Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Dorothee Laugwitz

Entwicklung einer Architektur zur Visualisierung von Smarthome-Umgebungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Dorothee Laugwitz

Entwicklung einer Architektur zur Visualisierung von Smarthome-Umgebungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Philipp Jenke
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 22. April 2016

Dorothee Laugwitz

Thema der Arbeit
Entwicklung einer Architektur zur Visualisierung von Smarthome-Umgebungen

Stichworte
Smart Home, Internet der Dinge, Visualisierung, Softwarearchitektur, RESTful

Kurzzusammenfassung
Die Automatisierung von Abläufen im privaten Haushalt wird zunehmend populärer und komplexer. Für Anwendungen im Bereich Smart Home sind daher passende Visualisierungen von besonderer Bedeutung. Ziel dieser Arbeit ist die Entwicklung einer Softwarearchitektur, über die Visualisierungslösungen gesammelt dargstellt werden können. Gleichzeitig soll die Verwaltung eines Netzes aus intelligenten Geräten und Sensoren ermöglicht werden.
Dorothee Laugwitz

Title of the paper
Development of an Architecture for the Visualisation of Smart Home Environments

Keywords
Smart Home, Internet of Things, Visualisation, Software Architecture, RESTful

Abstract
The automatisation of private homes continues to get more popular and complex. Therefore, adequate visualisations are of importance for Smart Home applications. This thesis aspires to develop a software architecture for displaying multiple results of visualisations. At the same time, the ability to manage a network of sensors and intelligent devices must be provided.

Inhaltsverzeichnis

1 Einführung **1**

1.1 Motivation 1
1.2 Zielsetzung 2
1.3 Gliederung 3

2 Grundlagen **4**

2.1 Smart Home 4
2.2 Das CgResearch Framework 5
2.3 HTTP 7
2.4 REST 8
2.4.1 Resource-Oriented Architecture 9
2.5 Schichtenarchitektur 11
2.6 NoSQL 12

3 Analyse **14**

3.1 Vergleichbare Arbeiten 14
3.1.1 FuseViz 14
3.1.2 CASAS 15
3.1.3 Living Place 16
3.1.4 Auswertung 17
3.2 Anforderungsanalyse 17
3.2.1 Funktionale Anforderungen 18
3.2.2 Nichtfunktionale Anforderungen 20
3.2.3 Integration in das CgResearch-Framework 21
3.3 Abgrenzung 21
3.4 Zusammenfassung 22

4 Konzept **23**
4.1 Architekturstil . . . 23
4.2 Domänenschicht . . . 25
4.2.1 Darstellung auf Ebenen . . . 25
4.2.2 Integration des Szenengraphen . . . 26
4.2.3 Verwalten von Geräten . . . 28
4.3 Datenhaltungsschicht . . . 28
4.3.1 Datenmanagement . . . 28
4.3.2 MongoDB . . . 30
4.3.3 Komponenten . . . 32
4.4 Präsentationsschicht . . . 35
4.4.1 Restlet . . . 36
4.4.2 Komponenten . . . 38
4.5 Beispielszenarien . . . 41
4.5.1 Aufruf über die REST-Schnittstelle . . . 41
4.5.2 Erstellen von Ebenen . . . 43
4.6 Zusammenfassung . . . 44

5 Bewertung **45**
5.1 Erfüllung der Anforderungen . . . 45
5.1.1 Funktionale Anforderungen . . . 45
5.1.2 Nichtfunktionale Anforderungen . . . 46
5.2 Performanztest . . . 47
5.3 Anwendungsbeispiele . . . 49
5.3.1 Zugriff über die REST-Schnittstelle . . . 49
5.3.2 Aufruf des Szenengraph . . . 50

6 Zusammenfassung und Ausblick **52**
6.1 Zusammenfassung . . . 52
6.2 Ausblick . . . 52

Literaturverzeichnis **54**

1 Einführung

Computer werden im alltäglichen Leben immer präsenter, verschwinden gleichzeitig jedoch zunehmend aus dem Sichtfeld. So werden auch im privaten Haushalt gerne intelligente Geräte genutzt, die über eine Verbindung zum Internet ferngesteuert werden können, zum Beispiel von einem Smartphone aus. So können über einen Knopfdruck die Fenster im Schlafzimmer geschlossen werden, während man auf der Arbeit sitzt. Heimautomatisierung ist dabei für den ein oder anderen zum Hobby geworden. In diesem Kontext entstand der Begriff des Smart Homes, welches seit einigen Jahren durch fortlaufend billiger werdende Computer auch außerhalb von Forschungsarbeiten immer relevanter wird.

1.1 Motivation

Obwohl der Markt rund um das Smart Home erst vergleichsweise jung ist, wird er oft als Zukunftsmarkt bezeichnet. Laut einer Prognose des Branchenverbands Bitkom [Vgl. Bit14] wird er kontinuierlich wachsen, sodass bis 2020 bereits eine Millionen intelligente Haushalte in Deutschland existieren sollen. Bei einer Umfrage, die von der Grieger Marktforschung durchgeführt wurde, gaben 30% der 1.017 befragten Deutschen an, bereits eine Smarthome-Anwendung zu nutzen. Weitere 50% sind an diesen Anwendungen interessiert. Besonders beliebt sind dabei Geräte aus den Bereichen Energiemanagement, Entertainment und Kommunikation [Vgl. Gru16]. Somit liegt das Smart Home im Trend. Viele Unternehmen sehen in diesem Markt Potential und bieten bereits entsprechende Produkte an. Im Handel findet man bereits diverse ferngesteuer-

te Kleingeräte, wie Lichtschalter oder Thermometer. Aber auch größere Geräte, wie intelligente Waschmaschinen und Kühlschränke, gehören inzwischen zum Angebot.

Für Nutzer und Hersteller entstehen im Bereich Smart Home jedoch oft Probleme bei der Einrichtung, die sich auf den Mangel an etablierten Standards zurückführen lassen. Häufig nutzen Geräte zur Kommunikation unterschiedliche Spezifikationen wie Bluetooth, ZigBee oder DECT-ULE und sind daher nicht immer zueinander kompatibel. Die Entwicklung von einheitlichen Standards und Smarthome-Plattformen ist somit ein wiederkehrendes Thema für Forschung und Industrie. Erst vor Kurzem schlossen sich zahlreiche namhafte Firmen zusammen, um sich auf einen Standard für die Kommunikation zwischen Geräten zu einigen [Vgl. Men16].

Weitere Probleme entstehen aufgrund der Menge an Daten, die in einem Smart Home durch die vielen verbauten Sensoren gesammelt werden. Diese müssen automatisch ausgewertet werden, damit der Bewohner über den Zustand seiner Umgebung informiert werden kann. Dabei entsteht oftmals erst genau dann ein Nutzen, wenn Daten verschiedener Natur in einem gemeinsamen Kontext betrachtet werden. So können zum Beispiel Messungen zur Raumtemperatur mit der täglichen Position des Bewohners kombiniert werden, um Aufschlüsse über dessen Heizverhalten und den entsprechenden Energieverbrauch zu geben. Ein wichtiges Werkzeug sind dabei passende Visualisierungen, die es Menschen erlauben, Muster und Tendenzen schnell zu erkennen. Eine Tabelle mit einer Reihe von Dateneinträgen ist auf den ersten Blick wenig aufschlussreich. Wird aus diesen Daten jedoch eine Kurve in einem Diagramm generiert, können Extremwerte sofort bestimmt werden. Visualisierungen sind daher für Smarthome-Umgebungen unerlässlich.

1.2 Zielsetzung

Diese Arbeit ist Teil eines Smart Home-Projektes der Hochschule für Angewandte Wissenschaften Hamburg. Das Projekt entstand unter der Leitung von Prof. Dr. Philipp Jenke und befindet sich zum Stand der Arbeit noch in den Kinderschuhen. Der Hauptfokus des Projektes liegt dabei auf der Entwicklung von Visualisierungslösun-

gen für Smarthome-Umgebungen. Um in Zukunft verschiedene Arbeiten in diesem Bereich unterstützen und vereinen zu können, soll im Rahmen dieser Arbeit eine Softwarearchitektur entwickelt werden, die die entsprechenden Grundlagen schafft.

1.3 Gliederung

Die Gliederung der Arbeit erfolgt in sechs Kapiteln. Nach dieser Einleitung folgt in Kapitel 2 eine Erläuterung der Grundlagen, die für das Verständnis der Arbeit benötigt werden. In Kapitel 3 werden verschiedene Arbeiten analysiert, die bereits in Rahmen anderer Smarthome-Projekte durchgeführt wurden. Die Ergebnisse der Beobachtungen dienen als Basis für die anschließende Entwicklung und Definition von Anforderungen, die an die zu entwickelnde Architektur gestellt werden. Kapitel 4 beschäftigt sich mit der Darstellung des Konzeptes, welches für die Erfüllung Anforderungen entwickelt wird. Dabei werden der Stil und die Komponenten der Architektur erläutert, sowie die verwendeten Technologien. Das entwickelte Konzept und dessen Umsetzung werden anschließend in Kapitel 5 bewertet. Kapitel 6 bietet schließlich eine Zusammenfassung dieser Arbeit, sowie einen Ausblick auf Erweiterungen.

2 Grundlagen

2.1 Smart Home

Unter einem *Smart Home* wird eine Wohnumgebung verstanden, in der durch Automatisierung die Lebensqualität der Bewohner gesteigert wird. Dabei sind verschiedene Geräte in der Regel drahtlos miteinander vernetzt und können über einen zentralen Punkt gesteuert werden, etwa über eine grafische Oberfläche. Die Abbildung 2.1 zeigt als Beispiel das Interface einer Smartphone-App, mit der sich Informationen über den Zustand verschiedener Bereiche des Smart Homes auslesen lassen.

Die Komplexität der Automatisierung von Geräten ist dabei sehr unterschiedlich. Dementsprechend ist es schwer zu definieren, ab wann eine Umgebung *intelligent* genannt werden kann. So gibt es auf der einen Seite Smart Homes, in denen gesammelte Daten lediglich visualisiert werden. Die Steuerung von Geräten wird dabei dem Menschen überlassen. Auf der anderen Seite werden Umgebungen entwickelt, in denen menschliches Verhalten beobachtet, ausgewertet und vorausberechnet wird, sodass Geräte komplett automatisch auf Menschen reagieren können. Dadurch ist es sogar möglich, Geräte zwischen verschiedenen Personen unterscheiden zu lassen.

Smart Homes werden oft mit den Begriffen *Ubiquitous Computing* und *Ambient Assisted Living* in Verbindung gebracht, da sich die jeweiligen Definitionen in vielen Bereichen überschneiden. Zusätzlich werden Smart Homes als Teil des *Internet of Things* betrachtet, da sich inzwischen viele alltägliche Haushaltsgegenstände an das Internet anschließen lassen.

[1] Abbildung aus http://users.informatik.haw-hamburg.de/~abo781/abschlussarbeiten.html

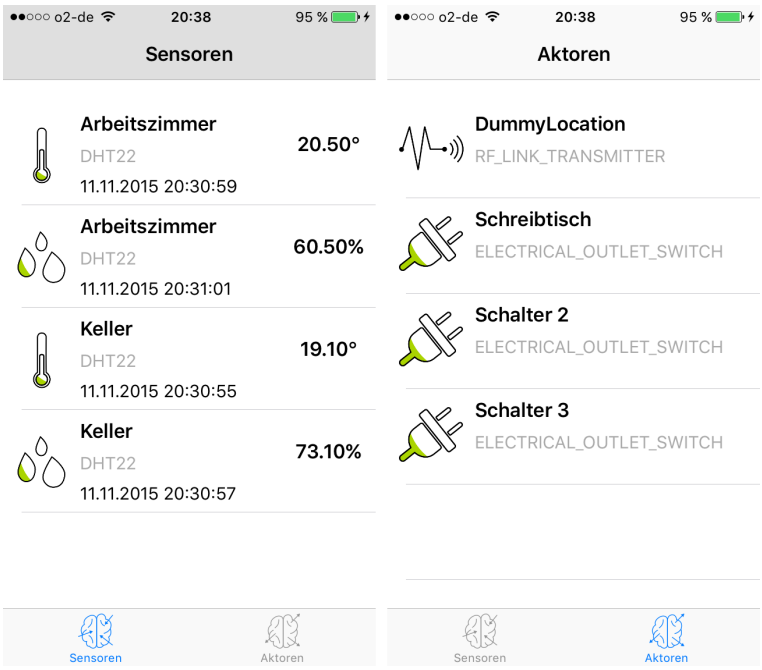

Abbildung 2.1: Eine mobile App zeigt Geräte des Smart Homes [1]

2.2 Das CgResearch Framework

Das CgResearch Framework[2] vereint Forschungsarbeiten, die unter der Leitung von Prof. Dr. Jenke durchgeführt werden. Es bietet grundlegende Funktionen zur Visualisierung von dreidimensionalen Objekten, welche in einem *Szenengraphen* verwaltet werden [Jen15]. Dabei handelt es sich um einen Baumgraphen, in dessen Knoten renderbare Daten enthalten sind. Abbildung 2.2 verdeutlicht diesen Aufbau. Wird eine Applikation des CgResearch Frameworks gestartet, wird der enthaltene Szenengraph berechnet und fortlaufend aktualisiert. Dabei kann festgelegt werden, ob die Darstellung durch jMonkey oder JOGL erfolgen soll. Zusätzlich besteht die Möglichkeit, ein eigenes Benutzerinterface zu implementieren. Die Umsetzung erfolgt mithilfe von Swing. Die für das Framework genutzte Programmiersprache ist dementsprechend Java 1.8.

