UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR INGENIEURWISSENSCHAFTEN
ABTEILUNG INFORMATIK UND ANGEWANDTE KOGNITIONSWISSENSCHAFT

Masterarbeit

Implementation und Evaluation von Longevity Digital Twins im Kontext von Smart Homes

Malte Singerhoff
Matrikelnummer: 3022557
Angewandte Informatik (Master)

Fachgebiet Verteilte Systeme, Abteilung Informatik
Fakultät für Ingenieurwissenschaften
Universität Duisburg-Essen

13. April 2023

Erstgutachter: Prof. Dr.-Ing. Torben Weis
Zweitgutachter: Prof. Dr. Gregor Schiele
Zeitraum: 16. September 2022 - 14. April 2023

Abstract

Die größer werdende Komplexität in Smart Homes und die große Anzahl an Standards von Smart Home Geräten führt zu einer immer schwierigeren Analysierbarkeit von Smart Home Systemen. Um dieses Problem zu lösen, wurde der *Longevity Digital Twin(LDT)* entworfen, der Smart Homes und *Digitial Twins(DT)* in Verbindung bringt und eine bessere Analysierbarkeit ermöglicht.
Diese Arbeit greift das Konzept des *LDT* auf und implementiert eine entsprechende Softwarelösung. Die Datenstruktur, die für den *LDT* entworfen wird, basiert auf der *Web of Things Thing Description(WoT-TD)* und in dieser Arbeit zeigt sich, dass die *WoT-TD* ein geeignetes Datenmodell für Smart Home Geräte ist. Die *WoT-TD* ermöglicht es durch *Interaction Affordances* genaue funktionale Beschreibungen von Smart Home Geräten zu erstellen und ihre Abhängigkeiten darzustellen. Diese Abhängigkeiten werden durch den *LDT* als Graph modelliert und dargestellt. Analyses der *WoT-TD* im Kontext unterschiedlicher Smart Home Protokolle, wie *Matter* und *KomeKit* zeigen, dass es möglich ist für jedes Smart Home Gerät ein entsprechendes Modell zu generieren.
Es werden *Bootstrapping* Methoden analysiert und angewandt, sodass der *LDT* in der Lage ist Datenmodelle für Geräte zu erstellen, mit denen er in direkter Verbindung steht, aber auch Datenmodelle für Geräte zu erstellen, mit denen er nicht in direkter Verbindung steht, indem der *LDT* die Daten von einem Smart Home Server abruft und dann die Modelle erstellt.
Tests der Funktionen des *LDTs*, für die beiden Arten an Smart Home Geräten als *HomeKit* Geräte auf einem *ESP32* implementiert und ein *Home Assistant* Smart Home Server benutzt werden, zeigen, dass ein *LDT* Gerät zwar eine schlechtere Performance aufweist, als ein nicht *LDT* Gerät, aber im Gegenzug auch, dass der *LDT* durch den entworfenen Graphen und der *WoT-TD* eine genauere Darstellung der Geräte erlaubt. Weitere Tests zeigen, dass der *WoT-TD* unterschiedliche Smart Home Geräte modellieren kann und diese Modelle sich im Speicherverbrauch kaum unterscheiden.
Diese Arbeit zeigt eine mögliche Implementierung eines *LDT* und die Einflüsse, die die große Anzahl an Standards und miteinander interagierenden Systemen auf diese haben. Es wird aber auch bewiesen, dass der *LDT* eine genormte Darstellung eines jeden in dieser Arbeit benutzen Standards ermöglicht und Vergleiche zwischen Smart Home Geräte unterschiedlicher Hersteller und Protokolle ermöglicht.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Der Markt für Smart Home Geräte ist in der momentanen Zeit und auch in der Zukunft ein immer größer werdendes und lukratives Geschäft. Die Verringerung der Kosten von Sensoren und Mikrocontroller führen dazu, dass für den Smart Home Markt immer mehr Produkte entwickelt werden. Dieses Interesse an dem Smart Home Markt bewirkt, dass die Erwartungen an Smart Home Geräte immer weiter wachsen. Ein Smart Home Gerät ist mittlerweile nicht nur dafür zuständig, das Leben angenehmer zu machen, sondern es soll auch für Effizienz und Sicherheit im Alltag sorgen[14]. Die Vorteile eines *Smart Homes* sind dann am effektivsten, wenn eine Vielzahl an Smart Home Geräten benutzt werden. Zum Beispiel sind Smart Home Sicherheitskameras am effektivsten, wenn neben den Kameras auch eine Vielzahl an Sensoren benutzt werden, um einen genauen Überblick darüber zu schaffen, ob ein Sicherheitsrisiko zu erkennen ist[14]. Der Markt für Smart Home Geräte geht über Komfort, bis hin zu Sicherheit und der Unterstützung von älteren Menschen und Menschen mit Behinderung[14].
Allerdings hat ein Smart Home auch Nachteile, die vor allem daran gebunden sind, dass es auf dem Smart Home Markt eine Vielzahl an Herstellern gibt, die ihr eigenes Smart Home System, inklusive eigener Protokolle, anbieten. Diese Systeme sind beispielsweise *HomeKit* von *Apple* oder das *Matter* Protokoll, das von einer Vielzahl an Herstellern wie *Google* und *Amazon* erstellt wurde [25]. Diese Abhängigkeit von den Herstellern und ihren Systemen führt dazu, dass die Wartung eines Systems verkompliziert wird und so Software-Updates eines Gerätes andere Geräte negativ beeinflussen können [46].
Um dieser Problematik entgegenzuwirken, ist es notwendig eine Architektur zu schaffen, die es ermöglicht diese Vielzahl an unterschiedlichen Produkten und ihre Protokolle zu organisieren. In diesem Zusammenhang wurde von Peter Zdankin et. al der *Longevity Digital Twin* entworfen [46]. *Digital Twins* sind digitale Abbilder von physischen Objekten und ermöglichen es eine Vielzahl an unterschiedlichen Analysen eines Produkts durchzuführen. Dazu gehören beispielsweise die Simulation von Prototypen bei dem Entwurf und Bau eines Produkts, aber auch Simulationen zur Laufzeit eines Produkts wie etwa die Einschätzung, ob ein Status momentan zu einem Fehler führen kann oder wie

zukünftige Konfigurationen das System negativ beeinflussen[41].
Der *Longevity Digital Twin* greift gerade die Konzepte eines *Digital Twins* auf, welche zur Laufzeit ein System analysierbar machen und bringt diese Konzepte in den Kontext eines Smart Home Systems. Durch diese Architektur soll es ermöglicht werden verschiedene Smart Home Geräte an einem zentralen Ort Protokoll unabhängig miteinander zu analysieren. Dabei wird dann über die von dem *Digital Twin* gesammelten Daten ein Abhängigkeitsgraf erstellt, der es ermöglicht zu analysieren, welche Geräte abhängig voneinander sind. Die Hoffnung dieser Architektur ist es, Smart Home Geräte langlebiger zu machen, da durch die Analyse der Geräte Fehler rechtzeitig erkannt werden und beispielsweise Softwareupdates auf Nebenwirkungen analysiert werden können[46].

1.2 Vorgehen bei der Implementierung

In dieser Arbeit wird eine auf die *Longevity Digital Twin(LDT)* Architektur basierte Softwarelösung entworfen, implementiert und evaluiert. Dabei werden als Erstes die Kernprobleme formuliert, die gelöst werden müssen. Das erste Problem ist es ein Datenmodell zu erstellen, das es ermöglicht eindeutig unterschiedliche Smart Home Geräte zu modellieren, um diese im *LDT* analysieren zu können. Dafür werden Anforderungen aufgestellt, die sich aus den Definitionen von einem *Digital Twin(DT)* und aus der Architektur des *LDT* ableiten lassen. Es wird analysiert, inwiefern die *Web of Things Thing Description(WoT-TD)* als Modell für einen *LDT* benutzt werden kann und es wird beispielhaft analysiert, wie sich die *WoT-TD* Repräsentation von Smart Home Geräten unterschieden. Dazu wird eine Smart Home Glühbirne als *Matter Light, HomeKit Lightbulb* und *HomeAssistant Light* in der *WoT-TD* modelliert und verglichen. Nach der zufriedenstellenden Analyse wird sich entschieden, das *WoT-TD* als Datenmodell für die Implementation zu benutzen.
Ein weiteres Problem, das in dieser Arbeit beleuchtet wird, ist das Bootstrapping des *LDT*. Dabei wird sich vor allem darauf fokussiert zu untersuchen, wie der *LDT* Daten eines Smart Home Gerätes erhält, um ein Datenmodell zu generieren. Dafür werden drei Klassen an Gerätearten definiert, die je beschreiben, wie ein *LDT* an die Daten kommt.

- Ein *LDT* Gerät steht direkt mit dem *LDT* in Verbindung. Dieses Gerät muss spezielle Software ausführen, die es erlaubt, direkt mit einem *LDT* zu kommunizieren und Daten auszutauschen.
- Ein zentrales Smart Home Gerät steht mit einem Smart Home Server in Verbindung. Das Gerät, steht nicht mit dem *LDT* in direkter Verbindung.
- Ein dezentrales Gerät steht mit keinem anderen Gerät in Verbindung, hostet aber eine Schnittstelle, die es erlaubt auf die Daten des Gerätes zuzugreifen.

Für die erste Klasse wird ein *LDT* Gerät implementiert, das ein *HomeKi*t Gerät erweitert und in direkter Verbindung mit dem *LDT* steht, damit der *LDT* dieses modellieren kann. Für die zweite Klasse wird einerseits ein klassisches *HomeKit* Gerät ohne *LDT* Funktionalität implementiert und es wird die Funktion implementiert registrierte Geräte eines *HomeAssistant* Webservers abzurufen und zu analysieren. Die dritte Klasse wird nicht implementiert, da diese zu Implementation abhängig ist.
Der letzte Aspekt, der untersucht wird, ist es mithilfe des *LDT* Abhängigkeiten darzustellen. Der *LDT* ist in der Lage aufgrund des größeren Wissens, das er über die Geräte hat, Abhängigkeiten darzustellen. Diese Abhängigkeiten werden direkt in den *WoT-TD* Modellen moduliert und als Graph über einen Webserver dem Nutzer sichtbar gemacht.
Die Implementation wird evaluiert. Dabei wird einerseits verglichen, wie sehr sich ein *LDT* Gerät und ein zentrales Gerät in den Speicheranforderungen und der Performance unterscheiden. Es wird außerdem ein beispielhaftes Modell generiert, das als Graph anzeigt, welche Abhängigkeiten in einem Smart Home vorhanden sind und es wird analysiert, wie der Speicherverbrauch der *LDT* Modelle skaliert.

1.3 Aufbau der Arbeit

Die Arbeit umfasst sieben Kapitel. Das Einführungskapitel bietet eine Motivation und ein Anwendungsbeispiel der Implementierung eines *Longevity Digital Twin*.
Im zweiten Kapitel werden die Grundlagen aufbereitet, die für das Verständnis dieser Arbeit benötigt werden. Bei diesen Grundlagen handelt es sich einerseits um eine Übersicht von Smart Home Geräten und dem Aufbau eines Smart Home Systems. Es werden mit *HomeKit* und *Matter* zwei Smart Home Systeme vorgestellt und es wird im Detail betrachtet, wie das Datenmodell eines *HomeKit* und *Matter* Gerätes aussieht. Es wird das Smart Home Konzept in den Kontext des *Internet of Things(IoT)* gebracht und anschließend wird auf das *Web of Things* und die *Thing Description* eingegangen, um ein Hersteller unabhängiges Datenmodell für *IoT* Geräte vorzustellen.
Im dritten Kapitel werden zwei Arbeiten vorgestellt, die sich mit ähnlichen Problemen beschäftigen, die auch bei der Implementierung eines *LDT* gelöst werden müssen. Die Arbeit *Internet of Things Patterns of Device bootstrapping and Registration* von Lukas Reinfurt et. al. [39] beschäftigt sich mit unterschiedlichen Bootstrapping Mechanismen im IoT und es wird analysiert, welche dieser *Bootstrapping* Mechanismen für ein *LDT* infrage kommen, um die Kommunikation zu starten und den Startpunkt für die Modellgenerierung zu bestimmen. Die Arbeit *IoTSanitizer* von Nguyen et. al. [36] beschäftigt sich auch wie der *LDT* darum, wie ein Smart Home System auf Fehler analysiert werden kann, aber anstatt einen *Digital Twin* zu benutzen wird durch *Model Checking*, das auf den Programmcode von *Samsung Smart Apps* angewandt wird, die Fehleranalyse durchgeführt.
Im vierten Kapitel werden die Problemstellungen aufgestellt, die bei der Implementierung des *LDT* zu lösen sind. Dabei werden die Probleme in zwei Gruppen strukturiert.

Die eine Gruppe von Problemen beschäftigt sich mit dem internen Aufbau des *LDT*, wie zum Beispiel die Anforderungen an das Datenmodell, um Smart Home Geräte zu modellieren. Die zweite Gruppe von Problemen sind alle Probleme, die durch die Kommunikationspartner des *LDTs* entstehen. Dabei wird auf die Art an Kommunikationspartner eingegangen und welche Aufgabe die Kommunikation mit diesen Partnern hat.
Im fünften Kapitel werden Lösungen für die aufgestellten Probleme aufgestellt. Es wird argumentiert und beispielhaft bewiesen, dass sich *HomeKit* und *Matter* Geräte als *Web of Things Thing Descriptions* modellieren lassen. Es wird ein *Dependency* Graph entworfen, der auf dem Datenmodell der *Web of Things Thing Description* aufbaut, es werden Lösungsansätze zwischen den verschiedenen Kommunikationen aufgestellt und es wird aufgezeigt, wie ein Smart Home Gerät auf ein *ESP32* implementiert werden kann.
Im fünften Kapitel werden zu jedem Problem, zu dem eine Lösung implementiert wurde, kurz zusammengefasst, wie diese gelöst wurden und es werden Beispiele gegeben.
Im sechsten Kapitel wird der *LDT* in zwei Aspekten untersucht. Einerseits wird untersucht, wie viel Auswirkung der *LDT* auf Laufzeitperformance und Speicherverbrauch eines Smart Home Gerätes hat und andererseits wird untersucht, wie gut sich ein *LDT* skalieren lässt und es wird ein Beispiel angegeben, wie ein Graph von einem *LDT* gebaut werden kann, um ein komplexeres Smart Home zu analysieren.
Im sechsten Kapitel wird eine Bewertung gegeben, inwiefern die Implementation ein Erfolg war und es wird eine Übersicht gegeben, welche Probleme nicht beachtet wurden oder noch gelöst werden müssen. Anschließend wird ein Ausblick gegeben, wie ein *LDT* in der Zukunft noch erweitert werden kann.

Kapitel 2

Grundlagen

In dem Grundlagenkapitel wird eine Übersicht über die die Funktionsweise eines Smart Home Systems gegeben und es werden am Beispiel von dem *HomeKit Accessory Protocol(HAP)* und *Matter*, zwei Industriestandards vorgestellt, mit denen Smart Home Geräte entwickelt werden können. Es wird eine praxisorientierte Definition eines *Digital Twins(DT)* gegeben und der *Longevity Digital Twin(LDT)* vorgestellt. Es wird argumentiert, wieso es sich bei dem *LDT* um einen praktischen *DT* handelt. Außerdem werden die relevantesten Bausteine der *Web of Things Thing Description(WoT-TD)* vorgestellt.

2.1 Smart Home Grundlagen

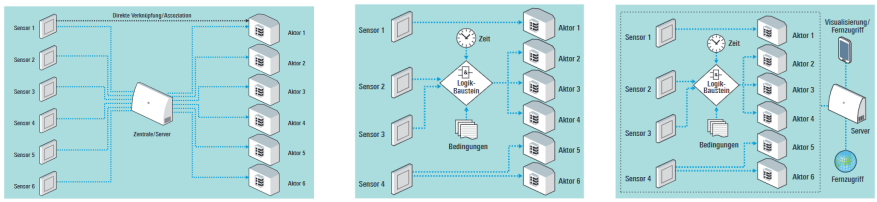
In diesem Kapitel werden die grundlegenden Arten von Smart Home Geräten vorgestellt und es wird eine Übersicht über unterschiedliche Aufbauten von Smart Home Systemen gegeben.

2.1.1 Smart Home Geräte

Smart Home Geräte sind der Kernbestandteil eines Smart Home Systems. Diese können in zwei Gruppen aufgeteilt werden: Sensoren und Aktoren, auch Aktuator. Sensoren messen die Umgebung und geben diese Messwerte an das System weiter. Ein möglicher Sensor wäre ein Lichtsensor. Aktoren sind die ausführenden Komponenten eines Smart Homes und sie beeinflussen die Umgebung. Mögliche Aktoren sind ein Motor oder eine Glühbirne. Damit Smart Home-Geräte miteinander kommunizieren können, werden diese entweder direkt miteinander verbunden oder es werden drahtlose Kommunikationsprotokolle benutzt [25].

2.1.2 Aufbau eines Smart Home Systems

Die Smart Home Geräte in einem Smart Home sind in der Regel Teil eines Netzwerks, das alle Geräte miteinander verbindet. Es lassen sich drei generelle Aufbauten eines Smart Home Systems formulieren, die die Logik des Systems entweder im Server, in den Smart Home Geräten oder in beidem ausführen.



(a) (b) (c)

Abbildung 2.1: (a) Ein zentrales System bei dem ein Smart Home Server die zentrale Komponente ist[25] (b) Ein dezentrales System bei dem es keine zentrale Komponente gibt [25] (c) Eine Mischung aus beiden Systemen [25]

1. Bei einem zentralen System, dargestellt in Fig. 2.1(a), liegt die Logik des Smart Homes in dem Server. Sensoren und Aktoren haben eine minimale Logik und der Server berechnet bei eingehenden Sensordaten, welche Aktionen ausgeführt werden müssen und führt bei Bedarf die nötigen Aktoren aus. Die Vorteile dieses Systems sind, dass der Server Wissen über alle Aktionen des Systems hat und der Preis pro Gerät ist geringer, da die Geräte weniger komplex sind. Der Nachteil eines solchen Systems ist die fehlende Redundanz, da bei einem Ausfall des Servers das ganze System in der Funktionalität beeinträchtigt wird und im schlimmsten Fall komplett ausfällt[25].

2. Ein dezentrales, illustriert in Fig. 2.1(b), besteht aus Sensoren und Aktoren, deren Verhalten vorprogrammiert ist und unabhängig von einem Server auf Umgebungseinflüsse reagieren können. Der Vorteil des Systems ist eine geringere Fehleranfälligkeit. Die Nachteile sind ein höherer Preis und eine höhere Komplexität bei der Wartung des Systems, da es kein zentrales Steuerelement gibt, um alle Geräte gleichzeitig zu überwachen. Es muss also im schlimmsten Fall bei einem Fehler jedes einzelne Gerät kontrolliert werden, um den Ursprung des Fehlers zu finden[25].

3. Der dritte Aufbau für ein Smart Home ist eine Kombination aus beiden vorher genannten Systemen. Dieser Aufbau, zu sehen in Fig. 2.1(c), besteht aus einem zentralen Server und aus intelligenten und unintelligenten Sensoren und Aktoren. Dadurch können die Vorteile beider Systeme ausgenutzt werden und die Nachteile reduziert werden. Es können bei komplexen und kritischen Abläufen intelligente

Sensoren und Aktoren benutzt werden, um zu gewährleisten, dass diese auch beim Ausfall des Servers funktionieren. Gleichzeitig können einfachere Aufgaben von günstigen Sensoren und Aktoren ausgeführt werden. Ein Server steht mit allen Geräten in Verbindung und es kann ein genauer Überblick über den momentanen Zustand von dem System dargestellt werden. Falls der Server ausfällt, können die Automationen zwischen komplexeren Smart Home Geräte weiter ausgeführt werden [25].

2.1.3 Erweiterung durch eine Cloud

Um die Rechenleistung eines Smart Home Systems zu erhöhen, ist es möglich, dieses mit einer Cloud zu verbinden. Dadurch werden Berechnungen, die für den lokalen Server zu viele Ressourcen kosten, in die Cloud ausgelagert. Um diese Option zu benutzen, muss das System Zugriff auf das Internet haben[25].

2.1.4 Smart Home im Internet of Things

Das *Internet of Things* lässt sich als internetbasiertes Netzwerk beschreiben, dessen Teilnehmer hauptsächlich aus Geräten bestehen, die nicht die Funktionalität eines klassischen PCs liefern[42]. Durch die immer größere Funktionalität, die Smart Home Geräte mit sich bringen und ihre Fähigkeit sich selbstständig mit kabellosen Netzwerken zu verbinden, ist das Smart Home Konzept ein Teil des *Internet of Things*[40]. Das bedeutet, dass sich auf Smart Homes, *IoT* Konzepte anwenden lassen, wie das in 2.4 vorgestellte *Web of Things*.

2.2 Smart Home Protokolle und Standards

Im Smart Home Bereich gibt es eine Vielzahl an Protokollen und Standards[25]. Dadurch, dass die Kommunikation zwischen den Smart Home Geräten zueinander und des Servers meistens kabellos funktioniert, gibt es eine Vielzahl an unterschiedlichen Funkprotokollen, die alle ihre Vor- und Nachteile haben. Vorteile eines Funkstandards können geringere Kosten und geringerer Strombedarf sein, während Nachteile extra benötigte Hardware oder eine geringe Reichweite sind [25]. In dieser Arbeit wird hauptsächlich davon ausgegangen, dass es sich bei den Smart Home Geräten auch um IoT fähige Geräte handelt und das *Internet Protokoll(IP)* das benutzte Kommunikationsmittel ist. Das hat den praktischen Grund, dass Smart Home Geräte, die *IP* benutzen, einfach in jedes Wi-Fi-Heimnetzwerk integrierbar sind und keine weitere Hardware benötigt wird[30].
Smart Home Geräte benutzen in der Regel den entsprechenden Standard für das von

dem Hersteller entworfene Smart Home System, um die Kommunikation der Systemteilnehmer zu organisieren[25]. Es gibt eine Vielzahl an Herstellern, die ihren eigenen Standard veröffentlichen. Beispiele dafür sind *Amazon*, *Google* und *Apple*. Der Hersteller *Amazon* bietet mit dem *Alexa Skill Kit* [1] einen Standard, um cloudbasierte Geräte mit dem *Alexa* Sprachcomputer zu kontrollieren. *Apple* hat mit dem *HomeKit* und der *HomeApp* ein eigenes System [29] und der offizielle Standard von *Google* ist der *Matter* Standard in Kombination mit dem *Google Home* System[23].
In diesem Kapitel wird *HomeKit* und das zugehörige *HomeKit Accessory Protocol(HAP)* vorgestellt, da es eine Vielzahl an Open Source Implementationen gibt, die das *HAP* implementieren, wie zum Beispiel *HomeSpan*[27] und die Geräte auch ohne Cloud Anbindung voll funktionsfähig sein können. Es wird außerdem auch auf den *Matter* Standard eingegangen, da dieser nicht nur von *Google*, sondern von über 30 weiteren Unternehmen unterstützt wird. Zu diesen Unternehmen gehören unter anderem auch *Apple* und *Amazon*[45].

2.2.1 HomeKit und HomeKit Accessory Protocol(HAP)

Der Hersteller *Apple* bietet mit unterschiedlichen Smart Home Geräten wie Glühbirnen und Thermostaten eine Vielzahl von intelligenten Geräten für ein Smart Home an[29]. Dazu bieten *Apple* mit der Softwarelösung *HomeKit* und das damit verbundene *HomeKit Accesory Protocol(HAP)* eine Möglichkeit, Smart-Devices zu beschreiben und mit einem Server zu verbinden, der *HomeKit* ausführt. Das *HAP* ist in der nicht-kommerziellen Version frei verfügbar und enthält alle nötigen Informationen, um ein *HomeKit* kompatibles Gerät zu konfigurieren[26].
Der Hauptbestandteil des *HAP* sind *Accessoires*, dies sind Identitäten, die aus mehreren *Services* und *Characteristics* bestehen können. Jeder von Apple vordefinierte *Service* besitzt mindestens eine verpflichtende *Characteristic* und unterschiedlich viele optionale *Characteristics*. Ein Beispiel für ein *Accessory* bestehend aus zwei *Services* wäre zum Beispiel eine Glühbirne mit einem Lichtsensor. Dieses *Accessory* würde dann aus dem *Service, Lightbulb* mit der verpflichtenden *Characteristic, On* und dem *Service, Light Sensor* mit der *Characteristic, Current Ambient Light Level* bestehen. Damit die Glühbirne auch die Lichtstärke regulieren kann, müsste die *Characteristic, Brightness* zum *Lightbulb Service* hinzugefügt werden. *Services* und *Characteristics* besitzen ein *Universally Unique Identifier(UUID)* und einen *Type*, um sie eindeutig identifizieren zu können. *Characteristic* haben außerdem noch weitere Definitionen, abhängig von ihrem Datenformat. Diese Definitionen beschreiben genau, wie die Werte einer *Characteristic* zu verstehen sind. Zum Beispiel ist der Wert von einer *Brightness Characteristic* als Prozentwert definiert, der als *integer*, mit Werten von 0 bis 100, abgespeichert wird. Des Weiteren werden noch weitere Parameter einer *Characteristic* definiert. Zum Beispiel, ob diese gelesen oder beschrieben werden darf. Ein *Accessory* besteht immer mindestens aus dem *Accessory Information Service*. Dieser Service hat sechs verpflichtendende *Cha-*

racteristics, um Metadaten des Geräts abzuspeichern und einsehbar zu machen. Diese *Characteristics sind Firmware Revision, Identify, Manufactuerer, Model, Name, Serial Number*. Die *Identify Characteristic* beschreibt durch einen *bool*, ob die *Identify* Routine ausgeführt wird, die Implementation abhängig dazu gedacht ist ein Gerät zu identifizieren, wie zum Beispiel durch das Aufblinken einer Kontroll-LED. Die anderen *Characteristics* speichern als *String*, die entsprechenden Daten.
Diese Bausteine erlauben es durch vorgefertigte *Services* eine Vielzahl an *HAP* konforme Smart Home Geräte zu erstellen.

2.2.2 Matter

Der *Matter* Standard ist eine Initiative von einer Vielzahl von *Smart Home* Herstellern einen Standard zu schaffen, der Kompatibilität von Geräte unterschiedlicher Hersteller erlaubt. Dabei ist der *Matter* Standard das Protokoll, das die Kommunikation der Geräte zueinander definiert und auch das Datenmodell, wie sich die Geräte beschreiben und für die anderen Teilnehmer des Netzwerkes zu erkennen sind. Ein System, das den *Matter* Standard benutzen soll, benötigt ein anderes *Smart Home System*, welches sich um die Automationen und andere Funktionalitäten kümmert[22]. Ein Smart Home System, das den Matter Standard unterstützt, ist das Smart Home System von *Apple*[28].
Die *Matter* Dokumentation ist in drei Dokumenten aufgeteilt.

- *Matter 1.0 Core-Specification* [35]
  Dieses Dokument beschreibt Anforderungen der *Matter* Spezifikation, um das *Matter* Protokoll über *IP* zu implementieren

- *Matter Device Library Specification Version 1.0* [34]
  Dieses Dokument beschreibt alle grundlegenden Arten an Gerätetypen, die das *Matter* Protokoll unterstützt

- *Matter-1.0 Application-Cluster-Specification* [33]
  Dieses Dokument beschreibt eine Vielzahl an Cluster, die von den Geräten, aus der *Device Library* benutzt werden können.

