Technische Universität München

Fakultät für Informatik

Bachelorarbeit in Informatik: Games Engineering

Verknüpfung von Sprachassistenten mit anderen Steuerungsmethoden in intelligenten Umgebungen

Florian Bayer

Technische Universität München

Fakultät für Informatik

Bachelorarbeit in Informatik: Games Engineering

Verknüpfung von Sprachassistenten mit anderen Steuerungsmethoden in intelligenten Umgebungen

Connecting speech assistants to other control methods in intelligent spaces

Autor: Florian Bayer

Aufgabenstellerin: Prof. Dr. Gudrun Klinker

Betreuer: Sandro Weber

Abgabedatum: 16.10.2017

Erklärung

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Datum, Unterschrift

Gliederung

1 Abstract

In dieser Arbeit geht es darum, einen Sprachassistenten mit einem neuartigen Interface zu verbinden, welches es ermöglicht, bestimmte Funktionen des Sprachassistenten durch externe Steuerungsmethoden auszulösen. Im ersten Schritt wird die Verwendbarkeit der momentan bekanntesten Sprachassistenten geprüft und verglichen. Im nächsten Schritt kann nun der Sprachassistent auf einem kleinen Einplatinencomputer wie zum Beispiel einem Raspberry Pi zum Laufen gebracht und folgend so modifiziert werden, dass er durch ein einfaches Webinterface programmierbar und steuerbar ist. Daraufhin sollen Steuerungsmethoden einfach auf das System zugreifen können, wobei es möglich sein muss, Geräte über das Webinterface zu konfigurieren und deren Funktionen mit dem Sprachassistenten und anderen Steuerungsmethoden zu verbinden.

The focus of this work is about creating a new type of interface, allowing speech assistants to be triggered and controlled by external control methods. At first the usability of the different available speech assistants must be compared. In the next step the speech assistant is implemented on a small development board like the Raspberry Pi. After that the speech assistant is modified so that it can be triggered and controlled from an external web interface. As soon as this functionality was implemented, the web interface could be expanded, making it easy for other control methods to interact with the speech assistant. Configuring devices and setting up their triggering behavior over the web interface was also made possible.

2 Einführung

Sprachassistenten haben sich im letzten Jahrzehnt durch viele Neuerungen in der Informatik, wie zum Beispiel leistungsfähigere Rechner und Cloud Services mit maschinellem Lernen, fortentwickelt. Auch im Bereich der Smart Home Geräte und deren Verknüpfung mit anderen Diensten hat sich einiges getan. So wurden unter anderem neue und sehr einfache anzuwendende Produkte auf den Markt gebracht, wie Lichter, deren Farbe über Smartphones einstellbar sind, Umgebungssensoren, deren Ausgabedaten direkt mit dem Heizungssystem im Haus kommunizieren können und vieles mehr. Da sich Kamera unterstützte Gestenerkennung und Bewegungssensoren immer weiter an einen alltagstauglichen Standard annähern, ist es möglich, diese Geräte in einfachen Smart Home Umgebungen einzubinden. All diese Themengebiete waren der Ursprung dieser Arbeit. Es geht darum, eine Möglichkeit zu finden diese ganzen Neuerungen auf einfache Art und Weise zu verbinden, um eventuell einen Vorteil für den Nutzer oder Betreiber zu erschaffen.

2.1 Erklärung verwendeter Begriffe

Im Titel dieser Arbeit geht es um drei Bestandteile: Sprachassistenten, andere Steuerungsmethoden und intelligente Umgebungen.

Sprachassistenten wandeln gesprochenen Text in geschriebenen Text um und verarbeiten diesen.

Andere Steuerungsmethoden erkennen Gesten oder registrieren Eingaben von Nutzern und rufen daraufhin bestimmte vorprogrammierte Funktionen über einen HTTP Link auf.

Mit intelligenten Umgebungen sind alle Umgebungen gemeint, welche über ein Netzwerk verfügen, das mit mindestens einem Smart Home Gerät gefüllt ist. Diese wiederum müssen durch Anweisungen über das Netzwerk steuerbar sein.

2.2 Ziel der Arbeit

Das Ziel dieser Arbeit besteht aus zwei Teilen. Zunächst muss ein Interface programmiert werden, welches es ermöglicht einen modernen Sprachassistenten zu steuern, um ihm im

Folgenden durch eine Benutzeroberfläche bestimmte Funktionen zuzuteilen. Diese sollen dann über eine Schnittstelle durch andere Steuerungsmethoden aufrufbar sein.

Aus diesen Zielen entstand der Name des Interfaces Smart Home Terminal: SMATER.

Mit dem Interface wird zum Beispiel folgendes Szenario ermöglicht:

Ein Nutzer löst via einer Deutgeste die Funktion „Trigger Küchenlicht“ aus. Das Interface erkennt, das die Lampe ausgeschaltet ist und frägt den Nutzer mittels Sprachassistent nach der gewünschten Farbe für das Licht „Küchenlicht“. Der Nutzer antwortet dem Sprachassistenten mit „Grün“. Das Interface macht „Küchenlicht“ in der Farbe „Grün“ an. Wird die Funktion erneut getriggert, erkennt SMATER, dass die Lampe bereits eingeschalten ist, schaltet diese aus und teilt über den Sprachassistenten mit „Ich schalte Küchenlicht aus“.

3 Vorbereitung

Vor der Umsetzung von SMATER waren einige Vorbereitungen notwendig. Zunächst wurde eine Liste mit vom Interface zu erfüllenden Anforderungen erstellt, um nach einem geeigneten Sprachassistenten mit passendem Betriebssystem und Hardwareumgebung zu finden. Erst nach Vollendung dieser Vorbereitungen, war es möglich mit der Umsetzung des Interfaces zu beginnen.

3.1 Anforderungen

Nach einigen Überlegungen wurden folgende Anforderungen an SMATER zusammengestellt.

- SMATER ist ein einfach zu bedienendes und leicht zu erreichendes Webinterface.
- In dem Webinterface ist es möglich, bestimmte Funktionen zu erstellen.
- Diese Funktionen sind über eine simple HTTP-GET Schnittstelle aufrufbar (für die Kommunikation mit externen Steuerungsmethoden.)
- Nutzer erhalten somit die Möglichkeit, bestimmte Vorgänge in der intelligenten Umgebung zu automatisieren.
- Funktionen bestehen aus in Textform formalisierten Aufrufen an einen Sprachassistenten.
- Beim Ausführen einer Funktion wird der Aufruf an den Sprachassistenten weitergegeben, als hätte der Nutzer diesen wörtlich gesagt.
- Bei Rückfragen des Sprachassistenten kann der Nutzer mündlich antworten.
- Sobald eine Anfrage erfolgreich durchgeführt wurde, wird das passende Smart Home Gerät geschaltet oder es erfolgt eine Sprachausgabe.

3.2 Sprachassistenten

Um einen geeigneten Sprachassistenten zu finden, wurde zunächst eine Tabelle mit allen modernen Sprachassistenten erstellt und nach Ausschlusskriterien sortiert, um eine erste grobe Auflistung geeigneter Kandidaten zu finden.

Tabellenschlüssel	
Potentieller Kandidat	Ausgeschieden

Sprachassistenten	Plattform	Ausgeschieden weil
Alexa	Integriert auf ECHO Geräten. Quelldateien zum Schreiben einer eigenen Integration sind verfügbar	/
Google Assistant	Integriert auf Google Home Geräten. Quelldateien zum Schreiben einer eigenen Integration sind verfügbar	/
Hound	Open Source API (Alle Plattformen)	/
Lucida	Open Source API (Alle Plattformen)	/
Mycroft	Open Source API (Alle Plattformen)	/
Aido	Aido Roboter	Nur auf Aido Roboter
BlackBerry Assistant	Blackberry Mobiltelefone	Nur für Blackberry
Braina	Windows Betriebssysteme	Nur für Windows Betriebssysteme. Client nicht modifizierbar.
Cortana	Windows Betriebssysteme	Nur integriert in Windows verfügbar.
S Voice	Samsung Geräte	Nur auf Samsung Geräten verfügbar

Siri	Apple Geräte	Nur auf Apple Geräten verfügbar
Viv	Noch nicht verfügbar	Nicht verfügbar

Abbildung 1: Vergleich verschiedener Sprachassistenten sortiert nach deren Ausscheidungskriterien

Viele Sprachassistenten wurden leider nur für bestimmte Plattformen entwickelt und bieten deshalb keine API an, um diese zu modifizieren. Aus diesem Grund hat sich die Suche sehr schnell eingegrenzt. In der engeren Auswahl haben sich nur noch Amazon Alexa, Google Assistant, Hound, Lucida und Mycroft gefunden. Es wurde sich gegen die eher unbekannten Assistenten entschieden, da diese vergleichsweise nur wenig Community-Support bezüglich deren API aufweisen konnten. Folglich bestand die engste Auswahl aus dem Google Assistant und Amazon Alexa. Beide Sprachassistenten unterscheiden sich hinsichtlich der wichtigsten Faktoren, Modifizierbarkeit und Implementierbarkeit, nur sehr geringfügig. Letztendlich wurde sich für Amazon Alexa entschieden, da diese zum Zeitpunkt der Vorbereitungen den größten Umfang an „Skills“ anbieten konnte. Der Google Assistant diente als Ersatz für Alexa.

