice(
148                                     Message message, String deviceName) {
149         for (INutzerWeltenColleague nwDevice :
150                 listeningNutzerWeltenHandlers) {
151             if (deviceName.equals(nwDevice.getNutzerWeltenDeviceName())) {
152                 nwDevice.updateThing(message);
153             }
154         }
155     }
156 
157     private void addUnknownDevice(String deviceName) {
158         logger.info("undiscovered Device found " + deviceName);
159         discoveredThings.add(deviceName);
160         nutzerWeltenDiscoveryService.addNutzerWeltenDevice(deviceName);
161     }
162 
163     @Override
164     public boolean registerListeningThing(INutzerWeltenColleague thing) {
165         if (listeningNutzerWeltenHandlers.add(thing)) {
166             logger.info("added " + thing.getNutzerWeltenDeviceName() +
167                     " to listening devices");
168             return true;
169         } else {
170             logger.info("device " + thing.getNutzerWeltenDeviceName() +
171                     " already exists in listening devices");
172             return false;
173         }
174     }
175 
176     @Override
177     public boolean unregisterListeningThing(INutzerWeltenColleague thing) {
178         boolean nofail = false;
179         if (listeningNutzerWeltenHandlers.remove(thing)) {
180             logger.info("device removed " +
181                     thing.getNutzerWeltenDeviceName() +
182                     " removed from ListeningHandlers");
183             nofail = true;
184         } else {
185             logger.info("device " + thing.getNutzerWeltenDeviceName() +
186                     " does not exist in ListeningHandlers");
187         }
188         discoveredThings.remove(thing.getNutzerWeltenDeviceName());
189         return nofail;
```

```
190
191     }
192
193     /**
194      * sends a Message to the Communicator
195      *
196      * @param message
197      */
198     @Override
199     public synchronized void sendMessage(Message message) {
200         communicator.sendMessage(message);
201     }
202
203     @Override
204     public void dispose() {
205         getDataScheduledFuture.cancel(true);
206         logger.debug("ListenerThread is interrupted");
207         messageListenerThread = null;
208         communicator.stopCommunicator();
209     }
210
211     /**
212      * Registers the DiscoveryService of the Device to the Bridge
213      *
214      * @param nutzerWeltenDiscoveryService
215      */
216     public void registerDiscoveryService(
217             NutzerWeltenDiscoveryService nutzerWeltenDiscoveryService) {
218         this.nutzerWeltenDiscoveryService = nutzerWeltenDiscoveryService;
219     }
220
221     /**
222      * Unregisters the Discovery Service for the Device
223      *
224      * @param nutzerWeltenDiscoveryService
225      */
226     public void unregisterDiscoveryService(
227             NutzerWeltenDiscoveryService nutzerWeltenDiscoveryService) {
228     }
229
230 }
```

Listing 9: NutzerWeltenCookerWatcherHandler.java

```
1 package org.openhab.binding.nutzerwelten.handler;
2
3 import static org.openhab.binding.nutzerwelten.
4         NutzerWeltenBindingConstants.*;
5
6 import org.eclipse.smarthome.core.library.types.DecimalType;
7 import org.eclipse.smarthome.core.library.types.StringType;
8 import org.eclipse.smarthome.core.thing.ChannelUID;
```

```
9  import org.eclipse.smarthome.core.thing.Thing;
10 import org.eclipse.smarthome.core.thing.ThingStatus;
11 import org.eclipse.smarthome.core.types.Command;
12 import org.openhab.binding.nutzerwelten.tools.Message;
13 import org.slf4j.Logger;
14 import org.slf4j.LoggerFactory;
15
16 /**
17  * Handler for the NutzerWelten cookerwatcher
18  *
19  * @author w_Gerlach
20  *
21  */
22 public class NutzerWeltenCookerWatcherHandler
23                                         extends BaseNutzerWeltenHandler {
24     private static final int TOOHOT_STATE = 2;
25     private static final int ACTIVE_STATE = 1;
26     private static final int OFF_STATE = 0;
27
28     Logger logger = LoggerFactory.getLogger(
29             NutzerWeltenCookerWatcherHandler.class);
30
31     private static final String COOKER_TO_HOT_MSG = "The cooker is to hot";
32     private static final String COOKER_ACTIVE_MSG = "The cooker is cooking";
33     private static final String COOKER_OFF_MSG = "The cooker is off";
34
35     private static final String COOKER_TO_HOT =
36             "maximale herd temperatur Ã¼berschritten";
37     private static final String COOKER_ACTIVE = "herd eingeschaltet";
38     private static final String COOKER_OFF = "herd inaktiv";
39
40     public NutzerWeltenCookerWatcherHandler(Thing thing) {
41         super(thing);
42     }
43
44     @Override
45     public void initialize() {
46         try {
47             super.initialize();
48             Message message =
49                     new Message(deviceId, deviceNumber, "detected");
50             nutzerWeltenbridge.sendMessage(message);
51             updateStatus(ThingStatus.ONLINE);
52         } catch (NullPointerException e) {
53             logger.info(e.getMessage());
54             updateStatus(ThingStatus.UNINITIALIZED);
55         } catch (IllegalArgumentException e) {
56             logger.info(e.getMessage());
57             updateStatus(ThingStatus.UNINITIALIZED);
58         }
59     }
```

```
60
61      @Override
62      public void updateThing(Message message) {
63          String data = message.getData();
64          if (data.equals(COOKER_OFF)) {
65              repotIsOff();
66          }
67          if (data.equals(COOKER_ACTIVE)) {
68              reportIsActive();
69          }
70          if (data.equals(COOKER_TO_HOT)) {
71              reportTooHot();
72          }
73      }
74
75      private void repotIsOff() {
76          logger.info(COOKER_OFF_MSG);
77          updateState(new ChannelUID(getThing().getUID(),
78                  CHANNEL_COOKER_STATE), new DecimalType(OFF_STATE));
79          updateState(new ChannelUID(getThing().getUID(),
80                  CHANNEL_COOKER_STATE_STRING), new StringType(
81                  COOKER_OFF_MSG));
82      }
83
84      private void reportIsActive() {
85          logger.info(COOKER_ACTIVE_MSG);
86          updateState(new ChannelUID(getThing().getUID(),
87                  CHANNEL_COOKER_STATE), new DecimalType(ACTIVE_STATE));
88          updateState(new ChannelUID(getThing().getUID(),
89                  CHANNEL_COOKER_STATE_STRING), new StringType(
90                  COOKER_ACTIVE_MSG));
91      }
92
93      private void reportTooHot() {
94          updateState(new ChannelUID(getThing().getUID(),
95                  CHANNEL_COOKER_STATE), new DecimalType(TOOHOT_STATE));
96          updateState(new ChannelUID(getThing().getUID(),
97                  CHANNEL_COOKER_STATE_STRING), new StringType(
98                  COOKER_TO_HOT_MSG));
99      }
100
101     @Override
102     public void handleCommand(ChannelUID channelUID, Command command) {
103
104     }
105
106     @Override
107     public void dispose() {
108         super.dispose();
109     }
110
```

```
111 }
```

Listing 10: NutzerWeltenFallDetectorHandler.java

```
1  package org.openhab.binding.nutzerwelten.handler;
2
3  import static org.openhab.binding.nutzerwelten.
4          NutzerWeltenBindingConstants.*;
5
6  import java.util.concurrent.ScheduledFuture;
7  import java.util.concurrent.TimeUnit;
8
9  import org.eclipse.smarthome.core.library.types.DecimalType;
10 import org.eclipse.smarthome.core.library.types.OnOffType;
11 import org.eclipse.smarthome.core.library.types.OpenClosedType;
12 import org.eclipse.smarthome.core.library.types.StringType;
13 import org.eclipse.smarthome.core.thing.ChannelUID;
14 import org.eclipse.smarthome.core.thing.Thing;
15 import org.eclipse.smarthome.core.thing.ThingStatus;
16 import org.eclipse.smarthome.core.types.Command;
17 import org.openhab.binding.nutzerwelten.tools.Message;
18 import org.slf4j.Logger;
19 import org.slf4j.LoggerFactory;
20
21 /**
22  * Handler for the NutzerWelten fall detector
23  *
24  * @author w_Gerlach
25  *
26  */
27 public class NutzerWeltenFallDetectorHandler
28                                         extends BaseNutzerWeltenHandler {
29
30     private static final int FALL_STATE = 2;
31     private static final int WARN_STATE = 1;
32     private static final int OK_STATE = 0;
33
34     private static final String FALL_WARNIG = "sturzwarnung";
35     private static final String FALL_ALERT = "sturzalarm";
36     private static final String BUTTON_ALERT = "buttonalarm";
37
38     private static final String FALL_WARNIG_MSG = "fall detected warning";
39     private static final String FALL_ALERT_MSG = "fall detected alert";
40     private static final String BUTTON_ALERT_MSG =
41             "emergency button pressed";
42     private static final String EVERYTHING_OK_MSG = "everything is alright";
43     private static final String ALIVE = "alive";
44
45     Logger logger =
46             LoggerFactory.getLogger(NutzerWeltenFallDetectorHandler.class);
47     boolean init = false;
48
```

```
49      ScheduledFuture<?> aliveScheduledFuture, alertScheduledFuture;
50
51      public NutzerWeltenFallDetectorHandler(Thing thing) {
52          super(thing);
53      }
54
55      @Override
56      public void initialize() {
57          try {
58              super.initialize();
59              Message message = new Message(deviceId,
60                      deviceNumber, "detected");
61              nutzerWeltenbridge.sendMessage(message);
62              updateStatus(ThingStatus.ONLINE);
63          } catch (NullPointerException e) {
64              logger.info(e.getMessage());
65              updateStatus(ThingStatus.UNINITIALIZED);
66          } catch (IllegalArgumentException e) {
67              logger.info(e.getMessage());
68              updateStatus(ThingStatus.UNINITIALIZED);
69          }
70          startResetAliveScheduler();
71      }
72
73      @Override
74      public void updateThing(Message message) {
75          String data = message.getData();
76          if (data.equals(FALL_ALERT)) {
77              reportFall();
78          }
79          if (data.equals(FALL_WARNIG)) {
80              reportWarn();
81          }
82          if (data.equals(BUTTON_ALERT)) {
83              reportButton();
84          }
85          if (data.equals(ALIVE)) {// Only a reminder that the Alive impuls
86                                   // is read by the System every other
87                                   // message does count as well
88          }
89          startResetAliveScheduler();
90          updateState(new ChannelUID(getThing().getUID(), CHANNEL_ALIVE),
91                  OpenClosedType.OPEN);
92      }
93
94      private void reportFall() {
95          logger.info(FALL_ALERT_MSG);
96          updateState(new ChannelUID(getThing().getUID(), CHANNEL_FALL_ALERT),
97                  new DecimalType(FALL_STATE));
98          updateState(new ChannelUID(getThing().getUID(),
99                  CHANNEL_FALL_ALERT_STRING), new StringType(FALL_ALERT_MSG));
```