[2] Verfügbar unter https://github.com/pjenke/computergraphics
[3] Abbildung aus [Jen15]

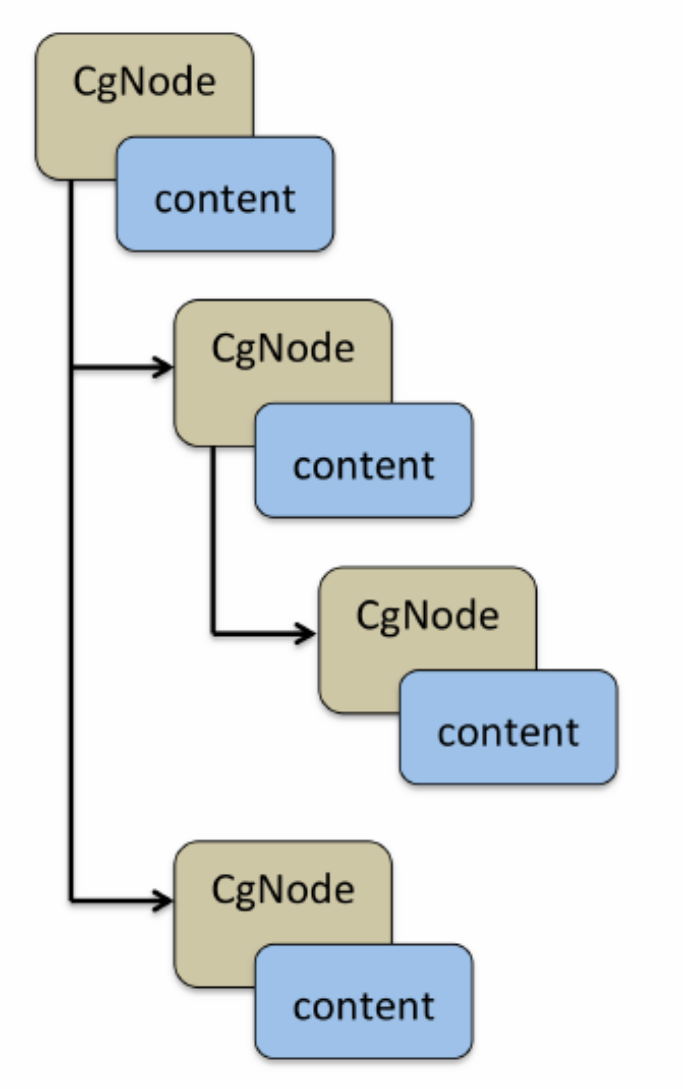

Abbildung 2.2: Baumstruktur des Szenengraph [3]

HTTP-Methode	Funktion
GET	Erfragt die Repräsentation einer Ressource.
PUT	Verändert eine existierende Ressource oder erschafft eine neue. Dabei weiß der Client, welche URI die neue Ressource haben soll.
POST	Verändert eine existierende Ressource oder erschafft eine neue. Dabei entscheidet der Server, welche URI die neue Ressource haben soll.
DELETE	Löscht eine existierende Ressource.

Tabelle 2.1: Die wichtigsten HTTP-Methoden

2.3 HTTP

Das *Hypertext Transfer Protocol* wird von Webkomponenten genutzt, um Daten auf der Anwendungsschicht zu übertragen. Der Austausch wird dabei durch einen Client initiiert, der eine Anfrage an einen Server stellt. Die Anfrage muss entsprechend des Protokolls strukturiert sein und alle Informationen enthalten, die der Server zur Verarbeitung benötigt. Dazu gehören unter anderem eine *URI*[4], mit der die geforderte Ressource bestimmt wird, sowie eine der standardisierten HTTP-Methoden. Die vier am häufigsten genutzten Methoden und ihre jeweilige Funktion sind in Tabelle 2.1 beschrieben.

Wenn ein Server einen HTTP-Request erhalten und verarbeitet hat, schickt er dem Client eine Antwort. Diese enthält einen Statuscode, welcher Aufschluss darüber gibt, wie die Anfrage verarbeitet wurde und ob sie erfolgreich war. Die Tabelle 2.2 zeigt einige HTTP-Statuscodes und ihre Bedeutungen. Zusätzlich enthält die Antwort eine Repräsentation der adressierten Ressource, falls dies im Request gefordert wurde. Nachdem der Server eine Antwort abgeschickt hat, ist die Kommunikation beendet. Da es sich bei HTTP um ein zustandsloses Protokoll handelt, speichert der Server keine Informationen über den Zustand der Verbindung zum Client.

[4]Unified Resource Identifier, Definition unter https://www.w3.org/DesignIssues/Axioms

Statuscode	Nachricht	Bedeutung
200	OK	Die Anfrage wurde erfolgreich bearbeitet.
201	Created	Die Anfrage wurde erfolgreich bearbeitet und es wurde eine Ressource erstellt.
400	Bad Request	Die Anfrage war fehlerhaft aufgebaut.
404	Not Found	Die angeforderte Ressource konnte nicht gefunden werden.
500	Internal Server Error	Es ist ein unerwarteter Fehler im Server aufgetreten.

Tabelle 2.2: Auswahl an HTTP-Statuscodes

2.4 REST

Die Abkürzung *REST* steht für *Representational State Transfer*. Es handelt sich dabei um eine Menge von Bedingungen für die Implementation von effizienten und skalierbaren Webanwendungen. Die Konzepte von REST wurden von Thomas Roy Fielding im Rahmen seiner Dissertation entwickelt, um als Leitfaden bei der Entwicklung von Internetstandards zu dienen. Somit stellt REST eine Repräsentation des idealen World Wide Webs dar. Eine Webanwendung wird genau dann RESTful genannt, wenn sie die entsprechenden Bedingungen erfüllt [Vgl. Fie00, S. 78-85]:

Client-Server Architektur: Dem Prinzip "Separation of concerns" folgend bestehen Webanwendungen aus Clients, die Anfragen stellen und Servern, die Anfragen verarbeiten.

Zustandslosigkeit: Ein Server speichert keine Informationen über den Zustand von Sitzungen. Anfragen an den Server müssen daher voneinander isoliert geschehen. Dabei müssen jegliche Informationen mitgegeben werden, die der Server zur Verarbeitung benötigt.

Cache: Daten können als *cacheable* markiert werden. Es ist einem Client erlaubt diese Daten zu speichern, um sie zu einem späteren Zeitpunkt wiederzuverwenden.

Einheitliche Schnittstellen: Die Kommunikation zwischen Webkomponenten erfolgt über standardisierte Schnittstellen. Diese unterliegen vier Bedingungen:

1. Ressourcen müssen identifizierbar sein.
2. Ressourcen müssen mithilfe von Repräsentationen veränderbar sein.
3. Jede Nachricht enthält genug Informationen, um selbstbeschreibend zu sein.
4. *Hypermedia as the Engine of Application State (HATEOAS)* – Clients navigieren Webanwendungen ausschließlich durch Hypermedia, die von einem Server als Teil einer Repräsentation bereitgestellt wird.

Schichten: Dem Prinzip des *Separation of concerns* folgend sind Webanwendungen in Schichten implementiert.

Code-On-Demand (optional): Clients können Applets oder Skripte herunterladen und ausführen.

Diese Bedingungen wurden mit dem Ziel entwickelten, die Komponenten des Internets skalierbar zu machen und ihre Komplexität zu verringern, indem einheitliche Schnittstellen geschaffen werden. Gleichzeitig soll die Performanz der Kommunikation zwischen Webkomponenten erhöht werden [Vgl. Fie00, S. 105].

2.4.1 Resource-Oriented Architecture

Die von Fielding entwickelten Konzepte setzen keine festen Rahmen für Webapplikationen voraus und lassen Details zur Umsetzung offen. Mit dem Ziel, eine tiefergehende Definition für RESTful Architekturen zu vermitteln, wurde in [RR07] der Begriff der *Resource-Oriented Architecture (ROA)* gefestigt. Im Gegensatz zu REST handelt es sich hier nicht um eine Sammlung von Bedingungen, sondern um einen Architekturstil,

welcher konkrete Vorgaben zur Implementation von RESTful Webservices macht. Im Folgenden werden die Hauptmerkmale von ROAs beschrieben.

Ein wichtiger Bestandteil von ROA ist die Bereitstellung von Daten als *Ressourcen*. Eine Ressource kann alles sein, was sich beschreiben lässt. Die zugrunde liegenden Daten können sich unter Umständen dynamisch ändern, der Name einer Ressource bleibt jedoch gleich. So beschreibt zum Beispiel die Ressource “die Zahl PI” stets den gleichen Wert, während die Ressource “das Wetter von gestern” vom aktuellen Datum abhängig ist.

Um das von Fielding beschriebene Konzept der einheitlichen Schnittstellen zu implementieren, wird in ROAs die Adressierung von Ressourcen durch URIs vorausgesetzt. Diese sind gleichzeitig der Name von genau einer Ressource und sollten daher aussagekräftig sein. In Verbindung mit HTTP-Methoden können so gezielt Ressourcen angesprochen werden. Die Verwendung des HTTP ist dabei typisch für ROAs, da es sich um ein zustandsloses Protokoll handelt. Jeder HTTP Request ist isoliert und enthält die vollständigen Informationen, die ein Server zur Verarbeitung des Requests benötigt. Somit wird eine weitere von Fielding geforderte Bedingung erfüllt, nämlich die Zustandslosigkeit der Verbindung zwischen Client und Server.

Ein weiterer wichtiger Bestandteil von ROAs ist die Verwendung von *Repräsentationen*. Hierbei handelt es sich um Daten, die den Zustand einer Ressource beschreiben. Diese Daten können dabei in verschiedenen Formaten vorliegen, zum Beispiel als Grafik, als JSON-Objekt oder als HTML-Datei. Bei mehreren verfügbaren Optionen wird über den Request entschieden, welches Format geliefert wird.

Abbildung 2.3 stellt einen typischen ressourcenorientierten Request an einen Webservice dar. Ein Client schickt eine HTTP GET-Methode an einen Server, der in diesem Falle Informationen zu verschiedenen Katzenarten bereitstellt. Mithilfe der URI kann der Server bestimmen, welche Ressource gefordert ist und antwortet mit der entsprechenden Repräsentation. Am Ende speichert der Server keine Informationen über die Verbindung.

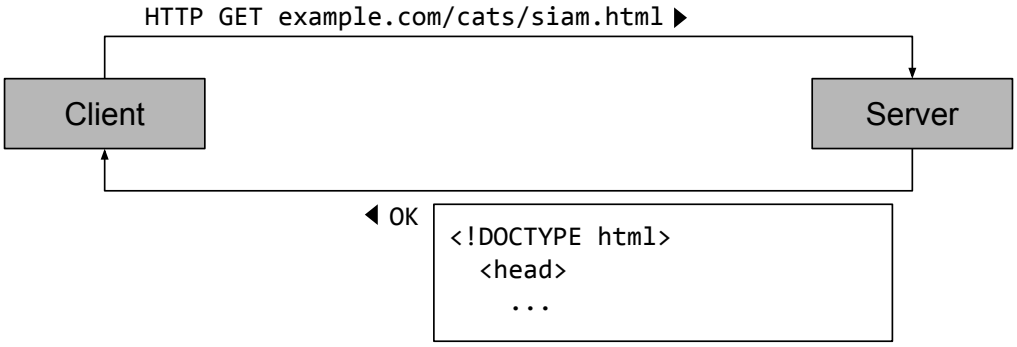

Abbildung 2.3: Beispiel-Request an einen RESTful Webservice

Für die Entwicklung von Webkomponenten werden inzwischen viele Frameworks genutzt, die ressourcenorientierte Architekturen unterstützen oder sogar voraussetzen. Beispiele für solche Frameworks sind *Django* und *Ruby on Rails*.

2.5 Schichtenarchitektur

Eine weit verbreitete Technik bei der Entwicklung von Softwarearchitekturen ist die Aufteilung von Komponenten auf voneinander getrennten Ebenen. Die Trennung anhand von Funktionalitäten geschieht in der Regel nach dem Drei-Schichten-Prinzip, welches in [Fow03, S. 17-24] beschrieben wird. Jede Schicht darf dabei nur auf die Schnittstellen ihrer direkt darunterliegenden Schicht zugreifen. Die Bezeichnung der Schichten, sowie ihre Zuständigkeiten, sind in Tabelle 2.3 dargestellt.

Die Trennung durch Ebenen hat nach Fowler eine Reihe von Vorteilen:

- Die Funktion einer einzelnen Ebene kann verstanden werden, ohne viel über die anderen Ebenen zu wissen.
- Das Austauschen von Implementierungen wird erleichtert.
- Abhängigkeiten zwischen den Ebenen werden minimiert.

Bezeichnung	Zuständigkeit
Präsentationsschicht	Stellt eine Schnittstelle zur Interaktion mit dem Nutzer zur Verfügung. Dies kann zum Beispiel über eine grafische Oberfläche oder eine textbasierte Konsole geschehen.
Domänenschicht	Verwaltet die Geschäftslogik der Anwendung.
Datenhaltungsschicht	Zuständig für die Kommunikation mit der Datenbank oder Messaging-Systemen.

Tabelle 2.3: Zuständigkeiten in einer Drei-Schichten-Architektur

- Implementierte Schichten sind wiederverwendbar und können von mehreren höherliegenden Schichten genutzt werden.

Die Trennung durch Ebenen hat nach nach Fowler jedoch auch Nachteile. So kann es passierten, dass für das Hinzufügen einer neuen Funktion eine Änderung auf jeder Ebene nötig ist, etwa wenn durch das Klicken auf einen Button der grafischen Oberfläche ein Datenbankeintrag getätigt werden soll. Zusätzlich können Aufrufe, die über mehrere Ebenen getätigt werden, die Performanz der Software verschlechtern. Da die Vorteile jedoch überwiegen und sich die Qualität des Quellcodes erheblich verbessert, ist die Implementierung von Schichten inzwischen ein Standard der Softwareentwicklung.

2.6 NoSQL

NoSQL steht für *non SQL* und ist ein Überbegriff für alle Arten von Datenbanken, die nicht relational sind. Im Gegensatz zu SQL-Datenbanken, bei denen Datenmodelle mithilfe von Tabellen fest definiert sind, muss in einer NoSQL-Datenbanken in der Regel kein Schema implementiert werden. Das Datenmodell bleibt somit flexibel. Anhand von Strategien, nach denen Daten gespeichert und gelesen werden, lassen sich die NoSQL-Datenbank in verschiedene Kategorien einteilen. Im Folgenden werden einige dieser Kategorien und ihre jeweiligen Strategien beschrieben.

