technische universität
dortmund

Meisterarbeit

FLASH: Ein on-premise Workflow Automation Service im Kontext Smart Home – Prototypische Implementierung basierend auf dem Eclipse Open IoT Stack

Konstantin Tkachuk
Februar 2017

Gutachter:
Prof. Dr.-Ing. Olaf Spinczyk
Dr. Sven Seiler

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl Informatik 12
http://ls12-www.cs.tu-dortmund.de

In Kooperation mit:
Materna GmbH

Zusammenfassung

Abstract text

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Motivation und Hintergrund

1.2 Ziele der Arbeit

Ziel dieser Arbeit ist es die Welten von Smart Home und webbasierten Task Automation Services zusammen zu bringen. Es soll ein Demonstrator entwickelt werden, der die Funktionalitäten beider Ansätze kombiniert und in einem gemeinsamen Kontext anbietet. Konkreter sind die Ziele in Sektion 2.2.1 erläutert.

1.3 Aufbau der Arbeit

Nach der Einleitung wird der aktuelle Stand der Forschung bezüglich *Internet der Dinge*, *SmartHome* und *Task Automation Services* vorgestellt und die Ziele der Arbeit daraus abgeleitet. Daraufhin wird in Kapitel 3 das Framework Eclipse SmartHome (ESH) vorgestellt. In Kapitel 4 wird konkret betrachtet, wie die Funktionalitäten eines Task Automation Services in ESH integriert werden können.

Im folgenden Kapitel 5 wird die Implementierung des Entwurfs mit Hilfe von entsprechenden Diagrammen vorgestellt. Der Quell-Code ist auf der mit der Arbeit mitgelieferten CD einsehbar. Danach wird in Abschnitt 6 die Implementierung evaluiert. Schließlich folgt noch eine kurze Zusammenfassung der geleisteten Arbeit und ein Ausblick auf mögliche Weiterentwicklungen in Kapitel 7.

Kapitel 2

Stand der Forschung

In diesem Kapitel wird der aktuelle Stand der Forschung des Internets der Dinge erläutert. Es wird besonders auf die Themen Smart Home und Task Automation Services eingegangen, sowie darauf, welche Vorteile sich durch das Kombinieren dieser Ansätze erhoffen lassen.

2.1 Internet of Things

Die enorm steigende Anzahl von „intelligenten Gegenständen“ mit eingebetteten Computern, die den Menschen im alltäglichen Leben unterstützen sollen, hat zu der Prägung des Begriffs „Internet der Dinge“ (IoT) geführt. Jedes dieser Dinge hat seine eigene Funktionalität und im Verbund stellen sie eine große Menge an Daten zur Verfügung. Im Rahmen von zahlreichen Forschungsprojekten [12] werden Möglichkeiten untersucht, IoT mit unterschiedlichen Technologien zu kombinieren. Unter anderem werden Technologien, wie Cloud Computing, Machine-2-Machine Learning und Semantic Web kritisch betrachtet. Außerdem werden Kernprinzipien, wie Architektur und Standardisierung intensiv recherchiert. Sie werden in dedizierten Forschungsprojekten [14][13] immer wieder aufgegriffen.

Im Laufe der Zeit haben sich verschiedene Aspekte des IoT herausgebildet, unter anderem das Smart Home.

2.1.1 Smart Home

Ein mit IoT eng verwobenes Thema ist das Smart Home[15], welches die elektronische Steuerung von ausgewählten Geräten mit z.B. einer Rule Engine kombiniert um eine Automatisierung des Geräteverhaltens in einem Zusammenspiel zwischen Sensorik und Aktortik zu erreichen. Smart Home grenzt sich von IoT ab indem es auf Sensoren und Aktoren spezialisiert ist, die im Kontext eines Hauses relevant sind.

Bis dato wurden zahlreiche Smart Home Lösungen von verschiedenen Anbietern entwickelt. Man kann prinzipiell zwei Arten von Lösungen unterscheiden. *Proprietäre* Produkte (z.B. *RWE SmartHome*[4]) spezialisieren sich auf eine sehr begrenzte Anzahl von Geräten und bemühen sich maximale Unterstützung für diese Geräte zu bieten. Dies sorgt für eine Fragmentierung des Marktes. *Open Source* Lösungen hingegen verfolgen das Ziel möglichst offen für verschiedene Geräte und Protokolle zu bleiben.

Da Open Source Lösungen aktuell noch aktiv in der Entwicklung sind und hauptsächlich als Software existieren, erfordert ihr Einsatz von Nutzern ein gewisses Maß an technischen Vorkenntnissen. In der Regel muss der Nutzer passende Hardware bereitstellen und die Software selbst installieren. Bei Problemen kann Support bestenfalls auf Foren von der Community gefunden werden.

Diese Gründe führen dazu, dass Open Source Lösungen derzeit noch nicht massenhaft zum Einsatz kommen.

2.1.2 Task Automation Services

Die Automatisierung von Aufgaben ist eins der zentralen Bestreben unseres alltäglichen Lebens. Es macht das Leben einfacher und erlaubt uns kostbare Zeit zu sparen. Ob Notifikation auf dem Smartphone, wenn eine Email eingeht oder das Einschalten von Lampen, wenn ein Raum betreten wird, solche Automatisierung ist heutzutage überall zu finden. Lange Zeit musste jede derartige Automatisierung einzeln entworfen, konfiguriert und implementiert werden. Doch die steigende Anzahl von intelligenten Gegenständen und die Allgegenwärtigkeit des Internets belassen dies der Vergangenheit. Nun hat sich der Ansatz der Task Automation Services[11] gebildet.

Ein Task Automation Service (TAS) ist ein Dienst, der es Endnutzern ermöglicht das Verhalten von verschiedenen Services und Geräten in eigenen Szenarien jederzeit selbst zu automatisieren. Solche Szenarien basieren auf Event-Condition-Action (ECA) Regeln[9], welche es ermöglichen, auf Events unter festgelegten Bedingungen mit entsprechenden Aktionen zu reagieren. Meistens wird dies durch einen intuitiven visuellen Regel Editor ermöglicht.

