HAW HAMBURG

BACHELOR THESIS
Fabian Erdmann

Undo-Operationen in einem Publish-Subscribe basierten Smart Home

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Fabian Erdmann

Undo-Operationen in einem Publish-Subscribe basierten Smart Home

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Lehmann
Zweitgutachter: Prof. Dr. Kai von Luck

Eingereicht am: 26. Februar 2023

Fabian Erdmann

Thema der Arbeit

Undo-Operationen in einem Publish-Subscribe basierten Smart Home

Stichworte

Smart-Home, Undo, Publish, Subscribe

Kurzzusammenfassung

Diese Bachelorarbeit fokussiert sich auf die Konzeption und Implementierung einer "Undo"-Funktionalität in einem Publish-Subscribe-basierten Smart Home System. Für die Realisierung einer Undo-Option ist der Zugriff auf vorherige Zustände eines Aktors unerlässlich. Verschiedene Methoden zur Datenhaltung, insbesondere die Nutzung von Logfiles, einer Datenbank und einer Event-Streaming-Plattform, werden in dieser Arbeit gegeneinander abgewogen. Nach sorgfältiger Betrachtung der Alternativen erfolgte die Implementierung mit dem Ansatz der Verwendung eines Logfiles in einem Python-Skript. Außerdem wurden zwei verschiedene Arten von Undos, das chronologische und das Undo vom Undo vorgestellt und Unterschiede aufgezeigt. Am Ende wurde die implementierte Lösung anhand der Anforderungen inklusive der Round Trip Latenz ausgewertet.

Fabian Erdmann

Title of Thesis

Keywords

Smart-Home, Undo, Publish, Subscribe

Abstract

This Bachelor's thesis focuses on the conception and implementation of an "Undo" functionality within a Publish-Subscribe-based Smart Home system. Access to an actuator's prior states is crucial for realizing an Undo option. Various data storage methods, specifically utilizing logfiles, a database, and an event-streaming platform, are weighed against

each other in this work. After careful consideration of the alternatives, the implementation was carried out using a logfile implemented within a Python script. In addition, two different types of undo, the chronological undo and the undo of the undo, were presented and the differences pointed out. Finally, the implemented solution was analysed based on the requirements, including the round trip latency.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Karl Kraus hat einmal gesagt: “Das Gegenteil von gut ist nicht schlecht, sondern gut gemeint”[32]. Viele der Automatisierungen und Aktionen vom Smart Home sind gut gemeint, aber nicht immer genau das, was der Benutzer[1] möchte. Damit wird, mit immer mehr Automatisierungen und Eingriffen des Smart Home, die Steuerung des Zuhauses aus der Hand der Benutzer genommen. Smart Home Systeme werden immer verfügbarer und erschwinglicher[21, S. 1].

Mit der zunehmenden Verbreitung von Smart Homes in immer mehr Wohnungen und der Zunahme der Interaktionsmöglichkeiten zwischen Mensch und Maschine steigt auch die Wahrscheinlichkeit von Missverständnissen zwischen den beiden Parteien. Viele verschiedene programmierte Automatisierungen können zu Effekten führen, die der Benutzer in bestimmten Situationen nicht wünscht. Oder ein Zustand wird unbeabsichtigt verändert und der Benutzer möchte einfach zum vorherigen Zustand zurückkehren. Der Benutzer braucht eine Möglichkeit, diese gut gemeinten Befehle aufzuheben[28].

Um dem Benutzer ein Stück weit die Kontrolle über das Smart Home zurückzugeben und nicht mehr hilflos den Automatisierungen ausgeliefert zu sein, wurde diese Arbeit erstellt.

1.2 Ziel

Das Ziel dieser Arbeit ist die Integration einer Möglichkeit, Befehle in einer Smart Home Umgebung rückgängig zu machen und einen vorherigen Zustand wiederherzustellen.

[1] Bei Personenbezeichnungen wird in dieser Arbeit die männliche Form verwendet, um die Lesbarkeit zu erleichtern. Diese Bezeichnungen gelten grundsätzlich für alle Geschlechter und sind nicht wertend.

Dabei soll es keine Rolle spielen, ob der Befehl von einem Menschen oder einer Maschine ausgelöst wurde. Diese Undo-Möglichkeit soll benutzerfreundlich und einfach in bestehende Smart Home Umgebungen integriert werden können. Ein weiteres Ziel ist es verschiedene Möglichkeiten der Persistierung vorzustellen und eine geeignete Möglichkeit für diese Arbeit zu finden. Als Umgebung für das Experiment dient das Living Place Labor[8] als Teil des Creative Space for Technical Innovations[2] an der HAW-Hamburg, in dem bereits eine Smart Home Umgebung aufgebaut ist.

1.3 Aufbau

Im Analysekapitel der Arbeit wird der fachliche Kontext zum Thema hergestellt und das Problem dargestellt. Anschließend wird die Laborumgebung präsentiert und wichtige Komponenten aufgezeigt. Danach werden zwei verschiedene Möglichkeiten zur Interpretation von Undo und vier Persistierungsarten im Smart Home beschrieben. Nachdem die Möglichkeiten vorgestellt wurden, erfolgt eine Anforderungsdefinition für das zu implementierende Undo-Tool.

Das darauf folgende Kapitel beschäftigt sich mit dem Design und der Implementierung der Lösung. Angefangen wird mit der Vorstellung des Konzepts und der Erläuterung der Wahl der Persistenz. Danach wird die umgesetzte Lösung in die Schichten Präsentation, Applikation und Datenbank unterteilt und präsentiert.

Abschließend erfolgt eine Bewertung der Umsetzung im Hinblick auf die Anforderungen sowie ein Fazit der Arbeit.

2 Analyse des Konzeptes

Im folgenden Kapitel wird die Einbettung der Arbeit in den übergeordneten fachlichen Kontext gebracht. Danach wird das Problem dieser Arbeit anhand eines Beispiels erläutert und die Projektumgebung näher vorgestellt. Im darauffolgenden Kapitel werden zwei verschiedene Möglichkeiten zur Interpretation eines Undo-Vorgangs aufgezeigt und anschließend unterschiedliche Persistenzsysteme vorgestellt. Zum Abschluss werden die ausgearbeiteten Anforderungen an das System definiert.

2.1 Smart Home Systeme

Ein Smart Home besteht im Wesentlichen aus einer zentralen Steuereinheit und diversen Aktoren und Sensoren, die mit der Steuereinheit verbunden sind[22]. Sensoren nehmen Daten aus der Umgebung auf und stellen sie der zentralen Steuereinheit zur Verfügung. Aktoren sind in der Lage, durch Manipulation physischer Objekte aktiv in die Umwelt einzugreifen und diese zu verändern.

Die zentrale Steuereinheit werden durch Bridges von Herstellern wie Homematic[6] oder durch Open-Source-Projekte wie HomeAssistant[17] realisiert. Alternative Hersteller wie Zigbee oder Phillips Hue Bridge bieten weitere Arten von zentralen Steuereinheiten an. Diese Bridges verbinden die Aktoren und Sensoren mit der herstellereigenen App oder auch Plattformen zur Sprachsteuerung des Gerätes durch Amazon (Alexa), Google oder Microsoft (Bing). Bei der Anbindung der Aktoren an die Bridge gibt es folgende Möglichkeiten, die hauptsächlich genutzt werden[16]:

- WiFi
- Bluetooth
- IEEE 802.15.4

- Z-Wave
- LTE-Advanced

Beispiele für Aktoren und Sensoren wären:

- Heizungsthermostate
- Fensteransteuerung
- Beschattungsansteuerung
- Umweltsensoren wie Temperatur, Luftfeuchtigkeit und Luftdruck

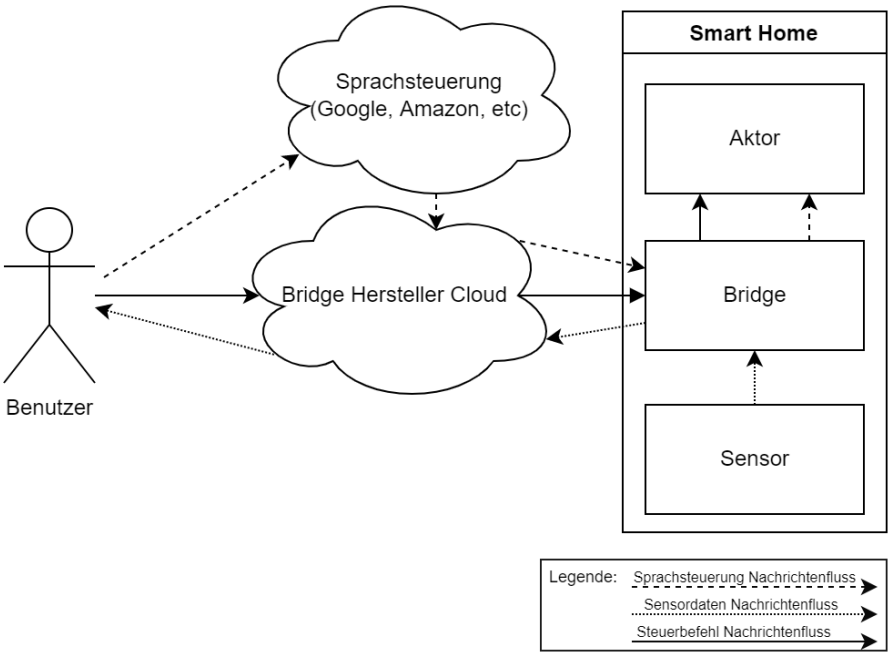


Abbildung 2.1: Nachrichtenfluss Smart Home

Die zentrale Steuereinheit oder Bridge vereint alle Aktoren und Sensoren in einem System, um diese Geräte von einem Punkt aus steuern zu können. Dies hat den Vorteil, dass der Benutzer nur mit einer einzigen Schnittstelle interagieren muss, um die verschiedenen Aktoren und Sensoren im Smart Home schalten zu können.

Für die Steuerung der Bridge durch den Nutzer setzen viele Hersteller auf eine eigene Cloud-Lösung mit entsprechender App für das Smartphone[16]. Ein beispielhafter Nachrichtenfluss eines Sprachbefehls und einer Ansteuerung über die Applikation ist in der

Abbildung 2.1 sichtbar. Die Cloud des Sprachbefehls ist hier häufig von der Cloud des Bridge-Herstellers getrennt und Befehle werden zwischen den Clouds geroutet. Bei einer lokalen Lösung wird die Verbindung direkt zwischen Nutzer und Bridge hergestellt, die Cloud-Lösung des Herstellers entfällt oder kann alternativ zugeschaltet werden.

Der Vorteil der Anbindung möglichst vieler Geräte an eine Bridge liegt darin, dass die auf der Bridge erstellten Automatisierungen alle Komponenten des Smart Home beeinflussen können. Beispielsweise kann der Benutzer, anstatt einem Heizungsaktor einen Sollwert für die Heizung vorzugeben, der Bridge das Ziel vorgeben, die Temperatur auf einen bestimmten Wert zu bringen. Die Bridge kann dann mithilfe vordefinierter Automatismen selbst entscheiden, wie dieses Ziel erreicht werden kann. Es ist denkbar, dass die Zieltemperatur durch das Öffnen eines Fensters erreicht wird, anstatt ein Heizungsthermostat zu aktivieren. Darüber hinaus ist es für den Nutzer einfacher, sich nur mit einer Brücke verbinden zu müssen, anstatt verschiedene Anwendungen für unterschiedliche Komponenten verschiedener Hersteller zu verwenden.

Es gibt verschiedene Versuche von großen Herstellern einen Standard für Smart Home Komponenten zu definieren. Mit dem Matter Standard haben Amazon, Apple, Google, Comcast und die Connectivity Standards Alliance (ehemals Zigbee) eine Projektgruppe ins Leben gerufen, um einen Standard zu entwickeln[1]. Dieser Matter Home Automation Standard ist ein offener, lizenzfreier und proprietärer Verbindungsstandard für Smart Home Anwendungen, der von immer mehr IoT-Komponenten unterstützt wird.

Ziel des Smart Home ist es, dem Nutzer die Steuerung einzelner Aktoren teilweise oder ganz abzunehmen[25]. Dies wird erreicht, indem Sensoren ausgewertet und Entscheidungen zur Steuerung von Aktoren vom Smart Home selbstständig getroffen werden[18].

2.2 Kommunikation mithilfe von Publisch Subscribe (MQTT)

Für die Kommunikation der einzelnen Aktoren und Sensoren mit der Bridge wird in IoT-Anwendungen häufig die Publish-Subscribe-Architektur[31] verwendet. Ziel dieser Architektur ist es, die Kommunikation zwischen mehreren Komponenten zu vereinfachen.

Ein häufiges Problem bei lose gekoppelten Anwendungen, wie z.B. einem Smart Home, ist es, die Kommunikation zwischen den Komponenten so zu ermöglichen, dass Nachrichten gezielt und effizient versendet werden können. Die lose Kopplung aller Komponenten ist mit der Publish-Subscribe-Architektur einfach zu realisieren und bietet viele Vorteile für Smart Home Anwendungen.

- leichte Skalierbarkeit ohne anlernen neuer Geräte
- leichte Integration des Brokers in das Smart Home
- Wird von vielen Smart Home Anwendungen und Aktoren unterstützt

In einer Publish-Subscribe-Architektur kennt jede Komponente nur einen Kommunikationspartner. Dieser Kommunikationspartner oder Broker kennt alle Komponenten und verteilt die Nachrichten entsprechend der Topics. Die Kommunikationspartner selbst müssen sich nicht aufeinander beziehen, dadurch wird eine räumliche oder referenzielle Entkopplung erreicht[30, S. 54].

Das Topic wird bei jedem Nachrichtenversand vom Sender, einem Publisher, angegeben und dient als Zuweisung, welche Komponenten die Nachricht erhalten. Dies geschieht, indem sich die Empfänger im Voraus mit dem entsprechenden Topic bei dem Broker registrieren. Durch dieses Subscribe hat der Broker Kenntnis, wie die Komponente erreichbar ist und welche Nachrichten für diese Komponente relevant sind.

Für die abonnierten Komponenten ist es irrelevant, von welcher Komponente die Nachricht stammt.

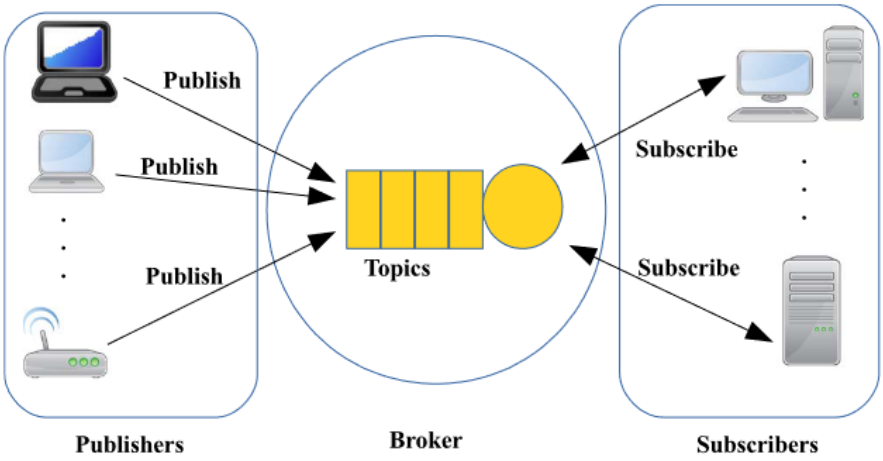


Abbildung 2.2: Die Architektur von MQTT[16, S. 8]

Das MQTT-Protokoll[9] wird häufig in IoT-Anwendungen und Smart-Home-Umgebungen eingesetzt[29] um eine Publish-Subscribe-Architektur umzusetzen. Dabei handelt es sich um ein Netzwerkprotokoll, das für die Kommunikation von Maschine zu Maschine (M2M) entwickelt und von der Organization for the Adanvement of Structured Information Standards (OASIS) standardisiert wurde[12].