Ein Gerät, das der *Matter* Spezifikation entspricht, wird in einem hierarchischen Datenmodell beschrieben. Das *Device* ist der Hauptknoten der Struktur. Ein *Device* kann aus einer beliebigen Anzahl an *Nodes* bestehen. Dabei sind *Nodes* eine komplette Beschreibung eines Produkts. Im normalen Fall besteht ein physikalisches *Device* aus einem *Node*. Ein *Node* besteht aus einer beliebigen Anzahl an *Endpoints*. Dabei ist jeder *Endpoint* die Beschreibung von einem bestimmten Gerätetypen. Diese *Endpoints* haben eine beliebige Anzahl an *Clustern*. *Cluster* beschreiben, wie mit einem Gerät interagiert werden kann und welche Attribute dieses Gerät hat. Bei *Clustern* wird unterschieden, ob es sich um einen Server oder einen Client handelt. Ein Server speichert den Zustand

eines Gerätes ab und reagiert auf eingehende Befehle. Ein Client kommuniziert mit einem Server, um Zustände beim Server zu verändern oder abzurufen[32]. In Fig. 2.2 ist

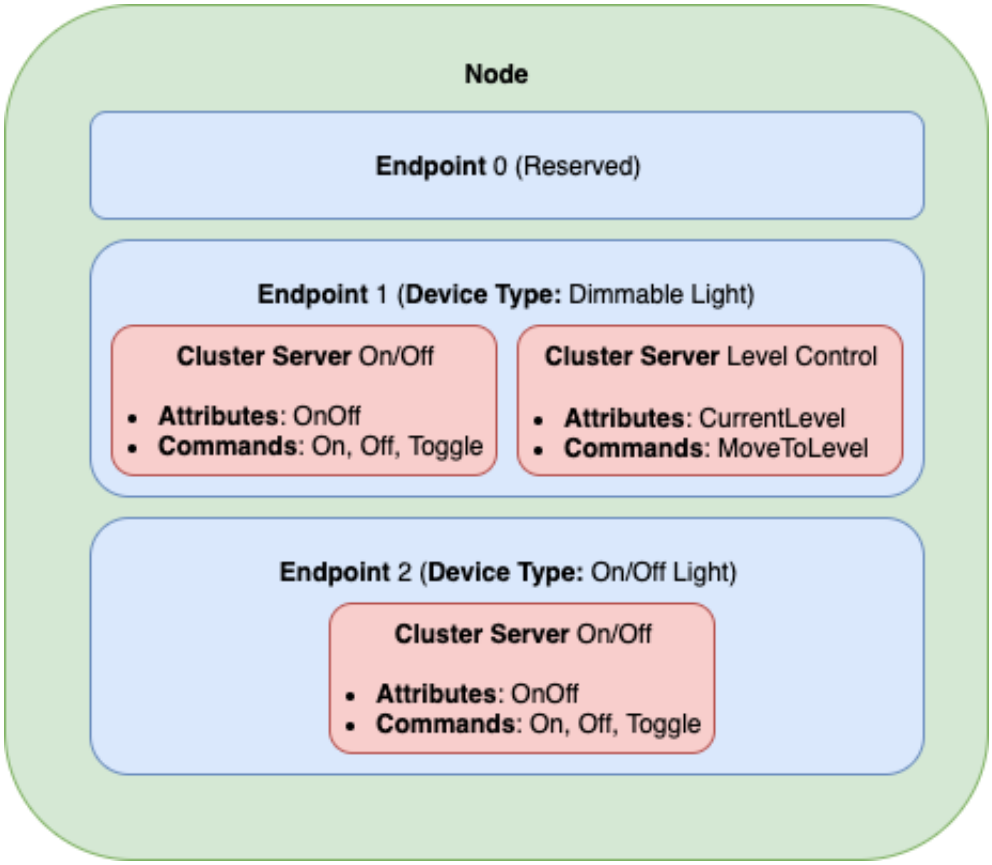


Abbildung 2.2: Datenstruktur eines *Matter* Geräts, das aus zwei Glühbirnen besteht.[43]

der Aufbau eines *Matter Devices*, das aus einem *Node* und mehreren *Endpoints* besteht, illustriert. Im *Endpoint 0* werden Metadaten eines *Nodes* abgespeichert und ist Teil eines jeden *Nodes*. In diesem Beispiel besteht das restliche *Device* aus zwei weiteren *Endpoints* bzw *Device Types*. Diese *Device Types* entsprechen vordefinierten *device types* aus der *Matter Device Libary*. Bei dem *Endpoint 1* handelt es sich um eine Lampe, deren Licht gedimmt werden kann und besteht deshalb aus zwei *Cluster Servern*. Der *On/Off Cluster Server* beschriebt, ob diese Lampe an ist und durch den *Level Control Cluster* wird beschrieben, wie hell die Lampe leuchtet. *Attributes* und *Commands* werden in der *Application-Cluster-Specification* genauer definiert. Beispielhafte weitere Informationen, die durch die *Level Control* definiert werden können, sind der maximale und minimale Wert der dimmbaren Lampe. Bei dem *Endpoint 2* handelt sich um eine Lampe, die nur an oder aus geschaltet werden kann und enthält deshalb nur den *On/Off Cluster Server*.

2.2.3 Vergleich der Matter- und HAP-Modelle

Die Komponenten der Smart Home Standards sind sehr ähnlich zueinander aufgebaut und jede Komponente hat ein funktional vergleichbares Gegenstück im anderen Standard. Ein *Node* entspricht einem *Accessory*, es ist jeweils der Rahmen eines Gerätes an sich. Die *Endpoints* sind vergleichbar mit den *Services,* es handelt sich jeweils, um die Beschreibung der physischen Objekte, aus denen die Geräte bestehen. Beide Standards haben einen extra *Endpoint*, bzw *Service*, um die Metadaten des Geräts abzuspeichern. Die kleinsten Komponenten sind auch vergleichbar in Form der *Cluster* und der *Characteristics*, um die eigentlichen Attribute der physischen Objekte zu beschreiben. Der größte Unterschied der beiden Standards ist, dass *Matter* genauer beschreibt, wie mit einem Gerät interagiert werden kann. In *Matter*, wie in Fig. 2.2 gezeigt, kann genau definiert werden, wie mit der Glühbirne interagiert werden kann, das ist in *HAP* nicht möglich. Im Laufe der Arbeit, können durch diese Gemeinsamkeiten Datenmodelle erstellt werden, die benutzt werden können, um beide Standards zu modellieren.

2.3 Digital Twins

Eine einfache Beschreibung eines *Digital Twins(DT)* ist nach Grübel et al. [24] eine Software, die etwas aus der Realität digital beschreibt. Da diese Definition allerdings nicht eindeutig ist, sondern auch andere Konzepte beschreiben könnte, wurden von Grübel et al. [24] weitere Aspekte aufgeführt, die einen DT auszeichnen:

1. Der DT liefert ein perfektes digitales Spiegelbild des *Physical Objects(PO)*.
2. Der DT kann durch Simulation kritische Ereignisse voraussagen und entsprechend reagieren. Dabei ist das System fähig zu adaptieren, falls Vorhersagen und tatsächliche Ereignisse unterschiedlich sind.
3. Der DT und das PO sind nicht dasselbe Objekt. Das bedeutet, dass das Objekt nicht die Software des DT ausführen muss, sondern nur mit diesem in Verbindung steht.
4. Der DT kann das PO beeinflussen.

Gerade die ersten beide Aspekte, das perfekte digitale Spiegelbild und die Simulationen sind in der Kombination schwierig bis hin zu unmöglich zu erreichende Ziele. Um diese zu erreichen, müssten komplexe Simulationen durchgeführt werden, während perfekte Informationen über das zu beschriebene Objekt vorhanden seien müssen und es müsste durchgängig eine Verbindung zwischen DT und PO herrschen, in denen alle Daten durchgängig synchronisiert werden. Grübel et al. haben aus diesem Grund eine Definition für einen *Practical* DT erstellt, welche aus fünf Komponenten besteht. Diese Definition hat das Ziel, die Definition eines DT zu simplifizieren und hat dabei keinen Anspruch auf

ein perfektes digitales Spiegelbild, sondern eine praxisorientierte komponentenbasierte Struktur zu definieren, die einen *DT* beschreibt [24].
Diese Komponenten sind:

- *Physical Environment*, in dieser Komponente werden die Sensoren des *PO* und ihre Daten ausgewertet und das *PO* wird durch Aktuatoren beeinflusst.
- *Data Enviroment*, in dieser Komponente wird der Speicher des *DT* abgebildet. Die Daten können lokal oder in einer Cloud abgespeichert werden.
- *Analytical Enviroment*, in dieser Komponente werden die Daten ausgewertet und die Ergebnisse werden zum *Data Enviroment* weiter geleitet.
- *Virtual Enviroment*, in dieser Komponente wird eine Schnittstelle geliefert, um eine Interaktion mit dem *DT* zu erlauben. Diese Interaktion umfasst die Kommunikation zwischen *DT* und Nutzern wie auch anderen Maschinen in beiden Richtungen.
- *Connection Enviroment*, in dieser Komponente werden die nötigen Funktionen geliefert, um eine Kommunikation zwischen den unterschiedlichen Komponenten zu gewährleisten.

Mit dieser Definition ist es möglich, einen idealen *DT* teilweise zu implementieren.
Im Folgenden wird auf den *Longevity Digtial Twin* eingegangen, eine Architektur eines *DT*, die für Problemstellengen im Smart Home Kontextes entworfen wurde [46] und es wird bewertet, ob diese der praktischen Definition eines *DT* entspricht oder adaptiert werden muss.

2.3.1 Die Longevity Digital Twins(LDT) Architektur

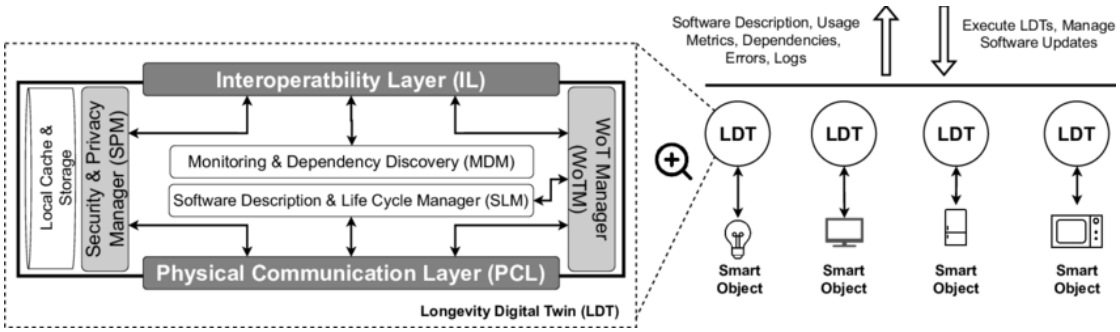


Abbildung 2.3: Die Longevity Digital Twin Architektur[46]

Der *Longevity Digital Twin(LDT)* ist eine von Zdankin et al. vorgestellte Architektur eines *DT*, die es ermöglichen soll, Abhängigkeiten in einem Smart Home festzustellen, um Updates zu bewerten und so unerwünschte Nebeneffekte zu verhindern. In diesem

Kapitel wird die Architektur des LDT nach [46] vorgestellt.
Die Architektur, illustriert in Fig. 2.3, eines solchen *LDT* besteht aus:

- Dem *Physical Communication Layer(PCL)*, der alle nötigen Funktionalitäten liefert, damit der *LDT* Funktionen des *PO* auslösen kann und Daten des *PO* erhalten kann.
- Der *Interoperability Layer(IL)* der die Kommunikation zwischen *LDT* und Partnern organisiert, die nicht das *PO* sind. Das könnte ein Benutzer sein oder auch andere Maschinen.
- Dem *Web of Things Manager(WoTM)* der Daten nach dem *Web of Things* Standard beschreibt.
- Der *Monitoring and Dependicies Manager(MDM)* und der *Software Description and Life Cycle Manager(SLM)*, liefern die analytischen Komponenten des *LDT*. Der *MDM* analysiert Interaktionen über den *IL* mit Benutzern und Maschinen und erstellt passenden Abhängigkeiten und Beziehungen. Der *SLM* erkennt und beschreibt die Komponenten des *PO* und interagiert mit dem *WoTM*, um das *PO* im *Web of Thing* Standard zu beschreiben.

Des Weiteren wird mit dem der *Orchestral and Development Manager(ODM)* eine zusätzliche Komponente geliefert, um Abhängigkeiten von *LDTs* zueinander auszuwerten.
Der *ODM* besteht aus:

- *LDTs Orchestrator(LO)* zur Organisation mehrerer *LDTs*
- *Dependency Graph Manager(DGM)* zur Erstellung eines Abhängigkeitsgrafen und der Auswertung dieses
- *Software Update Manager(SUM)*, um Updates anhand des DGM zu bewerten

Die gegebenen Komponenten des *LDT* lassen sich auf die praktische Definition eines *DT* anwenden. Der *PCL* entspricht der *Physical Environment* und der *IL* entspricht dem *Virtual Enviroment. Daten Environment* und *Analyitical Environment* werden durch das *MDM* und *SLM* abgebildet. Dem *Connection Environment* entspricht die in Fig.2.3 durch Pfeile illustrierten Verbindungen zwischen den Komponenten. Des Weiteren liefert die *LDT* Architektur mit dem *ODM* eine weitere Komponente, um mehrere *LDTs* untereinander zu organisieren.

2.4 Web of Things und die Thing Description

Das Ziel des *Web of Things(WoT)* ist es, der „Fragmentierung des *IoT* entgegenzuwirken, indem bestehende und standardisierte Web Technologien erweitert werden“(vlg. [16]). Zentrale Bestandteile des *WoT* sind *WoT Interactions* bestehend aus *Properties,*

Actions und Events und die *Web of Things Thing Descriptions(WoT-TD)*. Die *WoT Interactions* beschreiben, wie mit einem *Thing* interagiert werden kann und die *WoT-TD* definiert, wie ein *Thing* im *JSON*-Format aufgebaut ist [15]. In diesem Kapitel wird eine Übersicht über die *WoT-TD* gegeben, mit dem Fokus auf die *WoT Interactions*. Es wird außerdem ein Überblick über den *link*-Baustein gegeben, da dieser Abhängigkeiten von *Things* untereinander abbildet. Im Anhang sind zwei komplette *WoT-TD* abgebildet, die einerseits ein *HomeKit* Gerät(siehe Listing. A.1.1) und anderseits ein *HomeAssistant* Gerät(siehe Listing. A.1.2) beschreiben.

2.4.1 Thing Description

Bei der *Web of Things Thing Descriptions(WoT-TD)* handelt es sich um ein Model, das die Metadaten und Kommunikationsmöglichkeiten eines *Thing* beschreibt. Dieses Unterkapitel basiert auf *Web of Things (WoT) Thing Description 1.1[31]*. Das Model ist im *JSON*-Format spezifiziert und ermöglicht so, dass es sowohl vom Menschen als auch von Maschinen verarbeitet werden kann. Eine *TD* besteht aus vier Bausteinen, um ein *Thing* zu beschreiben, den Metadaten, Interaktionsbeschreibungen, Daten-Schemas und Weblinks. Im Folgenden werden die Bausteine beschrieben und ihre wichtigsten Felder vorgestellt.

2.4.1.1 Aufbau des Thing Objekt

Die Daten werden im Wurzel-Objekt *Thing* gespeichert. Alle anderen Objekte sind Teil dieses Objekt. Das *Thing*-Objekt hat vier benötigte Felder: *@context, title, security* und *securityDefinitions*. Das *@context*-Feld ist das Wurzelelement und gibt die benutzte Version des *WoT-TD* an. Es können weitere Kontexte aufgeführt werden, wenn zum Beispiel Begriffe von einer anderen Spezifikation benutzt werden und diese als solche identifiziert werden sollen. Der *Title* ist ein Name in einem Menschen lesbaren Format. Bei den *security*- und *securityDefinitions*-Feldern werden die Sicherheitsprinzipien angegeben, die dieses *TD* erfüllt. In dieser Arbeit wird, falls nicht anders erwähnt, hauptsächlich die *no_sec* Definition benutzt. Das bedeutet, dass keine weiteren Sicherheitsmechanismen benutzt werden bei der Interaktion mit dem *TD*. Dazu wurde entschieden, da Sicherheit im *DT*- und Smart Home-Kontext außerhalb des Bereichs dieser Arbeit liegt.
Das *Thing*-Objekt hat eine Vielzahl an optionalen Feldern. In diesem Kapitel werden auf die Felder *Properties, Actions, Events* und *Links* eingegangen, da diese am relevantesten sind. Bei *Properties, Actions, Events* handelt es sich um Unterkategorien der *Interaction Affordances*.

2.4.2 Interaction Affordances und das Form-Objekt

Das einzige verpflichtende Feld einer *Interaction Affordance* ist das *forms*-Feld, bestehend aus beliebig vielen Form-Objekten.
Bei einer Form handelt es sich um eine Beschreibung, wie diese Interaktion durchgeführt werden kann. In einer Form wird über das *op*-Feld die Art der Operation beschrieben. Dabei gibt es eine Vielzahl an vorgeschriebenen Operationen. Relevante Operationen sind unter anderem: *readproperty, writeproperty* oder *observeproperty*. Im Weiteren wird im *href*-Feld eine URI angegeben. Diese URI kann zum Beispiel eine Webadresse sein, die aufgerufen werden muss, um die Interaktion auszuführen. Über den *contentType* wird angegeben, in welchem Format Daten gesendet und zurückgegeben werden.
Weitere Felder der *InteractionAffordance* sind die Felder: *title* und *description*, um für einen menschlichen Leser die *Interaction* zu benennen und zu beschreiben.

- Bei einer *Property* handelt es sich um eine Eigenschaft von einem *Thing*, wie zum Beispiel die Helligkeit einer Lampe. Eine Property kann ausgelesen werden oder gesetzt werden. *Properties* werden durch eine *PropertyAffordance* beschrieben, die alle Felder einer *InteractionAffordance* erben und das *observable* Feld hinzufügt. Das Observable Feld gibt an, ob diese Property beobachtet werden kann. Außerdem erbt die *PropertyAffordance* von *DataSchema*, um eine genaue Beschreibung der Daten zu ermöglichen, wie zum Beispiel den Minimal oder Maximalwert von einer *Property*.

- Bei einer *Action* handelt es sich um eine Funktion, die auf dem Device ausgeführt werden kann. Das kann ein einfaches Toggeln einer Lampe sein oder komplexere Aktionen. Die *ActionAffordance* erweitert die *InteractionAffordance* um *input, output, safe, idempotent* und *synchronous* Felder. Die *input*- und *output*- Felder sind *DataSchemas*, die beschreiben, welches Format die Eingabe und Ausgabe der Aktion hat. Das *Safe*-Feld gibt an, ob durch die *Action* Veränderungen bei dem *Thing* hervorgerufen werden oder nur Daten ausgelesen werden. Falls Veränderungen hervorgerufen werden, ist die Interaktion nicht *safe*. *Idempotent* gibt an, ob wiederholtes Aufrufen der Aktion das gleiche Ergebnis hat und *synchronus* gibt, an, ob mehrmaliges Aufrufen der *action* unterschiedliche Ergebnisse liefert.

- Ein *Event* ist ein Ereignis, das von dem *Thing* getriggert wird. Das kann zum Beispiel eine Benachrichtigung sein, wenn sich eine *Property* verändert. *EventAffordance* erweitert die *InteractionAffordance*, um alle nötigen Felder um eine asynchrone Kommunikation zu ermöglichen. Mit dem *subscription*-Feld wird angegeben, welche Daten erwartet werden, um das *Event* zu abonnieren. *Data* beschreibt, welche Daten das *Thing* beim Eintreten des Events sendet. *Dataresponse* beschreibt, wie eine Antwort auf diese Daten aussehen muss und das *cancellation*-Feld gibt an, wie das Event gekündigt werden kann. Alle vier Felder sind vom Typ *DataSchema*.

- Ein *Link* beschreibt die Beziehung von einem *Thing* zu einem anderen *Thing* oder zu weiterer Dokumentation, die das *Thing* weiter beschreibt. Ein *Link* hat ein verpflichtendes Feld *href*. In dem *href*-Feld kann zum Beispiel die *URI* einer *Form* einer *ActionAffordance* einer *TD* eines anderen *Things* angegeben werden, um zu beschreiben, dass dieses *Thing* von der *Action* des anderen *Things* beeinflusst wird. Mit dem *rel* Feld kann beschrieben wird, um was für eine Beziehung dieser *Link* beschreibt. Für diese Arbeit ist vor allem die *controlledBy* Beziehung, um zu beschreiben, dass das *Thing* von dem *Link* kontrolliert wird und die *Collection*, sowie *Item* Beziehungen, relevant. *Collection* und *Item* können benutzt werden, um zu beschreiben, dass mehrere *Things*, als Items, Teil eines anderen *Things* sind, der *Collection*.

2.4.3 Bewertung der WoT-TD

Die *WoT-TD* bietet alle Bausteine, um ein Smart-Gerät zu beschreiben. Es können Meta Informationen im Wurzel-Objekt abgespeichert werden und jegliche Funktionalität als Properties ausgedrückt werden. Die Aktionen, die von einem Gerät ausgeführt werden können, werden durch die *Events* und *Actions* abgebildet. Die *Forms*-Objekte bieten die Möglichkeit alle Interaktionen klar zu definieren und durch die Link-Felder können Interaktionen zwischen Geräten dargestellt werden.

Kapitel 3

Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten vorgestellt, die sich mit ähnlichen Problemstellungen beschäftigt haben.

3.1 Bootstrapping

Um den *LDT* einzurichten, stellt sich die Frage, wie das *Smart Home* Gerät von dem *LDT* erfährt und wie die Registrierung und das Erstellen des Modells des Geräts beim *LDT* ausgeführt wird. In diesem Unterkapitel werden unterschiedliche Ansätze vorgestellt, die in der Arbeit *Internet of Things Patterns for Device Bootstrapping and Registration* von Lukas Reinfurt et. al. [39] beschrieben werden und es wird darauf eingegangen, wie diese Ansätze für das *Bootstrapping* eines *Digital Twins* benutzt werden können und welche am besten geeignet sind.

3.1.1 Grundlagen

Das Ziel von *Bootstrapping* ist es ein Mechanismus zu entwickeln, der im Kontext des *IoT* die Kommunikation zwischen zwei Geräten startet. Der Ablauf, wie ein Gerät Teil eines *IoT* Netzwerk wird, kann nach [39] in drei Schritte unterteilt werden. Im Folgenden werden alle drei Schritte und die zugehörigen Mechanismen vorgestellt, die das *Bootstrapping* ermöglichen.

3.1.2 Device Bootstrapping

Im ersten Schritt wird die Verbindung von dem Gerät zum Server aufgebaut. Dieser Schritt ist das *Device Bootstrapping*. Ziel dieser Phase ist es, dem Gerät die nötigen Daten, wie z.B. die *IP*-Adresse vom Server, zu liefern, die zum Verbindungsaufbau vonnöten sind. Es gibt drei verschiedene Arten von *Device Bootstrapping*.

- *Factory Bootstrap*
  Die Daten des *Servers* werden bei der Herstellung dem Gerät übergeben. Die Vorteile und Nachteile dieses Vorgehens werden nicht im Paper weiter erarbeitet, aber es lassen sich folgende Vorteile und Nachteile ableiten. Vorteile, die dieses Vorgehen haben könnten, wären Sicherheit, solange dem Hersteller vertraut werden kann und Einfachheit, da das Gerät vom Werk aus schon alle wichtigen Daten zur Verfügung hat, keine weitere Hardware benötigt wird und skalierbar ist. Nachteile wären ein Mangel an Flexibilität, da das Ändern des Servers nicht ohne weiteres möglich ist.

- *Medium-based Bootstrap*
  Die Daten des Servers werden durch ein externes Medium dem Gerät übergeben. Die Vorteile des Vorgehens sind Unabhängigkeit und Flexibilität, da das externe Medium frei konfiguriert werden kann. Es bietet auch eine erhöhte Sicherheit, da die kritischen Daten, die für den Informationsaufbau ausgetauscht werden müssen, nicht über Netzwerke kommuniziert werden.
  Die Nachteile des Vorgehens sind vor allem dadurch bedingt, dass es ein physischer Gegenstand ist, der mit jedem neuen Gerät verbunden werden muss. Es muss außerdem möglich sein, dass das Gerät physikalisch erreichbar ist und Medium muss immer auf dem aktuellsten Stand sein. Des Weiteren erhöhen sich die Kosten des Systems, da jedes neue Gerät mit dem *Bootstrapping*-Gerät verbunden werden muss und jedes Gerät auch die nötige Hardware liefern muss. Sicherheit technisch gibt es die Nachteile, dass einerseits das Medium gestohlen werden könnte und so Angreifer eigene Geräte mit dem Server verbinden könnten. Anderseits könnte das *Bootstrapping*-Gerät ausgetauscht werden und neue Geräte würden sich mit dem Server des Angreifers verbinden.

- *Remote Bootstrap*
  Beim *Remote Bootstrap* wird dem Gerät, z.B. per *Factory Bootstrap*, die Adresse eines *Bootstrap Servers* mitgeteilt. Diese Adresse wird benutzt, damit sich das Gerät mit dem Server verbindet und über den Server die Adressen des gewünschten Ziels erhält. Vorteile dieses Vorgehens sind, dass im Gegensatz zum *Factory Bootstrap* die eigentlichen *Bootstrap* Informationen flexibel sind. Es ist möglich das System zu skalieren und das Gerät benötigt keine neue Hardware. Nachteile sind die Abhängigkeit zu einem anderen *Bootstrapping* Mechanismus um sich mit dem Remote Server zu verbinden, außerdem müssen diese Server auch gewartet werden.

Für einen *LDT* bietet sich sowohl das *Factory Bootstrap* als auch das *Remote Bootstrap* an. Es kann entweder die Adresse des *LDT* per *Factory Bootstrap* kommuniziert werden, wenn sich der *LDT* in einem Heimnetzwerk befindet oder wenn sich der *LDT* in einer *Cloud* befinden würde, wäre der *Remote Bootstrap* eine gute Wahl. Der *Medium-Based Bootstrap* ist eher ungeeignet für einen *LDT*, da Smart Home Geräte oft nur über Netzwerklose Kommunikation kommunizieren und keine Anschlüsse besitzen, um ein Gerät

an dieses anzuschließen.

3.1.3 Device Registration

Sobald das Gerät mit dem Server, nach dem *Bootstrapping* Vorgang, verbunden, ist, muss in den meisten Fällen das Gerät auch im Server registriert werden, um die Kommunikation zwischen Server und Gerät zu organisieren.

- *Automatic Client-Driven Registration*
  Bei diesem Vorgang schickt der Client alle nötigen Informationen über einer *API* an den Server.
- *Automatic Server-Driven Registration*
  Bei diesem Registrierungsvorgang fordert der Server alle nötigen Daten von dem Client an, um eine Repräsentation des Geräts auf dem Server zu speichern.
- *Manual User-Driven Registration*
  Bei der *Manual User-Driven Registration* muss der Benutzer selbst die nötigen Daten an den Server übergeben. Dazu stellt der Server die nötigen Funktionalitäten zur Verfügung.

Für einen LDT bietet sich nur die *Automatic Client-Driven Regristration* an. Die *Automatic Server-Driven Registration* würde bedeuten, dass der *LDT* für jedes Gerät, das sich mit ihm verbinden will, wissen muss, welche Daten er von den Geräten abfragen kann und weiß, welche Daten relevant sind. Es ist einfacher, wenn der *LDT* eine *API* anbietet, die jeder Client benutzen kann und so dem *LDT* alle relevanten Daten überträgt. Dadurch muss jedes Gerät nur einmal die *API* aufrufen und der *LDT* muss nur eine *API* implementieren für eine Vielzahl an Geräten. Die *User-Driven Registration* ist nicht empfehlenswert, da ein *LDT* nur Geräte abbilden sollte, mit denen er auch kommuniziert. Es müsste also ein weiterer Mechanismus implementiert werden, der verifiziert, dass das *Device*, das ein Nutzer registriert hat, auch wirklich in dieser Art existiert.

3.1.4 Device Model Patterns

Sobald sich das *Device* registriert hat, muss ein Modell für dieses Gerät erstellt werden.

- *Server-Driven Model*
  Der Server generiert Modelle für jedes Gerät und speichert diese auf dem Server.
- *Pre-Defined Device-Driven Model*
  Die Devices haben vorgefertigte Modelle, die sie sich aussuchen müssen und dann dem Server mitteilen, welche der Modelle ihr System am besten beschreibt.

- *Device-Driven Model*
  Das Gerät erstellt selber ein Modell seiner Daten und schickt dieses Modell an den Server, der dieses abspeichert.

Für einen *LDT* bietet sich das *Server-Driven Model* an. Der *LDT* und das Gerät müssen die nötigen Metadaten austauschen, sodass der *LDT* das Gerät eindeutig identifizieren kann und entsprechend abspeichert. Es wäre möglich das *Device-Driven Model* zu benutzen, wenn auf den Geräten ein Modell generiert wird, das genau dem Datenmodell, das der *LDT* erwartet entspricht. Da aber der Vorteil des *Device-Driven* Models gerade in dezentralen Systemen liegt, ist es für einen *LDT* eine unnötige Stufe der Komplexität auf jedem Gerät das Modell des *LDT* zu implementieren, da der *LDT* der Kern eines zentralen Systems ist.

