Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Voice Controlled Smart Spaces

Florian Gratzer

TECHNISCHE UNIVERSITÄT MÜNCHEN

DEPARTMENT OF INFORMATICS

BACHELOR'S THESIS IN INFORMATICS

Voice Controlled Smart Spaces

Sprachgesteuerte Smart Spaces

Author	Florian Gratzer
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Marc-Oliver Pahl, Stefan Liebald
Date	December 14, 2016

Informatik VIII
Chair of Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, December 14, 2016

Signature

Abstract

Voice recognition systems are increasingly used. There is a variety of well known systems like Siri and Cortana, which are so called personal assistants. These assistants can be used to call numbers or to get weather forecasts. But voice recognition can be used for more. The voice recognition system developed in this thesis can be used to control so called Smart Spaces. In Smart Spaces, Smart Devices are used to interact with their environment These devices can be controlled via software. The system developed in this thesis can be used to interact with devices like lights, shutters or thermometers via voice commands. It is modularly designed to provide scalability, adaptability and extensibility. The functionality is implemented in different services using the Distributed Smart Space Operating System and its Middleware, the Virtual State Layer. The system can be easily modified via a Graphical User Interface. The system has a high reliablity. It can processes voice input in a short amount of time.

Zusammenfassung

Spracherkennungssoftware immer häufiger verwendet. Es existiert eine Vielzahl an bekannten Systemen, wie Siri oder Cortana. Die meisten dieser Systeme sind sogenannte persönliche Assistenten. Diese Assistenten erlauben es, Nummern zu wählen oder Wetterberichte abzurufen. Spracherkennungssoftware kann jedoch für mehr verwendet werden. Die Spracherkennungssoftware, die in dieser Arbeit entwickelt wird, kann dazu verwendet werden um mit sogenannten Smart Spaces zu kommunizieren. In einem Smart Space befinden sich Smart Devices, Geräte welche dazu verwendet werden um mit der Umgebung zu interagieren und mittels Software gesteuert werden. Das System, welches in dieser Arbeit entwickelt wurde ermöglicht es mit Lampen, Rollläden, Öfen oder Thermometern mittels Sprachkommandos zu kommunizieren. Es ist modular aufgebaut um skalierbar, änderbar und erweiterbar zu sein. Die Funktionalität ist dabei in verschiedene Dienste aufgeteilt, welche das Distributed Smart Space Operating System und seine Middleware, den Virtual State Layer verwenden. Das System hat eine hohe Zuverlässigkeit und kann Spracheingabe schnell verarbeiten.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last years, a variety of voice controlled software was released. Most of it is a so called personal assistant (PAs). It is mainly used on mobile devices like Smartphones or tablets. PAs can be used to fulfill simple tasks like calling a phone number, getting a weather forecast or getting the address of the nearest restaurant. A well known personal assistant is Siri. It is available on Apple's mobile devices like the iPhone or the iPad.

Voice recognition can be used in a variety of fields. One of these field is Smart Space Orchestration (S2O). S2O is the research field of orchestrating Smart Devices within a Smart Space. Smart Devices are devices, which communicate with their environment via sensors and actuators. They can be remotely controlled via a network [1]. These Smart Devices are usually controlled via ordinary input methods like remote controls or web interfaces. Also, they can be controlled via voice commands.

Different commercial solutions are available to enable voice control of Smart Devices. Two popular solutions are Amazon Alexa, which is available on a wireless speaker called Amazon Echo and Apple HomeKit, which allows iPhone/iPad users to use Siri to control Smart Devices. Both solutions have in common that they do not provide any functionality without an Internet connection. None of these two systems supports custom built devices. Instead, special (expensive) Smart Devices have to be bought to use Amazon Alexa or Apple HomeKit.

1.1 Goals of the thesis

The goal of the thesis is to develop a *Voice Control System for Smart Space Orchestration*. To reach this goal the Distributed Smart Space Orchestration System (DS2OS) [2] is used for Smart Space Orchestration. The developed system has to match voice input to actions within the Virtual State Layer (VSL). This thesis aims to discuss how this mapping can be realized. The system developed in this thesis has to include a subsystem, which allows non experts to configure this mapping. The system has to provide voice feedback for users.

1.2 Methodology

This thesis is divided into several steps. First of all, a problem analysis is done. DS2OS is inspected. Afterwards, the background of voice recognition and speech synthesis is analyzed. Then, different speech-to-text and text-to-speech engines are observed and evaluated based on the knowledge gained in the steps before. Hardware for voice controlled in Smart Spaces Orchestration is compared. In the system developed in this thesis, voice commands have to be mapped to actions within the VSL. Different possibilities were evaluated. This mapping has to be configurable by non experts. It is inspected how this can be done.

After the analyis, it is evaluated how a voice controlled orchestration of Smart Spaces is done by related work.

Based on the analysis and related work, a system is designed. The functionality is divided into subsystems. These subsystems provide services which uses the VSL for communication and to store context. An interface is specified for every service. This is done by context models.

When the design is completed, the services are implemented. First, the services which interact with users are implemented. Next, an adaption service is created to interact with the environment. When the previous services are functionable, they are connected by a service, which handles the mapping. At last, a Configuration Interface is developed.

After implementation, the system is evaluated to see if it fulfills the requirements specified in the analysis section.

1.3 Outline

In chapter 2, the background of this thesis is analyzed. Requirements for a Voice Control System for Smart Space Orchestration are identified and discussed. Section 2.1 introduces DS2OS. Section 2.2 and section 2.3 focus on the transformation of speech signals into text and vice versa. These section also introduce different software which can be used for this purpose.

Section 2.4 introduces the hardware used in this thesis. This chapter closes with a list of requirements for a Voice Control System for Smart Space Orchestration (section 2.7).

Chapter 3 compares different related work, starting with Apple HomeKit. Amazon Echo is introduced as well as offline voice control solutions for Smart Spaces.

Chapter 4 introduces the subsystems of a Voice Control System for Smart Space Orchestration.

Chapter 5 focuses on implementation details of the services introduced in chapter 4. Furthermore, it is discussed how each service can be configured.

In Chapter 6, the developed system is evaluated to see if it can be used for Smart Space Orchestration.

Finally, the thesis concludes with chapter 7 where the thesis is summarized and possible future work is discussed.

Chapter 2

Analysis

The analysis chapter is about different aspects in creating a Voice Control System for Smart Space Orchestration. The first section (2.1) introduces the Distributed Smart Space Orchestration System (DS2OS) [1].

The second part (section 2.2 focuses on the theoretical background of voice recognition. It treats with different steps and aspects of converting a speech waveform into information which can be used to control Smart Spaces. Different solutions which process voice input analyzed.

Section 2.3 introduces speech synthesis, which is the process of transforming a written text into its spoken representation. Different software which transforms a text into its spoken representation is discussed.

In the fourth part (section 2.4) the hardware used in this thesis are introduced. A Raspberry Pi is used for subsystems which are responsible for the interaction with users. Actuators and Sensor, used for interaction with their environment are controlled by Arduino boards.

The system has to map voice commands. Therefore, a mapping format is required. This mapping format is discussed in section 2.5. The mapping has to be configurable. This can be done in the Configuration Interface. This Interface is focussed in section 2.6.

This chapter closes with the requirements for a system which uses voice commands to control Smart Spaces (section 2.7).

2.1 Distributed Smart Space Orchestration System (DS2OS)

In this thesis, the Distributed Smart Space Orchestration System (DS2OS) is used for Smart Space Orchestration. In this section the relevant aspects of DS2OS are introduced. More about DS2OS and the background of Smart Space Orchestration can be found in appendix A.

DS2OS is developed at the chair of network architectures and services at the Technical University of Munich. The system provides a middleware for brokering and storing state context between devices within a Smart Space [2]. DS2OS is written in Java and consists of three blocks [1]:

- The *Virtual State Layer (VSL)* is a middleware, which acts as distributed operating system in a Smart Space [1].
- The *Smart Space Service Management (S2S)* manages the services available within the VSL [1].
- The *Smart Space Store (S2Store)* is a global service manager, which supports crowd-sourcing of services [1].

In this thesis, different VSL services are going to be implemented. The S2S is transparent for these services and the S2Store is not used. Therefore, only the VSL is discussed in this thesis.

2.1.1 Virtual State Layer (VSL)

The Virtual State Layer is a middleware. It is formed by self-organizing unstructured peers. The VSL manages the information of a Smart Space [3]. This information is called context. The peers are autonomous and called Knowledge Agents (KAs). The VSL is completely self-managed. It encapsulates the functionality from the KAs [3]. Services which produce context are decoupled from services consuming context. Each of these services is connected to exactly one KA. This KA is responsible for storing data and manages the retrieving of data from other services [3]. When services communicate with each other, they do not have to differentiate whether the other service is connected to the same KA or to another KA.

To describe the structure of context produced by services, *context models* (see section 2.1.1.1) are used. Each KA stores all required context models locally. This increases the resilience [3]. Furthermore, context models can be automatically synchronized. This can be done by a central *context model repository (CMR)* [1]. The content of the CMR is synchronized between all KAs. The context models are the interfaces of the services. Services communicate by accessing context of other services. The access can either be a read access via *get* or a write access via *set* [1]. Additionally, services can *subscribe* to *data nodes*.

Figure 2.1 shows, how services communicate over the VSL. On the bottom, there are Smart Devices which are connected to the VSL by so called adaption services. These services can be used by other (orchestration) services. These services do not directly communicate with each other, but use the VSL to connect to each other. The VSL allows to search nodes via its type identifier [1], which is the context model. This allows dynamic binding of services. The context of each service is described by exactly one context model [3].

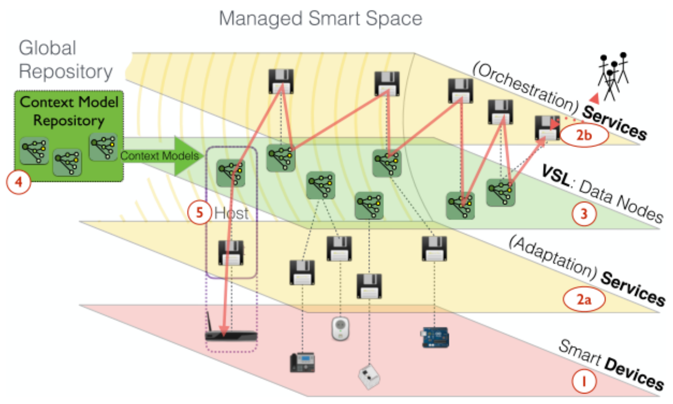


Figure 2.1: The VSL [3]

2.1.1.1 Context Models

Context Models describe the structure of context. Context is information within a Smart Space. Nowadays, different types of context models are used [4]. This thesis only describes the approach taken in the VSL. This model uses hierarchically structured type key-value pairs. Additionally, the key-value pairs include management meta data including access rights, version infos [4]. The key-value pairs (tuples) are hierarchically structured by their addresses and form a tree [4]. The different level are separated by “/” [4]. Each context model is represented in XML markup, which enables syntax validation [4]. This XML representation is located in the Context Model Repository (CMR) and is synchronized by all Knowledge Agents [3]. At start up, each service is bound to a context model. Context models are identified by their name and each context model has a type [4]. This type can be used for composing and sub-typing to create new context models. As a result an extensible type system is formed. This type system consists of three basic data types.

Each of these types can have restrictions [4]:

- */basic/number* represents a number and can be restricted by its upper and lower bound [4].
- */basic/text* represents a text and can be restricted by regular expressions [4].
- */basic/list* represents a list of context nodes and can be restricted by the minimum and maximum amount of entries and the type of the entries [4].

All types are either basic types or composed from existing context models. All types can have default values. These values are the values between the <> tags. As an example, the type Boolean can be represented as [1]:

```
<boolean type="/basic/number"
         restrictions="minimum=0,maximum=1">
         0
</boolean>
```

The context model has to be saved in the CMR. In this case the model of boolean and all following models are located in the sub-folder *derived* in the repository. With this type, a traffic light can be represented [1]:

```
<lamp type="/basic/derived" >
         <on type="derived/boolean">0</on>
</lamp>
```

```
<trafficLight type="/basic/derived" >
         <red type="derived/lamp">1</red>
         <yellow type="derived/lamp">0</red>
         <green type="derived/lamp">0</red>
</trafficLight>
```

More about DS2OS can be found in appendix A.

2.2 Voice recognition

The system developed in this thesis has to process voice commands. Voice commands are processed by analyzing their speech wave form. This wave form can be directly taken from a microphone. Also, a prerecorded sound file can be analyzed. To find suitable solutions for voice processing, the background of voice recognition is inspected. Voice recognition is performed in different steps. These steps are focussed in section 2.3.1.

The second part (section 2.2.2) introduces different software which is used for voice recognition. First, DragonDictate [5] is introduced. It was one of the first commercial voice recognition solutions. Next, cloud based solutions (Siri [6] , Cortana [7]) are introduced. In the end, a list of offline voice regnition engines are introduced. It is discussed whether they are suitable for the system developed in this thesis.

2.2.1 Steps in voice recognition

The human voice is a slowly timed varying signal with stationary characteristics, when examined over a short time of 5-100 ms [8]. This signal is dependent on speaker, speaking rate and acoustic conditions [8]. In speaker recognition, differences between two speakers is a used property. This differences can be problematic for a voice recognition system, which does not distinguish between different speakers. Different processing steps are required to allow speech-to-text engines to fast and reliably process voice [9]. Before a signal is processed it is filtered to reduce noise. Then, a level adjustment is done to facilitate the processing [9]. After this analog pre procedure, the first processing step is called **Digital Signal Processing (DSP)**. It generates a digital representation of the analog voice signal. The result contains too much information to be compared to known samples [10]. Relevant information is extracted by a second processing step to reduce the amount of data without loosing statistically relevant information. This step is called **feature extraction** (see section 2.2.1.1) [11]. The last step uses the extracted features. It tries to match the input with known samples. These samples can either be structural descriptions of single words or phrases or recorded audio samples from one or more speakers. This step is called **feature matching** (see section 2.2.1.2) [11].

2.2.1.1 Step 1: Feature Extraction

A voice signals varies, even when the said utterance is the same. Figure 2.2 shows the wave forms of the word “test” spoken six times by the same speaker.

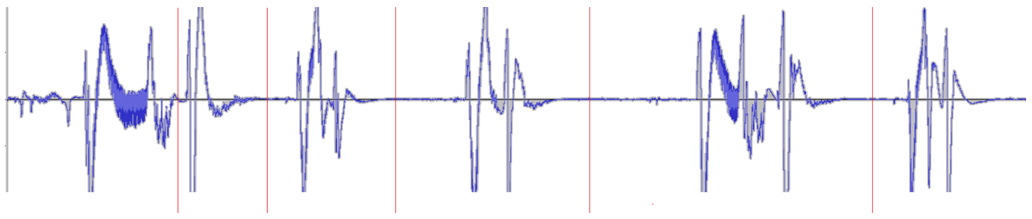


Figure 2.2: Wave forms of the word “test”

Therefore it is impossible to directly match a voice signal with known samples or statistical descriptions [10]. As a result, an additional processing step is required before a signal can be compared with words within a dictionary. This step is called *feature extraction*. The objective of feature extraction is to convert the speech wave form of a voice sample into a parametric representation to facilitate further processing [12]. This is done by extracting so called features. Features are parts of a signal, which can be used to distinguish between different sounds [10]. Features are independent from speaker, speaking rates, contents and acoustic conditions [8]. Features can be best extracted from the frequency domain of a signal [9]. Although analog filtering is usually done before, additional filtering is also possible in feature extraction.

A widely used technique is **Mel-frequency cepstral coefficients (MFCC)** [9–11, 13]. MFCC consists of the steps shown in Figure 2.3

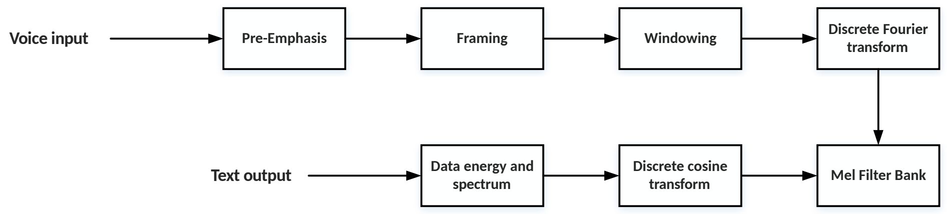


Figure 2.3: MFCC block diagramm (based on [9])

The first step is called *Pre-Emphasis*. Many sounds have a decreasing signal energy of about 6dB/octave. This results in a low signal to noise ratio (SNR) in high frequent parts of a signal. The *Pre-Emphasis* is a special filter, which helps to get a better SNR in these parts of a signal. *Pre-Emphasis* is usually realized with a differentiator of the form of equation 2.1 [9]. This results in lowering the low frequent parts of the signal. Afterwards the whole signal is amplified, so the result is the required raise of the SNR in the high frequent parts of a signal. The effect can be seen in figure 2.4.

$$y_n = x_n - ax_{n-1}, \quad 0.9 \leq a < 1 \tag{2.1}$$

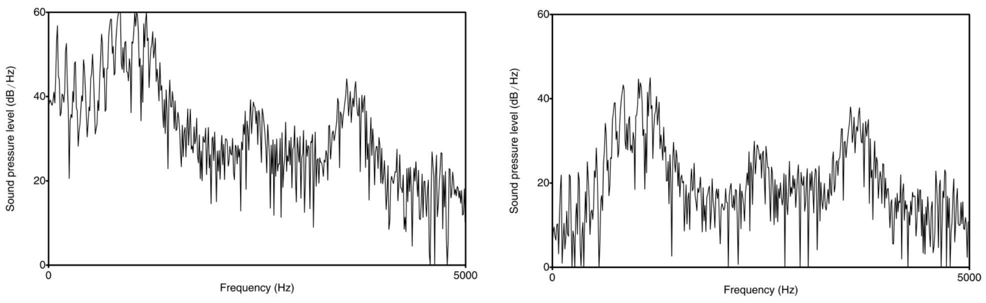


Figure 2.4: Before/After Pre-emphasis [9]

The second step, called *Framing* is used to split speech samples into small, non-overlapping frames with a length between 20 and 40 ms [11].

The third step is called *windowing*. Windowing is used to separate the frames from step two again, but in this step, the parts are overlapping to get better results [9]. MFCC normally uses an overlapping of 80% [9]. This means, that a windows doesn't start after the one before, but after 20% of the window before. Although a simple rectangular windows could be used, a Hamming window is used, which allows a smooth fade-in and fade-out [9]. The difference can be seen in Figure 2.5. This Hamming window with a size of T = 2τ is defined as $w(t)$, the result of the windowing is $y(t)$ and $x(t)$ is the input for the windowing. A typical value for τ is 10ms [9].

$$w(t) = \begin{cases} 0.54 + 0,46 * cos(\frac{\pi t}{\tau}) & -\tau \leq t \leq \tau \\ 0 & otherwise \end{cases} \tag{2.2}$$

$$y(t) = w(t) * x(t) \tag{2.3}$$

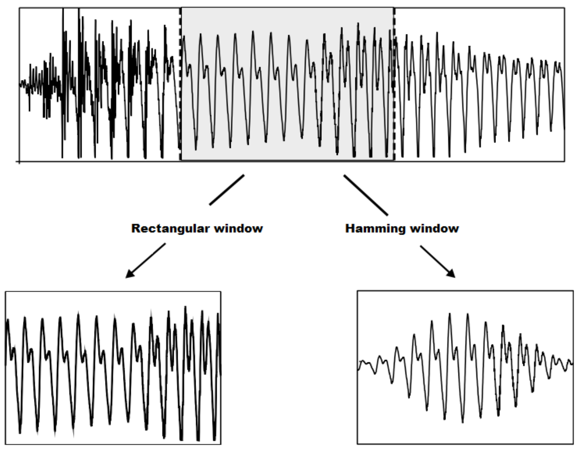


Figure 2.5: Windowing with rectangular window and Hamming window ([9] (modified))

After windowing, each window is transformed into the frequency domain using *discrete Fourier transform*. This step is necessary, because the signal can be analyzed better in the frequency domain [9].

After transformation, the signal has to be filtered, using a *Mel filter bank* [11]. The design of these filters is very complex and beyond the scope of this thesis and is therefore not discussed here.

After filtering, the signal has to be transferred back into the time domain. The result can then be represented as a series of vectors [9]. They can be matched in the next step - feature matching.

2.2.1.2 Step 2: Feature Matching

After *feature extraction*, the extracted features have to be matched with words inside a dictionary. There are different methods to implement this matching. A simple approach is via *pattern matching* [10]. Another approach is called *statistical voice recognition* [10]. The objective of a speech recognizer is to determine, if a word is part of the vocabulary and if it is inside the dictionary it has to be determined which word was spoken [10]. To match a spoken utterance to a text inside the dictionary, it is important that extracted features are used and not the wave form of the utterance [10]. Therefore the previous step *feature extraction* is required (see section 2.2.1.1).

Pattern Matching

With this method, *features* of the spoken word are compared with *features* of pre-recorded samples [10]. These samples can be single recordings or an average of different recordings of the same sample. The second approach results in a more reliable matching

[10]. To compare a sample with a spoken word, a value called distance is required. The distance has to be designed in a way that it is as large as possible for different words, but as small as possible for the same word, independent from speaker, voicing and acoustic conditions [10]. The Mel-Cepstrum is generally considered to be good fitting to be used to calculate the distance between two words [10, 11, 14]. It is independent from the intensity of the spoken word. The Mel-Cepstrum describes the form of the frequency domain independent from the base frequency [10].