Die Architektur von MQTT ist in der Abbildung 2.2 aus der Publikation "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications"[16, S. 8] dargestellt. In der Mitte der Abbildung befindet sich der Broker, der Nachrichten empfängt und verteilt. Auf der linken Seite befinden sich verschiedene Publisher, die Nachrichten versenden, während auf der rechten Seite die Komponenten dargestellt sind, die Topics abonniert haben. Ein Publisher kann auch auf verschiedene Topics abonnieren und gleichzeitig auf andere Topics publischen.

2.3 Aufbau der Laborumgebung Living Place

Das Smart Home Labor der Hochschule für Angewandte Wissenschaften Hamburg im Creative Space for Technical Inovations[2] (CSTI) ist das Living Place[8] (LP). Das LP bietet Studierenden und Forschenden die Möglichkeit, verschiedene Smart Home Technologien in einer Laborumgebung aufzubauen und Anwendungen zu erforschen.

Es besteht aus einem großen Loft, das durch verschiedene Trennwände in folgende Räume unterteilt ist:

- Küche
- Schlafzimmer
- Lounge
- Esszimmer
- Bad

Die komplette Übersicht über die Architektur des LP ist in der Abbildung 2.3 dargestellt. Als Bridge des Living Place wird das Open Source Projekt HomeAssistant[17] genutzt.

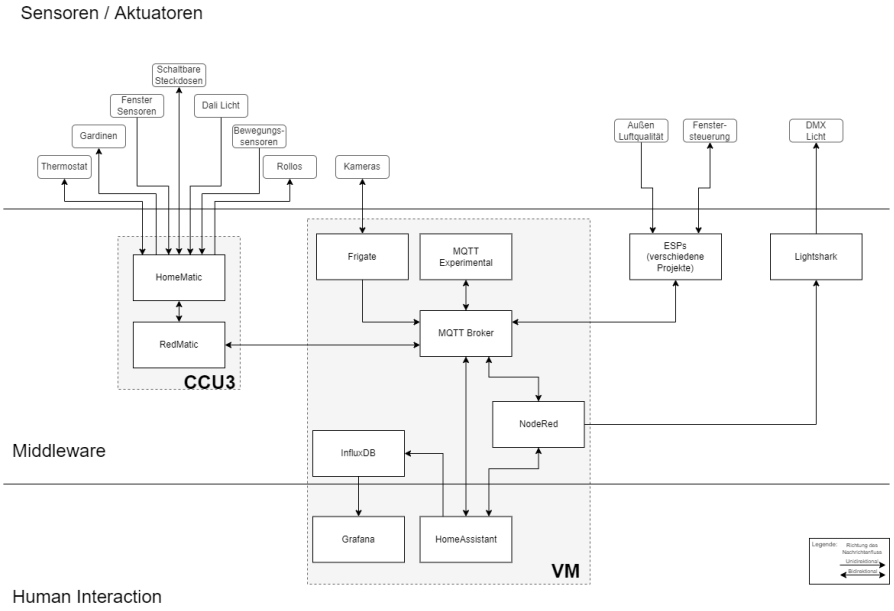


Abbildung 2.3: Architektur des Living Place

Diese Bridge ist in der Abbildung 2.3 im Human Interaction Layer dargestellt. HomeAssistant ermöglicht die Integration einer Vielzahl unterschiedlicher Komponenten verschiedener Hersteller, die über ein gemeinsames Frontend gesteuert werden können.

Zusammen mit Grafana bildet das Frontend von HomeAssistant in diesem Layer die Möglichkeit für den Benutzer, mit dem Smart Home zu interagieren. Grafana[4] ist eine Open-Source-Anwendung zur grafischen Darstellung von Forschungsdaten, die in einer InfluxDB[7] des LP gespeichert sind.

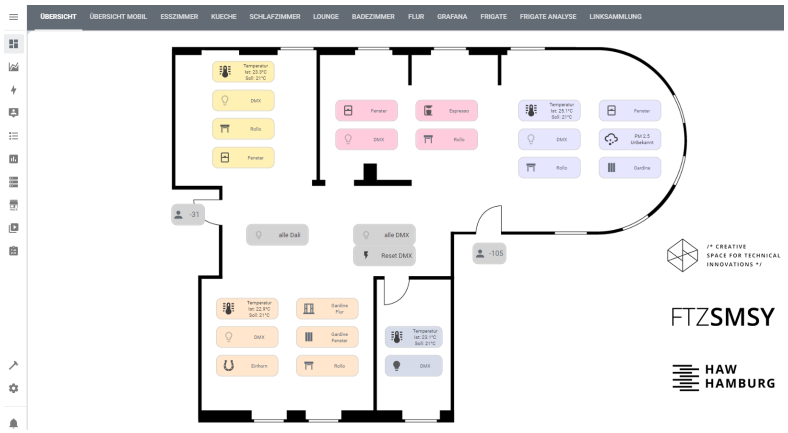


Abbildung 2.4: Das Lovelace Dashboard des LP

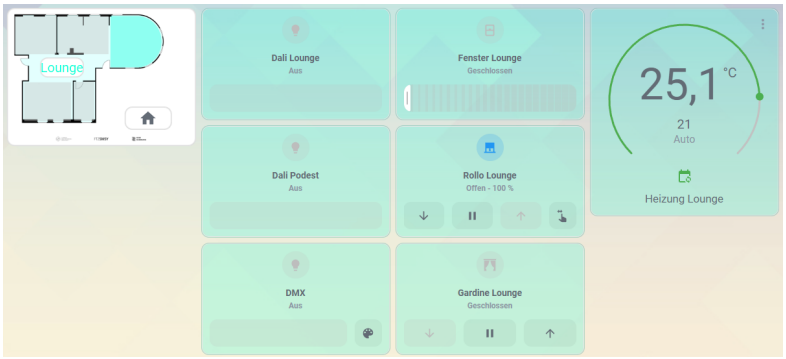


Abbildung 2.5: Lovelace Dashboard der Lounge im LP

Das Dashboard von HomeAssistant, Lovelace genannt, ist eine Webschnittstelle, die vom Benutzer angepasst werden kann. So können in verschiedenen Dashboards unterschiedliche Cards ausgewählt werden, um Informationen von Sensoren anzuzeigen und Aktoren anzusteuern.

Eine Übersicht über das Dashboard des LP zeigt die Abbildung 2.4. Von hier aus können die verschiedenen Räume angewählt oder Aktoren direkt geschaltet werden. Als Hintergrund dient der Raumplan des LP. Dadurch ist eine einfache Zuordnung der Aktoren und Sensoren zu den Räumen möglich. Detaillierte Informationen finden sich in den Registerkarten des Dashboards. Diese werden im HomeAssistant Views genannt. Ein solcher View für die Lounge ist in der Abbildung 2.5 dargestellt. Hier werden die Zustände der Aktoren mithilfe von Cards detaillierter visualisiert. Dieses Frontend ist auch als Applikation für Mobiltelefone verfügbar.

Das LP soll die Integration verschiedenster Technologien ermöglichen, indem ein möglichst offener Kommunikationsstandard verwendet wird. Aus diesem Grund ist ein MQTT Broker für die Kommunikation im LP integriert. Dieser MQTT Broker ist in der Abbildung 2.3 in der Middleware zu finden und wird mithilfe eines Mosquitto MQTT Brokers[3] realisiert. Zusätzlich zum zentralen MQTT Broker ist ein experimenteller MQTT Broker integriert, um kleinere Projekte schnell integrieren zu können, ohne die Infrastruktur des gesamten LPs zu gefährden. Bei Fehlern im Projekt kann der experimentelle MQTT Broker abgeschaltet werden, ohne die Grundfunktionalität des LPs zu beeinträchtigen.

HomeAssistant bietet nur Möglichkeiten für einfache “If this than that”Automatisierungen. Aus diesem Grund wurde das LP um NodeRed[10] erweitert. Bei NodeRed handelt es sich um ein browserbasiertes grafisches Entwicklungswerkzeug. Es basiert auf der Programmiersprache JavaScript. Komplexe Programmierungen sind auch ohne Vorkenntnisse

einer Programmiersprache mithilfe von Bausteinen, den Nodes, möglich. Diese Automatisierungen lassen sich zudem leicht protokollieren und sind somit gut für eine Forschungsumgebung geeignet.

NodeRed, mit seinen tiefgreifenden Programmier- und Protokollierungsmöglichkeiten, kombiniert mit einem MQTT Broker, schafft für das LP eine flexible Möglichkeit, verschiedene Komponenten zu integrieren. Zusammen mit HomeAssistant, der auf Smart Home Anwendungen spezialisiert ist, wird diese Kombination zu einem Smart Home, das sowohl für den Alltag als auch für die Forschung geeignet ist.

Als Datenbank kommt im LP eine InfluxDB zum Einsatz. Sichtbar in der Middleware-Schicht der Abbildung 2.3.

Die oben beschriebenen Komponenten der Middleware laufen auf einer eigenen virtuellen Maschine (VM) innerhalb der Infrastruktur des CSTI. Jede Komponente wird durch einen eigenen Docker Container virtualisiert.

Neben der VM befindet sich im Middleware Layer die Bridge von Homematic des Herstellers eQ-3 AG[6]. Die Anbindung der Homematic Produkte an die Bridge erfolgt über den von der eQ-3 AG entwickelten Funkstandard BidCoS[15]. Zusätzlich enthält die Bridge, ähnlich wie die VM, eine eigene NodeRed Instanz. Diese wird vom Hersteller RedMatic genannt. In RedMatic können MQTT-Verbindungen von und zu HomeMatic-Komponenten hergestellt werden, indem spezielle Nodes des Herstellers verwendet werden.

In jedem Raum sind verschiedene DMX Lichtleisten installiert, die auf verschiedene DMX Universen verteilt sind. Mit dem Lightshark[5] wird dieses Ambiente Licht des LP gesteuert. Es handelt sich um eine DMX-Steuereinheit, die über das OSC-Protokoll[13] direkt vom NodeRed der Infrastruktur angesteuert wird.

Zuletzt beinhaltet der Middleware Layer verschiedene IoT Geräte, die meist auf ESP32 Mikrocontrollern basieren. Diese sind über eine MQTT Bibliothek mit der Infrastruktur des Living Place verbunden.

Im Sensor und Aktor Layer der 2.3 werden alle Sensoren und Aktoren dargestellt.

2.4 Darstellung des Problems

Die Integration von Smart Homes in unser tägliches Leben bringt viele Vorteile mit sich, indem sie den Komfort, die Energieeffizienz und die Sicherheit verbessert[22]. Sie birgt aber auch das Potenzial für unbeabsichtigte Aktionen oder das Auslösen von Automatisierungsregeln, die nicht vom Nutzer initiiert wurden. Gerade das plötzliche Auslösen von Aktionen ohne Zutun des Nutzers kann dazu führen, dass der Nutzer das Gefühl bekommt die Kontrolle zu verlieren.

Stellen wir uns z.B. einen Bewohner vor, der im Wohnzimmer in Ruhe ein Buch liest. Während der Bewohner liest, stellt das Smart Home eine Verschlechterung der Luftqualität im Wohnzimmer fest. Das Smart Home wird nun aufgrund der eingestellten Automatisierungen das Fenster im Wohnzimmer öffnen, um frische Luft hereinzulassen. Um die Luftqualität in den Räumen zu erhalten, ist diese Automatisierung grundsätzlich eine sinnvolle Erweiterung des Smart Home. Im Verhalten des Smart Home liegt also kein Fehler vor.

Die Umgebung des Bewohners wurde jedoch verändert, ohne dass er dies wollte oder zugestimmt hat. Anstatt in Ruhe das Buch weiterlesen zu können, dringen nun störende Geräusche von draußen herein.

Der Bewohner hat durch diesen Eingriff einen Teil der Kontrolle über die Wohnung verloren. Was mit der Wohnung geschieht, wird nicht mehr nur durch den Willen des Bewohners, sondern auch durch die Automatisierungen des Smart Home bestimmt. Dem Bewohner muss also die Möglichkeit gegeben werden, in diese Vorgänge wieder eingreifen zu können. Anstatt ungewollte Veränderung der Umgebung durch das Smart Home einfach hinzunehmen, muss der Bewohner einen Befehl erhalten, den alten Zustand wiederherzustellen.

Die Möglichkeit, automatisch ausgeführte Befehle jederzeit rückgängig machen zu können, schafft eine Basis für die Akzeptanz des Smart Home durch den Bewohner. In dieser Bachelorarbeit wird eine Möglichkeit zur Implementierung einer solchen Funktion beschrieben.

2.4.1 Arten von Undo

In dieser Arbeit werden zwei grundlegende Arten von Undos unterschieden, die ein Benutzer erwarten könnte. Diese sind ein chronologisches Undo und ein Undo von Undo[20]. Neben diesen beiden Arten gibt es auch nichtlineare oder selektive Undos[33]. Bei diesen Undos wird nur ein selektiver Teil einer Liste von Zuständen rückgängig gemacht[34]. Da dies im Prinzip nur eine Auswahlmöglichkeit für ein chronologisches Undo ist und dessen Funktionen nur erweitert, wird es hier nicht weiter betrachtet. Die beiden betrachteten Undos werden im Folgenden anhand der beiden Abbildungen 2.6 und 2.7 genauer dargestellt.

Die Zustände einer Komponente werden in beiden Abbildungen durch Blöcke repräsentiert, die mit einem "S" und einer Zahl fortlaufend nummeriert sind. Jeder Turm aus Blöcken steht für einen einzelnen Aktor zu verschiedenen Zeitpunkten, der nach rechts weitere Blöcke, das heißt Zustände, erhält. Es müssen mindestens zwei Zustandsänderungen im Speicher sein, damit eine Rücknahme ausgelöst werden kann. Zum einen der Zustand, der rückgängig gemacht werden soll und zum anderen der Zustand, in den gesprungen werden soll. Beginnend mit der Zeitleiste links in der Abbildung wird mit dem Aufbau der Zustände begonnen. In jedem Schritt kommt ein weiterer Zustand hinzu, bis der Zustand S4 erreicht ist und ein Undo ausgelöst wird.