3.1.5 Auswertung

Es zeigt sich, dass sich allgemeine *Bootstrapping* Ansätze auch auf einen *LDT* anwenden lassen. Der *LDT* ist in diesem Kontext vergleichbar mit einem *IoT*-Server und die Ansätze, die sich für ein zentrales *IoT*-System anbieten, sind auch für den *LDT* relevant. Die einzige Ausnahme sind Benutzer gesteuerte Modelle. Der *LDT* muss in der Lage sein, aus den Informationen, die ein Gerät dem *LDT* schickt, ohne Benutzerinteraktion die Registrierung durchzuführen und Modelle zu erstellen. Das hat den Grund, dass sonst auf dem *LDT* Daten registriert oder gespeichert werden könnten, die nicht von dem *LDT* nachvollziehbar und kontrollierbar sind.
Ein weiterer Aspekt der beachtet werden sollte, bei dem Bootstrapping auch relevant ist, ist die Kommunikation zwischen *LDT* und anderen *Home* Servern. Der *LDT* kann durch *Factory Bootstrap* oder *Remote Bootstrap* die Adressen von *Home Servern* im Netzwerk erhalten und so eine Verbindung mit diesen aufbauen. Daraufhin muss der *LDT* sich abhängig von dem *Bootstrapping* Prozesses des jeweiligen *Smart Home Systems* registrieren, und die Daten aus dem Server auslesen und in das für den *LDT* verwendete Datenmodell konvertieren.

3.2 IoTSanitizer (IoTSan)

IoTSan ist ein Ansatz von Nguyen et al. der versucht *IoT* Systeme zu organisieren. Dabei wird *Model Checking* benutzt, um fehlerhafte oder unsichere Konfigurationen in einem System zu erkennen[36]. Die Motivation der Arbeit ist es Systeme, die typisch sind für *IoT* Systeme zu analysieren und Fehler zu erkennen. Diese Fehler können zum Beispiel auftreten, dadurch, dass ein *IoT* System aus mehreren Geräten unterschiedlicher Hersteller besteht und es auch eine Vielzahl an Nutzer geben könnte, die das

System unterschiedlich konfigurieren. Eine weitere Motivation ist die Sicherheit, da fehlerhafte Konfigurationen, die zum Beispiel durch Ausfallen von Geräten von Angreifern ausgenutzt werden könnten. In diesem Kapitel wird das Konzept von *IoTSan*[36] kurz vorgestellt und argumentiert, welche Aspekte für einen *LDT* aufgegriffen werden können und welche Aspekte nicht für einen *LDT* verwendbar sind.

3.2.1 Samsung SmartThings

IoTSan benutzt in dem vorgestellten *Paper* die *IoT* Plattform *Samsung SmartThings*. Diese Plattform benutzt die Programmiersprache *Groovy*. Das System besteht aus den *Smart Things*, einem *Hub* und einer *App*. Die Geräte werden virtuell mitsamt den *Smart Apps* in einer Cloud ausgeführt.

3.2.2 System

IoTSan besteht aus fünf Komponenten, die dafür zuständig sind ein typischen *IoT* System in ein Modell umzuwandeln und es zu analysieren.

1. *App Dependency Analycer*
   Der *App Dependency Analyzer* baut einen *Dependency* Graph auf, um alle Interaktionen auszusortieren, bei denen Apps nicht miteinander interagieren. Der Graph wird aufgebaut, in dem *event handlers* von den Apps ausgelesen werden. Die *event handler* bestehen aus einem oder mehr *input events* und null bis beliebig viele *output events*. Mit diesen *input* und *output events* wird dann ein gerichteter Graph aufgebaut. Jede Knoten ist ein *event handler*, jede eingehende Kante ist ein *input Event* und jede ausgehende Kante ein *Output event*. Sobald der Graph aufgebaut wurde, werden *related sets* und *conflicting sets* definiert. Ziel dieser Sets ist es *event handler* zu erkennen, die eine Status erreichen können, der einen anderen *event handler* ausschließt. Beispielsweise wäre eine *event handler* der als *outgoing event* eine Lampe ausschaltet, mit einem *event handler* der als *outgoing event* die gleiche Lampe einschaltet im Konflikt.

2. *Translator*
   Bei dem *translator* wird das *Bandera Tool Set* benutzt, welches eine Sammlung von *model checking* Algorithmen für *Java Source Code* ist. Für dieses wurde ein *Groovy* nach *Java Code* Konverter gebaut, um so den Code mit dem *Bandera Tool* analysieren zu können.

3. *Configuration Extractor*
   Der *Confuguration Extractor* holt sich die Metadaten von installierten Apps aus

der Manager App. Dies wurde durch eine *Java* Implementation gebaut, die, nachdem ein Passwort und User Name gegeben wird, alle Daten aus der Web-App von *Samsung* extrahiert.

4. *Model Generator*
   Der *Model Generator* baut ein *IoT* System aus den vorher extrahierten Daten und jedes *IoT* Gerät anhand der Spezifikation jedes Gerät.

5. *Output Analyzer*
   Der *Output Analyzer* analysiert die vorher erstellten Logs und gibt dem Nutzer eine Auskunft, ob eine App falsch konfiguriert wurde oder gelöscht werden sollte. Dabei wird die gegebene Konfiguration mit allen möglichen richtigen Konfigurationen der App abgeglichen, falls die Fehlerquote eine bestimmte Schwelle überschreitet, wird die App als fehlerhaft eingestuft. Anderseits wird getestet, ob die App mit allen vorher installierten Apps eine Fehlerquote erreicht, die über diese Schwelle tritt. Falls dies auch nicht der Fall ist, wird die App akzeptiert.

3.2.3 Auswertung

IoTSan zeigt sich als ein erfolgreiches-Konzept, das Nguyen et al. in ihrer Evaluation beweisen. In der Evaluation wurde eine große Anzahl an Fehler gefunden, die durch falsche Konfigurationen Sicherheitsrisiken für den Nutzer bedeutet hätten.
Die Zielsetzung von *IoTSan* und dem *LDT* sind vergleichbar. Durch das Modellieren des Systems und dem Analysieren von *dependencies*, wird ein System leichter auf Fehler zu untersuchen. Der große Unterschied zwischen *IoTSan* und dem *LDT* ist, dass der *LDT* in der Lage sein muss, die Geräte einer Vielzahl von Herstellern vergleichbar zu machen. Nguyen et al. bieten zwar auch Konzepte an, um auch andere Apps zu übersetzen und zu analysieren, aber dann gibt es immer noch das Problem, womit ein *LDT* umgehen können muss, und zwar dass der *LDT* auch Geräte repräsentieren können sollte, die nicht extra für den *LDT* erstellt wurden.
Der Ansatz Abhängigkeiten durch Interaktionen auszudrücken ist ein Ansatz, der von dem *LDT* aufgegriffen werden kann. Dabei müssen die Interaktionen, die Geräte an sich ausführen können und Interaktionen, die ein Gerät ausführt und dabei ein anderes Gerät beeinflusst, modelliert werden können.

Kapitel 4

Problemstellungen für den Longevity Digital Twin(LDT)

In diesem Kapitel werden alle Problemstellungen aufgeführt, die sich aus dem Grundlagenkapitel ableiten lassen und für diese Arbeit relevant sind. Dabei wird in zwei Kategorien unterschieden. Die erste Kategorie sind interne für den *LDT* relevante Problemstellungen und die zweite Kategorie sind Kommunikation relevante Probleme.

4.1 Anforderungen an den LDT

Aus der in Kapitel 2.3 vorgestellten Architektur eines *LDT*[46] und der Definition eines *practical DT*[24] lassen sich für die internen Komponenten Anforderungen ableiten, die in diesem Kapitel vorgestellt werden.

4.1.1 Anforderungen an den Datenstrukturen

Datenstrukturen sind ein zentraler Bestandteil des *LDT*, da diese nötig sind, um ein Smart Home Gerät zu beschreiben, zu organisieren und zu beschreiben. Bei dem Design der Datenstrukturen muss bedacht werden, dass nicht nur die Daten effizient gespeichert werden, sondern dass diese auch von Maschinen gelesen und von einer Komponente zu nächsten geschickt werden können.
In diesem Kapitel werden zwei Gruppen an Anforderungen vorgestellt, die die Datenstrukturen des *LDT* erfüllen müssen, die benötigten Anforderungen und die optionalen Anforderungen.
Benötigte Anforderungen sind alle Anforderungen, die minimal erfüllt werden müssen, damit der *LDT* entsprechend der Definition eines *DT* funktioniert. Die optionalen Bedingungen sind Bedingungen, die erstrebenswert sind, um weitere Funktionalitäten eines *LDT* zu ermöglichen.
Die benötigten Anforderungen sind:

1. Die Daten müssen das Smart Home Gerät beschreiben. Dadurch, dass sich an der praktischen Definition eines *Digital Twin* orientiert wird und es nicht darum geht ein perfektes digitales Spiegelbild eines physikalischen Objekts zu schaffen, muss hauptsächlich die Funktionalität des Smart Home Geräts beschrieben werden. Es ist also nicht Ziel jedes Detail zu beschreiben, wie zum Beispiel welcher Pin genau eines Mikrocontrollers eine Glühbirne schaltet, sondern es liegt der Fokus darin, ein Modell zu schaffen, das diese Glühbirne eindeutig modelliert und so direkt klar ist, welche Interaktionen mit der Glühbirne möglich sind, welche Status die Glühbirne haben kann und mit welchem Protokoll diese Implementiert wurde

2. *Machine to Machine Communication* und Maschinenlesbarkeit muss gewährleistet sein, damit die Daten zwischen den Komponenten weitergereicht werden können und eine automatische computergesteuerter Analyse mehrerer *LDTs* durchgeführt werden kann, um zum Beispiel den Abhängigkeitsgraph zu erstellen.

3. Die Unabhängigkeit zum Smart Home Gerät muss gegeben sein. Es muss möglich sein, mit derselben Struktur, alle Funktionalitäten eines beliebigen Geräts darzustellen, unabhängig welche Funktionen dieses hat oder welches Protokoll es benutzt. Dies ist notwendig, damit der *LDT* unabhängig von Geräteherstellern funktioniert. Es muss also möglich sein, beispielsweise einen *Matter* konformen Lichtsensor, wie auch eine *HAP* konforme Glühbirne mit derselben Struktur zu beschreiben.

4. Die letzte benötigte Bedingung ist eine Vergleichbarkeit der Strukturen. Es muss möglich sein, die Datenstrukturen auf zwei Arten zu vergleichen. Einerseits muss verglichen werden können, wie mit den Smart Home Gerät interagiert werden kann und zweitens muss verglichen werden können, welche Funktionalitäten ein Smart Home Gerät hat.
   Gerade der erste Vergleich ist wichtig für den *Dependency Graph Manager*, um einen Abhängigkeitsgraf erstellen zu können. Der zweite Vergleich kann für Automationen benutzt werden.

Optionale Bedingungen sind:

1. Menschen lesbare Strukturen
   Die Daten könnten in einer Art gespeichert werden, sodass ein Mensch diese Daten lesen kann. Dies ermöglicht einen schnelleren Überblick über die gespeicherten Daten und verringert das Fehlerpotenzial, wenn die Rohdaten nicht formatiert werden müssen, um vom Benutzer analysiert werden zu können.

2. Modulare Strukturen
   Die Daten sollten modular aufgebaut werden, damit auch Teile der Strukturen benutzt werden können, um Teile der Daten zu analysieren
   Dies kann die Performance verbessern, wenn zu Analyse eines bestimmten Aspektes nur Teile der Daten benutzt werden können und nicht das komplette Datenset.

3. Erweiterbarkeit der Strukturen zur Laufzeit
   Es könnte passieren, dass ein Smart Home Gerät zur Laufzeit mit neuer Funktionalität ausgestattet wird, während die Kernfunktionalität bestehen bleibt. Die bestehenden Datenstrukturen mit der neuen Funktionalität zu erweitern, anstatt nach jeder Änderung neue Datenstrukturen zu erstellen, kann fehlertoleranter sein.

4. Effiziente Speicherung der Daten
   In Systemen mit hunderten Smart Geräten kann es nötig sein, aus Performance und Speichergründen, die Daten zu komprimieren.

5. Serialisierbarkeit der Daten
   Um die Daten einfach von Komponente zu Komponente zu schicken, ist es vorteilhaft, wenn diese in bekannte Datenschemata, wie *JSON* deserialisiert und serialisiert werden können.

4.1.2 Anforderungen für die weiteren Funktionen des LDT

Der *LDT* muss neben der Modellierung der Smart Home Geräte weitere Funktionalität liefern, um ein *Smart Home System* komplett zu beschreiben. In dieser Arbeit wird sich auf drei Funktionalitäten konzentriert, die sich aus dem *LDT* und den Anforderungen an einen *Digital Twin* ableiten lassen:

1. Kommunikation zwischen den verschiedenen Komponenten. Diese Funktionalität ist ein zentraler Bestandteil des *LDT* und wird im Unterkapitel 4.2 weiter beleuchtet.

2. Erstellen der Datenstrukturen. Sobald eine Kommunikation zwischen den *LDT* und Smart Home Gerät aufgebaut wurde, müssen die Datenstrukturen erstellt werden.

3. Abhängigkeiten der Smart Home Geräte zueinander müssen von dem *LDT* erkannt werden und dem Nutzer dargestellt werden.

4.2 Kommunikation des LDT

Aus dem im Grundlagenkapitel vorgestellten Aufbau eines Smart Home Systems 2.1 lassen sich drei Kommunikationspartner ableiten, die für den *LDT* relevant sind.

> Die Hauptkommunikationspartner des *LDT* sind Smart Home Geräte. Es muss möglich sein, den Zustand von Smart Home Geräte auszulesen und zu manipulieren, falls dies eine Automation oder ein Benutzer verlangt.

Der zweite Kommunikationspartner sind Benutzer, die den *LDT* veranlassen, entsprechende Veränderungen bei einem Smart Home Geräte auszuführen, Automationen zu starten oder die Daten des *LDT* abzufragen.

Der Dritte Kommunikationspartner sind Smart Home Server. Der *LDT* muss in der Lage sein, einen Informations-Austausch mit Smart Home Server durchzuführen, um alle Daten des Systems modulieren zu können.

Bei der Kommunikation zwischen *LDT* und Smart Home Geräte kann unter drei Arten unterschieden werden. Eine direkte Verbindung zwischen *LDT* und Smart Home Gerät, bei dem auf dem Smart Home Gerät alle nötigen Funktionalitäten implementiert wurden, um den *LDT* über jede Statusveränderung direkt benachrichtigen zu können. Dafür muss neben der normalen Funktionalität des Smart Home Geräts, die Funktionalität erweitert werden, sodass das Smart Home Gerät neben der routinemäßigen Kommunikation mit dem Smart Home Server die ausgehenden Pakete auch an den *LDT* geschickt werden. Gleichzeitig muss das Smart Home Gerät auf eingehende Pakete vom *LDT* warten und im Falle eines eingehenden Befehls des *LDT*, die Nachricht analysieren, entsprechend den eigenen Zustand ändern und den Smart Home Server, sowie den *LDT* über die Veränderung des Zustandes benachrichtigen. Erst, wenn der *LDT* die Benachrichtigung der Veränderung des Zustandes erhält, dürfen die entsprechenden Daten im *LDT* aktualisiert werden. Auf diese Weise wird gewährleistet, dass der *LDT* nur Zustände abspeichert, die dem Zustand des Smart Home Geräts entsprechen.
Falls es nicht möglich ist, dass das Gerät eine direkte Verbindung mit dem *LDT* aufbauen kann, wenn es sich zum Beispiel um ein Smart Home Gerät handelt, das nicht für einen *LDT* implementiert wurde, gibt es zwei Optionen, wie ein *LDT* die Daten des Geräts mit den eigenen Daten synchronisieren kann. Die eine Option ist, falls das Smart Home Gerät eine Schnittstelle bereitstellt, um die eigenen Daten einsehbar zu machen, ist es den momentanen Zustand des Geräts aktiv abzurufen. Dies könnte zum Beispiel ein Webserver sein, der vom Smart Home Gerät gehostet wird, auf dem der Zustand der Aktoren und Sensoren bereitgestellt werden. Dieser Webserver könnte dann vom *LDT* benutzt werden, um den Zustand des Smart Home Geräts zu aktualisieren. Falls das Smart Home Gerät allerdings keinerlei Möglichkeiten hat, die eigenen Daten für eine dritte Partei bereitzustellen, wie es zum Beispiel in einem zentralen System passieren könnte, muss der *LDT* die Daten von diesen Geräten indirekt über den Smart Home Server synchronisieren.
Es lassen sich also drei Klassen an Smart Home Geräte aufstellen, die von einem *LDT* abgebildet werden müssen.

- Ein *LDT* Gerät hat alle Funktionen, die nötig sind, um eine direkte Kommunikation zwischen *LDT* und Gerät zu gewährleisten. Die Daten, die auf dem *LDT* gespeichert wurden, und von einem *LDT* Gerät stammen, haben die höchste Wahrscheinlichkeit, dass diese aktuell und akkurat sind.

- Ein dezentrales Gerät ist ein Gerät, das nicht mit einem Home Server verbunden

ist und kein *LDT* Gerät ist. Damit ein dezentrales Gerät von einem *LDT* abgebildet werden kann, muss es selber eine Schnittstelle anbieten, die von dem *LDT* abgerufen werden kann. Zum Beispiel könnte das dezentrale Gerät einen Webserver hostet, mit dem sich der *LDT* verbinden kann. Die Wahrscheinlichkeit, dass die Daten, die auf dem *LDT* gespeichert sind, durchgängig aktuell sind, ist abhängig von der Frequenz in dem der *LDT* die Daten abfragt. Es besteht die Möglichkeit, dass Datenänderungen übersehen wurden, falls die Abfragerate zu gering ist oder dass sinnlose Abfragen getätigt werden, falls die Abfragerate höher ist als die Rate in denen neue Daten generiert wurden.

- Ein zentrales Gerät tauscht seine Daten ausschließlich mit einem Smart Home Server über entsprechende Protokolle aus. Bei diesem Gerätetyp kommen die gleichen Nachteile zu Geltung wie bei einem dezentralen Gerät und es kommen Abhängigkeiten von der Smart Home Server Implementierung hinzu. Es könnte sein, dass auf dem Smart Home Server Zustände schon verarbeitet werden und abgespeichert werden und der *LDT* nur Zugriff auf diese abgeänderten Daten hat. Das könnte zum Beispiel dazu führen, dass der Zustand einer Lampe, die auf dem Gerät als Boolean gespeichert wird, auf einem Webserver als String mit dem Wert *on* oder *off* gespeichert wird und so die Präzision der auf dem *LDT* gespeicherten Daten leidet.

4.2.1 Bootstrapping

Die unterschiedlichen Geräteklassen haben auch eine Auswirkung auf den *Bootstrapping* Vorgang. Die hier benutzen *Bootstrapping* Begriffe stammen aus dem im Kapitel 3.1 vorgestellten Paper *Internet of Things Patterns for Device Bootstrapping and Registration* von Lukas Reinfurt et. al. [39]. Bei einem *LDT* Gerät ist es möglich vom Werk aus die entsprechenden Informationen wie die Adresse des *LDT* auf dem Gerät abzuspeichern und eine direkte Kommunikation zu ermöglichen, sobald das Gerät eingeschaltet wird. Bei einem dezentralen Gerät wäre ein *Factory Bootstrap* auch möglich, in dem auf dem *LDT* die Adresse des Gerätes abgespeichert wird und der *LDT* so die Daten des dezentralen Geräts abrufen kann. Allerdings hat dieses Vorgehen ein Problem mit der Skalierbarkeit und der Praktikabilität, da der *LDT* eine Vielzahl von unterschiedlichen Adressen abspeichern müsste und zur Laufzeit keine neuen Geräte hinzufügen könnte. Es muss also eine Alternative zum *Factory Bootstrap* gefunden werden, bei einem dezentralen Gerät. Für das *Bootstrapping* eines zentralen Geräts wäre wie bei einem LDT Gerät auch der *Factory Bootstrap* eine Möglichkeit. Hierbei wird dem *LDT* die Adresse des Webserver mitgeteilt. Dieses Vorgehen bietet sich an, da sich der Home Server in der Regel nicht ändert in einem Smart Home System. Eine Möglichkeit für alle drei Geräteklassen wäre ein Nutzer gesteuertes Bootstrapping, bei dem der Benutzer dem Gerät beim Einrichten die IP-Adresse des *LDTs* mitteilt oder zur Laufzeit dem *LDT* die Adresse eines dezentralen Geräts mitteilt oder die Adresse eines Webservers.

Für die *Device Registration* haben die drei Geräteklassen unterschiedlich gut funktionierende *Bootstrapping* Mechanismen. Die *Automatic Client-Driven Registration* funktioniert nur bei einem *LDT* Gerät, denn nur auf einem *LDT*-Gerät kann davon ausgegangen werden, dass auf dem Gerät die API Anfragen implementiert wurden, die dazu führen, dass der *LDT* das Gerät eindeutig registriert. Für die beiden anderen Fällen bietet sich eine *Automatic Server-Driven Registration* an. Der Server fragt die Daten eines dezentralen Geräts oder die Daten des Home Servers ab und registriert anhand der Antworten die Geräte.
Für das Modellieren der Geräte selbst bieten sich für ein zentrales und dezentrales Gerät in der Regel nur das *Server Driven Modell* an. Der Server muss anhand der Daten, die zur Verfügung stehen, ein entsprechendes Modell generieren. Die einzige Ausnahme wäre, wenn die Geräte den Standard benutzen würden, um ihre Funktionalität zu beschreiben, der auch vom *LDT* benutzt wird. Zum Beispiel könnte ein dezentrales Gerät eine *WoT-TD* Repräsentation von sich selbst abrufbar machen. Dann wäre eine *Device-Driven* Modellierung möglich. Das Gleiche gilt auch für LDT Geräte, die theoretisch so implementiert werden könnten, sodass sie ihre Funktionalitäten *LDT* konform modellieren und an den *LDT* schicken, da aber ein *LDT* eine höhere Leistung besitzt als die Smart Home Geräte, die es modelliert, ist das *Server-Driven* Modell das sinnvollste.
Eine Übersicht der unterschiedlichen Kommunikationspartner und die Verbindung dieser mit dem *LDT* und zwischen einander wird in Fig. 4.1 dargestellt. In dieser Darstellung werden auch Beispiele für verwendete Netzwerkprotokolle für diese Kommunikation angegeben. Bei dem *Smart Sensor* und dem *Smart Contact Sensor* handelt es sich um *LDT* Geräte, die direkt mit dem *LDT* verbunden sind. Die *Smart Lightbulb* hingegen ist ein zentrales Gerät, das nur mit einem Home Server verbunden ist. Die Modellierung der Geräte passiert durch den *LDT*, und die Geräte beeinflussen die Modelle. In dem abgebildeten System ist auch zu erkennen, dass es zwei Smart Home Server gibt, die beide mit dem *LDT* in Verbindung stehen.

4.2.2 Auswertung

Die drei Geräteklassen führen jeweils zu einer anderen Erhöhung der Komplexität des Designs eines Smart Home System, das einen *LDT* benutzt. Bei einem *LDT* Gerät muss das Gerät selbst mit zusätzlicher Funktionalität ausgestattet werden und so erhöht sich der Aufwand der Produktion eines jeden *LDT* Gerät. Der Aufwand für die Entwicklung des *LDT* ist Skalierung unabhängig für *LDT* Gerät, weil der *LDT* nur eine API entwickeln muss, die von jedem *LDT* Gerät benutzt werden kann, um die Kommunikation zu gewährleisten. Der Netzwerkverbrauch des Geräts vergrößert sich bei einem *LDT* Gerät verglichen mit einem zentralen Gerät, da jede Zustandsänderung sowohl an den Home Server als auch den *LDT* geschickt werden müssen. Dafür bieten *LDT* Geräte die exakteste Beschreibung ihres Zustands. Die Einschätzung eines dezentralen Geräts auf

die Produktionskosten und Aufwand ist abhängig von der Implementierung. Im besten Fall würde das dezentrale Gerät eine *WoT-TD* Repräsentation von sich selbst an den *LDT* schicken und so müsste der *LDT* kaum überarbeitet werden. Andererseits könnte ein dezentrales Gerät ein nicht standardisiertes Protokoll benutzen. Dies würde bedeuten, dass der *LDT* eine für dieses Geräts spezifische Implementation benötigt. Diese Implementation hat auch einen direkten Einfluss auf die Zuverlässigkeit der Daten und den Datenverbrauch. Bei einem zentralen Gerät verändert sich weder die Produktionskosten noch der Netzwerkverbrauch des Geräts, da die Kommunikation mit dem *LDT* von dem Webserver gehandhabt wird. Der *LDT* muss für jeden Webserver implementieren, wie er die Daten auslesen und analysieren kann und muss in regelmäßigen Abständen die Daten abfragen. Die Exaktheit der Daten ist hier basierend auf die Exaktheit des Webservers und die Abfragerate des *LDT*s. In dieser Arbeit wird sich vor allem auf *LDT* Geräte und zentrale Geräte konzentriert, da diese Implementation unabhängiger sind als ein dezentrales Gerät.

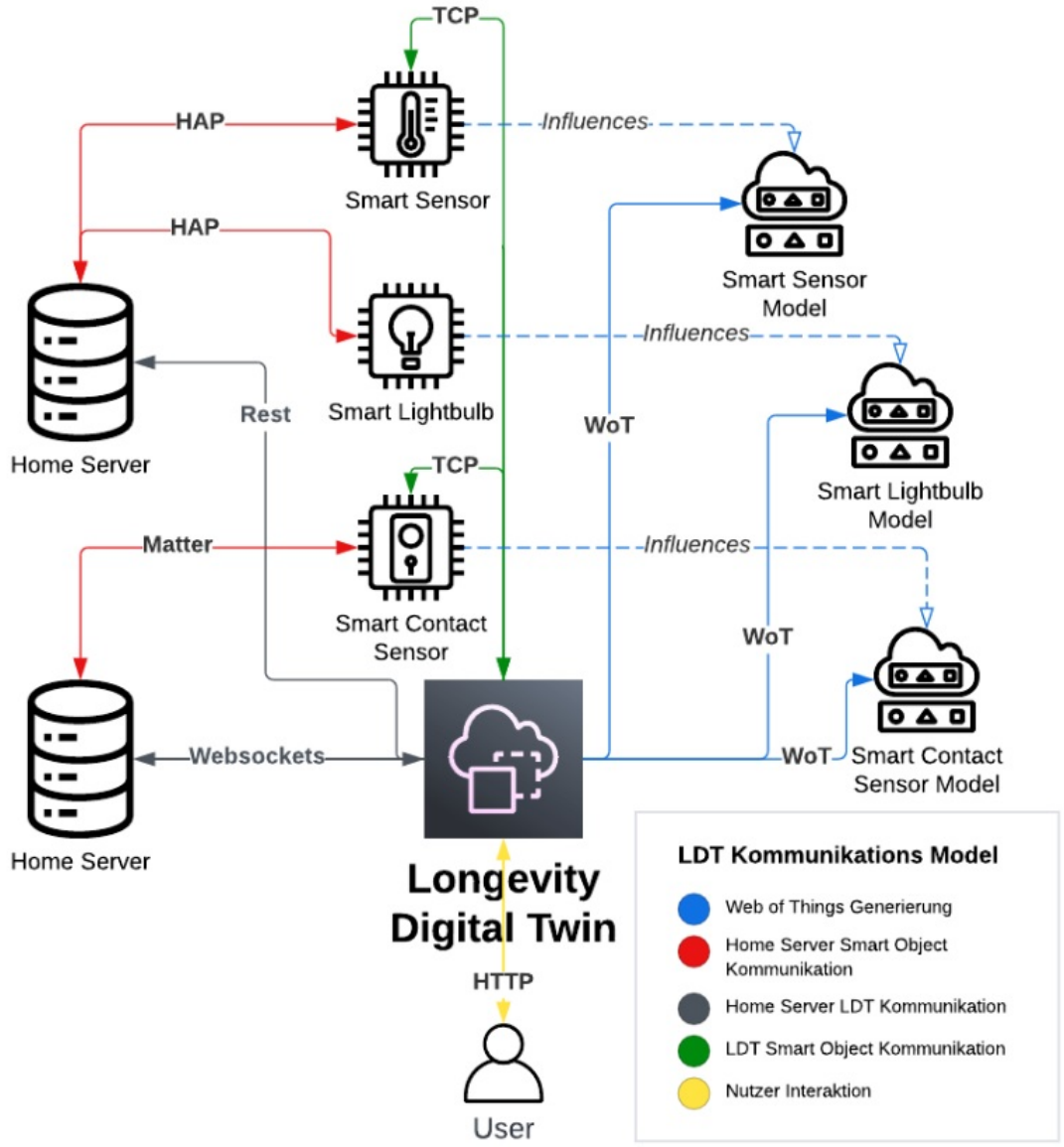


Abbildung 4.1: Mögliches Smart Home System mit einem *LDT* und beispielhaften Kommunikationspartner, sowie Protokolle, die zur Kommunikation benutzt werden. Mögliche Kommunikationspartner sind Benutzer, Smart Home Geräte und Smart Home Server. Der *LDT* erstellt anhand der Daten, die er vom Server und den Geräten, erhält, die *WoT-TD* Modelle der Geräte und verändert den Status der Smart Home Geräte, falls es der Benutzer oder eine Automation verlangt.