3.3 Plattformen

SMATER muss möglichst wenig Platz einnehmen und einfach in ein Netzwerk integrierbar sein. Es bot sich daher an, einen Einplatinencomputer zu verwenden, auf dem die gesamte Software ausgeführt wird. Der Computer muss eine Netzwerk Schnittstelle und ausreichend Rechenleistung haben, um gleichzeitig Sprachassistent und Webserver darauf laufen lassen zu können. Außerdem sollte es möglich sein, ein Linux basiertes Betriebssystem zu installieren, da dies am wenigsten Speicherplatz benötigt, wenig allgemeinen Overhead hat, sehr zuverlässig ist und eine passende Umgebung für die gesamte Software liefert. Da der Raspberry PI 3 Model B all diese Anforderungen trifft und zudem noch der mit Abstand kostengünstigste Computer ist, wurde sich bei der Umsetzung für diesen entschieden.

3.4 Versuchsaufbau

Nachdem alle gedanklichen Vorbereitungen getroffen waren, wurde mit dem Zusammenstellen des Versuchsaufbaus begonnen. Dieser besteht aus sechs wichtigen Einzelteilen:

1. Raspberry Pi (Mit Bildschirm)
2. Mikrofon (Befindet sich im Gehäuse)
3. Lautsprecher
4. Hue Bridge
5. Switch
6. Hue Lampe (Nicht auf der Abbildung zu sehen)

In folgender Abbildung wird der Aufbau mit den nummerierten Einzelteilen dargestellt. Die Philips Hue Lampe ist auf dem Bild nicht zu sehen, da diese an einer beliebigen Steckdose in der intelligenten Umgebung angebracht werden kann und daher nur indirekt zum Versuchsaufbau gehört. Sie muss also nicht direkt mit dem System verbunden sein, sondern sich lediglich in der Wireless Reichweite der Hue Bridge befinden.

Abbildung 2: Versuchsaufbau SMATER

4 Realisierung

4.1 Übersicht

Da die Realisierung von SMATER auf sehr vielen Einzelbereichen basiert, die alle miteinander verknüpft sind, wird im folgenden Abschnitt zunächst eine grobe Übersicht geschaffen, um im darauffolgenden Teil genauer auf die einzelnen Elemente eingehen zu können.

- Raspberry Pi

  Auf Ihm laufen der lokale Webserver und die Alexa App. Der lokale Webserver bietet das User Interface und das Backend zur Kommunikation mit allen anderen benötigten Diensten an. Weiter befindet sich hier eine MySQL Datenbank und alle Programme, die notwendig sind, um mit den anderen unten genannten Bereichen zu Daten auszutauschen. Der Raspberry Pi ist mit demselben Netzwerk verbunden, mit dem auch alle anderen Smart Home Links verknüpft sind. (Im Versuchsaufbau ist die Hue Bridge über den Switch mit dem Raspberry Pi gekoppelt). Außerdem hat das Netzwerk eine Verbindung zum Internet um die Kommunikation mit Online Diensten zu ermöglichen.

- Voice RSS

  Ein mündlicher Befehl durch den Nutzer wird umgangen, indem der Raspberry Pi Texte an Voice RSS sendet, die dort in synthetisierte Sprachdateien umgewandelt, und anschließend zurück an den Computer übertragen werden. Dieser bewahrt die Audiodateien auf, um Befehle an den Alexa Server senden zu können.

- Amazon Server

  Der Raspberry Pi schickt entweder den Nutzer Input vom Mikrofon oder eine vorgefertigte Sprachdatei von Voice RSS hierhin. Daraufhin wandelt Amazon die Audiodateien in Text um, und erkennt, ob der SMATER Skill aufgerufen wird oder ob es sich um einen anderen Skill handelt, der nativ behandelt werden muss. Falls es sich

um den SMATER Skill handelt, verarbeitet dieser die gewünschte Anfrage und sendet einen Befehl zum Erstellen einer Aktion an den SMATER Webserver.

- SMATER Webserver

  Dieser Webserver ermöglicht dem SMATER Skill mit dem Raspberry Pi zu kommunizieren, ohne sämtliche Ports öffnen zu müssen oder DNS Umleitungen einzurichten. Einerseits sendet der SMATER Skill Anfragen an diesen Server, um den Status bestimmter Geräte in der Umgebung abzurufen. Andererseits übermittelt der Raspberry Pi bei jeder Änderung der Geräte in seinem Netzwerk eine Aktualisierung hierhin, wodurch dem SMATER Skill ermöglicht wird, dauerhaft auf dem aktuellen Stand zu sein. Falls der Skill ein Gerät ansteuern will, liefert er einen Befehl zur Erstellung einer Aktion, um dieses zu schalten. Dieser wird zunächst in der SMATER Datenbank abgelegt, die der Raspberry Pi frägt regelmäßig abfragt und folglich die Ausführung der Aktion einleitet.

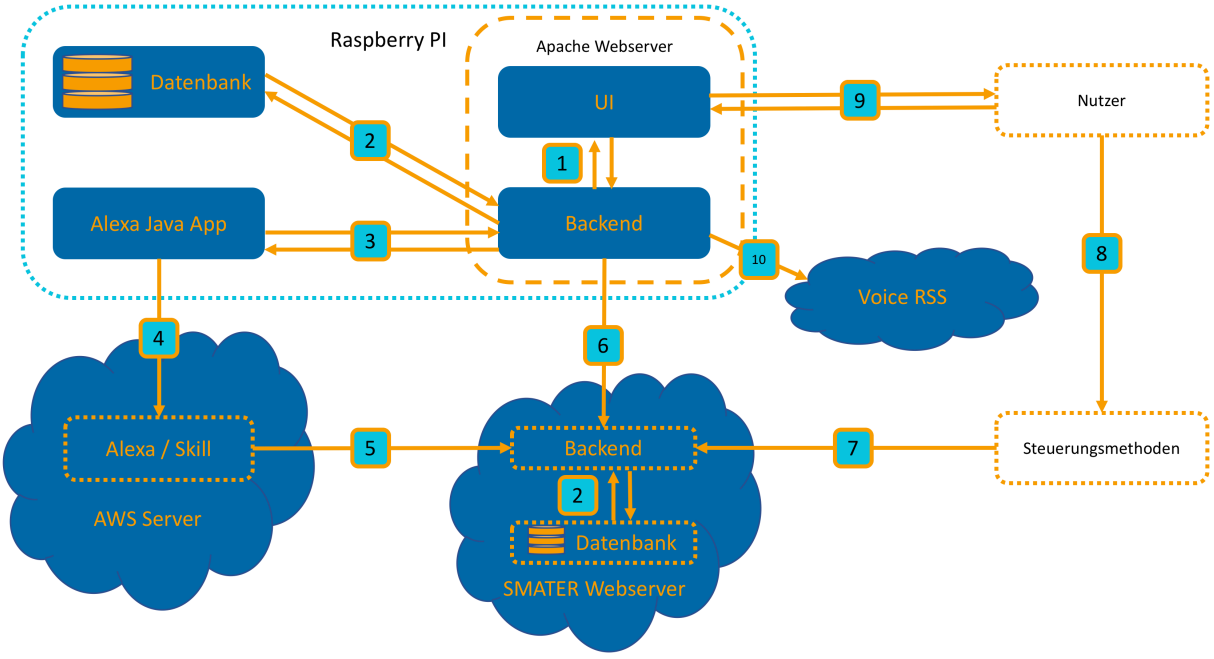

Abbildung 3: Grober Aufbau des SMATER Gesamtsystems

Im obigen Schema sind die vier bereits genannten Bereiche grob dargestellt, welche im folgenden Abschnitt kurz erläutert werden. Dabei symbolisieren die orangenen Pfeile die Kommunikationen zwischen ihnen. Besonders beachtenswert ist, dass der Raspberry Pi keine eingehenden Verbindungen aufweist, was anhand der Pfeilrichtungen zu erkennen ist.