Chronologisches Undo von Zuständen

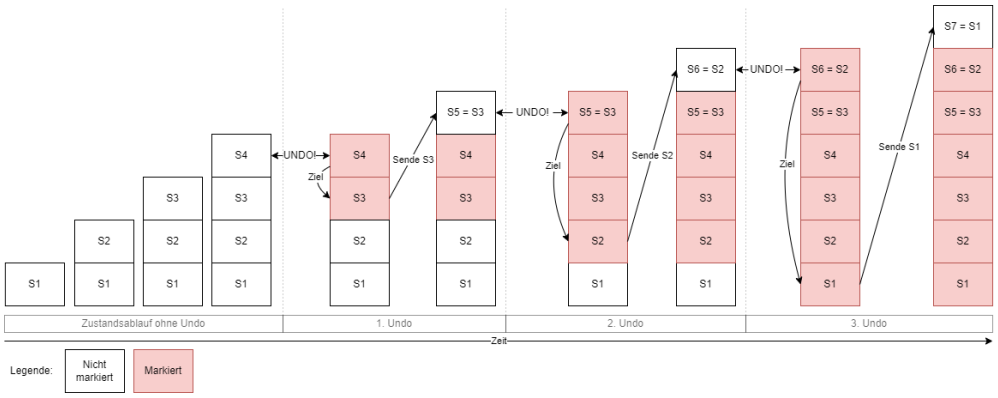


Abbildung 2.6: Ablauf eines chronologischen Undos

Beim chronologischen Undo von Zuständen erwartet der Benutzer, dass die letzten Zustände eines Aktors rückwärts durchlaufen werden, wie in 2.6 dargestellt. Diese Art des

Zurücknehmens wird heutzutage in einer Vielzahl von Anwendungen implementiert[20]. Im Beispiel soll zunächst der Zustand S4 im ersten Undo rückgängig gemacht werden.

Als vorheriger Zustand des Aktors ist S3 der neue erwartete Zustand S5 des Aktors und wird so als Befehl an den Aktor gesendet. Die Zustände S3 und S4 sind nach dem ersten Undo markiert. Der Zustand S4 wird markiert, weil dieser in diesem ersten Undo zurückgenommen wurde. Der Zustand S3 wird markiert, weil er gleich dem Zustand S5 ist und daher nicht als neuer erwarteter Zustand betrachtet werden kann. Diese Markierung ist in der Abbildung 2.6 durch eine rote Füllfarbe der Blöcke visualisiert.

Folgt nun ein zweites Undo, ohne dass es zu weiteren Zustandsänderungen des Aktors gekommen ist, so wird im Falle eines chronologischen Undos der Zustand S2 als zu erwartender Zustand angenommen. Die Zustände S3 und S4 sind markiert und werden beim zweiten Undo übersprungen. Nach dem zweiten Undo gilt dasselbe wie vorher für Zustand S3 und S4 für die Zustände S5 und S2. Beide Zustände sind rot markiert.

Analog zum ersten und zweiten Undo erfolgt ein drittes Undo. Die Zustände S3 bis S5 waren bereits Teil eines Undos und werden nicht berücksichtigt und S2 ist gleich S6 und kann daher nicht Ziel dieses Undos sein.

Der Ablauf eines chronologischen Undos führt am Ende dazu, dass die Ausgangszustände vor dem ersten Undo S1 bis S4 nacheinander rückwärts durchlaufen werden. Für ein chronologisches Undo ist es daher wichtig, Zustände, die bereits Ziel eines Undos waren, vom nächsten Undo ausschließen zu können.

Undo vom Undo

Erwartet der Benutzer ein Undo vom Undo, wechselt das Bild zu 2.7. In diesem Bild wird die gleiche Ausgangssituation dargestellt, wie sie auch im Chronologischen Undo zu sehen ist. Auch das erste Undo läuft genauso ab wie das erste Undo des Chronologischen Undos, indem der Zustand S4 zurückgenommen wird und S3 der neue Zustand dieses Aktors wird. Wenn das zweite Undo eintrifft, wird nun das vorherige Undo rückgängig gemacht. Also das Undo des Undos. Vor dem ersten Undo war der Zustand S4 der Zustand des Aktors. Daher ist nach erfolgreichem zweiten Undo der Zustand S6 gleich dem Zustand S4.

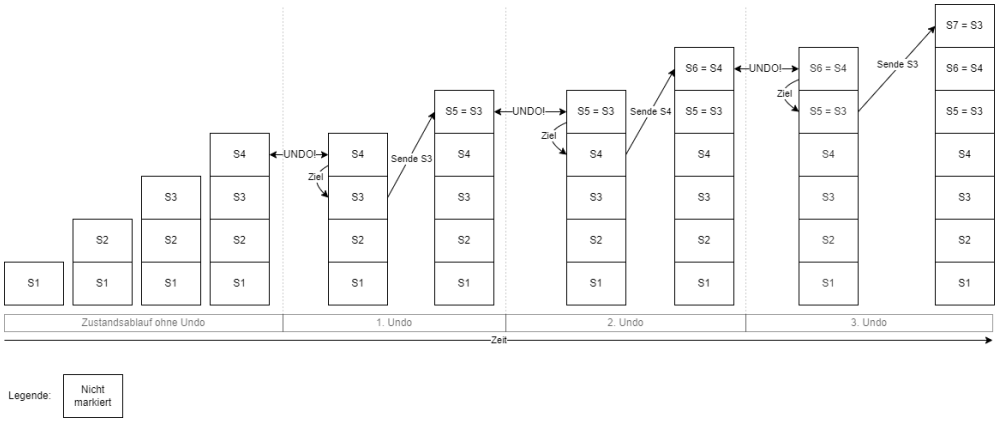


Abbildung 2.7: Ablauf eines Undos von Undo

Mit jedem weiteren Undo-Befehl ergibt sich eine Schleife, in der sich Zustand S3 und Zustand S4 abwechseln, anstatt wie beim chronologischen Undo die Zustände rückwärts zu durchlaufen. Markierungen auf bestimmte Zustände sind hier also nicht erforderlich.

2.4.2 Persistierungsmöglichkeiten

Um eine Undo-Funktion implementieren zu können, müssen die vergangenen Zustände der Aktoren ausgelesen werden können. Aus diesem Grund ist es wichtig, die Möglichkeiten der Persistierung in einem Smart Home zu betrachten. Im folgenden Kapitel werden vier verschiedene Möglichkeiten vorgestellt, wie eine Persistierung innerhalb des Undo Tools realisiert werden könnte. Wobei es letztlich auch denkbar wäre, anstatt einer eigenen Persistierung der Zustände für das Undo Tool aufzubauen, die Persistierung des Smart Home selbst zu nutzen, wenn diese vom Smart Home angeboten wird.

Aus jeder Persistierungsmöglichkeit müssen die Zustände der Aktoren und dessen zeitlicher Abfolge ausgelesen werden können. Nach dem Empfang werden die Daten mit allen relevanten Informationen, je nach Persistenzmöglichkeit, gespeichert.

Folgende Informationen sind für das Undo Tool wichtig:

- Index
- Aktor
- Zustand

Der Index muss eine eindeutige Zahl sein und von Zustandsänderung zu Zustandsänderung ansteigen. Dies ist wichtig, um verschiedene Zustandsänderungen, auch vom gleichen Aktor in den gleichen Zustand, voneinander unterscheiden und in eine zeitliche Abfolge bringen zu können. Der aufsteigende Index ist wichtig, um die Reihenfolge der Zustandsänderungen bestimmen zu können. Dieser Index kann auch der Zeitstempel des Auftretens der Zustandsänderung sein.

Um feststellen zu können, welcher Aktor den Zustand geändert hat, wird auch der Aktor gespeichert. Außerdem muss der neue Zustand des Aktors gespeichert werden, um eine Rückkehr in diesen Zustand zu ermöglichen.

Die folgenden Persistenzmöglichkeiten ermöglichen die Speicherung dieser Informationen und werden näher betrachtet.

Logfile

Die Erstellung eines Logfiles ist eine schnelle und einfache Möglichkeit, eine Persistenz zu implementieren. Für die Erstellung eines Logfiles ist kein zusätzliches Programm notwendig. Die Daten müssen in der Datei strukturiert abgelegt werden. In welcher Form diese Strukturierung umgesetzt wird, ist grundsätzlich nicht relevant. Entscheidend ist nur, dass die Daten schnell gespeichert und wieder ausgelesen werden können.

Datenbanken

Eine effizientere Art der Datenspeicherung als eine Logdatei ist die Verwendung eines Datenbanksystems. Zur Verwaltung der Datenbank wird jedoch ein Programm benötigt. Diese Datenbankmanagementsysteme sind speziell darauf ausgelegt, große Datenmengen zu speichern und schnell zugänglich zu machen. Für die Persistierung von Zuständen werden hier zwei verschiedene Datenbankmodelle betrachtet. Die relationalen Datenbanken (wie zum Beispiel durch NoSQL[24] realisiert) und die Zeitreihendatenbanken (wie zum Beispiel durch InfluxDB realisiert).

Eine relationale Datenbank speichert Daten in Tabellen mit Zeilen und Spalten. Die Beziehung zwischen verschiedenen Datensätzen wird durch Schlüssel hergestellt. Diese Art von Datenbank wird häufig verwendet, wenn komplexe Beziehungen zwischen Entitäten modelliert werden müssen. Sie bieten eine komplexe Abfragemöglichkeit über SQL, was

für das Auslesen der Zustände von Aktoren nützlich sein kann. Bei großen Datenmengen können sie jedoch an ihre Leistungsgrenzen stoßen.

Eine Zeitreihendatenbank speichert Datenpunkte, die über die Zeit gesammelt wurden. Jeder Datenpunkt ist mit einem Zeitstempel versehen und die Daten sind oft chronologisch sortiert. Sie sind darauf optimiert, große Mengen an zeitbezogenen Daten effizient zu speichern und abzurufen. Dadurch sind sie besonders geeignet, um Daten, bei denen der Zeitfaktor entscheidend ist, wie z.B. Sensordaten oder Zustände, effizient zu speichern. Beim Abruf der Daten liegt der Schwerpunkt auf der Aggregation über Zeitintervalle und trendbasierten Analysen.

Beide Lösungen lassen sich mit ähnlichem Aufwand in das Smart Home integrieren. Aufgrund der besseren Spezialisierung auf Daten mit Zeitfaktor, wie Sensordaten, wird hier im Zusammenhang mit dem Undo-Tool von den beiden Datenbankmodellen die Zeitreihendatenbank bevorzugt[35].

Aktoren mit eigener Persistierung

Anstelle eines zentralen Persistenzpunktes kann jeder Aktor selbst mit einer Persistenz ausgestattet werden. Damit speichert jeder Aktor seine eigenen Zustandsänderungen. Diese Zustandsänderungen können von außen abgefragt und analysiert werden. Die Zustandsänderungen können in einem flüchtigen Speicher, zum Beispiel RAM, gehalten werden. Sollen die Zustandsänderungen des Aktors jedoch auch bei Ausfällen des Aktors selbst erhalten bleiben, müsste jeder Aktor mit einem nichtflüchtigen Speicher ausgestattet werden. Dies würde für jeden Aktor im Smart Home eine zusätzliche Komponente bedeuten und auch die Programmierung der Aktoren komplexer machen. Um im Smart Home selbst zu funktionieren und die Undo-Funktion zu bieten, müssten die Aktoren also auch eine Undo-Funktion unterstützen.

Es ist jedoch interessant, diese Idee etwas weiter zu fassen, indem nicht nur die Persistenz selbst, sondern auch die komplette Auswertung des Undos auf die Aktoren übertragen wird. Jeder Aktor wäre ein Software-Agent in dem System und erhält die Nachricht, dass ein Undo durchgeführt werden muss und die Agenten finden selbst heraus, welcher Agent dieses Undo durchführen soll. Dies würde jedoch die Programmierung der Aktoren weiter verkomplizieren und zusätzliche Anforderungen an Aktoren stellen. Diese Art von Aktoren geht über den Rahmen dieser Arbeit hinaus, in der die grundsätzliche

Implementierung eines Undos näher betrachtet wird, und könnte in einer eigenen Arbeit genauer untersucht werden.

Event Streaming Platform

Eine weitere Möglichkeit, Zustandsdaten zu persistieren, bietet eine Event Streaming Platform[26]. Eine Event Streaming Plattform, wie zum Beispiel KafKa, konzentriert sich auf die kontinuierliche Verarbeitung von Ereignisdaten in Echtzeit. Die Daten werden persistent in einem Nachrichtenstrom gespeichert. Dies ermöglicht es, vergangene Ereignisse zu speichern und später darauf zuzugreifen. Sie sind für den Einsatz in verteilten Umgebungen konzipiert und ermöglichen die Skalierung über mehrere Server oder Cluster hinweg, um eine hohe Verfügbarkeit und die effiziente Verarbeitung großer Datenmengen zu gewährleisten.

2.5 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen zur Analyse weiter ausgearbeitet mit dem Ziel den Rahmen der Funktionen und Bedingungen zu konkretisieren.

2.5.1 Funktionale Anforderungen

Undo bedeutet, dass der letzte Befehl des letzten Aktors einer Gruppe zurückgenommen wird. Damit die Zustände bei einem Neustart des Systems nicht verloren gehen, ist eine persistente Speicherung der Zustände der Aktoren notwendig. Ohne diese Speicherung könnten die Aktoren nach einem Systemabsturz oder Neustart erst wieder zurückgesetzt werden, wenn mindestens zwei Zustandsänderungen vorliegen.

Eine Gruppe definiert sich entweder aus einer Gerätegruppe der Gruppen: DMX, Dali, Fenster, Rollo, Steckdose, Vorhang und Heizung oder aus einem der Räume des LP: Wohnzimmer, Flur, Küche, Esszimmer, Schlafzimmer oder Badezimmer. Der letzte Aktor beschreibt den Aktor der angegebenen Gruppe, der die letzte Zustandsänderung durchgeführt hat. Diese letzte Zustandsänderung ist also der letzte Befehl, der in der Gruppe

ausgeführt wurde und ist das Ziel des Undo. Das Rückgängigmachen ist als chronologisches Rückgängigmachen von Zuständen zu implementieren, wie im Kapitel2.4.1 beschrieben. Das bedeutet, dass der vorherige Zustand derjenige ist, welcher noch nicht in einem vorherigen Undo des letzten Aktors enthalten war.

Binäre Aktoren, die nur die Zustände ein oder aus annehmen können, müssen ebenso unterstützt werden wie mehrwertige Aktoren, die einen Zählwert haben können. Die zu unterstützenden binären Aktoren für das LP sind folgende:

- schaltbare Steckdosen
- Gardinen im Wohn- und Schlafzimmer

Sowie die mehrwertigen Aktoren des LP:

- Heizungsthermostate
- Fenstersteuerung
- Beschattung der Fenster
- DMX Licht
- Dali Licht

Ähnliche Aktoren, die in verschiedenen Räumen zu finden sind, wie die Beschattung für jeden einzelnen Raum, müssen vom Zielsystem getrennt voneinander behandelt werden. Außerdem soll dem Benutzer ein Feedback gegeben werden, je nachdem ob das Undo möglich und durchgeführt wurde oder unmöglich ist.

