UNIVERSITY OF WEST BOHEMIA

Bachelor Thesis

Voice-enabled Smart Home Modules

Author:
Josef Šanda

Supervisor:
Ing. Martin Bulín, MSc.

A thesis submitted in fulfillment of the requirements for the degree of Bachelor (Bc.)

in the

Department of Cybernetics

April 19, 2021

Declaration of Authorship

I, Josef Šanda, declare that this thesis titled, "Voice-enabled Smart Home Modules" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

"*Something supersmart.*"

Your hero

UNIVERSITY OF WEST BOHEMIA

Abstract

Faculty of Applied Sciences
Department of Cybernetics

Bachelor (Bc.)

Voice-enabled Smart Home Modules

by Josef ŠANDA

Your abstract goes here...

Acknowledgements

Your acknowledgements go here...

Contents

List of Figures

List of Tables

List of Abbreviations

REST	**RE**presentational **S**tate **T**ransfer
JSON	**J**ava**S**cript **O**bject **N**otation
BSON	**B**inary **JSON**
MQTT	**M**essage **Q**ueuing **T**elemetry **T**ransport
HTTP	**H**yper**t**ext **T**ransfer **P**rotocol
CLI	**c**ommand-**l**ine **i**nterface
REST	**RE**presentational **S**tate **T**ransfer
ASR	**A**utomatic **S**peech **R**ecognition

Chapter 1

Introduction

Your intro... Thesis ref example: Bulín, 2016, Misc ref example: Šmídl, 2017, Article ref example: McCulloch and Pitts, 1943, Online webpage ref example: Bradley, 2006

1.1 State of the Art

1.2 Thesis Objectives

1.3 Thesis Outline

Chapter 2

Dialog Systems

2.1 Automatic Speech recognition

ASR (Automatic speech recognition) is a way of converting sound into text.

Sound is nothing more than vibrations of the air that we humans are trained exceptionally well to decode. Moreover, now, we are teaching our computers how to do this. In the beginning, we have a stream of words that a person has uttered. The sound is picked up by a microphone and converted to a digital signal through a sound card, which means a stream of ones and zeroes.

The first step the ASR system need to do is process the sound. It steps the sound to have chunks of speech (shown in Fig. 2.1) that can be worked with and that can be mapped to letters. These chunks are called phones.

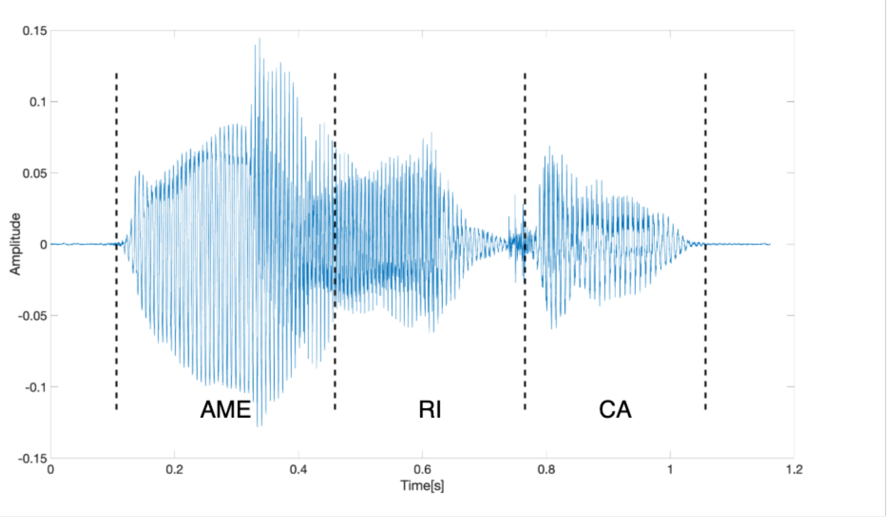

FIGURE 2.1: Chunked voice

The part of ASR responsible for mapping sound to phones is called the acoustic model as a set of building blocks, boxes which contain models for all phones in a given language as showing Fig. 2.2.