Kapitel 5

Longevity Digital Twin

5.1 Einführung

In diesem Kapitel werden Lösungsansätze für die Problemstellungen und Anforderungen, die in Kapitel 4 dargestellt wurden, erarbeitet.

5.2 WoT Thing Description als Datenmodell für den LDT

Die in Kapitel 2.4 vorgestellte *WoT Thing Description* verspricht ein plattformunabhängiges Datenmodell zu sein, um Smart Geräte zu beschreiben. In diesem Unterkapitel wird bewertet, ob die *WoT-TD* ein geeignetes Datenmodell ist. Dafür werden die Anforderungen, die in dem Kapitel 4.1.1 aufgestellt wurden, dahin überprüft, ob die *WoT-TD* diese Anforderungen erfüllt.
Die realitätsnahe Beschreibung des Smart Home Geräts wird in der *WoT-TD anhand* von *Properties* geregelt. Es ist möglich jede einzelne Funktionalität als ein eigenes Property zu beschreiben und anhand von den *Interaction Affordances* können Interaktionen, die auf das *Property* angewandt werden können, beschrieben werden. Dabei ist es notwendig, dass das *Smart Objekt* in klare definierbare *Properties* unterteilt werden kann. Mögliche Interaktionen von *Smart Objekten* untereinander können durch *Links* ausgedrückt werden. Es ist also möglich, das Gerät realitätsnah zu beschreiben, indem die Eigenschaften des Geräts auf die *Properties* abgebildet werden.
Die *M2M Kommunikation* und die Maschinenlesbarkeit ist durch das *WoT-TD* gegeben, da es im *JSON-LD* Format kodiert ist.
Die *WoT-TD* ist darauf ausgelegt nur abhängig von *W3C* Standards zu sein, sodass die gegebenen Bausteine, wie *Properties* und die *Interaction Affordances*, dazu benutzt werden können jegliches Gerät zu beschreiben, ohne dabei Hersteller spezifisch zu werden.
Die Vergleichbarkeit der Daten ist dadurch gegeben, dass jedes Gerät dieselbe Version des *WoT-TD* benutzt und Details der Datenformate durch die *Data Format Klasse* definiert werden können. Damit die Interaktionen der Geräte zwischen einander analysiert werden kann, kann die Interaktion über den *Link* Datentyp dargestellt werden.

```
1  "@context": ["https://www.w3.org/2022/wot/td/v1.1",
2  {
3  "HAP": "http://localhost:8080/hap"
4  }
5  ],
6  "@type": "HAP:public.hap.service.lightbulb",
7  "id": "http://192.168.192.15:2",
8  "title": "Dimmable LED",
9  "securityDefinitions": {
10   "nosec_sc": {
11   "scheme": "nosec"
12   }
13 },
14 "security": "nosec_sc",
15   "properties": {
16   ...
17 }
```

Abbildung 5.1: Aufbau des Thing Objekts eines *HAP* Geräts in der *WoT-TD*

Von den optionalen Bedingungen ist die menschliche Lesbarkeit durch das *JSON* Format gegeben, außerdem ist es durch den modularen Aufbau der *WoT-TD* möglich beispielsweise nur eine bestimmte *Property* eines Geräts zu analysieren. Das Modell kann zur Laufzeit beliebig erweitert werden und auch eine Serialisierbarkeit ist gegeben.
Die Daten der *TDs* werden durch das *JSON*-Format nicht am effizientesten gespeichert. Um eine effizientere Speicherung der Daten zu gewinnen, müsste allerdings die Menschen Lesbarkeit darunter leiden und ein Format wie *CBOR* benutzen, bei dem die Daten in Binär gespeichert werden[13].

5.2.1 Ein HAP Accessory in der WoT-TD

Wie in Kapitel 2.2.1 beschrieben, besteht ein *HAP Accessory* aus einem Service mit mehreren Charakteristiken. In diesem Kapitel wird eine *HAP* konforme Glühbirne, die ihre Lichtstärke verändern kann, in *WoT* moduliert. Eine Glühbirne ist in dem *HAP* der *Service Lightbulb* mit der verpflichtenden *Characteristik ON*, um die Lichtstärke auch noch zu beschreiben, muss die weitere *Charakteristik Brightness* mit moduliert werden. In Fig. 5.1 ist die *HAP* Glühbirne im *WoT* Standard dargestellt.

Damit Begriffe wie *Brightness* und *ON* so gekennzeichnet werden, dass gezeigt wird, dass diese aus der *HAP* Spezifikation stammen, wird der *HAP*-Kontext in Zeile 5 definiert und die entsprechenden Begriffe werden mit dem Präfix *HAP* erweitert. Die angegebene

URL weist daraufhin, wo die *HAP*-Spezifikation zu finden ist, zum Beispiel lokal auf einem Server. Die *ID* muss eindeutig sein und besteht in diesem Beispiel aus der IP-Adresse des Gerätes, da eine *URI* verlangt wird und eine IP-Adresse eine *URI* ist. Die Zeilen 11-16 beschreiben, dass dieses *Thing* keine Security benutzt.

Die *Properties*, die Teil des *Thing* sind, werden genauer in Fig. 5.2 aufgeführt. Der Type der beiden *Properties* entstammt der *HAP* und sind dementsprechend markiert. Der Datentyp der *Properties* wird im *type*-Feld angegeben und stammen wie die minimal und maximal Werte der *Brightness Property* auch aus der *HAP*. Im *forms*-Feld wird angegeben, welche direkte Interaktionen mit dieser Property möglich sind und mit welcher *URI* diese Interaktion aus geführt werden kann. Dabei wird mit den *readproperty* und *writeproperty* das Schreiben und Lesen dieser angegeben. Die durch Punkte gekennzeichneten fehlenden Zeilen der *writeproperty* funktioniert analog zur *readproperty* und die fehlenden Werte der *Forms* der *On* Property besteht aus einer weiteren *readproperty*. Diese Felder wurden für die Übersicht entfernt.
Das *HAP*-Objekt stellt außerdem, eine Verbindung zwischen diesem und einem anderen *Things* dar. Die in 5.2 angegebene *Brightness* Property wird von dem gemessenen Lichtwert von einem anderen *Thing* kontrolliert. Das wird durch das *links*-Feld in Zeile 17-21 definiert. Durch *rel* wird angeben, um welche Form von Beziehung es sich handelt und durch die Angeben *URI* wird angegeben, was die kontrollierende Komponente ist.
Ein weiterer Aspekt der durch die *Web of Things TD* beschrieben werden kann, sind Situationen, bei dem ein Gerät aus mehreren Teilgeräten besteht. Das könnte zum Beispiel ein *HomeKit Accesory* sein, das nicht nur die Glühbirnen *characteristic* besitzt, sondern auch die *AccessoryInformation characteristic*, die Metadaten über ein Gerät darstellt. Solche Geräte können in der *Web of Things TD* über *Collections* dargestellt werden. Eine *Collection* ist eine *Web of Things TD,* die *links* benutzt, um zu beschreiben, welche Geräte Teil dieser *Collection* sind. Dieses *WoT-TD* hat einen *link* zu jedem Gerät der *Collection* mit dem *relType item*. Jedes *Item* muss daraufhin ergänzt werden mit einem Link, der durch den *reltype collection* angibt, dass dieses *Thing* Teil einer *Collection* ist. Auf diese Art und Weise können *HAP Accessories* dargestellt werden, die aus mehreren Services bestehen.

5.2.2 Ein Matter Device in der WoT-TD

In Fig. 5.3 wird eine Glühbirne, mit derselben Funktionalität wie im *HAP* Beispiel, in der Matter Spezifikation dargestellt. Es lässt sich sofort erkennen, dass der Aufbau des *WoT-TD* identisch ist, nur dass die Bezeichnungen der *Properties* unterschiedlich sind. Außerdem muss für ein *Matter Device* zwei Kontexte gegeben werden, da die *Matter* Spezifikation in unterschiedliche Dokumente aufgeteilt ist. Der Begriff der *Dimmable Light* stammt aus der *Device Libary* und genaue Beschreibungen des *On/Off Clusters* und der *Level Control* entstammen dem *Cluster* Dokument. Dieses Modell wurde auch um einen Link erweitert, der beschreibt, dass dieses Gerät eine Abhängigkeit von einem

```
"HAP:Brightness": {
  "@type": "HAP:public.hap.characteristic.brightness",
  "type": "integer",
  "forms": [
    {
      "op": "readproperty",
      "href": "http://localhost:8080/status/Dimmable_LED/HAP
         :Brightness",
      "contentType": "application/json"
    },
    {
      "op": "writeroperty",
      ...
    }
    ],
    "minimum": 0,
    "maximum": 100,
    "links ": [
      "href": "http://localhost:8080/status/Light_Sensor/HAP
         :Light Level",
      "type": "application/json",
      "rel ": "controlledBy"
    ]
},
"HAP:On": {
  "@type": "HAP:public.hap.characteristic.on",
  "type": "boolean",
  "forms": [
    ...
  ],
}
```

Abbildung 5.2: Aufbau der Properties eines *HAP*-Geräts in der *WoT-TD*

```
{
  "@context": [
  "https://www.w3.org/2022/wot/td/v1.1",
  {
    "MTC": "http://localhost:8080/mtc",
    "MTD": "http://localhost:8080/mtd"
  }
  ],
  "@type": "MTD:Dimmable Light",
  "id": "http://192.168.192.123:1",
  "title": "Dimmable Light",
  ...
  "properties": {
    "MTC:Level Control": {
    "@type": "MTC:Level Control",
    "type": "integer",
    "forms": [
      {
        "op": "readproperty",
        "href": "http://localhost:8080/status/Dimmable_Light
           /MTC:Level Control",
        "contentType": "application/json"
      },
      ...
    ],
    ...
    "links ": [
      {
        "href": "http://localhost:8080/status/Light_Sensor/
           HAP:Light Level",
        "type": "application/json",
        "rel ": "controlledBy"
      }
    ]
    },
    "MTC:OnOff": {
    ...
    }
  }
}
```

Abbildung 5.3: Aufbau einer *WoT-TD* eines *Matter* Gerätes

HAP-Lichtsensor hat. Es lässt sich also eine Abhängigkeit, unabhängig vom Protokoll, beschreiben.
Die angegebenen *WoT-Objekte* wurden in der vollständigen Form mit dem *Thing Description Playground*[38] überprüft und es wurde damit verifiziert, dass diese Modelle syntaktisch korrekt ist.

5.2.3 Erweiterung der Modelle im Falle von mehreren Integrationen

Falls ein Gerät nicht nur über den *LDT* angesprochen werden kann, sondern das Modell auch von einem Smart Home Server angesprochen wird, müssen die *Forms* erweitert werden (siehe 5.4.3). In Fig. A.1.1 ist ein vollständiges *HAP* Objekt dargestellt, das nicht nur über den *LDT* angesprochen werden kann, sondern auch in einen *HomeAssistant* Webserver eingebunden ist. Der größte Unterschied im Vergleich zu dem vorherigen Modell ist die Erweiterung der *securityDefinitions*. Dadurch dass der *Home Assistant* Webserver *bearer tokens* benutzt, muss die *securityDefinition* erweitert werden. Dabei steht das Format für die Benutzung von *JSON Web Tokens* und das *alg* Feld für den benutzen Sicherheitsalgorithmus. Dadurch, dass nun zwei unterschiedliche Sicherheitsmechanismen benutzt werden, je nachdem, ob mit dem *LDT* oder dem *Home Assistant* Webserver interagiert wird, muss jede *form* der *Properties* erweitert werden. Das *security* Feld gibt nun an, welche *Security* erwartet wird, um diese Interaktion zu benutzen.

5.2.4 Eine HomeAssistant Entity in der WoT-TD

Im Falle eines Smart Home Gerätes, das auf einem Webserver abgespeichert ist, das allerdings keine direkte Verbindung zu dem *LDT* hat, muss eine *WoT-TD* anders erstellt werden (siehe 5.4.3). In diesem Unterkapitel wird eine solche Modellierung anhand einer *Home Assistant Entity* beschrieben. In Fig. A.1.2 wird eine solche Entität dargestellt. Zur Erstellung dieses Modells wurde bei einem *Home Assistant* Server eine *HAP* Glühbirne registriert und es wurde der Status dieser Glühbirne über einen Webserver ausgelesen. Dies ermöglicht einen genauen Vergleich zwischen dem *HAP* basierten *WoT* Modell und dem *Home Assistant* basierten *WoT* Modell, des gleichen Objektes.
Eine *entity* wird durch ein *state*-Objekt beschrieben. Das *state*-Objekt besitzt das *state*-Feld, welches den Status des Hauptattributes angibt[8]. Dieser *state* ist nicht eindeutig. Im Falle dieser Glühbirne beschreibt der *state*, ob diese Lampe an oder aus ist[5]. Im Falle eines Lichtsensors würde dieser state den letzten gemessenen Lichtwert beinhalten[7]. Dies zeigt, die Wichtigkeit des *Context* und das *@type* Feld des Thing Objektes selbst ist, um ein Objekt genau zu beschreiben. Der Kontext ist in diesem Fall die *Integrations* von *Home Assistant* und *light* Typ identifiziert das *Thing* als *light* Integration. Der *brightness* Wert ist eine Zahl zwischen 1 und 255.

5.2.5 Auswertung

Das Darstellen des funktional gleichen Smart Home Geräts in unterschiedlichen Repräsentationen zeigt, dass jede *WoT-TD* im Zusammenhang des Kontextes genau beschreiben kann, wie ein Gerät aufgebaut ist. Der *LDT* muss aber für jedes *WoT-TD* entsprechend dem Kontext unterschiedliche Daten unterschiedlich interpretieren. So ist zum Beispiel die *brightness Property* eines *Home Assistant Light* und die *brightness Property* einer *HAP Lightbulb* nur im Kontext vergleichbar. Es kann zum Beispiel der Helligkeitswert des *Home Assistant Light* in Prozent umgerechnet werden, um diesen mit einer *HAP Lightbulb* zu vergleichen. Der Kontext ist auch relevant, um analysieren zu können, was die unterschiedlichen *Properties* genau beschreiben. So ist es nur durch den Kontext möglich, die *Properties MTC:OnOf, HAP:On* und *HAS:state* zu vergleichen. Der *LDT* muss also je nach Integration immer auch die entsprechenden Konvertierungen vollführen, um die Werte zu vergleichen und der *LDT* muss erkennen, welche *Properties* sich gegenseitig entsprechen. Es wäre auch möglich, diese drei Objekte in einer standardisierten Form abzuspeichern und beispielsweise jeweils die drei unterschiedliche Repräsentationen von der Helligkeit einer Glühbirne als eine *LDT Brightness Property* zu modellieren. Das hätte den Vorteil, dass ein direkter Vergleich der *Thing* Objekte möglich wäre ohne den Kontext zu beachten, allerdings würde das dem Ziel widersprechen, eine möglichst genaue Repräsentation des Smart Home Geräts zu schaffen. Im Kontext von *Digital Twins* ist eine genaue Repräsentation der Geräte wichtiger als der höhere Aufwand der Verarbeitung der Daten.
Die *WoT-TD* eignet sich also als Datenmodell für den *LDT*. Es werden alle wichtigen Daten repräsentiert und der Kontext ist eine gute Möglichkeit gerade für *Digital Twins* Datenmodelle zu bauen, die ein Smart Home Gerät, so genau wie möglich dem Protokoll entsprechend zu beschreiben.

5.3 Dependency Graph

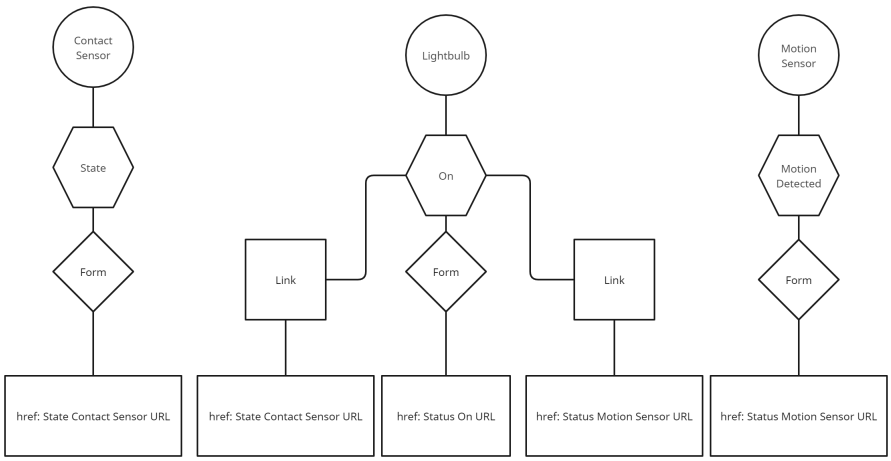
Damit der *Longevity Digital Twin* fehlerhafte Konfigurationen und fehlerhafte Statusänderungen erkennt, ist es nötig, die Abhängigkeiten zwischen den unterschiedlichen Smart Home Geräte analysieren zu können. Dafür bietet sich ein Graph an. Ein Graph ermöglicht es, die Abhängigkeiten zwischen den Geräten für den Nutzer zu visualisieren und für den *Twin* analysierbar zu machen. Um den Graphen erstellen zu können, muss in den *Digital Twin* Datenstrukturen eine Abhängigkeit zwischen den Geräten dargestellt werden können, was durch die *Links* der *WoT-TD* gegeben ist. Der Graph baut auf den Grundgedanken von *IoTSan* auf, Abhängigkeiten durch Interaktionen abzubilden und abhängige Untermengen zu bilden aus[36]. Dieser Ansatz wird mit der *WoT-TD* in Verbindung gebracht, um einen Graphen zu erstellen, der Abhängigkeiten zwischen Geräten anhand der Interaktionen, die durch die *WoT-TD* beschrieben werden, abzubilden.

5.3.1 Design des Graphen

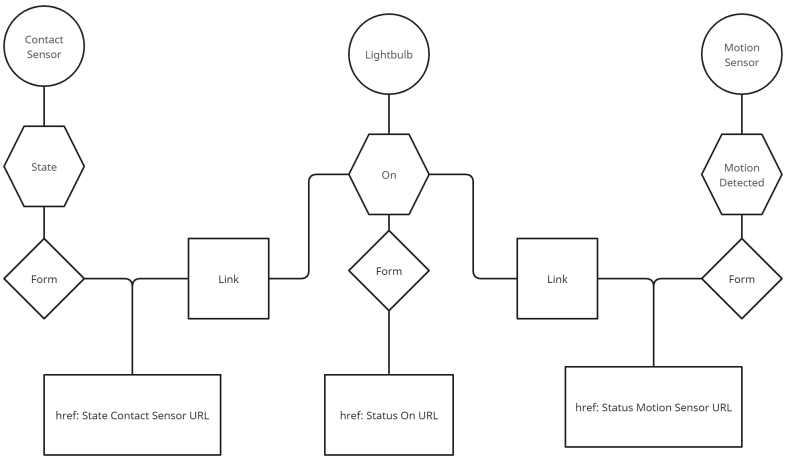
Der Fokus beim Design des Graphen ist es, die Interaktionsmöglichkeiten des Smart Home Geräts in Abhängigkeiten der *Properties* darzustellen. Der Grund dafür ist, dass *Links, Forms, Events* und *Properties* alle Interaktionsmöglichkeiten eines Geräts beschreiben. In Fig. 5.4(a) wird ein einfaches Smart Home-System, das durch eine *WoT-TD* beschrieben wurde, in Form von einem Graphen dargestellt. Das System besteht aus drei Smart Home Geräten: einem Kontaktsensor, einer Glühbirne und einem Bewegungssensor. In diesem System gibt es eine Automation, die die Glühbirne anhand des Kontakt-Sensors und des Bewegungssensors reguliert. Dafür wird der Status des Kontaktsensors und des Bewegungssensors regelmäßig abgerufen und darauf hin an oder aus geschaltet. Diese Verbindung wird durch *Links* der *ON Property* dargestellt. Dadurch, dass die *Links* und die *Forms* die gleiche URL haben, können die doppelten Knoten aus dem Graphen gelöscht werden und die Kanten neu verbunden werden, wie in Fig. 5.4(b) zu sehen. Auf diese Art ist die Verbindung der Geräte im Graphen direkt zu erkennen. Dieser Graph kann daraufhin reduziert werden. Es kann wie in Fig. 5.4(c) der Graph auf die wichtigsten Felder reduziert werden, die für die Beziehungen relevant sind. Dabei handelt es sich um die *Properties* und das Gerät selbst. Die Links können durch gerichtete Pfeile ersetzt werden, da es laut der *WoT-TD* nur eine *controlledBy* Beziehung gibt, wenn eine Interaktion zwischen zwei TD beschrieben wird. Es wird also immer die *Property*, die einen *Link* hat, von einer anderen *Property* beeinflusst.

5.3.2 Erkennbare Fehler durch den Graphen

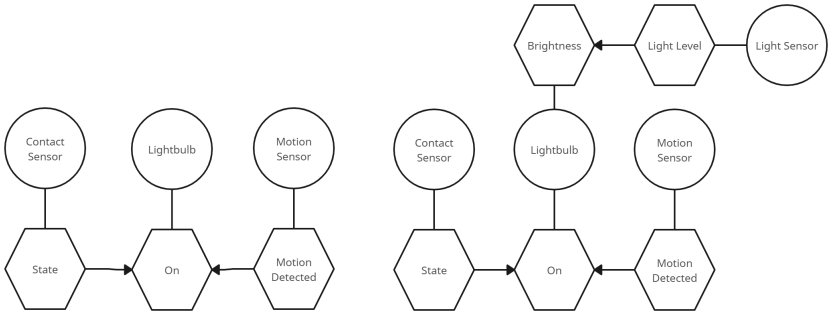
Der Graph ermöglicht es in der reduzierten Form klar zu erkennen, welche *Properties* eine andere Property beeinflussen. Ein weiteres Beispiel ist in Fig. 5.4(d) gegeben. In diesem Beispiel wurde das System um einen Lichtsensor erweitert, der die Helligkeit der Glühbirne reguliert. Aus diesem Graph lassen sich zwei Abhängigkeiten erkennen. Die Glühbirne ist abhängig von drei anderen Geräten, da die *Properties* insgesamt drei eingehende Kanten besitzt. Die andere Abhängigkeit ist eine direktere Abhängigkeit der *ON Property*, da diese von zwei Geräten beeinflusst wird und dadurch gibt es auch eine potenzielle Abhängigkeit von dem Kontaktsensor und dem Bewegungssensor. Durch dieses Wissen kann der *LDT* beispielsweise Automationen gezielter kontrollieren. Ein Beispiel wäre, wenn die Automation des Kontaktsensors das Licht ausschalten würde, der Bewegungssensor aber noch eine Bewegung misst und das Licht eingeschaltet bleiben soll. Der *LDT* kann durch den Graphen diese Abhängigkeit erkennen und entsprechend intervenieren.
Eine weitere Fehlerquelle, die durch den Graphen visualisiert wird und ausgewertet werden kann, sind sich verändernde Kommunikationsparameter der Geräte. Wenn sich zum Beispiel die URL des Kontaktsensors ändern würde, da das Gerät ausgetauscht wurde



(a) Drei *TD* als Graphen



(b) Verbinden der Graphen anhand des *href*-Feldes



(c) Reduzierter Graph

(d) Reduzierter Graph mit einer zusätlichen Abhängigkeit

Abbildung 5.4: Kompletter Graph von drei Abhängigen *WoT-TDs* und seine Reduzierungen

oder sich das Protokoll verändert hat und die ausführende Automation nicht aktualisiert wurde, wird anhand des Graphen dieser Fehler erkennbar sein, da der Kontaktsensor nicht mehr mit dem *Link* der *On Property* verbunden wäre.

5.3.3 Aktualisierung des Graphen

Der Graph muss nach jedem Update eines Gerätes aktualisiert werden und es müssen alle Kanten neu aufgebaut werden, um sicherzustellen, dass alle Verbindungen exakt dargestellt werden.

5.4 Kommunikation

In diesem Kapitel werden auf alle unterschiedlichen Kommunikationspartner eingegangen und Kommunikationsmodelle für jeden Partner entworfen.

5.4.1 Kommunikation zwischen LDT und ein LDT Gerät

Für die Kommunikation zwischen *LDT* und Smart Home Gerät bietet es sich an, die *TCP/IP* Protokolle zu benutzen, da Smart Homes teil des *IoTs* ist und so davon ausgegangen werden kann, die für das *LDT* benutzte Smart Home Geräte *IoT*-Geräte sind. Der Einfachheit halber wird auch davon ausgegangen, dass sich der *LDT* und die *IoT*-Geräte im selben Heimnetzwerk befinden. In der Praxis könnte es sein, dass der *LDT* zum Beispiel in einer Cloud liegt und sich die Smart Home Geräte übers Internet mit diesem verbinden.
Aufgaben des *LDTs*:

1. Der *LDT* agiert als Server im Netzwerk, der auf eingehende Verbindungen wartet
2. Der *LDT* reagiert auf Nachrichten des Smart Home Geräts. Dabei können zwei Fälle unterschieden werden.
    a) Registrierung des Smart Home Geräts beim *LDT*.
    Der *LDT* muss das Smart Home Gerät intern eindeutig abspeichern und alle relevanten Daten des Geräts als Modell erstellen.
    b) Updates des Status des Geräts beim *LDT*.
    Der *LDT* muss intern die eingehenden Daten dem entsprechenden Modell des Smart Home Gerät zu ordnen.
3. Der *LDT* schickt Änderungsanfragen an das Smart Home Geräts.

Das Smart Home Gerät muss angepasst werden, sodass es die nötige Funktionalität liefert, um sich neben dem normalen Ablauf seiner Routinen, gleichzeitig mit dem *LDT* zu verbinden, auf die *LDT* Nachrichten zu reagieren und den *LDT* über Veränderungen in Kenntnis zu setzen.
Ein wichtiges Problem, um die Verbindung zwischen *LDT* und Smart Home Gerät zu gewährleisten, ist das *Bootstrapping*-Problem und die Identifizierung des Smart Home Geräts. Das Smart Home Gerät muss die Adresse des *LDT* wissen, um eine Verbindung aufzubauen. Der Server muss durch die eingehenden Nachrichten das Smart Home Gerät eindeutig zuordnen können und die entsprechenden Strukturen aufbauen, die das Gerät genau beschreiben. Bei dem Herausfinden der richtigen Adresse des *LDT* kann der *Factory Bootstrap* benutzt werden. Es kann also auf dem Smart Home Gerät die entsprechende IP-Adresse des *LDT* im Programm eingebaut werden. Diese Adresse könnte in der Praxis ein *Remote Server*, der weitere *Bootstrapping* Mechanismen ermöglicht oder die direkte Netzwerk-Adresse des *LDTs*. Für die richtige Modellierung des Smart Home Geräts bietet sich das *Server-Driven*-Model an, um die Daten, die zwischen *LDT* und Smart Home Gerät geschickt werden müssen zu reduzieren. Es muss also möglich sein, mit möglichst wenig Informationen das Gerät zu identifizieren.

Für die Kommunikation eines *HAP*-Geräts sind sechs Informationen nötig.