Key-Value-Datenbanken: Auf jeden Eintrag in der Datenbank verweist ein eindeutiger Schlüssel, mit dem auf die Daten zugegriffen werden kann. Ein Beispiel für eine solche Datenbank ist Redis.

Dokumentenorientierte Datenbanken: Hier werden Daten in Form von Dokumenten gespeichert, deren Inhalt in der Regel aus beliebig vielen Key-Value-Paaren besteht. Die Dokumente werden so gespeichert, dass sie nachträglich identifizierbar sind. Sie besitzen also zum Beispiel eine eindeutige ID. Bekannte dokumentenorientierte Datenbanken sind CouchDB und MongoDB.

Graphdatenbanken: Um komplexe Vernetzungen abbilden zu können, werden die Relationen zwischen zwei Entitäten als Kanten zwischen zwei Knoten gespeichert. Das resultierende Datenmodell ist ein Graph. Ein Beispiel für solch eine Datenbank ist Neo4j.

Spaltenorientierte Datenbanken: Für diese Datenbanken werden zwar Tabellen genutzt, die Daten werden jedoch nicht zeilenorientiert, sondern spaltenorientiert ausgelesen. Ein Beispiel für solch eine Datenbank ist Cassandra.

3 Analyse

Dieses Kapitel befasst sich mit der Analyse der in 1.1 vorgestellten Problemstellung. Es werden zunächst Beobachtungen zu verwandten Arbeiten aus dem Bereich Smart Environments gemacht. Eine kurze Erläuterung der jeweiligen Ziele und verwendeten Technologien soll Aufschluss über Anwendungsfälle geben, die typisch für Smart Homes sind. Die Ergebnisse dieser Beobachtungen dienen anschließend als Grundlage für die Entwicklung von Anforderungen an die Architektur dieser Arbeit.

3.1 Vergleichbare Arbeiten

3.1.1 FuseViz

FuseViz ist ein Framework zur Fusionierung und Visualisierung von Daten, die in einem Smart Home gesammelt werden. Es wurde an der University of Texas at Arlington entwickelt [Vgl. GD12].

Ein großer Fokus liegt bei FuseViz auf der möglichst komfortablen Interaktion eines Bewohners mit dessen intelligenter Umgebung. Daten werden automatisch gesammelt, aufbereitet und durch eine beliebige Anzahl von Applikationen visualisiert. Diese Applikationen unterliegen dabei dem Anspruch, von jedem Gerät aus (zum Beispiel Smartphone oder Tablet) bedienbar zu sein.

Die Architektur von FuseViz ist auf vier Ebenen realisiert. Die Sammlung von Daten aus verschiedenen Quellen geschieht auf der untersten Ebene. Da auch größere Mengen von Anfragen verarbeiten werden sollen, haben sich die Entwickler dazu entschieden, die

NoSQL-Datenbank CouchDB zu verwenden. Über deren RESTful API können Geräte ihre Daten im JSON-Format direkt an die Datenbank schicken. Diese werden nach erfolgreicher Validierung auf der zweiten Ebene persistiert. Anschließend werden die Daten auf der dritten Ebene unter Verwendung von MapReduce-Funktionen fusioniert. Applikationen können dann über HTTP GET-Requests auf die Ergebnisse der Fusion zugreifen und diese visuell darstellen. Sie bilden die vierte Ebene.

3.1.2 CASAS

CASAS (Center for Advanced Studies in Adaptive Systems) ist ein Projekt der Washington State University, bei dem eine Vielzahl von Arbeiten zu intelligenten Umgebungen entstanden sind. Das Erkennen und Vorhersagen von menschlichen Bewegungsmustern durch die Software werden hier vorrangig untersucht [Vgl. Coo+12].

Bei der Architektur von CASAS handelt es sich um eine Drei-Schichten-Architektur. Die unterste Schicht besteht aus Sensoren und Aktoren, welche zu einem drahtlosen ZigBee-Netzwerk zusammengeschlossen sind. Die oberste Schicht besteht aus verschiedenen Applikationen, wie zum Beispiel der Aktivitätserkennung oder der Überwachung der Energieeffizienz. Beide Schichten sind mit einer Middleware verbunden, über die jegliche Kommunikation durch den Austausch von Nachrichten über XMPP läuft. Hierfür wird ein Publish/Subscribe-Manager genutzt, der eine Reihe von Kanälen verwaltet. Darüber hinaus ist die Middleware für die Vergabe von Zeitstempeln und eindeutigen IDs verantwortlich.

PyViz

Während der Arbeit an CASAS wurden auch mehrere Frameworks zur Visualisierung der gesammelten Daten entwickelt. Dabei stellte sich *PyViz* als besonders wertvoll heraus, dessen Name durch die Implementation in Python geprägt ist [Vgl. TC11].

PyViz kann als Agent mit der CASAS Middleware kommunizieren und sich für Kanäle registrieren. Empfangene Daten können durch eine Anbindung an eine SQL-Datenbank

gespeichert und wieder gelesen werden. Zusätzlich ist es möglich, Daten aus lokal gespeicherten Dateien zu lesen. Die anschließende Visualisierung erfolgt durch die Generierung von SVG-Grafiken auf Grundlage der erhaltenen Smarthome-Daten.

3.1.3 Living Place

Das *Living Place* ist ein laufendes Projekt an der Hochschule für Angewandte Wissenschaften Hamburg. Es handelt sich dabei um eine Smarthome-Umgebung, die aus einem komplett funktionsfähigen Wohnbereich und einem Kontroll- und Entwicklungsbereich besteht. Seit der Gründung des Projektes in 2009 ist das Living Place immer wieder Objekt von interdisziplinären Forschungsarbeiten zum Thema Ubiquitous Computing. Dabei wird hauptsächlich untersucht, auf welche Art ein Bewohner mit seinem Smart Home interagiert.

Der Wohnbereich des Living Place ist mit Sensoren ausgestattet, welche Informationen verschiedener Natur liefern. So geben zum Beispiel Bewegungssensoren Aufschluss über die Position von Menschen oder Gegenständen. Zusätzlich können Daten über das Wetter oder Ähnliches aus dem Internet abgefragt werden. Die Auswertung und Interpretation dieser Daten geschieht mithilfe einer Software-Architektur, die bereits Thema einiger Forschungsarbeiten war.

Die ursprüngliche Software des Living Place wurde im Stile einer Blackboard-Architektur realisiert [Vgl. Ell+11]. Mit dem Ziel Latenzen beim Austausch von Nachrichten zu verringern, wurde im Rahmen von [Eic14] eine agentenbasierte Middleware entwickelt. Diese steht den Entwicklern nun als Alternative zum Blackboard zur Verfügung. Beide Lösungen nutzen die nachrichtenbasierte Kommunikation über *ActiveMQ*, bei der Nachrichten im JSON-Format entsprechend des Publish-Subscribe-Patterns ausgetauscht werden.

3.1.4 Auswertung

Die Untersuchung bestehender Projekte bietet einen Ausblick auf Anwendungsfälle, welche typisch für Smart Homes sind. Die grundlegende Voraussetzung für ein derartiges System ist die Integration einer physikalischen Schicht. Ein Netz aus Sensoren und Aktoren muss mit dem entsprechenden Smarthome-System gesteuert werden können. Dabei lohnt es sich, den Austausch von Informationen zwischen Geräten und Anwendungen möglichst effizient zu gestalten, sodass zum Beispiel Aktoren schnell reagieren können. Zusätzlich ist es notwendig, diese Informationen zu speichern, um Operationen auch auf Basis von historischen Daten ausführen zu können.

Auffällig ist, dass Smart Homes potentiell auf vielen verschiedenen Ebenen weiterentwickelt werden. Durch die wachsende Auswahl an intelligenten Geräten, die in einem Smart Home genutzt werden können, entstehen immer wieder neue Anwendungsfälle und Ideen. Forschungsarbeiten in diesem Bereich werden dabei oft auch interdisziplinär durchgeführt. Dies macht Smarthome-Projekte besonders agil, wodurch entsprechende Anforderungen entstehen. So sollte bei der Konzipierung der Softwarearchitektur darauf geachtet werden, Strukturen möglichst flexibel zu halten.

Neben FuseViz, CASAS und dem Living Place existieren noch eine Vielzahl von weiteren Projekten aus dem Bereich der intelligenten Umgebungen. Die vorgestellten Projekte bilden also die vergleichbaren Arbeiten nicht vollständig ab und sollen lediglich als repräsentative Beispiele dienen.

3.2 Anforderungsanalyse

In diesem Abschnitt wird erläutert, welche Anforderungen an die zu entwickelnde Architektur gestellt werden. Dabei wird unterschieden, ob eine Anforderung funktional oder nichtfunktional ist. Während die funktionalen Anforderungen vor allem aktuelle Industrietrends aufgreifen, basieren die nichtfunktionalen Anforderungen auf den Beobachtungen zu den in 3.1 vorgestellten Projekten.

Ziel der zu entwickelnden Architektur soll in erster Linie jedoch nicht sein, die funktionalen Anforderungen selbst zu erfüllen. Vielmehr soll eine Plattform entwickelt werden, die eine nachträgliche Umsetzung dieser Anforderungen unterstützt und die in möglichst viele Richtungen erweiterbar ist. Die funktionalen Anforderungen dienen daher als Richtlinien für die Implementierung von Beispielobjekten, mit denen die Erfüllung der nichtfunktionalen Anforderungen getestet werden soll.

3.2.1 Funktionale Anforderungen

Unter funktionalen Anforderungen werden Funktionen verstanden, die ein System seinem Endbenutzer bieten muss. Die folgenden vier Szenarien beschreiben grundlegende Anwendungsfälle, welche bei der Interaktion zwischen dem Nutzer und einem Smarthome-System auftreten.

Szenario I

Name	Registrierung von Geräten
Akteure	Nutzer, System
Beschreibung	Der Nutzer eines Smart Homes möchten gerne eine Sammlung von Geräten verwalten. Dies geschieht über eine zentrale Schnittstelle, über die neue Geräte registriert und bei Bedarf wieder abgemeldet werden können. Die Identität dieser Geräte muss daher eindeutig bestimmbar sein.
Beispiel	Ein Bluetooth-Thermometer wird angeschlossen.

Szenario II

Name	Kommunikation mit Geräten
Akteure	Geräte, System
Beschreibung	Die Geräte eines Smart Homes erheben Daten, die ausgewertet werden sollen. Es muss einen Punkt geben, an dem die übermittelten Daten gespeichert und verarbeitet werden können. Zugleich müssen auch die Geräte selbst ansprechbar sein, um gesteuerte Automatisierung zu ermöglichen.
Beispiele	• Ein Thermometer übermittelt die aktuelle Temperatur. • die Klimaanlage wird automatisch eingeschaltet, wenn das Thermometer über 26°C misst.

Szenario III

Name	Gruppierung von Geräten
Akteure	Nutzer, System
Beschreibung	Um dem Nutzer die Verwaltung von Geräten zu erleichtern soll es die Möglichkeit geben, diese logisch zu gruppieren. Eine solche Gruppe soll auch benannt werden können.
Beispiele	Ein Thermometer und ein intelligenter Lichtschalter bilden die Gruppe *Küche*.

Szenario IV

Name	Darstellung von Daten
Akteure	Nutzer, System
Beschreibung	Die gespeicherten Daten sollen durch passende Visualisierungen dargestellt werden.
Beispiele	Eine App auf einem Smartphone stellt vergangene Messungen in einer Kurve dar.

Die beschriebenen Szenarien sollen in erster Linie einen Leitfaden bieten, nach dem sich das Konzept der Architektur orientiert. Sie beschreiben jedoch nicht die vollständigen funktionalen Anforderungen, die an ein Smarthome-System gestellt werden. Weitere Anforderungen können durchaus existieren oder in Zukunft entstehen.

3.2.2 Nichtfunktionale Anforderungen

Unter nichtfunktionalen Anforderungen werden Anforderungen an die Qualität der Software verstanden. Dabei kann unterschieden werden, ob die Qualitätsansprüche aus Sicht eines Entwicklers oder eines Benutzers entstehen. Da die zu entwickelnde Architektur als Grundlage für ein neues Projekt dient, werden die qualitativen Ansprüche besonders stark gewichtet. Nachfolgende Forschungsarbeiten im Bereich Smarthome-Visualisierung sollen bestmöglich unterstützt werden. Für die Entwicklung der nichtfunktionalen Anforderungen wird sich nur auf die Verwendung der Software für die Forschung konzentriert, sodass Anforderungen aus Sicht eines eventuellen Endnutzers nicht beachtet werden.

Folgende nichtfunktionale Anforderungen werden an die Software-Architektur gestellt:

Erweiterbarkeit Am Beispiel des Living Place wird deutlich, dass Smart Home Projekte Potential zur kontinuierlichen Erweiterung haben. Die Erweiterbarkeit der

dazugehörigen Software muss also gewährleistet werden, sodass nachfolgende Arbeiten mit wenig Aufwand an die bestehende Software anschließen können. Besonders für Arbeiten im Bereich der Visualisierung gilt die Voraussetzung, dass ein Zugriff von mehreren Geräten aus möglich ist. Schnittstellen sollten daher klar definiert und universal verwendbar sein.

Robustheit Fehler bei der Nutzung der Software müssen so behandelt werden, dass ihre Funktionsweise nicht eingeschränkt wird. Für jede Eingabe muss es eine definierte Reaktion geben. Ursachen für Fehlermeldungen müssen deutlich erkennbar sein.

Wartbarkeit Die Architektur soll für zukünftige Änderungen offen bleiben. Fehlerursachen müssen behoben werden können.

Performanz Mit Hinblick auf das potentielle Wachstum des Smart Home Projekts soll die Performanz vor allem bei der Wahl von Technologien eine Rolle spielen.

3.2.3 Integration in das CgResearch-Framework

Eine übergreifende Anforderung, die von Professor Jenke gestellt wurde, ist die Integration der Architektur in das vorhandene CgResearch-Framework. Die Einbindung des Szenegraphen soll ermöglichen, die Ergebnisse verschiedener Forschungsarbeiten gesammelt zu visualisieren. Die Architektur soll daher über eine Schnittstelle verfügen, mit der die Übergabe von Knoten des Szenengraphen möglich wird.