Aktuell gibt es noch vergleichsweise wenige TAS. Ein Überblick über existierende Services bietet Abbildung 2.1.

Wie in der Abbildung zu sehen ist, gibt es unterschiedliche Ansätze. Einige TAS sind in der Cloud angesiedelt, was bedeutet, dass sie, sofern Internet verfügbar ist, jederzeit und von überall erreichbar sind. Die aktuell mächtigsten und bekanntesten TAS sind *IFTTT* [6] und *Zapier*[8]. Sie unterstützen hunderte unterschiedlicher Web Services, bieten aber keine Möglichkeit mit Geräten direkt zu interagieren. Ihr Fokus ist es zu ermöglichen eine Vielzahl von Szenarien auf eine einfache Art und Weise zu erstellen. Auf komplexere Regeln

		Web						Smartphone				Home	
		Ifttt	Zapier	CloudWork	Elastic.io	ItDuzzit	Wappwolf	On{x}	Tasker	Atooma	Automatelt	WigWag	Webee
Channels	Web channel support	✓	✓	✓	✓	✓	Few	Few	✓	✓	✓	✓	✓
	Device channel support	Few	×	×	×	×	×	✓	✓	✓	✓	✓	✓
	Smartphone resources as channels	✓	×	×	×	×	×	✓	✓	✓	✓	×	×
	Public channels support	✓	×	×	✓	✓	×	✓	×	×	×	×	×
	Pipe channel support	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group channel support	×	×	×	×	×	×	×	×	×	×	Few	×
	Device channel discovery	×	×	×	×	×	×	×	×	×	×	✓	Few
Rules	Multi-event rules	×	×	×	×	×	×	✓	✓	×	×	✓	×
	Multi-action rules	×	×	×	✓	×	×	✓	✓	×	✓	✓	✓
	Chain rules	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group rules	×	×	×	×	×	×	×	×	×	×	Few	×
	Collision handling	×	×	×	×	×	×	×	×	×	×	×	×
	Predefined common rules	×	×	✓	×	✓	✓	×	×	×	✓	✓	✓
	Rule execution profile	WD	WD	WD	WD	WD	WD	DD	DD	DD	DD	DD	DD
TAS	Visual rule editor	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓
	Provides API	×	✓	×	✓	×	×	×	×	×	×	×	×
	Programming language	×	×	×	✓	×	×	✓	Few	×	✓	✓	✓

* ✓ = supported; × = not supported; Few = few support; WD = Web-driven execution profile; and DD = device-driven execution profile.

Abbildung 2.1: Überblick über existierende Task Automation Services [11]

und Szenarien sind ihre Rule Engines nicht ausgelegt. Um diese TAS zu nutzen, muss man jedoch bereit sein, sämtliche Zugriffsrechte, die für die zu automatisierenden Dienste (z.B. *Facebook*, *Twitter*, etc.) benötigt werden, dem Cloud Service anzuvertrauen.

Andere Task Automation Services arbeiten lokal auf Smartphones. Solche TAS konzentrieren sich auf die Automatisierung von den auf dem Gerät laufenden Services. Die Unterstützung von der Automatisierung von Web Services ist nur in dem Umfang gegeben, in dem diese Web Services direkten Kontakt mit dem Smartphone haben.

Schließlich gibt es TAS, die auf einer dedizierten Basis im Haus arbeiten. Solche Task Automation Services konzentrieren sich auf die direkte Steuerung von Geräten mithilfe der entsprechenden Protokolle. Im Grunde sind sie äquivalent zu Smart Home.

Im Rahmen dieser Arbeit wird Smart Home jedoch getrennt von TAS behandelt, da der Fokus der Automatisierung völlig unterschiedlich ist. Mehr dazu in Sektion 2.2.

IFTTT

Ein Beispiel für Task Automation Services in der Cloud ist IFTTT, welches eine große Anzahl verschiedener Services (*Facebook*, *Dropbox*, *Philips Hue*, etc.) integriert und eine rudimentäre Rule Engine anbietet, die es erlaubt auf eine einfache Art und Weise service-

übergreifende „if this than that“ Anweisungen zu hinterlegen, die der klassischen „EDV“ ähneln. Es wird eine feste Trigger Komponente gewählt, die etwas auslöst, daraufhin wird die Aktion festgelegt, die ausgeführt werden soll. Solche Condition/Command Paare werden in IFTTT *Recipes* genannt.

Bis dato lassen sich komplexere Szenarien mit *IFTTT* nicht abbilden. Das bedeutet, dass es keine Möglichkeit gibt, beispielsweise, Und- und Oder-Bedingungen zu definieren, die es ermöglichen würden, mehrere Conditions in einem Recipe zu verknüpfen.

Ein weiterer Aspekt von IFTTT ist, dass es für die Anbindung von Services Zugriffsrechte auf sämtliche Accounts des Users bedarf. Aus einer Datenschutz-Perspektive[10] stellt das ein Risiko für den Endnutzer dar, da seine sämtlichen Account-Daten an einer Stelle gesammelt sind. Im Falle einer Sicherheitslücke bei IFTTT wären alle damit gekoppelten Services in Gefahr.

2.2 Zentrale Idee

Wie in Sektion 2.1.2 zu sehen ist, kann man drei prinzipiell unterschiedliche Arten von Task Automation Services unterscheiden: Cloud Service, Smartphone Applikation und Smart Home. Sie weisen zwar viele Gemeinsamkeiten auf, doch allein aus den Namen lässt sich ablesen, dass es sich dabei um grundlegend verschiedene Lösungen handelt.