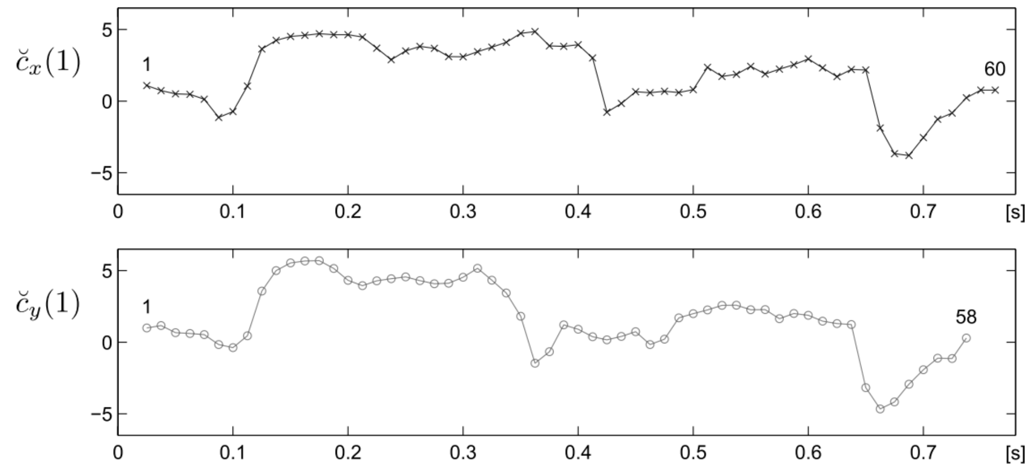


Figure 2.6: Mel-frequency cepstral coefficients of a recording (top) and a sample (bottom) both coefficient series are from the same word [10]

A remaining problem is, that the Mel-Cepstrum is still dependent on speaking rhythm and duration of the spoken word/phrase [10]. This can be seen in Figure 2.6. To overcome this, *dynamic time warping (DTW)* is used. DTW reduces the distance between two recordings of the same utterance significantly more than the distance of two different sound series [10]. The idea behind *DTW* is to reduces the distance between two samples by locally changing the time axis. The warping curve defines, which feature of a sample i has to be compared to which feature of sample j (equation 2.4).

$$\phi(t) = (i(k), j(k)) \tag{2.4}$$

The warping curve has three important properties: **monotony**, **local continuity** and defined **boundaries**.

- **Monotony:** The timed order of the features has to stay the same for each sample. Otherwise words with the same letters, but in different order could be matched to each other [10].

$$\begin{aligned} i(k) &\geq i(k-1) \\ j(k) &\geq j(k-1) \end{aligned} \tag{2.5}$$

- **Local continuity:** The time wrapping must not skip bigger parts (e.g. whole sounds) of a sample [10] Without this property, the word "**m**ech**ani**cs" could be matched with "many".

$$\begin{aligned} |i(k) - i(k-1)| &\leq g_x \\ |j(k) - j(k-1)| &\leq g_y \end{aligned} \tag{2.6}$$

- **Boundaries**: It is important that the start- and end time of a word are correct and part of the pattern matching. Otherwise it would be possible to match only parts of word which have (possibly) completely different meanings [13].

$$\begin{aligned} \phi(1) &= (1,1) \\ \phi(T) &= (T_x, T_y) \end{aligned} \tag{2.7}$$

After warping, the distance can be calculated. One method is to use the Mel-Cepstrum. This cepstrum is represented as a sequence of vector **X** from the recorded word and a sequence of vectors **Y** for each known sample [10].

$$\begin{gathered} \mathbf{X} = x_1 x_2 \ldots x_{T_x} \quad with \quad x_i = \begin{pmatrix} c_x i(1) \\ c_x i(2) \\ \vdots \\ c_x i(D) \end{pmatrix} \\ \\ \mathbf{Y} = y_1 y_2 \ldots y_{T_y} \quad with \quad y_i = \begin{pmatrix} c_y i(1) \\ c_y i(2) \\ \vdots \\ c_y i(D) \end{pmatrix} \end{gathered} \tag{2.8}$$

The distance is calculated as sum of all squared differences: [10]

$$d(x_i, y_j) = \sqrt{\sum_{n=0}^{D} [c_{xi}(n) - c_{xj}^2]} := d(i,j) \tag{2.9}$$

A word is matched, when the distance is smaller than the distance of all other words and smaller than a fixed value to avoid matching words outside the vocabulary.

Statistical Voice Recognition Statistical voice recognition is another approach to recognize spoken utterances from extracted features of a speech wave form. In contrast to pattern matching, the target is not to minimize the distance of each single word, but to maximize the probability that the spoken utterance is the same as the matched one [15]. The model behind statistical voice recognition is shown by Figure 2.7:

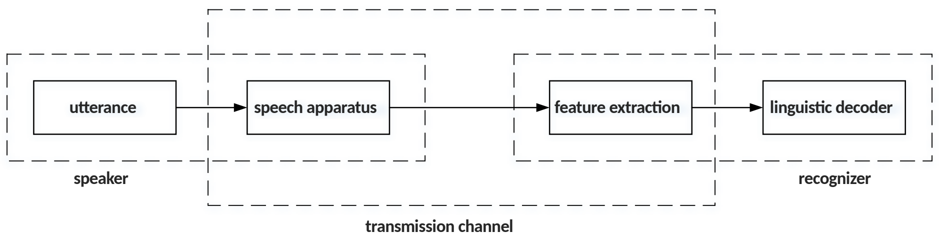


Figure 2.7: human speech generation process (based on [10])

First of all, an utterance is spoken. As mentioned before, the outcome is highly dependent on speaking rate and acoustic conditions [8]. This step is called *acoustic processing*. Afterwards, the output is transferred over a transmission channel, which is usually air, but it can also be additionally transmitted over wires (e.g voice recognition over phone). At this step, the speech signal can possibly be altered by noise. Afterwards a voice recognizer receives the signal and extracts the features of it (see 2.2.1.1). Then the voice recognition system tries to match the features to a word inside its dictionary. The target is to maximize the probability that the whole sample is exactly the spoken utterance. There are three steps involved in creating a statistical voice recognition system [15]:

1. Specify the recognition task: First of all the vocabulary has to be specified. This includes the pronunciation of words, the units which have to be recognized (single words, whole sentences, ...). The selection of the vocabulary is important, because it can help to facilitate the recognition, by maximizing the distance between words [15]. The vocabulary should be as small as possible without omitting required words or phrases [10].

2. Train the model: Each statistical voice recognition system has to be trained before it can be used [15]. This step is required to allow a system to create a dictionary. The result is a description of different ways of pronouncing words, a grammar model and a language model. The language model is used to describe, which combination of words are meaningful in the context of the voice recognition system [15]. This is required to determine the probability of each utterance.

3. Performance evaluation: The last step is an evaluation of the performance of a system. There are two error rates, which have to be evaluated: *word error rate* and *task error rate* [15]. The first one indicates the probability, that a word is incorrectly matched. The second one is the probability of performing a wrong action based on the result of the voice recognition [15]. It is possible, that one

error rate is acceptable, when the other one is not, so both error rates have to be checked for a reliable system.

As discussed in section 2.2.1.1, the result of the feature matching is a sequence of feature vectors $\mathbf{X} = x_1x_2\dots x_{T_x}$. The target is to find a series of sounds $\hat{W} = w_1w_2\dots w_{T_w}$ which has a higher probability to generate the sequence of $\mathbf{X}$ than every other known series of sounds [10]. A miss-matching happens, when at least one word differs from the said utterance. The rule to determine that is called *maximum-a-posteriori-rule (MAP-rule)* [10]:

$$\hat{W} = \underset{w \in Vocabulary}{\text{argmax}} \quad P(W|\mathbf{X}) \tag{2.10}$$

It is impossible to determine the probability $P(W|\mathbf{X})$ directly, but the probabilities $P(\mathbf{X}|W)$ and $P(W)$ are known after the training and evaluation phase of a voice recognition system . This probabilities can be used by *Bayes' Theorem* [10]:

$$P(W|\mathbf{X}) = \frac{P(\mathbf{X}|W) * P(W)}{P(\mathbf{X})} \tag{2.11}$$

The denominator is independent from the W and does not influence the decision, which word has the highest probability to produce the series of features $\mathbf{X}$. This results in the following equation to determine the word order with the highest possibility to produce the feature series $\mathbf{X}$ [10]:

$$\hat{W} = \underset{w \in Vocabulary}{\text{argmax}} \quad P(\mathbf{X}|W) * P(W) \tag{2.12}$$

To describe the probability that a word generates a sequence of features, *Hidden Markov Models (HMMs)* are used [10]. Markov Models are models which describe stochastic processes, where the probability that a state can be observed is dependend on previously observed states. HMMs are Markov Models where these states can not be directly observed. Every state has indicators which can be used to determine, that a HMM is in a specific state with a certian probability [9].

When developing a speech-to-text engine, which uses statistical voice recognition, the following three parts have to be specified for every scenario [16].

- **Acoustic Model:** An acoustic model is used to describe which features of a spoken utterance are matched to an according sound. An acoustic model is usually built for every language but not for every scenario. To create an acoustic model, deep linguistic knowledge of a language is required [17]. Consequently, acoustic models are reused unless a special scenario is not compatible with a general acoustic model of a language. Acoustic models are often part of a speech-to-text engine, but many speech-to-text engines offer more than one language. In this case, one acoustic model for every language is required.

- **Dictionary:** A dictionary contains a list of words and mappings from sounds to these words. It has to include all words which have to been recognized in a specific scenario. To increase the reliability of a speech-to-text engine, it is advisable to remove unused words from its dictionary to increase the distance between words. In many languages, a word can be pronounced differently, depended on its context (the word "the" for example). Therefore the same word can be matched with more than one sound series.
- **Language Model:** A language model restricts the matching results by defining which words can appear in the context of other previously recognized words [18].

2.2.1.3 Start- and end time detection

Another important aspect of speech recognition is the detection of the start- and end time of an utterance to avoid the recognition of noise when no one is speaking. There are different solutions to solve this problem.

Push to Talk A simple Solution is push to talk (PTT) [10], where a user of a voice recognition system has to push a button while speaking. PTT is more reliable than (semi-)automatic solutions, because a user knows best, when he wants to talk [10]. A disadvantage is, that a user always needs a device with a button around him. Otherwise he cannot start the recognition. This is not possible in many scenarios, like voice recognition in phone calls. Moreover, it is inconvenient for users, when they have to push and hold a button every time they want to use a voice recognition system.

Semi-automatic In semi-automatic processes, a user determines the start time by clicking or pushing a button. This helps a system in noisy environments to start the recognition process compared to automatic methods [10]. The end time has to be determined automatically. This can be done like in an automatic processes.

Automatic Different methods are used to detect the boundaries of an utterance automatically. They differ in their complexity (and therefore in the required processing power) and the applicability in noisy environments [10]. This thesis does not discuss processes which work in noisy environments. A simple approach for detection start- and end time uses *signal intensities, threshold values* and *timely constraints* [10]. This process has three parameters, the lower threshold value T, the minimal length of an utterance t_a and the maximum length of a pause within an utterance t_b [10]. The process works as following [10]: All sounds are recorded at every time. When the sound level is above T for more than t_a seconds, a system recognizes an utterance. This utterance ends, when the sound level is below T for more than t_b seconds. This approach is highly depended on the Parameters T, t_a and t_b and works reliable in scenarios with little noise, when configured properly [10].

2.2.1.4 Natural Language Processing (NLP)

The result of the feature matching process is a string (this can be a single word or a whole phrase). This string can then be directly mapped to an action. Another approach of matching utterances into information which can be processed further is *Natural Language Processing (NLP)* [19]. NLP uses a machine learning approach, which analyzes the semantics of a word. The idea behind NLP is to represent each word as a vector [19]. This vector describes a word itself and is independent from the vectors used for feature extraction (for example, the features of the words "house" and "building" are completely different, but the NLP vectors are close to each other) [19].

When words (or phrases) are directly matched with actions, two different rules have to be specified, when two synonyms are used for the same command (e. g. using "LIGHT ON" and "LAMP ON" to turn a lamp on). NLP uses the vector of a voice command isto determine the command which has to be executed. Therefore NLP ,can generalize a language model [19]. This results in a system which is able to replace words in a phrase by words with a similar meaning. This enables a more natural language processing than the simple static approach. As NLP is using a machine learning approach, it has to be trained with a vocabulary, before it can be used [19].

In NLP, different layers are used to determine the vector of a specific word [19]. Simplified, a layer takes the information of the layer below to derive information, which can be used by the layer above. The result of the top layer is the vector, which represents a word [19]. Two opposing approaches are used to generate the vector of a word. One uses about 50 dimensions and many layers to identify the vector of the word [19]. The other approach uses a 200-300 dimensional vector and only 3 layers of abstraction [19]. The result of both approaches is nearly the same, but depending on the available resources one of the two approaches is more suitable. The first approach can be trained faster, where the second approach uses less memory to operate [19].

2.2.2 Voice recognition software

A software, which converts a speech into text is called speech-to-text(STT) engine. A variety of STT engines is available. In section 2.2.2.1, DragonDicate is discussed as one of the first commercial STT engines.

STT engines are used for personal assistants which are able to perform fixed tasks. In section 2.2.3, Siri and Cortana are introduced. Siri is developed by Apple. It is available on Apple's mobile devices like iPhones and iPads [6]. Cortana developed by Mircosoft is available on Windows 10 [7]. Both are not able to perform offline voice recognition. When they are used, the recorded speech is compressed and sent to a server. These services processes the voice and sends the result back to the user devices [6, 7].

Section 2.2.4 introduces offline STT engines and discusses which of them is suitable for the system developed in this thesis.

2.2.2.1 Dragon Dictate

DragonDictate 30.000, developed by Dragon Systems was the first commercial general purpose large-vocabulary speech recognition system. DragonDictate is speaker dependent. It has to be adapted to every single speaker [5]. In the early nineties, other voice recognition 'systems were only capable of recognizing a fixed vocabulary of a few hundred words. They had to be trained with every single word within their vocabulary. In this approach, a complete acoustic model is generated for every single word [5]. When a speech input has to be matched, it is compared with all words in the dictionary and the word closest to the input is matched.

DragonDictate is a discrete-speech recognition system. This means that Dragon dictate requires a long pause of at least 250 ms to recognize a gap between two words.

DragonDictate, released in 1992, is able to recognize 25.000 words. With this large number of words it is impractical to train the vocabulary with every single word. Consequently, a different approach had to be discovered [5]. DragonDictate uses an approach where words are divided into phonemes. A phoneme is the smallest sound unit of a language [5]. This approach is also used by many modern voice recognition engine including CMUSphinx. These phonemes are then examined in context to the phonemes before and after an examined phoneme [5]. The vocabulary used by DragonDictate was inspected to find a subset of the 25.000 words, which contain most PICs. A subset of 8.000 words was found, which contained most PICs [5].

The acoustic model of DragonDictate was built by using a speech of a reference speaker including these 8.000 words. This reduced the amount of training words by over 66% [5]. It was a new method to train speech recognition systems at that time. DragonDictate used an acoustic model for a whole language not for every single word [5].

A remaining problem was the matching of a speech waveform to its phenome. The solution was to extract so called phonetic elements from the speech waveform of a word [5]. They are comparable to the features introduced in the analysis chapter (see 2.2.1.1). To get the result of a matching, Hidden-Markov-Models (HMMs) are used [5].

DragonDicate was developed over 20 years ago. Voice recognition engines greatly improved since then, so it cannot compete with modern voice recognition system. Nevertheless it has some important properties in common. First of all, it used an acoustic model for the whole language and divided words in different phonemes. Also the separation between feature extraction and feature matching was already part of DragonDictate.

2.2.3 Personal Assistants

2.2.3.1 Siri

Siri (short for **S**peech **I**nterpretation and **R**ecognition **I**nterface) is a digital assistant. It was originally developed as an App for iOS by Siri Inc. until this company was acquired by Apple in 2010. One year later, Siri was released as a part of iOS 5 for the iPhone 4s [20]. Today it is available on most mobile Apple devices including the current generation of iPhones and iPads [6].

Since release, the available functions of Siri increased. They include controlling the iOS media player, control telephone functions of iPhones and many more. Siri also provides voice feedback to acknowledge commands [6].

When Siri is used, the speech waveform is compressed and sent to a server, where it gets processed [21]. The result is then sent back to the device of the user. Therefore, an Internet connection is required to use Siri. This approach enables a Apple access to data of over one billion weekly requests [22]. As a result, the recognition algorithm of Siri can be adapted to fit best for most users. This results in a higher reliability for most users, compared to other voice recognition systems, which provide offline voice recognition.

2.2.3.2 Cortana

Cortana is a digital assistant developed by Microsoft, available on Windows 10. The name originates from an artificial intelligence in the X-Box first-person shooter Halo [23]. Cortana is integrated in the operating system of Windows 10 and can be controlled via voice input as well as written input. Cortana's features include sending emails and texts, open applications and find facts, places or other information [7]. There are different reasons why Cortana is not suitable for the system developed in this thesis. First of all, an Internet Connection is required for Cortana to be fully functional [7]. Also it is only available for Windows 10 which is not intended to be used on devices with a low power consumption like a Raspberry Pi.

2.2.4 Offline STT Engines

The following STT engines do not require an active Internet connection. All of them can process voice locally.

2.2.4.1 CMU Sphinx:

CMU Sphinx is an open source voice recognition toolkit developed by Carnegie Mellon University [16]. It includes two spech-to-text engines:

- Pocketsphinx is written in C and recommended for embedded devices. It takes less resources than Sphinx [16].
- Sphinx is written in Java and therefore can be directly integrated in a VSL service.

Both engine require a dictionary, an acoustic model and a language model. All three can be customized, but they there are also generic instances available [24].

Additionally, CMUSphinx includes Sphinxtrain, a tool which can be used to train acoustic models. To train a model, deep knowledge on the phonetic structure of the language is required. Even when this would be the case, it would take about 1 month to train an acoustic model (according to the developers of CMUSphinx) [17]. Consequently, a generic acoustic model of the English language is used.

Sphinx's dictionary can be created in a short amount of time (<1 day) when the scenario is fixed. Also the language model can be created without a large time extent (<1 day). In contrast, it is much more complicated and time intensive to create an acoustic model for a specific scenario. Therefore, dictionaries and language models are often customized to fit best for a specific scenario and acoustic models are not. Generic acoustic models are used instead [17].

For this thesis both versions were inspected. Pocketsphinx and Sphinx4 (the current version of Sphinx) provide a comparable detection rate and are easy to use. As Sphinx4 is written in Java, Sphinx4 is more suitable for this thesis.

2.2.4.2 Julius:

Another voice recognition engine is Julius. Julius was developed by a team from Nagoy Institute of Technology in Japan [25]. It is intended for large vocabulary continuous speech recognition [25], which is in contrast to the secenario of this thesis where a small vocabulary is used to increase the detection rate (see section 2.2.6). As a result of this design, the detection rate of Julius was below the detection rate of Sphinx. Therefore, Julius is not inspected further and will not be used in this thesis.

2.2.4.3 HTK:

HTK, short for Hidden Markov Model Toolkit is a toolkit to create and adapt Markov models. It can be used for speech recognition, but also for other scenarios like DNA sequencing [26]. It was developed at the Machine Intelligence Laboratory of the Cambridge University Engineering Department and was transferred to Entropic Research Laboratory Inc. which was later bought by Microsoft [26]. HTK is freely available to be used for academic purposes, but it is not allowed to redistribute a product which uses HTK [27]. In order to use HTK for voice recognition, the hidden Markov Model requires training [28]. The system developed in this thesis includes the possibility for users to change the dictionary. Therefore it is not reasonable to use a STT engine which requires training. Consequently, HTK is not inspected further.

2.2.4.4 Kaldi:

All of the above solutions use hidden Markov Models for voice recognition. In contrast, Kaldi is based on deep neural networks. It is written in C++ and available under Apache License. Like HTK, Kaldi needs to be trained with the vocabulary before it can be used. Consequently it is not used in this thesis.

2.2.4.5 Jasper:

Jasper is an open source platform for developing voice controlled applications [29]. Jasper provides access to STT engines including Pocketsphinx and Julius. Jasper also features cloud-based solutions like Google STT or AT&T STT [30]. The biggest advantage of Jasper is that it covers the access to the mentioned STT engines (and also to text-to-speech-engines). As a result, the STT and TTS engines can be easily exchanged, when Jasper is used. Jasper is easy to install and the voice recognition works as reliable as using the STT engines directly. Jasper also has two major drawbacks. Firstly, only Raspberry Pis Model B are supported (an unofficial release for Raspberry Pi Model B+ is also available) [31]. Secondly, Jasper is not design to be used by other software, but as front end. Therefore the effort to integrate Jasper is higher than using the accoring STT engine directly. As a consequence, Jasper is not used for the system developed in this thesis.

Out of all evaluated solutions, Sphinx4 fits best for the system developed in this thesis. It provides offline voice recognition. Sphinx4 is written in Java so it can be integrated in a VSL service directly.Sphinx is easy adaptable. The dictionary and language model can be exchanged without effort. Sphinx does not require any training of the vocabulary.

2.2.5 MOVI Arduino Shield

Another approach is to use a dedicated board for voice recognition. One of these boards is the *MOVI Arduino Shield*. It is manufactured by *Audeme* [32]. MOVI is a board, which is intended for speech recognition and sythesis with an *Arduino Uno* [32, 33]. It was funded via *Kickstarter*, a croudfounding platform [34]. It is also possible to use the MOVI Shield with other devices, including the *Intel Galileo* [32, 35]. Figure 2.8 shows the MOVI Arduino Shield.

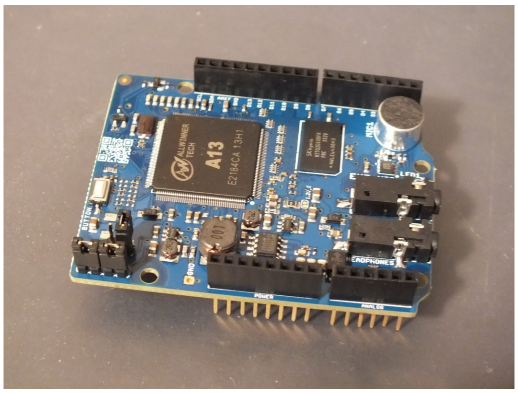


Figure 2.8: MOVI Arduino Shield [36]

The processing power of an Arduino (Uno) is not suffient for a voice control system. As a result an *Allwinner A13* [37] processor is used for voice recognition. The MOVI Arduino Shield includes a built-in microphone and also supports external microphones [32]. For voice output an additional speaker is required. MOVI uses the Arduino's serial interface to communicate with the Arduino. As a results, no additional pins are blocked by the shield [32]. Consequently, additional shields like an Arduino Ethernet shield [38] can be used for communication with other devices.

The manufacturer claims that MOVI is able to recognize 150 full-sentence voice commands without training [32]. For voice recognition, *CMU Sphinx* [16] is used. For speech output, eSpeak is used [32, 39]. Both engines are running on a *Debian* [40] system.

As this board was recently published, there are no independent tests available at the time this thesis is written. All of the features discussed here are claims of the manufacturer. When developing Smart Devices with MOVI, the Arduino environment is used, so there are no restrictions concerning the interfaces. This makes it suitable for a middleware like DS2OS. The reliability of MOVI is dependent on CMUSphinx. However, there are no information from the developers how to customize the dictionary and the language model. So it is questionable, how reliable the voice recognition of MOVI is in practice.