2.5.2 Nicht-Funktionale Anforderungen

Das Zielsystem muss so erweiterbar sein, dass weitere Teile und Aktoren hinzugefügt werden können, ohne dass der Aktor selbst angepasst oder das Zielsystem umprogrammiert werden muss. Die Zeit zwischen der Eingabe eines Befehls durch den Benutzer und der Rückmeldung muss gering sein. Diese Round Trip Latenz muss gemessen werden und unter 600 Millisekunden liegen, um eine erfolgreiche Mensch-Maschine-Interaktion darzustellen[27]. Bei der Implementierung der Persistenz muss der Speicherverbrauch der Daten begrenzt werden. Das System darf also auch nach längerem Betrieb nicht mehr Speicher verbrauchen als notwendig.

2.5.3 Use Case Ableitung

Name	Undo der Fenster in der Lounge
Akteur	Benutzer, Undo Tool, Smart Home, Luftqualitätssensor, Fenster Aktor
Trigger	Automatisierung öffnet Fenster der Lounge
Kurzbeschreibung	Eine Automatisierung im Smart Home öffnet die Fenster der Lounge. Der Benutzer will die Fenster geschlossen halten und führt den Undo Befehl aus.
Vorbedingung	Fenster ist geschlossen
Essenzielle Schritte	1. Luftqualitätssensor in der Lounge nimmt neuen Luftqualitätswert auf und propagiert ihn. 2. Eine Automatisierung wird durch den Sensorwert ausgelöst 3. Die Automatisierung öffnet die Fenster der Lounge 4. Benutzer will Fenster der Lounge geschlossen halten 5. Benutzer löst Undo der Gruppe Fenster aus 6. Undo Tool registriert ein Undo mit der Gruppe Fenster 7. Undo Tool identifiziert den zuletzt angesteuerten Aktor der Gruppe Fenster als den Fenster Aktor der Lounge 8. Undo Tool liest alle aufgezeichneten Zustände vom Fenster Aktor der Lounge aus 9. Undo Tool findet den vorherigen Zustand des Fenster Aktors der Lounge 10. Undo Tool sendet vorherigen Zustand als Befehl an den Fenster Aktor der Lounge 11. Undo Tool sendet eine Benachrichtigung an den Benutzer, dass das Undo erfolgreich durchgeführt wurde
Erweiterung	5a. Benutzer löst Undo der Gruppe Lounge aus 5a1. Undo Tool registriert ein Undo mit der Gruppe Lounge 5a2. Undo Tool identifiziert den zuletzt angesteuerten Aktor der Gruppe Lounge als den Fenster Aktor der Lounge 5a3. Undo Tool liest alle aufgezeichneten Zustände vom Fenster Aktor der Lounge aus 5a4. Undo Tool findet den vorherigen Zustand des Fenster Aktors der Lounge 5a5. Undo Tool sendet vorherigen Zustand als Befehl an den Fenster Aktor der Lounge 5a6. Undo Tool sendet eine Benachrichtigung an den Benutzer, dass das Undo erfolgreich durchgeführt wurde
Ausnahmefälle	9a. Undo Tool findet keinen validen vorherigen Zustand vom Fenster Aktor Lounge 9a1. Undo Tool sendet eine Benachrichtigung an den Benutzer, dass das Undo nicht durchgeführt werden kann
Zeitverhalten	Maximal 600ms von dem auslösen des Undos durch den Benutzer bis zur Verarbeitung des Undos und Rückmeldung an den Benutzer

Abbildung 2.8: Use Case Tabelle

Aus den Punkten der vorherigen beiden Kapiteln wird der Use Case “Undo der Fenster Lounge” aus Abbildung 2.8 abgeleitet. In dieser Ableitung wird ein Fenster Aktor in der Lounge geschaltet und daraufhin das Undo durch den Benutzer ausgelöst. Zusätzlich wird die Persistierung von Zuständen in dem Use Case “Persistierung von Zuständen” der Tabelle 2.9 dargestellt.

Name	Persistierung von Zuständen
Akteur	Undo Tool, Fenster Aktor
Trigger	Fenster Aktor ändert seinen Zustand
Kurzbeschreibung	Der Fenster Aktor wird geschaltet und ändert seinen Zustand. Das Undo Tool empfängt diese Änderung und speichert sie ab.
Essenzielle Schritte	1. Fenster Aktor erhält Steuerbefehl 2. Fenster Aktor sendet Zustandsänderung 3. Undo Tool empfängt Zustandsänderung 4. Undo Tool überprüft, ob der neue Zustand gleich dem zuletzt aufgezeichneten Zustand ist 5. Zustand ist ungleich dem vorherigen Zustand des Fenster Aktor 6. Undo Tool speichert die Zustandsänderung ab
Erweiterung	5a. Zustand ist gleich dem vorherigen Zustand des Fenster Aktor 5a1. Zustandsänderung wird nicht abgespeichert
Ausnahmefälle	6a. Zustand kann nicht abgespeichert werden 6a1. Fehlermeldung an den Benutzer weiterleiten

Abbildung 2.9: Use Case Tabelle

Zur besseren Orientierung im Kapitel ist jede Anforderung noch einmal in der Tabelle 2.10 zusammengefasst.

2.6 Fazit Konzept

Smart-Home-Umgebungen bieten dem Nutzer viele Möglichkeiten, mit den Komponenten seines Hauses bequemer zu interagieren. Smart Homes helfen auch, durch Automatisierung Geld zu sparen, indem Geräte wie z.B. Heizungen automatisch gesteuert werden. Solange es jedoch keine Standardisierung von Protokollen und Schnittstellen gibt, bleibt dem Nutzer meist nur die Möglichkeit, zwischen verschiedenen Herstellern und ihren Geräten zu wählen.

Dieser Fakt wurde auch bei der Entwicklung des LP berücksichtigt und hat vieles verkompliziert. Die Integration aller Komponenten an einer Stelle hat die Notwendigkeit einer Vielzahl von Adaptern zur Folge, die auch programmiert werden müssen. Für den Aufbau eines personalisierten Smart Home sind daher oft komplexe Lösungen und Programmierkenntnisse erforderlich.

Bei der Entwicklung von Smart Homes werden in Zukunft herstellerunabhängige Standards wie der Matter Standard eine entscheidende Rolle spielen.

ReqNr	Beschreibung
RQ-1	Benutzer kann Undo Befehl auslösen und führt einen gesuchten Aktor in einen vorherigen Zustand zurück
RQ-2	Benutzer kann eine Gruppe zum Einschränken des Undos angeben
RQ-3	Gruppen sind entweder Gerätegruppen oder Räume
RQ-4	Valide Gerätegruppen sind: DMX, Dali, Fenster, Rollos, Steckdose, Gardine und Heizung
RQ-5	Valide Räume sind: Lounge, Esszimmer, Küche und Schlafzimmer
RQ-6	Erfolgreiche Undos werden dem Benutzer mitgeteilt
RQ-7	Gescheiterte Undos werden dem Benutzer mitgeteilt
RQ-8	Die Latenz zwischen ausgelösten Undo und Rückmeldung darf maximal 600ms betragen
RQ-9	Zustandsänderungen von validen Gerätegruppen werden abgespeichert
RQ-10	Die Anzahl der zu speichernden Zuständen je Aktor ist auf 20 begrenzt
RQ-11	Neue Aktoren und Räume können ohne Anpassung der Implementierung integriert werden
RQ-12	Das Undo setzt ein chronologisches Undo um

Abbildung 2.10: Anforderungstabelle

3 Design und Implementation

In diesem Kapitel wird das Design der entwickelten Lösung vorgestellt und beschrieben. Zunächst wird das Konzept der Lösung vorgestellt. Dabei werden die benötigten Schnittstellen aufgelistet und die drei Ebenen des Designs anhand der Three-Tier-Architektur[19] dargestellt. Im Anschluss wird die MQTT-Schnittstelle vorgestellt und erläutert, wie ein Undo abläuft. Danach wird die Auswahl der Persistierungsmöglichkeiten aus Kapitel 2.4.2 für diese Implementierung ausgewählt und begründet. Abschließend wird die Umsetzung beschrieben.

3.1 Konzept

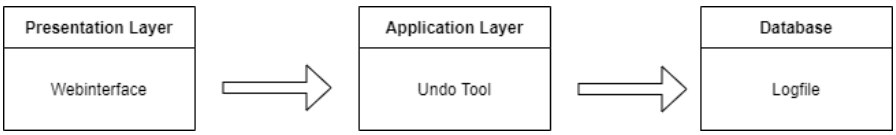


Abbildung 3.1: Three Tier Architektur der Lösung

Das Programm gliedert sich in die drei Bereiche Presentation Layer, Application Layer und Database. In der Abbildung 3.1 sind die einzelnen Schichten mit ihren Komponenten dargestellt. Das Webinterface stellt den Presentation Layer dar. Im Application Layer befindet sich das Undo Tool selbst. Im Database Layer befindet sich das Logfile dieser Implementierung.

Das Undo Tool fügt sich in den Middleware und Human Interaction Layer der LP Architektur 2.3 ein und erweitert diese zur Abbildung A.1. Alle Komponenten des Undo Tools sind auf der rechten Seite der Abbildung sichtbar abgegrenzt. Die Erweiterung des Dashboards befindet sich hier im Human Interaction Layer und kommuniziert mit dem Undo Tool über den MQTT Broker der Infrastruktur. Das Undo Tool selbst ist zusammen mit der Datenhaltung in der Middleware der LP-Architektur untergebracht. Als Übersicht über die Komponenten der realisierten Lösung dient die Abbildung 3.2.

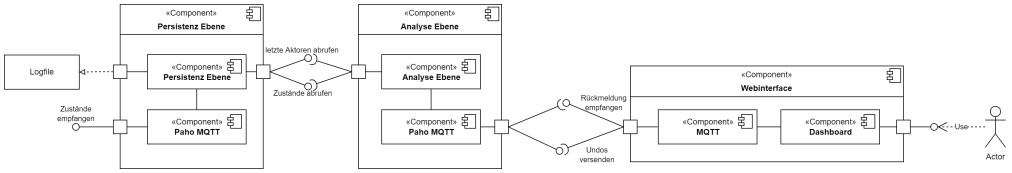


Abbildung 3.2: Komponentendarstellung der Lösung

3.1.1 Schnittstellen

Für die Eingabe der Undo-Befehle durch den Benutzer ist ein Human Maschine Interface (HMI) erforderlich, das die eingehenden Befehle empfängt und weiterleitet. Um die vom HMI empfangenen Befehle weiterzuleiten, wird eine Schnittstelle zum MQTT Broker der Infrastruktur benötigt.

Darüber hinaus benötigt die Analyseschicht des Undo-Tools zum Empfang der HMI-Befehle und der Aktorzustände selbst eine Schnittstelle zum MQTT-Broker der Infrastruktur. Auch die Rückmeldung an die HMI, ob das Undo erfolgreich war, erfolgt über diese MQTT-Schnittstelle. Zusätzlich zu dieser Schnittstelle ist eine Schnittstelle zwischen dem Undo-Tool und der Persistenzschicht erforderlich, um die Zustände der Aktoren weiterzuleiten.

Die Persistenzschicht wiederum benötigt eine Schnittstelle zum MQTT Broker, um Zustände zu empfangen und zu einem Datenspeicher, um die Zustände aufzeichnen zu können. Der Nachrichtenfluss der Schnittstellen ist in der Abbildung 3.3 dargestellt.

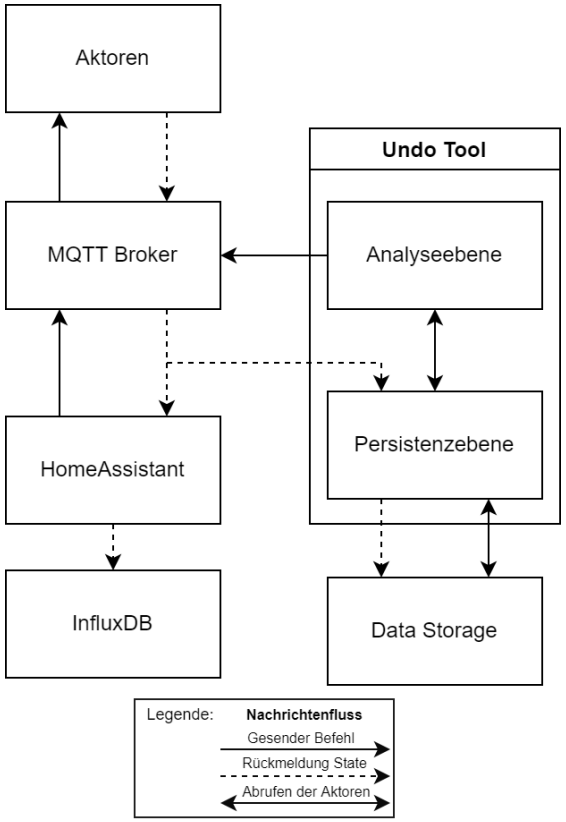


Abbildung 3.3: Darstellung des Nachrichtenflusses

3.1.2 MQTT Topics

Für die Kommunikation des Tools werden MQTT Topics benötigt, die in diesem Kapitel genauer beschrieben werden.

MQTT Topics Webinterface zu Undo Tool

Die Kommunikation von Webinterface zu Undo Tool ist über zwei Topics geregelt.

- LP/Undo/Command
- LP/Undo/Status

Über das Topic *LP/Undo/Command* sendet das Webinterface einen Undo-Befehl an das Undo-Tool. Die Gruppe, auf die der Befehl beschränkt werden soll, dient als Payload der Nachricht. Ohne Angabe einer Payload wird das Undo Tool den Befehl auf alle vorhandenen Räume und Gerätegruppen anwenden. Nach RQ-4 und RQ-5 (aus der Tabelle 2.10) sind die möglichen Payloads für das *LP/Undo/Command*-Topic:

```
Payloads:
Lounge, Esszimmer, Küche, Schlafzimmer,
DMX, Dali, Fenster, Rollos, Steckdose, Gardine, Heizung
```

Mit dem Topic *LP/Undo/Status* wird eine Rückmeldung über die Ausführung des Befehls zurückgegeben. Mögliche Payloads sind *success* und *failed*. Success beschreibt die erfolgreiche Ausführung eines Undo-Befehls und failed beschreibt einen fehlgeschlagenen Undo-Befehl.

MQTT Topics Aktoren zu Persistierung

Andererseits muss die Persistenz in der Lage sein, alle Zustände der Aktoren des LP zu Empfangen. Daher muss die Persistenz auf alle Topics der Aktoren abonnieren, also subscriben.

Das Topic für die Zustände sind im LP nach folgendem Muster aufgebaut:

```
LP/<Raum>/<Aktor>/Status/<Gerätenummer>
```

Daneben sind die Topics für Befehle nach diesem Muster aufgebaut:

```
LP/<Raum>/<Aktor>/Set/<Gerätenummer>
```

Mit Hilfe der Gerätenummer können mehrere Geräte innerhalb eines Raumes unterschieden werden.