3.3 Abgrenzung

Um eine erfolgreiche Softwarearchitektur zu implementieren, müssen viele verschiedene Bereiche beachtet werden. Auf die folgenden Anforderungen wird jedoch im Laufe der Arbeit nicht weiter eingegangen, da eine Diskussion dieser Bereiche über den zeitlichen Umfang dieser Arbeit hinausgeht:

- Sicherheit und Datenschutz von Endnutzern
- Installation und Auslieferung der Software
- Entwicklung grafischer Oberflächen zur Steuerung
- Anbindung an ein echtes Sensornetzwerk

3.4 Zusammenfassung

In diesem Kapitel wurden zunächst einige Arbeiten vorgestellt, in denen ebenfalls Architekturen für intelligente Umgebungen entwickelt wurden. Auf Basis der Beobachtungen, die sowohl zu den vorgestellten Projekten, als auch zu Trends im Bereich der Heimautomatisierung gemacht wurden, konnten anschließend eine Reihe von Anforderungen an die Softwarearchitektur der Smarthome-Umgebung definiert werden.

4 Konzept

In diesem Kapitel wird das Konzept für die Software-Architektur zur Visualisierung von Smarthome-Umgebungen beschrieben. Entscheidungen zur Architektur und den verwendeten Technologien basieren dabei auf den Anforderungen, die in Kapitel 3 entwickelt wurden.

4.1 Architekturstil

In 2.5 wurde bereits das Prinzip der Drei-Schichten-Architektur und die Vorteile einer solchen Implementierung beschrieben. Aufgrund der verbesserten Erweiterbarkeit, die eine solche Architektur bietet, ist die Realisierung von Softwaresystemen als Schichtenarchitektur inzwischen eine bewährte Technik. Dadurch entsteht zusätzlich der Effekt, dass Entwickler in der Regel mit diesen Strukturen bereits vertraut sind, was ihnen die Einarbeitung in ein solches Projekt erleichtert. Aus diesen Gründen soll die Software des Smarthome-Projekts ebenfalls als Drei-Schichten-Architektur realisiert werden.

Ein alternativer Ansatz wäre die Implementation einer Blackboard-Architektur, in der mehrere voneinander unabhängige Programme an einer gemeinsamen Lösung arbeiten [Vgl. Bus+96, S. 71-95]. Die Erfahrungen aus dem in 3.1 vorgestellten Projekt Living Place, bei dem ein solches Blackboard genutzt wird, zeigen jedoch einige Probleme bei der Verwendung dieses Architekturstils für Smarthome-Umgebungen auf. So ist das System laut [Eic14, S. 21] durch die hohe Anzahl an Programmen, die im Rahmen von Forschungsarbeiten entstanden sind, inzwischen sehr komplex, sodass Fehler nur

schwer gefunden werden können. Zusätzlich ist der Stil für Visualisierungslösungen nicht geeignet, da diese unabhängig voneinander entwickelt werden.

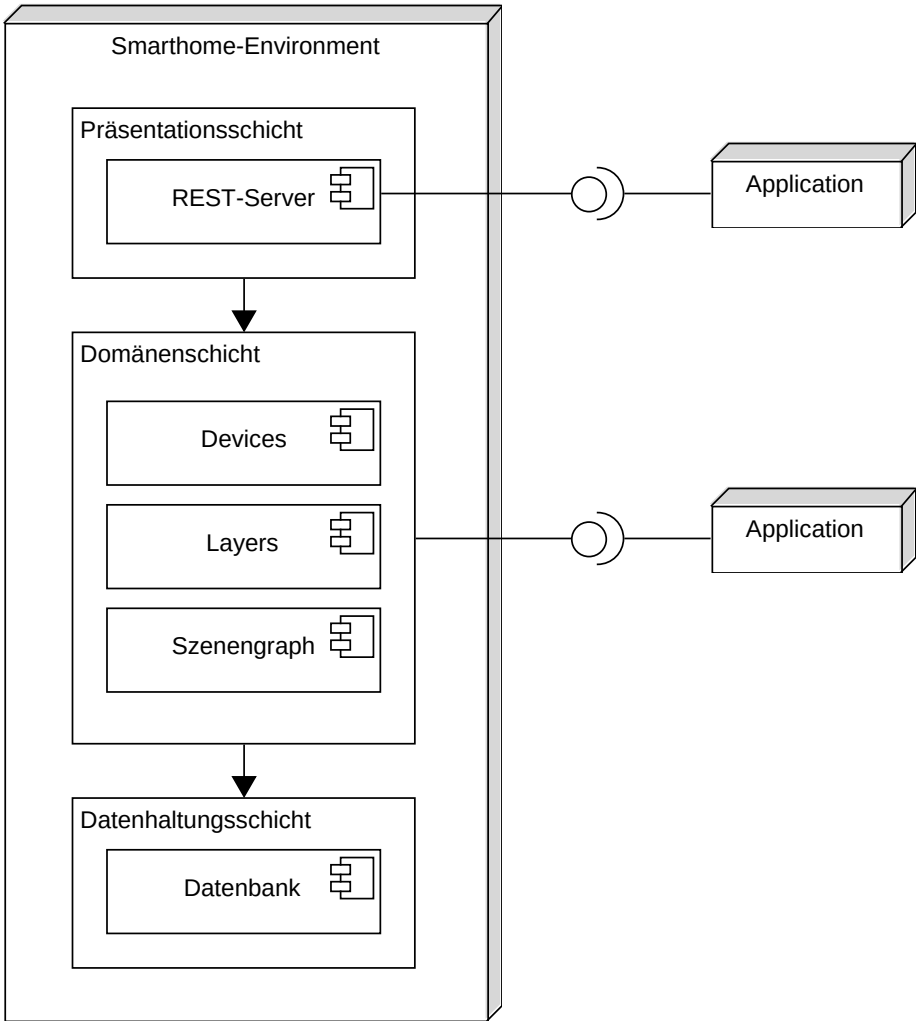

Abbildung 4.1: Architektur des Smarthome-Environment

Abbildung 4.1 zeigt die Implementierung des Smarthome-Systems in drei Schichten. Auf der untersten Schicht wird die Persistierung von Daten mithilfe einer Datenbank umgesetzt. Auf der Domänenschicht werden die Akteure des Smart Homes und Operationen auf diese abgebildet. Schließlich befindet sich auf der Präsentationsschicht

eine REST-Schnittstelle, durch die ein Zugriff auf das Smarthome-System über HTTP ermöglicht wird. Die Konzepte zu den einzelnen Schichten werden im Folgenden ausführlich beschrieben.

4.2 Domänenschicht

Die Domänenschicht bildet die Logik des Smart Homes ab. Zum Stand der Arbeit implementiert die Domänenschicht nur genau die Funktionalität, die zur Erfüllung der in 3.2 entwickelten Anforderungen benötigt wird. Sie soll dadurch vor allem als Ansatzpunkt für zukünftige Erweiterungen gelten und wird aus diesem Grund überschaubar gehalten.

4.2.1 Darstellung auf Ebenen

Durch die Beobachtung verschiedener Projekte in 3.1 lässt sich erkennen, dass Daten, die in einem Smart Home entstehen, oft verschiedener Natur sind. Dies hat zur Folge, dass auch die daraus entstehenden Visualisierungen entsprechend unterschiedlich sind. So können zum Beispiel Heatmaps, Diagramme oder Grundrisse dynamisch generiert werden. Trotz der Vielfalt soll jedoch ermöglicht werden, diese Visualisierungen an einer zentralen Stelle zu steuern.

Ziel ist es daher, die Ergebnisse verschiedener Visualisierungs-Projekte auf voneinander getrennten Ebenen zu speichern. Ein ähnliche Lösung wurde bereits im Rahmen von [Hün13] für die Architektur des MARS-Systems verwendet. Dabei handelt es sich um ein Framework zur agentenbasierten Simulation von Ökosystemen. Die verschiedenen Komponenten der Simulation werden auf Ebenen zusammengefasst, welche anschließend als Plugins der Simulation hinzugefügt werden können.

Der Ansatz zur logischen Kapselung auf verschiedenen Ebenen ist auch für die Smarthome-Umgebung interessant. Visualisierungslösungen, die in dieser Umgebung entstehen, unterscheiden sich zwar in ihrer Form, nutzen jedoch dieselbe Domäne. Dadurch

könnte die Anforderung entstehen, diese Visualisierungen beliebig kombinierbar anzeigen zu lassen, ähnlich wie in einem Geoinformationssystem[1]. Als Beispiel dafür soll die Abbildung 4.2 dienen. Für die Architektur soll daher auch das Konzept einer Ebene implementiert werden, die verschiedene Inhalte kapselt und eine gemeinsame Schnittstelle bereitstellt.

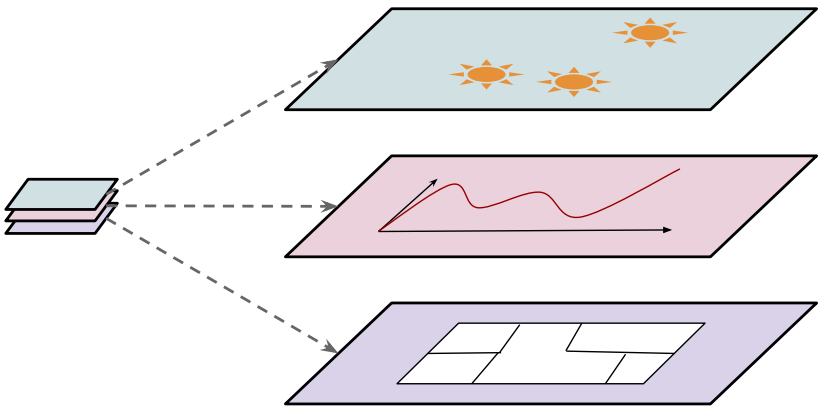

Abbildung 4.2: Visualisierung in Ebenen

4.2.2 Integration des Szenengraphen

Damit zukünftige Arbeiten als Teil des Smart Homes visualisiert werden können, soll der Szenengraph des CgResearch Frameworks in die Architektur integriert werden. Dieser bietet, wie bereits in 2.2 beschrieben, grundlegende Funktionen zur Verwaltung von renderbaren Daten. Durch die Einbindung von Klassen wie CgNode und CgApppplication kann auf die bereits implementierte Logik des Szenengraphen zugegriffen werden.

Die Integration des Szenengraphen wird umgesetzt, indem die Klasse SmartHome als Applikation des CgResearch Frameworks implementiert wird. Die Abbildung 4.3 zeigt, dass dies durch eine Vererbungsrelation geschieht. Somit hat das Smart Home auf den eigenen Szenengraphen Zugriff und kann dessen Wurzelknoten weitere Kinderknoten

[1] https://de.wikipedia.org/wiki/Geoinformationssystem

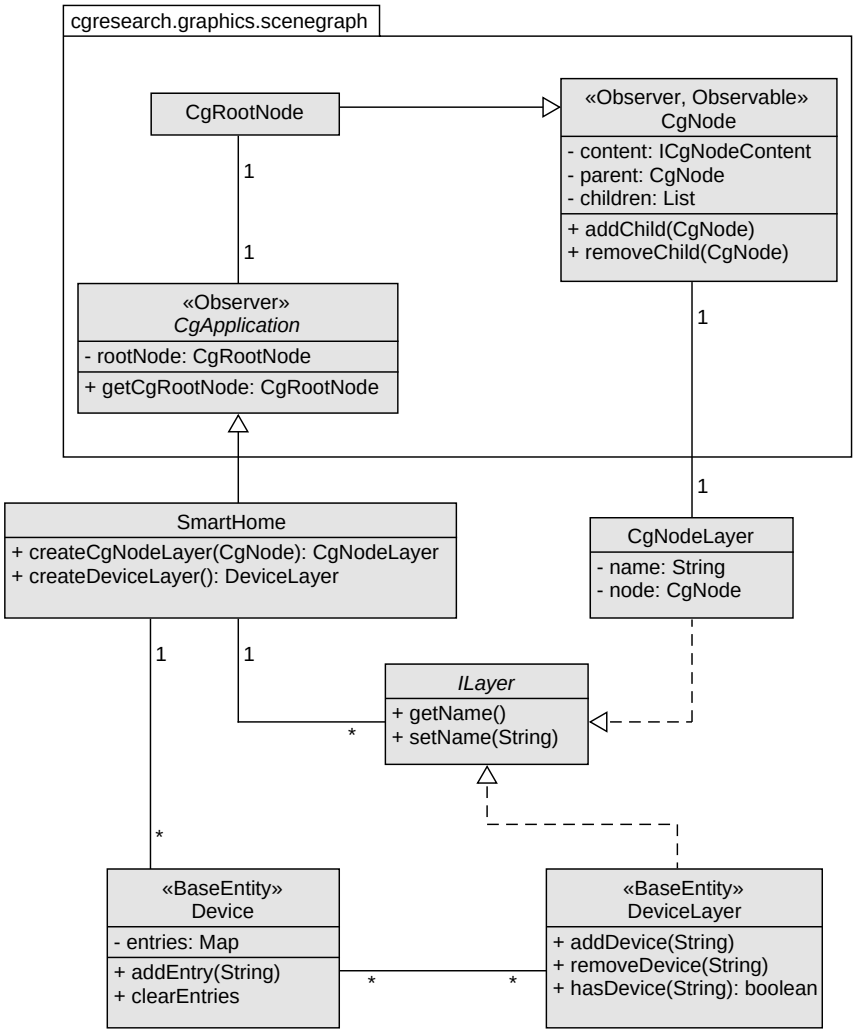

Abbildung 4.3: Komponenten der Domänenschicht

hinzufügen. Um den Ansatz der Inhaltsverwaltung mithilfe von Ebenen umzusetzen, wird jeder direkte Kinderknoten der Wurzel einem CgNodeLayer hinzugefügt, welche in dem Smart Home in einer Liste gespeichert werden. Auf solch einen Layer können dann Operationen der Schnittstelle ILayer ausgeführt werden. Für Operationen auf den Inhalt eines Knotens ist jedoch nicht die Ebene verantwortlich. Dies geschieht durch Applikationen, die ihre Inhalte an das Smart Home schicken. Das Smart Home stellt somit eine Plattform für die Ergebnisse gesammelter Arbeiten dar.