2.2.6 Dictionary, language model and acoustic model

Sphinx4 require a dictionary, a language model and an acoustic model to work properly. A dictionary contains all words, which should be recognized by a STT engine and the sounds a word consists of. A language model acts like a grammar in a natural language. It restricts the matching results by defining which words can appear in the context of other previously recognized words. An acoustic model describes which features of a spoken utterance are matched to an according sound. For all of them, there are generic instances, but they can also be created by oneself.

- **Dictionary:** A custom dictionary only contains words which are useful in the context of the system used. Therefore a custom dictionary for a Voice Control System for Smart Space Orchestration is by far smaller than a generic dictionary for the according language. Using a custom Dictionary increases the detection rate of Sphinx significantly. Creating a custom dictionary is possible in a reasonable amount of time (< 1 day).
- **Language Model:** A custom language model increases the detection rate by restricting the result by defining which words can appear in the context of other previously recognized words. Creating a custom language model is also possible in a reasonable amount of time (< 1 day).
- **Acoustic Model:** A custom acoustic model can increase the detection rate. According to the developers of Sphinx4, creating a custom acoustic model takes over 1 month and requires linguistic knowledge of the according language [41]. Consequently, custom acoustic models are not used by the system developed in this thesis.

For this thesis, a custom dictionary and a custom language model are used. The dictionary is editable in the configuration interface and the language model is modified when a new mapping is added (or an existing mapping is edited). In contrast, no custom acoustic model is used, because of the unreasonable amount of time it takes to create it.

2.3 Speech Synthesis

The system developed in this thesis has to provide users acoustic feedback. This is required to avoid dangerous situation. One of these situations is that a users unintendedly turns on an oven. The acoustic feedback provided in this thesis is a spoken representation of a text.

To find suitable solutions for this task, the theoretical background is analyzed first. Speech synthesis is done in different steps. These steps and further theoretical background is discussed in section 2.3.1.

The second part (section 2.3.2) introduces different software which is for speech synthesis. It is discussed whether the inspected solutions can be used in the system developed in this thesis.

2.3.1 Steps in Speech Synthesis

Speech synthesis is the task of transforming written text into its spoken representation [10]. This is analog to a person reading out a text. When designing an engine for speech synthesis, two major requirements have to be satisfied [10]:

- **Correct pronunciation of words:** The pronunciation of a letter is dependent on letters surrounding it. Therefore, rules have to be specified how to pronounce a letter depending on its surroundings [10]. Furthermore, the accentuation of syllables in polysyllabic words has to be correct. As a consequence, additional rules for polysyllabic words have to be specified [10].

- **Correctly placed sentence accents:** When a speech synthesis engine is used to read out whole sentences, some words have to be accentuated to indicate their importance. In long, complicated sentences it is useful to group words splitted by short gaps to foster the understanding of listeners [10].

Modern speech synthesis engine divide the process of speech synthesis into two processing steps [10]. The first step is voice independent and transforms the written text into a phonological representation [10]. A phonological representation consists of a sound series, an accentuation and a phrasing. The step to create these is called **transcription phase** [10]. The second step, called **phonoacoustic phase**, uses this phonological representation and an artificial voice to transform it into a speech signal [10]. Figure 2.9 shows this processing step. The transcription phase is discussed in section 2.3.1.1, the phonoacoustic phase is focused in 2.3.1.2.

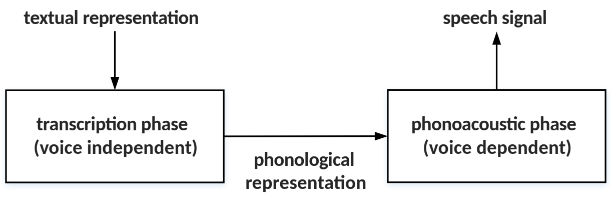

textual representation
speech signal
transcription phase (voice independent)
phonological representation
phonoacoustic phase (voice dependent)

Figure 2.9: Steps in speech synthesis (based on [10])

2.3.1.1 Transcription phase

Transcription is the first processing step in speech synthesis. The task of this step is to determine the phonological representation of written text [10]. Figure 2.10 shows the effect of this transcription phase. This transcription was done by the *"easy evaluation online converter"* [42].

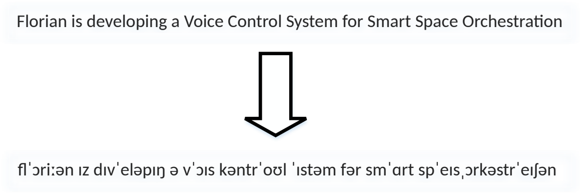

Florian is developing a Voice Control System for Smart Space Orchestration
flˈɔriːən ɪz dɪvˈeləpɪŋ ə vˈɔɪs kəntrˈoʊl ˈɪstəm fər smˈɑrt spˈeɪsˌɔrkəstrˈeɪʃən

Figure 2.10: Effect of the transcription phase

The same phonological representation can be used by different voices to generate a different spoken representation of the same written text [10]. Figure 2.11 shows the different steps required for this task.

First, a text is analyzed word by word and each word is transformed into a phonetic representation. This step is called *morphological analysis* [10]. Usually, this can be independently done for all single words. Most words have a unambiguous pronunciation, when its surrounding is not considered (the surrounding is considered in the next step) [10]. However, in the English language, there is also a small number of exceptions. The word *read (present thense)* and the word *read (past thense)* are written the same, but are pronounced differently. The result of this step is a list of words with their pronunciations independent from a sentence structure and other words [10].

After all sounds for all single words have been determined, the sentence structure is analyzed. This is done in several steps. The first step is a *syntactic analysis* [10]. It uses punctuation marks and grammatical constraints to determine the structure of a sentence. The result of this step is a sentence structure, which is purely based on the syntax of a sentence but not its semantics [10]. This sentence structure contains phase

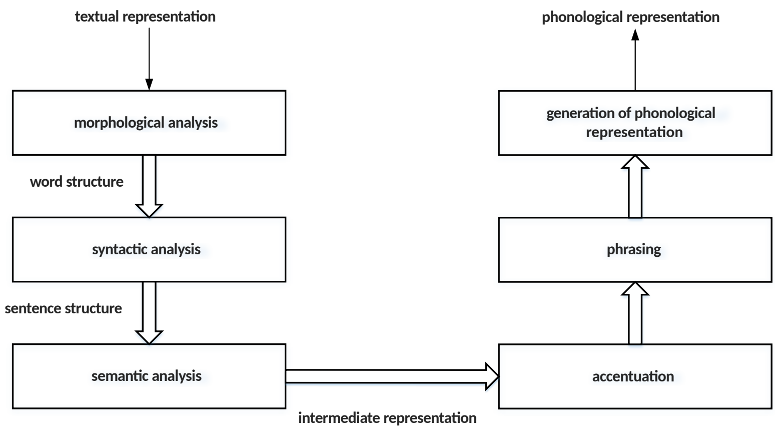


Figure 2.11: Steps in Transcription phase (based on [10])

boundaries and a distribution of accents. After a syntactic analysis, a *semantic analysis* is done, where words are analyzed in context to surrounding words [10]. The result of this semantic analysis is an intermediate structure. This structure can then be used by the *accentuation and phrasing* step to create a phonological representation, which can be used in the phonoacoustic phase [10].

Morphological Analysis It is not possible to directly transform a text into its phonetic representation [10]. Therefore, it has to be analyzed first. The first analysis step is called morphological analysis. It is done independently for every single word. It is not possible to analyze words as a whole [10]. In general neither the number of sounds in a word is the same as the letters, nor can a letter be matched to the same sound in all cases [10]. As a consequence, words have to be separated into smaller units. This units are called *morpheme* [10].

A *morpheme* is the smallest meaningful unit of a language [10]. In contrast to a letter, every morpheme has an unambiguous phonetic representation. The task of the morphological analysis is to split a word into its morphemes. To do this, linguistic knowledge is required [10]. Therefore, the details of a morphological analysis are not discussed further in this thesis. The result of the morphological analysis is a list of words with their phonetic representation(s).

Although a morpheme has an unambiguous representation, the result of the morphological analysis can be ambiguous [10]. The word *"visited"* for example can either be a verb ("I visited my aunt last week.") or an adjective ("The visited location was very nice."). Even though the meaning of these two words is different, the phonetic representation is the same. To decide, which meaning is given in a concrete sentence, a further analysis is required [10].

Syntactic and Semantic Analysis A *syntactic analysis* is required to determine the structure of a sentence [10]. This is done by using punctuation marks and grammatical constraints [10]. A syntax analysis also partly eliminates ambiguous results of the morphological analysis done before. The result of the syntactic analysis is a sentence structure, which can also be ambiguous [10].

To remove this ambiguity, this result is analyzed further. This is done by a *semantic analysis.* The task of this analysis is to remove ambiguity by finding out the meaning of a sentence [10].

Accentuation and Phrasing After the sentence structure has been analyzed, accentuation and phrasing is added.

Accentuation includes two parts: highlighting important words and highlighting parts of words. A correct highlighting of words can only be done, when the meaning of a sentence is known [10]. It the sentence *"Max is driving to Munich"*, different words can be highlighted to change the importance of the parts:

- **Max** is driving to Munich. (not his sister)
- Max is **driving** to Munich. (he is not flying)
- Max is driving to **Munich**. (not to Salzburg)

Often, the semantics of a sentence is unknown. Consequently, the syntax of a sentence is used to determine the words, which have to be highlighted [10]. In contrast, the parts of a word which have to be highlighted do not depend on the semantics of a word [10].

Phrasing is used to split long sentences in smaller phrases to foster the understanding of listeners. This separation is done based on the syntactic structure of a sentence [10].

2.3.1.2 Phonoacoustic phase

The phonoacoustic phase uses the phonological representation from the transcription phase to generate a speech signal. In contrast to the transcription phase, this phase is voice dependent. It is divided into two steps [10]:

- **Prosody Control:** The prosody control derives the prosodic parameters like base frequency, duration of sounds and intensity from the phonological representation [10].
- **Speech Signal Production:** The speech signal production creates the speech signal based on the phonological representation and the prosodic parameters [10].

Figure 2.12 shows the different steps required in the phonoacoustic phase.

Many different techniques are used to create a speech signal. There are approaches based on a model of the human speech production. Other approaches try to directly

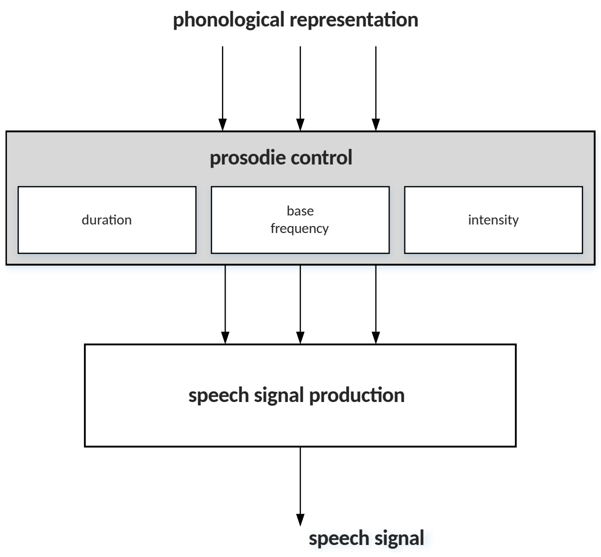


Figure 2.12: Steps in phonological phase (based on [10])

create the speech signal without taking care of the way how humans create a voice signal [10]. The details of these approaches are not in the focus of this thesis and are not discussed further.

2.3.2 Speech Synthesis Software

A software which converts a text into a spoken representation is called text-to-speech (STT) engine. In this section, two solutions are introduced and discussed.

2.3.2.1 MaryTTS:

MaryTTS is an open source TTS engine, developed by the *"Deutsches Forschungszentrum für Künstliche Intelligenz" (DFKI)*. Currently, 10 languages including German, American and British English are supported [43]. It is written in Java and can therefore be directly used by a VSL service.

MaryTTS uses a dictionary of known words for speech synthesis. When a word is in this dictionary, the phonetical representation is available for the according word. Otherwise, the phonetical representation is constructed using parts of known words. This consctruction is based on phonological principles [44].

The phonological representation is then transformed into a voice signal using a Hidden Markov Model based approach [43].

MaryTTS transforms a text into a Wave file which can then be played. The spoken text generated by MaryTTS is clearly understandable and sounds more natural than the spoken text of eSpeak. MaryTTS also supports different voices (male and female) for German and English. It is also possible to change the speaking rate and to use special effects, but none of this is needed to fit for this thesis.

2.3.2.2 eSpeak:

eSpeak is another open source speech synthesizer. It is written in C and available for Linux and Windows [39]. It supports over 30 languages including German and English [39]. Also, different male and female voices can be used. The amount of words per minute can be configured as well as the length of the gaps between two word. eSpeak can create a Wave file as well as speak the text directly using the default speaker of the operating system. All in all, eSpeak produces a clear speech, but the outcome is not as natural as the outcome of MaryTTS.

MaryTTS fits better for the system developed in this theis. It sounds more natural than all other observed solutions. It is written in Java and can be directly used by VSL Services.

2.4 Hardware used

The system developed in this thesis allows users to orchestrate a Smart Space via voice commands. In a Smart Space, computers with a large amount of processing processing power are not always present. Consequently, the system developed in this thesis has to be runnable on devices, which can be included into the environment without disturbing people. Two different types of hardware are required.

Firstly, Hardware is required, which can be used to build Smart Devices. These devices interact with their environments via sensors and actuators. In this thesis Arduino boards are used for this purpose. Arduino boards are boards with micro controllers and analog and digital pins to interact with actuators and sensor. Arduino boards are cheap (35 € [45]). The Arduino Uno and Mega can be programmed without an additioanl hardware component (programmer). They do not have any operating system. Consequently they are real time capable. This is important for some sensor and actuators which require strict timing (e.g individually controllable LED stripes). As a consequence they are optimal to be used to interact with the environment. For communication, extension boards are available. These boards allow Arduino Boards to communicate over Ethernet or WiFi. For this thesis any micro controller with general purpose input/output (GPIO) pins could be used. Arduino boards are chosen, because they can be programmed without any additional hardware (programmer). Also, jumper wires can be used to connect sensor and actuators with Arduino boards. Consequently soldering is not required. This enables easy prototyping. The Arduino platform is introduced in section 2.4.1.

Furthermore, hardware is required, which can be used for processing voice input and providing acoustic feedback. For this purpose, a Raspberry Pi is used in this thesis. A Raspberry Pi can be used to run Raspbian, a specially adapted Debian distribution. This system supports Java. DS2OS is written in Java. This makes the Raspberry Pi suitable as platform for Knowledge Agents. All Raspberry Pis have USB ports for microphones. The latest generation (Raspberry Pi 3) supports WiFi as well as Ethernet. It can be used to communicate with other Knowledge Agents or Smart Devices which have networking support. The processing power of a Raspberry Pi 3 is sufficient for DS2OS, MaryTTS and Sphinx. A Raspberry Pi costs about 35 Euros [46]. Alternatively, any other hardware which supports Java could be used for DS2OS, MaryTTS and Sphinx. The Raspberry Pi was chosen, because it is cheap and small.

	Arduino Uno	**Arduino Mega**
Microcontroller	ATmega328P	ATmega2560
Clock Speed	16MHz	
Flash Memory	32 kB	256 kB
SRAM	2	8
EEPROM	1	4
Digital IO Pins	14	54
Analog Pins	6	16

Table 2.1: Comparison of Arduino Uno and Arduino Mega

2.4.1 Arduino

In this section, the Arduino platform is introduced. It is an open-source platform intended for easy prototyping. It consist of two major parts [47]:

- **Software - the Arduino IDE:** a tool for writing code, which can be uploaded on compatible board
- **Hardware - the Arduino boards:** boards with a microcontroller and several digital and analog inputs and outputs. These are used to communicate with the environment via sensors and actuators.

Applications for the Arduino boards are programmed in C(++) in the Arduino IDE. Most of the boards have a common layout [33]. This increases interchangeability. Currently, there is a high number of compatible Arduino boards. Most of them have a low power consumption, but only a relatively small amount of memory and processing power.

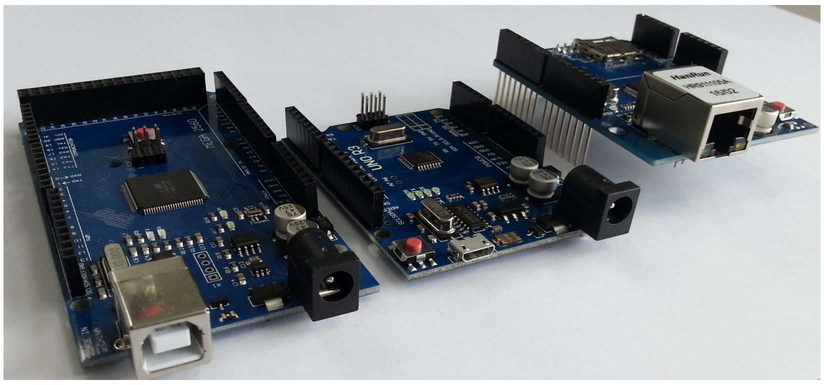


Figure 2.13: Arduino Mega (left), Arduino Uno (center), Ethernet Shield (right)

Two of the most used boards are the (cheaper) Arduino Uno and the (more powerful) Arduino Mega. Both use an 8-bit Atmel ATmega microcontroller. They can be seen in Figure 2.13. Further technical details can be found in the table below [48, 49]:

Typical applications of Arduino boards include shutter and light control or air quality and light sensing. Arduinos are able to send or respond to SMS or online messages like Twitter tweets [47].

Arduino boards can be extended by so called *shields*. Shields are extension boards. They can be plugged onto the Arduino to enable different features. Special shields are available for Ethernet (see Figure 2.13), WiFi, GSM, different motors [50]. Most of these shields are compatible with the Arduino Uno. Nevertheless it is recommended to use an Arduino Mega or similar boards for most of the shields because of the available resources (for details, the documentation of the according shield has to be checked).

2.4.2 Raspberry Pi

In this section, the Raspberry Pi is discussed. It is a single-board computer with the size of a credit card (15 x 10 x 2 cm) [51, 52]. It was developed to foster the education of computer science mainly in developing countries [53]. Moreover, Raspberry Pis are often used for applications, where energy efficiency is important and micro controllers cannot be used (because the power of microcontrollers is not sufficient or an operating system is required). This is the case in many Smart Spaces. A variety of frameworks and peripherals are developed and supported especially for Raspberry Pis. Additionally, the price of about 30 - 45 Euro [46] is below most comparable systems. Raspberry Pis are developed by the Raspberry Pi Foundation [54]. Currently, three generations of Raspberry Pis have been released.

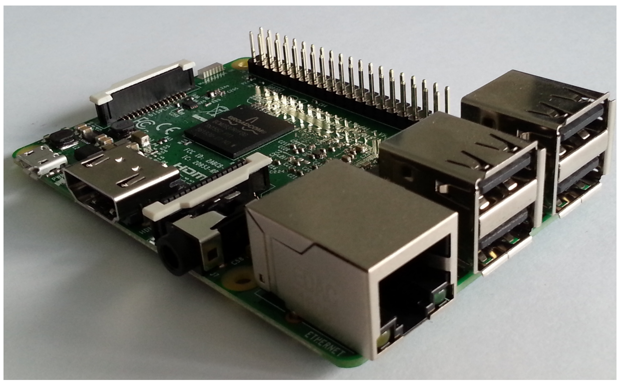


Figure 2.14: Raspberry Pi 3 Model B

The Raspberry Pi 3 Model B (current model in 2016) is based on a Broadcom BC2837 System on a Chip (SoC). It includes a Quad Core 1200 MHz ARM processor. It is shown in Figure 2.14. Technical details can be found in table 2.2 [55].

In this thesis, a Raspberry Pi is used for the voice recognition and speech synthesis subsystems. As operating system, a special Debian version, optimized for Raspberry Pis, called Raspbian is used. It has to be stored on a Micro SD card, because it is not possible to boot from an USB Storage in this version of the Raspberry Pi.

Processor Speed	1400 MHz (Quad Core)
RAM	1 GB (400 MHz)
Storage	Micro SD slot
Network interface	10/100 MBit/s Ethernet Port
USB Ports	4
GPIO Pins	40
Power Input	5V Micro USB Input Jack
WiFi & Bluetooth LE	Yes
Dimensions	85 x 56 x17 mm

Table 2.2: Technical detail of a Raspberry Pi 3 Model B

2.5 Mapping Format

The system developed in this thesis maps voice commands to VSL actions and feedback text. In this section, it is discussed how this mapping can be realized. It has to be conveniently configurable by non experts. Also it has to be easily processable by the system itself.

In section 2.5.1 it is discussed if it is useful to map multiple actions to a single command. Section 2.5.2 analyzes whether a seperation into an address and a value part is suitable for voice commands.

Furthermore, it is discussed, whether it is useful to use a key word to trigger a recognition. This trigger word is discussed in setion 2.5.3.

The resulting mapping format is introduced in section 2.5.4.

2.5.1 Commands per action

When commands are matched to VSL actions, they can be either matched to a single VSL action or to a list of actions.

- **One action per voice command:** When a command is only used for a single action it is clearer for a user what happens when he uses a voice command. When multiple devices are controlled at once it can happen that only a part of these actions are recognized by a user, whereas the other ones happens unintendedly. The chance of this behavior is significantly lower in a scenarios where only one action is mapped to each command.

- **Multiple actions per voice command:** Allowing users to map different actions to a single voice command increases the convenience for users when they want to control more than one device at once. One scenario where it is inconvenient to control every single device independendly with extra commands is when a user wants to turn off all lights in a building when he wants to go to sleep.

All in all, the first approach is more suitable for the system developed in this thesis. Some use cases (e.g. turning off all lights in a house) cannot be handled conveniently without allowing users to trigger multiple actions with a single command. The main problem of the first approach, that actions are triggered unintendedly, can be overcome by acoustic feedback. Consequently, multiple actions per voice command are allowed.

When multiple actions per voice command are allowed, two approaches are possible. Each action can be a single mapping. When two actions have to be triggered by the same command, they share the command. Another approach is to combine all mappings with the same voice command to a single mapping.

- **Combine all actions of a command to a single mapping:** When using a single mapping for all actions for a command, a user can clearly see all the actions mapped to a command. Also he can change the command for all of the actions at once. The drawback of this solution is, that it is not possible to edit the trigger phrase for a single action within a mapping with more than one action.
- **Use a single action per mapping and allow multiple mappings with same command:** When using a dedicated mapping for each command and action, the command of each actions can be changed without influencing other actions with the same command. The drawback of this approach is, that it takes longer to change the trigger phrase of all actions with a set of actions. Furthermore, it takes time to find all actions mapped to a command.