Dieses Muster kann durch die Möglichkeit von MQTT, Wildcards in Topics anzugeben, genutzt werden um leicht auf alle benötigten Topics zu subscriben. MQTT bietet die Verwendung von Single Level Wildcards (+) oder Multi Level Wildcards (#) an. Single

Level Wildcards dienen als Platzhalter für eine Ebene im Topic und Multi Level Wildcards, die nur am Ende stehen dürfen, lassen alle Topics zu, die dem Muster vor dem # entsprechen.

Durch die Verwendung von Wildcards kann auch die Skalierbarkeit für die RQ-11 der Anforderungstabelle 2.10 erreicht werden. Durch das Setzen des Raumes und des Aktors des Topics als Single Level Wildcard und der Gerätenummer als Multi Level Wildcard werden die Zustände aller Aktoren in allen Räumen gleichzeitig abonniert, auch wenn sich diese in Zukunft ändern sollten.

Um alle Zustände der Aktoren zu erhalten, muss die Persistenz das folgende Topic abonnieren:

```
LP/+/+/Status/#
```

3.1.3 Presentation Layer – Webinterface

Die HMI wird als Webinterface implementiert. Ein Webinterface ist eine einfach zu implementierende Möglichkeit für den Benutzer, mit dem System zu interagieren, die in das Ökosystem des LP integriert werden kann. In diesem Webinterface können alle gültigen Räume und Gruppen (vgl. RQ-4, RQ-5 2.10) vom Benutzer ausgewählt und das Undo ausgelöst werden. Darüber hinaus wird auf dem Webinterface das Feedback des Undo-Tools angezeigt.

Hier kann auch der Start des Undos aufgezeichnet und mit dem Zeitpunkt der Rückmeldung verglichen werden, um die Latenz zu messen. Die Komponenten des Webinterfaces sind in der Abbildung 3.2 sichtbar. Dort teilt sich das Webinterface in die zwei Komponenten MQTT und Dashboard. Das Dashboard stellt die Benutzeroberfläche für den Benutzer dar und bietet verschiedene Schaltflächen für die einzelnen Räume und Gerätegruppen. Die MQTT-Komponente verbindet das Dashboard mit dem Undo-Tool. Diese MQTT-Komponente kann über die entsprechenden Topics Rückmeldungen empfangen und Undo-Befehle weiterleiten.

3.1.4 Application Layer – Undo Tool

Das Undo-Tool selbst muss bei dieser Implementierung zwei Anforderungen erfüllen. Es muss die Undo-Befehle des Benutzers umsetzen und dazu die vergangenen Zustände der

Aktoren auslesen können. Dadurch wird das Undo-Tool in zwei verschiedene Komponenten aufgeteilt, die Analyseebene und die Persistenzebene. Diese beiden Komponenten sind in der Abbildung 3.2 sichtbar.

Die Analyse Ebene vergleicht alle Aktoren und Zustände, um zu entscheiden, welcher Aktor das Ziel des Undos ist und in welchen Zustand zu springen ist.

Daneben gibt es die Persistenzebene. Diese nimmt alle Zustände der Aktoren vom MQTT Broker entgegen und speichert sie in einem Datenspeicher.

Die Analyse der Analyse Ebene unterteilt sich in das Finden des richtigen Aktors und das Finden des richtigen Zustandes. Die Persistenzschicht muss der Analyseschicht zwei verschiedene Schnittstellen zur Verfügung stellen. Wie die Datenhaltung und damit die Persistenzebene implementiert wird, ist für die Analyseebene selbst nicht relevant.

Über die Schnittstelle “letzte Aktoren abrufen” kann die Analyse Ebene alle Aktoren mit dem letzten Zustand abrufen. Während über das “Zustände abrufen”-Interface alle Zustände eines einzelnen gespeicherten Aktors abgefragt werden können. Die Aufteilung in zwei verschiedene Interfaces hat den Vorteil, dass die Analyseebene zunächst feststellen kann, welcher Aktor gesucht wird, um dann alle Zustände von diesem Aktor abrufen kann.

Im ersten Teil der Analyse vergleicht die Analyseebene alle Zustände der Aktoren anhand des Indexes miteinander. Durch den Vergleich des letzten Zustands aller Aktoren kann festgestellt werden, welcher Aktor als letzter geschaltet hat. Da weitere Zustände pro Aktor für diese Auswertung nicht notwendig sind, können diese bei der Abfrage weggelassen werden.

Im zweiten Teil der Analyse werden alle Zustände des zuletzt geschalteten Aktors aus der Persistenzebene abgefragt. Die Analyseebene muss ein chronologisches Undo (RQ-12 2.10) implementieren. Daher muss die Analyseebene Zustände filtern, die bereits Teil eines Undos waren. Dieser Filter wird auf die abgerufenen Zustände des Aktors angewendet, um den richtigen Zustand zu finden. Wird nach der Filterung kein Zustand gefunden, kann kein Undo durchgeführt werden und es muss eine negative Rückmeldung zurückgegeben werden (RQ-7 2.10).

Nach erfolgreichem Abschluss der Analyse sendet die Analyseebene den neuen Zustand als Befehl an den gesuchten Aktor (RQ-1 2.10). Anschließend wird eine Rückmeldung über den erfolgreichen Abschluss des Undos an das Dashboard gesendet (RQ-6 2.10).

3.1.5 Database – Datenpersistierung

Nachfolgend wird das Design der Datenpersistierung beschrieben. Dabei teilt sich das Design in die Persistierung und in die Optimierung der Persistierten Daten auf.

Persistierung

Für die Implementierung der Datenhaltung der Persistenzebene stehen die Möglichkeiten von 2.4.2 zur Verfügung. In diesem Fall wurde die Logfile-Option gewählt. Bei dieser Möglichkeit werden keine zusätzlichen Anforderungen an die Infrastruktur des LP gestellt. Außerdem soll hier ein Undo untersucht werden, ohne auf die Unterschiede zu den komplexeren Persistierungsmöglichkeiten eines Undos einzugehen. Diese Persistenzmöglichkeiten hätten ihre eigenen Optimierungen, die neben dem Undo-Tool separat untersucht werden müssten.

Bei der Implementierung mithilfe eines Logfiles erhält die Persistenzschicht die Zustände der Aktoren direkt vom Broker, indem die Topics der Aktoren abonniert werden. Anschließend werden die Broker-Nachrichten serialisiert, indiziert und in einem Filesystem gespeichert.

Die für die Analyse benötigten Daten sind ein Index, das Topic und die Payload. Der Index ist eine Zählvariable der Zeilennummer, um als Schlüssel zu dienen und somit jeden Eintrag eindeutig zu machen. Dies ist wichtig, um im weiteren Verlauf vergangene Zustände als bereits bearbeitet markieren zu können und somit ein chronologisches Undo zu ermöglichen. Mit dem Topic kann der Aktor, der Raum und auch die Gerätegruppe des Aktors gespeichert werden, daher wird das Topic als Ganzes gespeichert, anstatt nur den Aktor zu identifizieren und diesen zu speichern. Nach der Abfrage durch die Analyseebene ist eine Deserialisierung der Daten und die Rückgabe der Aktoren mit ihren Zuständen erforderlich.

Die Persistenzschicht ist auch für die Optimierung der Logdatei zur Vermeidung eines Überlaufens des Dateisystems nach längerer Betriebszeit und zur Aufrechterhaltung einer stabilen Performance verantwortlich. Dies wird erreicht, indem nur eine bestimmte Anzahl von Zuständen pro Aktor gespeichert wird. Wie im Kapitel 2.4.1 beschrieben, werden mindestens zwei Zustände für ein Undo benötigt, was die Untergrenze der Persistenz darstellt. Als Obergrenze für den maximalen Speicherverbrauch der Persistenz

wurde für dieses Experiment ein Wert von 20 festgelegt. Wenn die Obergrenze erreicht ist, werden ältere Zustände des Aktors aus der Logdatei gelöscht.

Nach einer Analyse aller Befehle innerhalb der Laborumgebung wurde als Trennzeichen in der Zeile des Logfiles ein Semikolon gewählt, da dieses in keiner Payload eines Aktors vorkommt. Um auch Fälle abzufangen, in denen ein Semikolon in der Payload verwendet wird, kann ein anderes Trennzeichen gewählt oder die Datei in einem anderen Dateiformat wie z.B. JSON erstellt werden.

Eine Zeile im Logfile besteht aus dem Index der Nachricht (Zeilennummer), dem Topic, der Payload und dem Trennzeichen dazwischen.

Durch das Hinzufügen weiterer Zeilen im Logfile für jede weitere Zustandsänderung wird sichergestellt, dass die Nachrichten nach dem Index aufsteigend sortiert werden.

Optimierung der Speichernutzung in der Persistenz Ebene

Im Dauerbetrieb des Tools würde die Größe des Logfiles mit der Anzahl der Zustandsänderungen weiter ansteigen. Ohne eine Optimierung der Speichernutzung würde das Dateisystem früher oder später überlaufen. Außerdem steigt die Round Trip Latenz der Undo Durchführung mit steigender Anzahl an Zustandsänderungen im Log an. Zur Vermeidung dieser Situation ist eine Optimierung der Daten im Logfile notwendig.

Eine Möglichkeit ist das Löschen aller Daten, deren Index kleiner als eine gewünschte Zahl ist. Dabei wird das Logfile als Ganzes betrachtet und nur die letzten Zeilen bleiben im Logfile erhalten. Dies hat jedoch zur Folge, dass unter Umständen der letzte Zustand eines Aktors gelöscht wird und somit für ein Undo nicht mehr zur Verfügung steht. Einen solchen Fall zeigt die Abbildung 3.4.

Die Abbildung ähnelt dem Aufbau des Logfiles, indem die Zustände Zeile für Zeile, Zustand für Zustand nach unten weiter aufgebaut werden. Die Zustände von Aktor 1 und Aktor 2 sind gespeichert und werden mit dem Präfix “S” und einer aufsteigenden Anzahl von Zuständen des Aktors dargestellt. Aktor 2 hat seit dem Start des Programms zwei Zustandsänderung erfahren, während Aktor 1 bereits sechs verschiedene Zustandsänderungen durchlaufen hat. Wird beispielsweise alles vor Index 5 gelöscht, verliert Aktor 2 seinen Zustand S1 und ein Undo wäre, mit dem einen übrigen Zustand, nicht mehr möglich.

Daher muss die Optimierung der Persistenzebene für jeden einzelnen Aktor über alle Zustände dieses Aktors durchgeführt werden. Überschüssige Zustände werden dann pro Aktor gelöscht. Bei einer Optimierung von Abbildung 3.4 auf diese Weise und einem Grenzwert von zwei wäre für Aktor 1 dann noch *S6* und *S5* übrig. Für Aktor 2 demnach noch *S1* und *S2*.

Die Komplexität dieser Optimierung hängt von der Anzahl der Aktoren ab. Je mehr Aktoren im System vorhanden sind, desto länger dauert diese Optimierung. Auch die Anzahl der Zustände pro Aktor verlängert die Optimierung. Daher wird empfohlen, das Logfile regelmäßig optimieren zu lassen. Der Zeitpunkt für die Optimierung kann so gewählt werden, dass voraussichtlich kein Undo ausgelöst wird. Dadurch werden Wechselwirkungen vermieden.

Es könnte auch ein Zeitstempel als Index für jeden Zustand gespeichert werden. Der Benutzer könnte dann angeben, wie lange die Zustände gespeichert werden sollen. Beispielsweise wäre es denkbar, alle Zustände zu löschen, die älter als eine Woche sind. Es könnte argumentiert werden, dass es nicht sinnvoll ist, einen Zustand rückgängig zu machen, der älter als eine Woche ist. Da aber letztlich nicht gänzlich ausgeschlossen werden kann, dass es für selten angesteuerte Aktoren durchaus sinnvoll sein kann, auch den Zustand von vor einer Woche anzunehmen, wurde hier die Implementierung mit Hilfe von Indizes und einer maximalen Anzahl von Zuständen pro Aktor umgesetzt.

3.1.6 Ablauf eines Undos

Das Sequenzdiagramm A.2 zeigt den Ablauf des Szenarios aus 2.4. Da sich der Ablauf nicht zwischen den einzelnen Aktoren unterscheidet, wurden die Fensteraktoren allgemeiner als Aktoren gehalten.

Ein neuer Zustand (newState) eines Sensors wird über den MQTT Broker an HomeAssistant und die Persistenzschicht propagiert.

HomeAssistant selbst speichert diesen Zustand in einer InfluxDB, während die Persistenzschicht ihn intern in ein Logfile schreibt. Diese Zustandsänderung löst im HomeAssistant eine Automatisierung aus, die eine Zustandsänderung des Aktors vorsieht Über den MQTT Broker wird dann der neue Zustand (wantedState) als Befehl von HomeAssistant an den Aktor propagiert. Nach Erhalt des Befehls (wantedState) wechselt der Aktor in den neuen Zustand (newState = wantedState) und propagiert die Zustandsänderung über

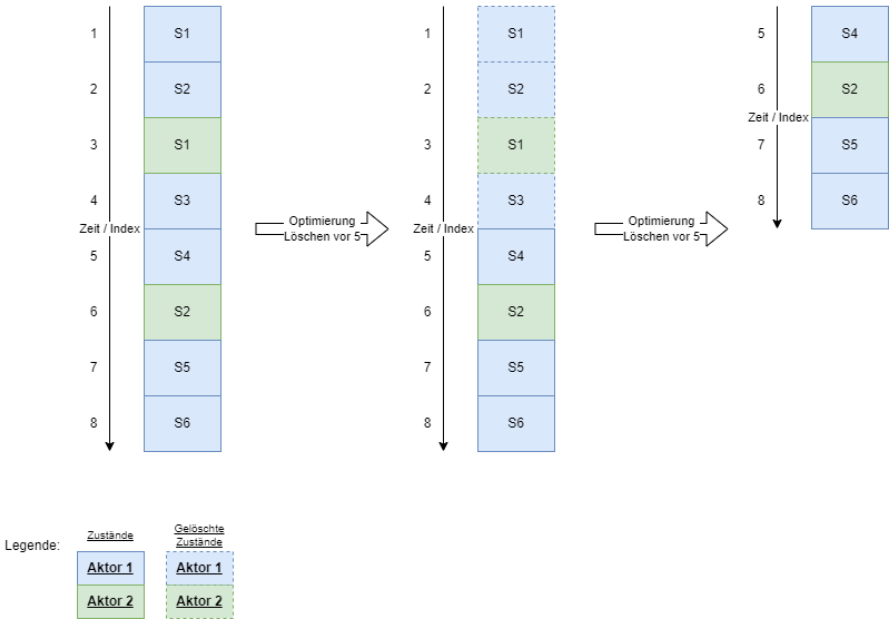


Abbildung 3.4: Optimierung - Löschen des letzten Zustandes von Aktor 2

den MQTT Broker. HomeAssistant und die Persistenzschicht erhalten diesen geänderten Zustand (newState) und persistieren den Zustand.