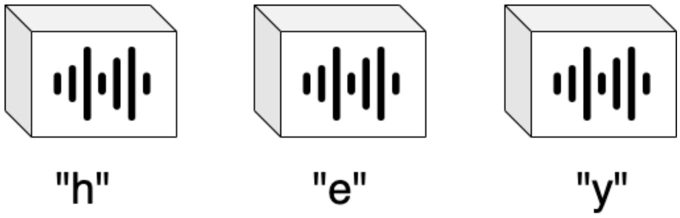

FIGURE 2.2: Phones boxes

There are boxes labelled, for example, A, B, C, depending on which phones are used in the particular language. On top of that, part of this construction set is also contextual probabilities. It means how likely a phone is to follow another. The acoustic model's task is to guess which phones have pronounced and how they combine into a word. The acoustic model processes the sound and compares it to the models of individual phones from its boxes. Since speech is very complex in a real scenario, the chunks that a person uttered will be similar to more than one box. The acoustic model takes this into account and also looks at the neighbouring chunks and their contextual probabilities. For example, string "HELLO" the first sound that a person uttered might have been H. However, it also could have been CH. The second sound looks most like E, but it could have been ə , A or even I, with different degrees of certainty. The next sound is probably L, but it also could be R. There are different probabilities of these phones in context, for example, H followed by E is more likely, at least in English, than H followed by I. The ASR system combines these bits of information and outputs the most likely result - a string of phones.(Stanislav, 2020)

The next step is to convert it into words. Nevertheless, this part can be tricky because the ASR does not know when a word starts or ends. Contrary to popular belief, there are no pauses between words in fluent speech. This particular string "heloumaj..." of phones can constitute several different phrases, for example, "hell oh my nay miss" or "hello mine aim is", or "hello my name is". The part of ASR responsible for mapping phones to words and phrases is called the language model.

Hidden Markov Models are widely used for the statistical approach for automatic speech recognition. Suppose that W=**** is a sequence of words, and O=**** is a sequence of phones. These sequences are taken with a period of 10 ms for segments of speech of length from 20 to 40 ms. To figure out a task to guess which phones have pronounced and how they combine into a word is used Bayes Theorem for conditional probability.

$$\boldsymbol{W'} = \underset{w}{\operatorname{argmax}} P(\boldsymbol{W} \mid \boldsymbol{O}) = \underset{w}{\operatorname{argmax}} \frac{P(\boldsymbol{W})P(\boldsymbol{O} \mid \boldsymbol{W})}{P(\boldsymbol{O})} \tag{2.1}$$

Where P(W) is the a priori probability of word W, P(O|W) is the probability that the sequence of phones O will be generated under the conditions of pronouncing the sequence of words W, P(O) is the a priori probability of the sequence of phones O.

Since the probability P(O) is independent of the sequence of words W, it is possible to modify the equation into the form:

$$\boldsymbol{W'} = \underset{w}{\mathrm{argmax}} P(\boldsymbol{W} \mid \boldsymbol{O}) = \underset{w}{\mathrm{argmax}} P(\boldsymbol{W}) P(\boldsymbol{O} \mid \boldsymbol{W}) \tag{2.2}$$

2.2 TextToSpeech - synthesis

The task of generating speech out of text information has originally two approaches:

1. Concatenative (unit selection)
2. Statistical parametric

With concatenative synthesis is based on sequential combining of shot prerecorded samples of speech. These samples can be stored in a database in the form of whole sentences, phrases, words and different phonemes. It depends on the application of the solutions. Building the unit selections synthesis model consists of three steps:

1. Recording of whole selected speech units in no possible context.
2. Labelling segmentation of units.
3. Choosing the most appropriate units.

The concatenated method is the most straightforward approach to speech generation. Disadvantages include the requirement to have ample storage for recorded units and an ability to apply various changes to a voice.