4.2.3 Verwalten von Geräten

Um verschiedene Geräte verwalten zu können, besitzt ein Smart Home eine Liste von Devices. Dabei handelt es sich um eine Klasse, die von der BaseEntity ableitet. Deren Definition erfolgt in 4.3. Ein Device repräsentiert ein übliches Gerät in einem Smart Home, etwa einen Sensor. Jedes Device besitzt eine Sammlung von Einträgen, die in einer HashMap gespeichert werden. Da es sich bei diesen Einträgen um Zeichenketten handelt, kann der Inhalt frei gewählt werden. Beim Hinzufügen eines Eintrags wird automatisch ein aktueller Zeitstempel als Schlüssel verwendet. Dieser ist bis auf die Millisekunde genau, sodass pro Sekunde mehrere Einträge gemacht werden können.

Beliebig viele Devices können zu einem DeviceLayer hinzugefügt werden. Dies wird über das Speichern von IDs gelöst. Dadurch können Devices beliebig unter einer Bezeichnung gruppiert werden, sodass zum Beispiel alle Geräte in der Küche gleichzeitig ansprechbar werden.

4.3 Datenhaltungsschicht

4.3.1 Datenmanagement

Da Smarthome-Umgebungen sich grundsätzlich durch das Vorhandensein verschiedener Geräte auszeichnen, soll das Persistieren von Daten vor allem auf diese ausgelegt

werden. Gesendete und empfangene Daten sind von unterschiedlicher Natur, Gerätestandards sind oft heterogen. Ein festes Datenschema könnte daher die souveräne Erweiterung des Smarthome-Systems einschränken.

Der agile Charakter von Smarthome-Projekten soll bei der Wahl der Datenbank bedacht werden. Gleichfalls soll dabei auch die Performanz von Lese- und Schreiboperationen eine wichtige Rolle spielen, denn bei einem Smart Home ist zu erwarten, dass die vorhandene physische Schicht fortlaufend wächst. Sensoren übertragen dabei zwar kompakte Inhalte, tun dies jedoch häufig. So kann es etwa vorkommen, dass Smarthome-Programme mehrere Messungen pro Sekunde auslösen. Gleichzeitig müssen Sensoren mit wenig Latenz ansprechbar sein. Für viele Projekte des Internet of Things, zu dem auch Smart Homes gehören, werden daher NoSQL-Datenbanken verwendet. So wird zum Beispiel für FuseViz die dokumentenorientierte Datenbank CouchDB [Vgl. GD12], für das Living Place MongoDB [Vgl. Vos11, S. 80] verwendet. In einem Performanztest, der in [Győ+15] durchgeführt wurde, schnitt MongoDB bei hohen Belastungen deutlich besser als MySQL ab. Dabei wurden unter gleichen Hardware-Bedingungen die Zugriffszeiten auf je eine Instanz der erwähnten Datenbanken gemessen. Das Erstellen von 10.000 Objekten, sowie das Auslesen, Verändern und Löschen von Daten konnte dabei mit einer MongoDB-Datenbank um ein Vielfaches schneller verarbeitet werden, als mit einer MySQL-Datenbank. Aus diesen Gründen soll die dokumentenorientierte Datenbank MongoDB auch für die Softwarearchitektur dieser Arbeit verwendet werden.

Objekte, die Teil des Szenengraphen sind oder solche besitzen, werden allerdings von der Persistierung ausgenommen. Grund hierfür ist, dass diese Objekte sehr komplex sind, was die automatische Umwandlung in Dokumente erschwert. Zusätzlich sollen die vorhandenen Implementierungen nicht durch Annotationen oder ähnliches erweitert werden, denn davon wäre das gesamte Framework betroffen. CgNodes und CgNodeLayer werden daher nicht in der Datenbank gespeichert. Da CgNodes jedoch dynamisch generiert (zum Beispiel aus Sensordaten) und dem Smart Home lediglich von außen übergeben werden, stellt dies keine Einschränkung dar.

4.3.2 MongoDB

MongoDB ist eine dokumentenorientierte Open-Source-Datenbank. Daten werden dabei im *BSON*[2]-Format in Dokumenten gespeichert und bestehen aus Key-Value-Paaren. Jedes Dokument repräsentiert ein Objekt in der Datenbank und besitzt eine eindeutige ID. Ein Dokument ist somit vergleichbar mit einer Zeile in einer SQL-Datenbank [Vgl. How+15, S. 33]. Das Pendant zu SQL-Tabellen stellen hier die Collections dar, welche eine Gruppe von Dokumenten verwalten. Das Beispiel 4.4 stellt den Inhalt eines MongoDB-Dokuments dar.

```
{
  "_id" : ObjectId("56fa98b600c3b319e0bcaefb"),
  "className" : "smarthomevis.architecture.data_access.Device",
  "name" : "Thermometer1",
  "entries" : {
    "29-01-2016 17:01:18:199" : "13"
  },
  "creationDate" : "15-01-2016 10:00:00:000",
  "lastChanged" : "29-01-2016 17:01:18:199",
  "version" : NumberLong(1)
}
```

Abbildung 4.4: Ein Dokument in MongoDB

Zum Auslesen von Daten, etwa über die Konsole, werden in MongoDB *Queries* verwendet. Dafür können eine Vielzahl an Funktionen zum Filtern und Sortieren von Dokumenten genutzt werden. Eine Query kann jedoch nur auf eine Collection zur Zeit ausgeführt werden.

Das Beispiel aus Abbildung 4.5 zeigt eine Query, bei der aus einer Sammlung von Büchern nur die Dokumente zurückgegeben werden sollen, die einen bestimmten Autoren enthalten. Die Rückgabe wird anschließend alphabetisch nach dem Titel sortiert, sowie auf die ersten fünf Dokumente limitiert.

[2] Binary JSON

```
db.books.find( { "Author" : "Terry Pratchett" } )
        .sort( { Title: 1 } ).limit( 5 )
```

Abbildung 4.5: Eine Query für MongoDB

Für die Nutzung von MongoDB stehen eine Reihe von Treibern für verschiedene Programmiersprachen zur Verfügung. Für diese Architektur wird dementsprechend der *Java MongoDB Driver* verwendet. Dieser enthält eine API für das Management von Daten in Java-Projekten, sowie eine BSON-Library [Vgl. Inc16].

Morphia

Bei *Morphia* handelt es sich um eine leichtgewichtige Java-Bibliothek, die Funktionalität zur Abbildung von Java-Objekten auf MongoDB-Dokumente bereitstellt, also um einen *Object Document Mapper*. Zur Umsetzung können eine Reihe von Annotationen für Klassen oder Felder genutzt werden. Die Tabelle 4.1 zeigt einige Beispiele für derartige Annotationen und ihre entsprechende Funktion.

Annotation	**Funktion**
@Entity	Markiert eine Klasse, deren Instanzen als Dokumente gespeichert werden soll. Die Klasse selbst repräsentiert dadurch eine Collection.
@Id	Markiert ein Feld, welches die Zeile *_id* in einem Dokument repräsentiert.
@Reference	Markiert ein Feld, das eine Referenz auf ein anderes Dokument speichert. Dabei kann entweder das komplette Objekt oder nur dessen ID gespeichert werden.
@Embedded	Markiert eine Klasse als Bestandteil einer anderen Klasse.

Tabelle 4.1: Annotationen in Morphia

So können auch die Beziehungen zwischen Objekten in der Datenbank unkompliziert abgebildet werden. Zusätzlich gibt es einige Annotationen, die das Verhalten von

Morphia konfigurierbar machen. Dadurch kann zum Beispiel bestimmt werden, ob Null-Felder gespeichert werden sollen oder nicht.

Für die Standardoperationen auf Datenbankobjekte, den sogenannten CRUD[3]-Operatoren, stellt Morphia sprechende Funktionen bereit. Sollen diese auf mehrere Objekte gleichzeitig ausgeführt werden, wird empfohlen die Datenbankobjekte vorher in einem Query-Objekt zu aggregieren. Für das Erstellen von Queries existiert eine API, welche die von MongoDB bereitgestellten Filter nutzt. So kann etwa mit dem Beispiel aus Abbildung 4.6 der Datenbank eine Liste von unterbezahlten Angestellten entnommen werden. Auf dieses Query-Objekt können anschließend Operationen ausgeführt werden, welche die entsprechenden Dokumente verändern.

```
1 underpaid = datastore.createQuery(Employee.class)
2                       .field("salary").lessThanOrEq(30000)
3                       .asList();
```

Abbildung 4.6: Eine Query für Morphia

Der Umgang mit der Datenbank und dem MongoDB Java Driver wird durch die Einbindung von Morphia erheblich erleichtert, da sprechendere Funktionen und Annotationen zur Verfügung gestellt werden. Gleichzeitig verringert sich die Komplexität der Software, wodurch sie leichter zu erweitern ist. Daher soll Morphia für diese Architektur verwendet werden.

4.3.3 Komponenten

Dieser Abschnitt beschreibt die Komponenten, die auf der Datenhaltungsschicht implementiert werden. Sie sind in 4.7 schematisch dargestellt.

[3] Create, Read, Update, Delete

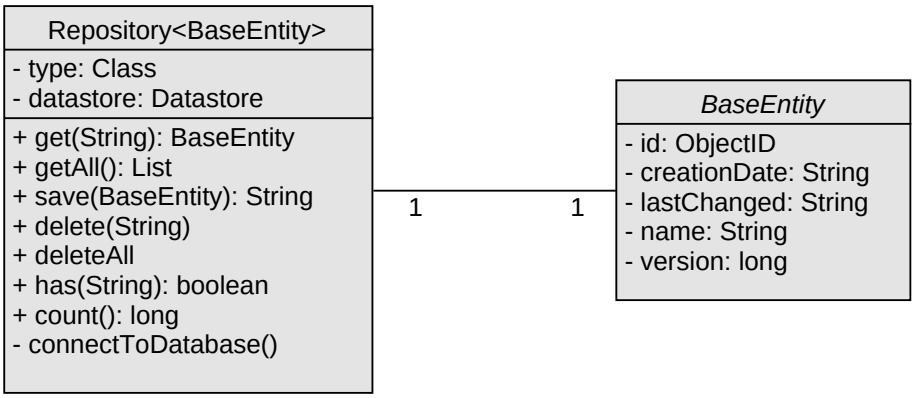

Abbildung 4.7: Komponenten der Datenhaltungsschicht

Repository

Bei dem Repository handelt es sich um eine generische Klasse, welche die Zugriffe auf eine Collection in der Datenbank kapselt. Es ist dem gleichnamigen Entwurfsmuster von [Fow03, S. 322] entsprechend implementiert. Fowler empfiehlt dabei, eine weitere Schicht zwischen dem Mapping von Daten und der Anwendungslogik zu implementieren. Auf dieser Schicht befindet sich das Repository. Durch die zusätzliche Abstraktion wird die Kohäsion der Software gestärkt und eine einheitliche Schnittstelle geschaffen.

Das Repository stellt die meistgenutzten Operationen auf Datenbankobjekte zur Verfügung. Dazu gehören neben den CRUD-Operatoren auch solche, die sich auf mehr als ein Objekt beziehen. Für die Umsetzung von Operationen auf eine Liste von Objekten werden Query-Objekte erstellt, welche dann von Morphia oder MongoDB verarbeitet werden können.

Für den Aufruf von Morphia speichert ausschließlich das Repository ein Datastore-Objekt. Damit stellt es den einzigen Punkt dar, an dem eine Verbindung zur Datenbank besteht. Dies ist von Vorteil, falls in Zukunft Änderungen an der Architektur gemacht werden sollen. Wenn zum Beispiel der Wunsch besteht, statt MongoDB eine andere Datenbank zu nutzen, muss nur die Implementation der vorhandenen Methoden in der Repository-Klasse verändert werden. Ein weiterer Vorteil besteht in der Generizität

der Repository-Klasse. Methoden, welche für alle Collections verfügbar sein sollen, müssen nur an einer Stelle hinzugefügt werden. Dies kann zum Nachteil werden, wenn eine Methode ausschließlich für einen Typ von Collection verfügbar sein soll. Dieser Fall wird jedoch wahrscheinlich nicht eintreten.

Eine Alternative Lösung zur Kapselung des Datenzugriffs stellt das Active Record-Pattern dar [Fow03, S. 160]. Im Gegensatz zum Repository, in dem Zugriffe in einer separaten Klasse gekapselt werden, implementiert ein Active Record-Objekt die entsprechenden Methoden selbst. Es enthält dabei gleichzeitig die Logik für das eigene Verhalten, sowie den Zugriff auf die zugehörige Zeile in der SQL-Datenbank, beziehungsweise auf das zugehörigen Dokument in der NoSQL-Datenbank. Ein Vorteil ist, dass Datenbank-Operationen direkt auf einem solchen Objekt aufgerufen werden können, ohne dass vorher ein Repository instanziiert werden muss. Ebenso können leichter Datenbankobjekt-spezifische Methoden implementiert werden. Der Nachteil befindet sich jedoch bei der Vermischung von Datenzugriffs- und Domänenlogik. Dies verletzt das Prinzip des *Separation of concerns* und erschwert die zukünftige Wartung des Codes, da für Änderungen an der Architektur oft an mehreren Stellen angesetzt werden muss. Das Active Record-Pattern soll daher nicht für diese Architektur verwendet werden.

Entitäten

Bei Entitäten handelt es sich um Objekte, die eine Identität und einen Lebenszyklus besitzen. Während ihre Identität stets eindeutig bleibt, können die Attribute des Objektes sich verändern.

Objekte des Smart Homes, die in der Datenbank als Dokumente gespeichert werden, leiten von der BaseEntity-Klasse ab. Diese abstrakte Basisklasse implementiert die benötigten Grundlagen für ein Datenbankobjekt. Dazu gehört ein Feld für die ID, mit der jedes Objekt identifiziert wird und ein Feld für die Versionierung durch Morphia. Zusätzlich werden Zeitstempel für die Erstellung und die letzte Änderung eines Dokuments automatisch gesetzt. Jedes Datenbankobjekt kann außerdem einen Namen besitzen. Die BaseEntity-Klasse stellt somit eine einfache Möglichkeit dar, die

Architektur um weitere Entitäten zu erweitern. Klassen, die von der Basisklasse erben, können umgehend persistiert werden.