```
100         updateState(new ChannelUID(getThing().getUID(),
101                 CHANNEL_ALERT_IMPULS), OpenClosedType.CLOSED);
102         startResetAlertScheduler();
103     }
104
105     private void reportWarn() {
106         logger.info(FALL_WARNIG_MSG);
107         updateState(new ChannelUID(getThing().getUID(), CHANNEL_FALL_ALERT),
108                 new DecimalType(WARN_STATE));
109         updateState(new ChannelUID(getThing().getUID(),
110                 CHANNEL_FALL_ALERT_STRING), new StringType(FALL_WARNIG_MSG));
111         updateState(new ChannelUID(getThing().getUID(),
112                 CHANNEL_ALERT_IMPULS), OpenClosedType.CLOSED);
113         startResetAlertScheduler();
114     }
115
116     private void reportButton() {
117         logger.info(BUTTON_ALERT_MSG);
118         updateState(new ChannelUID(getThing().getUID(),
119                 CHANNEL_ALERT_IMPULS), OpenClosedType.CLOSED);
120         updateState(new ChannelUID(getThing().getUID(),
121                 CHANNEL_BUTTON_ALERT), OpenClosedType.CLOSED);
122         startResetAlertScheduler();
123     }
124
125     @Override
126     public void handleCommand(ChannelUID channelUID, Command command) {
127         if (channelUID.getId().equals(CHANNEL_ALERT_RESET)) {
128             if (command instanceof OnOffType) {
129                 boolean resetInfo = OnOffType.ON.equals(command);
130                 if (resetInfo) {
131                     reportAlertReset();
132                 }
133             }
134         }
135     }
136
137     private void reportAlertReset() {
138         logger.info("The user has abrot the alert");
139         updateState(new ChannelUID(getThing().getUID(), CHANNEL_FALL_ALERT),
140                 new DecimalType(OK_STATE));
141         updateState(new ChannelUID(getThing().getUID(),
142                 CHANNEL_FALL_ALERT_STRING), new StringType(
143                 EVERYTHING_OK_MSG));
144         updateState(new ChannelUID(getThing().getUID(),
145                 CHANNEL_BUTTON_ALERT), OpenClosedType.OPEN);
146         updateState(new ChannelUID(getThing().getUID(),
147                 CHANNEL_ALERT_RESET), OnOffType.OFF);
148     }
149
150     private boolean startResetAlertScheduler() {
```

```
151         boolean ret = false;
152         if (alertScheduledFuture != null) {
153             ret = alertScheduledFuture.cancel(true);
154         }
155         alertScheduledFuture =
156                 scheduler.schedule(alertTask(), 1, TimeUnit.SECONDS);
157         return ret;
158     }
159 
160     private Runnable alertTask() {
161         return new Runnable() {
162             @Override
163             public void run() {
164                 updateState(new ChannelUID(getThing().getUID(),
165                         CHANNEL_ALERT_IMPULS), OpenClosedType.OPEN);
166             }
167         };
168     }
169 
170     private boolean startResetAliveScheduler() {
171         boolean ret = false;
172         if (aliveScheduledFuture != null) {
173             ret = aliveScheduledFuture.cancel(true);
174             logger.debug("Alive Scheduler reset");
175         } else {
176             logger.debug("Alive Scheduler initialised");
177         }
178         aliveScheduledFuture =
179                 scheduler.schedule(aliveTask(), 35, TimeUnit.MINUTES);
180         return ret;
181 
182     }
183 
184     private Runnable aliveTask() {
185         return new Runnable() {
186 
187             @Override
188             public void run() {
189                 updateState(new ChannelUID(getThing().getUID(),
190                         CHANNEL_ALIVE), OpenClosedType.CLOSED);
191             }
192         };
193     }
194 
195     @Override
196     public void dispose() {
197         super.dispose();
198         aliveScheduledFuture.cancel(true);
199         alertScheduledFuture.cancel(true);
200     }
201 }
```

Listing 11: NutzerWeltenMovementDetectorHandler.java

```
1  package org.openhab.binding.nutzerwelten.handler;
2
3  import static org.openhab.binding.nutzerwelten.NutzerWeltenBindingConstants.*;
4
5  import java.util.concurrent.ScheduledFuture;
6  import java.util.concurrent.TimeUnit;
7
8  import org.eclipse.smarthome.core.library.types.OpenClosedType;
9  import org.eclipse.smarthome.core.thing.ChannelUID;
10 import org.eclipse.smarthome.core.thing.Thing;
11 import org.eclipse.smarthome.core.types.Command;
12 import org.openhab.binding.nutzerwelten.tools.Message;
13 import org.slf4j.Logger;
14 import org.slf4j.LoggerFactory;
15
16 /**
17  * Handler for the NutzerWelten movement detector
18  *
19  * @author w_Gerlach
20  *
21  */
22 public class NutzerWeltenMovementDetectorHandler
23                                         extends BaseNutzerWeltenHandler {
24     private static final String MOVEMENT = "bewegung erkannt";
25     private static final String ALIVE = "alive";
26
27     Logger logger = LoggerFactory.getLogger(
28             NutzerWeltenWindowContactHandler.class);
29     ScheduledFuture<?> aliveScheduledFuture, movementScheduledFuture;
30
31     public NutzerWeltenMovementDetectorHandler(Thing thing) {
32         super(thing);
33     }
34
35     @Override
36     public void initialize() {
37         try {
38             super.initialize();
39             Message message =
40                     new Message(deviceId, deviceNumber, "detected");
41             nutzerWeltenbridge.sendMessage(message);
42             updateStatus(ThingStatus.ONLINE);
43         } catch (NullPointerException e) {
44             logger.info(e.getMessage());
45             updateStatus(ThingStatus.UNINITIALIZED);
46         } catch (IllegalArgumentException e) {
47             logger.info(e.getMessage());
48             updateStatus(ThingStatus.UNINITIALIZED);
49         }
50
```

```
51      }
52
53      @Override
54      public void updateThing(Message message) {
55          String data = message.getData();
56
57          if (data.equals(MOVEMENT)) {
58              reportMovement();
59          }
60          if (data.equals(ALIVE)) {// Only a Reminder that the Alive impuls is
61                                   // read by the System every other message
62                                   // does count as well
63          }
64
65          startResetAliveScheduler();
66          updateState(new ChannelUID(getThing().getUID(), CHANNEL_ALIVE),
67                  OpenClosedType.OPEN);
68      }
69
70      private void reportMovement() {
71          updateState(new ChannelUID(getThing().getUID(),
72                  CHANNEL_MOVEMENT_DETECTOR), OpenClosedType.CLOSED);
73          startResetMovementScheduler();
74      }
75
76      private boolean startResetAliveScheduler() {
77          boolean ret = false;
78          if (aliveScheduledFuture != null) {
79              ret = aliveScheduledFuture.cancel(true);
80              logger.debug("Alive Scheduler reset");
81          } else {
82              logger.debug("Alive Scheduler initialised");
83          }
84          aliveScheduledFuture = scheduler.schedule(aliveTask(), 35,
85                  TimeUnit.MINUTES);
86          return ret;
87
88      }
89
90      private Runnable aliveTask() {
91          return new Runnable() {
92
93              @Override
94              public void run() {
95                  updateState(new ChannelUID(getThing().getUID(),
96                          CHANNEL_ALIVE), OpenClosedType.CLOSED);
97              }
98
99          };
100     }
101
```

```
102     private boolean startResetMovementScheduler() {
103         boolean ret = false;
104         if (movementScheduledFuture != null) {
105             ret = movementScheduledFuture.cancel(true);
106             logger.debug("Movement Scheduler Reset");
107         } else {
108             logger.debug("Movement Scheduler Initialised");
109         }
110         movementScheduledFuture = scheduler.schedule(resetMovementTask(),
111                 10, TimeUnit.SECONDS);
112         return ret;
113     }
114 
115     private Runnable resetMovementTask() {
116         return new Runnable() {
117 
118             @Override
119             public void run() {
120                 updateState(new ChannelUID(getThing().getUID(),
121                         CHANNEL_MOVEMENT_DETECTOR), OpenClosedType.OPEN);
122             }
123         };
124     }
125 
126     @Override
127     public void handleCommand(ChannelUID channelUID, Command command) {
128 
129     }
130 
131     @Override
132     public void dispose() {
133         super.dispose();
134         movementScheduledFuture.cancel(true);
135         aliveScheduledFuture.cancel(true);
136     }
137 
138 }
```

Listing 12: NutzerWeltenPlugHandler

```
1  package org.openhab.binding.nutzerwelten.handler;
2 
3  import static org.openhab.binding.nutzerwelten.NutzerWeltenBindingConstants.
4          CHANNEL_ONOFF;
5 
6  import org.eclipse.smarthome.core.library.types.OnOffType;
7  import org.eclipse.smarthome.core.thing.ChannelUID;
8  import org.eclipse.smarthome.core.thing.Thing;
9  import org.eclipse.smarthome.core.thing.ThingStatus;
10 import org.eclipse.smarthome.core.types.Command;
11 import org.openhab.binding.nutzerwelten.tools.Message;
12 import org.slf4j.Logger;
```

```
13 import org.slf4j.LoggerFactory;
14
15 /**
16  * Handler for the NutzerWelten plug
17  *
18  * @author w_Gerlach
19  *
20  */
21 public class NutzerWeltenPlugHandler extends BaseNutzerWeltenHandler {
22
23     private static final String TURN_ON = "einschalten";
24     private static final String TURN_OFF = "ausschalten";
25     private static final String PLUG_ON = "eingeschaltet";
26     private static final String PLUG_OFF = "ausgeschaltet";
27
28     private Logger logger =
29             LoggerFactory.getLogger(NutzerWeltenPlugHandler.class);
30     private boolean plugStateOn;
31
32     public NutzerWeltenPlugHandler(Thing thing) {
33         super(thing);
34     }
35
36     @Override
37     public void initialize() {
38         try {
39             super.initialize();
40             Message message =
41                     new Message(deviceId, deviceNumber, "detected");
42             nutzerWeltenbridge.sendMessage(message);
43             updateStatus(ThingStatus.ONLINE);
44         } catch (NullPointerException e) {
45             logger.info(e.getMessage());
46             updateStatus(ThingStatus.UNINITIALIZED);
47         } catch (IllegalArgumentException e) {
48             logger.info(e.getMessage());
49             updateStatus(ThingStatus.UNINITIALIZED);
50         }
51
52     }
53
54     @Override
55     public void handleCommand(ChannelUID channelUID, Command command) {
56         if (channelUID.getId().equals(CHANNEL_ONOFF)) {
57             if (command instanceof OnOffType) {
58                 boolean turnedOn = OnOffType.ON.equals(command);
59                 Message message;
60                 if (turnedOn) {
61                     message = turnOnDevice();
62                 } else {
63                     message = turnOffDevice();
```