1. Die *Accessory Instance IDs(AID)*, ist eine von dem *HAP* definierte ID, die es ermöglicht, zwischen *Accessories* zu unterscheiden.
2. *Service Count*, ist ein Zähler, der ermöglicht zu unterscheiden, falls ein *Accessory*, mehrmals den gleichen Service hostet.
3. Die *IP*-Addresse des Geräts, ist nötig, um auf dem *LDT* jedes Smart Home Gerät eine Adresse zuzuordnen und die unterschiedlichen *TCP*-Verbindungen zu organisieren.
4. Die *UUID* des *Services* oder *Characteristic*. Der *LDT* kann anhand der *UUID* die entsprechenden Datenstrukturen aufbauen und so das *HAP*-Gerät modulieren
5. Der Wert einer Statusänderung, um bei *Updates* das Modell zu aktualisieren.
6. Der Name, mit dem sich das HAP-Gerät identifiziert.
   Der Name ist notwendig, um das Gerät bei der Kommunikation mit einer dritten Partei, wie einem Home-Server, identifizieren zu können und muss nur bei der Registrierung des Gerätes angegeben werden.

Die *AID* im Zusammenhang mit der IP-Adresse ermöglichen die genaue Identifizierung des *Accessory* eines *HAP* Gerätes. Die *AID* alleine ist nicht ausreichend, da es mehrere *HAP* Instanzen geben könnte, die sich mit dem *LDT* verbinden und die IP-Adresse alleine ist nicht ausreichend, wenn der *LDT* zwischen unterschiedlichen

Accessoires, die über die gleiche Verbindung kommunizieren, unterscheiden muss. *IP*-Addresse und *AID* alleine reichen allerdings auch nicht aus, falls ein *Accessory* eines Gerätes mehrmals denselben Service hostet. Über das Netzwerk müssen alle Parameter außer die IP-Adresse geschickt werden, da der *LDT* die IP-Adresse eingehender Verbindungen selbst erkennen kann und so die eingehenden Nachrichten richtig zuordnen.

Für die Kommunikation eines *Matter* Geräts sind ähnliche Informationen nötig

1. Die *Endpoint IDs* sind vergleichbar der *AID* des *HAP* und werden aus demselben Grund benötigt.

2. IP-Adresse ist auch analog zum *HAP*

3. Die *ID*s des *Device Types* oder *Clusters* und dazu der Name des entsprechenden Dokuments
   Da die IDs bei der *Matter* Spezifikation nicht eindeutig sind, da es mehrere Dokumente gibt, kann es dieselbe ID in der *Device Libary* und auch in der *Cluster Spezifikation* geben. Es muss also zwischen *Cluster* und *Device* differenziert werden.

4. Der Wert eine Statusänderung ist analog zum *HAP*

5. Name des Geräts ist auch analog zum *HAP*

Es muss also beachtet werden das für das *Matter* Protokoll, entweder ein weiterer Parameter hinzugefügt wird, um zu Kennzeichen das es sich um eine Registrierung oder ein Status Update handelt oder die zuständige Spezifikation mit angegeben wird. Da jeder Endpoint in der Regel aus einem *Matter* Device besteht, benötigt es keinen *Service Count*.

Daraus ergibt sich, dass je nach Protokoll ein anderes Datenschema benutzt werden muss.
Ein *HAP* Paket besteht aus:
(Protokoll-ID, ID des Services/Characteristic, AID, Service Count, Name/Wert).
Ein *Matter* Paket besteht aus:
(Protokoll-ID, ID des Device Types/Cluster, ID des Endpoints,Name/Wert)

Die Protokoll-ID ist dabei ein für den *LDT* eindeutiger Identifizieren für das entsprechende Protokoll. Mit dieser Protokoll-ID lässt sich auch das Problem der uneindeutigen *IDs* der *Matter*-Spezifikation lösen, da durch die *ID* gekennzeichnet werden kann, ob es sich um ein *Matter-Cluster* oder ein *Matter-Device* handelt.

5.4.2 Kommunikation zwischen LDT und Benutzer

Bei der Kommunikation zwischen *LDT* und Nutzer muss der *LDT* eine Schnittstelle bieten, die es ermöglicht Benutzer gesteuerte Veränderungen bei Smart Home Geräten auszulösen und den Status des *LDT* abfragen zu können.
Relevante Abfragen und die entsprechenden Antworten sind:

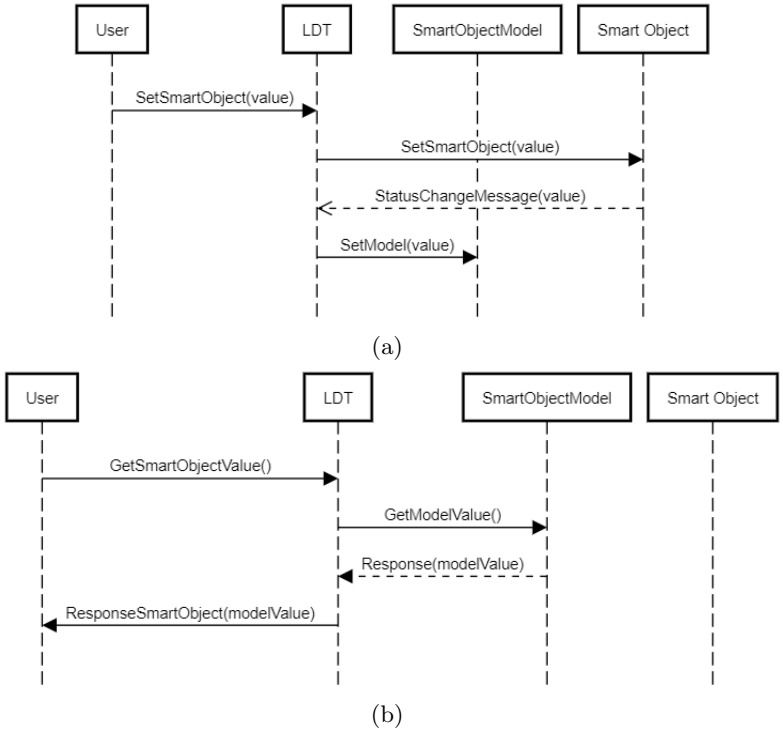


Abbildung 5.5: (a) Ein benutzergesteuertes Update (b) Abfrage an den Server

1. Statusabfrage einer Eigenschaft des *LDT*
   Gibt den momentanen Status als Wert zurück, dabei kann der *LDT* direkt auf das Modell zugreifen und den momentanen Wert abrufen, ohne dabei mit dem Smart Home Gerät zu kommunizieren (siehe Fig. 5.5(b).

2. Abfrage einer *LDT*-Instanz eines Smart Home Gerät
   Gibt das Modell der *LDT*-Instanz zurück, z.B. als *WoT-TD*

3. Anzeige der Verbindungen der *LDTs*
   Gibt alle Bekannten Verbindungen zwischen *LDTs* zurück z.b. in Form eines visualisierten Graphen

4. Abfrage einer Automation
   Gibt die entsprechende Automation zurück

5. Validierung, ob die Daten des *LDT* mit den Daten eines Home-Servers übereinstimmen

Relevante Anfragen und *LDT* interne Reaktionen sind:

1. Updaten eines Smart Home Gerät
   Wichtig dabei ist, dass der *LDT* bei Anfragen des Benutzers nicht direkt die entsprechenden Felder des Modells ändert, sondern eine Anfrage an das Smart Home Gerät schickt und die Statusänderung des Smart Home Geräts an den *LDT* geschickt wird. Dieser Prozess wird in Fig. 5.5(a) visualisiert. Dies ist nötig, damit der *LDT* im Modell nur Daten hat, die konsistent sind.

2. Generierung einer Automation
   Der *LDT* generiert eine Automation und startet eine Routine, die diese ausführt, dabei werden die Modelle aktualisiert, um die Abhängigkeit der *LDTs* zueinander darzustellen

5.4.3 Kommunikation zwischen LDT und Smart Home Server

In einem System, bei dem der *LDT* nicht der einzige Server ist, der die Smart Home Gerät organisiert, muss der *LDT* in der Lage sein, sich mit einem Smart Home Server in Verbindung zu setzen, um Daten abzugleichen. Die drei Hauptaufgaben dieser Kommunikation sind:

1. das Modell der Smart Home Geräte zu aktualisieren, um die Verbindung zwischen Smart Home Gerät und Smart Home Server zu modellieren,

2. Smart Home Geräte erkennen, die Teile des Smart Home-Systems sind, die aber noch nicht bei dem *LDT* registriert sind,

3. Fehler erkennen: Wenn der Status, der im *LDT* abgespeichert ist, nicht mit den Daten, die auf dem Home-Server liegen, übereinstimmen und den Benutzer darüber in Kenntnis zu setzen.

Die Kommunikation zwischen *LDT* und Smart Home-Server ist Implementation abhängig, da jeder Smart Home-Server eigene Methoden besitzt, wie mit diesem interagiert werden können. *Home Assistant*, beispielsweise, einer Open-Source Implementation für Smart Home-Automation [10], bietet hier für eine *Rest-* und eine *Websocket-API*, um mit dem Server zu interagieren[6][9]. Diese können vom *LDT* benutzt werden, um Daten abzufragen und Daten auf dem Server zu aktualisieren. Der Ablauf einer Kommunikation mit einem *Home Assistant* Webserver kann wie folgt aussehen:

1. Authentifizieren des *LDT* für *Home Assistant*.
   Home Assistant benutzt als Sicherheitsmechanismen Access Token, das bedeutet bevor irgendeine Kommunikation zwischen *LDT* und *Home Assistant* stattfinden kann, muss dem *LDT* ein *Access Token* übergeben werden, womit er sich authentifizieren kann.

2. Aufrufen von *GetStates*, um alle *States* von *Home Assistant* zu erhalten.
   Der *GetStates* Befehl bietet die Möglichkeit durch die API alle *States* von jeder *Entity* von *Home Assistant* zu erhalten. Dabei sind *Entities* der Grundbaustein jeder Komponenten von *Home Assistant*, die dann Form erweitert werden können, um unterschiedliche Gerätetypen darzustellen. Standardisierte *Entities* wie *Light* werden durch *Integrations* definiert[4].

3. Parsen der *States*. Bei dem Parsen der *States* können die vorher drei unterschiedlichen Fälle angewandt werden.

   - Fehlerüberprüfung
     Die *States* werden untersucht nach *Entities*, die auch auf dem *LDT* abgespeichert sind und ihr Status wird überprüft und abgeglichen.

   - Anpassung der *LDT* Modelle
     Die Modelle, die im *LDT* abgespeichert wurden und durch eine direkte Verbindung mit einem *LDT* Gerät erstellt wurden, müssen angepasst werden, falls, diese Geräte auch eine Verbindung mit dem Webserver verbunden werden, um diese Geräte korrekt zu modellieren. Diese Verbindung mit dem Server kann ausgedrückt werden, indem die *Forms* der *Properties*, die erkannt wurden, aktualisiert werden. Siehe 5.2.3

   - Erstellung von *Home Assistant WoT-TD*
     Falls beim Parsen *Entities* erkannt werden, die Smart Home Geräte entsprechen, die vom *LDT* modelliert werden sollen, muss der *LDT WoT-TDs* von diesen Geräten erstellen. Siehe 5.2.4

5.5 Erstellen eines Smart Home Geräts auf einem ESP32

Der *ESP32* ist ein 32-Bit Mikrocontroller der Firma *Espressif*. Der *ESP32* bietet Wi-Fi und Bluetooth Funktionalität und lässt sich mit der *Arduino*-Bibliothek programmieren[18] [19]. Mit dem *ESP32* ist es möglich, ein *HAP* kompatibles Smart Home Gerät zu erstellen und es mit einem *HAP* kompatiblen System zu verbinden. Dafür kann die *HomeSpan*-Bibliothek benutzt werden.

5.5.1 Arduino Bibliothek

Der Aufbau eines Programms auf dem *ESP32* mit der *Arduino*-Bibliothek besteht aus zwei Funktionen, die Setup und die Loop-Funktion. Die Setup Funktion wird einmalig beim Start des Mikrocontrollers aufgerufen. Sie wird dazu benutzt, um alle nötigen Funktionalitäten des Mikrocontrollers für den späteren Programmablauf zu aktivieren [3], wie zum Beispiel die Konfiguration des Wi-Fi-Chips, des Pins, der eine LED ansteuert oder die Erstellung von Datenstrukturen. In der Loop-Funktion wird in einer dauerhaften Schleife der standardmäßige Programmablauf des Programms geregelt[2]. Dies könnte zum Beispiel das Ablesen und Versenden von Sensordaten sein.

5.5.2 HomeSpan

HomeSpan ist eine *Open Source* Bibliothek für den *ESP32*. Sie bietet eine vollständige Implementation des *HAP*-Protokolls und ermöglicht, *HAP-Accesories* zu erstellen und diese bei einer *HAP-Bridge* zu registrieren. Die Hauptfunktionsweise der Bibliothek ist es, neue Accessoires zu erstellen, diese mit allen nötigen Informationen zu füllen und darauf hin die *HomeSpan poll* Funktion aufzurufen, die einen geregelten Ablauf des *HAP*-Protokolls gewährleistet. Es werden Updates von einem registrierten Hub abgerufen und Updates des Mikrocontrollers gesendet[27].

5.5.3 Bootstrapping mit dem HomeSpan Command Line Interface(CLI)

Die Kommunikation mit dem *LDT* und dem *ESP32* muss in irgendeiner Form gestartet werden, wie schon vorher erörtert, bietet es sich an für einen *LDT* Gerät auf dem *ESP32* an den *Factory Bootstrap* zu implementieren, also auf dem Gerät die Netzwerkadresse eines Lokales Netzwerkes und die *IP*-Adresse des *LDT* abzuspeichern. Das Problem ist, dass das Abspeichern der Netzwerk-Informationen im Source Code sicherheitstechnisch eher ungeeignet ist. Deswegen empfiehlt *HomeSpan* die Benutzung der *HomeSpan CLI*, um einem Gerät die Informationen über das Netzwerk zu geben, in dem sich eingeloggt werden soll. Die *HomeSpan CLI* erlaubt über den *Serial Monitor* Befehle an das Gerät zu schicken. Dadurch ist es möglich, sich über den *Serial Monitor* in W-LAN Netzwerke einzuloggen. Nachdem sich über den *Serial Monitor* sich mit den entsprechenden Befehlen in das W-LAN Netzwerk eingeloggt wurde, werden diese Daten im Flash gespeichert und das Gerät startet neu. *HomeSpan* bietet auch an, eigene *CLI* Befehle zu implementieren. Es lässt sich also die *CLI* erweitern, sodass mit der Benutzer beim Einrichten des Geräts um die W-LAN Daten zu übermitteln auch die Möglichkeit geben kann durch den *Serial Monitor* dem Gerät die IP-Adresse des *LDTs* mitzuteilen. Die IP-Adresse und der Port kann dann auch im *Non-volatile Storage*[20] gespeichert werden, sodass die Adresse des *LDTs* auch beim neuen Starten des Geräts gespeichert bleibt.

5.5.4 Ansatz für die Implementierung eines dezentralen Geräts

Um ein dezentrales Gerät zu implementieren, kann die Webserver Implementation von der *ESP32 Aurduino* Bibliothek benutzt werden[19]. Es muss eine Verlinkung für jeden einzelnen Zustand, den das Gerät haben kann, zu einer Webadresse erfolgen, die dann extern von dem *LDT* abgerufen werden kann. Damit dem *LDT* alle Funktionen des Geräts selbst mitgeteilt werden können, würde sich anbieten, dass das dezentrale Gerät eine *WoT-TD* von sich selbst erstellt, inklusive allen *Properties* und *Forms* und diese abrufbar macht. Dadurch kann der *LDT* die *WoT-TD* parsen, das Gerät abspeichern und alle momentanen Zustände der *Properties* auslesen.

Kapitel 6

Implementation

In diesem Kapitel wird die Implementation des *Longevity Digital Twins* vorgestellt

6.1 Einführung

Die Implementation des *Longevity Digital Twins* inklusive eines Anwendungsbeispiels besteht aus zwei Teilen. Die Hauptfunktionalitäten wurden in der Programmiersprache *Go* auf einem Windows Laptop implementiert. Auf einem *ESP32* wurde ein HAP-Kompatibles Smart Home-Device implementiert in der Programmiersprache *C++* und der *PlatformIO-IDE*. Ziel der Implementation war es, die Realisierbarkeit eines *Longevity Digital Twin* zu zeigen und ihn zu evaluieren.

6.2 Smart Home Geräte auf einem ESP32

Der für die Implementation genutzte Mikrocontroller ist der *ESP-32 Dev Kit C V4*. Das verbaute *ESP-32* Modul ist das *ESP32-WROOM-32*. Das *ESP-32Dev Kit C V4* hat eine Geschwindigkeit von bis zu *240MHZ* und einen Speicher von *512kbyte SRAM* [17]. Für die Implementation wurde die *Aruduino ESP32 Bibliothek* benutzt und *PlattformIO* als Build System.

6.2.1 PlattformIO

PlattformIO bietet mit der *PlatformIO IDE* ein Plugin für viele gängige *IDEs*, um Programme für Mikrocontroller zu entwickeln. Dabei liefert die *PlatformIO IDE* alle nötigen Funktionen, um für eine Vielzahl an Mikrocontrollern Programme zu compilieren und auf den Mikrocontroller hochzuladen[37]. Die Konfiguration des Programms inklusive des verwendeten Boards wird in der *platformio.ini* definiert. Die für dieses Projekt verwendete Konfigurationsdatei wird in Listing 6.1 angegeben. In dieser Datei wird das

verwendete *Board* angegeben, die *Plattform*, der *ESP32 Mikrocontroller* und das *Arduino* basierte *Framework*, das zur Programmierung benutzt wurde. Außerdem wurde die Serial Geschwindigkeit definiert, die für eine fehlerfreie Kommunikation zwischen PC und Mikrocontroller nötig ist, angegeben.

Listing 6.1: Die für dieses Projekt benutzte *platformio.ini*

```
[env:az-delivery-devkit-v4]
platform = espressif32
board = az-delivery-devkit-v4
framework = arduino
monitor_speed = 115200
lib_deps = https://github.com/HomeSpan/HomeSpan.git @ 1.7.1
```

Des Weiteren können externe Bibliotheken, die für das Programm benötigt werden, mit dem *lib_deps* Feld organisiert werden.

6.2.2 HAP auf einem ESP32

Die benutzte Bibliothek für ein *HAP* konformes Gerät ist die *HomeSpan* Bibliothek. Diese Bibliothek implementiert die komplette Kommunikation zwischen einem *HAP Server* und einem *HAP Device*. Der Ablauf eines *HomeSpan* Programms lässt sich in drei Teile aufteilen.

1. Aufrufen der *homeSpan.begin()* Funktion
   Diese Funktion erstellt alle nötigen Datenstrukturen und Felder für eine Kommunikation mit einem *HAP Server*.

2. Das Erstellen von *Accessoires*.
   HomeSpan führt intern eine Liste mit allen registrierten *Accessoires*. Sobald eine neue *Accessories* erstellt wird, können neue *SpanServices* erstellt werden. Dabei ist ein *SpanService* eine Klasse, von der geerbt werden kann, die drei Funktionen liefert, von der eine überschrieben werden muss, um zu definieren, was passiert, wenn mit dem Gerät kommuniziert werden soll. Diese Funktionen sind:

   Die *update()* Funktion:
   Diese Funktion wird immer ausgeführt, wenn von dem Server eine Anfrage an diesen *Service* gestellt wird.
   Eine Anwendung findet diese Funktion für die Glühbirne, da der *HomeServer* in der Lage ist, die Glühbirne an oder auszumachen und die Helligkeit zu regulieren. Sobald dieses *update* aufgerufen wird, wird in der Funktion überprüft, ob die Charakteristiken verändert wurden und falls sie verändert wurden, wird eine LED, die mit dem Gerät verbunden ist, entsprechenden reguliert.

Die *loop()* Funktion:
Diese Funktion wird bei jedem Aufruf von *homespan.poll()* aufgerufen, ohne auf Anfragen des *HomeServers* zu warten.
Diese Funktion wird für den Licht-Sensor benutzt. Sobald die *loop()* Funktion aufgerufen wird, wird überprüft wie viel Zeit vergangen ist seit dem letzten Update des Sensors und falls diese Zeit überschritten wurde, wird der momentane Lichtwert ausgelesen und der Wert der *Characteristik* wird durch die *setVal()* Funktion aktualisiert. Dies führt automatisch dazu, dass *HomeSpan* diesen Wert an den *HomeServer* weiterleitet.
Die *loop()* Funktion kann auch für einen Kontaktsensor benutzt werden. Dafür wird bei jedem Aufruf der *loop()* Funktion überprüft, ob sich der Status, beispielsweise eines Knopfes, verändert hat seit dem letzten Aufruf. Falls sich der Zustand verändert hat, wird der neue Zustand über die setVal() Funktion abgespeichert.

Die *button* Funktion:
Diese Funktion wird ausgeführt, wenn das Smart Device mit einem Button verbunden ist und dieser gedrückt wird.

3. Die letzte Funktion, die den Ablauf des *HAP*-Protokolls zur Laufzeit regelt, ist die *homespan.poll()* Funktion. Diese Funktion regelt die Kommunikation zwischen Server und Gerät und ruft die *loop()* Funktion jedes Device auf, falls diese implementiert wurden und die *update()* und *Button* Funktionen, falls diese ausgelöst wurden.

Die Kommunikation zwischen *LDT* und dem auf dem ESP32 implementierten *Homespan Device* kann dadurch ermöglicht werden, dass der in 6.2.2 geschilderte Ablauf um weitere Funktionen erweitert wird.

1. Nachdem *homeSpan.begin()* aufgerufen wurde, wird eine Verbindung mit dem LDT aufgebaut.

2. Nachdem die *Accessories* erstellt wurden, wird eine Registrierungs-Nachricht an den LDT geschickt. Außerdem müssen die *update, loop* und *button* Funktionen, je nach Gerät, erweitert werden. Diese Funktionen werden dahingehend erweitert, dass sobald eine Veränderung der Charakteristik erkannt wurde und diese aktualisiert wurden, der neue Status an den LDT geschickt wird. Das passiert im Falle der *update()* Funktion, nachdem die LED mit dem momentanen Status aktualisiert wurde und im Falle der *loop()* Funktion, nachdem ein neuer Wert vom Sensor ausgelesen wurde.

3. Nachdem die *homespan.poll* Funktion aufgerufen wurde, überprüft das Smart Gerät, ob es neue Daten vom *LDT* erhaltet hat. Falls es neue Daten erhalten hat, wird die Nachricht analysiert und es werden die Werte der Charakteristik aktualisiert. Dies

führt dazu, dass beim nächsten Aufrufen der *homespan poll* Funktion, die Charakteristik ihre *update* Funktion aufruft und so die LED mit den neuen Werten beschrieben wird und der *HomeSpan Server* informiert wird. Dabei ist anzumerken, dass nicht überprüft werden sollte, ob der *LDT* den Wert des Sensors verändern will, da ein Sensor nicht von außerhalb manipuliert werden sollte. Die eingehenden Nachrichten haben das Format (Name des *Accessories*, Name der *Charakteristik*, Wert).

6.3 Longevity Digital Twin Implementation

In diesem Kapitel wird eine Übersicht darüber gegeben, wie die vorher dargestellten Lösungsansätze in der Programmiersprache *Go* implementiert werden können.

6.3.1 Datenstrukturen in Go

Die Datenstrukturen wurden implementiert, mit dem Ziel Strukturen zu schaffen, die konform der *WoT-TD* sind. Dabei wurde die in *Go* integrierte *JSON*-Funktionalität benutzt. In *Go* können Felder von *structs* mit *JSON*-Schlüssel identifiziert werden, sodass beim Serialisieren des *structs* die Felder im *JSON* Format als Schlüssel/Werte Paar abgespeichert werden[12]. In Listing 6.2 ist zu sehen, wie die Datenstruktur eines *Things* aufgebaut ist.

Listing 6.2: *Go* Datenstruktur die, eine *WoT-TD* beschreibt

```
type ThingDescription struct {
    Context              []string                    `json:"@context"`
    TypeTD               string                      `json:"@type"`
    Id                   string                      `json:"id"`
    Title                string                      `json:"title"`
    SecurityDefinitions  map[string]SecurityDefinition `json:"securityDefinitions"`
    Security             string                      `json:"security"`
    Properties           map[string]Property         `json:"properties,omitempty"`
    Actions              map[string]Actions          `json:"actions,omitempty"`
    Events               map[string]Events           `json:"events ,omitempty"`
    Links                []Links                     `json:"links ,omitempty"`
}
```

Die in Rot geschriebenen Kennzeichnen geben an, wie die Schlüssel der Datenstruktur zu jedem Wert bei der *JSON* Serialisierung benannt werden. Diese Schlüssel entsprechen der *WoT-TD*. Optionale Felder werden durch *omitempty* charakterisiert. Diese Felder werden bei der Serialisierung ignoriert, falls diese den Null-Wert ihres Datentypen haben[12]. Die Felder *SecururityDefinition, Properties, Actions* und *Events* entsprechen jeweils aus einer *Map*, die als Schlüssel einen Namen hat und als Wert die jeweilige Datenstruktur. Die *Links* bestehen nicht aus einer *Map*, sondern aus einem *Array* an *Links*, wie in der *WoT-TD* angegeben. Die Datenstrukturen wie Property oder Actions

wurden analog aufgebaut und sind eine genaue Übersetzung der *WoT-TD* nach *Go*-Datenstrukturen mit den entsprechenden *JSON*-Schlüsseln. Auf diese Art und Weise liefert jedes erstellte *ThingDescription* in der *JSON*-serialisierten Form ein *WoT-TD* entsprechendes *JSON*-Objekt.

6.3.2 Kommunikation zwischen Smart Home Gerät und LDT

Der *LDT* dient als *TCP* Server der auf eingehende Verbindungen hört und für den Fall das eine Verbindung eingeht wird nebenläufig eine Routine gestartet, die auf Nachrichten wartet. Sobald Nachrichten eingehen, werden sie vom Parser verarbeitet. Die Verbindung kann auch dazu genutzt werden, um an den verbundenen *Device* Nachrichten zu senden, um bei dem *Device* Veränderungen zu triggern.

6.3.3 Parser

Bei dem Parser handelt es sich um die Komponente, die eingehende Nachrichten analysiert und daraufhin die entsprechenden Funktionen aufruft, um Datenstrukturen zu aktualisieren oder zu erstellen. Eingehende Nachrichten werden dabei immer im Zusammenhang mit der *IP*-Adresse des Smart Home Geräts zusammen analysiert, indem die eingehende IP-Adresse aus der Verbindung ausgelesen wird. Daraufhin wird im *LDT* nachgeschlagen, ob es für diese *IP*-Adresse schon registrierte *Twins* gibt und die *Twins* werden entweder aktualisiert oder es wird ein neuer *Twin* hinzugefügt. Die erwarteten Nachrichten bestehen aus den in Kapitel 5.4.1 definierten Werten. Der Parser geht die Nachricht Schritt für Schritt durch:

1. Der Parser überprüft, ob die Protokoll-*ID* implementiert ist, und wenn es existiert ruft er die entsprechende Handel-Funktion auf
2. Der Parser identifiziert zusammen mit der IP-Adresse und der Geräte ID, das entsprechende *Thing*
3. daraufhin folgen drei Fälle
    a) Falls es das *Thing* noch nicht gibt, wird davon ausgegangen, das es sich um eine Registrierung-Nachricht handelt. Das wird nochmals überprüft, indem die mitgesendete ID nachgeschlagen wird und falls es sich um eine ID eines Gerätes handelt und nicht die einer Eigenschaft, wird dieses Gerät mit allen verpflichtenden Eigenschaften generiert. Dabei wird der mitgesendete Name als Name des *Things* benutzt
    b) Falls es das *Thing* schon gibt, wird davon ausgegangen, dass es sich um eine Aktualisierung Nachricht handelt. Es wird anhand der mitgeschickten ID nach geschlagen, um welche Eigenschaft es sich handelt und dann wird der

Wert dieser Eigenschaft beim *Thing* gesetzt oder es wird diese Eigenschaft hinzugefügt, falls es diese noch nicht gab.

c) Das Generieren des *Thing* scheitert, falls eine Aktualisierung eines nicht existierenden *Thing* geschieht oder ein *Thing* mit unterschiedlichen Typen und derselben IP-Adresse und Geräte ID Paar registriert werden soll.

6.3.4 LDT Benutzer Interface

Der *LDT* dient auch als Webserver, auf dem eine einfache Benutzer-Oberfläche gehostet wird. Auf dieser ist es möglich, nach registrierten *TD* zu suchen. Dabei können auch explizit nach *Properties* von registrierten TD gesucht werden und ihren Status. Der *LDT* bietet auch eine Schnittstelle, um eine Nachricht an einen registrierten TD zu schicken.