Statistical parametric synthesis consists of two parts, as shown in Fig. 2.3. The training step's approach is to extract excitation parameters like fundamental frequency and dynamic features, and spectral parameters from the speech database. Then we estimate them using one of the statistical models. The Hidden Markov model or HMM is the most widely used for this task. It should be noted that HMM is conscious dependent it means that in this step, in addition to phonetic context, linguistic and prosodic context is taken into account. In the synthesis part, at first given sentence is converted into points with a dependent label sequence, and then their chance HMM is constructed according to this sequence. Next, spectrum and excitation parameters are generated from the utterance HMM, and finally, speech waveforms are synthesized from these parameters using excitation generation and the speech synthesis filter. The advantages of the statistical parametric approach are:

1. Small footprint
2. No need to store speech waveforms, only statistics language independence
3. Flexibility in changing voice characteristics speaking styles and emotions.

The most noticeable drawback is the quality of a synthesized speech.

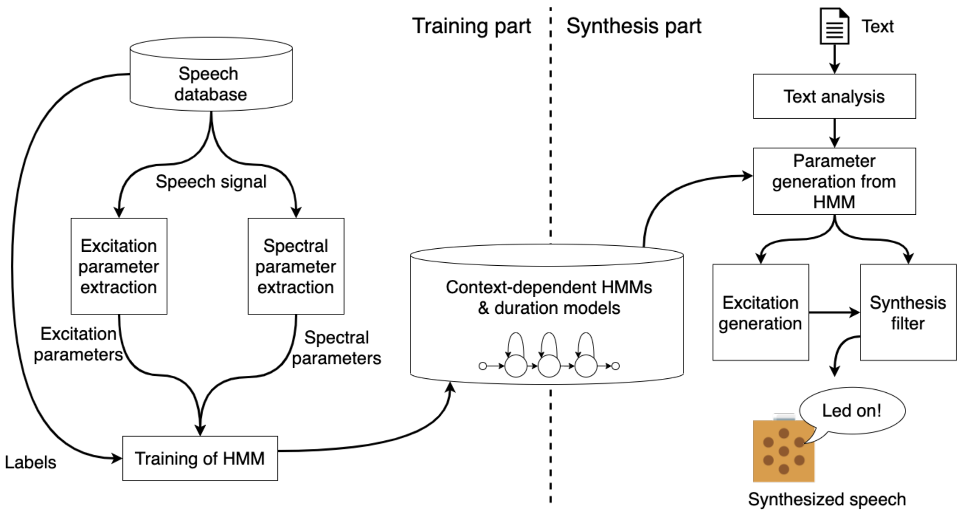

FIGURE 2.3: Synthesis model

2.3 KKY SpeechCloud

SpeechCloud is a system that connects ASR and TTS systems operating together via one interface. It is then possible to use these systems by many applications simultaneously through this interface. An independent instance is created for each dialogue system, allowing a client to create a characteristic language model, send a speech record to recognize, and receive the synthesized speech.

SpeechCloud provides the same services to all clients, unless otherwise limited or specified. Each client should have the same functions, but each device, experiment or project is separated from the others, so the results are not affected by the unwanted intervention.

The architecture of SpeechCloud and connection to client briefly visualizing Fig. 2.4.

SpeechCloud using the module SCAPIServer as a primary point for reach a connection with the client application. Thus, the module negotiates with the client the configuration for client applications, control communication channel, and authentication of a session. SCAPIServer also provides these pieces of information to other modules. The SIPSwitch module mediates the audio data transfer service between the SCWorker component and the client application. One instance of the SCWorker component is reserved for each client. This component instance holds one ASR and TTS instance. The SCWorker component has access to a TCP/IP network connection to obtain additional data sources.