Die Cloud Services unterstützen nur begrenzt die „direkte“ Steuerung von Geräten. Außerdem sind die zum Einsatz kommenden Rule Engines sehr rudimentär. Smart Homes hingegen haben in der Regel mächtige Rule Engines und bieten die Möglichkeit komplexe Szenarien zu definieren. Zusätzlich sind sie in der Lage, Geräte direkt anzusteuern. Im Rahmen dieser Arbeit werden die Smartphone Applikationen nicht näher betrachtet.

Die zentrale Idee dieser Arbeit ist es ein Smart Home mit einem Cloud Service TAS zu verschmelzen. Dadurch soll ein Ganzes entstehen, welches mehr ist, als die Summe seiner Teile: Ein (Work-)Flow Automation Service im Kontext Smart Home (FLASH).

2.2.1 Ziele

Ziel dieser Arbeit ist es die Welten von Smart Home und webbasierten Task Automation Services zusammen zu bringen. Durch die Integration mit Smart Home soll die direkte Steuerung von Geräten befähigt werden. Außerdem soll die Mächtigkeit der Smart Home Rule Engine genutzt werden, um komplexere Regeln und Szenarien zu ermöglichen. Schließlich sollen die Datenschutzprobleme gelöst werden, indem sämtliche Zugriffsdaten on-premise galagert und dadurch nie einem Risiko ausgesetzt werden.

Es soll ein Demonstrator entwickelt werden, der die Funktionalitäten von Smart Home und TAS kombiniert und in einer on-premise Anwendung anbietet. Hierzu soll das Eclipse

SmartHome Framework als Basis verwendet und um Funktionalitäten von webbasierten TAS angereichert werden. Die entstandene Anwendung soll auf einem Raspberry Pi laufen.

Die entstandenen Funktionalitäten sollen anhand von einer Reihe von konkreten Services demonstriert werden. Unter anderem sollen ein Wetterdienst, ein Filesharing Service und ein Social Media Service angebunden werden. Die Zusammenarbeit dieser Services untereinander und mit Smart Home Geräten soll anhand von Beispiel-Regeln demonstriert werden. Außerdem soll es möglich sein neue Regeln zum System über eine Benutzeroberfläche hinzufügen zu können.

Zum Schluss soll der entstandene Demonstrator evaluiert werden. Es soll geprüft werden, inwiefern Task Automation Services als on-premise Lösung sinnvoll sind unter Betrachtung von Aspekten wie Reaktionszeiten und Netzwerklast. Hierzu soll ein Vergleich der erstellten Anwendung mit *IFTTT* stattfinden.

2.2.2 Konkrete Anforderungen

In dieser Sektion werden aus den oben genannten Zielen konkrete Anforderungen an den resultierenden Demonstrator abgeleitet und im Detail festgehalten.

Es wird erwartet, dass der Demonstrator folgende Eigenschaften erfüllt:

1. Der Demonstrator soll on-premise auf einem Raspberry Pi 3 Model B laufen.
2. Er soll intern wie eine Smart Home Zentrale agieren und in der Lage sein, Geräte innerhalb des Hauses auch ohne Internetverbindung steuern zu können.
3. Bei vorhandener Internetverbindung soll er in der Lage sein ausgewählte webbasierte Dienste zu automatisieren.
4. Sämtliche (Zugriffs-)Daten sollen lokal auf dem Gerät gelagert werden.
5. Die Webdienste sollen nahtlos in die Rule Engine integriert werden, sodass komplexe Szenarien ermöglicht werden.
6. Es soll eine grafische Benutzeroberfläche angeboten werden, in der der Nutzer in der Lage ist eigene Szenarien zur Laufzeit zu definieren.

Anzubindende Services

Im Rahmen der Arbeit sollen folgende populäre Internetdienste in den Demonstrator exemplarisch integriert werden:

1. **Twitter**[5] Der Demonstrator soll in der Lage sein für den Nutzer Tweets zu schreiben, sowie auf Tweets zu reagieren. Beispielsweise soll es möglich sein, Medien in Tweets automatisch in die Dropbox zu speichern.

2. **Dropbox**[1] Es soll möglich sein, automatisch Dateien in die Dropbox zu speichern, sowie auf das Ändern von existierenden Dokumenten zu reagieren.

3. **Wetterdienst**[3] Es sollen Wetterdaten aus dem Internet bezogen und auf konkrete Wetterbedingungen reagiert werden können.

4. **Email** Der Demonstrator soll in der Lage sein an beliebige Adressen Emails zu versenden.

Kapitel 3

Eclipse Smarthome

In diesem Kapitel wird *Eclipse SmartHome* (ESH) [2] vorgestellt. Es werden zunächst die grundlegenden Konzepte des Frameworks erläutert. Anschließend wird auf für die Arbeit relevante konkrete Aspekte näher eingegangen.

3.1 Überblick

Hier wird ein kurzer Überblick über Smarthome gegeben - Was ist ESH? - Zugrunde liegende Technologien - Zentrale Features von ESH - Wo kommt es zum Einsatz -

Eclipse SmartHome positioniert sich als ein Framework, dass als Grundlage für die Entwicklung von konkreten SmartHome Endlösungen dienen soll. Tatsächlich kommt es in vielen bekannten Produkten zum Einsatz, wie z.B. openHAB und Qivicon.

Architektur

Eclipse SmartHome hat eine serviceorientierte Architektur, basierend auf dem Java OSGi Framework. OSGi spezifiziert eine dynamische Softwareplattform, die hardwareunabhängig ist und es ermöglicht, Anwendungen zu modularisieren und zu verwalten. Der Funktionsweise der Kommunikation ist in Abbildung 3.1 veranschaulicht.