6.3.5 Validierung zwischen HomeAssistant und LDT

Die Kommunikation mit *Home Assistant* über einen Websocket geschieht in drei Schritten.

1. Authentifizieren beim *Home Assistant Server*
   Um sich bei dem *Home Assistant Server* zu authentifizieren, muss eine Nachricht mit dem *Type auth* geschickt werden und einem *long access_token*. Wenn der Server diese Authentifizierung akzeptiert, wird diese bestätigt.
2. Sobald die Verbindung akzeptiert wurde, kann eine *GetStates* Anfrage geschickt werden, um alle verfügbaren *States* vom Server auszulesen.
3. Daraufhin muss durch alle *States* durchgearbeitet werden. Es wird hier auch wieder auf die eingebaute Funktionalität von Go Gebrauch gemacht, die es ermöglicht Datenstrukturen zu erstellen, in denen *JSON* Objekte deserialisiert werden können. Dadurch ist es möglich *State* Objekte zu erstellen, die mit der Antwort des *Home Assistant* Servers gefüllt werden können. Die *Entity* Namen, die in den *States* abgelegt werden, sind in dem Format domain.name abgelegt.

Im Falle einer Validierung wird nun mit dem Namen und der *Domain* gemeinsam genau identifiziert, welches Gerät und welche *Property* durch den *State* repräsentiert wird. Dabei gibt der *Domain* Name Aufschluss darüber, um welche Integration es sich bei der *Entity* handelt. Daraufhin können die Werte entsprechend konvertiert und verglichen werden. Dies wird für jedes Gerät durchgegangen und es wird ein Validierungsbericht für den Nutzer aufgestellt. Im Falle, dass der Webserver auf neue Geräte untersucht werden soll, werden die *Domain* und Namen Paare dafür genutzt, um *Home Assistant WoT-TDs* zu erstellen, die der entsprechenden *Domain* entsprechen.

6.3.6 Graph

Der Graph wurde in mit der *opensource* Bibliothek *go-echarts* erstellt. Bei dieser Bibliothek handelt es sich um eine in *Go* geschriebenen Adaption der eigentlich in *JavaScript* geschrieben *ECharts* Bibliothek von *Apache*[44][21]. *ECharts* bietet eine Vielzahl an unterschiedlichen Datenstrukturen für verschiedene Modelle, inklusive von Graphen mit Knoten und Kanten. Dieser *Graph* wurde für die Implementation benutzt. Außerdem bietet *go-echarts* eine einfache Methode, den erstellten Graphen zu visualisieren und als *html* Datei abzuspeichern. Dadurch kann der Graph für den Benutzer in eine Webseite eingebunden werden. Die *Graph*-Struktur wird befüllt, in dem *Nodes* und *Links* zu dem Graphen hinzugefügt werden. Dabei besteht ein *Link* aus einem Tupel, das aus Ursprung- und Ziel-Knoten besteht. Mit diesen Strukturen lässt sich durch Iterieren über registrierte *LDTs* ein Graph erstellen, der alle *LDT* in Relation setzt. Dafür muss der Name jedes *LDT* als Knoten dem Graphen hinzugefügt werden und daraufhin der Name aller *Properties*, *Events* und *Actions*. Diese Knoten werden alle mit dem Knoten, des *LDTs*, verbunden. Daraufhin werden die URL der *Forms* jeder *InteractionAffordance* dem Graphen hinzugefügt und mit ihrem entsprechenden Feld verbunden. Nachdem der Graph auf diese Weise befüllt wurde und jeder *Twin* inklusive seiner *InteractionAffordances* im Graphen abgespeichert sind, können die URLs der *Links* mit den *URLs* der *InteractionAffordances* verbunden werden.

Kapitel 7

Evaluation

7.1 Ziele der Evauluation

Ein Smart Home System, das durch *LDT*s unterstützt wird, wird auf zwei Arten beeinflusst. Einerseits durch die Performance von *LDT*-Geräten und andererseits durch die Performance des *LDT*s selbst.
Die Evaluation des *LDT* wird in zwei Teile aufgeteilt werden, um beide Einflüsse separat zu beurteilen. Im ersten Teil der Evaluation wird ein *LDT* Device evaluiert und es wird untersucht, wie sich die zusätzlichen Funktionalitäten des *LDT* Gerätes auf die Leistung des *LDT* Devices auswirkt. Relevante Untersuchungskriterien dabei sind:

- Auswirkungen der extra Nachrichten, die an den *LDT* geschickt werden, auf die Zeit bis eine Anfrage vollständig bearbeitet wurde.
- Auswirkungen der extra Funktionalitäten, Datenstrukturen etc. auf den Speicher.
- Codegröße eines *LDT* Gerätes.

Diese drei Untersuchungskriterien geben darüber Aufschluss, inwiefern die extra *LDT* Funktionalitäten Auswirkungen auf die minimal benötigte Hardware eines *LDT* Gerätes hat und wie die *LDT* Funktionalität die Reaktionsgeschwindigkeit beeinflusst.
Im zweiten Teil der Evaluation werden die Anforderung an das Gerät, das den *LDT* selbst hostet, analysiert. Das Ziel dieser Evaluation ist es, die Skalierung zu überprüfen und ob der *LDT* es schafft Abhängigkeiten darzustellen von unterschiedlichen Geräten unabhängig, ob es sich um ein zentrales oder dezentrales Gerät handelt. Relevante Untersuchungskriterien sind:

- Auswirkung von *WoT-TDs* auf den Speicher.
- Abhängigkeiten dargestellt durch den Graphen.

Anhand dieser Untersuchungen wird der *LDT* im Anschluss bewertet und es wird ein Fazit gegeben, ob die Implementierung des *LDT* ein Erfolg war, und in welchen Aspekten der *LDT* nicht die erhofften Ziele erfüllt.

7.2 Methodik

7.2.1 LDT Device Versuchsaufbau

Für die Versuche wurde neben dem *LDT* Programm für einen *ESP32* ein weiteres Programm entwickelt, das verschiedene *HomeKit Services* mit *HomeSpan* auf einem *ESP32* hosten kann. Beide Programme sind in der Lage, drei verschiedene *Services* zu hosten: eine Glühbirne, einen Lichtsensor und einen Kontaktsensor. Außerdem wurden beide Programme mit Funktionalität erweitert, um einerseits eine beliebige Anzahl an Geräten zur Laufzeit hinzuzufügen und andererseits wurden Testfunktionen hinzugefügt, die es ermöglichen Timer zu starten und auszulesen, um die Messungen durchzuführen. Die vier Testfälle, die untersucht worden sind:

1. Einschalten von *HomeKit* Glühbirnen über einen Smart Home Server.
2. Einschalten von *HomeKit* Glühbirnen mit *LDT* Funktionalität über einen Smart Home Server.
3. Aktualisierungen *HomeKit* Sensoren.
4. Aktualisierungen *HomeKit* Sensoren mit *LDT* Funktionalität.

Die ersten beiden Fälle erlauben einen direkten Vergleich über die Auswirkungen des *LDT* auf den normalen Betrieb eines Smart Home Aktuators. Die letzten beiden Fälle erlauben einen direkten Vergleich über die Auswirkungen des auf den normalen Betrieb eines Smart Home Sensors. Um die Auswirkungen auch für komplexere Systeme zu simulieren, wird jedes der vier Experimente mit jeweils einer unterschiedlichen Anzahl an *HomeKit* Accessoires mit jeweils den entsprechenden Charakteristiken erstellt. *HomeSpan* erlaubt eine Erstellung von bis zu 41 *Accesories* auf einem Gerät. Es wird in den Tests auch untersucht, ob diese 41 *Accesories* auf einem *HomeSpan* Gerät gehostet werden können und ob die maximale Anzahl an Geräten durch die *LDT* Funktionalität beeinflusst wird.

7.2.2 LDT Device Komponenten

Die Experimente wurden alle mit der gleichen Hardware durchgeführt und haben folgende Komponenten:

- Ein *ESP32 Dev Kit C V4* der die entsprechenden Programme ausführt. Der *ESP32* ist mit einem Lichtsensor, zwei LEDS, einem Button und einem Oszilloskop verbunden. Bei dem Lichtsensor handelt es sich um einen Fotowiderstand, der Lichtunterschiede erkennt. Die beiden LEDS sind einerseits die Status LED, die von *HomeSpan* unter anderem dafür benutzt wird, um anzuzeigen, ob das Gerät erfolgreich mit dem Smart Home Server verbunden ist. Die andere LED ist die LED,

die eine Smart Home Glühbirne simuliert. Der Kontaktsensor wird durch einen Knopf simuliert, der angibt, wenn der Knopf gedrückt wird, dass ein Kontakt besteht und wenn der Knopf nicht gedrückt wird, dass kein Kontakt besteht.

- Ein PC, der bis zu drei Programme hostet:
  1. Die *PlatformIO IDE* wird auf dem PC ausgeführt, um einerseits die Programme auf den *ESP32* zu laden und andereseits Interaktionen mit dem Serial Monitor zu erlauben. *PlatformIO* bietet auch eingebaute Funktionen, um den verbrauchten Flash und RAM eines Programms auszulesen.
  2. Die *Oracle VM VirtualBox* wird ausgeführt, um das *Home Assistant Operating System* zu hosten. Das *Home Assistant Operating System* erlaubt es den *ESP32* als *HomeKit* Gerät mit dem Server zu verbinden und bietet ein Benutzer Interface, auf dem per Knopfdruck alle Lampen eines Gerätes ausgeschaltet oder angeschaltet werden können. Außerdem bietet das OS eine GUI, um verifizieren zu können, dass eine Kommunikation zwischen *ESP32* und Smart Home Server besteht, wenn die Sensoren gemessen werden.
  3. Der *LDT* wird in jedem Experiment, bei dem er relevant ist in *GoLand* gestartet und es wird über die Konsole überwacht, dass die Kommunikation stattfindet.
- In den ersten Messungen wurde ein Oszilloskop verwendet, um Zeitmessungen durchzuführen, dabei wird dem Oszilloskop ein elektrisches Signal immer vor den Messungen geschickt und sobald die Messung abgeschlossen wurde, wird dieses Signal abgebrochen. Dadurch lassen sich über den Bildschirm des Oszilloskops genaue Zeitabstände ablesen. Aufgrund von technischen Problemen mussten spätere Tests über die interne *Clock* des *ESP32* durchgeführt werden.

7.2.3 Ablauf der Messungen

1. Löschen des Flash-Speichers des *ESP32* vor jedem Anwendungsfall, um sicherzustellen, dass keine Überreste eines anderen Programms im Flash liegen.
2. Hochladen des Programms in der entsprechenden Konfiguration und starten des Programms.
3. Verbinden mit dem W-LAN über das *HomeSpan CLI*.
4. Verbinden mit dem *LDT* über das *HomeSpan CLI*, falls *LDT* Device.
5. Erstellen eines Geräts im entsprechenden Modus über die *HomeSpan CLI*.
6. Paring des Gerätes in *Home Assistant*.
7. Starten des Tests über die *HomeSpan CLI*.

8. Triggern des entsprechenden Ereignisses, das getestet werden soll und Messungen im Serial Monitor ablesen.

Der 8. Punkt wird dann beliebig oft wiederholt und die Messungen werden abgelesen. Daraufhin kann durch die *HomeSpan CLI* ein neues Gerät hinzugefügt werden und die Verbindung mit *Home Assistant* über die *Home Assistant* UI aktualisiert werden.

7.2.4 Messungen des LDTs

Die Messungen an dem *LDT* werden nicht mit dem *ESP32* in Verbindung gebracht, da der *ESP32* wie schon vorher erwähnt hat nur eine limitierte Anzahl an Geräten hosten kann. Die Daten, die dem *LDT* gegeben werden basierend auf automatisch generierten Nachrichten, die dem in dieser Arbeit konformen Format entsprechen und diese Daten werden direkt dem Parser übergeben. Die Speichermessungen werden gemessen, indem, nachdem der Parser eine Nachricht geparst hat und das Datenmodell den Strukturen des *LDT*s hinzugefügt hat, der verbrauchte Speicher des *LDT*s ausgelesen und abgespeichert wird. Es werden außerdem Timestamps abgespeichert, um zu evaluieren, wie sich das Zeitverhalten des *LDTs* in Abhängigkeit der Anzahl an Geräten verhält. Es werden zwei Fälle unterschieden, das Generieren von Geräten mit einem einzelnen Ursprung und das Generieren von Geräten mit mehreren Ursprüngen. Diese beide Fälle simulieren die beiden Hauptquellen, von dem der *LDT* die Daten bekommt. Einerseits von einem Webserver, der die Quelle von einer Vielzahl an Geräten ist und anderseits von einer Vielzahl an *LDT* Geräten, die mit dem *LDT* in Verbindung stehen. Es wird der erste Fall einmal mit *HAP* Objekten und einmal mit *Home Assistant* Objekten ausgeführt, um den Speicherverbrauch im selben Szenario, aber mit unterschiedlichen Geräten zu vergleichen. Der *LDT* erstellt anhand der Daten mit *GoCharts* die entsprechenden Graphen.
Der *Dependency* Graph wird anhand von drei Anwendungsbeispielen evaluiert. Die durch drei unterschiedliche Szenarios darstellen, bei dem der Graph ein hilfreiches Analysewerkzeug sein soll. Die drei Szenarios und ihre Bedeutung sind:

1. Ein Webserver hat mehrere Glühbirnen, die von *LDT* abgespeichert werden.
   Dieses Szenario ist ein Beispiel dafür, wenn mehrere Geräte einen gemeinsamen Ursprung haben. Zu erkennen, wenn eine Vielzahl von Geräten abhängig von einem Knotenpunkt sind, kann die Wartung eines Smart Home Systems erleichtern, um zu erkennen, welche Teile eines Systems ausfallen würden, falls der Server oder ein *HAP* Gerät, das mehrere Accessoires hostet, ausfällt.

2. Eine Glühbirne wird sowohl von einem Lichtsensor und einem Kontaktsensor beeinflusst. Es könnte zum Beispiel ein Licht sein, das je nach Werten des Lichtsensors die Helligkeit ändert oder ausgeschaltet wird und sobald der Kontaktsensor geschaltet wird, wird die Lampe an- oder ausgeschaltet. Dieses Szenario bietet ein Beispiel, für ein Smart Home Gerät, das von mehreren anderen Geräten abhängig

ist.
Dieses Szenario ist ein Beispiel dafür, dass ein Gerät eine indirekte Abhängigkeit zu einem anderen Gerät hat. Der Zustand des Lichtsensors könnte beispielsweise für die Lampe egal sein, solange der Kontaktsensor schaltet. Es besteht also eine indirekte Abhängigkeit zwischen Kontakt und Lichtsensor. Das könnte dazu führen, dass ein Fehlverhalten des Kontaktsensors auch die Funktionalität einer Automation zwischen Glühbirne Lichtsensors beeinflusst.

3. Ein Bewegungssensor beeinflusst ein Alarmsystem, welches wiederum eine Lampe beeinflusst. Es könnte zum Beispiel ein Mechanismus sein, der simulieren soll, dass sich jemand zu Hause befindet. *HomeKit* bietet mit dem *SecruityService* ein Mechanismus, dem man mitteilen kann, welcher Modus das Sicherheitssystem hat[26]. Der Bewegungssensor erkennt, dass sich etwas um das Haus bewegt und teilt es dem Alarmsystem mit. Das Alarmsystem entscheidet anhand des internen Status, ob es eine Aktion ausführen soll. Falls es die Aktion ausführt, wird eine Lampe angemacht. Dieses Szenario bietet ein Beispiel dafür, wenn ein Smart Home Gerät eine direkte Abhängigkeit zu einem anderen Gerät hat ohne direkt mit diesem verbunden zu sein. Der Status der Lampe ist abhängig von dem Status des Lichtsensors, obwohl diese nicht direkt in Verbindung stehen.

Alle drei Tests werden gleichzeitig vom Graphen des *LDT* modelliert, dazu wurde der Parser mit den entsprechenden Nachrichten aufgerufen, um einen Bewegungssensor, einen Lichtsensor, einen Kontaktsensor und zwei Lampen zu generieren. Bei allen Geräten handelt es sich um *HAP* Geräte außer der Lampe für den Alarm. Bei der Alarmlampe handelt es sich um ein *Home Assistant* Objekt. Daraufhin wurden den Geräten *Links* hinzugefügt, die entsprechend der Szenarios die Geräte miteinander verbinden. Die Verbindungen zu *Collections* und *Links* wurden automatisch generiert.

7.3 LDT Gerät Evaluationsergebnisse und Analyse

In diesem Unterkapitel werden alle Messergebnisse vorgestellt, die für ein *LDT* Gerät generiert wurden. Vorher wird analysiert, welche Ergebnisse zu erwarten sind.

7.3.1 Erwartungen an die Messergebnisse

Die Erwartung an die Messergebnisse ist, dass die *LDT* Geräte eine längere Zeit beanspruchen, um alle Anfragen des Webservers zu bearbeiten. Das liegt daran, dass die *LDT* Geräte neben der normalen *HomeSpan* Routine ihre Daten auch an den *LDT* schicken müssen. Die Verwendung einer dimmbaren und nicht dimmbaren Glühbirne sollte dabei keinen Unterschied machen, da bei einem simplen Aus und Anschalten nur die *On* Charakteristik der entsprechenden Glühbirne gesetzt werden wird. Der Zuwachs zwischen

den benötigten gesendeten Daten, ist für jedes dazu kommende Gerät konstant, da genau eine zusätzliche Nachricht pro Gerät an den *LDT* geschickt wird und da es sich um dieselbe Art an Geräten handelt, sind diese Datenpakete gleich groß. Es erhöhen sich also die zu versendeten Daten um den Wert *Anzahl der Geräte*LDTMessageSize*.

7.3.2 Erste Messergebnisse mit einem Oszilloskop

Die erste Reihe an Messergebnissen wurden mit einem Wi-Fi-Hotspot eines Android Smartphones durchgeführt. Es wurde jeweils ein Test durchgeführt und die Zeit wurde über ein Oszilloskop abgelesen. Die Messergebnisse sind in Fig. 7.1. Es zeigt sich, dass in diesen Testreihen die Ergebnisse sehr überraschend und unlogisch erscheinen. Einerseits zeigen Messwerte, dass es schneller geht, fünfzehn Glühbirnen mit *LDT* zu schalten anstatt fünfzehn Glühbirnen ohne *LDT* oder vierzehn Glühbirnen mit *LDT*. Dadurch, dass *TCP* dafür sorgt, dass alle Pakete immer ankommen, bedeutet, dass wenn eine Verbindung, die fehlerhaft ist, dazu führen kann, dass Pakete neu gesendet werden müssen. Das ist gerade bei dem Setzen von Glühbirnen relevant, da der Webserver eine Nachricht an den *ESP32* schickt und der *ESP32* eine Antwort an den Webserver schickt und eine Nachricht an den *LDT* schicken muss. Es gibt also mindestens drei Nachrichten, die bei einem W-LAN Netzwerk, das fehlerhaft ist, zu Verzögerung führen können. Im selben Netzwerk wurden außerdem Lichtsensoren getestet. Diese Ergebnisse sind in Fig. 7.2. Die Ergebnisse der Lichtsensoren haben auch unlogische Schwankungen in den Daten, es ist zwar wie zu erwarten die Zeit, die ein Lichtsensor mit *LDT* dauert, konstant länger, allerdings sind bei den Daten ohne *LDT* auch unerklärbare Schwankungen zu erkennen, wie zum Beispiel das es 14ms dauert dreizehn Lichtsensoren upzudaten und zwölf Lichtsensoren dauern 16,68 ms. Da die Daten so inkonsistent sind, können diese Daten nicht dafür benutzt werden weitere Analyse durchzuführen. Diese Analyse wurden nach einem angepassten Versuchsaufbau wiederholt.

7.3.3 Aktualisierung der Messmethodik

Die Ergebnisse der ersten Tests haben dazu geführt, dass die Evaluation angepasst wurde. Die weiteren Tests wurden in einem stabilen W-LAN ausgeführt und jeder Test wird zehnmal wiederholt. Es wurden für diesen Test die *Clock* des *ESP32* benutzt. Der Grund, wieso die Tests alle mehrmals wiederholt worden sind, ist es Ausreißer zu erkennen und genauer Trends feststellen zu können. In den folgenden Testergebnissen wird immer der Median und der Durchschnitt angegeben bei jedem Test pro Gerät. Auf diese Weise lässt sich erkennen, ob es auch in einem stabileren W-LAN zu Ausreißern kommt. Anhand des Durchschnitts und durch den Median ist es möglich den Trend genauer bestimmen zu können, welche Auswirkung ein *LDT* auf die Performance hat. Es wurde außerdem Funktionalität implementiert, die es ermöglicht zur Laufzeit neue Geräte auf den *ESP32* hinzuzufügen und die Tests ohne neu starten durchzuführen. Das hat das Ziel, dass

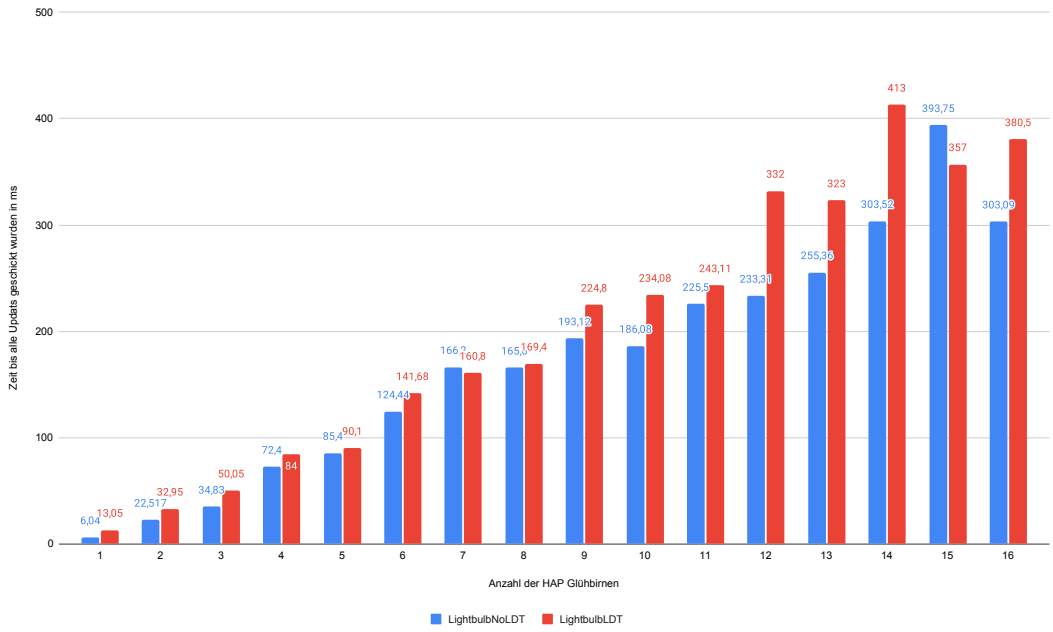


Abbildung 7.1: Testergebnisse von *HAP* Glühbirnen in einem Wi-Fi-Hotspot eines Android Smartphones

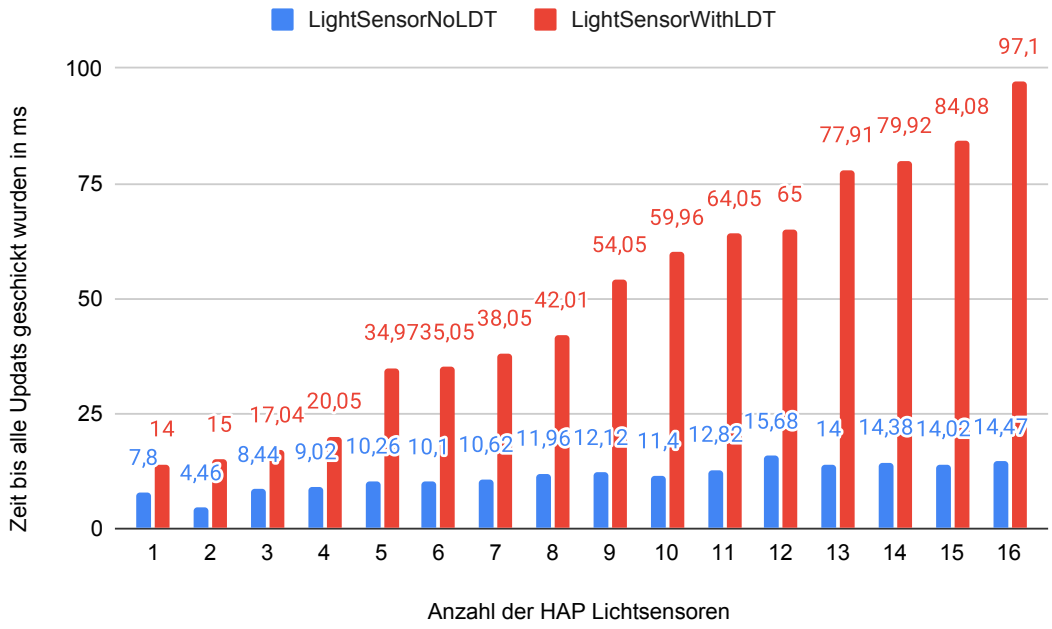


Abbildung 7.2: Testergebnisse von *HAP* Lichtsensor in einem Wi-Fi-Hotspot eines Android Smartphones

nicht immer eine neue Verbindung mit dem Webserver aufgebaut wird von dem *ESP32*, sondern die bestehende Verbindung wieder benutzt wird.

7.3.4 Lichtsensoren, Messergebnisse und Analyse

Ursprünglich war geplant, die Messungen der Sensoren weiterhin durch Lichtsensoren zu realisieren. Dabei wären bis zu 41 verschiedene Lichtsensoren generiert worden und jede 5 Sekunden hätten diese einen neuen Wert gemessen, der durch *HomeSpan* an den *Home Assistant* Webserver geschickt worden wäre und für die *LDT* fähigen Geräte an den *LDT*. Allerdings haben Messungen gezeigt, dass, wenn zur Laufzeit Lichtsensoren hinzugefügt werden, die gemessenen Zeiten wieder starke Schwankungen aufweisen und zu unbrauchbaren Ergebnissen führen. Das hat den Grund, dass die Lichtsensoren, nachdem sie einen neuen Wert gemessen haben und die *setVal* Methode aufrufen, jeweils einen eigenen Timestamp abspeichern, wann das letzte Mal eine Messung durchgeführt wurde. Das bedeutet das, wenn ein Lichtsensor erstellt wird, dieser direkt eine Messung durchführt und die Timestamp abspeichert. Ein weiterer Sensor, der hinzugefügt wird und eine Messung durchführt und einen Timestamp abspeichert, hat nun einen anderen Timestamp abgespeichert. Also haben beide Sensoren nicht mehr die gleiche Taktung, da der Timestamp des zweiten Sensors um die entsprechende Zeit verzögert ist, die es gebraucht hat, den Sensor zu erstellen. Es wäre möglich dieser Problematik zu entgehen, indem beim Erstellen jedes Lichtsensors eine Referenz abgespeichert wird und zum Testen von jedem Lichtsensor die *loop* Funktion aufgerufen werden würde und keine automatische Aktualisierung des Lichtsensors anzuwenden. Es wurde sich allerdings dagegen entschieden, da es im normalen Betrieb immer so ist, dass *HomeSpan* die *loop* Funktionen aufruft, beim Aufrufen von *HomeSpan.poll*. Es würde also die Testergebnisse verfälschen und nicht vergleichbar mit den Testergebnissen der Aktuatoren machen, wenn die *loop* Funktionen manuell aufgerufen werden. Eine Alternative ist es stattdessen immer ein Update zu schicken, wenn sich der Lichtwert verändert hat im Vergleich zur alten Messung. Aus praktischen Gründen wurde diese Art eines Sensors durch einen Kontaktsensor realisiert und dieser wurde gemessen, da ein Kontaktsensor einfacher ist manuell zu triggern, um ein Update aufzurufen.