Zum Stand der Arbeit werden auf der Domänenschicht bereits zwei konkrete Entitäten implementiert. Diese sind so gewählt, dass die Akteure eines Smart Homes abgebildet und die funktionalen Anforderungen an die Architektur umgesetzt werden können.

4.4 Präsentationsschicht

In 3.1 wurde das Projekt FuseViz vorgestellt, bei dem der Anspruch besteht, die implementierte Software von einer Reihe von Geräten aus steuern zu können. Zur Umsetzung wurde dafür eine REST-Schnittstelle genutzt. Dieser Anspruch soll auch an die Smarthome-Architektur gestellt werden, sodass in Zukunft eine Reihe von Applikationen entwickelt werden können, ohne die entsprechenden Schnittstellen anpassen zu müssen.

In [Wil07] fordert Wilde, dass die Konzepte von REST bei der Entwicklung von Anwendungen aus den Bereichen *Internet of Things* und *Ubiquitous Computing* verwendet werden sollten, sodass diese nahtlos in das Web integriert werden. Ressourcenorientierte Architekturen, wie sie bereits in 2.4 beschrieben wurden, sind daher besonders gut für Smarthome-Systeme geeignet. Ein großer Vorteil dieser Architekturen ist ihre Skalierbarkeit, die als Resultat der zustandslosen Kommunikation entsteht. Durch die Beschränkung der Kommunikation auf HTTP können Services leichter von Clients genutzt werden, da die Schnittstellen klar definiert sind. Während die HTTP-Methoden unverändert bleiben, müssen lediglich die über den Service verfügbaren Ressourcen und ihre Form kommuniziert werden. Da dies auch als Teil von Repräsentationen gelöst werden kann, zum Beispiel in Form von klickbaren Links, ist die Navigation durch RESTful Services oft sehr einfach. Infolge dessen kann ein Service über eine einzige Schnittstelle mit mehreren Applikationen kommunizieren, welche gleichzeitig auf dieselbe Menge von Ressourcen zugreifen. Dies ist in Abbildung 4.8 verdeutlicht.

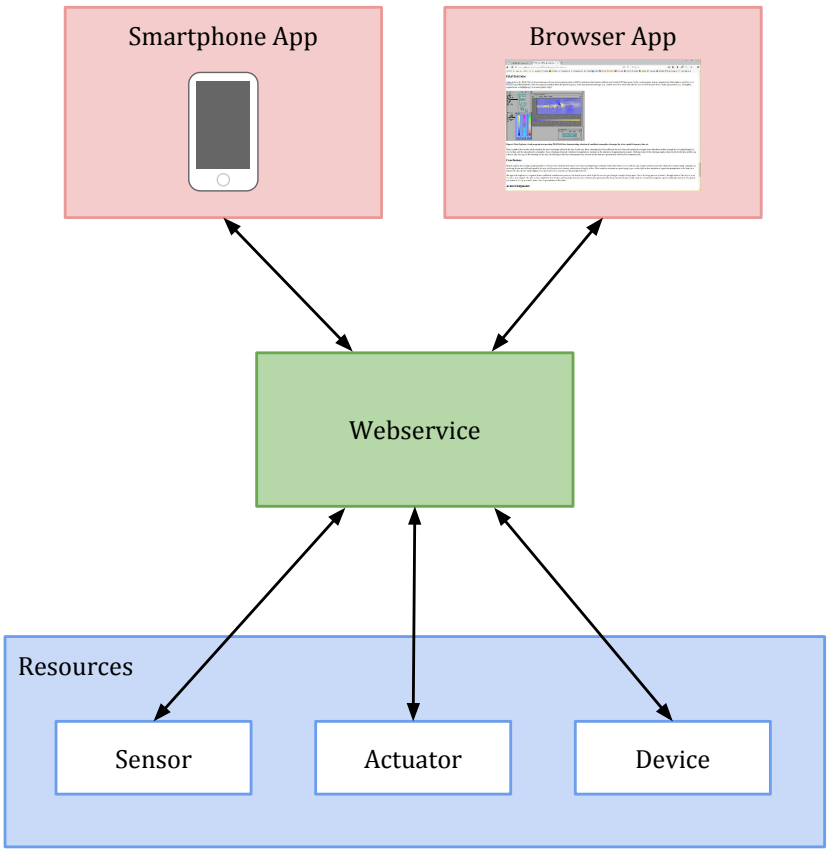

Abbildung 4.8: Ein RESTful Webservice

Für die Smarthome-Umgebung soll aus diesen Gründen ebenfalls eine REST-Schnittstelle implementiert werden, welche den Zustand der vorhandenen Geräte verfügbar macht.

4.4.1 Restlet

Um die Implementation der Schnittstelle zu erleichtern, soll ein Framework verwendet werden. Beispiele für Frameworks, welche die Implementation von RESTful Java-Anwendungen unterstützen, sind *Jersey*, *Restlet* oder *JAX-RS*. Da Restlet schon an einigen Stellen des CgResearch Frameworks verwendet wurde und somit bereits Teil des Projekts ist, soll es auch für das Smart Home verwendet werden.

Bei Restlet handelt es sich um ein Open-Source Framework, welches die Implementierung leichtgewichtiger REST-Konzepte in Java unterstützt. Dafür steht eine API zur Verfügung, mit der sowohl Clients als auch Server implementiert werden können. Das Ziel von Restlet ist, bei der Umsetzung so nah wie möglich an den von Fielding beschriebenen Konzepten zu bleiben [Vgl. RR07, S. 344], sodass es sich bei dem Framework um eine ressourcenorientierte Architektur handelt. Daher wird auch hier das Abrufen und Verändern von Ressourcen mithilfe von HTTP-Methoden und die Adressierung durch URIs genutzt.

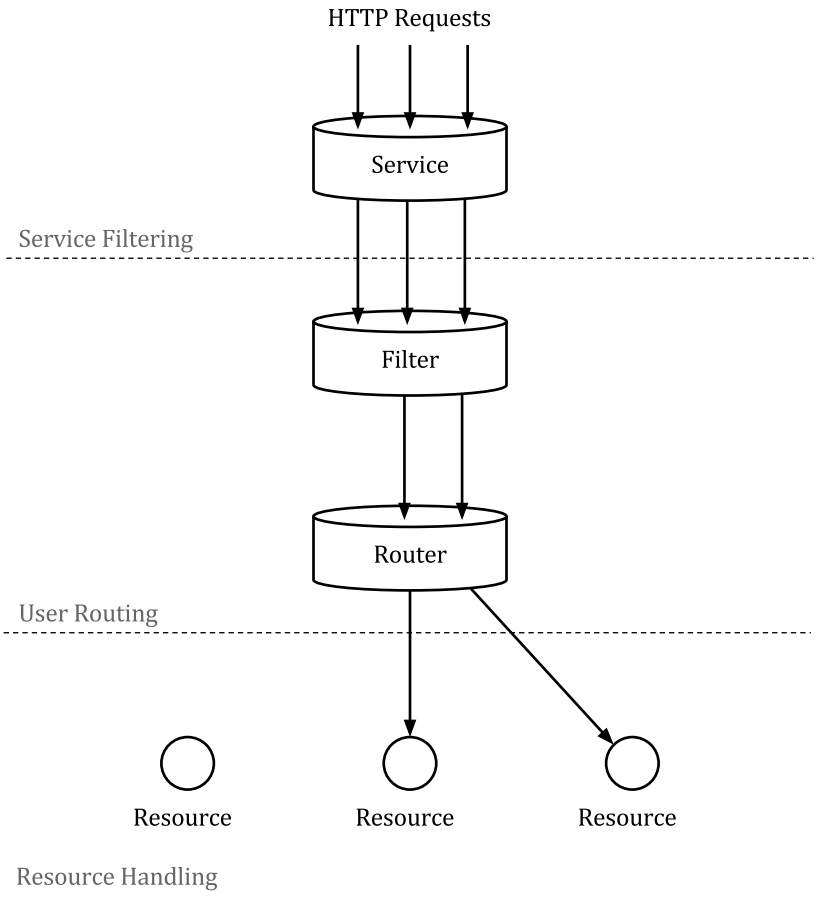

Abbildung 4.9: Request Handling in Restlet

Im Folgenden wird grob die Architektur des Request-Handling von Restlet beschrieben. Jeder eingehende Request durchläuft dabei drei Schichten [Vgl. LTB13, S. 15]. Diese sind in Abbildung 4.9 bildlich dargestellt.

1. **Service Filtering:** Die API bietet eine Reihe von Services, welche auf der ersten Ebene in Effekt treten. Dazu gehören zum Beispiel Services zum Decodieren von Requests oder der Logging-Service. Zusätzlich ist es möglich, eigene Services zu implementieren.

2. **User Routing:** Auf dieser Ebene können Filter konfiguriert werden, die auf jeden Request angewandt werden. Dazu gehören zum Beispiel das Prüfen von Berechtigungen und Inhalten oder das Blockieren bestimmter Absender. Daher kann es passieren, dass Requests auf dieser Ebene abgelehnt werden und der Server mit einem Fehlercode antwortet. Durchläuft ein Request erfolgreich einen Filter, greift auf derselben Ebene das Routing, indem URI-Muster auf Ressourcen abgebildet werden. Die Konfiguration dieser Routen übernimmt der Entwickler. Ein Request kann nur dann erfolgreich an eine Ressource übergeben werden, wenn eine Route dafür definiert ist.

3. **Resource Handling:** Schließlich werden Requests auf der dritten Ebene an Ressourcen zugestellt.

Die Gestaltung der Ressourcen ist vollständig Aufgabe des Entwicklers. Dabei können Methoden mit Annotationen versehen werden, die den HTTP-Methoden entsprechen. Somit kann für jede HTTP-Methode das gewünschte Verhalten der Ressource implementiert werden. Wurde ein Request erfolgreich verarbeitet, wird eine Antwort mit entsprechender Repräsentation an den Absender geschickt. Die Zustellung von Antworten wird durch das Restlet Framework erledigt.

4.4.2 Komponenten

In diesem Abschnitt werden die Komponenten beschrieben, die mithilfe von Restlet auf der Präsentationsschicht implementiert werden.

Repräsentationen

Für den Anfang werden über die Schnittstelle nur Repräsentationen von Ressourcen in Form von JSON-Zeichenketten ausgetauscht. Für die Umwandlung zwischen Zeichenketten und Objekten wird die Gson Library verwendet[4]. Die Repräsentationen können jedoch jederzeit um weitere Formate ergänzt werden.

Ressourcen

Im Folgenden wird verdeutlicht, wie die Ressourcen der REST-Schnittstelle gestaltet sind. Jede Ressource wird dabei in genau einer Tabelle dargestellt.

Device

URI	/device/{id}
Beschreibung	Der Zustand eines bestimmten Gerätes
Methoden	• GET liefert die Repräsentation eines bestimmten Gerätes. • PUT akzeptiert eine Repräsentation und speichert den Zustand des enthaltene Gerät unter der angegebenen ID. • DELETE löscht ein bestimmtes Gerät.

[4] https://github.com/google/gson

Devices

URI	/devices
Beschreibung	Der Zustand einer Liste von Geräten
Methoden	• GET liefert eine Repräsentation mit allen existierenden Geräten. • POST akzeptiert eine Repräsentation und speichert den Zustand des Gerätes unter der enthaltenen ID. Falls diese ID nicht existiert oder keine enthalten ist, wird ein neues Gerät erstellt. • DELETE löscht alle existierenden Geräte.

Device-Layer

URI	/layers/{id}
Beschreibung	Der Zustand einer bestimmten Ebene, die Geräte-IDs verwalten
Methoden	• GET liefert die Repräsentation einer bestimmten Ebene. • PUT akzeptiert eine Repräsentation und speichert den Zustand der enthaltenen Ebene unter der angegebenen ID. • DELETE löscht eine bestimmte Ebene.

Device-Layers

URI	/layers
Beschreibung	Der Zustand einer Liste von Ebenen, die Geräte-IDs verwalten
Methoden	• GET liefert eine Repräsentation mit allen existierenden Ebenen. • POST akzeptiert eine Repräsentation und speichert den Zustand der Ebene unter der enthaltenen ID. Falls diese ID nicht existiert oder keine enthalten ist, wird eine neue Ebene erstellt. • DELETE löscht alle existierenden Ebenen.

4.5 Beispielszenarien

Im Folgenden werden zwei Szenarien beschrieben, bei denen es sich um Beispiele für die typische Interaktion mit dem Smarthome-System handelt. Die Reihenfolge, in der die entsprechenden Komponenten aufgerufen werden, wird mithilfe von Diagrammen dargestellt.

4.5.1 Aufruf über die REST-Schnittstelle

In diesem Beispiel, dargestellt in Abbildung 4.10, trifft bei dem Server des Smart Homes ein HTTP Request ein, der eine Repräsentation eines bestimmten Gerätes fordert. Durch die URI kann der Server anhand der konfigurierten Routen die entsprechende Ressource aufrufen und die Informationen weiterleiten. Die Device-Ressource kann

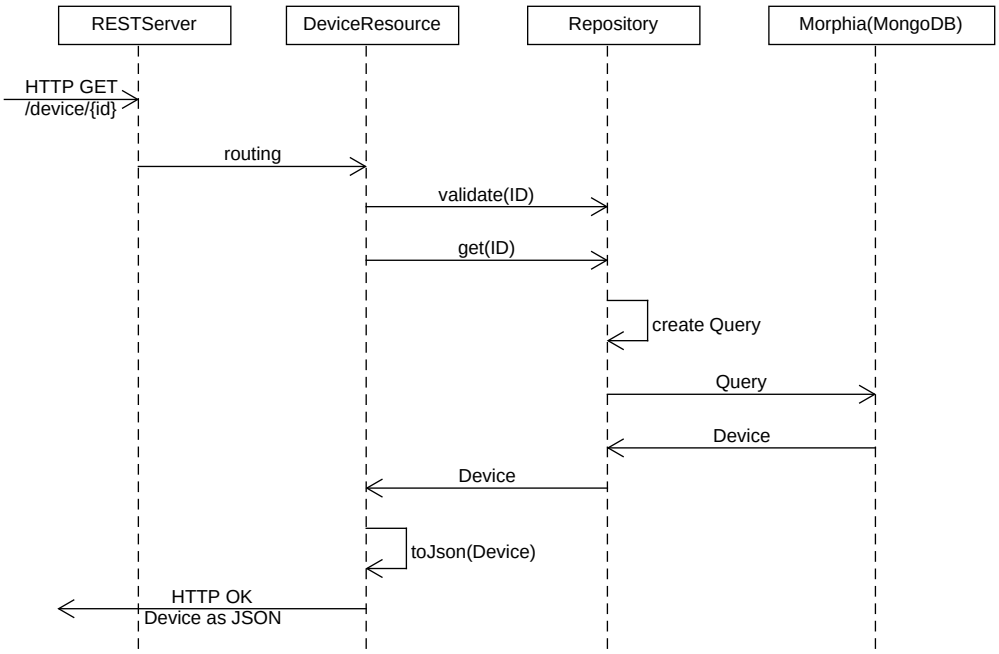

Abbildung 4.10: Ein Gerät wird über die REST-Schnittstelle hinzugefügt

auf ein GET reagieren und ruft ein Repository für Device-Objekte auf. Die ID des Zielobjektes wird dabei der URI entnommen.