The approach to use a single mapping for a command is more suitable for the system developed in this thesis. The decisive advantage of this approach is that it is clear, which actions are mapped to a command at all time. The reason for this is that only one mapping can exist for each voice command. When the other approach is used, all mappings have to be inspected to find all actions which are triggered by a voice command.

2.5.2 Seperation of address and value part

There are two types of commands which are used in the VSL. GET actions extract context from the VSL. For these actions, no seperation is required. SET actions are used to store context in the VSL. These actions need an address where the context has to be stored and a value which is saved at that location. In this section, it is discussed if is useful to also use this seperation for voice commands. This means, that the address and the value part of SET actions are specified independently for the mappings.

- **Separation into address and value part:** It would be convenient for users to have a separation between address part and value part. This would allow users to select a command to control a set of devices and an extension to define the action on these devices.
- **No separation into address and value part:** The separation between address and value part is convenient, but it is not possible to realize this in a sufficient way (in a reasonable time), when a command is used for two different device types (e.g. shutter and light control).

The separation between address and value part can be conveniently realized for commands which are only mapped to a single device type. The mapping for this system is designed in a way, that it gives users freedom in how a command is mapped. It is an unreasonable effort to realize this separtion for devices in a convenient way. Therefore there is no separation of value and address part in the mapping format used in this thesis.

2.5.3 Trigger word

A special key word can be used to trigger the voice recognition. A trigger word decreases the amount unintended recognitions. In contrast, the number of unrecognized commands increased (dependent on the key word). A trigger word is not neccessary in scenarios, where push to talk is used (see section 2.2.1.3). In scenarios, with background talk or voice feedback a trigger word is completely required. In the test scenario, the command *TEMPERATURE*, was used to get the current temperature from a sensor. It is followed by the voice output *"The current temperature …"*. This feedback text triggered the same action again. To solve this, a key word is used to trigger the voice recognition.

2.5.4 Used Mapping format

In this section, the mapping format of the system developed in this thesis is introduced. This format is based on the results of the previous design choices.

In short, this is the result of the design choices of this section:

- Multiple actions per voice command
- Multiple actions within one mapping
- Voice output support is required
- GET and SET commands

The following mapping format is used in this thesis:

- **GET:** *COMMAND:GET;/address//..///addressN;FEEDBACK TEXT*
- **SET:** *COMMAND:SET;/address1 value1//..///addressN valueN;FEEDBACK TEXT*

Each mapping starts with a trigger word. Thistrigger word is not explicitly saved in the mapping. This part ends with ":". The next part specifies the type of action (GET or SET) followed by ";".

The next part consists of the addresses (and values for SET commands) of the nodes which have to be queried or set. When multiple addresses are accessed with a command, the addresses are separated by "//". The address and value part for SET actions are separated by " ".

The addresses are followed by ";", and the text which is spoken by a text-to-speech engine can be specified. The output text of GET commands can contain placeholders for the result. To mark the spot, %i is used to be replaced by the result of the %i-th GET request (starting with 0).

For example, with the mapping

TEMPERATURE:GET;/agent1/service2/temperature/;The current temperature is 0 degrees.

the system triggers the output "*The current temperature is 25 degrees*" when the phrase "temperature" is recognized and the result of the GET request is 25.

2.6 Configuration Interface

The system developed in this thesis maps voice commands to actions within the VSL. This mapping has to be configurable by non-experts. Therefore an interface is required which allows users to perform this task. Most people are not used to changing configuration files or to using command line interfaces. As a consequence, a Graphical User Interface has to be included in the system. This user interface should be platform independed if possible. Also it is advantageous when no special software is required on the devices of users.

The design of this interface of this interface is influenced by the used mapping format. This mapping format is introduced in section 2.5.

2.6.1 Type of Application

The Configuration Interface has to enable non expert users to configure the mapping between voice commands and VSL actions. Three different types of applications were considered for this user interface:

- **Web interface:** A web interface is a website, which is hosted on a server. A web interface is accessible by all devices with a connection to the web server (i.e all devices in the same network as the web server). Therefore it can be used on smartphones and computers, independent of their operating system. A user using the interface only needs an Internet browser to access the interface. Also, all data is stored centrally. Transferring and synchronizing the data between different endpoints is not critical. The fact that a web interface can be accessed by all devices with a connection to the hosting server is also a drawback. The web interface has to be secured in a way, that unauthorized access is not possible. Most web interfaces use a user management for this purpose. Also, the security is dependent on the server, where the web interface is hosted on.
- **Desktop Application:** A desktop application is an application which runs on a personal computer with an operating system. This application can be platform dependent, but there are also technologies, which can be used to create platform independent solutions. One of these technologies is Swing [56]. Swing is a Java framework which allows to design a platform independent GUI. An advantage of a desktop application compared to a web interface is that there are more possibilities to design a desktop application. The major disadvantage of a desktop application is, that it has to installed or stored on the target system. Also, a desktop application is not available on mobile devices.

- **Mobile Application:** A mobile application, short App, is an application for mobile devices like Smartphones or Tablets. Apps are suitable, when a user wants to configure a Smart Space on a mobile device. The major disadvantage of Apps is, that they are only available on mobile devices. Also, most Apps are written for specific mobile operating systems and can not be used on other operating systems. Nevertheless, there are technologies, that allow a cross platform development of Apps (e. g. Xamarin [57]). Currently, the most used mobile operating systems are Android and iOS, which have a market share of 98.9% [58].

The Configuration Interface of the system developed in this thesis is implemented as a web interface. This is the only solution where users do not have to install or download additional software on their devices. Also, all data is stored centrally, so no synchronization between different devices is required. Also, a web interface is accessible by computers and mobile devices. This is not the case for desktop or mobile applications. The web interface developed in this thesis does not have any permission management. Consequently, everybody with access the server can configure the Voice Control System for Smart Space Orchestration. This is not a problem in the environment of this thesis. When this system is later used in a productive environment, a permission management has to be implemented to avoid malicious behavior.

2.6.2 Functionality

Independed from the type of application, there are several functions, which can be included into the Configuration Interface for a Voice Control System for Smart Space Orchestration :

- **Mapping Configuration:** Most important for this interface is that it includes a function, which allows users to configure the mapping between voice commands and actions within the VSL. Therefore, a subsystem is required, which allows users to create a command consisting of one or more words out of a vocabulary and a set of VSL actions, which are triggered when the according command is recognized. Furthermore, the configuration interface has to provide a possibility to configure a text which is spoken by a text-to-speech engine to provide acoustical feedback and to present the values of sensors. Additionally, all currently present mappings have to be displayed in some way. The design of a mapping is discussed in section 2.5.

- **Dictionary Selection:** In a configuration interface, the dictionary of the speech-to-text engine can be changed. There are two possibilities to do this.
  - Choose from predefined dictionaries: When this solution is used, a user can choose from multiple predefined dictionaries. On the one hand, this dictionaries can be selected by experts. Experts know how to select a dictionary which fosters a high detection rate. On the other hand, a user has only little freedom to select a dictionary, which fits best for a special scenario when the approach is used.
  - Generate a custom dictionary: In contrast to the previous solution, a user can freely choose the words within a dictionary when this solution is used. This gives users much more freedom when selecting the vocabulary. A drawback of this method is, that the detection rate can be less compared to a dictionary created by experts. Also, the system has to generate a mapping from words to sounds and a language model, when the user is allowed to freely create a custom dictionary.
- **Device Discovery:** When configuring the mapping from voice commands to action within the VSL, the controllable Smart Device can be non-configurable. These devices can also be managed via a configuration file, where only an expert (i.e. an administrator) can add or remove devices from the Configuration Interface. Another approach is to allow users to discover new devices or to remove unused ones. The advantage of this approach is, that a new device only has to be plugged in, before it can be used. This can also be done by non-experts. Afterwards a user can discover this device and add it to the devices, which are accessible by the configuration interface. The major drawbacks of this solution are security concerns. So it is possible for everybody with access to the Configuration Interface to maliciously add or remove devices, when there is no proper permission management.
- **Command Recording:** The configuration interface can also include a function for recording a command. This can be an alternative to a dedicated service for voice recognition. With this function, voice commands can be executed without the need of a direct connection to the VSL on the input device.

- **Language Selection:** Another possibility is to allow users to change the language of the Voice Input Service. For each different language, (at least) one dictionary, one language model and one acoustic model have to be maintained. Also, the mapping has to be stored either independent for each language or a subsystem responsible for translation has to be used. When using a custom dictionary, the script which generates the mapping between words and sounds also has to support different languages. Additionally, the surface of the Configuration Interface has to be displayed in the new language.

The Configuration Interface can be used to configure mappings, edit the dictionary used by the speech-to-text engine, discover new devices and to record commands. The configuration of mappings is the main part of the interface.

Furthermore, the possibility of editing the dictionary is included, because different devices require different commands. As these devices are not fixed, a configurable dictionary is necessary. This dictionary is freely editable, because this is more convenient for users and evaluations showed that the detection rate is not signifficantly worse then the detection rate when using a fixed dictionary. Also, the interface contains device discovery to allow non-experts to include new devices into the Smart Space without requiring an expert to install it. Finally, the command recording function is used as an alternative input. It allows users to execute actions without requiring to install or download an extra service on their devices.

Language selection is not included in the web interface, because it would be an unreasonable effort to implement it in a way that it is convenient for users.

2.7 Requirements for a Voice Control System for Smart Space Orchestration

The developed system has to fullfill a set of requirements to be usable for Smart Space Orchestration. The following requirements are the result of the analysis.

1. **R1: Short response Time:** A Voice Control System for Smart Space Orchestration is not used when its response time is too high. The accapted response time depends on the scenario, where the system is used in. Accaptable resonse times for voice recognition systems are not discussed in literature. Therefore a response time is treated as accaptable for the system developed in this thesis, when the response time is below the time it would take to push a button on a remote. It is assumed that this remote is in reach, but not in the hand of a user. For the process of pushing a button on a remote, three seconds are assumed.
2. **R2: Offline functionality:** Smart Space Orchestration Systems are not necessarily connected with the Internet. Therefore it is required, that a voice recognition system is able to operate without Internet connection.
3. **R3: Runnable on low power hardware:** In a Smart Space, computers with high processing power are not always available. Therefore a Voice Recognition System for Smart Space Orchestration has to be runnable on a cheap (<50 €) low power consumption hardware.
4. **R4: Easy configuration:** Voice inputs of users have to be mapped to actions of a Smart Device. This mapping has to be configurable. Therefore the system needs a component which allows users to configure this mapping. The configuration subsystem has to be designed that non experts can use it.
5. **R5: Adaptability at runtime:** Smart Devices may not be available at all time. The system has to support adding and removing of Smart Devices at runtime. Also the mapping from voice commands to VSL actions has to be changeable at runtime.
6. **R6: Low error rate:** Voice Recognition Systems need an detection rate of at least 95% [10] to be treated as reliable. When voice commands are used for Smart Space Orchestration, it is that the rate of false positives is low, because these can lead to dangerous situations (e.g.: turn on an oven instead of turning on a lamp)
7. **R7: Transparency:** The system has to provide users information which command is mapped to which action. Otherwise the system seems unreliable.
8. **R8: Voice output:** Every recognized command has to be acknowledged to allow a user to react to wrong interpretations of his input. Furthermore the system has to provide voice output for presenting the data of sensors.

Chapter 3

Related Work

In this chapter, different related work is examined. First, section 3.1 introduces Apple HomeKit. It is a framework which allows Apple's mobile devices to control Smart Devices.

In section 3.2 Amazon's wireless speaker Echo is introduced. Echo is a speaker, which includes microphones and uses Amazon's voice recognition service Alexa to process voice commands.

Furthermore, three approaches are inspected which use voice commands to interact with their environment. Section 3.3) discusses a system developed by a German computer magazine called "c' t". It uses a voice recognition software on a Raspberry Pi to control Phillips lamps. In section 3.4, a Smart Alarm Clock using a Raspberry Pi is inspected. This alarm clock was designed for an IoT contest organized by element14. Both system can be used without Internet connection.

The chapter closes with a comparison of related work.

3.1 Apple HomeKit

Apple HomeKit is a framework which allows Apple's mobile devices (iPads, iPhones) to control Smart Devices. This can be done via the HomeKit Mobile App [59]. This App can control this Smart Devices via button. Also, Siri is supported. Therefore voice commands can be used as input.

Today, there is a variety of devices available. This includes devices heating and cooling systems, locks, shades and different sensors [59]. A drawback of HomeKit is, that it cannot be used for custom built Smart Devices. Instead, a set of licensed products has to be used. These devices are more expensive than home built devices (e.g $\tilde{2}$00 € for 3 remote controlled Philips Hue lamps [60]). These devices are available from over 50 different manufacturers [59]. All products are reviewed by Apple [59]. Compatible devices are labeled with an according sticker.

Each HomeKit device is identified by an eight digit code. To add a device to the set of controlable devices, this code has to be scanned by the HomeKit App [61]. Then, the connection establishment is done automatically [61]. Afterwards, additional steps may be required, depending on the device.

HomeKit manages its devices in rooms. Also it can control multiple devices with a single command (voice command or button) [61]. It is also possible to define task, which are trigger in the future. For example, HomeKit can be used to automatically turn on the heating system every morning at 7 a.m. [59].

3.2 Amazon Echo

The Amazon Echo is a wireless speaker with built-in microphones, which allows users to control the speaker and its surrounding via voice commands [62]. The speaker is shown in Figure 3.1.

Figure 3.1: Amazon Echo speaker [63]

For voice control, a service called Alexa is used. Alexa can be used to control the music, which is played on Echo speakers. It can also perform simple requests like setting a timer and retrieve arbitrary information from a search engine [62].

Additionally, Amazon provides an App store, where developers can contribute to develop Apps for different purposes. Echo provides an interface to interact with Smart Devices. One of the supported devices are Phillips Hue lambs. Unfortunately, Amazon only provides interfaces for proprietary protocols so they do not foster the development of self made Smart Devices or middleware like DS2OS. [62].

3.3 Controlling Phillips Hue Lamps with Jasper

The *"Magazin für Computer Technik"* (*c't*) developed a system to control Phillips Hue lamps [64] via voice commands [65]. They used a Raspberry Pi as platform for their software. They used the following hardware setup [65]:

- 1 Raspberry Pi Model B [54]
- 1 Micro USB power supply
- 1 4GB SDHC memory card
- 1 Edimax N150 Wi-Fi Nano USB Adapter [66]
- 1 Akiro Kinobo USB microphone [67]

In their setup, they used a software called Jasper. Jasper is a platform for developing voice controlled applications, written in python [29]. It can be used as front end for different speech-to-text engines like Pocketsphinx or Google STT. Multiple text-to-speech engines like eSpeak or MaryTTS [30] are also supported. Jasper is designed to be used on a Raspberry Pi. Neverthelessit can be used with any Linux system and also on Mac OS [30]. Jasper maps voice commands to any actions, which can be performed on the target system (e.g on the Raspberry Pi). Therefore, the functionality is separated into so-called modules. Jasper distinguishes between standard modules and notification modules. Standard modules are modules, where a user starts the interaction. For example, these modules can be used to control lamps within a room. In notification modules, the interaction is started by Jasper itself. For example when Jasper is used to notify a user, when a new mail arrived. Some modules are already included like a control module for the media player *Spotify* [68] or a Facebook Notification service [69]. Additionally 14 third party can be downloaded on the Jasper homepage [70].

In the system of the *c't*, the developers chose to use a pre-configured Raspbian (a Debian based Linux distribution developed for Raspberry Pis). This image is provided by the developers of Jasper [65]. In their setup, Pocketsphinx was used as voice recognition engine and eSpeak as speech synthesis engine. Both work without an Internet connection.

In order to control Phillips Hue lamps, a standard module was developed. These lamps are controlled via a Hue Bridge, which can be controlled over network [65]. This bridge provides a *Representational State Transfer (REST)* interface to interact with the lamps. The developers of the *c't* matched voice input to REST commands.

3.4 Voice Controlled Alarm Clock

Frederick Vandenbosch built a Voice Controlled Alarm Clock for the *element14 Pi IoT Smarter Spaces Design Challenge*. With his project, he won this challenge. He created a system, which processes voice commands using a setup based on a Raspberry Pi [71].

To process voice commands he needed a voice control engine, which can be easily customized and works offline. Therefore, he decided to use *Pocketsphinx* [16] as speech-to-text engine. Furthermore, he uses a tool provided by the developers of Pocketsphinx, called lmtool [72] to create a custom dictionary and custom language model [73].

For speech synthesis, a software called *flite* is used. The main reason to use this text-to-speech engine is that it can be executed from the command line and it is optimized for embedded devices like a Raspberry Pi [74].

The Raspberry Pi running these two engines is also connected to a small display. It is integrated in a case to look like an alarm clock. The alarm clock can be seen in Figure 3.2.

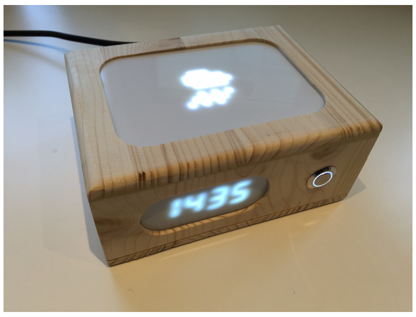


Figure 3.2: Smart Alarm Clock [75]

With this alarm clock, he is able to interact with several Smart Devices he has built himself, including a cat feeder and a tower light. On his project page, he describes how he built this Smart Devices and how this devices have to be connected [76].

On the software side, a speech-to-text engine was used, which does not require Internet connectivity. Therefore this part of his solution is comparable to the voice recognition part of the software developed in this thesis. Furthermore, a custom dictionary and a custom language model was used. In contrast to the system developed in this thesis, no middleware is used. As a result, it is likely that different parts of his software have to be changed, if the interface of a Smart Device gets changed. Furthermore, the mapping between commands and actions is fixed. Consequently, it cannot be configured by non expert users. Therefore it is not possible to exchange parts of his setup as easy as in the system developed in this thesis.

A demonstration of his project can be found on YouTube [77].

3.5 Comparison

In this section, it is evaluated, if the solutions introduced in this chapter fulfill the requirements specified in the analysis chapter (see section 2.7). It is not possible to evaluate all requirements specified in this section, because the information, whether the given solutions fulfills these requirements is not given for all solutions. Therefore, the following requirements are evaluated: Offline functionality, runability on low power hardware, easy configuration, adaptability at runtime and voice output.

	HomeKit	Alexa/Echo	c't	Smart Alarm Clock
Offline Functionality	- -	- -	++	++
Power Consumption	++	++	++	++
Easy Configuration	++	++	- -	- -
Voice Output	++	++	++	++

Table 3.1: Comparison of related work

- **Offline Functionality:** When Siri is used as input for HomeKit, an Internet connection is required. This is also true for Amazon Echo. The system from the c't does not require an Internet connection to work properly. The Smart Alarm Clock also provides offline functionality.
- **Runability on low power Hardware:** Siri runs only on Apple Devices like iPhones and iPads. These do not have a high power consumption. Alexa runs on special Speakers distributed by Amazon. This speakers can be used for Smart Space Orchestration. The solution built by developers of the c't and the Smart Alarm Clock use a Raspberry Pi. A Raspberry Pi is often used for Smart Space Orchestration and has a low power consumption.
- **Easy Configuration:** Supported devices can be easily added to the controllable devices when Apple HomeKit is used. Apple HomeKit allows users to define the location of devices and so the command can be partly customized. Nevertheless, it is not possible to freely choose the command which is used to control a certain Smart Device [61]. For Alexa, Amazon provides a Mobile Application which allows users to configure the mapping from commands to actions [62]. MOVI, the system developed by c't and the Smart Alarm Clock can only be configured by changing parts of the source code. This cannot be done by non expert users.
- **Voice Output:** All systems introduced in this chapter provide speech synthesis.

In conclusion, it can be seen, that all compared do not to provide an interface, which allows users to add custom devices without effort. When HomeKit or Echo are used, supported device can be added without an effort, but this is only possible for this set of proprietary hardware. They fail to provide an interface which can be used to communicate with custom devices. The other examined solutions provide functionality, comparable to the system developed in this thesis, but they can only be configured by users with programming skills.

Chapter 4

Design

The design chapter focusses on the design of the system developed in this thesis. The functionality is divided into a list of subsystems. Each subsystem offers a service. These services use the VSL of DS2OS to store data and for communication. In this chapter all developed services are introduced. As discussed in section 2.1, services do not communicate directly with each other. The communication is handled by the VSL. Most of the services developed in this thesis provide context for other services. Therefore a context model is required for each of these services. These context models are the interface of the according service. They are discussed in the section of the according service.

The system has to process voice commands. Therefore, subsystems are required which convert voice input into text. The voice input can either be a live recording from a microphone or from a pre-recorded audio file. Consequently, two subsystems are required for this task. To provide exchangeability, both subsystems have to provide the same interface. The service, which uses the microphone as input source is discussed in section 4.2.1. The service, which processes pre-recorded file is focussed in section 4.2.2

The system developed in this thesis has to be compatible to services which do not use voice commands as inputs. Therefore, a subsystem is required, which provides the same interface as the two subsystems introduced above. This service processes text input in the same way as the services above process voice. This service is discussed in section 4.2.3.

Furthermore, voice output has to be provided. This functionality has to be seperated from the voice processing subsystems. This allows the system to provide voice feedback by other devices than the devices which process voice input. Consequently an additional subsystem is required to provide acoustic feedback. It is discussed in section 4.2.4.

Voice commands have to be mapped to VSL actions. It has to be possible that a single service manages the commands of multiple Voice Input Services. Therefore, an additional subsystem is required. This subsystem subscribes to all available Voice Input Services. When a voice input is recognized, the according actions are triggered. Also, the acoustic feedback has to be triggered by this service. Voice Input Services and Voice Output Services can be found by their context model. The mapping service is focussed in section 4.2.5.

A subsystem is required, which allows users to configure the mapping between voice commands and VSL actions. This subsytem is a web interface. It can also be be used to change the dictionary and to discover new devices. Furthermore, it can be used to record commands which saved in a wave file. The Configuration Interface is discussed in section 4.2.6.

Finally, a subsystem is required to test the subsystems above. In this thesis, a simple adaption service is used, which controls an LED and uses a temperature sensor to sense the temperture. The design of this service is introduced in section 4.2.7.

The following services are provided:

- A service which provides an **interface to Sphinx** and uses a microphone as voice input source
- A service which provides an **interface to Sphinx** and uses a Wave file (sound file) as voice input source
- A service with the same interface as the first service, which **processes text** instead of voice input
- A service which provides an **interface to MaryTTS**
- A service which **processes voice commands** from the first three service, executes VSL actions and **provides feedback** via the fourth service
- A service which interacts with a Smart Device with a **temperature sensor** and an **LED** to test the services above.
- A **configuration interface** to configure mappings between commands and VSL actions.