1. Die Benutzeroberfläche speichert eingegebene Daten im Backend und erhält dynamische Inhalte zurück.
2. Verbindung zwischen Webserver und Datenbank
3. Das Backend sendet Befehle an die Alex Java App, um Alexa auszulösen oder eine vorgefertigte Funktion abzuschicken. Die Java App teilt dem Backend wiederum seinen Status mit.
4. Alexa Java App sendet Mikrofon Audio oder synthetisierte Sprachdateien an den Alexa Server
5. Der Alexa Skill übermittelt Befehle zum Erstellen von Aktionen an den SMATER Webserver und erhält auf Anfrage Gerätelisten.
6. Das Backend des Raspberry Pis sendet dauerhaft seinen Status und Gerätelisten an den SMATER Server und stellt Fragen nach Aktionslisten.
7. Steuerungsmethoden fordern via dem SMATER Webserver auf, Funktionen auf dem Raspberry Pi auszuführen.
8. Die Benutzer kopieren API Links vom lokalen Webinterface in das Interface der Steuerungsmethoden um diese zu konfigurieren.
9. Der Benutzer erstellt via dem User Interface Funktionen und verwaltet die Geräte in der intelligenten Umgebung.
10. Das lokale Backend fordert den Voice RSS Server auf, Sprachdateien zu erstellen.

4.2 Alexa Voice Service App

Amazon stellt ein Beispielprogramm des Alexa Voice Service inklusive Quellcode in Java bereit. Mithilfe dieses Applets wird es möglich, Alexa so auf einem Gerät zu installieren, dass es sich fast identisch zu einem Amazon Dot oder Amazon Echo verhält. Da hier jedoch ebenfalls der

Quellcode inklusive Kompilierungsanleitungen vorhanden ist, wird es möglich die Funktionalität des Programms zu erweitern. Im folgenden Abschnitt wird erläutert, welche Änderungen an der Beispielimplementierung vorgenommen wurden, um Alexa an die Anforderungen des Gesamtprojekts anzupassen.

4.2.1 Externer Input

Um Alexa durch einen externen Trigger zuhören zu lassen, oder direkt eine Anfrage an Alexa zu schicken, muss ein externes Programm gewisse Funktionen des Java Applets aufrufen können. Hierfür wurde eine neue Datei „`ExternalInputChecker.java`“ erstellt. Die Klasse dieser Datei liest dauerhaft Zeilen aus einer Textdatei „`ComFile.txt`“ aus, welche direkt danach entleert wird. Somit verhält sich die Textdatei wie eine Warteschlange von Befehlen, wobei jeder davon geparst und an das Programm weitergegeben wird. Ein anderes Programm, wie zum Beispiel das Webinterface, kann jetzt Alexa ansteuern.
Folgende Befehle wurden zu diesem Zweck implementiert:

- `start_listening`
  Alexa fängt sofort an zuzuhören, ohne durch das gesprochene Wakeword „Alexa“ aktiviert werden zu müssen.

- `text_command <wav_befehls_datei>`
  Hiermit kann eine schriftliche Anfrage an Alexa gesendet werden. Bei Rückfragen von Alexa kann dennoch der Mikrofoneingang verwendet, um dem Nutzer zuzuhören. Die Anfrage wird als `.wav` Audiodatei übergeben, deren Verarbeitung im folgenden Abschnitt erklärt wird.

4.2.2 Text to Speech via Voice RSS

Die Umwandlung von geschriebenen in gesprochenen Text (Text to Speech) ist notwendig, um mit Alexa zu kommunizieren, ohne Sprache benutzen zu müssen. Da die Alexa Voice Service

API diese Funktion nicht standardmäßig anbietet, wurde nach einer alternativen Lösung gesucht.

Nach einigen Versuchen, wobei ein Handy mit einer abspielenden Audiodatei an das Mikrophon des Raspberry Pi gehalten wurde, stellte sich heraus, dass Alexa auch auf synthetisierte Sprache antworten kann. Um diesen Workaround sinnvoll nutzen zu können, war die Umsetzung zweier Funktionalitäten notwendig. Erstens muss es automatisiert möglich sein, Text in eine passende synthetisierte Sprachdatei umzuwandeln, und zweitens muss diese Sprachdatei, anstatt dem Mikrophon Stream, an die Amazon Server übertragen werden.

Da die Umwandlung von Text in Sprache maschinelles Lernen und komplizierte Algorithmen benötigt, um ausreichend menschlich zu klingen, entfiel die Option der lokalen Lösung auf dem Raspberry Pi. Stattdessen wurde nach einer kostengünstigen Alternative in Form eines Web APIs gesucht. Nach einigen Recherchen wurde sich für `http://www.voicerss.org/` entschieden. Der Vorteil dieses API ist es, dass es nicht nur deutsche Sprache unterstützt, sondern sogar unterschiedliche Formate der Audiodatei anbieten kann, um diese bestmöglich an das von Alexa benötigte Format anzupassen. Zudem erlaubt Voice RSS 350 kostenlose Anfragen pro Tag. Eine detailliertere Erklärung über den Umgang mit den Sprachdateien auf dem Webserver und in der Datenbank folgt unter dem Punkt 4.3.3.

Um schließlich die Übertragung an Alexa zu ermögliche, wurde die Klasse „`AudioCapture.java`“ modifiziert. Hier wird normalerweise ein Thread geöffnet, welcher mit einer bestimmten Rate, rohe Daten aus dem Mikrofon-Stream in den Upload-Audio-Stream kopiert. An genau dieser Stelle wurde einige Abänderungen vorgenommen: der Thread kopiert sobald eine bestimmte Flagge gesetzt ist nicht mehr aus dem Mikrofonausgang, sondern aus einem File-Stream-Ausgang alle Daten der Audiodatei in den Upload-Audio-Stream. Dies geschieht sofort und nicht in bestimmten Raten. Somit wird auch die Dauer für die Verarbeitung längerer Anfragen reduziert. In Abbildung 4 wird der Aufbau der neuen Klasse grafisch dargestellt.

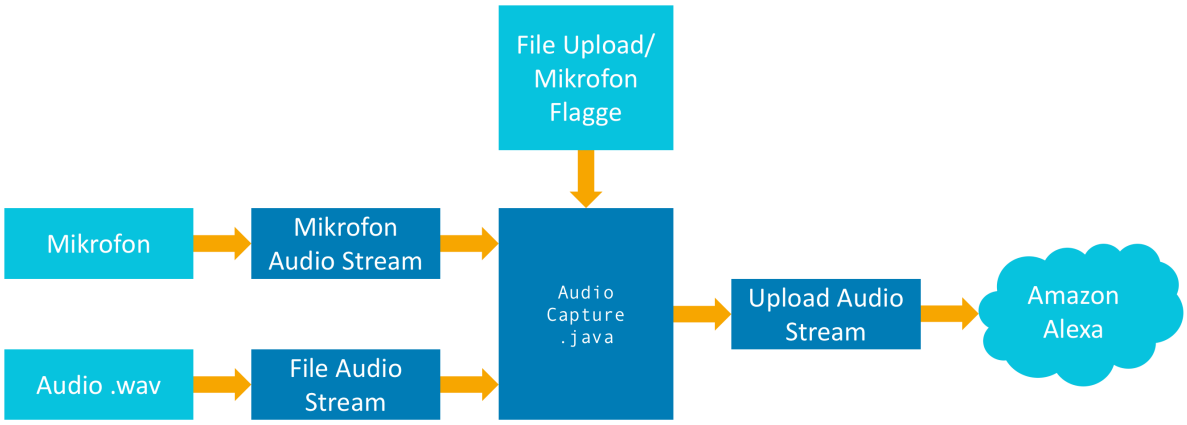

Abbildung 4: Funktionsweise von „AudioCapture.java“

4.2.3 Externer Output

Um dem Webinterface Feedback über den aktuellen Status von Alexa zu geben, wurde die statische Klasse „`ExternalStatusControler.java`“ erstellt. Diese beinhaltet eine Funktion, die jederzeit im Programm aufgerufen werden kann, um den aktuellen Status zu setzen. Bei diesem Prozess wird ähnlich wie beim Einlesen von Befehlen im „`ExternalInputChecker`“ in die Textdatei „`StatusFile.txt`“ geschrieben. Dabei wird die Datei immer zuerst entleert, und dann mit dem neuen Status überschrieben. Somit könnten auch andere Anwendungen den Status von Alexa abfragen um zum Beispiel eine Anzeige an dem Gerät zu erstellen, wie sie auch bei den Amazon Echos und Amazon Dots verwendet wird. Zuletzt wurde in einigen Teilen der Software ein Aufruf dieser Funktion eingefügt, um eine möglichst genaue Darstellung des aktuellen Standes der Anwendung zu ermöglichen.

Die folgenden vier Zustände können in die Textdatei vorkommen:

- `LISTENING`
  Alexa hört zu oder erhält eine Sprachdatei und spielt diese in den Upload Stream ein.

- `PROCESSING`
  Eine Anfrage wird von den Amazon Servern verarbeitet und Alexa wird in Kürze antworten oder eine Fehlermeldung ausgeben.