```
64                  }
65                  nutzerWeltenbridge.sendMessage(message);
66              }
67          }
68      }
69
70      private Message turnOnDevice() {
71          Message message;
72          message = new Message(deviceId, deviceNumber, TURN_ON);
73          plugStateOn = true;
74          return message;
75      }
76
77      private Message turnOffDevice() {
78          Message message;
79          message = new Message(deviceId, deviceNumber, TURN_OFF);
80          plugStateOn = false;
81          return message;
82      }
83
84      @Override
85      public void updateThing(Message message) {
86          logger.debug("Nachricht erhalten " + plugStateOn + " : " +
87                  message.getData().equals("eingeschaltet"));
88          String data = message.getData();
89          if (changedToOn(data)) {
90              reportChangeOn();
91          }
92          if (changedToOff(data)) {
93              reportChangeOff();
94          }
95          updateStatus(ThingStatus.ONLINE);
96
97      }
98
99      private boolean changedToOn(String data) {
100         return data.equals(PLUG_ON) && !(plugStateOn);
101     }
102
103     private void reportChangeOn() {
104         updateState(new ChannelUID(getThing().getUID(), CHANNEL_ONOFF),
105                 OnOffType.ON);
106         plugStateOn = true;
107     }
108
109     private boolean changedToOff(String data) {
110         return data.equals(PLUG_OFF) && plugStateOn;
111     }
112
113     private void reportChangeOff() {
114         updateState(new ChannelUID(getThing().getUID(), CHANNEL_ONOFF),
```

```
115                 OnOffType.OFF);
116         plugStateOn = false;
117     }
118
119 }
```

Listing 13: NutzerWeltenWaterDetectorHandler.java

```
1  package org.openhab.binding.nutzerwelten.handler;
2
3  import static org.openhab.binding.nutzerwelten.NutzerWeltenBindingConstants.*;
4
5  import java.util.concurrent.ScheduledFuture;
6  import java.util.concurrent.TimeUnit;
7
8  import org.eclipse.smarthome.core.library.types.OpenClosedType;
9  import org.eclipse.smarthome.core.thing.ChannelUID;
10 import org.eclipse.smarthome.core.thing.Thing;
11 import org.eclipse.smarthome.core.thing.ThingStatus;
12 import org.eclipse.smarthome.core.types.Command;
13 import org.openhab.binding.nutzerwelten.tools.Message;
14 import org.slf4j.Logger;
15 import org.slf4j.LoggerFactory;
16
17 /**
18  * Handler for the NutzerWelten water detector
19  *
20  * @author w_Gerlach
21  *
22  */
23 public class NutzerWeltenWaterDetectorHandler
24                                         extends BaseNutzerWeltenHandler {
25
26     private static final String ALARM_OLD = "alarm_old";
27     private static final String ALARM = "alarm";
28     private static final String BAT_LOW = "bat_low";
29     private static final String ALIVE = "alive";
30
31     private Logger logger = LoggerFactory.getLogger(
32             NutzerWeltenWaterDetectorHandler.class);
33
34     ScheduledFuture<?> aliveScheduledFuture, batteryScheduledFuture;
35
36     boolean aliveReceived = true;
37     boolean batteryLow = false;
38
39     public NutzerWeltenWaterDetectorHandler(Thing thing) {
40         super(thing);
41     }
42
43     @Override
44     public void initialize() {
```

```
45          try {
46              super.initialize();
47              Message message =
48                      new Message(deviceId, deviceNumber, "detected");
49              nutzerWeltenbridge.sendMessage(message);
50              updateStatus(ThingStatus.ONLINE);
51          } catch (NullPointerException e) {
52              logger.info(e.getMessage());
53              updateStatus(ThingStatus.UNINITIALIZED);
54          } catch (IllegalArgumentException e) {
55              logger.info(e.getMessage());
56              updateStatus(ThingStatus.UNINITIALIZED);
57          }
58          startResetAliveScheduler();
59      }
60
61      @Override
62      public void handleCommand(ChannelUID channelUID, Command command) {
63
64      }
65
66      @Override
67      public void updateThing(Message message) {
68          String data = message.getData();
69          if (data.equals(ALIVE)) {
70              reportOk();
71          }
72          if (data.equals(BAT_LOW)) {
73              reportBattery();
74          }
75          if (data.equals(ALARM)) {
76              reportAlarm();
77          }
78          if (data.equals(ALARM_OLD)) {
79              reportOk();
80          }
81
82          startResetAliveScheduler();
83          updateBattery();
84      }
85
86      private void reportBattery() {
87          batteryLow = true;
88          startSetBatteryScheduler();
89      }
90
91      private void reportAlarm() {
92          updateState(new ChannelUID(getThing().getUID(), CHANNEL_WATER),
93                  OpenClosedType.CLOSED);
94          aliveReceived = true;
95      }
```

```
96
97      private void reportOk() {
98          updateState(new ChannelUID(getThing().getUID(), CHANNEL_WATER),
99                  OpenClosedType.OPEN);
100         aliveReceived = true;
101     }
102
103     // TODO gleiche Methoden mit Wassermelder vereinen und in eigene Klasse
104     private boolean startResetAliveScheduler() {
105         boolean ret = false;
106         if (aliveScheduledFuture != null) {
107             ret = aliveScheduledFuture.cancel(true);
108             logger.debug("Alive Scheduler reset");
109         } else {
110             logger.debug("Alive Scheduler initialised");
111         }
112         aliveScheduledFuture = scheduler.schedule(aliveTask(),
113                 20, TimeUnit.MINUTES);
114         return ret;
115     }
116
117     private Runnable aliveTask() {
118         return new Runnable() {
119
120             @Override
121             public void run() {
122                 aliveReceived = false;
123                 updateBattery();
124             }
125         };
126     }
127
128     private boolean startSetBatteryScheduler() {
129         boolean ret = false;
130         if (batteryScheduledFuture != null) {
131             ret = batteryScheduledFuture.cancel(true);
132             logger.debug("Battery low Scheduler Reset");
133         } else {
134             logger.debug("Battery low Scheduler initialised");
135         }
136         batteryScheduledFuture = scheduler.schedule(batteryTask(),
137                 35, TimeUnit.MINUTES);
138         return ret;
139
140     }
141
142     private Runnable batteryTask() {
143         return new Runnable() {
144
145             @Override
146             public void run() {
```

```
147                 batteryLow = false;
148                 updateBattery();
149             }
150
151         };
152     }
153
154     private void updateBattery() {
155         if (everyThingOk()) {
156             updateState(new ChannelUID(getThing().getUID(),
157                     CHANNEL_BATTERY), OpenClosedType.OPEN);
158             updateState(new ChannelUID(getThing().getUID(),
159                     CHANNEL_ALIVE), OpenClosedType.OPEN);
160         }
161         if (batteryLow()) {
162             updateState(new ChannelUID(getThing().getUID(),
163                     CHANNEL_BATTERY), OpenClosedType.CLOSED);
164             updateState(new ChannelUID(getThing().getUID(), CHANNEL_ALIVE),
165                     OpenClosedType.OPEN);
166         }
167         if (thresholdToBatteryDeath()) {
168             updateState(new ChannelUID(getThing().getUID(), CHANNEL_BATTERY),
169                     OpenClosedType.CLOSED);
170             updateState(new ChannelUID(getThing().getUID(), CHANNEL_ALIVE),
171                     OpenClosedType.CLOSED);
172         }
173         if (batteryDeadOrCommunicationError()) {
174             updateState(new ChannelUID(getThing().getUID(), CHANNEL_BATTERY),
175                     OpenClosedType.CLOSED);
176             updateState(new ChannelUID(getThing().getUID(), CHANNEL_ALIVE),
177                     OpenClosedType.CLOSED);
178         }
179     }
180
181     private boolean everyThingOk() {
182         return aliveReceived && !batteryLow;
183     }
184
185     private boolean batteryLow() {
186         return aliveReceived && batteryLow;
187     }
188
189     private boolean thresholdToBatteryDeath() {
190         return !aliveReceived && batteryLow;
191     }
192
193     private boolean batteryDeadOrCommunicationError() {
194         return !aliveReceived && !batteryLow;
195     }
196
197     @Override
```

```
198     public void dispose() {
199         super.dispose();
200         aliveScheduledFuture.cancel(true);
201         batteryScheduledFuture.cancel(true);
202     }
203 
204 }
```

Listing 14: NutzerWeltenWindowContactHandler.java

```
1  package org.openhab.binding.nutzerwelten.handler;
2  
3  import static org.openhab.binding.nutzerwelten.NutzerWeltenBindingConstants.*;
4  
5  import java.util.concurrent.ScheduledFuture;
6  import java.util.concurrent.TimeUnit;
7  
8  import org.eclipse.smarthome.core.library.types.OpenClosedType;
9  import org.eclipse.smarthome.core.thing.ChannelUID;
10 import org.eclipse.smarthome.core.thing.Thing;
11 import org.eclipse.smarthome.core.thing.ThingStatus;
12 import org.eclipse.smarthome.core.types.Command;
13 import org.openhab.binding.nutzerwelten.tools.Message;
14 import org.slf4j.Logger;
15 import org.slf4j.LoggerFactory;
16 
17 /**
18  * Handler for the NutzerWelten window contact
19  *
20  * @author w_gerlach
21  *
22  */
23 public class NutzerWeltenWindowContactHandler
24                                         extends BaseNutzerWeltenHandler {
25 
26     private static final String BAT_LOW = "bat_low";
27     private static final String OPEN = "open";
28     private static final String CLOSE = "closed";
29 
30     Logger logger = LoggerFactory.getLogger(
31             NutzerWeltenWindowContactHandler.class);
32     ScheduledFuture<?> batteryScheduledFuture, dataReceivedScheduledFuture;
33 
34     boolean batteryLow = false;
35     boolean dataReceived = true;
36 
37     public NutzerWeltenWindowContactHandler(Thing thing) {
38         super(thing);
39     }
40 
41     @Override
42     public void initialize() {
```