In dem Repository wird mithilfe von Morphia-Operationen eine Query erstellt, die ein spezifisches Dokument in der Datenbank erfragt. Der Austausch mit MongoDB wird von Morphia übernommen. Anschließend wird das extrahierte Objekt über das Repository an die Ressource zurückgegeben. Hier wird es in eine JSON-Zeichenkette umgewandelt. Schließlich verschickt die Ressource eine HTTP Response, die eine JSON-Repräsentation des Gerätes enthält.

An dieser Stelle soll angemerkt werden, dass hier das Prinzip der strikten Drei-Schichten-Architektur aus [Fow03] verletzt wird. Die Ressource befindet sich auf der Präsentationsschicht, greift jedoch auf ein Repository auf der Datenhaltungsschicht zu. Streng genommen müsste die Ressource eine weitere Abstraktion auf der Domänenschicht aufrufen. Da zum Stand der Arbeit jedoch nur bewusst wenig Logik in der

Domänenschicht implementiert ist, würde eine weitere Abstraktion nur aus Gettern und Settern bestehen, was die Software unnötig komplexer macht. Die Architektur soll in erster Linie die Entwicklung von weiteren Forschungsarbeiten unterstützen und daher nicht zu starre Strukturen vorgeben. Daher soll an dieser Stelle eine Lockerung des Schichtenprinzips erfolgen, wie sie auch in [Bus+96, S. 45-46] beschrieben wird. Wird die Domänenlogik der Architektur in Zukunft komplexer, sollten Ressourcen jedoch nicht mehr direkt auf Repositories zugreifen, sondern zum Beispiel eine Facade nutzen[5].

4.5.2 Erstellen von Ebenen

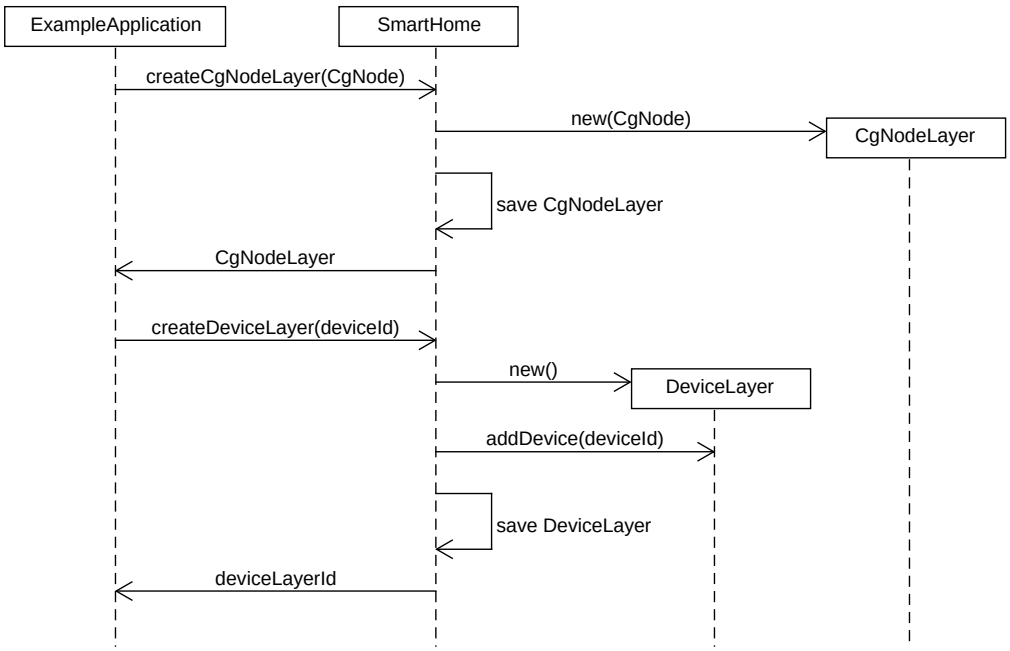

Abbildung 4.11: Zwei Ebenen werden dem Smart Home hinzugefügt

In diesem Beispiel, dargestellt in 4.11, werden die Ergebnisse einer Applikation in das Smart Homes aufgenommen. Der Aufruf geschieht dabei über eine Applikation, die

[5]Remote-Facade: http://martinfowler.com/eaaCatalog/remoteFacade.html

Teil des CgResearch-Frameworks ist. Zunächst wird ein Knoten des Szenengraphen übergeben, der zu visualisierende Daten enthält. Für diesen erstellt das Smart Home einen zugehörigen Layer und übergibt ihn an die aufrufende Applikation. Dabei kann es sich zum Beispiel um eine grafische Oberfläche handeln, die verschiedene Visualisierungsmöglichkeiten miteinander kombinierbar macht.

Anschließend soll ein weiterer Layer hinzugefügt werden, der eine Reihe von Geräten gruppiert. Da die Anzahl der enthaltenen Geräte beliebig ist, muss an dieser Stelle kein Argument übergeben werden. Der DeviceLayer wird erstellt und über ein Repository in der Datenbank gespeichert. Schließlich wird der aufrufenden Applikation die ID des DeviceLayers übergeben.

4.6 Zusammenfassung

In diesem Kapitel wird das Konzept für die Software-Architektur zur Visualisierung in Smarthome-Umgebungen beschrieben. Die Architektur wird dabei als Drei-Schichten-Architektur implementiert. Auf der Präsentationsschicht befindet sich eine REST-Schnittstelle, über die die Geräte eines Smart Homes verwaltet werden können. Diese wird mithilfe von Restlet implementiert. Die Domänenschicht implementiert Logik zur Visualisierung von Knoten des Szenengraphen, die dem Smart Home durch Applikationen übergeben werden. Jeder Knoten, sowie jede Gruppe von Geräten, wird dabei durch eine Ebene gekapselt. Auf der Datenhaltungsschicht wird durch Morphia das Mapping von Objekten auf Dokumente in der Datenbank MongoDB umgesetzt. Über ein Repository werden CRUD-Operatoren für Entitäten zur Verfügung gestellt.

5 Bewertung

In diesem Kapitel wird die vorgeschlagene Architektur bewertet. Dabei wird reflektiert, inwieweit die in Kapitel 3 gestellten Anforderungen erfüllt werden. Zusätzlich soll ein Test Aufschluss über die Performanz der Architektur geben. Schließlich wird an Beispielen erläutert, wie Applikationen auf zwei verschiedene Arten in das Smarthome-System integriert werden können.

5.1 Erfüllung der Anforderungen

5.1.1 Funktionale Anforderungen

Im Folgenden wird beschrieben, in welchem Umfang die einzelnen Szenarien mit der entwickelten Architektur umgesetzt werden können.

Szenario I: Geräte können per HTTP über die REST-Schnittstelle registriert werden. Zusätzlich ist es für Applikationen des CgResearch-Framework möglich, Geräte direkt über die Schnittstellen der Architekturschichten hinzuzufügen.

Szenario II: Geräte können Einträge in beliebigen Formaten speichern, die über einen Zeitstempel referenziert werden. Automatisierung von Geräten ist nicht Teil dieser Arbeit und wird daher auch nicht umgesetzt.

Szenario III: Geräte können logisch gruppiert werden, indem sie einer bestimmten Ebene hinzugefügt werden. Diese Ebenen können ebenfalls über die REST-Schnittstelle erstellt und modifiziert werden.

Szenario IV: Visualisierungsmöglichkeiten sind nicht Teil der Arbeit und werden daher auch nicht umgesetzt. Jedoch können dem Smart Home von Applikationen des CgResearch-Frameworks Knoten des Szenengraphen übergeben werden, welche renderbare Daten enthalten.

Entsprechend des Anspruchs werden die funktionalen Anforderungen nicht komplett erfüllt, da dies über den Rahmen der Arbeit hinausgehen würde. Die grundlegende Struktur, auf der weitere Projekte im Bereich des Smart Homes aufbauen können, ist jedoch implementiert.

5.1.2 Nichtfunktionale Anforderungen

Im Folgenden wird beschrieben, in welchem Umfang die Anforderungen an die Qualität der Software umgesetzt werden.

Erweiterbarkeit: Die implementierte REST-Schnittstelle ermöglicht, dass jedes HTTP-fähige Gerät das Smarthome-System nutzen kann. Durch die Implementation eines generischen Repository, sowie einer grundlegenden Entitätsklasse, können weitere Objekte der Architektur hinzugefügt und persistiert werden. Durch die Nutzung von NoSQL mit MongoDB ist es ebenso möglich, die Smarthome-Architektur komplett zu umgehen, gleichzeitig jedoch dieselbe Datenbank zu nutzen. Beliebig viele Knoten können dem Szenengraphen des Smart Homes hinzugefügt werden. Die Erweiterbarkeit der Software ist somit erfüllt.

Robustheit: Die Erkennung und Verarbeitung von Fehlern wird bereits an vielen Stellen von den genutzten Technologien erfüllt, etwa durch das Kommunizieren von HTTP-Statuscodes durch Restlet. Das schemalose Persistieren von Daten ist ebenfalls tolerant gegenüber Form und Inhalt von Einträgen. Ein in 5.2 durchgeführter Stresstest soll zeigen, dass das System auch gegenüber einer großen Menge von Anfragen robust ist. Bei der Umsetzung des Konzeptes wurden schichtenübergreifende Tests implementiert. Es sei jedoch angemerkt, dass eine hundertprozentige Robustheit nicht zu erreichen ist.

Wartbarkeit: Die Wartbarkeit wird erfüllt, indem bekannte Muster implementiert werden. Dazu gehört der Aufbau der Architektur in Schichten, sowie die Implementation der REST-Schnittstelle. Dadurch ist es möglich einzelne Module oder Technologien bei Bedarf auszutauschen, ohne die Funktionalität des Systems zu beeinträchtigen. Soll zum Beispiel statt MongoDB eine andere Datenbank verwendet werden, so muss lediglich die Schnittstelle des Repository entsprechend implementiert und das Mapping der Objektfelder der Basisentität angepasst werden. Ähnlich kann auch mit den weiteren Schichten der Architektur verfahren werden. Da die Logik des Systems auf der Domänenschicht gekapselt ist, lässt sich diese beliebig verändern, ohne das Datenmodell zu beeinträchtigen.

Performanz: Mit Hinblick auf die Performanz wird MongoDB zur Persistierung genutzt. Andere Technologien wurden jedoch nicht auf ihre Performanz untersucht. Der in 5.2 durchgeführte Test soll daher weitere Aufschlüsse geben.

Auf der Erfüllung der nichtfunktionalen Anforderungen lag während der Entwicklung der Architektur ein großer Fokus. Technologien und Softwaremuster wurden unter Berücksichtigung ihrer Flexibilität ausgewählt und umgesetzt. Das System bleibt somit offen für Erweiterungen auf jeder Schicht.

Zusätzlich wurde der Szenengraph des CgResearch-Frameworks in die Architektur integriert. Das Verwalten von Knoten geschieht mithilfe von Ebenen, welche dem Smart Home beliebig hinzugefügt werden können.

5.2 Performanztest

Mit diesem Test soll untersucht werden, wie sich das Smarthome-System bei hoher Belastung verhält. Dafür werden zehn Durchläufe eines Testszenarios durchgeführt. Über die REST-Schnittstelle per HTTP POST und über ein Repository werden pro Durchlauf jeweils 1000 Devices erstellt, benannt und gespeichert. Die benötigte Zeit wird anschließend notiert.

	benötigte Zeit in ms	
Durchlauf	**über Restlet**	**über Repository**
1	6.415	501
2	6.107	486
3	6.998	484
4	6.890	493
5	6.908	581
6	7.276	548
7	6.291	514
8	6.583	474
9	6.441	502
10	6.407	485
Durchschnitt	6.632	507

Tabelle 5.1: Erstellen von 1000 Devices

Die Tabelle 5.1 und die daraus resultierende Abbildung 5.1 zeigen, dass das Hinzufügen von Geräten über das Repository deutlich weniger Zeit benötigt, als über die REST-Schnittstelle. Grund hierfür ist die Verwendung des Restlet Frameworks, bei dem der Fokus nicht auf Performanz, sondern auf der Umsetzung einer robusten ROA liegt. Unklar ist dabei leider, wie viele Ressourcenobjekte durch Restlet erstellt werden.

Bei einem weiteren Test wurde die Anzahl der erstellten Devices auf 10.000 erhöht. Dabei lag die durchschnittlich benötigte Zeit bei der Erstellung über Restlet bei etwa 65.000 ms, erhöhte sich also ebenso um etwa das Zehnfache. Im Gegensatz dazu wurden beim Erstellen über das Repository im Durchschnitt etwa 2.150 ms benötigt, die benötigte Zeit erhöhte sich also um etwa das Fünffache. Für Applikationen, die besonders viele Zugriffe auf die Datenbank benötigen, sollte daher die REST-Schnittstelle nicht verwendet werden.