- `TALKING`
  Alexa Antwortet gerade auf eine Anfrage oder gibt eine Fehlermeldung aus.

- `„Leere Ausgabe“`
  Alexa ist momentan nicht in Verwendung und wartet auf neue Anfragen.

Das Webinterface hat nun die Möglichkeit aus einer Textdatei den Status von Alexa zu erhalten. Weitere Informationen zu der Verwendung der Statusdatei finden Sie unter 4.3.2.

4.3 Lokaler Webserver

Der lokale Webserver erfüllt zwei große Aufgaben. In erster Linie bietet er ein User Interface an, um SMATER zu konfigurieren. Außerdem verarbeitet er sämtliche API Aufrufe und Kommunikationen mit Alexa.
Das Webinterface stellt eine leicht zu erreichende lokale Webseite dar, auf der der Nutzer alle kompatiblen Smart Home Geräte sehen und dafür Funktionen erstellen kann. Diese kommunizieren entweder mit Alexa oder direkt mit den Geräten in der intelligenten Umgebung. Für jede erstellte Funktion wird nun automatisch ein Link erzeugt, um sie von anderen Steuerungsmethoden aus aufrufen zu können.
Bei der Umsetzung des lokalen Webservers wurde sich für einen einfachen LAMP Stack entschieden (Linux, Apache, MySQL, PHP). Dieser ermöglicht nicht nur die Entwicklung eines dynamischen User Interfaces, sondern auch die eines umfangreichen Backendes. Somit ist es sehr einfach, einen LAMP Stack auf einem Raspberry PI einzurichten, da auf diesem, wegen Alexa, bereits eine Linux Distribution läuft.

4.3.1 MySQL Datenbank

In der lokalen Datenbank befinden sich einige Tabellen mit Einstellungen, Smart Home Geräten, Funktionen und Sprachbefehlen an Alexa. Die einzelnen Tabellen und deren Verwendung werden in folgender Auflistung erläutert.

Tabelle	Daten	Nutzen
`devices`	Enthält sämtliche Daten aller Geräte in der aktuellen Umgebung. Diese werden vom lokalen Webserver ständig abgefragt und in diese Tabelle geschrieben. Sie beinhaltet für jedes Gerät eine eindeutige interne Identifikationsnummer (`internal_id`) und eine eindeutige Identifikationsnummer des bestimmten Gerätetyps (`unique_id`). Außerdem wird der aktuelle Zustand der Geräte (`state`), ihre Erreichbarkeit (`reachable`), ihr Typ (`type`) und ihr Name (`name`) abgelegt.	Der lokale Webserver aktualisiert ständig diese Liste. Sobald sich eine Änderung ergibt, wird die gesamte Tabelle mit dem SMATER Webserver synchronisiert. Somit kann auch Alexa Zugriff auf den aktuellen Status der Geräte erhalten.
`functions`	Vom Nutzer erstellte Funktionen. Eine Funktion besteht aus einer Identifikationsnummer (`id`), einem Namen (`name`) und einer Anfrageidentifikationsnummer (`request_id`).	Sobald der Nutzer eine Funktion erstellt, wird diese in eine Anfrage an Alexa umgewandelt, und zusammen mit ihrem Namen und einer eindeutigen Identifikationsnummer, hier abgelegt. Sobald eine Funktion ausgeführt wird, sucht der Webserver in dieser Tabelle nach der abgelegten Nummer, und kann mit dieser die Anfrage an Alexa finden.

`hue`	Verbundene Hue Bridges. Diese sind mit ihrer IP-Adresse (`ip`) und ihrem API-Login (`username`) abgelegt.	Nachdem eine Hue Bridge mit dem Webserver gekoppelt worden ist, wird deren Login in dieser Tabelle abgelegt, sodass dies nicht bei jedem Neustart erfolgen muss.
`requests`	Anfragen an Alexa. Beinhaltet eine Identifikationsnummer (`id`), die Textform der Anfrage (`requestString`) und den Namen der passenden Sprachdatei (`file`).	Wird vom Webserver abgefragt bevor eine neue Sprachdatei von Voice RSS erstellt wird. Somit wird eine Doppelung vermieden. Stattdessen wird die in der Datenbank hinterlegte Sprachdatei verwendet.
`settings`	Einstellungen und Hilfsvariablen für den Webserver. Jede Zeile stellt eine Einstellung mit einer Identifikationsnummer (`id`), einem Namen (`name`) und einem Wert (`value`) dar.	Hier sind zusätzliche Einstellungen, wie zum Beispiel der API Schlüssel, abgelegt.

Abbildung 5: Struktur der lokalen MySQL Datenbank

4.3.2 User Interface

Das User Interface besteht aus einer einfachen Website, die lokal auf dem LAMP Server des Raspberry Pi läuft. Sie wiederum enthält drei wichtige Unterseiten: Geräte, Funktionen und Einstellungen. Die Funktionsweise der einzelnen Seiten wird im Folgenden detailliert beschrieben.

- `Geräte`

Diese Seite enthält zwei Tabellen. In der oberen Tabelle sind alle verfügbaren und bereits gekoppelten Geräte angezeigt. Von hier aus können Sie testweise ein und aus geschalten werden um, deren Funktionalität zu überprüfen. In der unteren Tabelle wird eine Liste von koppelbaren oder bereits gekoppelten Hubs angezeigt (momentan unterstützt SMATER nur die „Philips Hue Bridge“). Der Nutzer kann hier auf den Knopf „Koppeln“ drücken um, ein neues Hub zu dem System hinzuzufügen und dessen Geräte freizuschalten, woraufhin diese automatisch in der oberen Tabelle angezeigt werden. Sobald alle Geräte eingetragen sind, kann der Nutzer mit der Seite „Funktionen“ fortfahren.

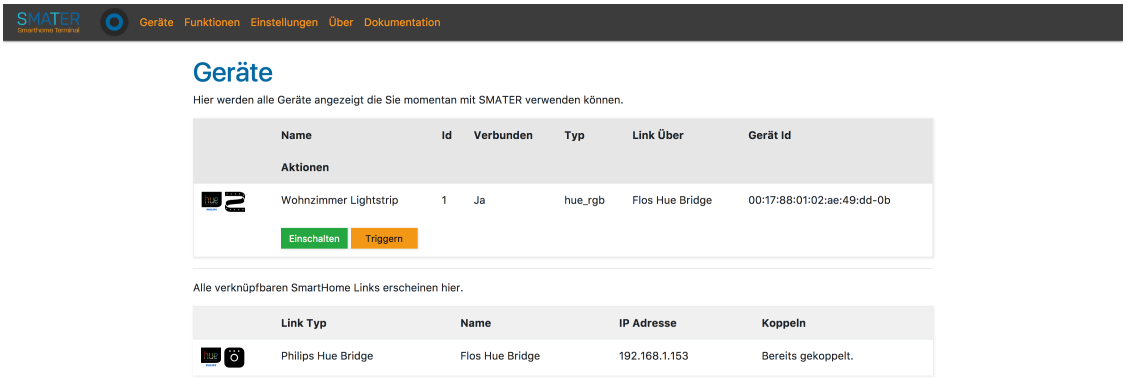

Abbildung 6: Seite „Geräte“

- **Funktionen**

  Hier wird eine Tabelle mit allen vom Nutzer erstellten Funktionen inklusive des zugehörigen API Links angezeigt. Durch einen Klick auf den Button „Ausführen“, kann jede davon einzeln ausgelöst werden. Mit dem Button „Entfernen“ können Funktionen gelöscht, oder über den Button „Audio“, deren Anfrage-Sprachdatei, angehört werden.