```
43          try {
44              super.initialize();
45              Message message =
46                      new Message(deviceId, deviceNumber, "detected");
47              nutzerWeltenbridge.sendMessage(message);
48              updateStatus(ThingStatus.ONLINE);
49          } catch (NullPointerException e) {
50              logger.info(e.getMessage());
51              updateStatus(ThingStatus.UNINITIALIZED);
52          } catch (IllegalArgumentException e) {
53              logger.info(e.getMessage());
54              updateStatus(ThingStatus.UNINITIALIZED);
55          }
56      }
57
58      @Override
59      public void updateThing(Message message) {
60          String data = message.getData();
61
62          if (data.equals(BAT_LOW)) {
63              reportBattery();
64          }
65          if (data.equals(OPEN)) {
66              reportOpen();
67          }
68          if (data.equals(CLOSE)) {
69              reportClosed();
70          }
71          updateBattery();
72      }
73
74      private void reportBattery() {
75          batteryLow = true;
76          startBatteryScheduler();
77      }
78
79      private void reportOpen() {
80          updateState(new ChannelUID(getThing().getUID(),
81                  CHANNEL_WINDOW_CONTACT), OpenClosedType.OPEN);
82          dataReceived = true;
83          startDataReceivedScheduler();
84      }
85
86      private void reportClosed() {
87          updateState(new ChannelUID(getThing().getUID(),
88                  CHANNEL_WINDOW_CONTACT), OpenClosedType.CLOSED);
89          dataReceived = true;
90          startDataReceivedScheduler();
91      }
92
93      @Override
```

```
94      public void handleCommand(ChannelUID channelUID, Command command) {
95
96      }
97
98      private boolean startBatteryScheduler() {
99          boolean ret = false;
100         if (batteryScheduledFuture != null) {
101             ret = batteryScheduledFuture.cancel(true);
102             logger.debug("Battery low Scheduler Reset");
103         } else {
104             logger.debug("Battery low Scheduler initialised");
105         }
106         batteryScheduledFuture = scheduler.schedule(batteryTask(),
107                 35, TimeUnit.MINUTES);
108         return ret;
109
110     }
111
112     /*
113      * TODO ich denke der batteryListener der dataListener sowie deren
114      * Scheduler lassen sich zusammen als Ã¼ber die gleichen Methoden
115      * setzten?? Am Besten in eigene Klasse ausgleidern
116      */
117     private Thread batteryTask() {
118         return new Thread(new Runnable() {
119             @Override
120             public void run() {
121                 batteryLow = false;
122                 updateBattery();
123             }
124         });
125     }
126
127     private boolean startDataReceivedScheduler() {
128         boolean ret = false;
129         if (dataReceivedScheduledFuture != null) {
130             ret = dataReceivedScheduledFuture.cancel(true);
131             logger.debug("data Received Scheduler Reset");
132         } else {
133             logger.debug("data Received Scheduler initialised");
134         }
135         dataReceivedScheduledFuture = scheduler.schedule(dataTask(),
136                 20, TimeUnit.MINUTES);
137         return ret;
138     }
139
140     private Runnable dataTask() {
141         return new Thread(new Runnable() {
142             @Override
143             public void run() {
144                 dataReceived = false;
```

```
145                 updateBattery();
146             }
147         });
148     }
149
150     // TODO wie die Scheduler in eine Eigene Klasse ausgliedern.
151     private void updateBattery() {
152         if (everythingOk()) {
153             updateState(new ChannelUID(getThing().getUID(), CHANNEL_BATTERY),
154                     OpenClosedType.OPEN);
155         }
156         if (batteryLow()) {
157             updateState(new ChannelUID(getThing().getUID(), CHANNEL_BATTERY),
158                     OpenClosedType.CLOSED);
159         }
160         if (thresholdToBatteryDeath()) {
161             updateState(new ChannelUID(getThing().getUID(), CHANNEL_BATTERY),
162                     OpenClosedType.CLOSED);
163         }
164         if (batteryDeadOrCommunicationError()) {
165             updateState(new ChannelUID(getThing().getUID(), CHANNEL_BATTERY),
166                     OpenClosedType.CLOSED);
167         }
168
169     }
170
171     private boolean batteryDeadOrCommunicationError() {
172         return !dataReceived && !batteryLow;
173     }
174
175     private boolean thresholdToBatteryDeath() {
176         return !dataReceived && batteryLow;
177     }
178
179     private boolean batteryLow() {
180         return dataReceived && batteryLow;
181     }
182
183     private boolean everythingOk() {
184         return dataReceived && !batteryLow;
185     }
186
187     @Override
188     public void dispose() {
189         super.dispose();
190         batteryScheduledFuture.cancel(true);
191         dataReceivedScheduledFuture.cancel(true);
192     }
193
194 }
```

A.4. Interface ICommunicator und UDPCommunicator

Listing 15: ICommunicator.java

```
1  package org.openhab.binding.nutzerwelten.communicators;
2
3  import org.openhab.binding.nutzerwelten.tools.Message;
4  /**
5   * Interface for the communication with the devices
6   *
7   * @author w_Gerlach
8   *
9   */
10 public interface ICommunicator {
11     /**
12      * Method to send a Message to a device
13      *
14      * @param message
15      */
16     public void sendMessage(Message message);
17
18     /**
19      * Method to receive a Message from the Communicator FIFO
20      *
21      * @return the message as Message Class
22      */
23     public Message receiveMessage();
24
25     /**
26      * Method to check if the FIFO is empty
27      *
28      * @return true if the Buffer contains more Messages
29      */
30     public boolean hasMoreMessages();
31
32     /**
33      * Method to Stop a Communicator if a Thread runs inside the Communicator
34      * kill it here
35      */
36     public void stopCommunicator();
37
38 }
```

Listing 16: UDPCommunicator.java

```
1  package org.openhab.binding.nutzerwelten.communicators;
2
3  import java.io.IOException;
4  import java.net.DatagramPacket;
5  import java.net.Inet6Address;
6  import java.net.InetAddress;
```

```
7  import java.net.MulticastSocket;
8  import java.net.NetworkInterface;
9  import java.net.SocketException;
10 import java.net.UnknownHostException;
11 import java.util.Enumeration;
12 import java.util.LinkedList;
13 import java.util.List;
14 import java.util.Properties;
15 import java.util.concurrent.BlockingQueue;
16 import java.util.concurrent.LinkedBlockingQueue;
17
18 import org.openhab.binding.nutzerwelten.tools.Message;
19 import org.slf4j.Logger;
20 import org.slf4j.LoggerFactory;
21 /**
22  * Class for the IPV6 over UDP Communication
23  *
24  * @author w_gerlach
25  *
26  */
27 public class UDPCommunicator implements ICommunicator {
28     private static final String OS_NAME = "os.name";
29     private static final String WINDOWS = "Windows";
30     private static final String LINUX = "Linux";
31
32     private static final String RAVEN = "RAVEN";
33     private static final String WLAN = "wlan";
34     private static final String ETH = "eth";
35     private static final String USB = "usb";
36     private static final int PORT = 4321; // Communication Port
37     private static final String MULTICAST_ADDR = "FF02::1";
38     // Multicast Link local address
39
40     // TODO Containerklasse fÃ¼r Die UDP Kommunikationsdaten
41     private InetAddress multicastGroup; // Group for receiving messages
42     private MulticastSocket multicastSocket = null;
43     private NetworkInterface networkInterface = null;
44
45     private Boolean listenerActive = false;
46     // Flag that shows that the listener Thread is running
47     private BlockingQueue<Message> messageQueue = null;
48     // Message Buffer for receiving new Data
49     private Thread socketListener = null;
50     // Listener Thread that Listen for new Messages
51
52     private Logger logger = LoggerFactory.getLogger(UDPCommunicator.class);
53
54     /**
55      * Constructor for the UDPCommunicator
56      *
57      * @throws SocketException
```

```
58      * @throws UnknownHostException
59      * @throws IOException
60      */
61     public UDPCommunicator(String networkConfig)
62                     throws SocketException, UnknownHostException, IOException {
63         networkInterface = getOrGenerateNetworkInterface(networkConfig);
64         messageQueue = new LinkedBlockingQueue<Message>(10);
65         multicastGroup = InetAddress.getByName(MULTICAST_ADDR);
66         multicastSocket = openSocketwithPortInterfaceAndGroup(
67                 PORT, networkInterface, multicastGroup);
68         socketListener = initSocketListener();
69         sendMessage(new Message(
70             "   nw", 1, "das Nutzerwelten System sagt Hallo Welt"));
71             // A Message must be send otherwise the System may ignore the first
72             // Message to be send so this message may go lost
73         logger.info("UDP Communicaton started");
74     }
75
76     private NetworkInterface getOrGenerateNetworkInterface(String networkConfig)
77                                                         throws SocketException {
78         NetworkInterface netInterface;
79         if (networkInterfaceIsValid(networkConfig)) {
80             netInterface = NetworkInterface.getByName(networkConfig);
81         } else {
82             netInterface = generateNetworkinterface();
83         }
84         return netInterface;
85     }
86
87     private boolean networkInterfaceIsValid(String networkConfig) {
88         boolean isValid = true;
89         if (networkConfig == null) {
90             logger.info("No Networkinterface chosen by User");
91             isValid = false;
92         } else {
93             try {
94                 NetworkInterface.getByName(networkConfig);
95             } catch (SocketException e) {
96                 logger.info("NetworkInterface not valid " + e.getMessage());
97                 isValid = false;
98             }
99         }
100        return isValid;
101    }
102
103    private NetworkInterface generateNetworkinterface() throws SocketException {
104        List<NetworkInterface> interfaces = getInterfacesByOSLessConditions();
105        Properties systemProperties = System.getProperties();
106        // TODO in eine Klasse mit Interface packen um mehr Linuxe zu
107        // unterscheiden
108        NetworkInterface networkInt = null;
```