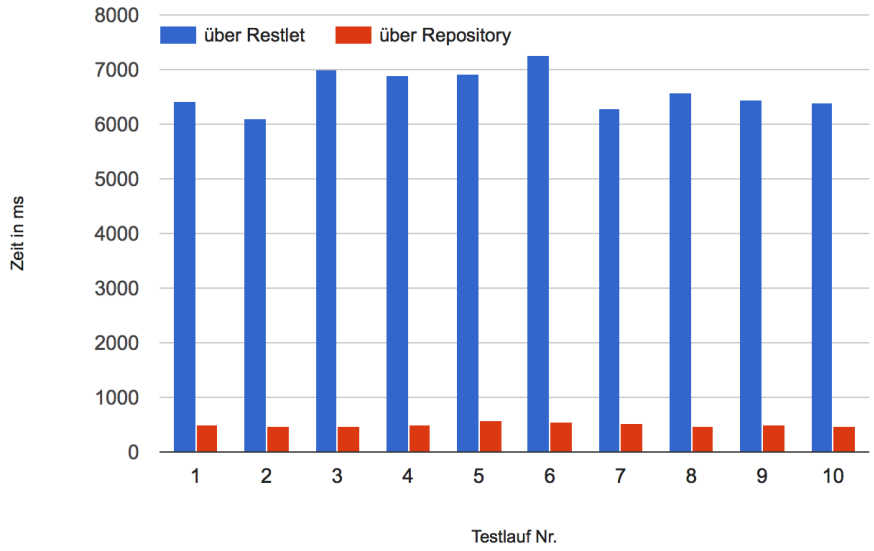

Abbildung 5.1: Hinzufügen von 1000 Devices

5.3 Anwendungsbeispiele

5.3.1 Zugriff über die REST-Schnittstelle

In diesem Beispiel wird gezeigt, wie über die REST-Schnittstelle auf das Smarthome-System zugegriffen werden kann. Dafür werden über curl eine Reihe von HTTP-Requests an den Server geschickt. Diese sind in Abbildung 5.2 dargestellt. Zunächst wird über ein POST ein Device in JSON-Notation an den Server verschickt. Dieser Antwortet mit der Id, die dem Device beim Eintrag in die Datenbank gegeben wurde. Mithilfe dieser Id kann anschließend über ein GET der Inhalt des Device angefordert werden. Dabei werden zusätzlich auch die vergebenen Zeitstempel für die Erstellung und die letzte Änderung sichtbar. Anschließend werden alle existierenden Devices über ein DELETE gelöscht. Ein erneutes GET zeigt, dass keine Devices mehr in der Datenbank vorhanden sind.

```
➜  ~ curl -X POST http://localhost:8183/smarthome/devices/ -H "Content-Type: application/json" \
-d '{"name" : "MySensor", "entries" : { "29-01-2016 17:01:18:199" : "13 Grad" } }'
570ca0ac1a3f0103522279c1%
➜  ~ curl -X GET http://localhost:8183/smarthome/devices/570ca0ac1a3f0103522279c1
{"entries":{"29-01-2016 17:01:18:199":"13 Grad"},"id":"570ca0ac1a3f0103522279c1","creationDate":"
12-04-2016 09:15:56:856","lastChanged":"12-04-2016 09:15:56:856","name":"MySensor"}%
➜  ~ curl -X DELETE http://localhost:8183/smarthome/devices/
➜  ~ curl -X GET http://localhost:8183/smarthome/devices/
[]%
➜  ~
```

Abbildung 5.2: Ansprechen der REST-Schnittstelle mit curl

5.3.2 Aufruf des Szenengraph

In diesem Beispiel wird gezeigt, wie Visualisierungslösungen für das Smart Home durch den Szenengraphen dargestellt werden. Eine Applikation des CgResearch Frameworks kann renderbare Daten in einer CgNode übergeben. Dieser Knoten wird in einem Layer verwaltet und an den Wurzelknoten des Szenengraphen des Smart Homes. Die Darstellung und Steuerung von Layern über die grafische Oberfläche ist zum aktuellen Zeitpunkt noch nicht möglich. Die Abbildung 5.3 zeigt das Ergebnis einer Forschungsarbeit zur Generierung von dreidimensionalen Grundrissmodellen [Vgl. Opi].

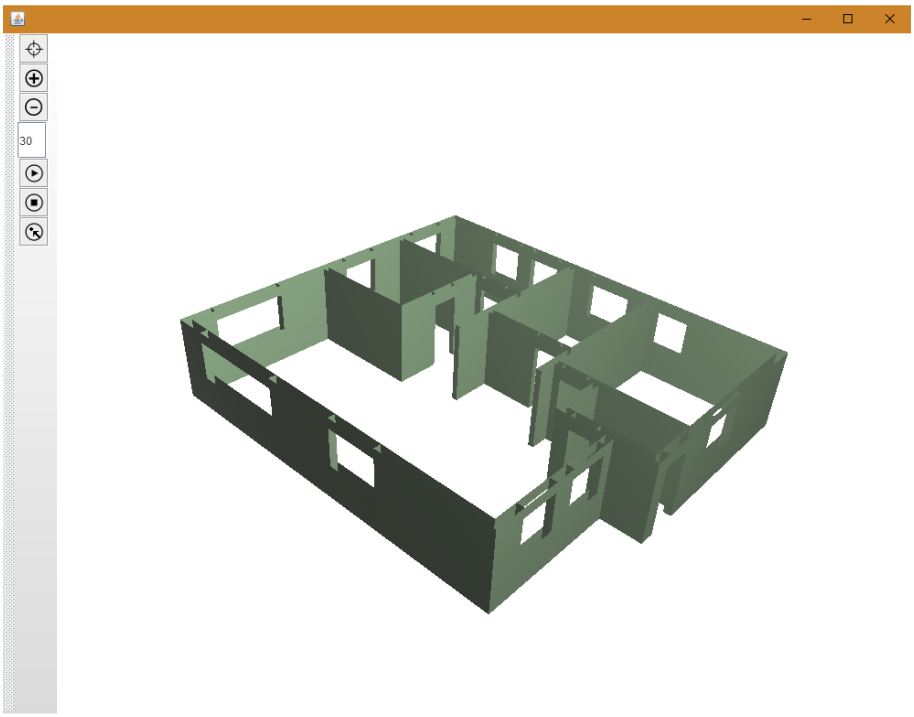

Abbildung 5.3: Darstellung einer Visualisierungslösung

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Konzept für eine Softwarearchitektur zur Visualisierung von Smarthome-Umgebungen entwickelt und umgesetzt. Dabei wurde Wert auf die Möglichkeiten zur Erweiterung der Architektur, sowie auf die flexible Integration von unterschiedlichen Smarthome-Applikationen gelegt. Mithilfe eines Szenengraphen können Visualisierungslösungen, die Teil des CgResearch-Frameworks sind, in Form von Ebenen über das Smart Home verwaltet werden. Durch die Kommunikation über eine REST-Schnittstelle ist es möglich, Geräte und Ebenen zu erstellen und zu modifizieren.

Das vorgestellte Konzept für die Infrastruktur der Architektur eignet sich durch seine Flexibilität und Offenheit für die Anforderungen, die an eine Smarthome-Plattform gestellt werden. Es bietet Grundlagen für die Automatisierung von Vorgängen, sowie das Verwalten diverser Visualisierungslösungen über einen zentralen Punkt. Durch ihre Erweiterbarkeit unterstützt die Architektur die nachfolgenden Arbeiten am Smarthome-Projekt.

6.2 Ausblick

Während der Entwicklung der Architektur wurden einige Aspekte, wie etwa in 3.3 beschrieben, bewusst vernachlässigt. Es existieren daher viele Möglichkeiten zur Erweiterung und Verbesserung des Systems. So sind vor allem die Themen Datenschutz

und IT-Sicherheit auch für Smart Homes von Interesse, da in diesem Bereich viele persönliche Daten gesammelt werden. Gleichzeitig macht die Anbindung an das Internet das Smarthome-System angreifbar.

Für die Implementation der Schnittstellen wurde bei der Entwicklung von einem hypothetischen Sensornetzwerk ausgegangen, sowie von hypothetischen Clients, die auf den RESTful Webservice zugreifen, da diese zum Stand der Arbeit noch nicht existieren. Die Anbindung an echte Geräte und Clients wird Ausblicke auf neue Anforderungen erbringen, die eine Smarthome-Plattform erfüllen muss. Die implementierte Logik der jeweiligen Schichten sollte daher kontinuierlich angepasst und ergänzt werden.

Weiteres Verbesserungspotential liegt in der Kommunikation mit den Geräten des Smart Homes. Die Beobachtungen zum Projekt Living Place haben gezeigt, dass Systeme mit wachsenden Sensornetzwerken von einem Austausch über Messaging-Technologien profitieren. Da das Smarthome-Projekt jedoch noch am Anfang steht, werden Arbeiten in dieser Richtung, wie zum Beispiel die Implementation einer Middleware, erst in Zukunft relevant.

Schließlich bietet das Projekt FuseViz [GD12] eine interessante Idee für Visualisierungslösungen. Hier werden Sensordaten direkt nach der Persistierung zu verschiedenen Streams fusioniert. So können Applikationen direkt auf Daten zugreifen, die in einem gemeinsamen Kontext entstehen.

Literaturverzeichnis

[Bit14] Bitkom. *Vor dem Boom - Marktaussichten für Smart Home. Fokusgruppe Connected Home des Nationalen IT-Gipfels*. 2014. URL: https://www.bitkom.org/Bitkom/Publikationen/Marktaussichten-fuer-Smart-Home.html (besucht am 09. 03. 2016).

[Bus+96] Frank Buschmann u. a. *Pattern-oriented Software Architecture. A System of Patterns*. Bd. 1. Wiley, 1996. ISBN: 0-471-95869-7.

[Coo+12] Diane J. Cook u. a. „CASAS: A Smart Home in a Box“. In: *Computer* 46 (7 2012), S. 62–69. ISSN: 0018-9162. DOI: 10.1109/MC.2012.328.

[Eic14] Tobias Eichler. „Agentenbasierte Middleware zur Entwicklerunterstützung in einem Smart-Home-Labor“. Masterarbeit. Hochschule für Angewandte Wissenschaften Hamburg, 2014.

[Ell+11] Jens Ellenberg u. a. „An Environment for Context-Aware Applications in Smart Homes“. In: *2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. 2011. URL: http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/papers/IPIN2011.pdf (besucht am 09. 03. 2016).

[Fie00] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Dissertation. University of California, Irvine, 2000.

[Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003. ISBN: 0-321-12742-0.

[GD12] Giacomo Ghidini und Sajal K. Das. „Fuseviz: A framework for web-based data fusion and visualization in smart environments“. In: *2012 IEEE 9th International Conference on Mobile Adhoc and Sensor Systems (MASS)*. Las Vegas, NV: IEEE, Okt. 2012, S. 468–472. ISBN: 978-1-4673-2433-5. DOI: 10.1109/MASS.2012.6502550.

[Gru16] Rolf Grupp. „Smart Home Monitor 2016“. In: *cci Dialog GmbH* (2016). URL: http://www.cci-dialog.de/branchenticker/2016/kw08/01/smart_home_monitor_2016.html (besucht am 14. 03. 2016).

[Győ+15] Cornelia Győrödi u. a. „A comparative study: MongoDB vs. MySQL“. In: *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. Oradea: IEEE, Juni 2015, S. 1–6. ISBN: 978-1-4799-7649-2. DOI: 10.1109/EMES.2015.7158433.

[How+15] David Hows u. a. *The Definite Guide to MongoDB. A complete guide to dealing with Big Data using MongoDB*. Apress, 2015. ISBN: 978-1-4842-1183-0.

[Hün13] Christian Hüning. „Konzeption und Entwurf einer Architektur zum Einsatz von Multi-Agenten-Simulationen in der ökologischen Systemmodellierung“. Bachelorarbeit. Hochschule für Angewandte Wissenschaften Hamburg, 2013.

[Inc16] MongoDB Inc. *Java MongoDB Driver*. 2016. URL: https://docs.mongodb.org/ecosystem/drivers/java/ (besucht am 05. 04. 2016).

[Jen15] Philipp Jenke. *Computergrafik Framework - Architekturdokumentation, Komponenten und Konzepte*. 2015. URL: https://github.com/pjenke/computergraphics/tree/master/doc/documentation (besucht am 06. 04. 2016).

[LTB13] Jérôme Louvel, Thierry Templier und Thierry Boileau. *Restlet in Action. Developing RESTful web APIs in Java*. Manning Publications Co., 2013. ISBN: 978-1-935-18234-4.

[Men16] Rainald Menge-Sonnentag. „Internet der Dinge: Das Open Internet Consortium wird zur Open Connectivity Foundation“. In: *heise online* (2016). URL: http://www.heise.de/developer/meldung/Internet-der-Dinge-Das-Open-Internet-Consortium-wird-zur-Open-Connectivity-Foundation-3113448.html (besucht am 14. 03. 2016).

[Opi] Leonard Opitz. „Generierung von dreidimensionalen Darstellungen von Gebäuden aus Grundrissen“. Bachelorarbeit. Hochschule für Angewandte Wissenschaften Hamburg. Noch nicht veröffentlicht.

[RR07] Leonard Richardson und Sam Ruby. *RESTful Web Services*. O’Reilly Media, Inc., 2007. ISBN: 978-0-596-52926-0.

[TC11] Brian L. Thomas und Aaron S. Crandall. „A Demonstration of PyViz, a Flexible Smart Home Visualization Tool“. In: *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. Seattle, WA: IEEE, März 2011, S. 304–306. DOI: 10.1109/PERCOMW.2011.5766889.

[Vos11] Sören Voskuhl. „Modellunabhängige Kontextinterpretation in einer Smart Home Umgebung“. Masterarbeit. Hochschule für Angewandte Wissenschaften Hamburg, 2011.

[Wil07] Erik Wilde. *Putting Things to REST*. School of Information, UC Berkeley, 2007. URL: http://dret.net/netdret/docs/wilde-irep07-015-restful-things.pdf (besucht am 05. 04. 2016).

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 22. April 2016 Dorothee Laugwitz