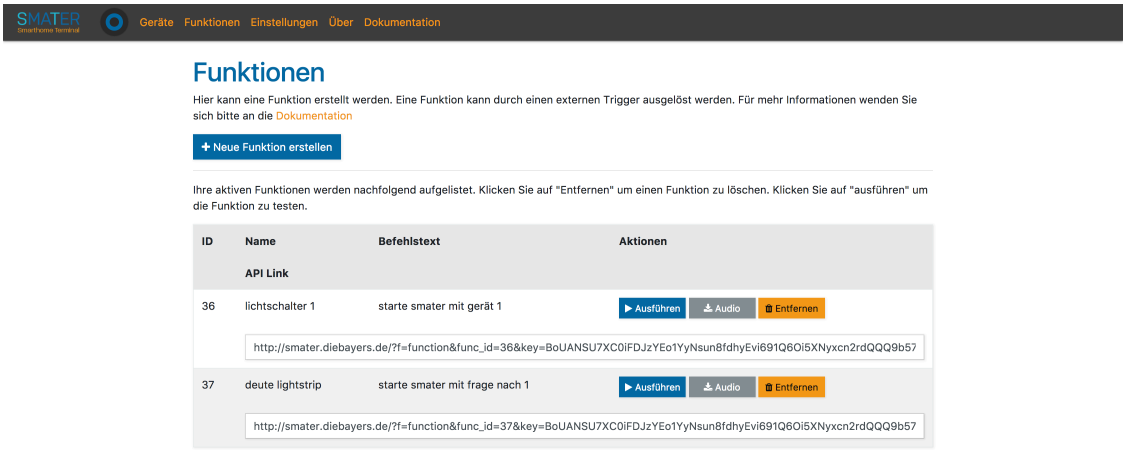

Abbildung 7: Seite „Funktionen“

Um eine neue Funktion zu erstellen, kann der Nutzer auf den Button „Neue Funktion erstellen“ klicken. Auf dieser Seite hat er nun die Möglichkeit, sie zu benennen und ihre Funktionsweise zu erstellen. Es gibt zwei Varianten zur Erstellung einer Funktion.

a) Direkte Eingabe eines Befehls als Text, der dann an Alexa gesendet wird (Abbildung 8)
b) Auslösen einer Funktion (Abbildung 9). Hier wird automatisch eine Liste mit allen verfügbaren Geräten angezeigt. Sobald eines davon selektiert worden ist, kann eine passende Funktion dafür ausgewählt werden.

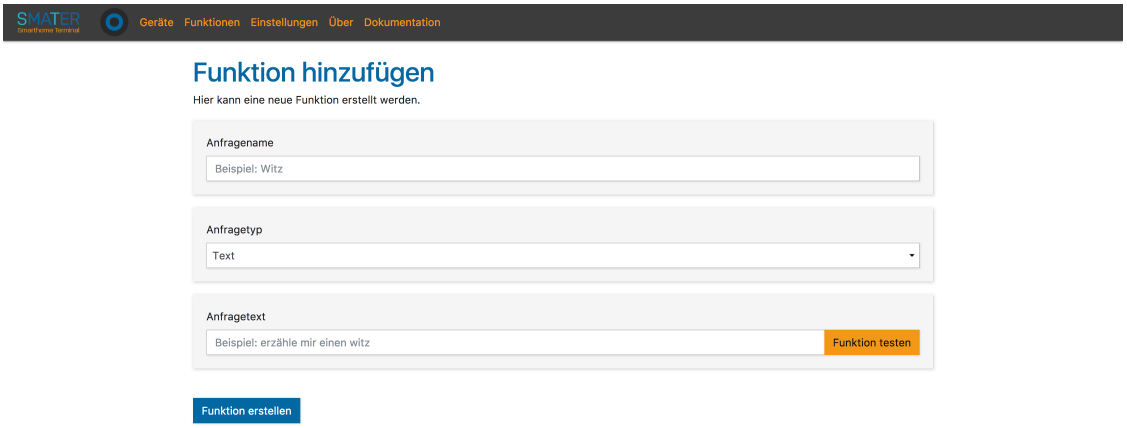

Abbildung 8: Seite „Funktion hinzufügen“ mit Variante „Text“

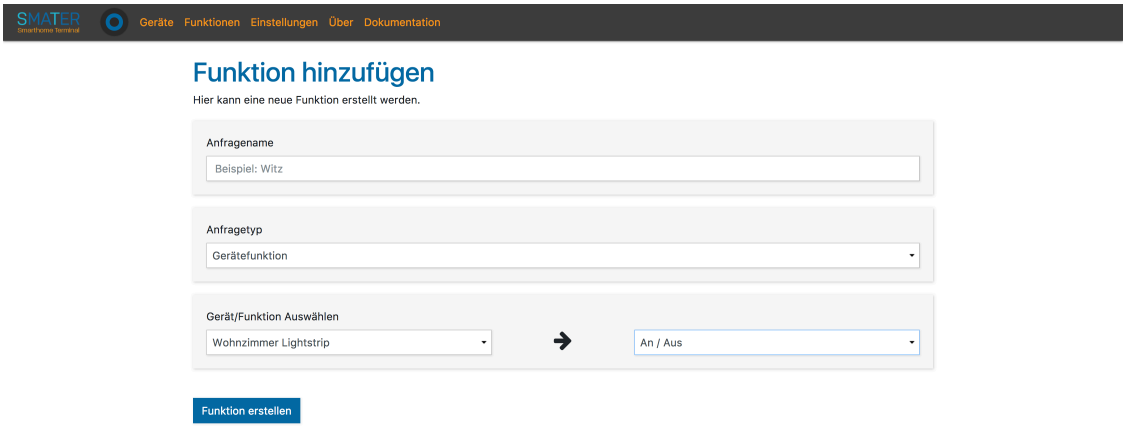

Abbildung 9: Seite „Funktion hinzufügen“ mit Variante „Gerätefunktion“

- **Einstellungen**

  Bei den Einstellungen wird der API Schlüssel eingefügt. Dieser muss nur in das entsprechende Feld kopiert werden. Danach wird er automatisch überprüft und in die Datenbank gespeichert, falls er gültig ist (dies wird durch ein grünes Häkchen neben dem Schlüssel symbolisiert). Weitere Information über den API Schlüssel finden Sie unter 4.5.1.

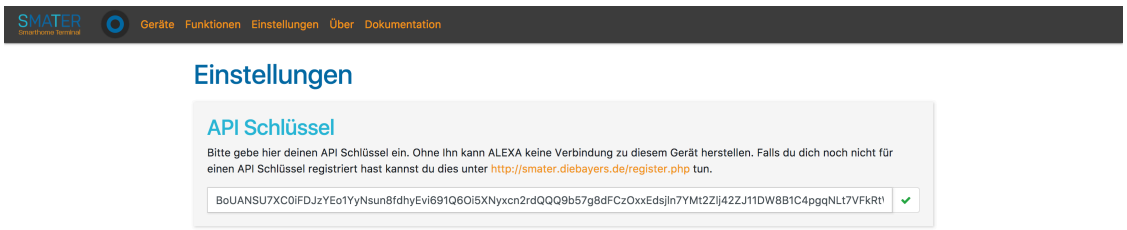

Abbildung 10: Seite „Einstellungen“

Über den drei Hauptseiten befindet sich zusätzlich noch eine Leiste, in der sich ein blaues rundes Statussymbol befindet. Dieses zeigt immer den aktuellen Status von Alexa an. Mit der Anzeige kann der Nutzer visuell erkennen, ob Alexa gerade eine Anfrage verarbeitet oder spricht. Durch Klicken auf das Symbol fängt Alexa an zuzuhören. Dies wird durch die beiden Textdateien aus 4.2.1 und 4.2.3 ermöglicht. Dabei liest ein kleiner JavaScript Loop ständig die Statusdatei aus und ändert mit dieser Information die CSS Klasse des Symbols, um seine

Darstellung zu verändern. Um das Zuhören von Alexa zu initiieren, schreibt ein PHP Skript „`start_listening`“ in die Befehlsdatei.

4.3.3 Backend

Das Backend auf dem Raspberry Pi basiert zum größten Teil auf einigen PHP Skripten, die sich ebenfalls auf dem LAMP Server befinden. Um jedoch periodische Abfragen zu ermöglichen, läuft neben dem Webserver noch eine Python Datei „`reloadPage.py`“. Diese ruft in gleichmäßigen zeitlichen Intervallen die Dateien „`deviceSync.php`“ und „`cron.php`“ auf.

- `deviceSync.php`
  Dieses Skript hat zwei Funktionen. Erstens aktualisiert es die IP Adresse des Raspberry Pi auf den SMATER Webserver und überprüft zweitens alle Geräte in der intelligenten Umgebung. Aus ihnen bildet es dann eine nummerierte Liste. Sobald diese erstellt wurde, wird ein Abgleich mit der lokalen Datenbank durchgeführt. Falls keine Übereinstimmung der beiden Listen vorliegt, wird die Neuerstellte zunächst in die lokale Datenbank und danach mit in die SMATER Datenbank aufgenommen.

- `cron.php`
  Dieses Skript hat die Aufgabe, den SMATER Webserver nach anstehenden Aktionen abzufragen und diese gegebenenfalls auszuführen. Im Abschnitt 4.5.4. wird dieses Vorgehen tiefergehend beschrieben.

Neben den beiden periodischen Skripten beinhaltet das Backend auch die Verwaltung der Sprachdateien. Sobald ein Nutzer eine Funktion erstellt oder aufruft, wird zunächst überprüft, ob bereits eine passende Audiodatei auf dem Server vorhanden ist. Alle bereits erstellten Dateien sind im Ordner „`requestAudio`“ auf dem lokalen Webserver abgelegt und, wie unter 4.3.1 erklärt, in der Datenbank angeordnet. Allgemein vergleicht der Server den zu erstellenden Befehl mit der Datenbank und gibt entweder direkt die vorhandene Sprachdatei zurück oder ruft die Datei „`voicerss_tts.php`“ auf. Diese wiederum löst einen API Aufruf an `http://api.voicerss.org/` aus, um eine passende Sprachdatei zu erhalten. Hat diese den

lokalen Webserver erreicht, wird sie zu den anderen Dateien in den Ordner „`requestAudio`“ gelegt und in der Datenbank mit dem zugehörigen Befehlstext verknüpft.