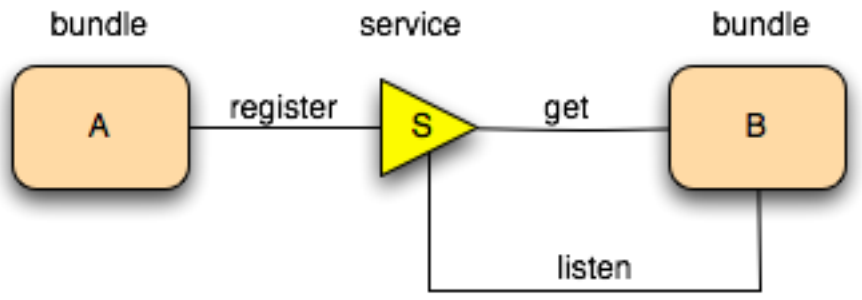

Abbildung 3.1: OSGi Komponentenmodell: Kommunikation zwischen Bundles [7]

Im Rahmen des OSGi Frameworks wird das gesamte Programm in verschiedene Softwarekomponenten (Bundles) aufgeteilt. Jede Komponente bietet und bezieht Services. Die angebotenen Services werden von den entsprechenden Bundles zur Laufzeit im System registriert, wonach andere Bundles, die diese Services beziehen, darüber informiert und ihrerseits gestartet werden können. Ein Bundle wird erst gestartet, wenn alle von ihm benötigten Services im System registriert wurden.

Ein derartiger modularer Aufbau erlaubt es verschiedene Bundles nahezu unabhängig von einander zu entwickeln und später einzelne Bundles gegen aktuellere Versionen problemlos auszutauschen. Eclipse SmartHome besteht derzeit aus über 130 solcher untereinander kommunizierender Softwarekomponenten.

Features

Trotz der Positionierung als Grundlage für konkrete Smart Home Lösungen anderer Anbieter, verfügt das Framework über den kompletten Service Stack, der in einem SmartHome zum Einsatz kommt. Es existiert ein Modellgerüst für die virtuelle Abbildung von realen Geräten. Für einige ausgewählte Geräte ist die Steuerung implementiert. Automatische Discovery von Geräten wird begrenzt unterstützt. Eine Rule Engine ist bereits integriert. Schließlich gibt es eine rudimentäre Benutzeroberfläche, die es erlaubt zur Laufzeit Geräte zum System hinzuzufügen und sie zu steuern. ESH hat eine ganze Reihe weiterer Features, wie z.B. Sprachunterstützung, die jedoch im Rahmen dieser Arbeit nicht von Interesse sind.

Zentrales Prinzip des gesamten Frameworks ist es möglichst generische Schnittstellen zu definieren und exemplarische Implementierungen anzubieten, die die beabsichtigte Funktionsweise veranschaulichen.

Im Folgenden werden die verschiedenen Aspekte von ESH, die für die Arbeit relevant sind, erläutert.

3.2 Modell

Die virtuelle Abbildung der realen Geräte im System veranschaulicht Abbildung 3.2.

Things

Things repräsentieren Entitäten, die zum System hinzugefügt werden können. In der Regel handelt es sich hierbei um verschiedene intelligente Geräte, wie beispielsweise eine *Hue* Lampe oder eine Wetterstation. Derartige Geräte bieten eine oder mehrere Funktionalitäten über Channels an.

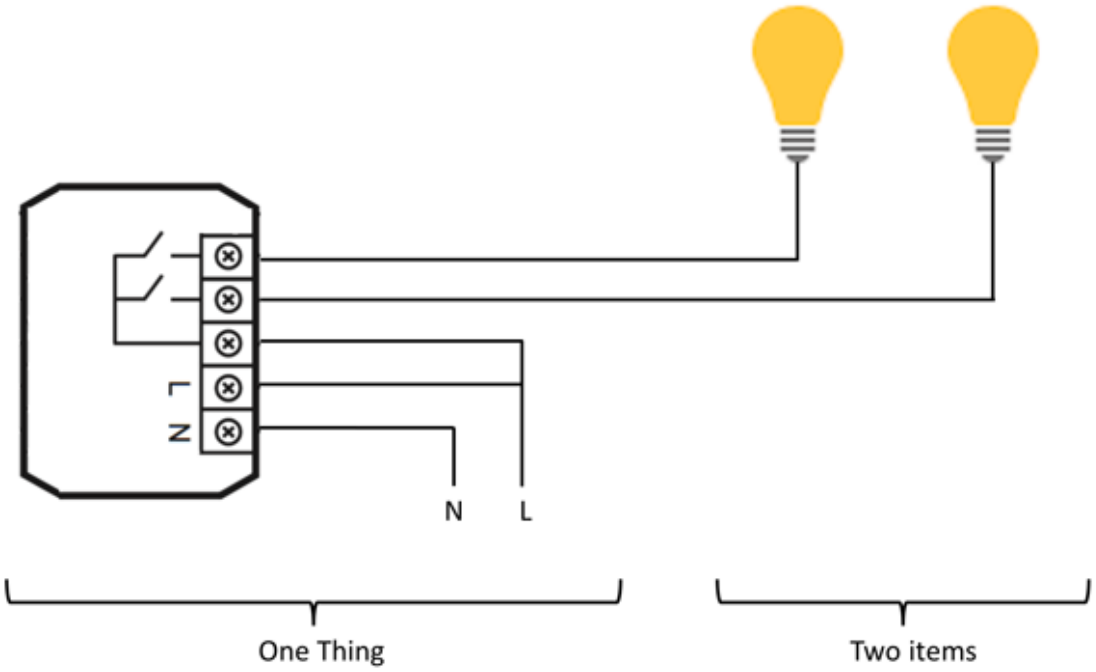

Abbildung 3.2: Architektur von *Things* und *Items* in ESH [2]

Things, die als Brücke zu anderen intelligenten Geräten dienen, werden als Bridges bezeichnet (z.B. *Hue* Bridge);

Channels

Ein Channel stellt eine bestimmte Funktionalität eines Things dar. Things können eine beliebige Anzahl von Channels anbieten. Beispielsweise kann eine Hue Lampe einen einzigen „Licht"-Channel anbieten, während eine Wetterstation die Channels „Temperatur", „Druck" und „Feuchtigkeit"unterstützt.