```
109         if (systemProperties.getProperty(OS_NAME).contains(WINDOWS)) {
110             networkInt = getWindowsInterface(interfaces);
111         }
112         if (systemProperties.getProperty(OS_NAME).contains(LINUX)) {
113             networkInt = getRaspianInterface(interfaces);
114         }
115         logger.info(
116                 "Network Interface " + networkInt.getName() + " "
117                 + networkInt.getDisplayName() + " chosen by System");
118         return networkInt;
119     }
120 
121     private List<NetworkInterface> getInterfacesByOSLessConditions()
122                                                         throws SocketException {
123         List<NetworkInterface> chosenInterfaces =
124                 new LinkedList<NetworkInterface>();
125         Enumeration<NetworkInterface> networkInterfaces =
126                 NetworkInterface.getNetworkInterfaces();
127         while (networkInterfaces.hasMoreElements()) {
128             NetworkInterface possibleInterface = networkInterfaces.nextElement();
129             if (matchesOSLessConditions(possibleInterface)) {
130                 chosenInterfaces = addInterfaceToListIfIPv6Interface(
131                         possibleInterface, chosenInterfaces);
132             }
133         }
134         return chosenInterfaces;
135     }
136 
137     private boolean matchesOSLessConditions(NetworkInterface possibleInterface)
138                                                             throws SocketException {
139         return possibleInterface.isUp() && possibleInterface.supportsMulticast()
140                 && !possibleInterface.isLoopback()
141                 && !possibleInterface.getDisplayName().contains("VirtualBox");
142                 // VirtualBox may not seen on other Systems than Windows
143                 // (not OSless) but cannot be verified
144     }
145 
146     private List<NetworkInterface> addInterfaceToListIfIPv6Interface(
147                                     NetworkInterface possibleInterface,
148                                     List<NetworkInterface> chosenInterfaces) {
149         Enumeration<InetAddress> Addresses =
150                 possibleInterface.getInetAddresses();
151         while (Addresses.hasMoreElements()) {
152             InetAddress address = Addresses.nextElement();
153             if (address instanceof Inet6Address) {
154                 if (!chosenInterfaces.contains(possibleInterface)) {
155                     chosenInterfaces.add(possibleInterface);
156                     // if interface has more than one IPV6 Address add it only
157                     // once
158                 }
159             }
```

```
160         }
161         return chosenInterfaces;
162     }
163 
164     /**
165      * Interface Search for Windows
166      * Tested on 7
167      * Should also run on 8, 9, 10 testing pending
168      *
169      * @param interfaces
170      * @return
171      */
172     private NetworkInterface getWindowsInterface(
173                                             List<NetworkInterface> interfaces) {
174         NetworkInterface chosenInterface = null;
175         int priority = 0;
176         for (NetworkInterface possibleInterface : interfaces) {
177             if (possibleInterface.getDisplayName().contains(RAVEN)) {
178                 chosenInterface = possibleInterface;
179                 priority = 10;
180             } else if (possibleInterface.getName().contains(WLAN)
181                     && priority < 2) {
182                 chosenInterface = possibleInterface;
183                 priority = 2;
184             } else if (possibleInterface.getName().contains(ETH)
185                     && priority == 0) {
186                 chosenInterface = possibleInterface;
187             } // TODO was wenn nichts passendes drin
188         }
189         return chosenInterface;
190     }
191 
192     /**
193      * Interface Search for Linux
194      * Only successful on Raspbian
195      * Does not run on Ubuntu Systems
196      * TODO Change to exclusive Raspian and add other Linux Systems
197      *
198      * @param interfaces
199      * @return
200      */
201     private NetworkInterface getRaspianInterface(
202                                             List<NetworkInterface> interfaces) {
203         NetworkInterface chosenInterface = null;
204         int priority = 0;
205         for (NetworkInterface possobleInterface : interfaces) {
206             if (possobleInterface.getName().contains(USB)) {
207                 chosenInterface = possobleInterface;
208                 priority = 10;
209             } else if (possobleInterface.getName().contains(WLAN)
210                     && priority < 2) {
```

```
211                 chosenInterface = possobleInterface;
212                 priority = 2;
213             } else if (possobleInterface.getName().contains(ETH)
214                         && priority == 0) {
215                 chosenInterface = possobleInterface;
216             } // TODO was wenn nichts passendes drin
217         }
218         return chosenInterface;
219     }
220 
221     private MulticastSocket openSocketwithPortInterfaceAndGroup(
222                                 int port, NetworkInterface netInterface,
223                                 InetAddress group) throws IOException {
224         MulticastSocket socket = new MulticastSocket(port);
225         socket.setNetworkInterface(NetworkInterface.getByName(
226                 netInterface.getName()));
227         socket.joinGroup(group);
228         return socket;
229     }
230 
231     private Thread initSocketListener() {
232         Thread listener = getSocketListener();
233         try {
234             listener.setPriority(Thread.MAX_PRIORITY);
235         } catch (SecurityException e) {
236             logger.info("Not allowed to change Priority " + e.getMessage());
237         } // Doesn't need to throw because the Systems runs even if this
238           // exception occurs
239         listener.start();
240         return listener;
241     }
242 
243     /**
244      * Method that creates a Thread Listening for Messages
245      *
246      * @return Thread
247      */
248     private Thread getSocketListener() {
249         return new Thread(new Runnable() {
250             @Override
251             public void run() {
252                 logger.debug("Starting Listener");
253                 listenerActive = true;
254                 while (listenerActive) {
255                     byte[] receivedMessage = new byte[256];
256                     DatagramPacket packet = new DatagramPacket(
257                             receivedMessage, receivedMessage.length);
258                     try {
259                         multicastSocket.receive(packet);
260                     } catch (IOException e) {
261                         logger.info("Data receive has faild  " +
```

```
262                                 e.getMessage());
263                     }
264                     Message message = Message.createMessageFromDatagram(packet);
265                     messageQueue = offerMessageToQueueIfValid(
266                             message, messageQueue);
267                 }
268                 multicastSocket.close();
269             }
270         });
271     }
272
273     @Override
274     public void sendMessage(Message message) {
275         // TODO vielleicht besser Exception werfen?
276         try {
277             DatagramPacket packet = Message.createDatagramFromMessage(
278                     message, multicastGroup, PORT);
279             multicastSocket.send(packet);
280         } catch (IOException e) {
281             logger.info("unable to send message " + e.getMessage());
282
283         }
284
285     }
286
287     @Override
288     public Message receiveMessage() {
289         Message ret = null;
290         try {
291             ret = messageQueue.take();
292         } catch (InterruptedException e) {
293             logger.info("interrupted while waiting on Messages " +
294                     e.getMessage());
295         }
296         return ret;
297
298     }
299
300     @Override
301     public void stopCommunicator() {
302         listenerActive = false;
303         sendMessage(new Message("nw", 0, "beende Communicator"));
304         try {
305             Thread.sleep(500);
306         } catch (InterruptedException e) {
307
308             logger.info("interuption while shuting down the Communicator " +
309                     e.getMessage());
310         }
311         while (socketListener.isAlive()) {
312             ;
```

```
313         }
314         logger.debug("Listener is interrupted");
315         socketListener = null;
316         multicastGroup = null;
317         messageQueue.clear();
318     }
319
320     @Override
321     public boolean hasMoreMessages() {
322         if (messageQueue.isEmpty()) {
323             return false;
324         } else {
325             return true;
326         }
327
328     }
329
330     private BlockingQueue<Message> offerMessageToQueueIfValid(
331                                 Message message, BlockingQueue<Message> queue) {
332         boolean queueHasFreeSpaces = false;
333         // TODO if else in eigene Funktion
334         if (message != null) {
335             logger.info(message.getMessage());
336             // TODO remove me later and aktivate in Bridge
337             queueHasFreeSpaces = queue.offer(message);
338             if (queueHasFreeSpaces) {
339
340             } else {
341                 logger.info(
342                         "Queue is Full Data is lost. Too many Messages " +
343                         "occour. If this Message occures on every Message " +
344                         "received the bridgehandler is not working " +
345                         "correctly, try restart the System");
346             }
347         } else {
348             logger.debug("Received Message is not a NutzerWelten device " +
349                     "Message");
350         }
351         return queue;
352     }
353 }
```

A.5. Handler Factory und Discovery Service

Listing 17: NutzerWeltenHandlerFactory.java

```
1  /**
2   * Copyright (c) 2014 openHAB UG (haftungsbeschraenkt) and others.
3   * All rights reserved. This program and the accompanying materials
4   * are made available under the terms of the Eclipse Public License v1.0
5   * which accompanies this distribution, and is available at
```

```
6   * http://www.eclipse.org/legal/epl-v10.html
7   */
8  package org.openhab.binding.nutzerwelten.internal;
9
10 import static org.openhab.binding.nutzerwelten.NutzerWeltenBindingConstants.*;
11
12 import java.util.HashMap;
13 import java.util.Hashtable;
14 import java.util.Map;
15 import java.util.Set;
16
17 import org.eclipse.smarthome.config.core.Configuration;
18 import org.eclipse.smarthome.config.discovery.DiscoveryService;
19 import org.eclipse.smarthome.core.thing.Bridge;
20 import org.eclipse.smarthome.core.thing.Thing;
21 import org.eclipse.smarthome.core.thing.ThingTypeUID;
22 import org.eclipse.smarthome.core.thing.ThingUID;
23 import org.eclipse.smarthome.core.thing.binding.BaseThingHandlerFactory;
24 import org.eclipse.smarthome.core.thing.binding.ThingHandler;
25 import org.openhab.binding.nutzerwelten.handler.BaseNutzerWeltenHandler;
26 import org.openhab.binding.nutzerwelten.handler.NutzerWeltenBridgeHandler;
27 import org.openhab.binding.nutzerwelten.handler
28         .NutzerWeltenCookerWatcherHandler;
29 import org.openhab.binding.nutzerwelten.handler
30         .NutzerWeltenFallDetectorHandler;
31 import org.openhab.binding.nutzerwelten.handler
32         .NutzerWeltenMovementDetectorHandler;
33 import org.openhab.binding.nutzerwelten.handler.NutzerWeltenPlugHandler;
34 import org.openhab.binding.nutzerwelten.handler
35         .NutzerWeltenWaterDetectorHandler;
36 import org.openhab.binding.nutzerwelten.handler
37         .NutzerWeltenWindowContactHandler;
38 import org.openhab.binding.nutzerwelten.internal.discovery
39         .NutzerWeltenDiscoveryService;
40 import org.osgi.framework.ServiceRegistration;
41
42 import com.google.common.collect.Sets;
43
44 /**
45  * The {@link NutzerWeltenHandlerFactory} is responsible for creating things
46  * and thing handlers.
47  *
48  * @author w_gerlach - Initial contribution
49  */
50 public class NutzerWeltenHandlerFactory extends BaseThingHandlerFactory {
51
52     private final static Set<ThingTypeUID> SUPPORTED_THING_TYPES_UIDS =
53             Sets.union(SUPPORTED_BRIDGE_TYPES, SUPPORTED_UDP_THING_TYPES);
54
55     private Map<ThingUID, ServiceRegistration<?>> discoveryServiceRegs =
56             new HashMap<>();
```