Der Benutzer löst nun über das Webinterface ein Undo in einem Raum aus. Der soeben geänderte Aktor befindet sich in diesem Raum und hat somit auch den Raumnamen als Topic. Über den Raumnamen aus der Payload des Befehls vom Benutzer fragt die Analyseebene von der Persistenzebene die zuletzt geschalteten Aktoren des Raumes ab. Der letzte Eintrag in der Liste ist der zuletzt geänderte Aktor und damit der gesuchte Aktor des Undo-Befehls. Mit dem gesuchten Aktor fragt die Analyseebene nun alle gespeicherten Zustände des Aktors von der Persistenzebene ab.

Betrachtet man noch einmal die Abbildung des chronologischen Undos 2.6, so wird die Abfolge zum Auffinden des gesuchten Zustands deutlicher. Die Reihenfolge der Schritte zum Erreichen des vorherigen Zustandes eines Aktors ist unabhängig von der Anzahl der bereits durchgeführten Undos. Daher wird im Folgenden der Punkt vor dem ersten Undo in der Abbildung 2.6 genauer betrachtet. Zur Ermittlung des vorherigen Zustandes des Aktors wird folgender Algorithmus verwendet:

- 1. Abrufen aller Zustände des Aktors (S1-S4)
- 2. Auslesen des zuletzt gespeicherten Zustands (S4) des Aktors

- 3. Markieren des Zustands S4
- 4. Filtern aller Zustände der Aktoren mit den bereits markierten Zuständen
- 5. Der neuste Zustand der gefilterten Liste (S3) ist vorheriger Zustand
- 6. vorheriger Zustand (S3) markieren

Die Betrachtung des Sequenzdiagramms A.2 wird nun fortgesetzt. Der vorherige Zustand (lastState) wird als Befehl an den Aktor weitergegeben. Der Aktor empfängt den Befehl und wechselt in den Zustand (newState = lastState) und propagiert diese Änderung über den MQTT Broker.

HomeAssistant und auch die Persistenzschicht erhalten diese Änderung und persistieren den Zustand.

3.2 Umsetzung

Im folgenden Kapitel wird die Implementierung beschrieben. Dabei wurde das Webinterface mit Hilfe von NodeRed und das Undo-Tool selbst in zwei Python-Klassen umgesetzt. Diese sind in eine Analyse- und eine Persistenzklasse unterteilt.

3.2.1 Implementierung der Benutzeroberfläche

Das Webinterface der implementierten Lösung wurde mit NodeRed unter NodeJs realisiert.

Für das Webinterface wird die Dashboard Palette[11] verwendet, um ein einfaches Human Machine Interface (HMI) zu realisieren. Durch die Verwendung der Dashboard Palette muss kein eigener Webserver auf dem System eingerichtet werden und es sind keine HTML-Kenntnisse erforderlich.

Eine MQTT Node stellt eine Verbindung zum MQTT Broker der LP Infrastruktur her. Über diesen MQTT Node werden die Befehle des Webinterfaces an den MQTT Broker und damit an das Undo Tool weitergeleitet. Jeder Raum und jede Gerätegruppe hat eine eigene Dashboard Button Node, der einen Button auf dem Webinterface darstellt. Jeder Button ist so konfiguriert, dass die Payload abhängig vom Label des Buttons

gesendet wird. Diese Buttons sind unter anderem mit dem MQTT Node verbunden, der die Payload der Buttons an das Undo Command Topic weiterleitet.

Außerdem ist der Button mit einer Funktionsnode verbunden, der die Startzeit des Buttons als Kontextvariable in NodeRed speichert. Zusätzlich wird die Payload der Nachricht in "Undo wurde gesendet..." geändert und an den Dashboard Text Node mit dem Label *Rückmeldung* weitergeleitet. Dieses Label zeigt dem Benutzer an, dass der Befehl empfangen wurde und das Undo ausgeführt wird.

Als Input für das Dashboard dient eine MQTT Node, die das Topic *LP/Undo/Status* abonniert hat. Über diese Node erhält das Dashboard die Antwort des Undo Tools. Sobald die Antwort eingegangen ist, wird sie an die Text Node "Rückmeldung" weitergeleitet. Die Eingabe MQTT Node ist wiederum mit einer Funktionsnode verbunden, die die aktuelle Zeit als Endzeit speichert.

Anschließend wird die Endzeit von der Startzeit subtrahiert, um die Round Trip Latency des Befehls zu erhalten. Diese Round Trip Latency wird an eine Dashboard Diagramm Node weitergeleitet und angezeigt. So kann die Round Trip Latency der letzten 5 Minuten abgelesen werden. Dieses Diagramm dient nur zur Auswertung dieser Round Trip Latenz und kann bei einer eventuellen Integration in das LP weggelassen werden.

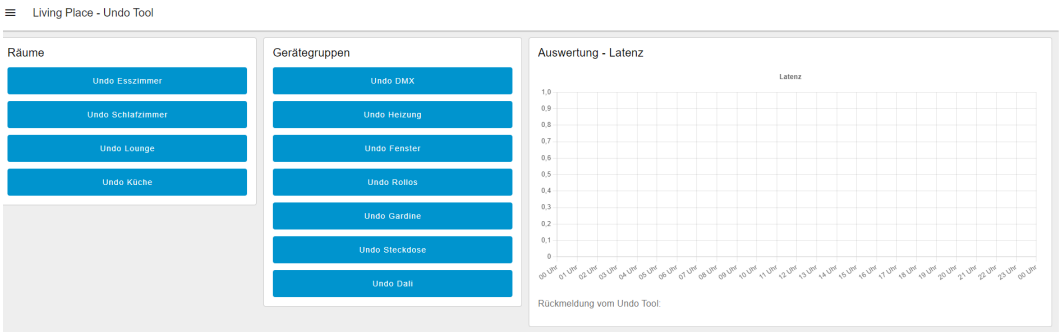


Abbildung 3.5: Dashboard des Undo Tools

In der Abbildung 3.5 ist das umgesetzte Dashboard sichtbar. Die Gruppen sind nach Raum- und Gerätegruppe in zwei Spalten aufgeteilt. Daneben ist das Diagramm für die Auswertung.

3.2.2 Implementierung des Undo Tools

In den folgenden Kapiteln wird die Implementierung des Undo Tools beschrieben. Wie in der Abbildung 3.2 sichtbar, wurde die Implementierung in zwei Klassen aufgeteilt. Die Implementierung selbst wurde mithilfe von der Python Programmiersprache umgesetzt. Um die Verbindung der Klassen mit dem MQTT Broker der Infrastruktur herzustellen, wurde die Paho Bibliothek[14] benutzt. Das Objekt der Persistenz Ebene wird in der Analyse Ebene erstellt und kann so direkt auf dessen Schnittstellen zugreifen.

Komponente Analyse Ebene

Zuerst muss die Analyseschicht eine Verbindung zum MQTT Broker der Infrastruktur aufbauen. Um die Verbindung zu halten, erstellt die Paho Library einen eigenen Thread. Dieser Thread stellt die Verbindung zum MQTT Broker her, hält die Verbindung, abonniert die Topics und leitet ankommende Nachrichten an die Analyse Ebene weiter.

Da die Analyseebene nur ein Topic abonniert hat, das *LP/Undo/Command* Topic, werden Nachrichten nur von diesem Topic empfangen. Bei der Klasseninitialisierung erhält die Analyseklasse über einen Parameter eine Referenz auf ein Objekt der Persistenzklasse.

Für die Markierung der Zustände aus den Schritten 3 und 4 des Algorithmus 3.1.6 wird eine Python-Liste benötigt, die bei der Initialisierung der Klasse erzeugt wird. In dieser Liste werden nur die Indizes der Zustandsänderungen aus dem Logfile gespeichert. Python-Listen-Objekte haben den Vorteil, dass über einen Index direkt auf den Inhalt zugegriffen werden kann und nicht über die Liste bis zum gesuchten Objekt iteriert werden muss. Außerdem behält eine Python-Liste die Sortierung der eingetragenen Objekte bei, was für die spätere Analyse wichtig ist.

Erhält das Topic einen Undo-Befehl, wird die Analyse der Aktoren mit der Methode "undoIt" der Analyseebene gestartet. Vom Persistenzklassenobjekt werden nun alle Aktoren abgerufen. Die Liste der ausgewählten Zustände wird als Parameter übergeben und von der Persistenzklasse zur Filterung verwendet.

Die Persistenzklasse gibt die aufgezeichneten Aktoren als Python-Liste zurück. Ein einzelner Eintrag in dieser Liste entspricht einer String-Repräsentation der Zeile des Logfiles. Die Liste wird aufsteigend nach der Indexnummer sortiert übergeben. Ein höherer Index

bedeutet auch eine spätere Änderung des Aktors. Der letzte Eintrag der Liste ist somit der gesuchte Aktor.

Wenn kein Aktor in der Liste vorhanden ist, kann kein Undo durchgeführt werden. Es wird eine Fehlermeldung "failed" über das MQTT Topic *LP/Undo/Status* gesendet und die Methode "undoIt" verlassen.

Die Persistenzklasse wird erneut abgefragt, nachdem der gesuchte Aktor identifiziert wurde. Alle Zustände des Aktors werden abgefragt, wobei die Liste der selektierten Zustände wieder als Filterparameter dient. Als Rückgabewert erhält die Analyse Klasse von der Persistenz Klasse eine Python Liste "history". Jedes Objekt dieser Liste ist wiederum eine String-Repräsentation einer Zeile aus dem Logfile. Auch diese Liste ist nach dem Index aufsteigend sortiert.

Ist mehr als ein Eintrag in dieser "history"Liste vorhanden, so ist ein Undo des Aktors möglich und die Methode wird fortgesetzt. Andernfalls wird eine Fehlermeldung über das MQTT Topic *LP/Undo/Status* zurückgegeben.

Für die Analyse ist der aktuelle Zustand als Markierung und der vorherige Zustand als neuer Zustand wichtig. Die Indizes dieser Einträge in der sortierten "history"-Liste sind daher:

```
Aktueller Zustand: Länge(history) - 1
vorheriger Zustand: Länge(history) - 2
```

Die Indizes beider Einträge werden an die Liste "pastundos" angehängt. Das Topic des vorherigen Status wird ausgelesen und in einer String-Variablen gespeichert. Da sich ein Command Topic von einem Status Topic nur durch den Status Teil bei einem Status und den Set Teil bei einem Command unterscheidet, kann der String leicht angepasst werden.

Mit der Methode String replace wird der Teilstring Status im Topic durch Set ersetzt. Dadurch wird aus dem Status Topic

```
LP/<Raum>/<Aktor>/Status/<Gerätenummer>
```

das Kommando Topic

`LP/<Raum>/<Aktor>/Set/<Gerätenummer>`

Der MQTT-Befehl für den vergangenen Zustand ist also wie folgt aufgebaut:

- Topic: Topic vergangener Zustand mit Status ersetzt durch Set
- Payload: Payload vergangener Zustand

Nach dem Senden des Befehls mithilfe des MQTT Client Objekts wird der Benutzer über das *LP/Undo/Status* Topic über das erfolgreiche Durchführen des Undos mit einem success in der Payload informiert.

Komponente Persistenz Ebene

Die Persistenzklasse besitzt wie die Analyseklasse ein Paho MQTT Client Objekt, das die Verbindung zum MQTT Broker der LP Infrastruktur hält und die Nachrichten der Aktoren empfängt. Wenn die Klasse initialisiert wird, wird die alte Logdatei, falls vorhanden, gelöscht und eine neue leere Logdatei mit dem Dateinamen *mqtt.log* erstellt. Zusätzlich wird eine Variable "rowNumber" zum Zählen der Zeilennummer als Index für das Logfile initialisiert.

Anschließend wird die Verbindung zum MQTT Broker der Infrastruktur aufgebaut und nach dem Verbindungsaufbau das Topic *LP/+/+/Status/#* abonniert.

Um das Logfile zu laden wurde eine Hilfsmethode "loadLog" implementiert. Diese Methode iteriert Zeile für Zeile über das gesamte Logfile, von der ersten bis zur letzten Zeile. Jede Zeile wird nacheinander an eine Liste angehängt. Anschließend gibt die Methode diese Liste als Rückgabewert zurück.

Nachdem eine Zustandsänderung eines Aktors empfangen wurde, wird die Methode zum Schreiben in das Logfile aufgerufen. Da es möglich ist, dass eine weitere Zustandsänderung eines Aktors empfangen wird, während das Logfile geschrieben wird, muss eine gegenseitige Ausschlusssperre für das Logfile eingerichtet werden. Diese Ausschlusssperre wird mit Hilfe einer Mutex realisiert, die solange gehalten wird, wie das Logfile zum Schreiben geöffnet ist. Sobald das Logfile geschlossen wird, wird die Mutex wieder für andere Threads freigegeben.

Nachdem die Zählvariable "rowNumber" inkrementiert wurde, wird eine neue Zeile in das Logfile geschrieben. Die neue Zeile hat die Form:

```
rowNumber;<Topic>;<Payload>
```

Bei der Implementierung in die Infrastruktur des LP ist es vorgekommen, dass ein Aktor eine Zustandsänderung gemeldet hat, ohne den Zustand tatsächlich geändert zu haben. Gemeldete Zustandsänderungen ohne tatsächliche Zustandsänderung würden dazu führen, dass das Undo-Tool zwar den vorherigen Zustand wiederherstellt aber es keine sichtbare Auswirkung auf den Aktor hätte. Deswegen müssen diese doppelten Zustandsänderungen herausgefiltert werden.

Dazu wird zunächst der letzte gespeicherte Zustand des empfangenen Aktors ausgelesen und mit dem neuen Zustand verglichen. Sind beide Zustände identisch, wird der Speichervorgang abgebrochen und die Mutex wieder freigegeben. Ansonsten wird weiter mit der Speicherung fortgefahren.

Die beiden Schnittstellen der Persistenzklasse sind über zwei Getter-Methoden implementiert. Der Methode “getActors” wird über einen Parameter ein Suchschlüssel übergeben. Der Suchschlüssel ist Teil des Topics und definiert sich aus den gültigen Raumgruppen und Gerätegruppen des Topics. Anschließend lädt die Methode mit der Hilfsmethode “loadLog” das Logfile in eine lokale Variable. In einer Iteration über alle Einträge der Liste wird jedes Topic eines Eintrags mit dem Schlüssel verglichen. Ist der Schlüssel im Eintrag und noch nicht in der neuen Liste “keyRows” enthalten, wird der Eintrag hinzugefügt. Nach der Iteration wird diese “keyRows”Liste als Rückgabewert zurückgegeben.

Die zweite Methode ist die Methode “getStates”. Diese Methode funktioniert sehr ähnlich wie die Methode “getActors”, nur dass hier alle Zustände eines Aktors ausgelesen werden. Als Schlüsselparameter der Methode dient auch hier das Topic. Als Rückgabewert wird eine Liste zurückgegeben, die alle Zustandsänderungen des Schlüsselaktors enthält, sortiert von der ersten Zustandsänderung an Indexposition Null bis zur letzten Zustandsänderung.

Um eine Optimierung des Logfiles zu implementieren, wurde die Methode “optimize” implementiert. Diese wird automatisch jeden Tag von der Persistenzschicht aufgerufen. Wichtig ist, dass die Optimierung nach dem Start der Methode den Mutex annimmt, um den gegenseitigen Ausschluss der anderen Threads zu erreichen. Sollte während der Optimierung ein Zustandswechsel stattfinden, ohne dass die Optimierung abgeschlossen ist, könnte das Logfile der Optimierung durch eine geladene Version eines anderen Threads überschrieben werden.