Items

Items repräsentieren konkrete, detaillierte Funktionalitäten eines Things. Beispielsweise kann ein Thing den Channel „Licht" unterstützen, wobei es 2 verschiedene Items besitzt, die beide jeweils eine physische Glühbirne verkörpern. Items haben einen Zustand, der im System gespeichert wird und können Commands empfangen.

Links

Links verbinden Items mit Things. Jeder Link assoziiert genau einen Channel des Things mit genau einem Item. Erst wenn ein Channel mit mindestens einem Item verbunden ist, gilt er als „aktiviert". Channels und Items können über eine beliebige Anzahl von Links miteinander verbunden werden.

3.2.1 Bindings

Ein Binding ist eine Erweiterung (in der Regel ein eigenständiges Bundle) des Eclipse SmartHome Frameworks, dass für die Integration einer externen Funktionalität in Form von Things verantwortlich ist. In der Regel handelt es sich dabei um verschiedene intelligente Geräte, doch es können auch beispielsweise Webservices angebunden werden.

3.2.2 Automatisierung

3.2.3 Deklarative Typen

Eclipse SmartHome unterstützt die Möglichkeit, konkrete Typen durch die Sm

3.2.4 Persistenz

In Eclipse SmartHome ist eine MapDB bereits integriert. Dabei handelt es sich um eine leichtgewichtige, eingebettete Datenbank, in der zur Laufzeit alle Things, Items und Rules persistiert werden.

3.3 Rule Engine

3.3.1 Triggers

3.3.2 Conditions

3.3.3 Actions

3.3.4 Events

3.4 Controller

3.4.1 Thing Handler

Ein Binding besteht aus einer Reihe von Komponenten. Die grundlegenden Elemente sind im Folgenden aufgeführt:

1. Eine

3.4.2 Trigger/Action Handler

3.5 Fazit

Kapitel 4

Entwurf

In diesem Kapitel wird vorgestellt, wie die neuen Funktionalitäten in ESH integriert werden sollen.

Elemente, die im vorherigen Kapitel vorgestellt wurden, wiederverwenden, zeigen, an welcher Stelle was wie eingebaut wurde.

Kapitel 5

Implementierung

5.1 Überblick

5.2 Bundles

Hier wird erläutert, welche Bundles im Laufe der Implementierung entstanden sind.

5.2.1 tka.binding.twitter

Hier werden die Detatils des Bindings erläutert

Triggers und Events

Actions

5.2.2 tka.binding.dropbox

Triggers und Events

Actions

5.2.3 tka.binding.weather

Triggers und Events

Actions

5.2.4 tka.binding.gmail

Triggers und Events

Actions

5.2.5 tka.flashui

Aufbau der implementierten GUI, sowie ihre Funktionalität.

5.3 Fazit

Kapitel 6

Evaluation

In diesem Kapitel wird die entstandene Implementierung gegen die Anforderungen verglichen und das Ergebnis evaluiert.

6.1 Erfüllte Ziele

Alle formalen Anforderungen, wie sie in Sektion 2.2.2 festgehalten sind, wurden erfüllt. Der Demonstrator läuft on-premise auf dem Raspberry Pi und verwaltet sämtliche Daten lokal. Es ist möglich komplexe Szenarien zu definieren, wobei intelligente Geräte mit Webdiensten frei zusammengefügt werden können. Dies ist möglich, da die integrierten Webservices ebenfalls als *Things* im System abgebildet sind. Schließlich ist eine Benutzeroberfläche vorhanden, die es dem Nutzer erlaubt zur Laufzeit neue Szenarien zu erstellen, zu editieren und zu löschen.

6.2 Vergleich mit Smart Home

Bei dem entwickelten Demonstrator handelt es sich um eine klassische Smart Home Lösung, die auf *Eclipse SmartHome* basiert und um weitere Funktionalitäten angereichert wurde. Dadurch verfügt er über alle üblichen Funktionalitäten, die in einem Smart Home enthalten sind - er ist in der Lage ausgewählte intelligente Geräte direkt anzusteuern und in Szenarien zu automatisieren.

6.3 Vergleich mit webbasierten Task Automation Services

Einen Einblick, wie sich der entstandene Demonstrator in die aktuell existierenden Task Automation Services eingliedern lässt, verleiht Abbildung 6.1. Wie zu sehen ist, gibt es Verbesserungen in vielen Aspekten gegenüber anderen Lösungen. Gegenüber den webbasierten TAS grenzt sich FLASH vor allem durch seine Fähigkeit ab intelligente Geräte

			Web						Smartphone				Home	
		FLASH	Ifttt	Zapier	CloudWork	Elastic.io	ItDuzzit	Wappwolf	On{x}	Tasker	Atooma	AutomateIt	WigWag	Webee
Channels	Web channel support	✓	✓	✓	✓	✓	✓	Few	Few	✓	✓	✓	✓	✓
	Device channel support	✓	Few	×	×	×	×	×	✓	✓	✓	✓	✓	✓
	Smartphone resources as channels	×	✓	×	×	×	×	×	✓	✓	✓	✓	×	×
	Public channels support	✓	✓	×	×	✓	✓	×	✓	×	×	×	×	×
	Pipe channel support	✓	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group channel support	NA	×	×	×	×	×	×	×	×	×	×	Few	×
Rules	Device channel discovery	✓	×	×	×	×	×	×	×	×	×	×	✓	Few
	Multi-event rules	✓	×	×	×	×	×	×	✓	✓	×	×	✓	×
	Multi-action rules	✓	×	×	×	✓	×	×	✓	✓	×	✓	✓	✓
	Chain rules	✓	×	×	×	✓	×	Few	Few	×	×	×	×	×
	Group rules	NA	×	×	×	×	×	×	×	×	×	×	Few	×
	Collision handling	×	×	×	×	×	×	×	×	×	×	×	×	×
	Predefined common rules	✓	×	×	✓	×	✓	✓	×	×	×	✓	✓	✓
	Rule execution profile	DD	WD	WD	WD	WD	WD	WD	DD	DD	DD	DD	DD	DD
TAS	Visual rule editor	×	✓	✓	×	✓	✓	✓	×	✓	✓	✓	✓	✓
	Provides API	✓	×	✓	×	✓	×	×	×	×	×	×	×	×
	Programming language	✓	×	×	×	✓	×	×	✓	Few	×	✓	✓	✓