4.4 Skill

Um die Interaktion von Alexa mit den einzelnen Smart Home Geräten zu ermöglichen, wurde ein Skill für Sie programmiert. Zur Verdeutlichung der Notwendigkeit dieses Skills, blicken wir zurück auf das Ziel der Arbeit: Alexa soll nach dem Auslösen einer bestimmten Funktion zunächst fragen, welchen Zustand ein bestimmtes Gerät einnehmen soll. Einige Skills sind bereits standardmäßig installiert, mit denen Alexa Smart Home Geräte ansteuern kann. Diese erfordern jedoch immer eine direkte Anweisung im Stil „`mache {gerät} mit Zustand {Zustand} an`“. Nachdem unsere Funktionen aus Anfragen an Alexa bestehen, brauchen wir einen Skill, der auf die Anfrage „`sage SMATER aktiviere {Gerät}`“ mit der Frage „`welchen Zustand soll {Gerät} haben`“ antwortet. Daraufhin muss der Nutzer nur „`{Zustand}`“ sagen und der Skill aktiviert `{Gerät}` mit dem passenden `{Zustand}`. Um diesen Vorgang deutlicher zu machen, wird in folgendem Sequenzdiagramm ein beispielhafter Ablauf dargestellt.

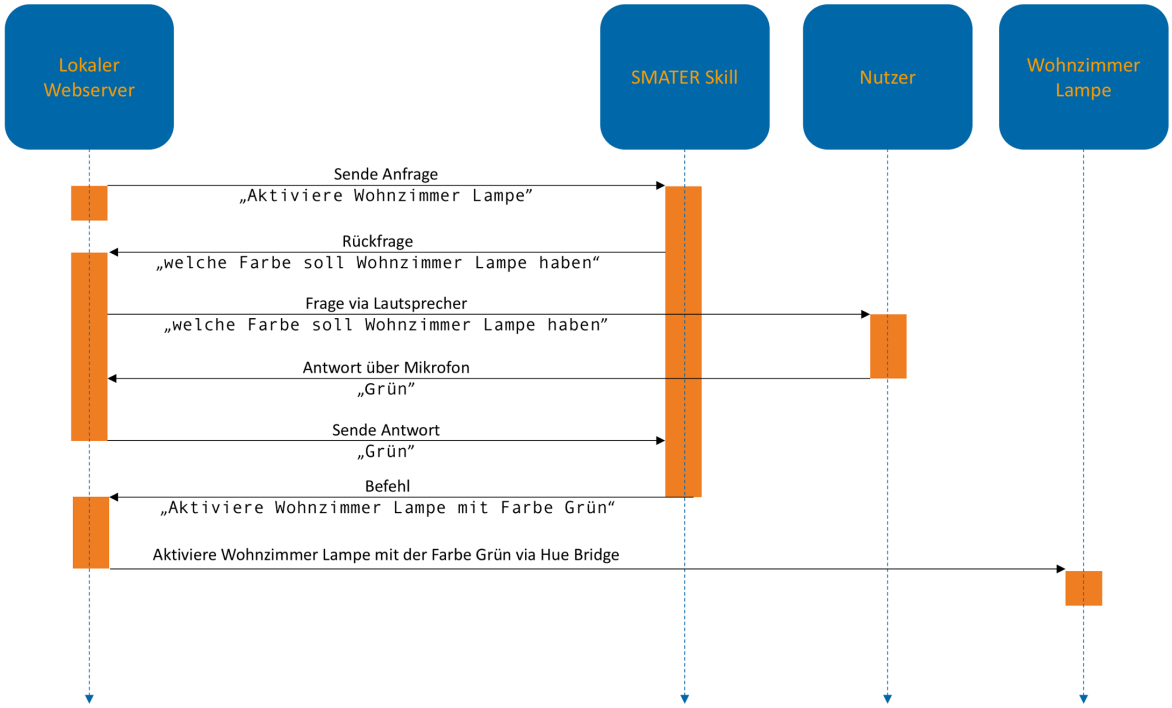

Abbildung 11: Sequenzdiagramm zur Veranschaulichung eines SMATER Skill Aufrufes.

4.4.1 Überblick

Um einen Skill für Alexa zu programmieren, sind zwei wichtige Komponenten zu erstellen. Erstens muss die grundlegende Struktur eines Aufrufs, auch Intent genannt, und alle Metadaten des Skills festgelegt werden. Dies geschieht direkt Online in der Amazon Entwicklerkonsole.

Zweitens programmiert man die Funktionalität des Skills, also die Verarbeitung bestimmter ausgelöster Intents. Dies wird mit einer sogenannten Lambda Funktion umgesetzt. Diese ist wie ein kleines, einfach auszuführendes Codesegment. Es läuft in einer flexiblen virtuellen Umgebung, die sich auf die aktuelle Last der Ausführung anpasst. Somit benötigt es bei niedrigem Ansturm wenige Ressourcen und kann für große Anfragemengen einfach hochskaliert werden. Die Lambda Funktion wir standardmäßig in einer Amazon Web Service Umgebung ausgeführt und in der Programmiersprache Node.js geschrieben.

4.4.2 Intents

Unser Skill hat drei grundlegende Intents. Diese werden in folgender Tabelle erläutert.

Intent	Struktur	Erklärung	Beispiel
`mit`	„mit Gerät `{device:AMAZON.NUMBER}`“	Gerät ohne Frage nach einer gewünschten Farbe an- oder ausschalten.	„`starte SMATER mit Gerät eins`“ Der lokale Webserver schaltet das Gerät mit der angegebenen lokalen Identifikationsnummer 1 an / aus.
`mit_fragend`	„mit Frage nach `{device:AMAZON.NUMBER}`“	Mit diesem Intent kann eine Frage nach der gewünschten Farbe eines Lichtes ausgelöst werden.	„`starte SMATER mit Frage nach Gerät drei`“ Falls das Gerät mit der lokalen Identifikationsnummer 3 ausgeschaltet ist, frägt der Skill zunächst, welche Farbe das Gerät haben soll, andernfalls schaltet er das Gerät aus.
`colorTold`	„`{color:color}`“	Dieser Intent wird dann relevant, sobald der Nutzer	„`grün`“

		den Intent „mit_fragend“ erfolgreich aufgerufen hat. Die gewünschte Farbe wird durch den Nutzer genannt, und an den Skill weitergegeben.	Der lokale Webserver schaltet das in einem vorherigen Intent genannte Gerät mit der Farbe Grün ein.

Abbildung 12: Beschreibung der einzelnen Intents des SMATER Skills

Die genaue Struktur der Intents wird im `.json` Format eingegeben und ist in der Datei „`skill.json`“ einsehbar.

4.4.3 Lambda Funktion

Sobald der Alexa Dienst unseren Skill erkennt, liefert er auch den Intent der Anfrage mit. Die Variablen „`device`“ und „`color`“ werden ebenfalls automatisch getrennt und mitübergeben. Sobald einer der beiden auslösenden Anfragen „`mit`“ oder „`mit_fragend`“ eingeht, stellt die Lambda Funktion eine Verbindung mit dem SMATER Webserver her. Von diesem ruft sie zunächst die Liste aller Geräte im Netzwerk ab (4.5.5 „`deviceList`“). Sobald diese erhalten wurde, wird nach dem Gerät mit der angegebenen Geräteidentifikationsnummer (Variable „`device`“) gesucht. Ist dieses nicht auffindbar, antwortet die Lambda Funktion mit einer Fehlermeldung. Anderenfalls kann eine weitere Unterscheidung erfolgen: Handelt es sich um den Intent „`mit_fragend`“, wird eine Frage nach der Farbe für das Gerät erstellt. Da neben der Identifikationsnummer auch der Namen des Gerätes bekannt ist, ist dies durch einfaches Zusammenfügen der String Werte leicht möglich. Falls es sich jedoch um den Intent „`mit`“ handelt, wird erneut eine Verbindung zu dem SMATER Webserver hergestellt und der Befehl zur Erstellung einer Aktion zum Umschalten eines Geräts gesendet (4.5.5 „`Execute`“).