Das Logfile wird dann mit der Hilfsmethode “loadLog” in einer lokalen Variablen gespeichert. Mit der Methode “getActors” werden alle gespeicherten Aktoren abgerufen und in einer Aktorenliste gespeichert. In einer neuen Liste “optLog” wird das Logfile optimiert aufgebaut, indem über die Aktorenliste iteriert wird und die folgenden Schritte für jeden Eintrag nacheinander ausgeführt werden:

- 1. Abrufen aller vorhandener States des Aktors mithilfe der “getStates” Methode
- 2. Rückwärts über die erhaltene States Liste iterieren
- 3. Bei jedem Eintrag überprüfen, ob die Zustandsgrenze erreicht wurde
- 4a. Nein: Eintrag wird in “optLog” hinzugefügt
- 4b. Ja: Eintrag wird nicht hinzugefügt und Iteration über diesen Aktor beendet
- 5. Wiederholen mit nächsten Aktor der Aktorenlsite

Die Zustandsgrenze wurde in dieser Umsetzung, wie im Design beschrieben, mit maximal zwanzig Zustandsänderungen pro Aktor definiert.

3.2.3 Integration in die Infrastruktur des Living Place

Das Webinterface über NodeRed wurde nur als experimentelles HMI für diese Implementierung verwendet und ist nicht notwendig, um in das Living Place integriert zu werden. Durch die Möglichkeit von HomeAssistant MQTT Schalter zu integrieren, können die Buttons des Dashboards fest in die Struktur des Lovelace Frontends des Living Place integriert werden. Über die Views der Räume, wie der Lounge 2.5, könnten die MQTT Schalter von HomeAssistant direkt als Cards dargestellt und geschaltet werden. Alternativ wäre auch eine direkte anzeige in dem Übersichts-Dashboard des LP 2.4 möglich.

Zusätzlich könnte das Undo Tool durch Sprachsteuerung, z.B. aus der Arbeit von Arthur Haarring[23] oder anderen MQTT fähigen Implementierungen, über Sprachbefehle gesteuert werden.

Das Undo-Tool selbst kann als Python-Skript einfach in die VM der Infrastruktur integriert werden. Zusätzlich bietet der HomeAssistant über eine Python Addon Schnittstelle die Möglichkeit, die Funktionalität von HomeAssistant direkt zu erweitern.

3.2.4 Interaktion mit dem Benutzer

Um den Benutzer das erfolgreiche Durchführen oder einen Fehler während des Undos anzuzeigen, kann das Dashboard von HomeAssistant genutzt werden. Durch NodeRed ist es aber auch möglich eigene Routinen zu starten, sobald das “success” oder das “failed” vom Undo erhalten wurde.

So ist zur Zeit der Umsetzung eine NodeRed Routine in der Infrastruktur des LP erstellt worden. Bei einem erhaltenen “success” blinkt das DMX Licht im LP fünf Sekunden lang grün auf. Sollte ein “failed” empfangen werden blinkt das DMX Licht fünf Sekunden lang rot auf.

3.3 Fazit Implementierung

Das Webinterface konnte mit Hilfe des NodeRed Dashboards gut dargestellt werden. Die Bausteinprogrammierung von NodeRed und die vielen verschiedenen Paletten, die in NodeRed zusätzlich installiert werden können, bilden eine gute Grundlage für die Umsetzung.

In der Programmiersprache Python konnte das Design der Implementierung gut umgesetzt werden. Schwierigkeiten gab es lediglich bei der Wahl der Darstellung der gesammelten Zustandsänderungen in Python. Es musste eine Darstellung gewählt werden, die leicht iterierbar ist und einen schnellen Zugriff auf einzelne Objekte ermöglicht. Wichtig war auch, dass eine Filterfunktion auf der Basis des Topics zur Verfügung steht.

Neben der Darstellung in einer einzelnen Liste wurde auch eine Umsetzung mittels Python Dictonaries in Betracht gezogen. Dort wäre eine direkte Suche nach dem Topic über die Schlüssel des Dictonarys möglich gewesen. Aufgrund der einfacheren Struktur der Liste und der einfacheren Iterationsmöglichkeit über die Liste wurde schließlich eine Python-Liste gewählt.

Bei der Implementierung war es auch sehr hilfreich, dass die Topics der LP strukturiert aufgebaut sind. Durch die Wildcards von MQTT konnte die Skalierbarkeit sehr einfach und effektiv umgesetzt werden. Auf die Skalierbarkeit wird im nächsten Kapitel näher eingegangen.

4 Evaluation der Lösung

In diesem Kapitel wird die umgesetzte Lösung gegen die Anforderungen aus Kapitel 2.5.1 durchgeführt. Danach folgt eine Betrachtung der Skalierbarkeit und Integration in Smart Home Umgebungen.

4.1 Funktionale Evaluation

Die funktionale Evaluation untersucht ausschließlich die technische Umsetzung und Funktionalität des Systems. Diese beginnt mit der Ausführung eines Undos. Anschließend wird die korrekte Umsetzung der Optimierung des Logfiles überprüft.

4.1.1 Ausführung eines Undos

Mit dem implementierten System können alle Anforderungen zur Wiederherstellung eines Zustandes von Aktoren innerhalb des Living Places realisiert werden. Die implementierte Lösung kommuniziert über die MQTT-Schnittstelle mit der Infrastruktur des Living Places. Sie reagiert auf die Eingaben des implementierten Dashboards.

Die Änderungen des Status werden in der Log-Datei gesammelt und können von dort ausgelesen werden. Über das implementierte Dashboard wird der Benutzer über ein erfolgreiches oder fehlgeschlagenes Undo informiert. Zusätzlich ist eine direkte Rückmeldung über die Umgebungsbeleuchtung des LP implementiert und in Betrieb.

4.1.2 Optimierung des Logfiles

Die gesammelten Zustände werden optimiert und jeder Aktor hält maximal 20 Zustandsänderungen im Logfile. Diese Optimierung findet einmal am Tag statt und konnte im Logfile beobachtet werden.

4.1.3 Round Trip Latenz

Für die Messung der Round Trip Latenz wurden drei verschiedene Experimente durchgeführt. Im ersten Experiment enthält das Logfile für jeden Aktor zwei Zustandsänderungen. Dies ist die minimale Anzahl von Zustandsänderungen pro Aktor, bei der ein Undo noch möglich ist.

Anzahl der Zeilen für das erste Experiment:

Anzahl Räume (4) * Anzahl Gerätegruppen (7) * minimale Anzahl Zustände (2) = 56 Einträge

Im zweiten Experiment wurde die Log-Datei bis zu der in der Arbeit festgelegten Grenze von 20 Zustandsänderungen für jeden Aktor des LP gefüllt. Anzahl der Zeilen für das zweite Experiment:

Anzahl der Räume (4) * Anzahl der Gerätegruppen (7) * maximale Anzahl der Zustände (20) = 560 Einträge.

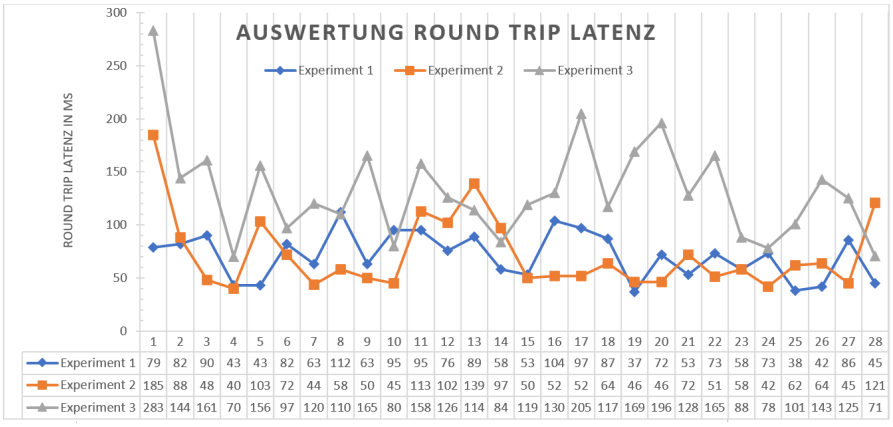
Im dritten Experiment wurde die Optimierung ausgeschaltet und es wurden zehnmal so viele Zustandsänderungen in der Log-Datei gespeichert wie im zweiten Experiment. Ziel des Experiments ist die Demonstration des Einflusses der Optimierung auf die Round Trip Latenz.

In jedem Experiment werden 28 Undos durchgeführt, um einen Mittelwert zu erhalten. Die Living Place VM wird als Testsystem verwendet. Auf dieser VM werden 6 Intel Xeon E3-1100 / E3-1200 Sandy Bridge CPUs mit einem Kerntakt von jeweils 2,1GHz betrieben.

Durch diese Experimente wird der Einfluss der Logfilegröße auf die Roundtripzeit sichtbar.

Die Abbildung 4.1 zeigt die Ergebnisse der drei Versuche. Betrachtet man die Mittelwerte in Abbildung 4.2, wird das Bild deutlicher. Während Experiment eins und zwei nur 0,75ms auseinander liegen, ist die Differenz zwischen Experiment zwei und Experiment drei mit 60,5ms um 84% größer als die Roundtrip-Zeit von Experiment zwei.

Mit maximal 185ms liegt die Round Trip Zeit aber unter der Vorgabe von 600ms (vgl. RQ-8 2.10) und ist so in einem akzeptablen Bereich für Benutzer.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Experiment 1	79	82	90	43	43	82	63	112	63	95	95	76	89	58	53	104	97	87	37	72	53	73	58	73	38	42	86	45
Experiment 2	185	88	48	40	103	72	44	58	50	45	113	102	139	97	50	52	52	64	46	46	72	51	58	42	62	64	45	121
Experiment 3	283	144	161	70	156	97	120	110	165	80	158	126	114	84	119	130	205	117	169	196	128	165	88	78	101	143	125	71

Abbildung 4.1: Auswertung der Round Trip Latenz der 3 Experimente

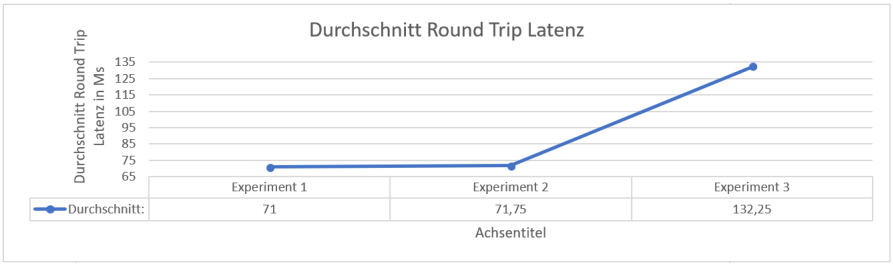


Abbildung 4.2: Auswertung der durchschnittlichen Round Trip Latenz der 3 Experimente

4.2 Skalierbarkeit und Integration

Solange die Struktur der MQTT Topics im LP eingehalten wird, können weitere Aktoren oder Raumunterteilungen eingeführt werden, ohne die Implementierung zu ändern. Neue Aktoren müssen in der Lage sein, einen Zustand zurückzumelden, um von der Persistenzebene erfasst zu werden. Dies sollte bei den meisten Geräten der Fall sein oder kann andernfalls mit einem NodeRed Adapter nachgebildet werden.

Es handelt sich um ein leichtgewichtiges Python-Skript. Als Installationsvoraussetzung ist nur eine externe Bibliothek erforderlich. Es setzt jedoch voraus, dass der Benutzer weiß, wie man Python-Skripte installiert.

5 Fazit und Ausblick

5.1 Fazit

Das Ergebnis dieser Arbeit ist eine Anwendung zur Implementierung einer Undo-Funktion, die es dem Benutzer ermöglicht, vorherige Zustände von Aktoren wiederherzustellen. Die Applikation wurde erfolgreich als einfach zu integrierende Anwendung implementiert. Die Round Trip Latenz liegt mit durchschnittlich 71ms in einem akzeptablen Bereich.

Ein Python-Skript war eine gute Möglichkeit, das Undo zu implementieren, um es einfach in die HomeAssistant-Umgebung integrieren zu können. Viele der Komponenten des HomeAssistant basieren auf der Programmiersprache Python und schaffen so Integrationsmöglichkeiten. Gerade die Aufteilung in zwei verschiedene Klassen und deren Verbindung über eine Schnittstelle schafft Austauschbarkeit. So kann die Persistierung leicht durch andere Persistierungsmöglichkeiten ersetzt werden.

Die Umsetzung der Persistenz mit Hilfe eines Logfiles war zwar gut umsetzbar, hatte aber das Problem der Optimierung zur Folge. Eine Optimierung ist zwar für jede Persistenz erforderlich, wird aber von den entsprechenden Programmen selbst durchgeführt. Diese Optimierung kommt dann auch ohne das Löschen von Zuständen, wie es bei der hier realisierten Lösung der Fall ist, aus.

Die Kommunikation über MQTT funktionierte mit Hilfe der Paho Bibliothek problemlos. Gerade die Wildcardfunktion des MQTT-Standards war hilfreich, um eine Skalierbarkeit zu erreichen. Allerdings ergeben sich dadurch zusätzliche Anforderungen an die Auswahl der Topics in einer Umgebung. Bei der Integration weiterer Aktoren könnten unterschiedliche Konzepte der Topic-Zuweisung zu Problemen führen.

5.2 Ausblick

Diese Arbeit ist ein erster Ansatz, die Kontrolle über das Smart Home wieder in die Hände der Nutzer zu legen. Sicherlich ist es ein guter Ansatz für Smart Home-Umgebungen, bestimmte Abläufe im Haus aus der Hand des Benutzers zu nehmen, um Komfort zu schaffen und Energie zu sparen. Aber ein überfürsorgliches Smart Home, das dem Benutzer immer mehr Entscheidungen abnimmt, sollte nicht das Ende sein. Selbst wenn es sich nur um eine Rollback-Funktion handelt, die in dieser Arbeit implementiert wurde, so handelt es sich doch um eine Form der Kontrolle, die wiedererlangt wird. Der Benutzer muss nicht hilflos zusehen, wie Geräte in seiner Umgebung automatisch umgeschaltet werden.

Dieses Rollback oder Undo schafft eine Basis, um die Akzeptanz von Smart Homes zu erhöhen. Es ist aber nur ein Anfang. Es muss eine Balance gefunden werden zwischen einem einfach zu bedienenden System mit seinen Automatismen auf der einen Seite und der Kontrolle des Nutzers über seine Umgebung auf der anderen Seite. Um dies zu erreichen, sollte aber auch nicht angestrebt werden, dass der Nutzer bei jeder Automatisierung erst um Erlaubnis gefragt werden muss.

Wie viel Kontrolle der Benutzer über seine Umgebung haben möchte, sollte er selbst entscheiden können.