```
57
58      @Override
59      public boolean supportsThingType(ThingTypeUID thingTypeUID) {
60          return SUPPORTED_THING_TYPES_UIDS.contains(thingTypeUID);
61      }
62
63      @Override
64      public Thing createThing(ThingTypeUID thingTypeUID,
65                               Configuration configuration,
66                               ThingUID thingUID, ThingUID bridgeUID) {
67          ThingUID newThingUID = thingUID;
68
69          if (SUPPORTED_BRIDGE_TYPES.contains(thingTypeUID)) {
70              if (thingUID == null) {
71                  newThingUID = new ThingUID(THING_TYPE_BRIDGE, "bridge");
72              }
73              return super.createThing(
74                      thingTypeUID, configuration, newThingUID, null);
75                      // The bridge has no bridge
76          }
77          if (SUPPORTED_UDP_THING_TYPES.contains(thingTypeUID)) {
78              if (thingUID == null) {
79                  String deviceType = (String) configuration.get(
80                          CONFIG_DEVICE_TAG);
81                  String deviceNumber = (String) configuration.get(
82                          CONFIG_DEVICE_NUMBER);
83                  newThingUID = new ThingUID(thingTypeUID, deviceType +
84                          deviceNumber, bridgeUID.getBindingId());
85              }
86              return super.createThing(
87                      thingTypeUID, configuration, newThingUID, bridgeUID);
88                      // Create Thing with bridge
89          }
90          throw new IllegalArgumentException(
91                  "The thing type " + thingTypeUID +
92                  " is not supported by the NutzerWelten binding.");
93      }
94
95      @Override
96      protected ThingHandler createHandler(Thing thing) {
97          ThingTypeUID thingTypeUID = thing.getThingTypeUID();
98          if (thingTypeUID.equals(THING_TYPE_BRIDGE)) {
99              return createBridgeHandler(thing);
100         }
101         return createThingHandlerFromUID(thing, thingTypeUID);
102     }
103
104     private ThingHandler createBridgeHandler(Thing thing) {
105         NutzerWeltenBridgeHandler handler =
106                 new NutzerWeltenBridgeHandler((Bridge) thing);
107         registerDiscoveryService(handler);
```

```
108         return handler;
109     }
110 
111     private ThingHandler createThingHandlerFromUID(Thing thing,
112                                                    ThingTypeUID thingTypeUID) {
113         // TODO weitere Handler Erstellung hier ----------------------------
114         if (thingTypeUID.equals(THING_TYPE_PLUG)) {
115             return new NutzerWeltenPlugHandler(thing);
116         }
117         if (thingTypeUID.equals(THING_TYPE_WATER_DETECTOR)) {
118             return new NutzerWeltenWaterDetectorHandler(thing);
119         }
120         if (thingTypeUID.equals(THING_TYPE_WINDOW_CONTACT)) {
121             return new NutzerWeltenWindowContactHandler(thing);
122         }
123         if (thingTypeUID.equals(THING_TYPE_MOVEMENT_DETECTOR)) {
124             return new NutzerWeltenMovementDetectorHandler(thing);
125         }
126         if (thingTypeUID.equals(THING_TYPE_FALL_DETECTOR)) {
127             return new NutzerWeltenFallDetectorHandler(thing);
128         }
129         if (thingTypeUID.equals(THING_TYPE_COOKER_WATCHER)) {
130             return new NutzerWeltenCookerWatcherHandler(thing);
131         }
132         return null;
133     }
134 
135     private synchronized void registerDiscoveryService(
136                                       NutzerWeltenBridgeHandler bridge) {
137         NutzerWeltenDiscoveryService discoveryService =
138                 new NutzerWeltenDiscoveryService(bridge);
139         discoveryService.activate();
140         this.discoveryServiceRegs.put(bridge.getThing().getUID(),
141                 bundleContext.registerService(
142                 DiscoveryService.class.getName(), discoveryService,
143                 new Hashtable<String, Object>()));
144 
145     }
146 
147     @Override
148     protected synchronized void removeHandler(ThingHandler thingHandler) {
149         if (thingHandler instanceof BaseNutzerWeltenHandler) {
150             ServiceRegistration<?> serviceReg = this.discoveryServiceRegs.get(
151                     thingHandler.getThing().getUID());
152             if (serviceReg != null) {
153                 // remove discovery service, if bridge handler is removed
154                 NutzerWeltenDiscoveryService service = (
155                         NutzerWeltenDiscoveryService) bundleContext
156                         .getService(serviceReg.getReference());
157                 service.deactivate();
158                 serviceReg.unregister();
```

```
159                 discoveryServiceRegs.remove(thingHandler.getThing().getUID());
160             }
161         }
162     }
163 }
```

Listing 18: NutzerWeltenDiscoveryService.java

```
1  package org.openhab.binding.nutzerwelten.internal.discovery;
2
3  import static org.openhab.binding.nutzerwelten.NutzerWeltenBindingConstants.*;
4
5  import java.util.Date;
6  import java.util.HashMap;
7  import java.util.Map;
8  import java.util.Set;
9
10 import org.eclipse.smarthome.config.discovery.AbstractDiscoveryService;
11 import org.eclipse.smarthome.config.discovery.DiscoveryResult;
12 import org.eclipse.smarthome.config.discovery.DiscoveryResultBuilder;
13 import org.eclipse.smarthome.core.thing.ThingTypeUID;
14 import org.eclipse.smarthome.core.thing.ThingUID;
15 import org.openhab.binding.nutzerwelten.handler.NutzerWeltenBridgeHandler;
16
17 import com.google.common.collect.Sets;
18
19 /**
20  *
21  * @author w_Gerlach
22  *
23  */
24 public class NutzerWeltenDiscoveryService extends AbstractDiscoveryService {
25
26     private final static Set<ThingTypeUID> SUPPORTED_THING_TYPES_UIDS =
27             Sets.union(SUPPORTED_BRIDGE_TYPES, SUPPORTED_UDP_THING_TYPES);
28     private static int timeout = 60;
29     private NutzerWeltenBridgeHandler bridgeHandler = null;
30
31     public NutzerWeltenDiscoveryService(NutzerWeltenBridgeHandler handler)
32                                             throws IllegalArgumentException {
33         super(timeout);
34         this.bridgeHandler = handler;
35     }
36
37     public void activate() {
38         bridgeHandler.registerDiscoveryService(this);
39     }
40
41     @Override
42     public void deactivate() {
43         removeOlderResults(new Date().getTime());
44         bridgeHandler.unregisterDiscoveryService(this);
```

```
45      }
46
47      @Override
48      protected void startScan() {
49      }
50
51      @Override
52      protected void stopScan() {
53          super.stopScan();
54      }
55
56      // @Override
57      public boolean addNutzerWeltenDevice(String discoveredThing) {
58          String id = null;
59          String number = null;
60
61          String[] devider = null;
62          devider = discoveredThing.split("\\+");
63
64          id = devider[0].trim();
65          number = devider[1].trim();
66
67          ThingTypeUID thingTypeUID = new ThingTypeUID(BINDING_ID, id);
68
69          if (SUPPORTED_UDP_THING_TYPES.contains(thingTypeUID)) {
70              ThingUID bridgeUID = bridgeHandler.getThing().getUID();
71
72              ThingUID thingUID =
73                      new ThingUID(thingTypeUID, bridgeUID, id + number);
74              Map<String, Object> properties = new HashMap<>(2);
75              properties.put(CONFIG_DEVICE_TAG, id);
76              properties.put(CONFIG_DEVICE_NUMBER, number);
77              DiscoveryResult discoveryResult =
78                      DiscoveryResultBuilder.create(thingUID).
79                      withProperties(properties).withBridge(bridgeUID).
80                      withLabel(id + number).build();
81              thingDiscovered(discoveryResult);
82          }
83          return false;
84      }
85
86  }
```

A.6. Message Klasse

Listing 19: Message.java

```
1  package org.openhab.binding.nutzerwelten.tools;
2
3  import java.net.DatagramPacket;
4  import java.net.InetAddress;
```

```
5
6   /**
7    * Class for Communication between the NutzerWelten ICommunicator and the
8    * System
9    *
10   * @author w_Gerlach
11   *
12   */
13  public class Message {
14
15      private final static int PREAMBLE = 0;
16      // Not used because of wrong calculations in some devices
17      private final static int DEVICE_NAME = 1;
18      private final static int DEVICE_NUMBER = 2;
19      private final static int DEVICE_MESSAGE = 3;
20      private final static int DEVICE_MESSAGE_COUNTER = 4;
21      // Not used because only the emergency button would need it
22
23      private String device;
24      private int number;
25      private String data;
26      private int pakNum;
27
28      /**
29       * Message construction for Message without packet Number
30       *
31       * @param device device name example "sd" for Power Plug
32       * @param number device number
33       * @param data data to transfer
34       */
35      public Message(String device, int number, String data) {
36          this.device = device;
37          this.number = number;
38          this.data = data.trim();
39          this.pakNum = 0;
40      }
41
42      /**
43       * Message construction for Message with packet Number mostly the
44       * Emergency Button
45       *
46       * TODO make it usable
47       *
48       * @param device device name example "nk" for the Emergency Button
49       * @param number device number
50       * @param data data to Transfer
51       * @param pakNum Packet Number
52       */
53      public Message(String device, int number, String data, int pakNum) {
54          this.device = device;
55          this.number = number;
```

```
56          this.data = data.trim();
57          this.pakNum = pakNum;
58      }
59
60      public String getDevice() {
61          return device;
62      }
63
64      public int getNumber() {
65          return number;
66      }
67
68      public String getData() {
69          return data;
70      }
71
72      public int getPakNum() {
73          return pakNum;
74      }
75
76      /**
77       * convert the data into the Message
78       *
79       * @return String with Message
80       */
81      public String getMessage() {
82          return device + ":" + String.format("%04x", number) + ":" + data;
83      }
84
85      /**
86       * returns the length of the Message
87       *
88       * @return
89       */
90      public int getMessageLength() {
91          /*
92           * 2 "sd" + 1 ":" + 4 "000a" + 1 ":"
93           */
94          return (device + ":" + String.format("%04x", number) +
95                  ":" + data).length();
96      }
97
98      /**
99       * return a unique device identifier for the OpenHab System consisting of
100      * the device name and the device number
101      *
102      * @return device tag plus number returns both with a plus for a better
103      *          differentiation From the eclipse SmartHome
104      *          UID example sd+af01
105      */
106     public String getFullDeviceName() {
```