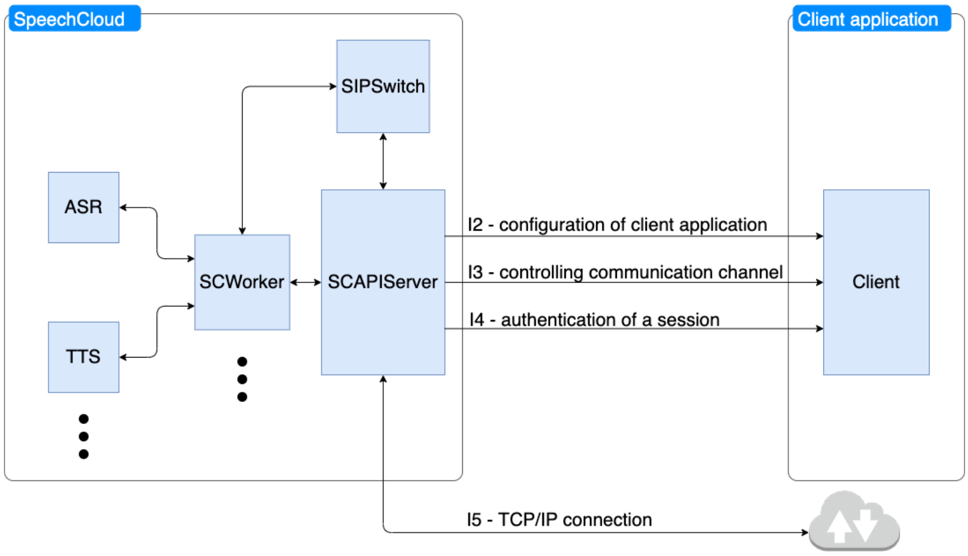

FIGURE 2.4: SpeechCloud schema

Solving the subject of the connection and transmission of data via Internet communication protocols is not the content of this work hence are used ready-made software components.

Chapter 3

Backend

Own engine running on Raspberry Pi 4 has been developed and serves as the backend for the project. The whole engine is coded in Python, and adheres to the following principles:

- *Simplicity*: write a straightforward code that is easily understandable for later rewriting.
- *Modifiability*: write a code with the ability to admit changes due to a new requirement or detect an error that needs to be fixed.
- *Modularity*: write a well-encapsulated code of modules, which do particular, well-documented functions.
- *Robustness*: write a code focusing on handling unexpected termination and unexpected actions.

3.1 Diagram description

This section briefly describes the architecture of the engine that is figured on a diagram - see Fig. 3.1.

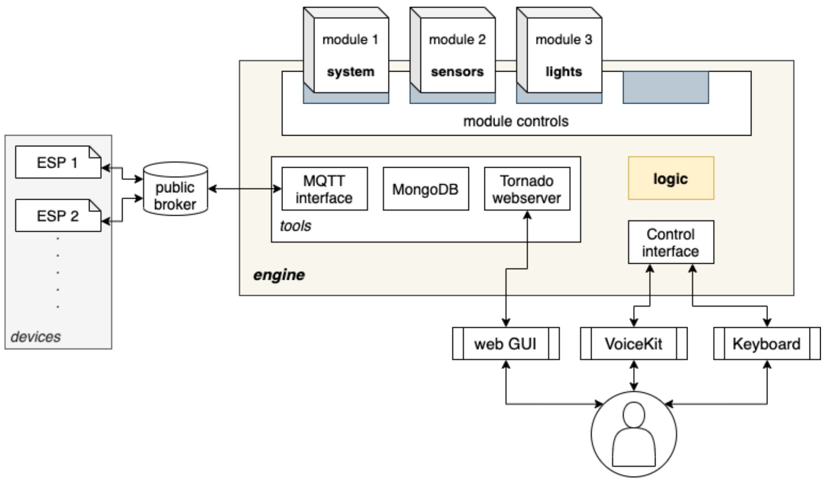

FIGURE 3.1: Project architecture

Engine uses tools like MQTT, MongoDB, Tornado web server that is described later. Each of them runs in its thread and concurrently. These tools create a basis for modules and mediate main functionalities such as database, web server and communication.

The engine is designed to easily remove, add or update any mutually independent modules that define functions used by a user interface. Each module is described in the Chap. 4.