Nicht nur im Smart Home besteht das Problem eines überfürsorglichen Systems. Ähnliche Probleme zeichnen sich auch in den Bereichen Smart Mobility und Smart Factory ab.

Ich selbst habe schon erlebt, dass mein Auto ohne mein Zutun die Entscheidung getroffen hat, in eine Richtung zu lenken. Diese Entscheidung wieder rückgängig zu machen, indem ich gegenlenke, ist in diesem Fall noch einfach und doch habe ich einen Moment der Hilflosigkeit gegenüber dem Verlust der Kontrolle empfunden.

Solche leicht umsetzbaren Undos sollten auch in anderen Bereichen verstärkt integriert werden, um die Nutzerakzeptanz dieser Systeme zu erhöhen. Wenngleich Spurassistenzsysteme ebenso wie Bremsassistenzsysteme sinnvolle Erweiterungen der Smart Mobility sind.

Es sollte weiter daran geforscht werden, diese Balance zwischen Fürsorge und Kontrolle zu finden und in andere Bereiche zu integrieren.

Literaturverzeichnis

[1] : *Build With Matter | Smart Home Device Solution.* – URL https://csa-iot.org/all-solutions/matter/. – Zugriffsdatum: 2024-02-15

[2] : *CSTI Creative Space for Technical Innovations.* – URL https://csti.haw-hamburg.de/. – Zugriffsdatum: 2024-02-13

[3] : *Eclipse Mosquitto.* – URL https://mosquitto.org/. – Zugriffsdatum: 2024-02-15

[4] : *Grafana: The open observability platform.* – URL https://grafana.com/. – Zugriffsdatum: 2024-02-13

[5] : *Home - Lightshark.* – URL https://lightshark.es/. – Zugriffsdatum: 2024-02-13

[6] : *Homematic online kaufen | ELV Elektronik.* – URL https://de.elv.com/technik-fuer-ihr-zuhause/smart-home-systeme/homematic/?utm_source=google&utm_medium=cpc&refid=Gads_Suche&utm_source=google&utm_medium=cpc&utm_campaign=Gads_de&refid=Gads&refid=Gads_Suche&gad_source=1&gclid=Cj0KCQiAw6yuBhDrARIsACf94RXoSedGLrzRn5LT3hOl1xLOyeizXVSuyPh13kIUoKgPKoUMuHwcB. – Zugriffsdatum: 2024-02-13

[7] : *InfluxDB | Real-time insights at any scale.* – URL https://www.influxdata.com/home/. – Zugriffsdatum: 2024-02-13

[8] : *Living Place |.* – URL https://livingplace.haw-hamburg.de/. – Zugriffsdatum: 2024-02-13

[9] : *MQTT V3.1 Protocol Specification.* – URL https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html. – Zugriffsdatum: 2024-02-25

[10] : *Node-RED*. – URL https://nodered.org/. – Zugriffsdatum: 2024-02-15

[11] : *Node-RED Dashboard 2.0*. – URL https://dashboard.flowfuse.com/. – Zugriffsdatum: 2024-02-13

[12] : *OASIS Message Queuing Telemetry Transport (MQTT) TC | OASIS*. – URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt. – Zugriffsdatum: 2024-02-25

[13] : *OSC index*. – URL https://opensoundcontrol.stanford.edu/. – Zugriffsdatum: 2024-02-13

[14] : *paho-mqtt: MQTT version 5.0/3.1.1 client class*. – URL http://eclipse.org/paho. – Zugriffsdatum: 2024-02-13

[15] : *Was ist BidCoS®? - HomeMatic-INSIDE*. – URL https://www.homematic-inside.de/faq/bidcos. – Zugriffsdatum: 2024-02-13

[16] Al-Fuqaha, Ala ; Guizani, Mohsen ; Mohammadi, Mehdi ; Aledhari, Mohammed ; Ayyash, Moussa: Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. 17, Nr. 4, S. 2347–2376. – URL https://ieeexplore.ieee.org/document/7123563. – Zugriffsdatum: 2024-01-15. – Conference Name: IEEE Communications Surveys & Tutorials. – ISSN 1553-877X

[17] Assistant, Home: *Home Assistant*. – URL https://www.home-assistant.io/. – Zugriffsdatum: 2024-02-13

[18] Balakrishnan, Sumathi ; Vasudavan, Hemalata ; Murugesan, Raja K.: Smart Home Technologies: A Preliminary Review. In: *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*, Association for Computing Machinery (ICIT 2018), S. 120–127. – URL https://doi.org/10.1145/3301551.3301575. – Zugriffsdatum: 2023-04-03. – ISBN 978-1-4503-6629-8

[19] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns | Wiley*. – URL https://www.wiley.com/en-us/Pattern+Oriented+Software+Architecture%2C+Volume+1%2C+A+System+of+Patterns-p-9780471958697. – Zugriffsdatum: 2024-02-16

[20] Cass, Aaron G. ; Fernandes, Chris S. T. ; Polidore, Andrew: An empirical evaluation of undo mechanisms. In: *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles*, Association for Computing Machinery (NordiCHI '06), S. 19–27. – URL https://dl.acm.org/doi/10.1145/1182475.1182478. – Zugriffsdatum: 2023-05-12. – ISBN 978-1-59593-325-6

[21] Castelli, Nico ; Ogonowski, Corinna ; Jakobi, Timo ; Stein, Martin ; Stevens, Gunnar ; Wulf, Volker: What Happened in my Home? An End-User Development Approach for Smart Home Data Visualization. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Association for Computing Machinery (CHI '17), S. 853–866. – URL https://doi.org/10.1145/3025453.3025485. – Zugriffsdatum: 2023-04-03. – ISBN 978-1-4503-4655-9

[22] Gaikwad, Pranay P. ; Gabhane, Jyotsna P. ; Golait, Snehal S.: A survey based on Smart Homes system using Internet-of-Things. In: *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*, S. 0330–0335

[23] Haarring, Arthur: Ein Open-Source basiertes System zur Sprachinteraktion in Smart-Homes.

[24] Han, Jing ; E, Haihong ; Le, Guan ; Du, Jian: Survey on NoSQL database. In: *2011 6th International Conference on Pervasive Computing and Applications*, URL https://ieeexplore.ieee.org/document/6106531. – Zugriffsdatum: 2024-02-15, S. 363–366

[25] Hargreaves, Tom ; Wilson, Charlie ; Hauxwell-Baldwin, Richard: Learning to live in a smart home. 46, Nr. 1, S. 127–139. – URL https://doi.org/10.1080/09613218.2017.1286882. – Zugriffsdatum: 2023-04-03. – Publisher: Routledge _eprint: https://doi.org/10.1080/09613218.2017.1286882. – ISSN 0961-3218

[26] Isah, Haruna ; Abughofa, Tariq ; Mahfuz, Sazia ; Ajerla, Dharmitha ; Zulkernine, Farhana ; Khan, Shahzad: A Survey of Distributed Data Stream Processing Frameworks. 7, S. 154300–154316. – URL https://ieeexplore.ieee.org/document/8864052. – Zugriffsdatum: 2024-02-15. – Conference Name: IEEE Access. – ISSN 2169-3536

[27] Kohrs, Christin ; Angenstein, Nicole ; Brechmann, André: Delays in Human-Computer Interaction and Their Effects on Brain Activity. 11, Nr. 1, S. e0146250.

– URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4712932/. – Zugriffsdatum: 2024-02-15. – ISSN 1932-6203

[28] Loregian, Marco: Undo for mobile phones: does your mobile phone need an undo key? do you? In: *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, Association for Computing Machinery (NordiCHI '08), S. 274–282. – URL https://dl.acm.org/doi/10.1145/1463160.1463190. – Zugriffsdatum: 2023-05-12. – ISBN 978-1-59593-704-9

[29] Pu, Cuiping ; Fang, Gang ; Ren, Jie ; Du, Yaowen: Research on Smart Home Based on MQTT. In: *Proceedings of the 3rd International Conference on Information Technologies and Electrical Engineering*, Association for Computing Machinery (ICITEE2020), S. 531–535. – URL https://doi.org/10.1145/3452940.3453044. – Zugriffsdatum: 2023-04-03. – ISBN 978-1-4503-8866-5

[30] Tanenbaum, Andrew S. ; Steen, Maarten v.: *Verteilte Systeme*. – URL https://elibrary.pearson.de/book/99.150005/9783863266684. – Zugriffsdatum: 2024-01-12. – ISBN: 9783863266684 Publisher: Pearson Deutschland

[31] Tarkoma, Sasu: *Publish / Subscribe Systems: Design and Principles*. Wiley. – Google-Books-ID: Ix32ugAACAAJ. – ISBN 978-1-119-95154-4

[32] Trossen, Arthur: *Das Gegenteil von GUT*. – URL https://www.in-mediation.eu/was-ist-das-gegenteil-von-gut/. – Zugriffsdatum: 2024-02-15. – Section: Feinsinniges

[33] Washizaki, Hironori ; Fukazawa, Yoshiaki: Dynamic hierarchical undo facility in a fine-grained component environment. In: *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*, Australian Computer Society, Inc. (CRPIT '02), S. 191–199. – ISBN 978-0-909925-88-8

[34] Yoon, Young S. ; Myers, Brad A.: Supporting selective undo in a code editor. In: *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, IEEE Press (ICSE '15), S. 223–233. – ISBN 978-1-4799-1934-5

[35] Zhu, Xiaoxiang ; Nie, Xiaowen ; Liu, Jibo: Time Series Database Optimization Based on InfluxDB. In: *2023 International Conference on Power, Electrical Engineering, Electronics and Control (PEEEC)*, URL https://ieeexplore.ieee.org/document/10414927. – Zugriffsdatum: 2024-02-15, S. 879–885

A Anhang

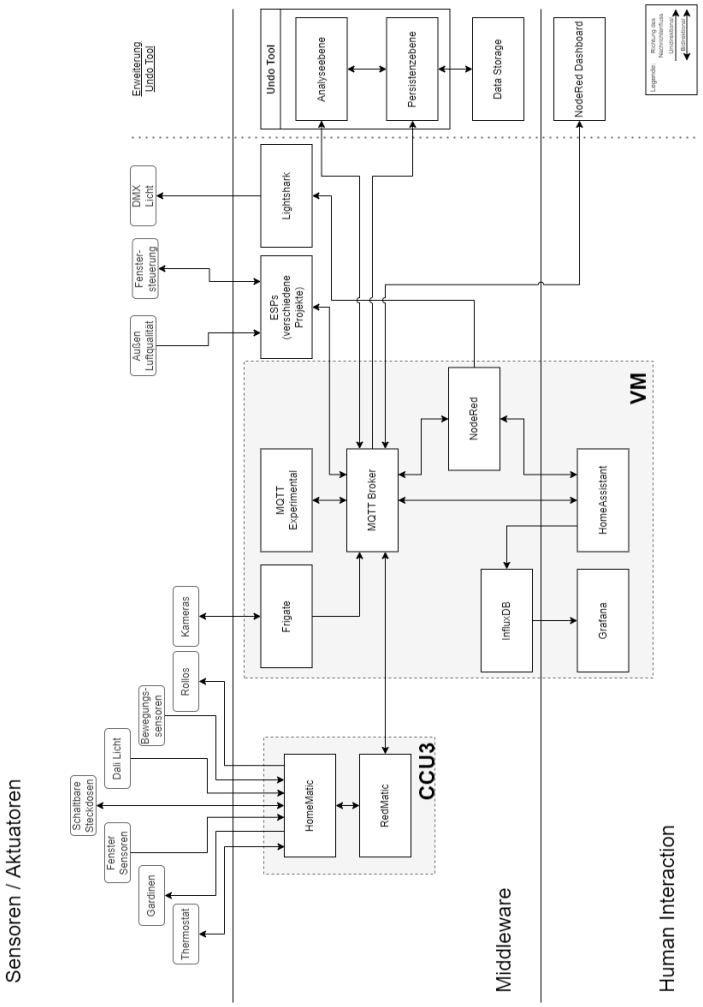


Abbildung A.1: Undo Tool Erweiterung der LP Architektur

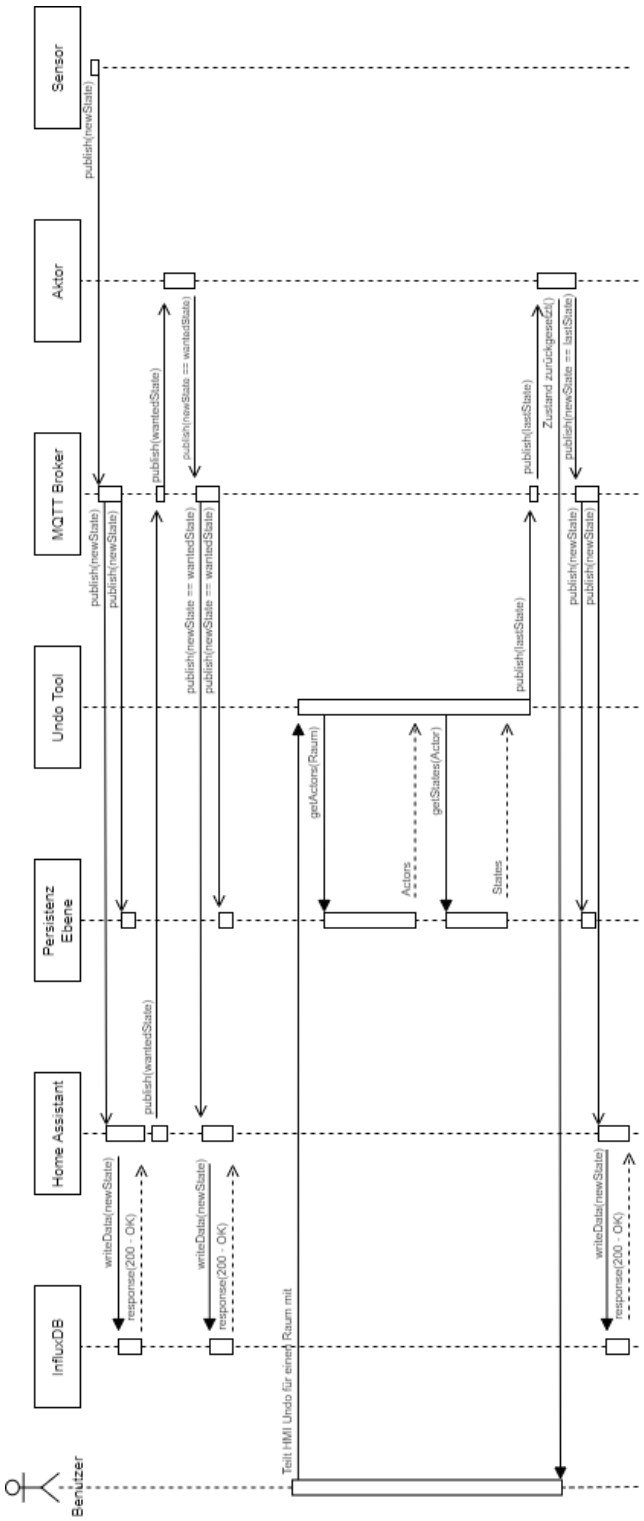


Abbildung A.2: Sequenzdiagramm des Szenarios

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original