Figure 4.1 shows the division into subsystems:

4.1 Service Interaction

All subsystems dveloped in this thesis are services which use the VSL for communication. Figure 4.2 shows the interaction of the different services. The services do not communicate directly with each other. Instead, all services communicate with their local Knowledge Agent and the communication between Knowledge Agents is transparent

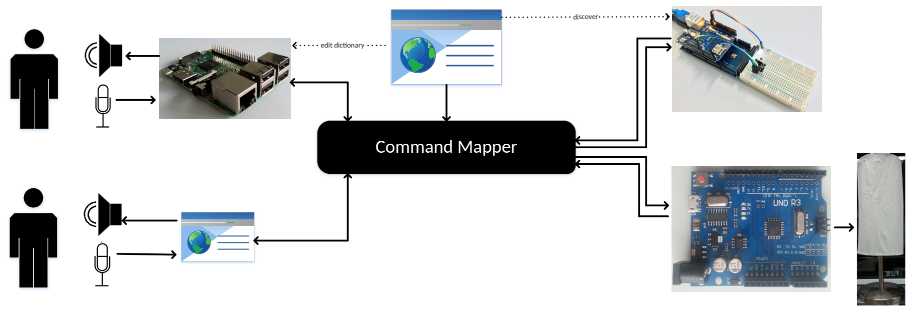


Figure 4.1: Service Coupling

for all services (see section 2.1. Therefore, all services have to be connected to a KA. This KA has to be started before the according service.

Voice commands are mapped to VSL actions. This is done by the Voice Mapping Service. These services subscribe to all Voice Input Services. Voice Input Services are searched via their context model. When a new command is recognized, the according action is triggered by the Voice Mapping Service. Therefore, the according service, which performs the action has to be started before the action can be executed. Furthermore, the Voice Mapping Service triggers the output of a feedback text. This spoken output is provided by all running Voice Output Services. These can also be searched by their type.

The services are described in detail in section 4.2.

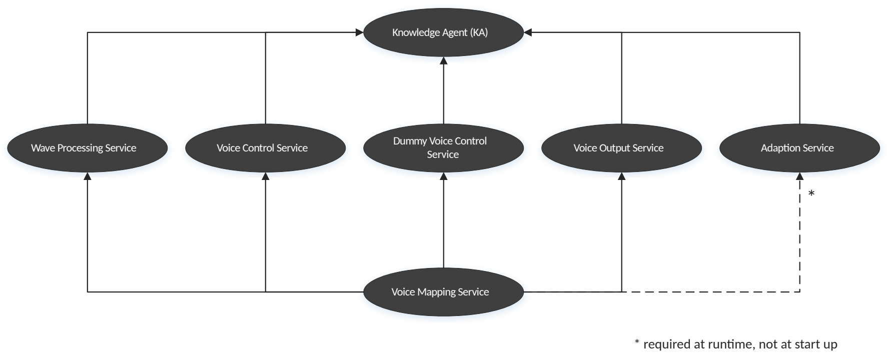


Figure 4.2: Service Dependencies

4.2 Services of the Voice Control System for Smart Space Orchestration

4.2.1 Voice Input Service

This service is responsible for processing voice input. It uses Sphinx4 from the CMU Sphinx suite to perform the transformation of voice commands to text. For this service, the default microphone of the system hosting it is used. As discussed in section 2.2, a dictionary, a language model and an acoustic model have to be specified. All three artifacts can be specified in a configuration file (see section 5.3). The used dictionary and the used language model can be created and modified via the Configuration Interface.

Acoustic models are provided by the developers of CMUSphinx. These acoustic models are currently available in 10 different languages including German and (US) English [41]. By default, an acoustic model for US English is used, but as mentioned above, this can be changed in the configuration file of this service.

To allow other services to process the textual representation of a voice command, a regular node is registered. This node uses the following context model. This context model is the interface of this service.

```
<voiceInput type="/basic/composed">
  <lastUtterance type="/basic/text" writer="" reader="*">
</voiceInput>
```

The service provides the last recognized utterance as text. This text can be read by every service, Only the the Voice Input Service itself is allowed to write it.

4.2.2 Voice File Input Service

This service processes pre-recorded voice commands. For this purpose, Wave files are processed by Sphinx4. It is not useful to process fixed Wave files with known commands (Fur this purpose, a dummy Voice Input Service, which processes text can be used (see section 4.2.3)). Instead, this service is intended to be used by other services, which allow users to record sound files. The file, which is processed can be specified in the configuration file of this service (see section 5.4). This file can be overwritten before the voice recognition is triggered. One service which triggers this service is included in the configuration web interface, which can be used to record audio files in a web browser (see section 4.2.6.4). To foster exchangeability with the *Voice Input Service*, the *Voice File Input Service* has to provide the same interface as the *Voice Input Service*. Additionally an attribute which triggers the processing of a recognition is required. Therefore it extends the model from the service above. The resulting model is shown below.

```
<voiceFileInput type="/voice/voiceInput">
  <trigger type="/basic/boolean" writer="*" reader="">
</voiceFileInput>
```

The service provides the last recognized utterance as text. This text can be read by every service. Only the the Voice File Input Service itself is allowed to write it. The service triggers, when the node “trigger” is written by any service.

4.2.3 Text Input Service

The system developed in this thesis is designed to process voice commands. Nevertheless, it has to be compatible to other systems which use different forms of input. Therefore a service is included, which processes text input instead of voice commands. This service uses the same context model as the *Voice Input Service*. Consequently, it is transparent for the *Command Mapping Service*, if it processes voice or text input. In contrast to the *Voice Input Service*, this service is not limited to the words in the dictionary, but words outside the vocabulary of the *Command Mapping Service* are not mapped to any VSL actions.

The context model used by this service looks like following:

```
<voiceInput type="/basic/composed">
  <lastUtterance type="/basic/text" writer="" reader="*">
</voiceInput>
```

The service provides text which can be entered in the command line. This text can be read by every service. Only the the Voice File Input Service itself is allowed to write it.

4.2.4 Voice Output Service

This service is responsible for speech synthesis. This means, that it transforms a text into its spoken representation. For this purpose, MaryTTS is used. MaryTTS creates a Wave file which is then played by the service.

Additionally, the service can play prerecorded Wave file (e.g. a siren). These Wave files can be specified in a file, which is specified in the configuration file of this service (see section 5.6). The format is **text : targetfile**:

The file can look like this:

```
EMERGENCY:siren.wav
WAKE UP:alarmClock.wav
                    ⋮
STARTUP FINISHED:welcome.wav
```

The processing of text works as following:

The service checks, if the given text fully matches the text part of a line in the output mapping file. When this is the case, the according sound file is loaded and played. Otherwise, MaryTTS is used to synthesize the text which is then saved into a new sound file. This sound file is then played.

To allow other services to use the *Voice Output Service*, a regular node is registered. This node uses the following context model.The service provides the attribute output, which is a text readable and writable by every services. Although no other service currently reads the output, it is permitted because it is possible that some service in the future process this information (e.g statistic services).

```
<voiceOutput type="/basic/composed">
  <output type="/basic/text" writer="*" reader="*">
</voiceOutput>
```

4.2.5 Command Mapping Service

This service is responsible for the mapping between voice commands and actions within the VSL. These mappings are stored in a mapping file. This file can be specified in the configuration file of this service (see section 5.8)

A mappings has the format: ***command:type;actions;feedback text***

- command: One or more words out of the vocabulary, which trigger the actions and the feedback text.
- type: GET for get actions, which are used to retrieve data from the VSL or SET for set actions, which write data in the VSL
- actions: specify, which actions the service should perform. This actions are formatted as follows:

  address1~value//address2~value2//. . . //addressN~valueN for SET commands and address1//address2//. . . //addressN for GET commands.

  – address: the address of the node, which is accessed
  – value: the value, which the address has to be set to

- feedback text: Text which is sent to voice output services. This text can contain placeholder for the result of GET commands. For the result of the GET command, ~0 is used, for the second ~1 and so on.

The command part of each mapping has to be unique. When more than one device should perform an action for a single command, this actions have to be part of a single mapping.

As this service does not need to provide information for other services within the VSL, no context model is required.

4.2.6 Configuration Interface

The *Configuration Interface* is a web interface, which includes the following functions:

- Edit the dictionary used by the *Voice Input Services*
- Discover Smart Devices
- Edit mapping list
- Record commands

Each feature is realized on a dedicated page. Additionally, an index page which is used to navigate between the different functions. It is shown in Figure 4.3.

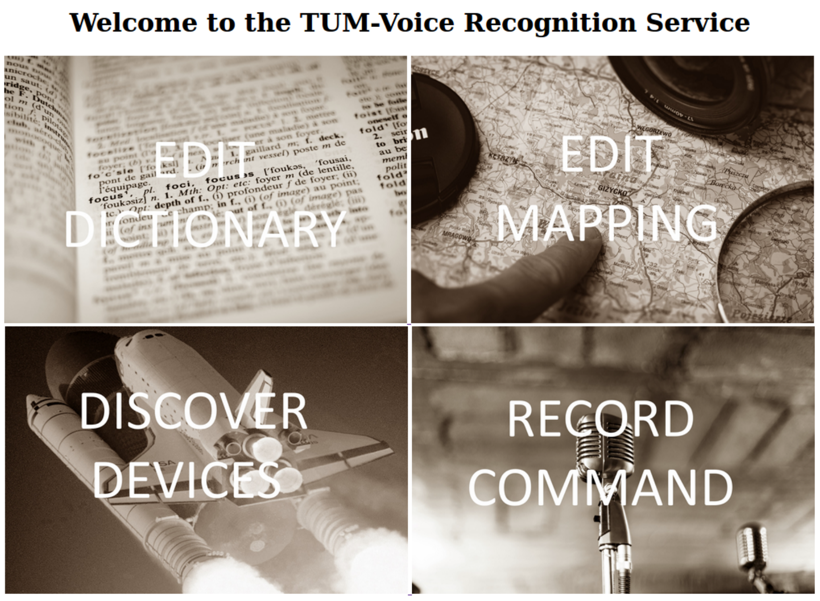
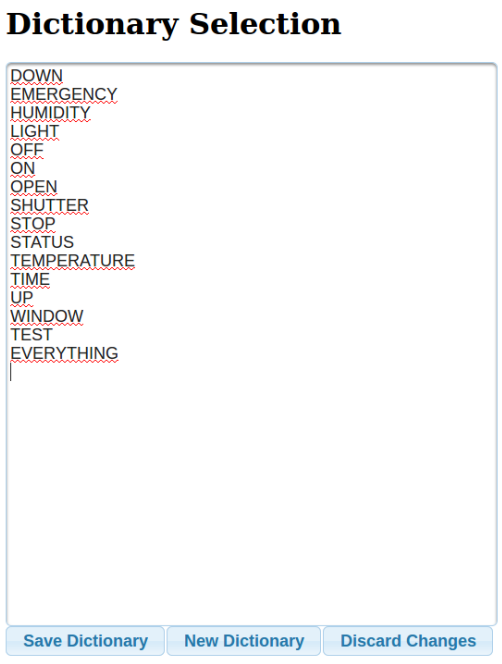


Figure 4.3: Index page of the *Configuration Interface*

4.2.6.1 Dictionary Editing Interface

The *Dictionary Editing Interface* can be used to modify the dictionary used by the *Voice Input Service*. It is shown in Figure 4.4.



Dictionary Selection

DOWN
EMERGENCY
HUMIDITY
LIGHT
OFF
ON
OPEN
SHUTTER
STOP
STATUS
TEMPERATURE
TIME
UP
WINDOW
TEST
EVERYTHING

Figure 4.4: *Dictionary Editing Interface*

It contains a text area, which consists of a word list. Each line has to contain exactly one word. When the word list is saved, a dictionary file is generated. Also, this word list is required to generate the language model file. Both can then be used by the *Voice Input Service*.

4.2.6.2 Device Discovery Interface

This interface is used to discover devices which are currently not present in the system. Furthermore, the names of currently present devices can be changed. The interface is shown in Figure 4.7 in the end of this chapter. Every Device has a name, a GET address and a SET address. Known device types like LEDs have a base address, which is the common part of the GET and SET address. The remaining part of the address is derived from the device type and its base address. The interface also supports custom device types. This types also have a base address. In contrast to known device types, the remaining part of the addresses of custom devices has to be specified. When a device has more than one property which can be accessed, it has to be added as two devices. The device discovery interface is divided into 3 parts.

- On the left side of the page, new devices can be discovered. Therefore, a type can be selected. When a user wants to discover custom types, the type has to be specified in the textbox next to the selection menu (this textbox is only shown, when "custom type" is selected). A device from the list of discovered devices can then be added to the available devices by using the *Use* button. When this button is used, the device is displayed at the bottom of the page.
- On the right side, all previously discovered devices are shown. These devices can be edited or removed by pressing the according button. When the edit button is used, the device is displayed at the bottom of the page.
- On the bottom of the page, the current device is displayed. A user can change the name of this device here. When a custom device is shown, the GET and SET addresses of this device can also be edited.

4.2.6.3 Mapping Editing Interface

This interface is used to edit the mapping between input commands and VSL actions and to configure new commands. It is shown in Figure 4.8. It is divided into four parts:

- In the top left part of the page, the command can be selected. All available words are displayed here. These words can be changed in the dictionary selection interface (see section 4.2.6.1). Below the available words, the currently selected words are shown.
- In the top right part, the available devices are selected. These devices can be discovered using the device discovery interface (see section 4.2.6.2)
- Below, the type of the VSL action (GET/SET) can be selected as well as the feedback text. On the right side, there are two buttons for saving/overwriting and discarding the current mapping.
- In the bottom part, all currently present mappings are displayed. They can be removed or edited.

4.2.6.4 Command Recording Interface

The *Command Recording Interface* can be used to record commands. In contrast to the service described in section 4.2.1, this does not require software (except for a browser) on the device of a user. This interface uses the service described in section 4.2.2. The *Command Recording Interface* is shown in Figure 4.5

Figure 4.5: Command Recording Interface

4.2.7 Adaption Service

In this thesis, a simple Smart Device with a temperature sensor and an L.E.D. (actuator) is used to test the other services. Figure 4.6 shows the Smart Device. It consists of the following hardware:

- 1 Arduino Mega board
- 1 Arduino Ethernet Shield
- 1 white LED
- 1 100Ω Resistor
- 1 KY-013 Temperature sensor module
- 5 flexible jumper wires

This Smart Devices allows to control a white L.E.D. and uses a temperature sensor to measure temperature. For communication, an Arduino Ethernet Shield is used. It has an IPv4 address and listens for commands on TCP port 23. To interact with this Smart Device, a simple protocol was used. When the Arduino receives "*0*", the LED is turned off, when it receives "*1*", the LED is turned on. When "*2*" is received, the temperature in °C is returned.

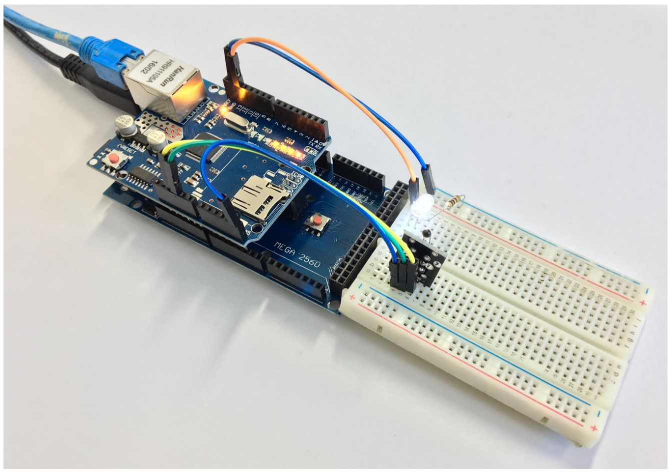


Figure 4.6: Smart Device for testing

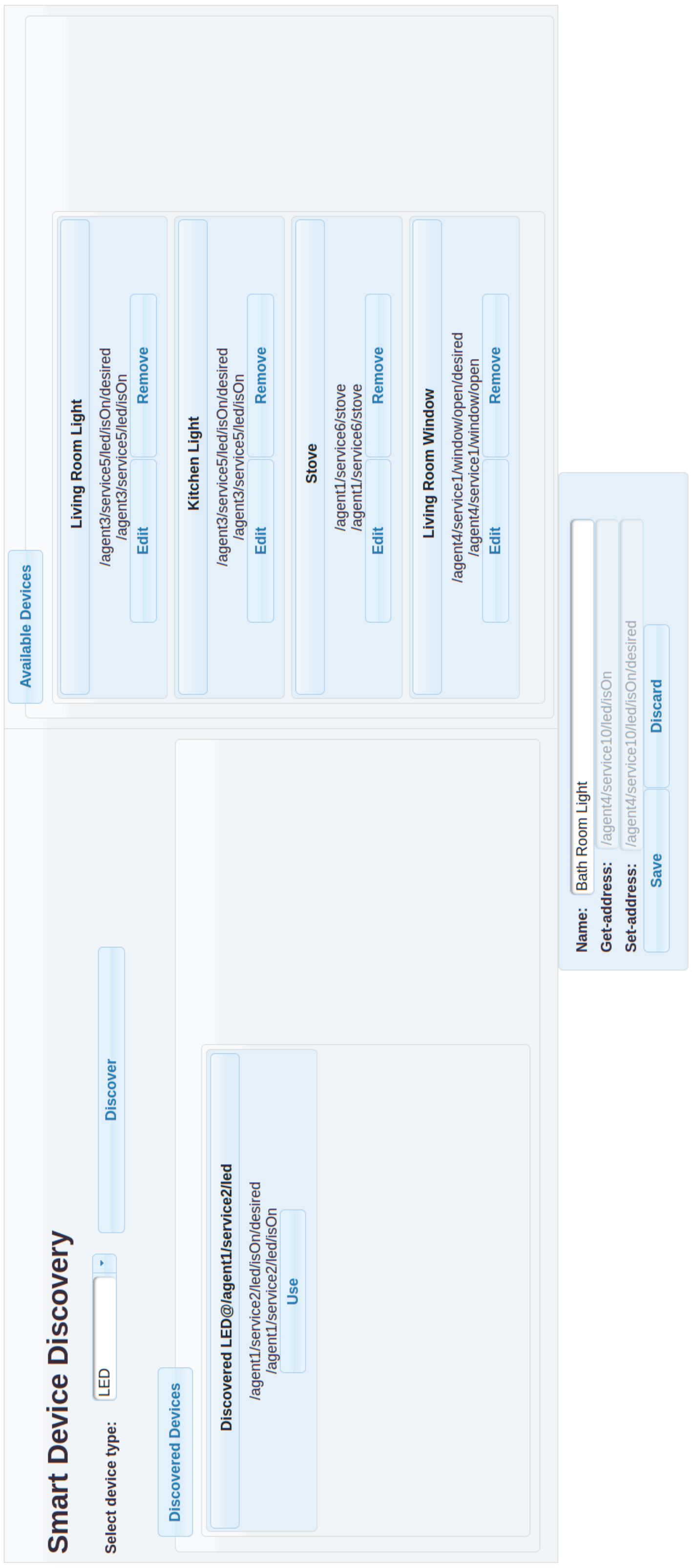
To use this Smart Device within the VSL, a simple Adaption Service has been written. It uses the following context model:

```
<smartDevice type="/basic/composed">
  <led type="/actuators/led" >
  </led>
  <temperature type ="/sensors/temperature">
  </temperature>
</smartDevice>

<led type="/basic/composed">
  <isOn type="/derived/boolean" writer="" reader="*">
  0
    <desired type="/derived/boolean" writer="*" reader="*">
    0
    </desired>
  </isOn>
</led>

<temperature type="/basic/number" writer="" reader="*">
</temperature>
```

To control the LED, the value *./led/isOn/desired* has to be modified. When the status of the LED is changed, it is written in *./led/isOn*. To get the temperature from the temperature sensor, the service queries the sensor every second. This value is stored in *./temperature*



Smart Device Discovery

Select device type: LED

Discover

Discovered Devices

Discovered LED@/agent1/service2/led
/agent1/service2/led/isOn/desired
/agent1/service2/led/isOn
Use

Available Devices

Living Room Light
/agent3/service5/led/isOn/desired
/agent3/service5/led/isOn
Edit Remove

Kitchen Light
/agent3/service5/led/isOn/desired
/agent3/service5/led/isOn
Edit Remove

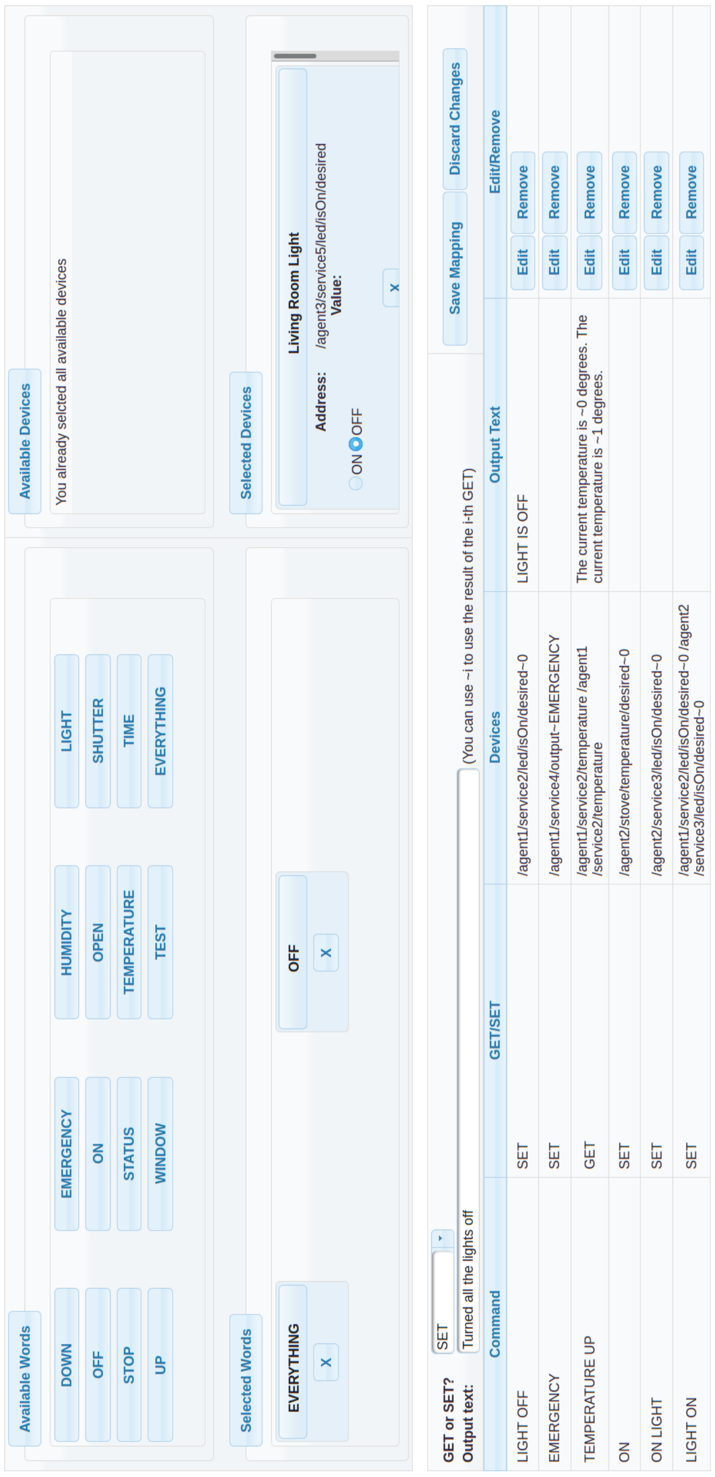
Stove
/agent1/service6/stove
/agent1/service6/stove
Edit Remove

Living Room Window
/agent4/service1/window/open/desired
/agent4/service1/window/open
Edit Remove

Name:	Bath Room Light
Get-address:	/agent4/service10/led/isOn
Set-address:	/agent4/service10/led/isOn/desired

Save Discard

Figure 4.7: Device Discovery interface



Available Words

DOWN EMERGENCY HUMIDITY LIGHT
OFF ON OPEN SHUTTER
STOP STATUS TEMPERATURE TIME
UP WINDOW TEST EVERYTHING

Available Devices

You already selcted all available devices

Selected Words

EVERYTHING x
OFF x

Selected Devices

Living Room Light
Address: /agent3/service5/led/isOn/desired
Value:
ON [✓ OFF]
x

GET or SET? SET
Output text: Turned all the lights off
(You can use ~i to use the result of the i-th GET)

Save Mapping
Discard Changes

Command	GET/SET	Devices	Output Text	Edit/Remove
LIGHT OFF	SET	/agent1/service2/led/isOn/desired~0	LIGHT IS OFF	Edit Remove
EMERGENCY	SET	/agent1/service4/output~EMERGENCY		Edit Remove
TEMPERATURE UP	GET	/agent1/service2/temperature /agent1/service2/temperature	The current temperature is ~0 degrees. The current temperature is ~1 degrees.	Edit Remove
ON	SET	/agent2/stove/temperature/desired~0		Edit Remove
ON LIGHT	SET	/agent2/service3/led/isOn/desired~0		Edit Remove
LIGHT ON	SET	/agent1/service2/led/isOn/desired~0 /agent2/service3/led/isOn/desired~0		Edit Remove

Figure 4.8: Mapping Editing Interface

Chapter 5

Implementation

This chapter discusses the implementation of the system developed in this thesis. First it is discussed how to start the *Voice Control System for Smart Space Orchestration* including the *Configuration Interface*. Afterwards, implementation details of all services are focussed. Every service can configured by a configuration file. These files have a common part. This part is introduced in section 5.2. Most of the configuration files include an individual part. It is introduced in the according section of every service.