```
107         return device + "+" + String.format("%04x", number);
108     }
109 
110     /**
111      * Creates a DatagramPacket from a Message
112      *
113      * @param message
114      * @return DatagramPacket
115      */
116     public static DatagramPacket createDatagramFromMessage(Message message,
117                                     InetAddress multicastGroup, int port) {
118         byte[] messageBuffer = new byte[256];
119         messageBuffer = ("p" + String.format("%03d",
120                 message.getMessageLength()) + ":" + message.getMessage())
121                 .getBytes();
122         return new DatagramPacket(messageBuffer, messageBuffer.length,
123                 multicastGroup, port);
124     }
125 
126     /**
127      * reads the received Message from a DatagramPacket
128      * Every letter is Transformed to lower Case like it should be transmitted
129      * Return NULL if Message not valid
130      *
131      * @param packet
132      * @return
133      */
134     public static Message createMessageFromDatagram(DatagramPacket packet) {
135         String message = new String(packet.getData()).toLowerCase();
136         String[] messageParts = null;
137         Message returnMessage = null;
138         if (messageIsValid(message)) {
139             if (messageIsEncrypted(message)) {
140                 // logger.info("encryption not implemented jet");
141             }
142             messageParts = message.split(":");
143             returnMessage = new Message(messageParts[DEVICE_NAME],
144                     Integer.parseInt(messageParts[DEVICE_NUMBER], 16),
145                     messageParts[DEVICE_MESSAGE]);
146 
147         }
148         return returnMessage;
149     }
150 
151     /**
152      *
153      * @param message
154      * @return
155      */
156     private static boolean messageIsValid(String message) {
157         boolean valid = false;
```

```
158         if (message != null) {
159             String msgbackup = message.substring(0, 5);
160             msgbackup = msgbackup.replaceFirst("[ep]", "Z");
161             msgbackup = msgbackup.replaceAll("[0-9a-f]", "X");
162             if (msgbackup.equals("ZXXX:")) {
163             // compare with remaining String after Masking
164                 valid = true;
165             }
166         }
167         return valid;
168     }
169 
170     /**
171      * return true if Message is encrypted
172      *
173      * @param message
174      * @return
175      */
176     private static boolean messageIsEncrypted(String message) {
177         boolean valid = false;
178         if (message.charAt(0) == 'e') {
179             valid = true;
180         }
181         return valid;
182     }
183 
184 }
```

A.7. XML Thing Definitionen

Listing 20: Bewegungsmelder bm.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <thing:thing-descriptions bindingId="nutzerwelten"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xmlns:thing=
5              "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
6          xsi:schemaLocation=
7              "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
8              http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">
9 
10     <!-- Window Contact Thing Type -->
11     <thing-type id="bm">
12         <supported-bridge-type-refs>
13             <bridge-type-ref id="nutzerweltenbridge" />
14         </supported-bridge-type-refs>
15 
16         <label>NutzerWelten Movement Detector</label>
17         <description>Movement Detector for NutzerWelten Binding</description>
18 
19         <channels>
```

```
20              <channel id="movementDetector" typeId="movementDetector"/>
21              <channel id="alive" typeId="alive"/>
22          </channels>
23
24          <config-description>
25              <parameter name="deviceTag" type="text">
26                  <label>Name of the device type</label>
27                  <description>
28                      Name of the device type.
29                      Equal for all devices of the same type.
30                  </description>
31                  <required>true</required>
32              </parameter>
33              <parameter name="deviceNumber" type="text">
34                  <label>Number of the device type</label>
35                  <description>Unique number of the device</description>
36                  <required>true</required>
37              </parameter>
38          </config-description>
39      </thing-type>
40
41
42  </thing:thing-descriptions>
```

Listing 21: BridgeHandler bridge.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <thing:thing-descriptions bindingId="nutzerwelten"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xmlns:thing=
5               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
6           xsi:schemaLocation
7               ="http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
8               http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">
9
10      <!-- NutzerWelten Bridge -->
11      <bridge-type id="nutzerweltenbridge">
12          <label>NutzerWelten Bridge</label>
13          <description>NutzerWelten Binding Bridge</description>
14          <properties>
15              <property name="vendor">Hochschule DÃ¼sseldorf</property>
16          </properties>
17          <config-description>
18              <parameter name="networkinterface" type="text">
19                  <label>Networkinterface</label>
20                  <description>Leave Empty for automatic search</description>
21              </parameter>
22          </config-description>
23
24      </bridge-type>
25  </thing:thing-descriptions>
```

Listing 22: Fensterdetektor fd.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<thing:thing-descriptions bindingId="nutzerwelten"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:thing=
            "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
        xsi:schemaLocation=
            "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
            http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">

    <!-- Window Contact Thing Type -->
    <thing-type id="fd">
        <supported-bridge-type-refs>
            <bridge-type-ref id="nutzerweltenbridge" />
        </supported-bridge-type-refs>

        <label>NutzerWelten Window Contact</label>
        <description>Window Contact for NutzerWelten Binding</description>

        <channels>
            <channel id="windowContact" typeId="windowContact"/>
            <channel id="battery" typeId="battery"/>
        </channels>

        <config-description>
            <parameter name="deviceTag" type="text">
                <label>Name of the device type</label>
                <description>
                    Name of the device type.
                    Equal for all devices of the same type.
                </description>
                <required>true</required>
            </parameter>
            <parameter name="deviceNumber" type="text">
                <label>Number of the device type</label>
                <description>Unique number of the device</description>
                <required>true</required>
            </parameter>
        </config-description>
    </thing-type>


</thing:thing-descriptions>
```

Listing 23: Herdüberwachung hu.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<thing:thing-descriptions bindingId="nutzerwelten"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:thing=
            "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
```

```
6           xsi:schemaLocation=
7               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
8               http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">
9
10      <!-- Fall Detector Thing Type -->
11      <thing-type id="hu">
12          <supported-bridge-type-refs>
13              <bridge-type-ref id="nutzerweltenbridge" />
14          </supported-bridge-type-refs>
15
16          <label>NutzerWelten Cooker Watcher</label>
17          <description>Cooker Watcher for NutzerWelten Binding</description>
18
19          <channels>
20              <channel id="cookerState" typeId="cookerState"/>
21              <channel id="cookerStateString" typeId="cookerStateString"/>
22          </channels>
23
24          <config-description>
25              <parameter name="deviceTag" type="text">
26                  <label>Name of the device type</label>
27                  <description>
28                      Name of the device type.
29                      Equal for all devices of the same type.
30                  </description>
31                  <required>true</required>
32              </parameter>
33              <parameter name="deviceNumber" type="text">
34                  <label>Number of the device type</label>
35                  <description>Unique Number of the device.</description>
36                  <required>true</required>
37              </parameter>
38          </config-description>
39      </thing-type>
40
41
42  </thing:thing-descriptions>
```

Listing 24: Notfallknopf nk.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <thing:thing-descriptions bindingId="nutzerwelten"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xmlns:thing=
5               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
6           xsi:schemaLocation=
7               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
8               http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">
9
10      <!-- Fall Detector Thing Type -->
11      <thing-type id="nk">
12          <supported-bridge-type-refs>
```

```
13              <bridge-type-ref id="nutzerweltenbridge" />
14          </supported-bridge-type-refs>
15
16          <label>NutzerWelten Fall Detector</label>
17          <description>Fall Detector for NutzerWelten Binding</description>
18
19          <channels>
20              <channel id="fallAlert" typeId="fallAlert"/>
21              <channel id="alive" typeId="alive"/>
22              <channel id="buttonAlert" typeId="buttonAlert"/>
23              <channel id="alertImpuls" typeId="alertImpuls"/>
24              <channel id="alertReset" typeId="alertReset"/>
25              <channel id="fallAlertString" typeId="fallAlertString"/>
26          </channels>
27
28          <config-description>
29              <parameter name="deviceTag" type="text">
30                  <label>Name of the device type</label>
31                  <description>
32                      Name of the device type.
33                      Equal for all devices of the same type.
34                  </description>
35                  <required>true</required>
36              </parameter>
37              <parameter name="deviceNumber" type="text">
38                  <label>Number of the device type</label>
39                  <description>Unique Number of the device.</description>
40                  <required>true</required>
41              </parameter>
42          </config-description>
43      </thing-type>
44
45
46  </thing:thing-descriptions>
```

Listing 25: Funksteckdose sd.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <thing:thing-descriptions bindingId="nutzerwelten"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xmlns:thing=
5               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
6           xsi:schemaLocation=
7               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
8               http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">
9
10      <!-- Power Plug Thing Type -->
11      <thing-type id="sd">
12          <supported-bridge-type-refs>
13              <bridge-type-ref id="nutzerweltenbridge" />
14          </supported-bridge-type-refs>
15
```

```
16          <label>NutzerWelten remote power-plug</label>
17          <description>NutzerWelten power plug thing </description>
18          <channels>
19              <channel typeId="onOff" id="onOff"></channel>
20          </channels>
21          <config-description>
22              <parameter name="deviceTag" type="text">
23                  <label>Name of the device type</label>
24                  <description>
25                      Name of the device type.
26                      Equal for all devices of the same type.
27                  </description>
28                  <required>true</required>
29              </parameter>
30              <parameter name="deviceNumber" type="text">
31                  <label>Number of the device type</label>
32                  <description>Unique Number of the device.</description>
33                  <required>true</required>
34              </parameter>
35          </config-description>
36      </thing-type>
37  </thing:thing-descriptions>
```

Listing 26: Wassermelder wm.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <thing:thing-descriptions bindingId="nutzerwelten"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xmlns:thing=
5               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
6           xsi:schemaLocation=
7               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0 http://eclipse.org/sma
8
9       <!-- Water Detector Thing Type -->
10      <thing-type id="wm">
11          <supported-bridge-type-refs>
12              <bridge-type-ref id="nutzerweltenbridge" />
13          </supported-bridge-type-refs>
14
15          <label>NutzerWelten Water Detector</label>
16          <description>Water Detector for NutzerWelten Binding</description>
17
18          <channels>
19              <channel id="water" typeId="water"/>
20              <channel id="battery" typeId="battery"/>
21              <channel id="alive" typeId="alive"/>
22          </channels>
23
24          <config-description>
25              <parameter name="deviceTag" type="text">
26                  <label>Name of the device type</label>
27                  <description>
```

```
28                      Name of the device type.
29                      Equal for all devices of the same type.
30                  </description>
31                  <required>true</required>
32              </parameter>
33              <parameter name="deviceNumber" type="text">
34                  <label>Number of the device type</label>
35                  <description>Uniqe Number of the device.</description>
36                  <required>true</required>
37              </parameter>
38          </config-description>
39      </thing-type>
40
41
42  </thing:thing-descriptions>
```