7.3.5 Kontaktsensoren, Messergebnisse und Analyse

Die Testergebnisse der Kontaktsensoren können in Fig. 7.3 betrachtet werden. Alle Messungen wurden mehrmals durchgeführt und aus jeder Testreihe wurde der Durchschnitt und der Median bestimmt, dies wurde getan, um zu überprüfen, wie viel Auswirkungen Ausreißer auf die durchschnittliche Zeit eines Updates haben und ob ein Muster erkennbar ist, wann diese Ausreißer auftreten oder ob es zufällig ist. Die Tests des No-*LDT*(NDT) ergeben, dass sich die Zeit eines Updates leicht erhöht in Abhängigkeit der Anzahl der Geräte. Ein einzelnes Gerät braucht im Median 5.4 ms und erhöht sich auf

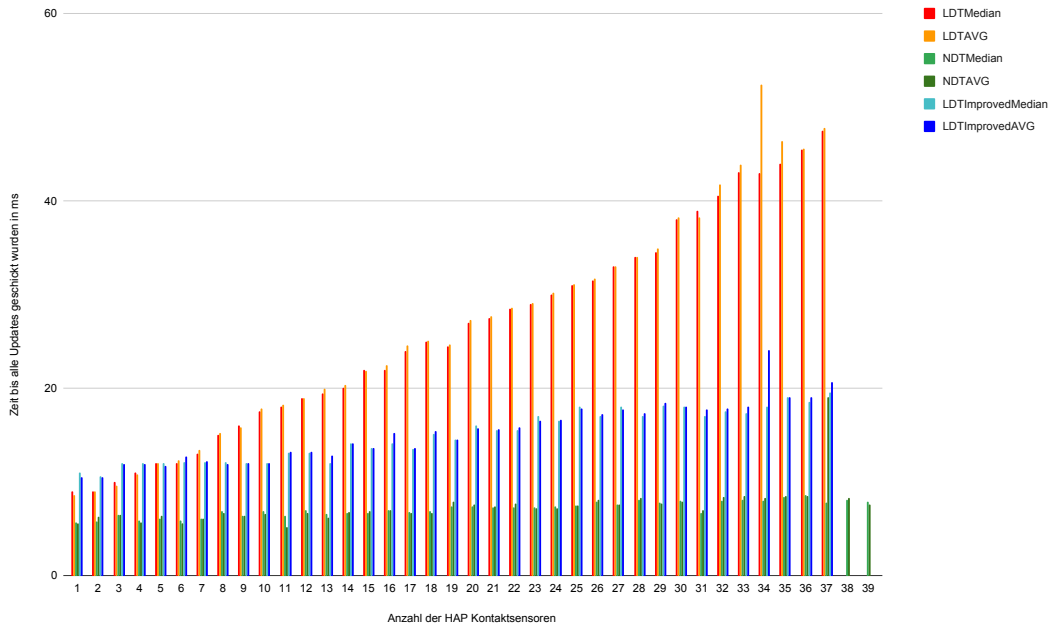


Abbildung 7.3: Testergebnisse von Kontaktsensoren mit *LDT*, No *LDT* und mit Buffer verbessert

7.5 ms bei vierzig Geräten. Der Graph eines *No-LDT* lässt sich als eine leicht steigende linearer Verlauf beschreiben. Die Daten der Tests eines *LDT* Gerätes haben eine exponentielle Form und die Zeit von einem einzelnen Gerät beträgt 8 ms und die Zeit von 40 Geräten beträgt 46 ms. Es ist also ein Anstieg um, 610Prozent erkennbar.
Der Grund für diese starken Unterschiede der Dauer eines Updates lässt sich folgendermaßen erklären. *HAP* erlaubt es, dass Updates von mehreren Charakteristiken in nur einer *HTTP PUT* Nachricht an den Server weiter gegeben werden können. Es kann also in das Update für jeden Sensor der in diesem *HomeSpan.poll* Aufruf eine Veränderung festgestellt hat, zusammen gefasst werden und an den Server in einer *HTTP* Nachricht geschickt werden. Der *LDT* schickt in der ursprünglichen Version immer, sobald die *loop* Funktion eines Sensors aufgerufen wurde und eine Veränderung festgestellt wurde, eine Nachricht über die laufende *TCP* Verbindung an den *LDT*. Das bedeutet, dass für jede Nachricht Overhead entsteht. Diese Theorie wurde überprüft, indem ein Buffer implementiert wurde. Es wird also nicht wie vorher bei jeder *loop* direkt eine Nachricht geschickt, sondern stattdessen wird die Nachricht einem Buffer hinzugefügt. Der Buffer wird dann direkt, nachdem die *HomeSpan.poll* Methode aufgerufen wurde, verschickt. Die zeitliche Verbesserung der Implementation mit Buffer ist in dem Graphen klar erkennbar. Es dauert leicht mehr als doppelt so lange, die Ergebnisse eines *LDT* Geräts mit Buffer verglichen mit einem Nicht *LDT* Gerät. Dadurch dass, das Paketschema von *HomeKit* effizienter ist, da weder das Protokoll noch die *UUID* mit verschickt werden muss, ist der Performance-Unterschied mehr als das zweifache an verbrauchter Zeit.

7.3.6 Glühbirnen Messergebnisse und Analyse

Die Daten der Messergebnisse von *HAP* Glühbirnen ist in Fig. 7.4 dargestellt. In diesem Test wurde für das *LDT* Gerät nur die Buffer Implementation benutzt. Die Daten für Glühbirnen zeigen, dass auch in einem konstanten W-LAN die Kommunikation Ausreißer existieren. Der Median beider Implementationen zeigt einen linearen Anstieg der Zeit, die es beansprucht für alle Updates. Der *LDT* dauert bis zu 60ms länger als ohne *LDT*, und die Zeiten sind insgesamt ein Vielfaches als der Kontaktsensor. Das lässt sich durch das Auslesen der von *HomeSpan* geschickten und erhaltenen Daten erklären. *HomeSpan* bietet eine Option, die Pakete, die zwischen dem Gerät und dem Smart Home Server ausgetauscht werden, auszulesen. Es zeigt sich in diesen Logs, dass *Home Assistant* die Glühbirnen einer nach der anderen schaltet. Das bedeutet, dass einerseits *HomeSpan* für jedes eingehende Setzen einer Glühbirne direkt eine Antwort an *Home Assistant* schickt, aber auch dass selbst, mit der Buffer Implementation des *LDTs*, jedes Setzen der Glühbirne zu einer *Update*-Nachricht an den *LDT* führt. Es bedeutet auch, dass die Zeit, die *Home Assistant* braucht, die Daten an den *ESP32* zu schicken, mit gemessen wird bei den Tests. Es entsteht durch diese zusätzliche Kommunikation auch eine weitere Quelle für mögliche Fehler, wodurch die starken Ausreißer zu erklären sind. Dadurch lässt sich der starke Unterschied zwischen den Ergebnissen zwischen Glühbirnen und

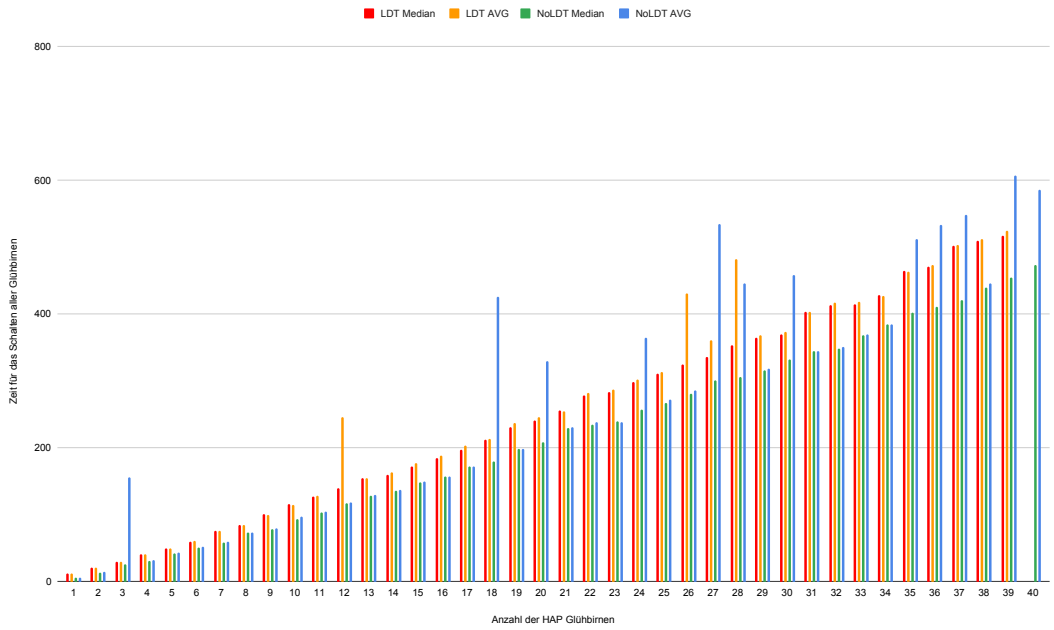


Abbildung 7.4: Testergebnisse von Glühbirnen mit *LDT* und *NoLDT*

	RAM	FLASH	Kontaktsensor	Dimbare LED	LED
kein LDT	56792	1010217	2400	2550	2300
LDT	57476	1051093	2400	2680	2456

Abbildung 7.5: Speicherverbrauch von *LDT* und nicht *LDT* Geräten, alle Werte sind in Byte

Sensoren erklären. Zusammen fassend lässt sich sagen, dass unabhängig von der *LDT* Funktionalität, das Schalten aller Aktuatoren durch Home Assistant ineffizient ist und die *LDT* Implementierung, dazu führt, dass diese Ineffizienz noch stärkere Auswirkungen hat. Es lässt sich in dem Graphen aber auch erkennen, dass der prozentuale Anstieg in der Zeit, die es braucht, durchgängig nur zwischen 15 und 20 Prozent ist.

7.3.7 Speicherverbrauch von LDT Geräten und nicht LDT Geräten

In Fig. 7.5 ist der Speicherverbrauch zwischen *LDT*-Gerät und nicht *LDT*-Gerät abgebildet. Die Felder Kontaktsensor, Dimmbare LED und LED geben an, wie viel RAM das Erstellen des jeweiligen Gerätes im Durchschnitt benötigt. Es kann nur im Durchschnitt angegeben werden, da intern *homespan* eine Datenbank mit allen registrierten

Geräten führt und diese teilweise zu Ausreißern bei der Generierung von neuen *Accessories* führt. Der Unterschied der unterschiedlichen Geräte ist sehr gering, das liegt daran, dass für *LDT* und nicht *LDT* Geräten die gleiche Datenstruktur verwendet wird und diese nur um Funktionsaufrufe erweitert wurden. Der größere Speicherverbrauch der *LEDs* lässt sich dadurch erklären, dass ein *LDT* Gerät eine *Map* erstellt, um bei eingehenden Nachrichten, des *LDTs*, die Nachrichten dem entsprechenden *Accessory* zuordnen kann und den Status zu setzen. Sensoren werden nicht in dieser *Map* gespeichert, also hat ein Kontaktsensor in beiden Variationen den gleichen Speicherverbrauch. Es zeigt sich zur Laufzeit, dass der Speicher sowohl mit und ohne *LDT*, ein Problem wird bei einer Vielzahl an Objekten und teilweise das Verbinden nicht mehr mit dem *Home Assistant* Server funktioniert, sobald mehr als 30 Geräte erstellt werden, aufgrund von mangelnden *RAM*. Das geschieht vor allem, wenn alle Geräte gleichzeitig verbunden werden sollen. Es konnte nicht genau analysiert werden, wann genau dieser Fehler auftritt, da es teilweise möglich war bis zu 41 Geräte zu erstellen und teilweise nach einem Neustart, das Programm vorher gestoppt ist. Dieser Fehler tritt, sowohl bei *LDT* und nicht *LDT* Geräten auf.
Die anderen Werte wurden erstellt, indem der Code von PlatformIO compiliert wurde und die Daten ausgelesen wurden. Es zeigt sich, dass die Auswirkungen auf den *RAM* Speicher gering sind und der größte Unterschied im Verbrauch des *FLASH* Speichers liegt. Das liegt daran, dass sowohl der Code als auch globale und *const* Variablen im Flash liegen. Es lässt sich zusammen fassen, dass der *RAM* Verbrauch der eigentlichen *Accessories* sich kaum unterscheidet, aber die *LDT* Funktionen einen starken Einfluss auf den *Flash* Verbrauch haben.

7.4 LDT Evaluationsergebnisse und Analyse

7.4.1 Erwartungen an die Messergebnisse

Die Messergebnisse der Erstellungen von Datenmodellen in *GO* ist zu erwarten, dass es einen linearen Anstieg an den im *Heap* verbrauchten *Bytes* sein wird. Jedes *Thing* Objekt sollte sich unabhängig von dem benutzten Ursprungsmodell in nicht vielen *Bytes* unterscheiden. Ob der *TypeAt* eines *Thing* einen *HAP* Type oder einen *Home Assistant* Type beschreibt, sollte nicht viel Auswirkungen auf den Speicherverbrauch der Objekte haben. Es sollten also bei jedem Test die gleichen Tendenzen feststellbar sein, wie sich der Speicher verhält. Der größte Unterschied zwischen den unterschiedlichen Modellen der verschiedenen Protokolle, sollte genau dann erkennbar sein, falls realistischere Datensets generiert werden. Es ist wahrscheinlicher, dass es in einem Haushalt eine Vielzahl an *HAP* Geräten geben, die alle ihr eigene IP-Adresse haben und es ist wahrscheinlicher, dass ein Haushalt nur einen *Home Assistant* Server besitzt. Die beiden Szenarios sollten durch die *Collection Things* einen stärkeren Unterschied in dem Speicherverbrauch haben. Aus diesem Grund werden drei Datensets erstellt. Einerseits werden 1000 *HAP* Nachrichten

geparst, die alle von einem Gerät stammen, es werden 10000 *HAP* Nachrichten geparst, die alle Nachrichten von unterschiedlichen Geräten stammen und es werden 1000 *Home Assistant* Nachrichten geparst, die alle von einem Gerät stammen. Alle Tests wurden einmal mit gezielter *GC* und einmal mit automatischer *GC* ausgeführt.

7.4.2 Testergebnisse mit Garbage Collection(GC)

Bei den Tests hat sich gezeigt, dass es bei den Speicherauslesungen starke Schwankungen gibt in den Daten. Es sind starke Abfälle zu erkennen bei ca. 3500000 Bytes (siehe Fig. 7.6, Fig. 7.7 und Fig. 7.8). Das lässt erklären durch den *GC* von *GO* der in bestimmten Intervallen alle außerhalb des *Scope* liegende Objekte aus dem Speicher löscht. Jedes Markieren der zu löschenden Objekte und das darauffolgende Löschen sind *GC Cycles*. [11]. Dadurch sind die Sprünge in dem Graphen erklärbar, da der ganze verbrauchte *Heap* ausgelesen wird und auch Daten, die gelöscht werden können. Sobald die *GC* startet, werden alle unbrauchbaren Daten gelöscht und der Verbrauch des *Heap* reduziert sich. In Fig. 7.6 ist zu sehen, wie sich der Graph bei 1000 *HAP* Geräten verhält, die jeweils einen anderen Ursprung haben. Es ist zu sehen, dass der *GC* zweimal getriggert wurde. Ansonsten ist der Speicherzuwachs linear. Der größte Speicherverbrauch liegt bei 5981864 Byte. In Fig. 7.7 und Fig. 7.8 sind die Graphen abgebildet bei denen jeweils 1000 *HAP* und 1000 *Home Assistant* Geräte geparst wurden, die jeweils von einem Ursprung kommen. Der größte Speicherverbrauch liegt bei 4625112 und 4522840 *Byte*. Beide Graphen haben einen nahezu identischen Graphverlauf. Der starke Unterschied zwischen den Tests von einem zu mehreren Ursprüngen kann dadurch erklärt werden, dass der *LDT* für jeden Ursprung eine *Collection* erstellt. Es werden also bei 1000 Nachrichten, die von 1000 Ursprüngen kommen, 1000 verschiedene *Collections* erstellt, zusätzlich zu den eigentlichen *Things*, die das Gerät selbst beschreiben. Es werden also 2000 *Thing Descriptions* erstellt. Da der Umfang der *Collections* nicht so umfangreich ist, sondern hauptsächlich aus den *Links* besteht, die auf die zugehörigen Geräte verweist, ist der Speicherverbrauch nicht doppelt so groß. Bei nur einem Ursprung wird nur eine *Collection* erstellt und es werden alle *Links* zu der einen *Collection* hinzugefügt.

7.4.3 Ergebnisse mit gezielter Garbage Collection

Die Tests wurden wiederholt und es wurden für die Tests die automatische *Garbage Collection* ausgeschaltet. Stattdessen wird immer vor jedem *Snapshot* einmal die *Garbage Collection* aufgerufen, um alle Elemente aus dem Speicher zu löschen, die nicht mehr benötigt wurden. Dies ermöglicht eine genauere Untersuchung, wie viel Platz die unterschiedlichen Objekte im Speicher benötigen und Tendenzen zu erkennen. Es zeigt sich in allen drei Graphen(siehe Fig. 7.9, Fig. 7.10 und Fig. 7.11), dass der Speicherverbrauch linear ist und es keine Abfälle gibt. Der Speicherverbrauch ist im Durchschnitt auch geringer, da in jedem Snapshot keine Daten mehr auf dem *Heap* liegen, die vom Parsen

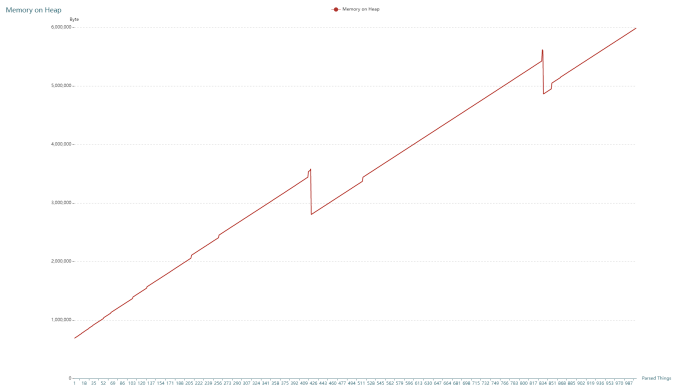


Abbildung 7.6: Speicherverbrauch von *HAP* Objekten mit mehreren Ursprüngen und aktiver *GC*

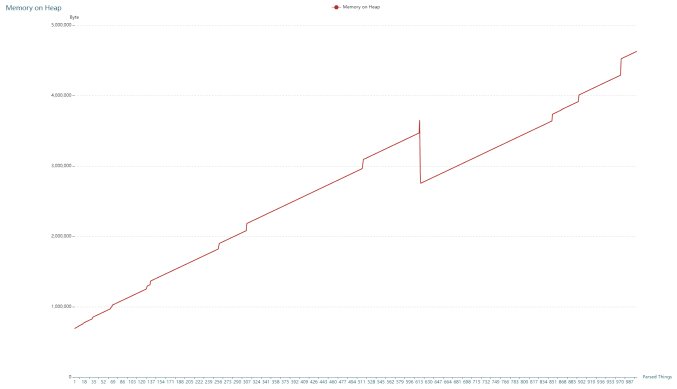


Abbildung 7.7: Speicherverbrauch von *HAP* Objekten mit einem Ursprung und aktiver *GC*

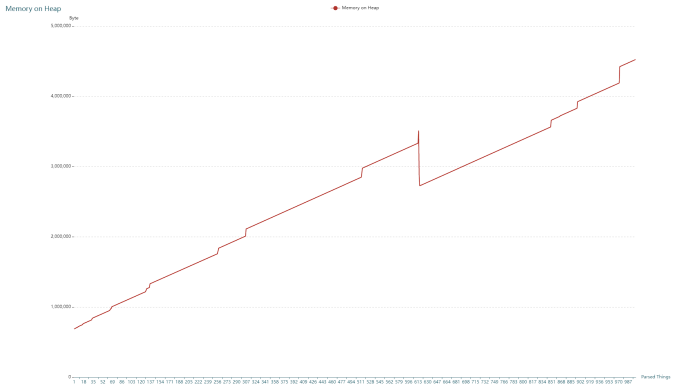


Abbildung 7.8: Speicherverbrauch von *Home Assistant* Objekten mit aktiver *GC*

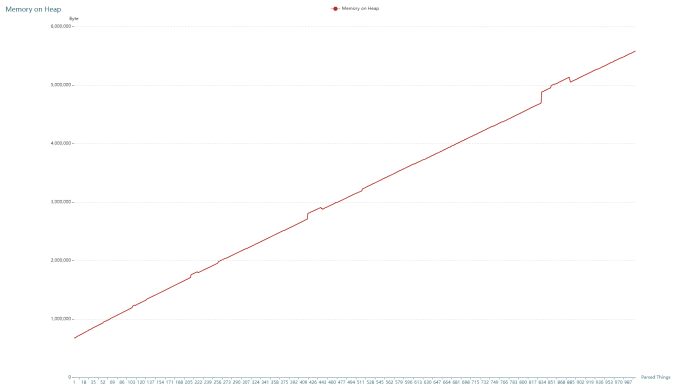


Abbildung 7.9: Speicherverbrauch von *HAP* Objekten mit mehreren Ursprüngen und gezielten *GC*

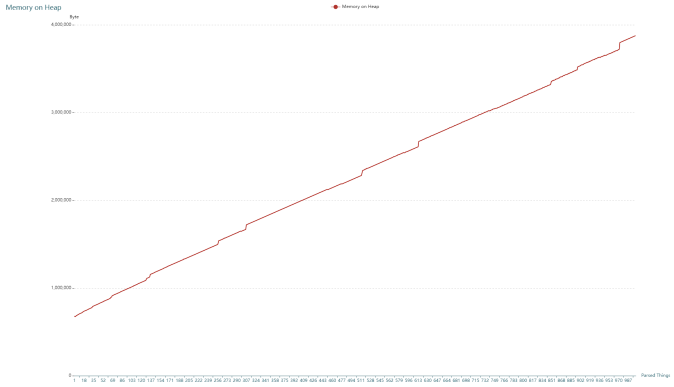


Abbildung 7.10: Speicherverbrauch von *HAP* Objekten mit einem Ursprung und gezielten *GC*

übrig geblieben sind. Der höchste Speicherverbrauch bei den Graphen mit nur einem Ursprung liegt bei jeweils 3875760 und 3828352*Bytes* und bei dem Test mit mehreren Ursprüngen liegt bei 5593352 *Bytes*. Nach jedem *GC* Cycle in den Graphen mit *GC*, ist der Speicherverbrauch nahezu identisch und geht dann auseinander bis zum nächsten *GC*. Es zeigt sich also, dass durch eine manuelle gezielte *GC* der Speicherverbrauch zu jedem gemessenen Zeitpunkt reduziert. Ein Aspekt, der dabei auch beachtet werden muss, ist die Laufzeit, wenn vor jedem Parser die *GC* ausgeführt wird, hat das Auswirkungen auf die Zeit, die es benötigt ein Objekt zu Parsen. Es wurde der Zeitverbrauch abgelesen, den es braucht eine Nachricht zu parsen, mit und ohne gezielter *Garbage Collection*. Zu sehen in Fig. 7.12 und Fig. 7.13. Es zeigt sich, dass der naive Ansatz, manuell die *GC* zu triggern keine kostenlose Operation ist, da die durchschnittliche Zeit, die es benötigt in Abhängigkeit mit den schon vorher geparsten Objekten mit der manuellen *GC* stärker ansteigt, als die durchschnittliche Zeit mit automatischen *GC*.

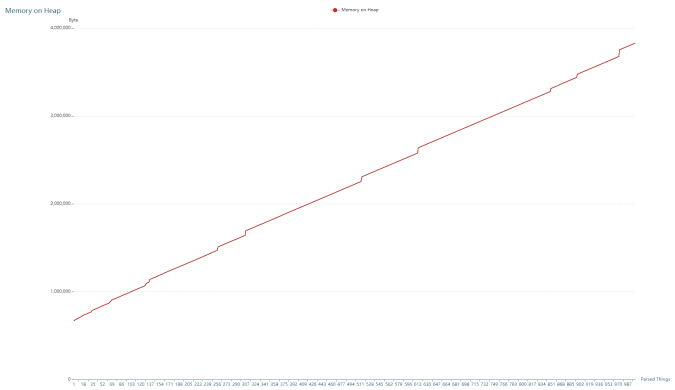


Abbildung 7.11: Speicherverbrauch von *Home Assistant* Objekten mit gezielten *GC*

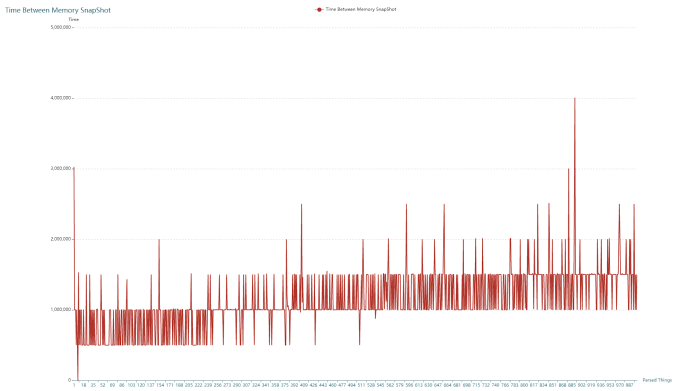


Abbildung 7.12: Zeitverbrauch von *HAP* Objekten mit mehreren Ursprüngen und aktiver *GC*

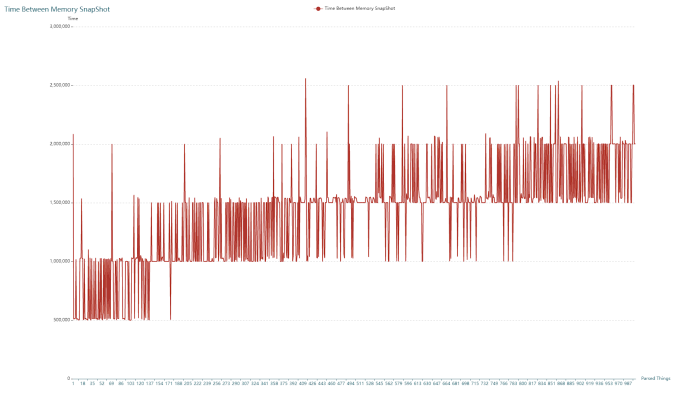


Abbildung 7.13: Zeitverbrauch von *HAP* Objekten mit mehreren Ursprüngen und gezielten *GC*

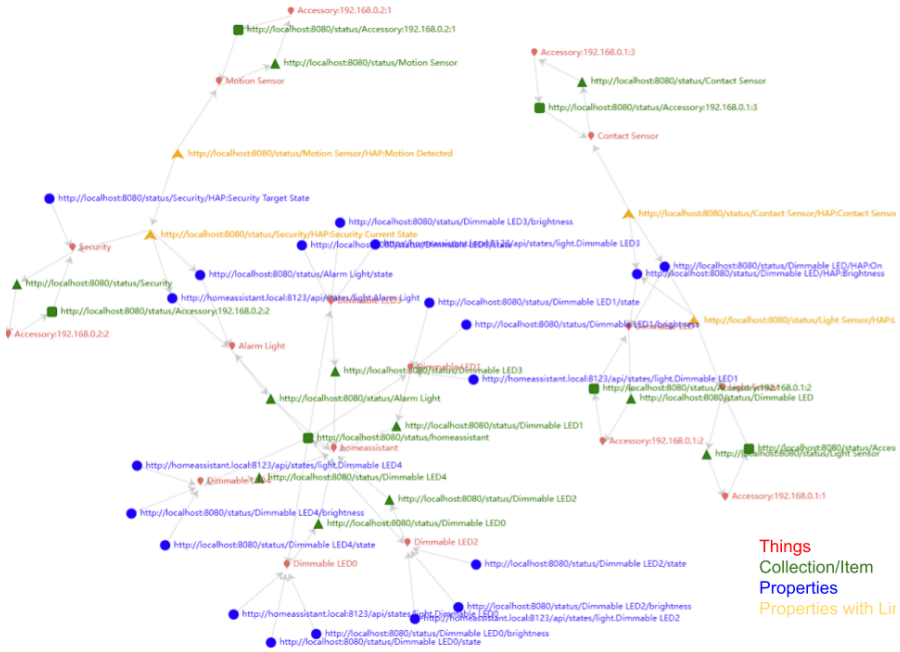


Abbildung 7.14: *LDT* Graph der alle drei Szenarios abbildet

7.4.4 Auswertung LDT Speicherverbrauch

Der Speicherverbrauch der Tests zeigen, dass der Speicherverbrauch der unterschiedlichen Geräte in der *WoT-TD* nah beieinander liegen und ein Erfolg ist. Der Speicherverbrauch von 1000 *HAP* Objekten und 1000 *Home Assistant* Objekten, die von demselben Ursprung kommen, haben einen nahen zu identischen Speicherverbrauch. Es zeigt sich aber auch, dass die Effizienz, indem die Geräte abgespeichert werden, abnimmt, wenn es mehrere Geräte gibt im Vergleich zu weniger Geräten aufgrund der *Collection*s. Es zeigt sich auch, dass es in *Go* nicht einfach ist genau zu wissen, wie viel Speicher gerade von dem Programm gerade tatsächlich benutzt wird und wie viel Speicher noch Überbleibsel vorheriger Funktionsaufrufe ist. Dieses Problem ist nicht naiv zu lösen, indem man gezielt die *GC* auslöst, da dies eine direkte Auswirkung auf die Laufzeit hat.