All VSL services are developed in Java. The *Configuration Interface* is a web interface developed with *PrimeFaces* [78]. PrimeFaces is a user interface framework for *Java Server Faces* (JSF) [79]. To host this web interface, a Glassfish [80] server is used.

5.1 How to start the system

As mentioned before, the VSL of DS2OS is used as middleware. This middleware is formed by Knowledge Agents (KAs). The KAs have to be started before a service can be connected to the VSL. After the KAs are started, the *Voice Input Services* (including the *Voice File Input Service* and the *Text Input Service* if required) and *Voice Output Services* have to be started before the *Command Mapping Service*. It is recommended to use only a single *Command Mapping Service*, even if there is more than one *Voice Input/Output Service* running. The adaption services can be started at any time when the KA they want to connect is running. The *Voice Mapping Service* does not require started adaption services at start up. They have to be started before they can be used. Figure 5.1 shows all services required to start a specific service.

The *Configuration Interface* can be started without a connection to the VSL. Without a connection to the VSL, the dictionary and the mapping can be edited, but it is not possible to discover devices and record commands without a connection to a KA.

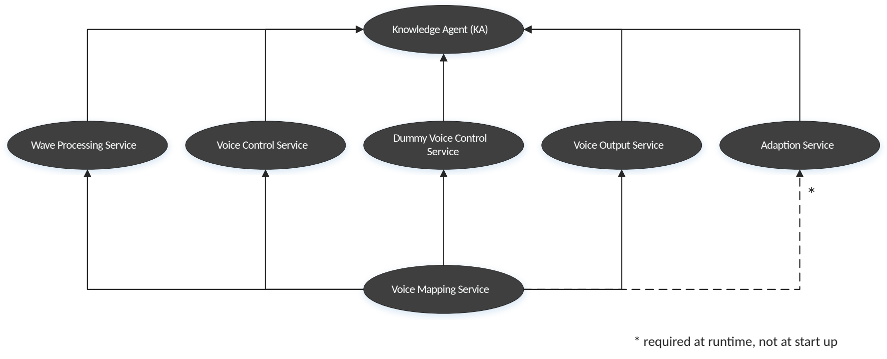


Figure 5.1: Service Dependencies

5.2 Common configration file

All service can be configured via a configuration file. These configuration files consist of two parts. A common part, which is the same for all services, except the Configuration Interface. Additionally, a service can include an individual part. The common part is discussed in this section. The individual part is introduced in the section of the according service.

The following properties can be specified in the common part of the configuration files:

- KAaddress: The address of the **Knowledge Agent**, which is used by the service.
- certificate: The location of the **certificate** file, which is used by the service.

To connect to a KA, the address of the according KA is required as well as a certificate. Both can be specified in the configuration file of the according service. The address of the KA can also be specified as command line argument. If this is the case, the respective entry in the configuration file is ignored.

The common part of the configuration file can look as follows:

```
KAaddress=https://127.0.0.1:8081
certificate=service2.jks
```

5.3 Voice Input Service

As mentioned before, this service uses Sphinx4 [16] as speech-to-text engine. This engine requires a dictionary, a language model and an acoustic model to work properly. The locations of the used dictionary and language model have to be specified in the configuration file (see below). The acoustic model can also be changed, but this is not recommended. The acoustic model used is a generic acoustic model of the English language. Furthermore, Sphinx4 supports a list of filler words (like “ah”, “mhm”,...). They could also be customized, but this service uses the predefined filler words from Sphinx4.

After start up, the service listens to commands by using the default microphone of the hosting system. When a command is recognized, the sub node */lastUtterance* is updated.

The following values can be configured in the individual part of the configuration file *config.txt*.

- acousticModel: The location of the **acoustic model** directory used by Sphinx4.
- languageModel: The location of the **language model** file used by Sphinx4.
- dictionary: The location of the **language model** file used by Sphinx4.

The configuration file can look as follows:

```
acousticModel=resource:/edu/cmu/sphinx/models/en-us/en-us
languageModel=dict/smart.lm
dictionary=dict/smart.dic
KAaddress=https://127.0.0.1:8081
certificate=service1.jks
```

5.4 Voice File Input Service

This service provides the same functionality as the *Voice Input Service*. The only difference of these two services is, that this service does not process a speech signal taken from the default microphone of the hosting system, but out of a Wave file. This wave file can be specified in the configuration file of this service.

The following values can be configured in the individual part of the configuration file *config.txt*.

- acousticModel: The location of the **acoustic model** directory used by Sphinx4.
- languageModel: The location of the **language model** file used by Sphinx4.
- dictionary: The location of the **dictionary** file used by Sphinx4.
- speechFile: The location of the **speech file**used by Sphinx4.

The configuration file can look as follows:

```
acousticModel=resource:/edu/cmu/sphinx/models/en-us/en-us
languageModel=dict/smart.lm
speechFile=recording.wav
dictionary=dict/smart.dic
KAaddress=https://127.0.0.1:8081
certificate=service1.jks
```

5.5 Text Input Service

The *Text Input Service* processes command line input as an alternative to the *Voice Input Service*. In contrast to the *Voice Input Service*, this service does not need dictionary, language model or acoustic model.

This service does not have an individual part in the configuration model.

The configuration file can look as follows:

```
KAaddress=https://127.0.0.1:8081
certificate=service2.jks
```

5.6 Voice Output Service

This service is used for voice output. Furthermore, special pre-configured sound files (e.g a siren) can be played. This has to be specified in a file. The design of this file can be found in section 4.2.4. The location of this file can be specified in the configuration file (see below). This service is not bound to the dictionary of the *Voice Input Service*. When the sub node *./output* is written, the service triggers sound output. First of all, it checks whether the received text matches a line in the sound mapping file. If this is the case, the sound file is loaded and played.

Otherwise, the text is processed by MaryTTS. MaryTTS can be customized, but this service uses the default configuration of MaryTTS. MaryTTS generates a Wave file with the processed text. This text is then loaded and played.

The following values can be configured in the individual part of the configuration file *config.txt*.

- outputMapping: The location of the file which contains the mappings of commands to special sound files.

The configuration file can look as follows:

```
outputMapping=outputMappings
KAaddress=https://127.0.0.1:8081
certificate=service2.jks
```

5.7 Adaption Service

This service communicates with an Arduino Mega board and is able to switch an LED on and off. Furthermore a temperature sensor can be used to get the current temperature.

The software on the Arduino is written in C. The Arduino hosts a telnet server on a fixed IP address. The VSL adaption service connects to this server and sends commands to the Arduino:

- **0:** Turns the LED off.
- **1:** Turns the LED on.
- **2:** Tells the Arduino to return the current temperature in °C

This service does not have an individual part in the configuration model.

The configuration file can look as follows:

```
KAaddress=https://127.0.0.1:8081
certificate=service2.jks
```

5.8 Voice Mapping Service

This service is responsible for mapping (voice) commands to VSL actions. Therefore, it searches the VSL for all services which can be used as input. This is done by searching for services with the type */voice/voiceInput*. Afterwards, the service subscribes to the sub nodes *./lastUtterance* of all found services.

This service loads a mapping list from a mapping file. This file can be specified in the configuration file. Then, the service searches for all available *Voice Input Services*. When a new command is available, this command is looked up in a map and the according VSL actions are performed. The according voice output text is sent to the previously found *Voice Output Services*.

The following values can be configured in the individual part of the configuration file *config.txt*.

- mapping: The location of the file which contains the mapping list

The configuration file can look as follows:

```
mapping=mappings
KAaddress=https://127.0.0.1:8081
certificate=service2.jks
```

5.9 Configuration Interface

The *Configuration Interface* is a web interface. It is implemented in PrimeFaces [78], a user interface framework for Java Server Faces (JSF) [79]. A Glassfish server [80] is used to host this interface. It is divided into four parts: the *Dictionary Editing Interface*, the *Device Discovery Interface*, the *Mapping Editing Interface* and the *Command Recording Interface*.

5.9.1 Dictionary Editing Interface

This interface is used to generate a dictionary file out of a word list. For this purpose, a Python script provided by the developers of Sphinx is used. This script is available in the GIT repository of CMUSphinx [81]. The word list is saved in a file. Furthermore, an acoustic model is required to generate a dictionary. This acoustic model is provided by the developers of CMU Sphinx [41].

5.9.2 Device Discovery Interface

This interface allows to search the VSL for nodes of a type which can be specified by a user. Therefore, a service is registered to connect to the VSL.

All discovered devices are saved in a file, which can be specified in the configuration file of the configuration interface (see section 5.9.5) This file is used by the *Mapping Editing Interface* (see section 5.9.3).

5.9.3 Mapping Editing Interface

This interface is used to create and edit mapping list between commands and actions within the VSL. Furthermore, a language model is created based on this mapping. The mapping file can be used by the *Voice Mapping Service* (see sections 4.2.5 and 5.8). The word list and the device list is loaded from two files. The mappings list is also loaded from a file and when a mapping is added, edited or removed, the changes are saved in the same file.

When the mapping list is changed, the language model is updated. To generate a language model, a Perl script provided by the developers of CMUSphinx is used. This script is available online [72].

The location of all files used for this interface can be specified in the configuration file of the Configuration Interface (see section 5.9.5).

5.9.4 Command Recording Interface

This service is used to record a command. This command is saved it in a Wave file, which is afterwards uploaded to the server. To process this file, an instance of the *Voice File Input Service* (see section 4.2.2) is used.

To record commands, a script of Chris Wilson is used. It can be found on his web page [82]. This script uses another script provided by Matt Diamond, which can be found on GitHub [83].

This interface uses the default microphone provided by the hosting system. When using this service, the browser requires the permission to access the microphone of the hosting system.

5.9.5 Configuration File

The following values can be configured in the configuration file of the web interface *config.txt*:

- **deviceFile**: The location of the file which contains a list of discovered devices
- **wordsFile**: The location of the file which contains a list of available words
- **mappingsFile**: The location of the file which contains a list of mappings
- **dictionaryFile**: The location of the file, where the dictionary should be saved
- **languageModelFile**: The location of the file, where the language model should be saved
- **acousticModelFile**: The location of the file which contains the acoustic model for generating the dictionary and the language model
- **KAaddress**: The address of the Knowledge Agent, which is used by the services.
- **certificateRec**: The location of the **certificate** file, which is used by the record service.
- **certificateDisc**: The location of the **certificate** file, which is used by the discovery service.

The configuration file can look as follows:

```
deviceFile=devices
wordsFile=smart.vocab
mappingsFile=mappings
dictionaryFile=smart.dict
acousticModelFile=g2p-seq2seq-cmudict
languageModelFile=smart.lm
KAaddress=https://127.0.0.1:8081
certificateRec=service4.jks
certificateDisc=service5.jks
```

Chapter 6

Evaluation

In this chapter, the requirements specified in the analysis chapter (see section 2.7) are focussed again. It is discussed if the system developed in this thesis fulfills these requirements. For this evaluation, a dictionary consisting of 16 words is used. It can be found in appendix D.

6.1 R1: Short response time

In this section, the response time of the *Voice Input Service* is evaluated. As discussed in section 2.7, the response time has to be less than the time it would take to perform the same action with a remote. For this process, 3 seconds are assumed. In this scenario, a light is turned on/off. Two commands *"KA LIGHT ON"* and *"KA LIGHT OFF"* were used. Each command was recorded 15 times.

To evaluate the response time, the *Voice Input Service* was altered. The altered version plays a sound file whenever a command is recognized. This sound is then used to calculate the response time as the time difference between the end of the voice command and the moment when the command was recognized. The VSL and the controlled device increase the response time of the system. This part of the response time is not different for voice and other input methods. Therefore, this additional delay is not evaluated.

Table 6.1 below shows minimum and maximum value as well as the mean value and the median of the two commands. The 0.25- and 0.75-quantiles are also shown. The response time for all recordings can be found in the table in appendix B The results are visualized in Figure 6.1. The box shows the 0.25 and 0.75 quantile. The line in the box is the median. The whiskers represent the minimum and maximum value.

	LIGHT ON	LIGHT OFF	both commands
min	0,70	0,80	0,70
max	1,59	1,76	1,76
mean	1,09	1,29	1,19
median	0,95	1,41	1,24
0.25-quantile	0,83	0,97	0,87
0.75-quantile	1,35	1,50	1,49

Table 6.1: Response times

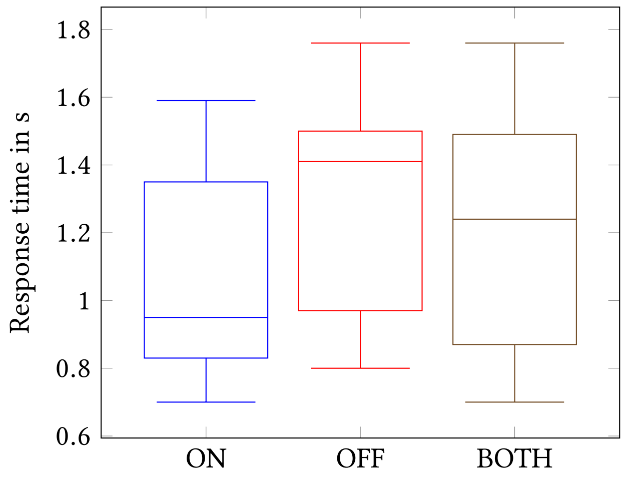


Figure 6.1: Response times of LIGHT ON/OFF

The response time of the *Voice Input Service* differs from the spoken command. It can be seen, that the mean and the median of *"KA LIGHT ON"* are below the according values of *"KA LIGHT OFF"*. The reason for this behavior are the sounds at the end of the words *"ON"* and *"OFF"*. When the word *"ON"* is spoken the voice recognition engine recognizes the end of the utterance faster, so the recognition of the utterance can start earlier. This can also be seen in the time spectra of the two utterances. These spectra can be seen in Figure 6.2.

The *Voice Input Service* has a response time below 3 seconds for all measurements. Therefore, the response time is below the time, which was specified as requirement in the analysis chapter (see section 2.7). Consequently, this requirement is fulfilled.

In the future, it can be evaluated further by using a wider range of words. It can also be evaluated, if the processing time is shorter, when a microphone with push-to-talk (see section 2.2.1.3) support is used.

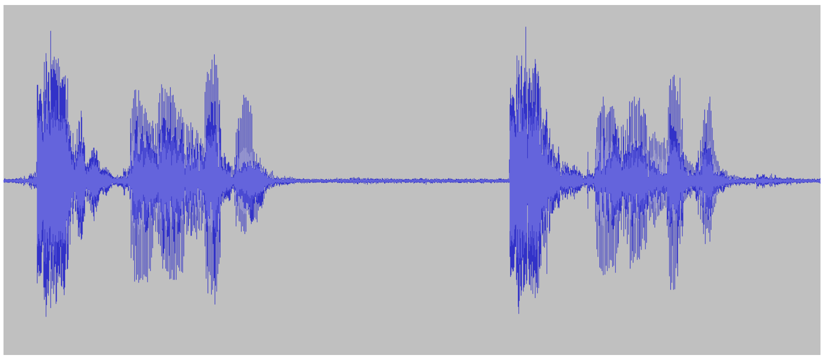


Figure 6.2: Time spectra of KA LIGHT ON (left) and KA light off (right)

6.2 R2: Offline functionality

The system developed in this thesis used Sphinx for voice recognition. MaryTTS is used for speech synthesis. Both do not require an active Internet connection to fulfill their purpose. Consequently, this requirement is satisfied.

6.3 R3: Runnable on low power hardware

In this thesis, Arduino [33] boards were used to build Smart Devices. To enable communication with other devices, Arduino Ethernet Shields were used. The subsystem, which are used for interaction with users (Voice Input Services, Voice Output Services, Configuration Interface) were hosted on a Raspberry Pi. This device was also responsible for the mapping from voice commands to VSL actions. Both devices cost below 50 €. They also have low power consumption. Consequently, this requirement is satisfied.

6.4 R4: Easy configuration

The system developed in this thesis includes a subsystems called Configuration Interface. It allows users to configure the mapping from voice commands to VSL actions. It can also be used to edit the dictionary and to discover new devices. This Interface is implemented as web interface. It is designed to be usable by non expert users. It cannot be objectively evaluated, if non experts are able to use the Configuration Interface. This has to be evaluated in the future.

6.5 R5: Adaptability at runtime

The system developed in this thesis uses VSL of DS2OS as middleware. Adding and removing devices at runtime is supported by the VSL. The Voice Control System for Smart Space Orchestration allows users to discover new devices, which are added to the controlable devices at runtime. The mapping can also be adapted at runtime. Therefore the system developed in this thesis is adaptable at runtime.

6.6 R6: Low error rate

Two get a low error rate, two different properties have to be fulfilled. The system has to trigger an action when the according command is spoken. Also the system must not trigger an action when the according command is not spoken. To avoid unintended actions, a trigger word can be used. This word has to be spoken before each command. This trigger word is focussed in the following subsection.

In section 6.6.2, it is evaluated how often the system maps a spoken utterance to the wrong VSL action.

6.6.1 Trigger word

A voice recognition system can use a word to trigger the recognition. It is also possible to use no trigger word at all. The advantage of a trigger word is, that it is less likely, that a command gets executed unintendedly when a trigger word is used. A good trigger word has to have a good detection rate and it should be short. Furthermore it should not appear too often in utterances which does not intend to trigger a recognition. The following trigger words were evaluated for this thesis:

- **no trigger word:** When no trigger word is used, the detection rate is not reduced by an additional word, which has to be recognized correctly. Using no trigger word works well in a quiet environment without background talk. Furthermore feedback text can be problematic for a voice recognition system without a trigger word. In such a scenario, it is possible, that the voice recognition system recognizes command synthesized by a speech synthesizer. This happend in the testing environment when the command "TEMPERATURE" was used the query the temperature. It was followed by the feedback "The current temperature …", which triggered the command again. Therefore, not using a trigger word is not a suitable solution.

- **command:** This word has a good detection rate (>95%). Also it is natural for users to start the recognition of a command with the word *“command”*. Furthermore it is unlikely, that the word *“command”* is spoken in an other context than starting a recognition in the testing environment. This is because the system was tested in a German speaking environment. A drawback of this solution is the length of the word “command”. All in all, *“command”* can be used as trigger word
- **KA:** This word is short and it very unlikely, that it appears in a different context than starting a voice recognition, even in an English speaking environment. The major drawback of this solution is, that *“KA”* is an acronym. Therefore, the translation between this words and the according sounds has to be done manually. This could be done within a few minutes. Test showed that the detection rate is better than the detection rate of all other tested possibilities (>98 %). Therefore, *“KA”* can be used as trigger word.

All in all, the word *“KA”* works best as trigger word. It is short and does not appear in general language usage. Also, the detection rate was higher than the detection rate of other solutions. The only problem, that the mapping between word and sounds has to be created manually, could be fixed with little effort.

6.6.2 Incorrect mappings

As mentioned in section 2.7, there are two error rates, word error rate and task error rate. The voice recognition service maps a command to an action, if the spoken command matches the specified command of the action exactly. Therefore, the task error rate is the product of the word error rates of the words of a command. To evaluate the task error rate, the commands *“KA LIGHT ON”* and *KA LIGHT OFF* were used. Each command was recorded 75 times. The table below is an excerpt of table C in the Appendix. It shows that all incorrectly recognized commands together with the commands spoken. In this test scenario, 3 commands were incorrectly recognized.