Nun zum Verhalten des Intents „`colorTold`“. Da die Gerätidentifikationsnummer, der Name und der Zustand des gewünschten Gerätes bereits vorliegen, muss nur noch aus der gewählten Farbe ein Rot-, Grün- und Blau-Wert erstellt werden, um die Lampe mit diesem zu schalten. Dafür wird ein einfaches Switch-Statement verwendet. Falls die angeforderte Farbe nicht erkannt wird, folgt eine Fehlermeldung. Anderenfalls wird dem SMATER Webserver mitgeteilt, dass er eine Aktion zum Setzen der Farbe des Gerätes mit der passenden Identifikationsnummer erstellen soll (4.5.5 „`Execute`“). Die Funktion liefert in allen Fällen passende Rückgabetexte, welche bei der Ausführung von Alexa vorgelesen werden.
Der Quellcode der Lambda Funktion befindet sich in der Datei „`Lambda.js`“.

4.5 SMATER Webserver

Um den Skill zu aktivieren, wird über das Backend ein Befehl an die Alexa Java App gesendet. Dieser wird von dort aus direkt an den Amazon Server weitergeleitet. Um jedoch dem Skill zu ermöglichen, Befehle an den Raspberry Pi zu senden und Informationen über die verbundenen Geräte zu erhalten, müsste dieser direkte Anfragen an den lokalen Webserver stellen können. Diese Option kommt bei der Umsetzung von SMATER leider nicht in Frage, da dafür bestimmte Ports in dem Netzwerk freigegeben werden müssten, in dem sich der lokale Webserver befindet. Da dies in vielen Situationen nicht möglich ist, wurde sich bei der Entwicklung für eine alternative Lösung mit einem „Zwischenserver“ entschieden. Dieser stellt folgende Funktionen bereit:

- Einfaches Entdecken des lokalen Webservers über eine statische Adresse (4.5.1)
- Einfaches Erstellen eines Accounts, um einen API Schlüssel zu erhalten (4.5.2)
- Tabelle zur Bereitstellung der Gerätelisten für den Skill durch den lokalen Webserver (4.5.3)
- Weiterleitung von Aktionen an den lokalen Webserver (4.5.4)

4.5.1 Statische Adresse

Um einfach die IP-Adresse des lokalen Webservers herauszufinden, muss der Nutzer lediglich die Seite `http://smater.diebayers.de/devices` besuchen. Von dort aus wird er automatisch an die richtige Adresse (`http://<LOKALE_IP>`) weitergeleitet. Dies funktioniert, indem der lokale Webserver ständig dem SMATER Webserver mitteilt, welche IP Adresse er innerhalb des Netzwerks besitzt. Der SMATER Server erhält zudem auch die globale IP Adresse des Netzwerks, worin sich der lokale Webserver befindet. Da er nun die beiden Adressen erhalten hat, legt er diese in seiner Datenbank ab. Sobald ein Nutzer dem oben genannten Link folgt, erhält der Server wieder die globale Adresse, da der Nutzer sich in demselben Netzwerk befindet wie der lokale Server. Falls ein passender Eintrag in der Datenbank vorhanden ist, leitet der Server an die eingetragene lokale Adresse weiter.

4.5.2 Benutzer/API Schlüssel System

Um die Kommunikation mit mehreren SMATER Servern zu ermöglichen, wurde ein Benutzersystem angelegt. Dieses System beinhaltet eine sehr einfache Registrierungs- und Anmeldeseite. Sobald ein Nutzer sich mit seiner E-Mail-Adresse und einem Passwort registriert hat, erhält er einen API Schlüssel. Dieser wird bei jedem Befehl an den SMATER Webserver mitgeliefert, wodurch diesem ermöglicht wird, alle Gerätelisten und Aktionen den richtigen lokalen Webservern zuzuordnen. Der erhaltene API Schlüssel muss erstens im Alexa Skill (siehe 7. lambda.js) und zweitens im lokalen Webserver eingetragen werden (siehe 4.3.2 Punkt „`Einstellungen`“).

4.5.3 Gerätelisten

Der lokale Webserver aktualisiert ständig seine Geräteliste. Sobald diese eine Änderung vorweist, wird sie automatisch an den SMATER Webserver weitergeleitet. Dieser legt die Liste inklusive Benutzeridentifikationsnummer in seine Datenbank ab. Der Alexa Skill kann dann über den SMATER Webserver auf diese Datenbank zugreifen und mithilfe des API Schlüssels die Liste eines bestimmten Geräts erhalten.

4.5.4 Aktionen/Funktionen Ausführen

Der SMATER Skill kann den SMATER Webserver nutzen, um Befehle an den Raspberry Pi zu senden, ohne ihn direkt zu kontaktieren. Er führt für jeden lokalen Server eine Liste mit Aktionen. Dabei kann es sich entweder um einen Aufruf einer Funktion oder einen Befehl zum Schalten eines bestimmten Gerätes handeln. Die Liste wird ständig von den einzelnen Raspberry Pis abgefragt. Sobald eine Aktion erhalten wurde, wird sie ausgeführt und aus der Tabelle entfernt. Außerdem können andere Steuerungsmethoden den SMATER Webserver nutzen, um von überall Funktionen auszulösen.

4.5.5 API

Alle oben genannten Funktionen können über den Aufruf mit der Struktur „`http://smater.diebayers.de/?f=<functionsname>&<parameter1>=<parameter1_wert>&...&<parameterN>=<parameterN_wert>`“ ausgeführt werden. In folgender Tabelle sind alle möglichen Befehle und deren benötigten Parameter beschrieben.

Funktion	Parameter	Beschreibung
`deviceList`	`key`: API Schlüssel	Gibt die Liste aller Geräte für den angegebenen API Schlüssel zurück.
`obtainKey`	`user`: Benutzername `pass`: Passwort	Liefert für den angegebenen Benutzernamen und Passwort den passenden API Schlüssel, falls der Nutzer bereits hinterlegt ist.
`actions`	`key`: API Schlüssel	Liefert die Liste aller Aktionen für den angegebenen API Schlüssel und entfernt diese aus der Datenbank.

`update_list`	`key`: API Schlüssel `deviceList`: Liste aller Geräte	Aktualisiert die Liste der Geräte für einen bestimmten API Schlüssel.
`verifyKey`	`key`: API Schlüssel	Gibt zurück, ob der angegebene Schlüssel gültig ist.
`execute`	`key`: API Schlüssel `state`: Der zu setzende Zustand des Geräts `deviceId`: Die lokale Geräteidentifikationsnummer `r,g,b`: Der Rot- Grün- und Blau-Wert der Lampe, falls es sich um solch eine handelt	Fügt eine Aktion zur Schaltung eines bestimmten Gerätes für den angegebenen API Schlüssel hinzu.
`function`	`key`: API Schlüssel `funcId`: Die Identifikationsnummer der auszuführenden Datei	Fügt eine Aktion zur Ausführung einer bestimmten Funktion für den angegebenen API Schlüssel hinzu.

Abbildung 13: Tabelle aller API Funktionen des SMATER Webservers

4.6 Beispiel

Um darzustellen, was mit SMATER letztendlich ermöglicht worden ist, wird ein Beispielszenario durchgeführt.

Im Wohnzimmer eines Nutzers ist eine intelligente Deutgestenerkennung installiert (Steuerungsmethode). Außerdem hat er dort eine Philips Hue Lampe angebracht (intelligente Umgebung).

Nachdem alle Grundeinstellungen getätigt wurden (Einrichten des API Schlüssels, Einrichten der Hue Bridge und deren Lampen), möchte er, dass eines von zwei Dingen passiert, wenn er

auf die Lampe deutet. Entweder sie geht aus, falls Sie bereits eingeschaltet ist oder ein Sprachassistent frägt ihn, mit welcher Farbe die Lampe anschalten werden soll, falls sie ausgeschaltet ist. Das Einstellen dieses Verhaltens kann er einfach über das SMATER Webinterface tätigen.

Zu Beginn muss er in seinem Browser dem Link `http://smater.diebayers.de/devices` folgen, um auf die Bedienoberfläche von SMATER zu gelangen.

Als nächstes überprüft er, ob seine Wohnzimmerlampe unter dem Tab „`Geräte`“ bereits aufgetaucht ist. Falls nicht, muss er noch seine Hue Bridge mit SMATER koppeln, was durch den Button „Koppeln“ ermöglicht wird.

Jetzt ist der Nutzer bereit, die passende Funktion zu erstellen. Unter dem Tab „Funktionen“ klickt er auf den Button „Neue Funktion erstellen“.

In dem Feld „Anfragename“, gibt er den Text „Deute Wohnzimmerlicht“ ein. Somit kann er in Zukunft immer die Funktion zuordnen. Unter „Anfragetyp“ wählt er die Option „Gerätefunktion“ aus. Es erscheint eine neue Auswahl „Gerät/Funktion Auswählen“. Hier wählt er nun die Lampe „Wohnzimmerlampe“ aus und stellt die auszulösende Funktion auf „Farbe fragen / Aus“. Um abzuschließen muss er nur noch auf den Button „Funktion erstellen“ klicken.