7.5 Evaluation Graph

In Fig. 7.14 zeigt sich, wie schnell und komplex die Abhängigkeiten von einem Smart Home System werden. Der Graph enthält alle Geräte, die notwendig sind, jedes der drei Szenarios aus Kapitel 7.2 darzustellen(siehe Fig. 7.16).

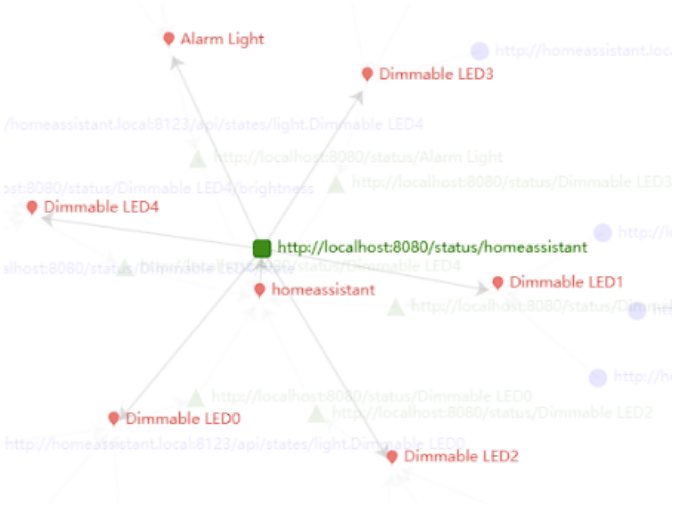


Abbildung 7.15: *collection*-Beziehung zwischen *homeassistant* und abhängigen Geräten

Abbildung 7.16: *Dimmable LED0* und *homeassistant*, verbunden durch eine *item*-Beziehung

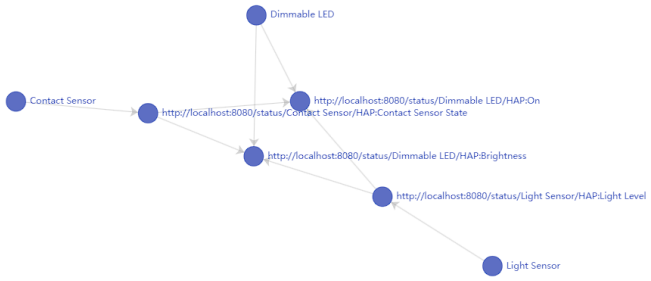


Abbildung 7.17: Eine Glühbirne, die abhängig von einem Kontakt und einem Lichtsensor ist

1. Das erste Szenario wird dadurch dargestellt, dass der *Home Assistant* Knoten mit jedem Gerät verbunden, der Teil von *Home Assistant* ist über den *Collection*-Knoten (siehe Fig. 7.15). Analog ist jedes Gerät über den *Link* Knoten mit *Home Assistant* verbunden

2. Das zweite Szenario wird wie im Kapitel 5.3 beschrieben von einem reduzierten Subgrafen dargestellt. In Fig.7.17 ist zu erkennen, dass sowohl die *On* als auch die *Brightness* Charakteristik von dem *Light Level* und dem *Contact Sensor State* der Sensoren beeinflusst wird. Es ist also klar erkennbar, dass diese beiden Charakteristiken neben der Glühbirne selbst von zwei weiteren Geräten beeinflusst werden.

3. Das dritte Szenario wird ebenfalls von einem reduzierten Subgrafen dargestellt. In diesem Beispiel stammt der Bewegungssensor und das Security-System aus dem *HAP* Protokoll, während die Lampe eine Home Assistant Lampe ist. In Fig.7.18 ist zu erkennen, dass einerseits das Security-System von dem Bewegungssensor beeinflusst wird und das Sicherheitssystem die Glühbirne beeinflusst. Das Sicherheitssystem hat zwei ausgehende Knoten, da es beim Verändern der Glühbirne, sowohl den Status des Smart Home Servers beeinflusst, als auch den Status des *LDT*s.

7.5.1 Fazit der Evaluation

Die Evaluation des *LDT* Gerätes zeigt, dass die Implementation eines *LDT* fähigen Gerätes, je nach Implementation und Szenario unterschiedlich starke Auswirkungen auf die Performance hat. Einerseits ließ sich erkennen, dass die Performance sowohl mit und ohne *LDT* Funktionalität sehr unterschiedlich sein kann und es immer in jedem Szenario Ausreißer gibt und eine Nachricht ein Vielfaches dauern kann als erwartet. Es ist kein Zusammenhang zwischen Ausreißer und *LDT* Funktionalität erkennbar. Im Median ist

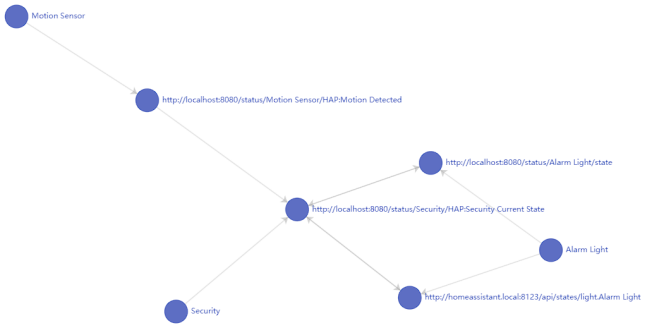


Abbildung 7.18: Ein *HomeKit Security Service* verbunden mit einem *HomeKit Motion Sensor* und einem *HomeAssistant light*

bei Sensoren eine Verschlechterung der Dauer eines Updates um das Zweifache zu erkennen, wenn ein Buffer benutzt wird. Ohne Buffer ist die Performance um ein Vielfaches schlechter. Es lässt sich argumentieren, dass bei den Sensor Tests auch das Worst-Case-Szenario getestet wurde, also dass alle Sensoren gleichzeitig ein Update schicken. Bei einer kleineren Anzahl an Sensoren ist der Unterschied zwischen Buffer und ohne Buffer nicht so ausschlaggebend. Bei dem Schalten von Glühbirnen über *Home Assistant* ist der Performance Unterschied im Verhältnis eher gering. Der Speicherverbrauch der *LDT* Geräte unterscheidet sich vor allem im Flashspeicher.
Als Fazit lässt sich ziehen, dass ein Smart Home Gerät mit *LDT* Funktionalität ausgestattet werden kann, es aber je nach Anwendungsgebiet angepasst werden muss. Es könnte sich zum Beispiel ein anderes Protokoll entwickelt werden, als das, was in dieser Arbeit vorgestellt wurde, um es zu erleichtern eine Vielzahl an Updates gleichzeitig zu schicken. Es könnte auch die Nachrichtengröße selbst reduziert werden, indem zum Beispiel die *UUID* reduziert wird und durch eine kürzere ID ersetzt wird, die für den *LDT* eindeutig Komponenten des *HAP* beschreibt.
Die Evaluation des *LDT* zeigt, dass die *Web of Things Thing Description* ein vielversprechendes Datenmodell für Smart Home Geräte ist. Der *LDT* hat die Möglichkeit unterschiedliche Abhängigkeiten wie Knotenpunkte festzustellen, wie ein Smart Home Server, der eine Abhängigkeit von vielen Geräten darstellt. Gleichzeitig lassen sich Automationen erfassen, die unterschiedliche Protokolle benutzen und die Abhängigkeit von Smart Home Geräten untereinander lässt sich auch feststellen. Der Speicherverbrauch unterschiedlicher Protokolle hat im *WoT-TD* Modell kaum einen Einfluss. Die Architektur hingegen eines Smart Homes hat einen stärkeren Einfluss auf den Speicherverbrauch des *LDT*. Im Falle eines zentralen Systems ist der Speicherverbrauch geringer als in einem dezentralen System, da die Anzahl an *Collection*s von der Anzahl an Geräten beeinflusst wird. Es hat sich auch gezeigt durch die naiven Experimente mit dem *GC*, dass es Optimierungsmöglichkeiten gibt, wie der Speicher verwaltet werden könnte. Dabei muss aber immer abgewägt werden, ob die zusätzlichen Zeitkosten, das wert sind.

Kapitel 8

Zusammenfassung

Die Zielsetzung dieser Arbeit war es durch eine Implementation und Evaluation zu überprüfen, ob ein *Longevity Digital Twin* in der Praxis anwendbar ist. Das Ziel des *LDT* ist es, alle Geräte in einem Smart Home zu modellieren und miteinander in Relation setzen zu können.
Vor der Implementation wurde theoretisch untersucht, welche Problemstellungen für einen *LDT* relevant sind und wie diese zu lösen sind. Dabei wurden auf Bootstrapping Mechanismen für das Internet of Things [39] eingegangen und es wurde argumentiert, wieso diese auch für ein *LDT* anwendbar sind. Das Ergebnis dieser Untersuchung ist, dass der *LDT* nicht alle Daten eines Smart Home Geräts benötigt, um dieses zu modellieren, solange das Gerät wichtige IDs an den *LDT* schickt, sodass er entsprechende Modelle generieren kann. Damit diese Modellierung eines Smart Home Geräts funktioniert, musste ein Datenmodell erstellt werden, das es ermöglicht, mit wenig Informationen Geräte komplett zu beschreiben. Die *Web of Things Thing Description(WoT-TD)* ist das Datenmodell, das für diese Arbeit adaptiert wurde. Die *WoT-TD* ermöglicht durch den *Context* eines jeden Modells genau zu definieren, aus welchem Protokoll dieses Gerät und die entsprechenden Attribute stammen. Es ist dadurch möglich beispielsweise nur anhand einer *UUID* eines *HomeKit Services* eine *Property* zu erstellen, die durch den Kontext genau beschreibt, wie diese *Property* zu verstehen ist. Auf diese Art und Weise wurden erfolgreich Modelle für *HomeKit, Matter* und *HomeAssistant* Geräte erstellt.
Damit die Abhängigkeit von Smart Home Geräten dargestellt werden kann, wurde ein Graph entworfen. Dafür wurde die Idee von *IoTSan* [36] aufgegriffen, Abhängigkeiten durch Interaktionen darzustellen. Der Graph wird anhand der *WoT-TD* Modelle aufgebaut, indem alle *Properties, Events, Actions* und *Links* eines jeden Model ausgelesen werden und diese werden durch das *href*-Feld miteinander verbunden. Dieses Vorgehen erlaubt es, einen Graphen zu erstellen, der ein komplettes Modell aller Interaktionen zwischen allen Geräten eines Smart Home Systems darstellt. Dieser Graph wird dann reduziert und so ist klar erkennbar, welche Geräte genau voneinander abhängig sind.
Es wurde dadurch ein Modell entworfen, das es erlaubt, Smart Home Geräte mit möglichst wenig Informationen eindeutig zu beschreiben und es ist in der Lage in einen Graphen konvertiert zu werden, der es ermöglicht Abhängigkeiten zu erkennen.
Damit dieses Modell auch in der Praxis überprüft werden kann, wurde auf einem *ESP32*

ein *HomeKit* Gerät mit der *HomeSpan* Bibliothek erstellt und unterschiedliche Kommunikationsmodelle entworfen. Einerseits gibt es das Modell, dass ein *LDT* Gerät mit dem *LDT* kommuniziert und ein direkter Datenaustausch möglich ist, andererseits gibt es das Modell, dass ein Gerät nur mit einem Zentralen Smart Home Server verbunden ist, und nur ein indirekter Datenaustausch möglich ist. Beide Modelle wurden in der Praxis implementiert. Es wurden unterschiedliche *HomeKit* Geräte implementiert, die mit der *HomeSpan* Bibliothek auf einem *ESP32* ausgeführt werden können und diese Implementation wurde in einem separaten Programm erweitert, um gleichzeitig auch eine Kommunikation mit dem *LDT* zu ermöglichen. Es wurde außerdem eine Verbindung zwischen *LDT* und *Home Assistant* Smart Home Server implementiert. Diese Verbindung kann dazu benutzt werden, um den Status eines *LDT* Geräts auf dem Server zu verifizieren oder auch um neue Modelle zu erstellen von Geräten, die nicht in direkter Verbindung mit dem *LDT* stehen.
Die Evaluation des *LDTs* zeigt, dass die Implementation des Graphen und des Datenmodells ein Erfolg war. Es lassen sich unterschiedliche Abhängigkeiten darstellen und Modelle für unterschiedliche Geräte erstellen. Abhängigkeiten, die darstellbar sind, sind Smart Home Geräte, die von ein oder mehreren anderen Geräten beeinflusst werden, eine Abhängigkeit zu einem Smart Home Server haben oder indirekt durch andere Geräte beeinflusst werden. Der Speicherverbrauch der Modelle selbst ist unabhängig vom Protokoll und unterscheidet sich nur durch den Aufbau des Smart Homes, da um so mehr Teilnehmer es in einem Smart Home gibt, das Modell komplexer wird. Die Evaluation des *LDT* Geräts zeigt hingegen, dass im direkten Vergleich zu einem Gerät ohne *LDT* teilweise starke Performance-Unterschiede zu erkennen sind. Bei der Evaluation mit den Sensoren ist eine doppelt so lange Zeit zu verzeichnen, die es benötigt ein Update zu schicken. Bei einer Glühbirne ist der Unterschied geringer, aber trotzdem merklich.
Diese Arbeit beweist, dass es möglich ist einen *LDT* zu implementieren, aber es gibt eine Vielzahl an weiteren Aspekten, die noch implementiert, werden müssen, um ein Smart Home System komplett zu modellieren. Es fehlen die Modellierung von *Actions* und *Events* der *WoT-TD* und es muss möglich sein Automationen von einem Smart Home Server auszulesen, die nicht durch den *LDT* erstellt wurden. Außerdem muss evaluiert werden, ob die Methoden, die in dieser Arbeit entworfen wurden, auch für andere Smart Home Protokolle und Systeme anwendbar sind. Ein weiterer Aspekt der Teil von zukünftigen Arbeiten sein sollte, ist das Entwerfen eines besseren Kommunikationsprotokolls für *LDT* Geräte, da die starke Performance Unterschiede, die Praktikabilität des *LDTs* beeinflussen
Der Verbindung von *Digital Twins*, *IoT* und Smart Homes ist ein vielversprechendes Konzept und diese Arbeit zeigt durch die Implementation des *LDTs* den Gewinn, diese Konzepte zu kombinieren und das Potenzial für zukünftige Arbeiten.

Anhang A

Anhang

A.1 Beispiele für Web of Things Thing Descriptions

A.1.1 Eine vollständige HAP WoT-TD

Listing A.1: Eine vollständige *HAP WoT-TD*, die ein dimmbare *HAP Lightbulb* modelliert, das sowohl mit dem *LDT* und einem *Home Assistant* Server verbunden ist.

```
{
"@context":[
  "https://www.w3.org/2022/wot/td/v1.1",
  {
  "HAP":"http://localhost:8080/hap"
  }
],
"@type":"HAP:public.hap.service.lightbulb",
"id":"http://192.168.178.64:2:2",
"title":"LDTDimLed2",
"base":"http://localhost:8080/status/LDTDimLed2",
"securityDefinitions":{
  "bearer_sc":{
  "scheme":"bearer",
  "in":"header",
  "format":"JWT",
  "alg":"HS256"
},
"nosec_sc":{
  "scheme":"nosec"
}
},
"security":"nosec_sc",
```

```
"properties":{
  "HAP:Brightness":{
  "@type":"HAP:public.hap.characteristic.brightness",
  "type":"integer",
  "forms":[
    {
    "op":"readproperty",
    "href":"http://localhost:8080/status/LDTDimLed2/HAP:
       Brightness",
    "contentType":"application/json",
    "security":"nosec_sc"
    },
    {
    "op":"writeproperty",
    "href":"http://localhost:8080/status/LDTDimLed2/HAP:
       Brightness",
    "contentType":"application/json",
    "security":"nosec_sc"
    },
    {
    "op":"readproperty",
    "href":"http://homeassistant.local:8123/api/states/light
       .ldtdimled2",
    "contentType":"application/json",
    "security":"bearer_sc"
    }
  ],
  "minimum":0,
  "maximum":100,
  "readOnly":false,
  "writeOnly":false,
  "observable":true
  },
  "HAP:On":{
  "@type":"HAP:public.hap.characteristic.on",
  "type":"boolean",
  "forms":[
    {
    "op":"readproperty",
    "href":"http://localhost:8080/status/LDTDimLed2/HAP:On",
    "contentType":"application/json",
    "security":"nosec_sc"
```

```
63      },
64      {
65      "op":"readproperty",
66      "href":"http://homeassistant.local:8123/api/states/light
           .ldtdimled2",
67      "contentType":"application/json",
68      "security":"bearer_sc"
69      }
70    ],
71    "readOnly":false,
72    "writeOnly":false,
73    "observable":true
74    }
75  },
76  "links":[
77    {
78      "href":"http://localhost:8080/status/Accessory:2",
79      "type":"application/td+json",
80      "rel":"collection"
81    }
82  ]
83  }
```

A.1.2 Eine vollständiges Home Assistant WoT-TD

Listing A.2: Eine vollständige *Home Assistant WoT-TD*, die ein*Home Assistant Light* modelliert, mit *Brightness* Attribute. Das Gerät ist sowohl mit dem *LDT* und einem *Home Assistant* Server verbunden ist.

```
1  {
2      "@context": [
3          "https://www.w3.org/2022/wot/td/v1.1",
4          {
5              "HAI": "https://www.home-assistant.io/
                   integrations"
6          }
7      ],
8      "@type": "HAI:light",
9      "id": "homeassistant.local:8123/2",
10     "title": "ldtdimled2",
11     "base": "http://localhost:8080/status/ldtdimled2",
```

```
12      "securityDefinitions": {
13          "bearer_sc": {
14              "scheme": "bearer",
15              "in": "header",
16              "format": "JWT",
17              "alg": "HS256"
18          },
19          "nosec_sc": {
20              "scheme": "nosec"
21          }
22      },
23      "security": "nosec_sc",
24      "properties": {
25          "brightness": {
26              "@type": "HAI:brightness",
27              "type": "number",
28              "forms": [
29                  {
30                      "op": "readproperty",
31                      "href": "http://localhost:8080/status/
                           ldtdimled2/brightness",
32                      "contentType": "application/json",
33                      "security": "nosec_sc"
34                  },
35                  {
36                      "op": "readproperty",
37                      "href": "http://homeassistant.local
                           :8123/api/states/light.ldtdimled2",
38                      "contentType": "application/json",
39                      "security": "bearer_sc"
40                  }
41              ],
42              "minimum": 1,
43              "maximum": 255,
44              "readOnly": false,
45              "writeOnly": false,
46              "observable": false
47          },
48          "state": {
49              "@type": "HAI:state",
50              "type": "string",
51              "forms": [
```

```
52                  {
53                      "op": "readproperty",
54                      "href": "http://localhost:8080/status/
                           ldtdimled2/state",
55                      "contentType": "application/json",
56                      "security": "nosec_sc"
57                  },
58                  {
59                      "op": "readproperty",
60                      "href": "http://homeassistant.local
                           :8123/api/states/light.ldtdimled2",
61                      "contentType": "application/json",
62                      "security": "bearer_sc"
63                  }
64              ],
65              "readOnly": false,
66              "writeOnly": false,
67              "observable": false
68          }
69      }
70  }
```

Literatur

[1] Amazon.com. *What is the Alexa Skills Kit?* 2023. URL: https://developer.amazon.com/en-US/docs/alexa/ask-overviews/what-is-the-alexa-skills-kit.html (besucht am 12. 04. 2023).

[2] Arduino. *loop()*. 2023. URL: https://www.arduino.cc/reference/en/language/structure/sketch/loop/ (besucht am 12. 04. 2023).

[3] Arduino. *setup()*. 2023. URL: https://www.arduino.cc/reference/en/language/structure/sketch/setup/ (besucht am 12. 04. 2023).

[4] Home Assistant. *Entities: integrating devices & services*. 2023. URL: https://developers.home-assistant.io/docs/architecture/devices-and-services/ (besucht am 12. 04. 2023).

[5] Home Assistant. *Light*. 2023. URL: https://www.home-assistant.io/integrations/light/ (besucht am 12. 04. 2023).

[6] Home Assistant. *REST API*. 2023. URL: https://developers.home-assistant.io/docs/api/rest/ (besucht am 12. 04. 2023).

[7] Home Assistant. *Sensor*. 2023. URL: https://www.home-assistant.io/integrations/sensor (besucht am 12. 04. 2023).

[8] Home Assistant. *State Objects*. 2023. URL: https://www.home-assistant.io/docs/configuration/state_object/ (besucht am 12. 04. 2023).

[9] Home Assistant. *WebSocket API*. 2023. URL: https://developers.home-assistant.io/docs/api/websocket (besucht am 12. 04. 2023).

[10] Home Assistant. *Home Assistant*. URL: https://www.home-assistant.io/ (besucht am 12. 04. 2023).

[11] The Go Authors. *A Guide to the Go Garbage Collector*. URL: https://tip.golang.org/doc/gc-guide (besucht am 12. 04. 2023).

[12] The Go Authors. *json*. URL: https://pkg.go.dev/encoding/json (besucht am 12. 04. 2023).

[13] C. Bormann und P. Hoffman. *Concise Binary Object Representation (CBOR)*. STD 94. RFC Editor, Dez. 2020.

[14] Arindom Chakraborty u. a. "Smart Home System: A Comprehensive Review". In: *Journal of Electrical and Computer Engineering* 2023 (März 2023), S. 1–30. DOI: 10.1155/2023/7616683.

[15] World Wide Web Consortium. *Documentation*. 2023. URL: https://www.w3.org/WoT/documentation/ (besucht am 12. 04. 2023).

[16] World Wide Web Consortium. *W3C Web of Things*. 2023. URL: https://www.w3.org/WoT/documentation/ (besucht am 12. 04. 2023).

[17] AZ-Delivery. *ESP-32 Dev Kit C V4*. URL: https://www.az-delivery.de/en/products/esp-32-dev-kit-c-v4 (besucht am 12. 04. 2023).

[18] *ESP32 Series Datasheet*. Version 4.2. Espressif Systems. 2023.

[19] Espressif. *Espressif/arduino-ESP32: Arduino core for the ESP32*. URL: https://github.com/espressif/arduino-esp32.

[20] Ltd. Espressif Systems (Shanghai) Co. *Non-volatile Storage Library*. 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/storage/nvs_flash.html (besucht am 12. 04. 2023).

[21] The Apache Software Foundation. *Aoache ECHarts*. 2023. URL: https://echarts.apache.org/en/index.html (besucht am 12. 04. 2023).

[22] DIGITAL-ROOM GmbH. *FAQ – die häufigsten Fragen zu Matter*. 2023. URL: https://matter-smarthome.de/ (besucht am 12. 04. 2023).

[23] Google. *Was ist Google Home?* 2023. URL: https://home.google.com/intl/de_de/what-is-google-home/ (besucht am 12. 04. 2023).

[24] Jascha Grübel u. a. "The Hitchhiker's Guide to Fused Twins: A Review of Access to Digital Twins in situ in Smart Cities". In: Juni 2022. DOI: 10.48550/arXiv.2202.07104.

[25] Frank-Oliver Grün. *Handbuch Smart Home*. Stiftung Warentest, 2022.

[26] *HomeKit Accessory Protocol Specification Non-Commercial Version*. Release R2. Apple Inc. 2019.

[27] HomeSpan. *HomeSpan*. 2023. URL: https://github.com/HomeSpan/HomeSpan (besucht am 12. 04. 2023).

[28] Apple Inc. *Developing apps and accessories for the home*. 2023. URL: https://developer.apple.com/apple-home/ (besucht am 12. 04. 2023).

[29] Apple Inc. *Home App*. 2023. URL: https://www.apple.com/de/home-app/ (besucht am 12. 04. 2023).

[30] Ricardo Izzi. *SMART HOME MIT WIFI-FUNKTION*. 2021. URL: https://www.net4energy.com/de-de/smart-living/smart-home-wifi (besucht am 12. 04. 2023).

[31] Takuki Kamiya u. a. *Web of Things (WoT) Thing Description 1.1*. W3C Working Draft. https://www.w3.org/TR/2022/WD-wot-thing-description11-20220803/. W3C, Aug. 2022.

[32] Google Ireland Limited. *The Device Data Model*. 2022. URL: https://developers.home.google.com/matter/primer/device-data-model (besucht am 12. 04. 2023).

[33] *Matter Application Cluster Specification*. Version 1.0. Connectivity Standards Alliance. 2022.

[34] *Matter Device Library Specification*. Version 1.0. Connectivity Standards Alliance. 2022.

[35] *Matter Specification*. Version 1.0. Connectivity Standards Alliance. 2022.

[36] Dang Tu Nguyen u. a. "IotSan: Fortifying the Safety of IoT Systems". In: *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT '18. Heraklion, Greece: Association for Computing Machinery, 2018, S. 191–203. ISBN: 9781450360807. DOI: 10.1145/3281411.3281440. URL: https://doi.org/10.1145/3281411.3281440.

[37] PlatformIO. *PlatformIO IDE*. 2023. URL: https://www.az-delivery.de/en/products/esp-32-dev-kit-c-v4 (besucht am 12. 04. 2023).

[38] Thingweb project. *Thing Description Playground*. 2022. URL: http://plugfest.thingweb.io/playground/ (besucht am 12. 04. 2023).

[39] Lukas Reinfurt u. a. "Internet of Things Patterns for Device Bootstrapping and Registration". In: *Proceedings of the 22nd European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM, 2017. DOI: 10.1145/3147704.3147721.

[40] Laila Salman u. a. "Energy efficient IoT-based smart home". In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. 2016, S. 526–529. DOI: 10.1109/WF-IoT.2016.7845449.

[41] Angira Sharma u. a. "Digital Twins: State of the art theory and practice, challenges, and open research questions". In: *Journal of Industrial Information Integration* 30 (2022), S. 100383. ISSN: 2452-414X. DOI: https://doi.org/10.1016/j.jii.2022.100383. URL: https://www.sciencedirect.com/science/article/pii/S2452414X22000516.

[42] Vivek Singhania. "The Internet of Things: An Overview Understanding the Issues and Challenges of a More Connected World". In: Okt. 2015.

[43] Kedar Sovani. *Matter: Clusters, Attributes, Commands*. 2021. URL: https://blog.espressif.com/matter-clusters-attributes-commands-82b8ec1640a0 (besucht am 12. 04. 2023).

[44] go-echarts dev team. *go-echarts*. 2023. URL: https://github.com/go-echarts/go-echarts (besucht am 12. 04. 2023).

[45] Daniel Wroclawski. *The Matter Smart Home Standard Is Finally Available: Here's What It Means for Your Home*. 2022. URL: https://www.consumerreports.org/smart-home/matter-smart-home-standard-faq-a9475777045/ (besucht am 12. 04. 2023).

[46] Peter Zdankin u. a. “A Digital-Twin Based Architecture for Software Longevity in Smart Homes”. In: Okt. 2022. DOI: 10.1109/ICDCS54860.2022.00070.

Versicherung an Eides Statt

Ich versichere an Eides statt durch meine untenstehende Unterschrift,

- dass ich die vorliegende Arbeit - mit Ausnahme der Anleitung durch die Betreuer - selbstständig ohne fremde Hilfe angefertigt habe und
- dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus fremden Quellen entnommen sind, entsprechend als Zitate gekennzeichnet habe und
- dass ich ausschließlich die angegebenen Quellen (Literatur, Internetseiten, sonstige Hilfsmittel) verwendet habe und
- dass ich alle entsprechenden Angaben nach bestem Wissen und Gewissen vorgenommen habe, dass sie der Wahrheit entsprechen und dass ich nichts verschwiegen habe.

Mir ist bekannt, dass eine falsche Versicherung an Eides Statt nach §156 und §163 Abs. 1 des Strafgesetzbuches mit Freiheitsstrafe oder Geldstrafe bestraft wird.

Essen, 13. April 2023 _____

(Ort, Datum) (Vorname Nachname)