* ✓ = supported; × = not supported; Few = few support; WD = Web-driven execution profile; and DD = device-driven execution profile; NA = not applicable

Abbildung 6.1: Vergleich des Demonstrators mit existierenden Task Automation Services

direkt anzusteuern, sowie der deutlich mächtigeren Rule Engine ab. Es ist möglich komplexe ECA-Szenarien zu definieren, mit beliebig vielen Triggern, Bedingungen und Befehlen. Teile des Szenarios werden sequentiell abgearbeitet und können als Eingabe für spätere Elemente dienen.

Der entstandene Demonstrator wurde einem umfassenden Test unterzogen, im Laufe dessen Durchsatz und Reaktionszeiten gemessen und mit IFTTT verglichen wurden.

6.3.1 Kopieren von Dateien in der Dropbox

Um Durchsatz zu messen, wurde ein Szenario definiert, dass sämtliche Dateien, die in einen bestimmten Ordner in Dropbox hinzugefügt wurden, in einen anderen Ordner in der Dropbox zu kopieren. Zu Beachten ist, dass IFTTT sich auf die Abarbeitung von bis zu 15 Dateien pro Abfrage begrenzt und Dateien, die größer, als 30 MB sind, nicht beachtet. Zusätzlich wird vom Service gewarnt, dass die Ausführung aller Szenarien sich um bis zu 1 Stunde verzögern kann. Diese Tatsache führt zur Streuung, die in den nachfolgenden Grafiken festzustellen ist.

Der Demonstrator lief auf einem Raspberry Pi 3 Model B, der an einen Router mit VDSL (50 MBit) Verbindung angeschlossen war.

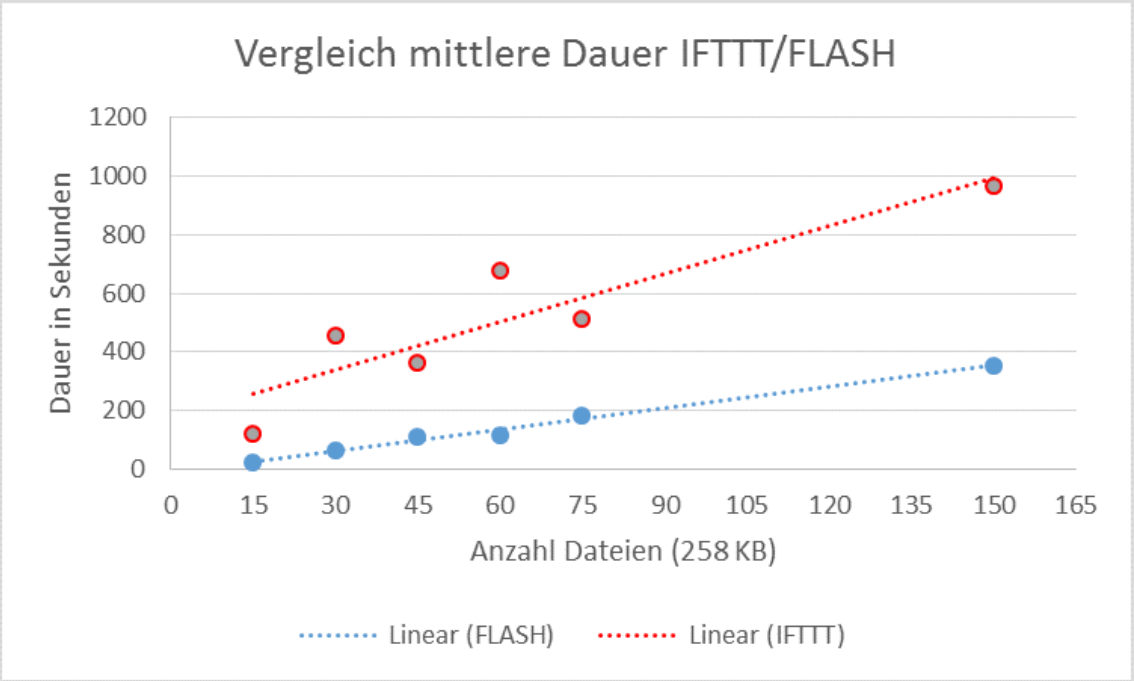

Abbildung 6.2: Vergleich des Mittelwerts zwischen IFTTT und FLASH für unterschiedliche Dateimengen

Es wurde geprüft, wie gut die beiden Anwendungen mit einer großer Anzahl kleiner Dateien, sowie mit geringer Anzahl von großen Dateien umgehen können.

Zahlreiche kleine Dateien

Es wurden Mengen von 258 KB großen Dateien gleichzeitig in einen Dropbox Ordner hochgeladen. Es wurden jeweils Vielfache von 15 verwendet, da dies die maximale Anzahl von Dateien ist, die IFTTT in einer Abfrage bearbeitet.

Es entstanden die Grafiken 6.2 und 6.3.

Für die Erstellung von Grafik 6.2 wurde für jede hochgeladene Datei einzeln gemessen, wie viele Sekunden es dauert, bis eine Kopie im Zielordner vorhanden ist. Es wurde anschließend ein Mittelwert gebildet und die Messung mehrmals wiederholt. Schließlich wurde ein gemeinsamer Mittelwert über die gesammelten Werte gebildet und in der Grafik für die verschiedenen Mengen von Dateien dargestellt.