A.8. XML Channel Definitionen

Listing 27: Kanäle channels.xml

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <thing:thing-descriptions bindingId="nutzerwelten"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xmlns:thing=
5               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0"
6           xsi:schemaLocation=
7               "http://eclipse.org/smarthome/schemas/thing-description/v1.0.0
8               http://eclipse.org/smarthome/schemas/thing-description-1.0.0.xsd">
9
10      <!-- Channel Switch Type -->
11      <channel-type id="onOff">
12          <item-type>Switch</item-type>
13          <label>Switch</label>
14      </channel-type>
15
16      <channel-type id="alertReset">
17          <item-type>Switch</item-type>
18          <label>Resets the alert of the emergency button</label>
19      </channel-type>
20
21
22      <!-- Channel Contact Type -->
23      <channel-type id="water">
24          <item-type>Contact</item-type>
25          <label>Water detection</label>
26      </channel-type>
27
28      <channel-type id="battery">
29          <item-type>Contact</item-type>
30          <label>Battery low warning</label>
31      </channel-type>
```

```
32
33      <channel-type id="windowContact">
34          <item-type>Contact</item-type>
35          <label>Shows if window is open or closed</label>
36      </channel-type>
37
38      <channel-type id="movementDetector">
39          <item-type>Contact</item-type>
40          <label>Shows Movement when closed</label>
41      </channel-type>
42
43      <channel-type id="alive">
44          <item-type>Contact</item-type>
45          <label>Shows that there is a transmission Problem</label>
46      </channel-type>
47
48      <channel-type id = "buttonAlert">
49          <item-type>Contact</item-type>
50          <label>Shows that someone hits the emergency Button</label>
51      </channel-type>
52
53      <channel-type id = "alertImpuls">
54          <item-type>Contact</item-type>
55          <label>triggers an impuls for emergency Rules</label>
56      </channel-type>
57
58      <!-- Channel Number Type -->
59      <channel-type id="fallAlert">
60          <item-type>Number</item-type>
61          <label>indicates for the state of the Fall detector </label>
62      </channel-type>
63
64      <channel-type id= "cookerState">
65          <item-type>Number</item-type>
66          <label>indicator for the state of the cooker watcher</label>
67      </channel-type>
68
69
70      <!-- Channel String Type -->
71      <channel-type id="fallAlertString">
72          <item-type>String</item-type>
73          <label>Shows the state of the fall detector</label>
74      </channel-type>
75      <channel-type id="cookerStateString">
76          <item-type>String</item-type>
77          <label>Shows the state of the cooker</label>
78      </channel-type>
79
80  </thing:thing-descriptions>
```

Literatur

[1] apple Deutschland: *Mit Apple HomeKit kompatibles Zubehör finden.* `https://support.apple.com/de-de/HT204903`, besucht: 15. Juni 2016.

[2] Bauer, Christian: *jupnp/jupnp.* `https://github.com/jupnp/jupnp/blob/master/bundles/org.jupnp/src/main/java/org/jupnp/transport/impl/MulticastReceiverImpl.java`, besucht: 06. Juni 2016.

[3] Bundeszentrale für politische Bildung: *Bevölkerungsentwicklung und Altersstruktur.* `http://www.bpb.de/nachschlagen/zahlen-und-fakten/soziale-situation-in-deutschland/61541/altersstruktur?zahlenfakten=detail`, besucht: 20 April 2016.

[4] Eclipse Foundation: *Coding Guidlines.* `http://www.eclipse.org/smarthome/documentation/development/guidelines.html`, besucht: 3. Mai 2016.

[5] Eclipse Foundation: *Eclipse SmartHome.* `http://www.eclipse.org/smarthome/`, besucht: 25.04.2016.

[6] Eclipse Foundation: *Eclipse SmartHome.* `https://projects.eclipse.org/projects/technology.smarthome`, besucht: 25. April 2016.

[7] Eclipse Foundation: *Items.* `http://www.eclipse.org/smarthome/documentation/concepts/items.html`, besucht: 27. April 2016.

[8] Eclipse Foundation: *Things.* `http://www.eclipse.org/smarthome/documentation/concepts/things.html`, besucht: 13.06.2016.

[9] Eclipse Foundation: *Things.* `http://www.eclipse.org/smarthome/documentation/concepts/things.html`, besucht: 26. April 2016.

[10] Elkstein, M.: *Learn REST: A Tutorial.* `http://rest.elkstein.org/`, besucht: 29. April 2016.

[11] Foundation, Apache Software: *Overview.* `https://karaf.apache.org/manual/latest/overview.html`, besucht: 17. Juni 2016.

[12] Gerd Wütherich, Nils Hartmann, Bernd Kolb und Matthias Lübken: *Die OSGI Service Plattform.* dpunkt.verlag, ISBN 987-3-89864-457-0.

[13] GNU: *Lizenzen.* `http://www.gnu.org/licenses/#LGPL`, besucht: 16. Juni 2016.

[14] Goll, Joachim und Manfred Daussmann: *Architektur- und Entwurfsmuster der Softwaretechnik*. Springer Vieweg, ISBN 978-3-8348-2431-8.

[15] Götz, Christian: *MQTT: Protokoll für das Internet der Dinge*. `http://heise.de/-2168152`, besucht: 12. Mai 2016.

[16] Heinle, Stefan: *Heimautomation mit KNX, DALI,1-Wire und Co.* Rheinwerk Computing, ISBN 978-3-8362-3461-0.

[17] Hildner, Klaus: *Scripts*. `https://github.com/openhab/openhab/wiki/Scripts`, besucht: 6. Mai 2016.

[18] Hochschule Düsseldorf: *Studie SICHERHEIT Informationsbroschüre für Professionelle*.

[19] ITWissen: *UDP (user datagrm protocol)*. `http://www.itwissen.info/definition/lexikon/user-datagram-protocol-UDP-UDP-Protokoll.html`, besucht: 07. Juni 2016.

[20] Jan Schäfer, Marcus Thoss, Reinhold Kröger, Oliver v. Fragstein, Fabian Pursche, Wolfgang Lux und Ulrich Schaarschmidt: *WieDAS Pflichtenheft*. Hochschule Rhein-Main & Fachhochschule Düsseldorf, 2010.

[21] Julienne Gaelle, Ngan youe: *Konzeption einer Benutzerschnittstelle für Nutzerwelten in HTML5*.

[22] Kai Kreuzer und Thomas Eichstädt Engelen: *Durchbruch Offene Plattformen verhelfen Smart Home zum Erfolg*. In: iX Developer 2/2014 (Herausgeber): *heise Verlag*.

[23] Kai Kreuzer und Bjorn Hamra: *smarthome/extensions/binding/org.eclipse.smarthome. binding.hue/ESH-INF/thing/*. `https://github.com/eclipse/smarthome/tree/master/extensions/binding/org.eclipse.smarthome.binding.hue/ESH-INF/thing`, besucht: 9. Mai 2016.

[24] Kaps, Reiko: *Netzwerke mit UPnP einrichten und steuern*. `http://heise.de/-221520`, besucht: 17. Mai 2016.

[25] Kreuzer, Kai: *Developing a new binding for openHAB 2*. `https://github.com/openhab/openhab-distro/blob/master/docs/sources/development/bindings.md`, besucht: 27. April 2016.

[26] Kreuzer, Kai: *"LeapingËdge Home Automation at JavaOne*. http://kaikreuzer.blogspot.de/2013/09/leaping-edge-home-automation-at-javaone.html, besucht: 25. April 2016.

[27] Kreuzer, Kai: *openHAB 2 Distribution*. https://github.com/openhab/openhab-distro, besucht: 27. April 2016.

[28] Kreuzer, Kai: *openHAB 2.0 and Eclipse SmartHome*. http://kaikreuzer.blogspot.de/2014/06/openhab-20-and-eclipse-smarthome.html, besucht: 25. April 2016.

[29] Kreuzer, Kai: *openHAB Foundation*. www.kaikreuzer.de/2016/05/21/openhab-foundation/, besucht: 30. Mai 2016.

[30] Kreuzer, Kai: *openHAB is out*. http://kaikreuzer.blogspot.de/2010/02/openhab-is-out.html, besucht: 25. April 2016.

[31] Kreuzer, Kai: *Privacy in the Smart Home - Why we need an Intranet of Things*. http://kaikreuzer.blogspot.de/2014/02/privacy-in-smart-home-why-we-need.html, besucht: 25. April 2016.

[32] Meuter, Kim: *Implementierung der Funktionalität des NutzerWelten Systems in openHAB*.

[33] Müller, Oliver: *Zeitgemäß Advanced Encryption Standard*. In: iX Developer 2/2014 (Herausgeber): *heise Verlag*.

[34] Oracle Corporation: *Class MulticastSocket*. https://docs.oracle.com/javase/8/docs/api/java/net/MulticastSocket.html, besucht: 24. Mai 2016.

[35] Oracle Corporation: *Java SE Embedded 8 Compact Profiles Overview*. http://www.oracle.com/technetwork/java/embedded/resources/tech/compact-profiles-overview-2157132.html, besucht: 29. April 2016.

[36] Riegler, Gerhard: *openhab/openhab2-addons*. https://github.com/openhab/openhab2-addons/blob/master/addons/binding/org.openhab.binding.homematic/src/main/java/org/openhab/binding/homematic/internal/communicator/server/BinRpcNetworkService.java, besucht: 09. Juni 2016.

[37] Selfhtml Wiki: *XML*. https://wiki.selfhtml.org/wiki/XML, besucht: 18. Juni 2016.

[38] statistisches Bundesamt: *Anzahl der Pflegebedürftigen steigt vor allem bei den Hochbetagten*. `http://www.demografie-portal.de/SharedDocs/Informieren/DE/ZahlenFakten/Pflegebeduerftige_Anzahl.html`, besucht: 21. April 2016.

[39] Ullenbron, Christian: *Java ist auch eine Insel*. Rheinwerk Computing. `http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_14_004.htm#mj2f866e14a50f07accd817e14e068e022`, besucht: 23. Mai 2016.

[40] www.mikrocontoller.net: *FIFO*. `http://www.mikrocontroller.net/articles/FIFO`, besucht: 19. Juni 2016.

[41] XMPP Standards Foundation: *An Overview of XMPP*. `https://xmpp.org/about/technology/overview.html`, besucht: 30. Mai 2016.