Nr.	Spoken Command	Recognized Command
	⋮	
16	KA LIGHT OFF	KA LIGHT OFF ON
	⋮	
18	KA LIGHT OFF	SHUTTER OFF ON
	⋮	
86	KA LIGHT OFF	KA UP ON

Table 6.2: Task error rates of “KA LIGHT ON” and “KA LIGHT OFF” (excerpt)

This evaluation shows, that the system has a task error rate of 2% in this scenario. Consequently, the system has a detection rate of 98%. This is above 95%, which is considered as reliable (see section 2.7). Therefore, the system has a low error rate.

6.7 R7: Transparency

All currently present mappings are displayed in the mapping interface. This interface can also be printed. Consquently, a user has the option to inform himself which command is mapped to which mapping. All in all, this requirement is satisfied.

6.8 R8: Voice output

The system developed in this thesis includes a subsystem, which is responsible for voice output. This subsystem uses voice output to acknowledge commands. Also it can be used to present the result of GET actions. These actions are used to get the result of sensor. They can also be used to get other context, which is stored within the VSL. The subsystem uses MaryTTS [43] for speech synthesis. MarryTTS can synthized arbitrary text. The system developed in this thesis satisfies this requirement.

6.9 Conlusion

In this section, the system was evaluated to check if it fulfills the requirements specified in the analysis chapter (section 2.7). The result is that it satisfies all requirements, which can be measured objectively. It provides fast and reliable voice recognition without an Internet connection. Also, new devices can be discovered at runtime. The system is runnable on low power consumption hardware. Also voice output is provided. In contrast to the other requirements, it cannot be objectivly evaluated, whether the system is configurable by non experts. In the future it has to be evaluated if the Configuration Interface is easy to use (see section 7.1).

Chapter 7

Conclusion

In this thesis, a system was developed, which allows users to control Smart Devices via voice commands. For this purpose, the VSL of DS2OS was used as middleware. This led to a design where the functionality is divided into different services, which communicate not directly with each other but via the VSL. As a result, the services are only loosely coupled and can easily be exchanged. The following services were developed:

- **Voice Input Service:** A service which is responsible for transforming speech input into text. The speech signal is taken from a microphone.
- **Voice File Input Service:** A service which is responsible for transforming speech input into text. The speech signal is taken from a Wave file.
- **Text Input Service:** A service which emulates a *Voice Input Service* and offers text input, which is processed like the result of a *Voice Input Service:*.
- **Voice Output Service:** A service which is responsible for transforming text into speech output.
- **Voice Mapping Service:** A service which is responsible for the matching from voice commands to actions within the VSL.
- **Adaption Service:** A service which interacts with actuators and sensors to test the services above.

Furthermore, a *Configuration Interface* was developed to allow users to configure the mapping from voice commands to action within the VSL. This is realized as web interface. With this interface, users can edit the dictionary used by the Voice Input Services, discover Smart Devices and configure the mapping between voice commands and actions within the VSL.

The system developed in this thesis was compared with other currently available systems. Today a variety of voice control software is available. Most of it, including well known solutions like Siri or Cortana are used as a personal assistant. Siri and Cortana require an active Internet connection to be functional. This is not the case for the system developed in this thesis. It provides offline voice recognition.

Also there is a small number of voice control solutions available, which can be used to control Smart Devices via voice recognition. Apple Homekit, a framework which allows Apple's mobile devices to control Smart Devices was observed. Three solutions were examined which use CMU Sphinx to control Smart Devices. The MOVI Arduino Shield, a system developed by the c't magazine for controlling Phillips Hue lamps and a Smart Alarm Clock. All three solutions have in common, that it is not possible to exchange the used Smart Devices without changing other parts of the software too. This is in contrast to the design of this solutions, where it is possible to add or remove Smart Devices without needing to change other parts of the software. This is possible, because the services communicate via the VSL.

In the end of this thesis, it was evaluated, if the speech recognition of the system developed in this thesis fulfills the requirements specified in the analysis step. Therefore, a scenario was chosen in which a light was turned on/off with the command *"KA LIGHT ON/OFF"*. The results showed, that system fulfills these requirements.

7.1 Future work

This thesis offers a variety of possible future works. First of all, the *Configuration Interface* can be improved. Currently, no security functions like a user management has been implemented. This is no problem in a testing environment, but it can be problematic in a different scenario.

Currently, new devices have to be discovered manually via the Device Discovery Interface. This discovery is independent from the dictionary. In the future, it could be implemented that every device has a predefined set of commands. When a device is discovered, this set of commands could be automatically included in the dictionary.

Another possible future work is the development of another voice recognition service. Currently, CMUSphinx is used as speech-to-text engine. It fitted better than all other examined solution. It is likely that new speech-to-text engines are going to be released in the future. As the *Voice Input Service* is only loosely coupled with other services, a new voice control service, which uses another speech-to-text engine can easily be integrated in the system.

A service can be included, which provides an interface to WhatsApp [84]. WhatsApp is an instant messaging service available on Android, iOS and Windows Phone [85]. It can be used to record and send sound files. This service could be used as an alternative to the existing voice processing services.

Furthermore, the current system was evaluated by measuring the recognition time and the detection rate in a specific scenario. It would be useful to evaluate the system in different scenarios to see, how the system performs when the scenario is changed. Also, the user experience of a user who is not used to a voice control system could be evaluated.

Appendix A

Smart Space Orchestration

This section discusses Smart Space Orchestration (S2O) and ubiquitous computing. First, the origin and different aspects of ubiquitous computing are discussed. After the definition of important terms, the Distributed Smart Space Orchestration System (DS2OS) is introduced.

A.1 Ubiquitous Computing

The research field of ubiquitous computing goes back to the early 1990s, when Mark Weiser, a researcher at Xerox Palo Alto Research Center (PARC) had the idea of including computers into the environment to facilitate everyday life, without grabbing attention of people using it [86]. His vision was to include computers everywhere in a way they disappear, meaning that everybody can use it in a natural way without thinking and focusing on them [87]. This stands in contrast to virtual reality, where the aim is to make a “real world” inside the computer, which is diametrically opposed to the idea of ubiquitous computing. The different aspects of ubiquitous computing can be seen in Figure A.1. Therefore ubiquitous computing can be splitted up into these parts:

A.1.1 Mobile Computing

Mobile computing means, that a computer can be transported. Mobile computing devices have the limitation, that they can not obtain information about the environment [88]. Examples of mobile computing devices are tablets, Smartphones and laptops. Although mobile computing wasn’t reality in the beginnings of ubiquitous computing, it has become reality until today.

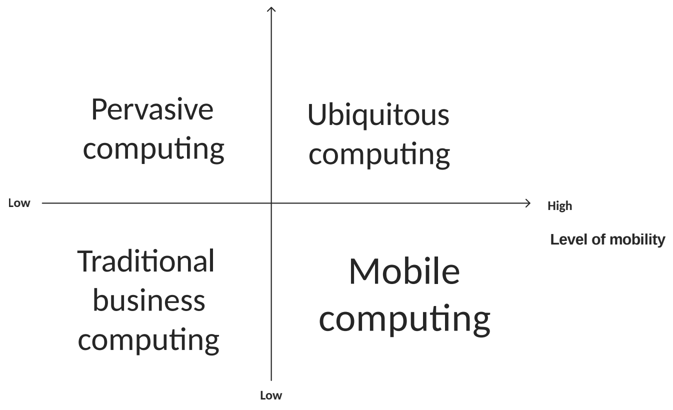


Figure A.1: Fields of ubiquitous computing (based on [88])

A.1.2 Pervasive Computing

The research field of pervasive computing is about making an environment “intelligent”. An “intelligent” environment consist of one or more *Smart Devices.* These Smart Devices use sensors (inputs) and actuators (outputs) to communicate with their environment. In contrast to mobile computing, pervasive computing has not become reality yet [89].

A.1.3 Distributed Computing

Distributed computing is a field of computing which influences ubiquitous compting In a distributed system, an application runs distributed on more than one independent computer connected via a network. A distributed system appears as a single coherent system for users. This is usually provided by *middleware* [88].

A.2 Definitions

In this thesis, the terms *Smart Device*, and *Smart Space (Orchestration)* are used. These terms are differently defined in literature. In this thesis, the following definitions are used:

1. **Smart Device:** An embedded device, which can communicate with its environment using sensors and actuators and can be controlled via a network [1].
2. **Smart Space:** A physical space that contains one or more Smart Devices [1].
3. **Smart Space Orchestration:** Managing Smart Devices within a Smart Space with software [1].

A.2.1 Heterogeneity

A major reason why pervasive computing isn't reality today is the heterogeneity of Smart Devices. This resulted in the emerging of so called silos. Devices of one silo are only able to operate with other devices within the same silo, but not with devices from another silo. There are different kinds of silos:

Figure A.2: Silos [90]

- **Vendor:** Different vendors design their devices in a way that they communicate over special protocols with each other. These protocols are designed in a way that they are incompatible to protocols of their competitors to protect their market share [1].
- **Functional:** A Smart Device can only interoperate with devices from the same operational group (e.g. heating,ventilation) [1].
- **Spatial:** Two devices can only communicate with each other, when they are physically close to each other [1].
- **Technical:** Different Technologies are more suitable for devices with different purposes [1].
- **Middleware:** Applications programmed for different middlewares are unable to communicate with each other [1].

A.3 Middleware

As mentioned in section A.2.1, different components (e.g Smart Devices) can be heterogeneous. As a result of this homogeneity, an additional layer is required to cover this problem. This layer is called middleware and operates between the platform interface (dependent on hardware and operating system) and the user applications [91]. Therefore it provides an *Application Programming Interface (API)* for the user applications, which is independent from the platform interface: A middleware is responsible for the

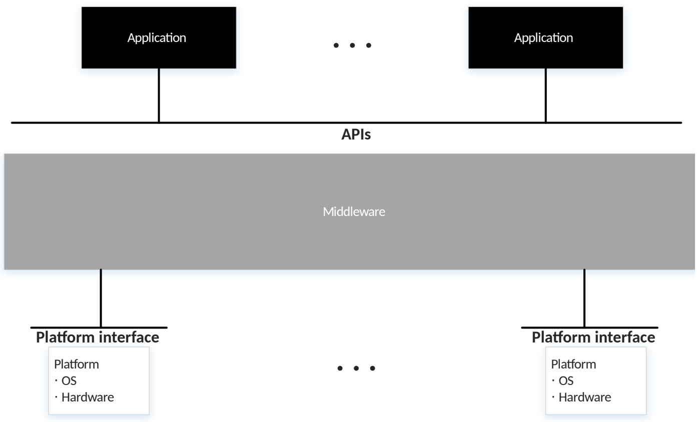


Figure A.3: Architecture of middleware (based on [91])

communication between an application and the underlying layer. As a result, user applications do not directly communicate with the underlying layer This layered architecture can be seen in Figure A.3. A middleware enables portability of user applications [91]. Additionally it can provide different kinds of *transparency* [92]:

- **Access transparency** covers representation and access of data [92]
- **Location transparency** hides the location of a resource [92]
- **Migration transparency** masks migration of resources [92]
- **Relocation transparency** migration transparency while resource in use [92]
- **Replication transparency** conceals duplication of a resource [92]
- **Concurrency transparency** covers the shared access of a resource [92]
- **Failure transparency** hides failure and recovery of a resource [92]

A.4 Context

Smart Devices need to store and process information about the real world [4]. This information is called *context*. Moreover, Smart Devices need to communicate with each other. This is done by exchanging context. In detail, context describes all information needed by a service to orchestrate a Smart Space [4]. To describe a context, *context models* can be used. A language which can be used to create context models is called *meta model*. An instance of a context model is called *virtual object* [4]. A virtual object describes a real world object (e.g a car). The structure of a real world object is defined by so called *ontologies*. For the same object, there can be different ontologies used for different purposes, this ontologies are called *domain ontology*. All domain ontologies of an object share a set subset of concepts. This common subset is called *upper ontology* [1]. Figure A.4 visualizes the connection between these terms.

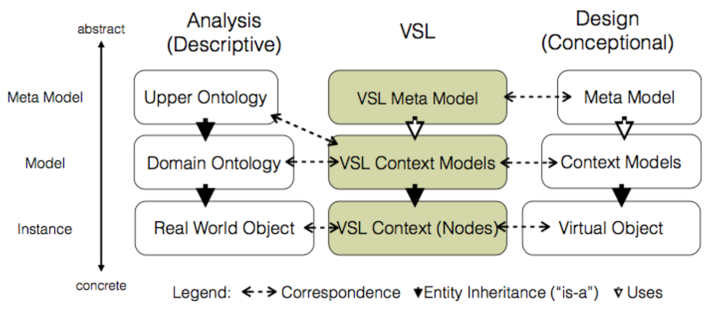


Figure A.4: Context Model Terminology [1]

Context models need to be exchanged between Smart Devices. This can be done via a central repository, the *context model repository (CMR)* There are different suitable representations for context models [4]. In this thesis, only the representation used in DS2OS will be discussed in section A.5

A.5 Distributed Smart Space Orchestration System (DS2OS)

The Distributed Smart Space Orchestration System (DS2OS) is a system to orchestrate Smart Spaces, which is developed the chair of network architectures and services at the Technical University of Munich. The system provides a middleware for brokering and storing state context between devices within a Smart Space [2]. DS2OS is written in Java and consists of three blocks [1]:

- The *Virtual State Layer (VSL)* is a middleware, which acts as distributed operating system in a Smart Space [1].
- The *Smart Space Service Management (S2S)* manages the services available within the VSL [1].
- The *Smart Space Store (S2Store)* is global service manager, which supports crowd-sourcing of services [1].

In this thesis, different VSL services are going to be implemented. The S2S is transparent for these services and the S2Store is not used. Therefore, only the VSL is discussed in this thesis.

A.5.1 Virtual State Layer (VSL)

The Virtual State Layer is a middleware. It is formed by self-organizing unstructured peers and manages the information of a Smart Space [3]. This information is called context. The peers are autonomous and called Knowledge Agents (KAs). The VSL is completely self-managed and encapsulates the functionality from the KAs [3]. Services, which produce context are decoupled from services consuming context. Each of these services is connected to exactly one KA. This KA is responsible for storing data and manages the retrieving of data from other services [3]. When services communicate with each other, they do not have to differentiate, whether the other service is connected to the same KA or to another KA.

To describe the structure of context produced by services, *context models* (see section A.5.2) are used. Each KA stores all required context models locally. This increases the resilience [3]. Furthermore, context models can be automatically synchronized. This can be done by a central *context model repository (CMR)* [1]. The content of the CMR is synchronized between all KAs. This design results in the context models being the interfaces of their services. Services communicate by accessing context of other services. The access can either be a read access via *get* or a write access via *set* [1]. Additionally, services can *subscribe* to *data nodes*.

Figure A.5 shows, how services communicate over the VSL. On the bottom, there are Smart Devices which are connected to the VSL by so called adaption services. These services can be used by other (orchestration) services. As mentioned before, these services do not directly communicate with each other, but use the VSL to connect to each other. More about the different types of services can be found in section A.5.3

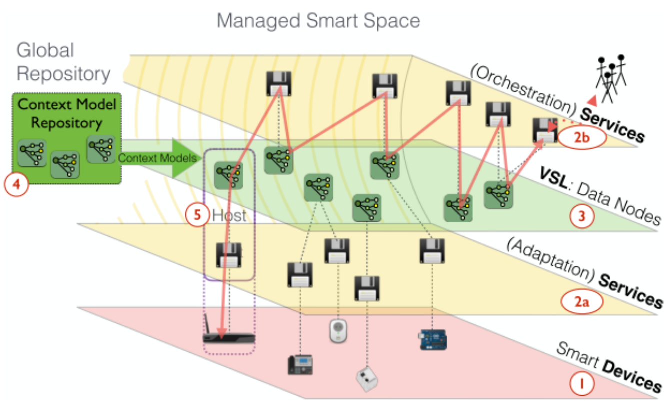


Figure A.5: The VSL [3]

There are two different types of data nodes [1]:

- A **regular node** is a node used to store context within the VSL [1]. All regular nodes form a tree. With this nodes, a service can subscribe any sub-tree and gets notified, when anything within this sub-tree is changed [3].
- **Virtual nodes** are nodes which are not used to store context within the VSL, but use callback functions to provide context [1]. They form a so called *virtual sub-tree*. When this sub-tree is accessed, callback handlers are used to handle the Request [3]. It is also possible to access a virtual node with additional parameters. This parameters are passed by accessing a sub-tree of the requested tree.

For a requesting service, it is transparent, whether the accessed node is a regular node or a virtual node. The context in the VSL is independent from the service implementation. This fosters portability [3]. Additionally, the VSL allows to search nodes via its type identifier [1], which is the context model. This allows dynamic binding of services. The context of each service is described by exactly one context model [3].

The design goals of the VSL are performance, resilience and security-by-design [3].

A.5.2 Context Models

Context Models describe the structure of context. Context is information within a Smart Space. Nowadays, there are different types of context models used [4]. This thesis only describes the approach taken in the VSL. This model uses hierarchically structured type key-value pairs. Additionally, the key-value pairs include management meta data including access rights, version info, type, address and value [4]. The key-value pairs (tuples) are hierarchically structured by their addresses and form a tree [4]. The different level are separated by "/" [4]. Each context model is represented in XML markup, which enables syntax validation [4]. This XML representation is located in the Context Model Repository (CMR) and synchronized by all Knowledge Agents [3]. At start up, each service is bound to a context model. Context models are identified by their name and each context model has a type [4]. This type can be used for composing and sub-typing to create new context models. As a result an extensible type system is formed. This type system consists of three basic data types. Each of this types can have restrictions [4]:

- */basic/number* represents a number and can be restricted by its upper and lower bound [4].
- */basic/text* represents a text and can be restricted by regular expressions [4].
- */basic/list* represents a list of context nodes and can be restricted by the minimum and maximum amount of entries and the types of entries [4].

All types are either basic types or composed from existing context models. All types can have default values. These values are the values between the <> tags. As an example, the type Boolean can be represented as [1]:

```
<boolean type="/basic/number"
         restrictions="minimum=0,maximum=1">
         0
</boolean>
```

The context model has to be saved in the CMR. In this case the model of boolean and all following models are located in the sub-folder *derived* in the repository. With this type, a traffic light can be represented [1]:

```
<lamp type="/basic/derived" >
         <on type="derived/boolean">0</on>
</lamp>
```

```
<trafficLight type="/basic/derived" >
        <red type="derived/lamp">1</red>
        <yellow type="derived/lamp">0</red>
        <green type="derived/lamp">0</red>
</trafficLight>
```

A.5.3 Service Orientation

The architecture of DS2OS is service oriented [3]. In this architecture, components are coupled via services. This design fosters reusability. There are different types of services used in Smart Space Orchestration [1]:

- **Adaptation services** are responsible for the communication between Smart Devices and the VSL
- **Advanced reasoning services** generate new context based on existing context.
- **Emulator services** simulate the behavior of other services.
- **Remote access services** are used to communicate with services from remote DS2OS sites.
- **User interface services** provide a User Interface to other service. This user interface can be a voice recognition system.

The difference of these types is their purpose, not the way, they are handled by the VSL.

Appendix B

Response time table

Command	End of Command	Time of Recognition	Reaction Time
LIGHT ON	1.08	2.28	1.20s
LIGHT OFF	3.77	5.35	1.58s
LIGHT ON	7.06	8.48	1.42s
LIGHT OFF	10.07	11.55	1.48s
LIGHT ON	13.37	14.64	1.27s
LIGHT OFF	16.31	17.70	1.39s
LIGHT ON	19.29	20.84	1.55s
LIGHT OFF	22.38	23.21	0.83s
LIGHT ON	25.05	26.21	1.16s
LIGHT OFF	27.80	29.31	1.51s
LIGHT ON	31.00	31.70	0.70s
LIGHT OFF	33.27	34.68	1.41s
LIGHT ON	36.30	37.10	0.80s
LIGHT OFF	38.80	40.13	1.33s
LIGHT ON	41.60	42.50	0.90s
LIGHT OFF	43.82	44.81	0.99s
LIGHT ON	46.30	47.15	0.85s
LIGHT OFF	48.66	50.18	1.52s
LIGHT ON	51.69	53.27	2.58s
LIGHT OFF	54.70	55.65	0.95s
LIGHT ON	57.31	57.98	0.95s
LIGHT OFF	59.48	60.28	0.80s
LIGHT ON	64.98	65.75	0.77s
LIGHT OFF	67.23	68.72	1.49s
LIGHT ON	70.23	71.09	0.86s
LIGHT OFF	72.67	74.14	1.47s

LIGHT ON	75.66	77.25	1.59s
LIGHT OFF	78.73	79.64	0.91s
LIGHT ON	81.23	81.94	0.71s
LIGHT OFF	83.99	85.75	1.76s

Table B.1: Response Time of the Voice Recognition Service

Appendix C

Error rate table

Nr.	Spoken Command	Recognized Command
1	KA LIGHT ON	KA LIGHT ON
2	KA LIGHT OFF	KA LIGHT OFF
3	KA LIGHT ON	KA LIGHT ON
4	KA LIGHT OFF	KA LIGHT OFF
5	KA LIGHT ON	KA LIGHT ON
6	KA LIGHT OFF	KA LIGHT OFF
7	KA LIGHT ON	KA LIGHT ON
8	KA LIGHT OFF	KA LIGHT OFF
9	KA LIGHT ON	KA LIGHT ON
10	KA LIGHT OFF	KA LIGHT OFF
11	KA LIGHT ON	KA LIGHT ON
12	KA LIGHT OFF	KA LIGHT OFF
13	KA LIGHT ON	KA LIGHT ON
14	KA LIGHT OFF	KA LIGHT OFF
15	KA LIGHT ON	KA LIGHT ON
16	KA LIGHT OFF	KA LIGHT OFF ON
17	KA LIGHT ON	KA LIGHT ON
18	KA LIGHT OFF	SHUTTER LIGHT OFF
19	KA LIGHT ON	KA LIGHT ON
20	KA LIGHT OFF	KA LIGHT OFF
21	KA LIGHT ON	KA LIGHT ON
22	KA LIGHT OFF	KA LIGHT OFF
23	KA LIGHT ON	KA LIGHT ON
24	KA LIGHT OFF	KA LIGHT OFF
25	KA LIGHT ON	KA LIGHT ON
26	KA LIGHT OFF	KA LIGHT OFF