In der Grafik 6.3 wurde analog gehandelt, mit dem Unterschied, dass hier dargestellt ist, was die längste Dauer zwischen hochladen einer Datei und der Bereitstellung der Kopie ist. Da alle Dateien nahezu gleichzeitig hochgeladen werden, kann man diesen Wert auch als Gesamtdauer, bis alle Dateien verschoben wurden, betrachten.

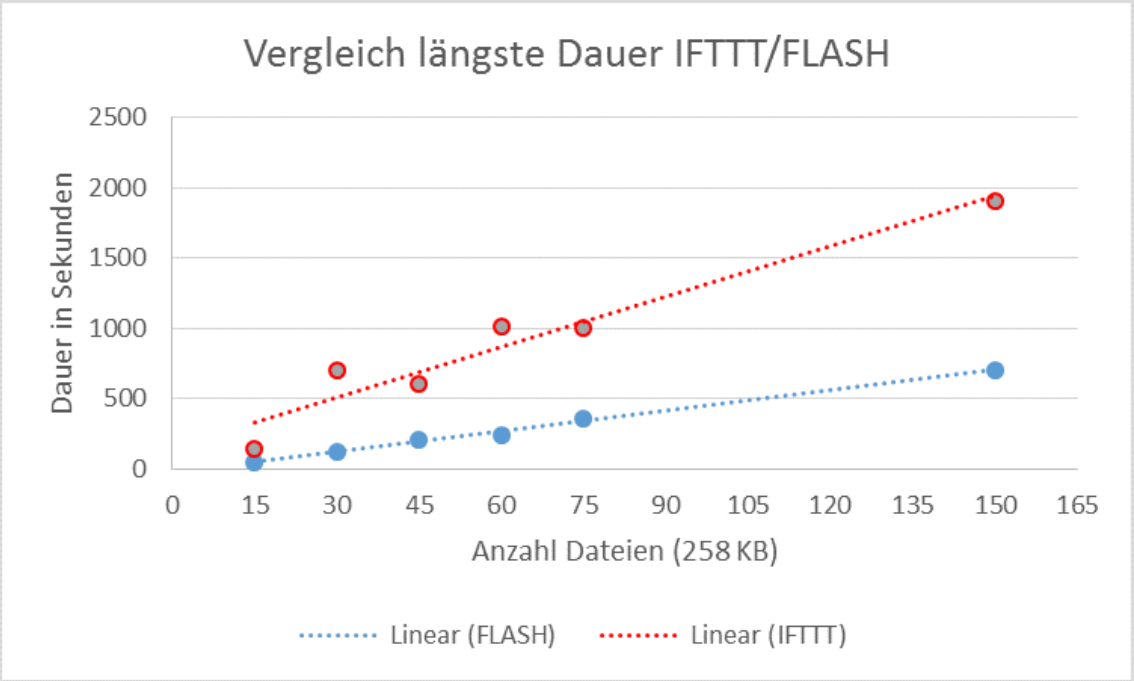

Abbildung 6.3: Vergleich der längsten Dauer (equivalent zu Gesamtdauer, da ca. 1-2 Sekunden vernachlässigt werden können) zwischen IFTTT und FLASH für unterschiedliche Dateimengen

Große Dateien

IFTTT ist in der Lage mit Dateien umzugehen, die bis zu 30 MB groß sind. Eine Wiederholung der oben genannten Messung mit 25 MB großen Dateien hat ergeben, dass sich die Ergebnisse analog verhalten.

Der Demonstrator ist in der Lage mit größeren Dateien umzugehen. Im Test gab es auch bei Verschiebung von 300 MB großen Dateien keinerlei Probleme.

6.3.2 Steuern von Lampen bei Tweets

Es wurde sowohl auf IFTTT als auch auf dem Demonstrator ein Szenario erstellt, dass bei jedem Tweet eine Philips Hue Lampe ausschaltet. Anschließend wurde gemessen, wie lange es dauert, bis die Lampe tatsächlich ausgeschaltet wird, nachdem der Tweet veröffentlicht wurde. Dabei steuert ITTTT die Lampe über die von Hue bereitgestellte Webschnittstelle an, während der Demonstrator das Gerät über WLAN bedient.

Es hat sich ergeben, dass die Reaktionszeit des Demonstrators stets unter einer Sekunde lag, während die Schaltung durch IFTTT eine sehr variable Dauer aufwies. Die schnellste Reaktionszeit lag bei ungefähr 10 Sekunden.

6.3.3 Auswertung

Vorteile

Aus den Messungen hat sich gezeigt, dass der Demonstrator IFTTT in vieler Hinsicht deutlich voraus ist. Dadurch, dass er sich im Hause des Nutzers befindet, ist er in der Lage mit Geräten direkt zu kommunizieren, was Grund für merklich bessere Reaktionszeiten in Szenarien, die intelligente Geräte im Haus betreffen, ist.

Auch in Szenarien, die nur Web-basiert sind und viel Download/Upload-Volumen mit sich ziehen, hat sich der Raspberry Pi als überlegen erwiesen. Dies lässt sich dadurch erklären, dass es sich bei dem Demonstrator im Gegensatz zu IFTTT um ein dediziertes Gerät handelt, dass nur einen konkreten User bedient. Dies führt auch dazu, dass er keine arbiträren Eingrenzungen besitzt, wie z.B. die Begrenzung der Dateigröße.

Schließlich ist der Raspberry Pi aus einer Datenschutzperspektive dem Cloud-Service zu bevorzugen, da sämtliche Zugriffsdaten nur lokal auf dem Gerät gesichert werden.