Das SMATER Interface erstellt nun automatisch die Funktion und leitet den Nutzer zurück auf den Tab „Funktionen“, in dem „Deute Wohnzimmerlicht“ in der Tabelle erscheint.

Das Verhalten der Funktion wird durch einen Klick auf den Button „Ausführen“ getestet.

Zuletzt sollte noch der API Link der neuen Funktion kopiert und in die Deutgestensteuerung eingesetzt werden.

Sobald der Nutzer jetzt auf die Lampe deutet, ruft die Deutgestensteuerung den kopierten API Link auf und Alexa frägt den Nutzer nach der Farbe des Lichts, falls es aus ist, ansonsten macht SMATER das Licht aus.

4.7 Probleme bei der Realisierung

Als größtes Problem bei der Realisierung stellte sich das Übertragen von Text als Befehle an Alexa heraus. Die Implementierung ist in gut strukturiertem Java geschrieben, jedoch war der

Quellcode komplett unkommentiert und sehr umfangreich. Deshalb wurde es zum Problem, zusätzliche Funktionalitäten einzubauen, welche zum Beispiel das Übertragen synthetisierter Audiodateien umfasst.
Des Weiteren stellte sich heraus, dass es schwierig ist, den Namen bestimmter Geräte so in Sprache umzuwandeln, dass Sie Alexa sofort erkennen kann. Es wurde sich daher für eine alternative Lösung mit Identifikationsnummern entschieden. Dies erleichterte erstens das Erstellen der Sprachdateien und zweitens konnte Alexa diese sehr zuverlässig erkennen und in ganzzahlige Integer-Werte umwandeln.

5 Tests

Diese Arbeit wurde in enger Zusammenarbeit mit Alexander Biederers Arbeit „Deutgestenerkennung zur Steuerung von intelligenten Umgebungen“ geschrieben. In seiner Arbeit geht es um die Implementierung einer Deutgestenerkennung auf Basis einer einfachen Kamera- und Softwarelösung, welche mit SMATER kompatibel ist. Um die Nutzbarkeit des Systems ausführlich testen zu können, wird das funktionierende Interface von Biederers Arbeit benötigt, weswegen auf diese verwiesen wird. Dort kann ein konkreter Test gefunden werden, in dem beide Systeme zusammen in einer intelligenten Umgebung von echten Nutzern getestet wird.

6 Zusammenfassung und Ausblick

Nach einigen Vorbereitungen und Recherchen wurde ein Ziel für die Arbeit aufgestellt. Es sollte überprüft werden, ob es möglich wäre, bestimmte Steuerungsmethoden mit einem Sprachassistenten zu verknüpfen, um auf intuitive Art und Weise mit Smart Home Geräten interagieren zu können. Nach einigen Schwierigkeiten und Komplikationen wurde jedoch ein funktionierender Prototyp fertiggestellt. Dieser kann momentan nur die Standardfunktionen von Alexa aufrufen und mit Philips Hue Geräten kommunizieren. Allerdings zeigt er aber auch die Option ein Interface zu entwickeln, welches es ermöglicht, Sprachassistenten mit anderen Steuerungsmethoden zu kombinieren, um diese schließlich zusammen bestimmte Aktionen in intelligenten Umgebungen ausführen zu lassen.

In Zukunft könnte das Interface weitere Geräte unterstützen und nicht nur auf Lichter, sondern auch auf weitere Elemente intelligenter Umgebungen zugreifen. Beispiele dafür wären die Steuerung eines Fernsehers oder einer Musikanlage (Frage nach gewünschtem Sender / Playliste). Außerdem muss die Kommunikation mit dem Sprachassistent vereinfacht werden. Dies könnte eventuell durch eine Zusammenarbeit mit Amazon oder zur Not durch die Verwendung eines alternativen Sprachassistenten, wie Google Assistant ermöglicht werden. Dadurch würde zudem die Wartezeit für Rückfragen des Sprachassistenten deutlich reduzieret und eine aktivere Kommunikation zwischen Mensch und Maschine umsetzbar

gemacht werden. Zusätzlich würden die Kosten für ein Text to Speech API wie das verwendete von Voice RSS wegfallen. Zum Schluss ist zu bemerken, dass die Installation des jetzigen Prototyps sehr aufwändig ist und keine benutzerfreundliche Angelegenheit darstellt. Dieser Aspekt sollte in zukünftigen Versionen ebenfalls in Betracht gezogen werden. Eine mögliche Lösung wäre die Entwicklung einer passenden Hardware für das System.

7 Quellcode und Installation

Um den Quellcode des Projekts mitzuliefern, wurden unter http://smater.diebayers.de zwei Links eingerichtet.

1. GitHub-Repository mit dem im Folgenden erklärten Quellcode für das Projekt inklusive Installationsanleitungen.
2. Download eines Images für eine 32gb SD Karte um die komplette Installation zu vereinfachen. Das Passwort für den Dropbox Ordner lautet „SMATER2017“.

Im Folgenden werden die im GitHub Repository aufgeführten Dateien erläutert und deren Verwendungsort geschildert. Außerdem befinden sich im Repository noch weitere Anleitungen, um die Dateien auf einem eigenen System zu installieren.

- Quellcode
  - Amazon
    - Lambda
      - lambda.js

        Die Lambda Funktion die vom SMATER Skill aufgerufen wird und dessen Funktionalität beschreibt. ACHTUNG: Um diese Datei funktionsfähig zu machen, muss die APIKEY variable mit dem richtigen Schlüssel ersetzt werden!
    - Skill
      - skill.json

        Die .json Datei, um den SMATER Skill im Amazon Entwickler Portal zu erstellen.
  - Raspberry_PI
    - ALEXA
      - alexa-client

- companionService

  Ein kleiner zusätzlicher Webserver. Er speichert die Login Daten des Amazon Servers und leitet sie an die Java App weiter.

- javaclient

  Der modifizierte Alexa Java Client inklusive Quell Code

- wakeWordAgent

  Helfer für den Alexa Java Client zur Erkennung des Wortes „Alexa“.

- alexa-start.sh

  Eine Bash Datei, die automatisch beim Systemstart des Raspberry Pi aufgerufen wird und alle notwendigen Programme der Reihe nach startet.

- ComFile.txt

  Der Java Klient liest aus dieser Datei alle Befehle, die ihm der Webserver übergibt.

- py_cron

  - reloadPage.py

    Diese Python Datei wird von „alexa-start.sh“ aktiviert und lädt in einer Endlosschleife alle benötigten PHP Dateien auf dem lokalen Webserver, um mit dem SMATER Server zu kommunizieren und Befehle vom Alexa Skill zu empfangen.

  - StatusFile.txt

    Der Java Klient schreibt ständig seinen aktuellen Status in diese Datei.

- MySQL

  - database_structure.sql

    Diese .sql Datei enthält alle Befehle, um die vom lokalen Webserver benötigte Datenbankstruktur zu erstellen.

- Webserver

  In diesem Ordner liegt der Quellcode des lokalen Webservers inklusive aller PHP, HTML, CSS und JavaScript Dateien.

- Smater_Webserver
  - MySQL
    - database_structure.sql

      Diese .sql Datei enthält alle Befehle, um die vom SMATER Webserver benötigte Datenbankstruktur zu erstellen.
  - Webserver

    In diesem Ordner liegt der Quellcode des SMATER Webservers inklusive aller PHP, HTML, CSS und JavaScript Dateien.

8 Literaturverzeichnis

- tutorials-raspberrypi.de 2016 - *Amazon Alexa (Deutsch) auf dem Raspberry Pi installieren*
  https://tutorials-raspberrypi.de/raspberry-pi-amazon-alexa-deutsch-installieren/
- Philips 2016 - *Philips hue API Dokumentation*
  https://developers.meethue.com/philips-hue-api
- Coding the Amazon Echo (Alexa) 25.07.2016 - *How to create custom Alexa skills from scratch with AWS Lambda*
  https://www.youtube.com/watch?v=zt9WdE5kR6g
- Amazon Developer 2017 - *Build Skills with the Alexa Skills Kit*
  https://developer.amazon.com/docs/ask-overviews/build-skills-with-the-alexa-skills-kit.html
- Wikipedia 9.7.2017 - *Intelligenter persönlicher Assistent*
  https://de.wikipedia.org/wiki/Intelligenter_Pers%C3%B6nlicher_Assistent
- PHP Documentation Group 2017 - *Dokumentation von PHP*
  http://php.net/manual/en/
- 1&1 Internet SE 13.07.2016 - *Den Raspberry Pi als Webserver nutzen*
  https://hosting.1und1.de/digitalguide/server/konfiguration/einen-raspberry-pi-webserver-mit-lamp-einrichten/