The engine also contains a separate block for *logic*. This block captures a command from the VoiceKit or keyboard interface, then browsing a predefined list of each module's commands and determines the best match for the user voice command or command written on the keyboard. If it does not find the voice command in lists, it replies that the command has not found with the recognized command.

3.2 Database

MongoDB is an open-source document database built upon a NoSQL database and written in C++. Database's horizontal, scale-out architecture support vast volumes of both data and traffic. One document can have others embedded in itself, and there is no need to declare the structure of documents to the system - documents are self-describing.(Jayaram, 2020)

Before using this type of database, we have to be familiar with different terminology compare to traditional SQL databases:

SQL Server	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Column	Field
Joining	Linking & Embedding
Partition	Sharding (Range Partition)
Replication	ReplSet

TABLE 3.1: MongoDB terminology

We use this type of database because it is famous for its use in agile methodologies, and the project tends to enlarge in the future. The main benefits are:

- MongoDB is easy to scale.
- Schema-less database: we do not need to design the database's schema because the code we write defines the schema, thus saves much time.
- The document query language supported by MongoDB is simplistic as compared to SQL queries.
- There is no need for mapping application's objects to database's objects in MongoDB.
- No complex joins are needed in MongoDB. There is no relationship among data in MongoDB.

- Because of using JSON[1] format to store data, it is effortless to store arrays and objects.
- MongoDB is free to use. There is no cost for it.
- MongoDB is simple to set up and install.

For adding a new field, the field can be created without affecting all other documents in the collection, without updating a central system catalog, and without taking the system offline.

In the project, we save all incoming messages from MQTT to MongoDB to a collection based on a name of interest module.

3.3 Communication

Communication is the backbone of the whole project among several devices over the internet. Therefore, it had to be found robust, scalable, and cost-effective protocols that transmit messages and data securely.Based on the survey, we choose three protocols that, in combination, satisfy all our requirements, and we will delve deeper into them in the following sections.

3.3.1 MQTT

MQTT is a standardized protocol by the OASIS MQTT Technical Committee used for message and data exchange. The protocol is designed specifically for the Internet of Things. The protocol is developed in vast language diversity from low-level to high-level programming language and designed at light versions for low-performance devices. Hence, it suits our use-case perfectly because each module possesses tons of various devices with limited resources that are already included or will arise later on. (Malý, 2016)

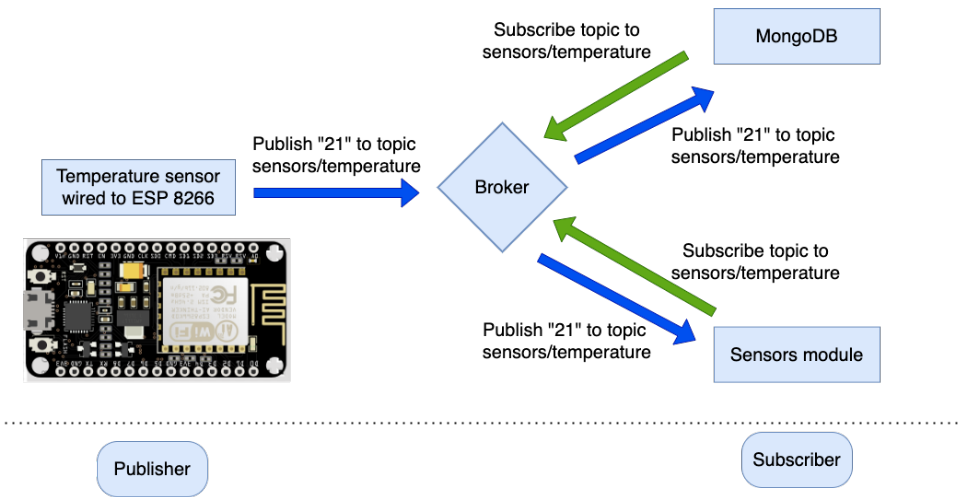

FIGURE 3.2: MQTT publisher/subscriber pattern

The design principles are to minimize network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. The protocol determines errors by TCP