Nachteile

Negative Aspekte des Demonstrators sind Konsequenz der Tatsache, dass es sich dabei um eine on-premise Lösung handelt. Sämtliche Szenarien, die Webdienste automatisieren, funktionieren nur solange eine Internetverbindung vorhanden ist. Sollte sie ausfallen, bleiben nur die charakteristischen Smart Home Funktionalitäten erhalten. In diesem Aspekt ist IFTTT dem Demonstrator voraus, da die Ausfallsicherheit in einem Rechenzentrum gegenüber der eines herkömmlichen Heimanschlusses weit überlegen ist.

Die Tatsache, dass sämtliche Zugriffsdaten auf Accounts des Users ausschließlich lokal auf dem Gerät vorhanden sind, zieht mit sich, dass im Falle eines Ausfalls (z.B. Hardware-Fehler) sämtliche Daten verloren gehen können. Dies könnte dazu führen, dass sämtliche Geräte, Dienste und Szenarien nach einer Reparatur komplett neu aufgesetzt werden müssten.

6.4 Eclipse SmartHome und Quellcode

6.4.1 Positive Aspekte

Im Laufe der Arbeit is klar geworden, dass das Eclipse SmartHome Framework sich gut als Grundgerüst für unterschiedlichste Lösungen handelt. Die sehr generische Struktur des Frameworks hat es ermöglicht, die im Rahmen der Arbeit implementierten Funktionalitäten nahtlos in das Gesamtkonstrukt zu integrieren.

Dies, sowie OSGi im Allgemeinen, ermöglicht es ebenfalls, weitere Funktionalitäten (z.B. weitere Webdienste) der Anwendung hinzuzufügen , ohne bereits existierenden Code ändern zu müssen. Dank der Option, Szenarien im JSON-Format zu definieren, ist es sogar begrenzt möglich, neu hinzugefügte Webdienste in Regeln zu automatisieren, ohne die Benutzeroberfläche anzupassen.

6.4.2 Negative Aspekte

Eclipse SmartHome

Negativ aufgefallen ist die mangelnde Dokumentation von ESH, die den Einstieg in das Framework deutlich erschwert. Verstehen der einzelnen Funktionalitäten bedarf der Betrachtung des Quellcodes, wobei relevante Stellen unter den mehr als hundert Bundles erst noch identifiziert werden müssen.

Eclipse SmartHome befindet sich derzeit noch auf Version 0.9.0 und wird aktiv weiterentwickelt. Dies hat unvermeidbar dazu geführt, dass an einigen Stellen der Code nicht fehlerfrei ist. Im Laufe der Implementierung mussten an mehreren Stellen Bugs zunächst ausfindig gemacht und danach behoben werden.

Quellcode

Vor allem der Mangel eines visuellen Regeleditors begrenzt derzeit die Nutzungsmöglichkeiten des Demonstrators. Zwar ist es möglich, Szenarien zur Laufzeit zu editieren, so bedarf dies dennoch Kenntnissen von JSON, sowie den existierenden Event-Typen, sowie der Eingaben und Ausgaben von Triggern, Bedingungen und Aktionen.

6.5 Fazit

Die Implementierung erfüllt alle explizit definierten Anforderungen an die Funktionsweise und weist deutliche Verbesserungen in wesentlichen Aspekten gegenüber der Konkurrenz auf. Sie lässt sich leicht um neue Elemente und Funktionalitäten erweitern.

Kapitel 7

Zusammenfassung

7.1 Zusammenfassung

Hier eine Zusammenfassung

7.2 Ausblick

Hier ein Ausblick

Anhang A

Weitere Informationen

Der Quellcode der Implementierung befindet sich auf der mit der Arbeit abgegebenen CD.

Abbildungsverzeichnis

Literaturverzeichnis

[1] *Dropbox*. Available online at `https://www.dropbox.com/`; visited on February 6th, 2017.

[2] *Eclipse SmartHome Homepage*. Available online at `https://www.eclipse.org/smarthome/`; visited on February 6th, 2017.

[3] *OpenWeatherMap*. Available online at `https://openweathermap.org/api`; visited on February 6th, 2017.

[4] *RWE SmartHome*. Available online at `http://www.rwe-smarthome.de/`; visited on February 6th, 2017.

[5] *Twitter*. Available online at `https://twitter.com/`; visited on February 6th, 2017.

[6] *IFTTT*. Website, 2017. Available online at `http://ifttt.com`; visited on February 6th, 2017.

[7] *The OSGi Architecture*. Website, 2017. Available online at `https://www.osgi.org/developer/architecture/`; visited on February 6th, 2017.

[8] *Zapier*. Website, 2017. Available online at `http://zapier.com/`; visited on February 6th, 2017.

[9] Beer, Wolfgang, Volker Christian, Alois Ferscha und Lars Mehrmann: *Modeling Context-Aware Behavior by Interpreted ECA Rules*, Seiten 1064–1073. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[10] Bhadauria, Rohit und Sugata Sanyal: *Survey on Security Issues in Cloud Computing and Associated Mitigation Techniques*. CoRR, abs/1204.0764, 2012.

[11] Coronado, M. und C. A. Iglesias: *Task Automation Services: Automation for the Masses*. IEEE Internet Computing, 20(1):52–58, Jan 2016.

[12] European Research Cluster on the Internet of Things: *IERC Projects Portfolio*. https://www.smart-action.eu/publications/detail/114/90c9734fda7c5c2c68e631bf29a9a9d2/.

[13] Jacobs, Tobias, Markus Joos, Carsten Magerkurth et al.: *Adaptive, faulttolerant orchestration of distributed IoT service interactions.* Technischer Bericht D2.5, The Internet of Things - Architecture, 2012.

[14] Menoret, Stephane et al.: *iCore - Final architecture reference model.* Technischer Bericht D2.5, iCore Project, 2014.

[15] Ran, Chen, Baoan Li und Jianjun Yu: *CEIS 2011 Research and Application on the Smart Home Based on Component Technologies and Internet of Things.* Procedia Engineering, 15:2087 – 2092, 2011.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 7. Februar 2017

Konstantin Tkachuk