27	KA LIGHT ON	KA LIGHT ON
28	KA LIGHT OFF	KA LIGHT OFF
29	KA LIGHT ON	KA LIGHT ON
30	KA LIGHT OFF	KA LIGHT OFF
31	KA LIGHT ON	KA LIGHT ON
32	KA LIGHT OFF	KA LIGHT OFF
33	KA LIGHT ON	KA LIGHT ON
34	KA LIGHT OFF	KA LIGHT OFF
35	KA LIGHT ON	KA LIGHT ON
36	KA LIGHT OFF	KA LIGHT OFF
37	KA LIGHT ON	KA LIGHT ON
38	KA LIGHT OFF	KA LIGHT OFF
39	KA LIGHT ON	KA LIGHT ON
40	KA LIGHT OFF	KA LIGHT OFF
41	KA LIGHT ON	KA LIGHT ON
42	KA LIGHT OFF	KA LIGHT OFF
43	KA LIGHT ON	KA LIGHT ON
44	KA LIGHT OFF	KA LIGHT OFF
45	KA LIGHT ON	KA LIGHT ON
46	KA LIGHT OFF	KA LIGHT OFF
47	KA LIGHT ON	KA LIGHT ON
48	KA LIGHT OFF	KA LIGHT OFF
49	KA LIGHT ON	KA LIGHT ON
50	KA LIGHT OFF	KA LIGHT OFF
51	KA LIGHT ON	KA LIGHT ON
52	KA LIGHT OFF	KA LIGHT OFF
53	KA LIGHT ON	KA LIGHT ON
54	KA LIGHT OFF	KA LIGHT OFF
55	KA LIGHT ON	KA LIGHT ON
56	KA LIGHT OFF	KA LIGHT OFF
57	KA LIGHT ON	KA LIGHT ON
58	KA LIGHT OFF	KA LIGHT OFF
59	KA LIGHT ON	KA LIGHT ON
60	KA LIGHT OFF	KA LIGHT OFF
61	KA LIGHT ON	KA LIGHT ON
62	KA LIGHT OFF	KA LIGHT OFF
63	KA LIGHT ON	KA LIGHT ON
64	KA LIGHT OFF	KA LIGHT OFF
65	KA LIGHT ON	KA LIGHT ON

66	KA LIGHT OFF	KA LIGHT OFF
67	KA LIGHT ON	KA LIGHT ON
68	KA LIGHT OFF	KA LIGHT OFF
69	KA LIGHT ON	KA LIGHT ON
70	KA LIGHT OFF	KA LIGHT OFF
71	KA LIGHT ON	KA LIGHT ON
72	KA LIGHT OFF	KA LIGHT OFF
73	KA LIGHT ON	KA LIGHT ON
74	KA LIGHT OFF	KA LIGHT OFF
75	KA LIGHT ON	KA LIGHT ON
76	KA LIGHT OFF	KA LIGHT OFF
77	KA LIGHT ON	KA LIGHT ON
78	KA LIGHT OFF	KA LIGHT OFF
79	KA LIGHT ON	KA LIGHT ON
80	KA LIGHT OFF	KA LIGHT OFF
81	KA LIGHT ON	KA LIGHT ON
82	KA LIGHT OFF	KA LIGHT OFF
83	KA LIGHT ON	KA LIGHT ON
84	KA LIGHT OFF	KA LIGHT OFF
85	KA LIGHT ON	KA LIGHT ON
86	KA LIGHT OFF	KA UP OFF
87	KA LIGHT ON	KA LIGHT ON
88	KA LIGHT OFF	KA LIGHT OFF
89	KA LIGHT ON	KA LIGHT ON
90	KA LIGHT OFF	KA LIGHT OFF
91	KA LIGHT ON	KA LIGHT ON
92	KA LIGHT OFF	KA LIGHT OFF
93	KA LIGHT ON	KA LIGHT ON
94	KA LIGHT OFF	KA LIGHT OFF
95	KA LIGHT ON	KA LIGHT ON
96	KA LIGHT OFF	KA LIGHT OFF
97	KA LIGHT ON	KA LIGHT ON
98	KA LIGHT OFF	KA LIGHT OFF
99	KA LIGHT ON	KA LIGHT ON
100	KA LIGHT OFF	KA LIGHT OFF
101	KA LIGHT ON	KA LIGHT ON
102	KA LIGHT OFF	KA LIGHT OFF
103	KA LIGHT ON	KA LIGHT ON
104	KA LIGHT OFF	KA LIGHT OFF

105	KA LIGHT ON	KA LIGHT ON
106	KA LIGHT OFF	KA LIGHT OFF
107	KA LIGHT ON	KA LIGHT ON
108	KA LIGHT OFF	KA LIGHT OFF
109	KA LIGHT ON	KA LIGHT ON
110	KA LIGHT OFF	KA LIGHT OFF
111	KA LIGHT ON	KA LIGHT ON
112	KA LIGHT OFF	KA LIGHT OFF
113	KA LIGHT ON	KA LIGHT ON
114	KA LIGHT OFF	KA LIGHT OFF
115	KA LIGHT ON	KA LIGHT ON
116	KA LIGHT OFF	KA LIGHT OFF
117	KA LIGHT ON	KA LIGHT ON
118	KA LIGHT OFF	KA LIGHT OFF
119	KA LIGHT ON	KA LIGHT ON
120	KA LIGHT OFF	KA LIGHT OFF
121	KA LIGHT ON	KA LIGHT ON
122	KA LIGHT OFF	KA LIGHT OFF
123	KA LIGHT ON	KA LIGHT ON
124	KA LIGHT OFF	KA LIGHT OFF
125	KA LIGHT ON	KA LIGHT ON
126	KA LIGHT OFF	KA LIGHT OFF
127	KA LIGHT ON	KA LIGHT ON
128	KA LIGHT OFF	KA LIGHT OFF
129	KA LIGHT ON	KA LIGHT ON
130	KA LIGHT OFF	KA LIGHT OFF
131	KA LIGHT ON	KA LIGHT ON
132	KA LIGHT OFF	KA LIGHT OFF
133	KA LIGHT ON	KA LIGHT ON
134	KA LIGHT OFF	KA LIGHT OFF
135	KA LIGHT ON	KA LIGHT ON
136	KA LIGHT OFF	KA LIGHT OFF
137	KA LIGHT ON	KA LIGHT ON
138	KA LIGHT OFF	KA LIGHT OFF
139	KA LIGHT ON	KA LIGHT ON
140	KA LIGHT OFF	KA LIGHT OFF
141	KA LIGHT ON	KA LIGHT ON
142	KA LIGHT OFF	KA LIGHT OFF
143	KA LIGHT ON	KA LIGHT ON

144	KA LIGHT OFF	KA LIGHT OFF
145	KA LIGHT ON	KA LIGHT ON
146	KA LIGHT OFF	KA LIGHT OFF
147	KA LIGHT ON	KA LIGHT ON
148	KA LIGHT OFF	KA LIGHT OFF
149	KA LIGHT ON	KA LIGHT OFF
150	KA LIGHT OFF	KA LIGHT OFF

Table C.1: Error rates of “KA LIGHT ON” and “KA LIGHT OFF”

Appendix D

Custom Dictionary

```
AGENT       EY  JH  AH  N  T
COMMAND K  AH  M  AE  N  D
DOWN        D  AW  N
EMERGENCY           IH  M  ER  JH  AH  N  S  IY
EMERGENCY(2)        IY  M  ER  JH  AH  N  S  IY
HUMIDITY            HH  Y  UW  M  IH  D  AH  T  IY
KA          K  AH  EY
LIGHT       L  AY  T
OFF         AO  F
ON          AA  N
ON(2)       AO  N
OPEN        OW  P  AH  N
STATUS      S  T  AE  T  AH  S
STATUS(2)           S  T  EY  T  AH  S
SHUTTER SH  AH  T  ER
TEMPERATURE         T  EH  M  P  R  AH  CH  ER
TEMPERATURE(2)      T  EH  M  P  ER  AH  CH  ER
TIME        T  AY  M
UP          AH  P
WINDOW      W  IH  N  D  OW
```

Bibliography

[1] M.-O. Pahl, "Distributed smart space orchestration," Dissertation, Technische Universität München, München, 2014.

[2] Ds2os – the distributed smart space orchestration system. (Last accessed on: November 26, 2016). [Online]. Available: http://ds2os.org/?site=projects/__DS2OS

[3] M.-O. Pahl, G. Carle, and G. Klinker, "Distributed smart space orchestration," in *Network Operations and Management Symposium 2016 (NOMS 2016) - Dissertation Digest*, May 2016.

[4] M.-O. Pahl and G. Carle, "Crowdsourced context-modeling as key to future smart spaces," *2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–8, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6838362

[5] M. A. Mandel, "A commercial large-vocabulary discrete speech recognition system: Dragondictate," *Language and speech*, vol. 35, no. 1, p. 237, Jan 01 1992, last updated - 2013-02-23. [Online]. Available: http://search.proquest.com/docview/1299113000?accountid=14439

[6] Siri. (Last accessed on: November 26, 2016). [Online]. Available: http://www.apple.com/ios/siri/

[7] What is cortana? (Last accessed on: November 26, 2016). [Online]. Available: https://support.microsoft.com/en-us/help/17214/windows-10-what-is

[8] U. Shrawankar and V. Thakare, "Techniques for Feature Extraction in Speech Recognition System : a Comparative Study," *International Journal Of Computer Applications In Engineering, Technology and Sciences (IJCAETS)*, pp. 412–418, 2013.

[9] K. Fellbaum, *Sprachverarbeitung und Sprachübertragung*, 2nd ed., 2012.

[10] B. Pfister and T. Kaufmann, *Sprachverarbeitung*, 2008.

[11] M. Shaneh and A. Taheri, "Voice command recognition system based on MFCC and VQ algorithms," *World Academy of Science, Engineering and ...*, 2009. [Online]. Available: http://www.waset.org/publications/4967

[12] A. V. Bhalla and S. Khaparkar, "Performance Improvement of Speaker Recognition System," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. March 2012, 2012.

[13] L. Muda, M. Begam, and I. Elamvazuthi, "Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques," vol. 2, no. 3, pp. 138–143, 2010.

[14] B. Shannon and K. Paliwal, "A comparative study of filter bank spacing for speech recognition," *Microelectronic engineering* …, vol. 41, pp. 2–4, 2003. [Online]. Available: https://maxwell.ict.griffith.edu.au/spl/publications/papers/merc03{_}ben.pdf

[15] L. R. Rabiner and B. H. Juang, "Speech recognition: Statistical methods," *Encyclopedia of Language Linguistics*, pp. 1–18, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B008044854200907X

[16] Cmu sphinx homepage. (Last accessed on: November 26, 2016). [Online]. Available: http://cmusphinx.sourceforge.net/

[17] Cmu sphinx - training acoustic model for cmusphinx. (Last accessed on: November 26, 2016). [Online]. Available: http://cmusphinx.sourceforge.net/wiki/tutorialam

[18] Cmu sphinx building language model. (Last accessed on: November 26, 2016). [Online]. Available: http://cmusphinx.sourceforge.net/wiki/tutoriallm

[19] G. Goth, "Deep or shallow, NLP is breaking out," *Communications of the ACM*, vol. 59, no. 3, pp. 13–16, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2897191.2874915

[20] Siri - everything you need to know! (Last accessed on: November 26, 2016). [Online]. Available: http://www.imore.com/siri

[21] ios security. (Last accessed on: November 26, 2016). [Online]. Available: https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[22] Siri gets 1 billion requests a week. (Last accessed on: November 26, 2016). [Online]. Available: http://gadgets.ndtv.com/mobiles/news/siri-gets-1-billion-requests-a-week-nearly-1-million-locations-to-accept-apple-pay-apple-701411

[23] Why microsoft named its siri rival 'cortana' after a 'halo' character. (Last accessed on: November 26, 2016). [Online]. Available: http://www.nbcnews.com/tech/mobile/why-microsoft-named-its-siri-rival-cortana-after-halo-character-n71056

[24] Cmu sphinx - basic concepts of speech. (Last accessed on: November 26, 2016). [Online]. Available: http://cmusphinx.sourceforge.net/wiki/tutorialconcepts

[25] Open-source large vocabulary csr engine julius. (Last accessed on: November 26, 2016). [Online]. Available: http://julius.osdn.jp/en_index.php

[26] Htk speech recognition toolkit. (Last accessed on: November 26, 2016). [Online]. Available: http://htk.eng.cam.ac.uk/

[27] Htk speech recognition toolkit faq. (Last accessed on: November 26, 2016). [Online]. Available: http://htk.eng.cam.ac.uk/docs/faq.shtml

[28] Htk speech recognition toolkit manual. (Last accessed on: November 26, 2016). [Online]. Available: http://htk.eng.cam.ac.uk/prot-docs/htkbook.pdf

[29] Jasper | control everything with your voice. (Last accessed on: November 26, 2016). [Online]. Available: http://jasperproject.github.io/

[30] Jasper | documentation configuration. (Last accessed on: November 26, 2016). [Online]. Available: http://jasperproject.github.io/documentation/configuration/

[31] Jasper | documentation hardware. (Last accessed on: November 26, 2016). [Online]. Available: http://jasperproject.github.io/documentation/hardware/

[32] Audeme homepage. (Last accessed on: November 26, 2016). [Online]. Available: http://www.audeme.com/movi.html

[33] Arduino homepage. (Last accessed on: November 26, 2016). [Online]. Available: https://www.arduino.cc/

[34] Movi kickstarter campaign. (Last accessed on: November 26, 2016). [Online]. Available: https://www.kickstarter.com/projects/310865303/movi-a-standalone-speech-recognizer-shield-for-ard

[35] Intel galileo. (Last accessed on: November 26, 2016). [Online]. Available: https://www.arduino.cc/en/ArduinoCertified/IntelGalileo

[36] Movi arduino shield photo. (Last accessed on: November 26, 2016). [Online]. Available: http://www.audeme.com/uploads/4/3/9/9/43997575/_7462811_orig.jpg

[37] Allwinner a13. (Last accessed on: November 26, 2016). [Online]. Available: http://linux-sunxi.org/A13

[38] Arduino ethernet shield. (Last accessed on: November 26, 2016). [Online]. Available: https://www.arduino.cc/en/Main/ArduinoEthernetShield

[39] Espeak homepage. (Last accessed on: November 26, 2016). [Online]. Available: http://espeak.sourceforge.net/

[40] Debian home page. (Last accessed on: November 26, 2016). [Online]. Available: https://www.debian.org/

[41] Cmu sphinx available acoustic models. (Last accessed on: November 26, 2016). [Online]. Available: https://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/

[42] Easy pronunciation phonetic transcription converter. (Last accessed on: November 26, 2016). [Online]. Available: http://easypronunciation.com/de/english-phonetic-transcription-converter

[43] Marytts - introduction. (Last accessed on: November 26, 2016). [Online]. Available: http://mary.dfki.de/

[44] Marytts - architecture walkthrough. (Last accessed on: November 26, 2016). [Online]. Available: http://mary.dfki.de/documentation/module-architecture.html

[45] Arduino mega shop. (Last accessed on: November 26, 2016). [Online]. Available: https://store.arduino.cc/product/A000067

[46] Raspberry pi 3 - allied electronics. (Last accessed on: November 26, 2016). [Online]. Available: http://www.alliedelec.com/raspberry-pi-raspberry-pi-3/70816528/

[47] What is arduino. (Last accessed on: November 26, 2016). [Online]. Available: http://www.arduino.org/learning/getting-started/what-is-arduino

[48] Arduino uno - technical specs. (Last accessed on: November 26, 2016). [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno

[49] Arduino mega - technical specs. (Last accessed on: November 26, 2016). [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMega2560

[50] Arduino shields. (Last accessed on: November 26, 2016). [Online]. Available: http://www.arduino.org/products/shields

[51] R. Pi, "Raspberry pi," *Raspberry Pi*, vol. 1, p. 1, 2012.

[52] Intel galileo. (Last accessed on: November 26, 2016). [Online]. Available: https://www.amazon.de/Raspberry-Pi-3-Model-B/dp/B01CEFWQFA/ref=sr_1_4?s=computers&ie=UTF8&qid=1481463356&sr=1-4&keywords=raspberry+pi+3

[53] A 15 pound computer to inspire young programmers. (Last accessed on: November 26, 2016). [Online]. Available: http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html

[54] Raspberry pi. (Last accessed on: November 26, 2016). [Online]. Available: https://www.raspberrypi.org/

[55] Raspberry pi 3, pi 2, b+, a+, compute module dev kit comparison chart. (Last accessed on: November 26, 2016). [Online]. Available: https://www.element14.com/community/docs/DOC-68090

[56] Java swing. (Last accessed on: November 26, 2016). [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/swing/

[57] Xamarin. (Last accessed on: November 26, 2016). [Online]. Available: https://www.xamarin.com/

[58] Mobile os market share. (Last accessed on: November 26, 2016). [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[59] ios 9 - homekit. (Last accessed on: November 26, 2016). [Online]. Available: http://www.apple.com/ios/homekit/

[60] Apple shop - phillip hue. (Last accessed on: November 26, 2016). [Online]. Available: http://www.apple.com/shop/product/HJCA2VC/B/philips-hue-white-and-color-wireless-ambiance-starter-kit-a19-e26

[61] Use the home app on your iphone, ipad and ipode touch. (Last accessed on: November 26, 2016). [Online]. Available: https://support.apple.com/de-de/HT204893

[62] Amazon echo. (Last accessed on: November 26, 2016). [Online]. Available: https://www.amazon.com/Amazon-Echo-Bluetooth-Speaker-with-WiFi-Alexa/dp/B00X4WHP5E

[63] Amazon echo picture. (Last accessed on: November 26, 2016). [Online]. Available: https://www.bhphotovideo.com/images/images2500x2500/amazon_b00x4whp5e_echo_1187819.jpg

[64] Hue white ambiance starter set. (Last accessed on: November 26, 2016). [Online]. Available: http://www.philips.de/c-m-li/hue-persoenliche-kabellose-beleuchtung/hue-white-ambiance

[65] Der eigene butler. (Last accessed on: November 26, 2016). [Online]. Available: http://www.heise.de/ct/ausgabe/2016-2-Digitaler-Assistent-mit-Offline-Spracherkennung-im-Eigenbau-3057626.html

[66] Edimax n150 wi-fi nano usb adapter. (Last accessed on: November 26, 2016). [Online]. Available: http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/wireless_adapters_n150/ew-7811un

[67] Akiro kinobo usb microphone. (Last accessed on: November 26, 2016). [Online]. Available: http://www.kinobo.co.uk/product/kinobo-akiro-usb/

[68] Spotify. (Last accessed on: November 26, 2016). [Online]. Available: https://www.spotify.com/

[69] Jasper | documentation usage. (Last accessed on: November 26, 2016). [Online]. Available: http://jasperproject.github.io/documentation/usage/

[70] Jasper | documentation modules. (Last accessed on: November 26, 2016). [Online]. Available: http://jasperproject.github.io/documentation/modules/

[71] element14 pi iot smarter spaces design challenge. (Last accessed on: November 26, 2016). [Online]. Available: https://www.raspberrypi.org/blog/element14-pi-iot-smarter-spaces-design-challenge/

[72] Lm tool. (Last accessed on: November 26, 2016). [Online]. Available: http://www.speech.cs.cmu.edu/tools/lmtool-new.html

[73] [pi iot] alarm clock 12: Voice control | element14 | pi iot. (Last accessed on: November 26, 2016). [Online]. Available: https://www.element14.com/community/community/design-challenges/pi-iot/blog/2016/08/02/pi-iot-alarm-clock-12-voice-control-repost

[74] [pi iot] alarm clock : Text to speech. (Last accessed on: November 26, 2016). [Online]. Available: https://www.element14.com/community/community/design-challenges/pi-iot/blog/2016/05/24/pi-iot-alarm-clock-01-project-description

[75] [pi iot] alarm clock 16: Wiring | element14 | pi iot. (Last accessed on: November 26, 2016). [Online]. Available: https://www.element14.com/community/community/design-challenges/pi-iot/blog/2016/08/15/pi-iot-alarm-clock-16-wiring

[76] [pi iot] alarm clock 1: Project description. (Last accessed on: November 26, 2016). [Online]. Available: https://www.element14.com/community/community/design-challenges/pi-iot/blog/2016/05/24/pi-iot-alarm-clock-01-project-description

[77] [demo video of frederick vandenbosch's smart alarm clock. (Last accessed on: November 26, 2016). [Online]. Available: https://www.youtube.com/watch?v=RgOo1w5xSqU

[78] Primefaces. (Last accessed on: November 26, 2016). [Online]. Available: http://www.primefaces.org/

[79] Java server faces. (Last accessed on: November 26, 2016). [Online]. Available: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[80] Glassfish server. (Last accessed on: November 26, 2016). [Online]. Available: https://glassfish.java.net/

[81] Git repository for dictionary script. (Last accessed on: November 26, 2016). [Online]. Available: https://github.com/cmusphinx/g2p-seq2seq/blob/master/g2p_seq2seq/g2p.py

[82] Audio recording script from chris wilson. (Last accessed on: November 26, 2016). [Online]. Available: https://webaudiodemos.appspot.com/AudioRecorder/js/main.js

[83] Github - mattdiamond/recorderjs: A plugin for recording/exporting the output of web audio api nodes. (Last accessed on: November 26, 2016). [Online]. Available: https://github.com/mattdiamond/Recorderjs

[84] Whatsapp. (Last accessed on: November 26, 2016). [Online]. Available: https://www.whatsapp.com/

[85] Whatsapp download. (Last accessed on: November 26, 2016). [Online]. Available: https://www.whatsapp.com/download/

[86] M. Weiser, "Some computer science issues in ubiquitous computing," *Cacm*, vol. 36, no. 7, pp. 75–84, 1993.

[87] M. Weiser, "The computer for the 21st Century," *IEEE Pervasive Computing*, vol. 1, no. 1, pp. 19–25, 2002. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=993141

[88] K. Lyytinen and Y. Yoo, "Issues and Challenges in Ubiquitous Computing," *Communications of the ACM*, vol. 45, no. 12, pp. 62–65, 2002.

[89] G. D. Abowd, "What next, ubicomp?" *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12*, p. 31, 2012. [Online]. Available: http://dx.doi.org/10.1145/2370216.2370222

[90] Different silos picture. (Last accessed on: November 26, 2016). [Online]. Available: https://ilab2.net.in.tum.de/courses/pictures/ilab2_net_in_tum_de-iLab2_winter_15_smart_space-course_b6e47ec0c1b73fb6493270b151e0d729/silos.png

[91] P. A. Bernstein, "Middleware," *Communications of the ACM*, vol. 39, no. 2, 1996.

[92] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms, 2/E*, 2007. [Online]. Available: https://vowi.fsinf.at/images/b/bc/TU_Wien-Verteilte_Systeme_VO_(G%C3%B6schka)_-_Tannenbaum-distributed_systems_principles_and_paradigms_2nd_edition.pdf