[1] **J**ava**S**cript **O**bject **N**otation

and orchestrates communication by the central point - broker. The protocol architecture uses a publish/subscribe pattern (also known as pub/sub) shown in Fig. 3.2, which provides an alternative to traditional client-server architecture. Architecture decouples publishers and subscribers who never contact each other directly and are not even aware that the other exist. The decoupling give us the following advantage:

- *Space decoupling*: publisher and subscriber do not need to know each other.
- *Time decoupling*: publisher and subscriber do not need to run at the same time.
- *Synchronization decoupling*: operations on both components do not need to be interrupted during publishing or receiving.

When the publisher sends his message, it is handled by the broker who filters all incoming messages and distributes them to accredited subscribers. The filtering is based on topic or subject, content and type.

In the case of MQTT, the filtering is subject-based and therefore, every message including a subject or a topic. The client subscribes to the topics he is interested in, and the broker distributes the messages accordingly as shown in Fig. 3.3.

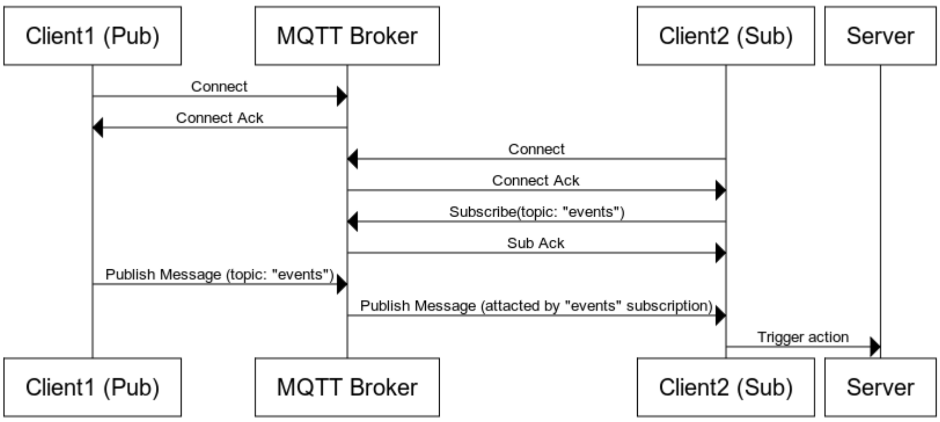

FIGURE 3.3: Diagram illustrating how communication in MQTT flow.

The topics are generally strings with a hierarchical structure that allow different subscription levels. It is feasible to use wildcards to subscribe, for example, sensors/# to receive all messages related to the sensors, for example, sensors/temperature or sensors/illuminance.

The MQTT protocol has the Quality of Service (QoS) levels essential to any communication protocol. The level of QoS can be specified for each message or topic separately according to its importance.

In MQTT, there are 3 QoS levels:

- *QoS 0*: This level is often called "fire and forget" when a message is not stored and retransmitted by a sender.

- *QoS 1*: Is is guarantees that a message is delivered at least one time to the receiver. The message is stored on a sender until it gets a PUBACK packet from a receiver.
- *QoS 2*: It is the highest level, and it guarantees that each message received only once by the intended recipients.

It is vital to mention MQTT have the feature retained messages that are mechanisms where the broker stores the last retained message for a specific topic. This feature allows a client does not have to wait until a new message is published to know the last known status of other devices.

3.3.2 WebSocket

In this work, WebSockets are used to provide communication between the client and the engine. WebSocket provides a low-latency, persistent, full-duplex connection between a client and server over TCP. The protocol is chiefly used for a real-time web application because it is faster than HTTP concerning more transfers by one connection. The protocol belongs to the stateful type of protocols, which means the connection between client and server will keep alive until either client or web server terminate it. The protocol fits for us in use between client and web server in case of real-time response.(Wang, Salim, and Moskovits, 2013)

The main benefits are:

- *Persistent*: After an initial HTTP handshake, the connection keeps alive using a ping-pong process, in which the server continuously pings the client for a response. It is a more efficient way than establishing and terminating the connection for each client request and server response. Server terminating connection after an explicit request from the client, or implicitly when the client goes offline.
- *Secure*: WebSocket Secure uses standard SSL and TLS encryption to establish a secure connection. Although we do not pursue this issue in our work, it is a valuable feature to add later.
- *Extensible*: Protocol is designed to enabling the implementation of sub-protocols and extensions of additional functionality such as MQTT, WAMP, XMPP[2], AMQP[3], multiplexing and data compression. This benefit makes WebSockets a future-proof solution for the possible addition of other functionalities.
- *Low-latency*: WebSocket significantly reduces each message's data size, drastically decreasing latency by eliminating the need for a new connection with every request and the fact that after the initial handshake, all subsequent messages include only relevant information.
- *Bidirectional* - This enables the engine to send real-time updates asynchronously, without requiring the client to submit a request each time, as is the case with HTTP.

[2]**Ex**tensible **M**essaging and **P**resence **P**rotocol - messaging and presence protocol based on XML and mainly used in a near-real-time exchange of structured data.

[3]**A**dvanced **M**essage **Q**ueuing **P**rotocol - an open standard application layer protocol for message-oriented middleware.

We will apply this protocol for transfer between clients such as VoiceKit, keyboard or web interface and engine in case of real-time response.

3.3.3 REST

In other cases like fetching data only once or data that is not required very frequently, we use RESTfull web service on a web server. This service enables us to transfer a lightweight data-interchange format JSON trivially and reliably - see Fig. 3.4. We use a standard GET REST request on a defined URI and then decode it like JSON for fetching data.

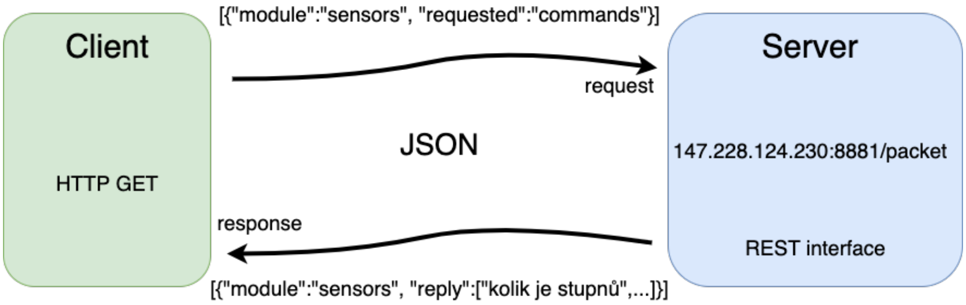

FIGURE 3.4: REST principle

3.4 Controllers

3.4.1 Keyboard

The keyboard is a python script with a particular purpose for developing new voice commands. This script opens up a CLI built upon a voicehome controller. The developer can quickly type a voice command with high accuracy through the command-line and debug the command thoroughly in various forms.

3.4.2 VoiceKit

VoiceKit is a building kit made by Google that lets users create their natural language processor and connect it to the Google Assistant or Cloud Speech-to-Text service. By pressing a button on top, users can ask questions and issue voice commands to their programs. All of this fits in a handy little cardboard cube powered by a Raspberry Pi.(*Voice Kit*)

3.4.3 Website

The second interface next to the already mentioned VoiceKit is a website. The webserver is implemented in Python using the Tornado framework.

Chapter 4

Modules

test

Chapter 5

Examples

5.1 XOR Function

Chapter 6

Discussion

Discussion starter...

6.1 Recapitulation of Methods

6.2 Summary of Results

Chapter 7

Conclusion

Conclusion text...

MongoDB project's application is as simple as possible because it is not the topic of the thesis. Is there plenty of room for improvement and streamlining.

7.1 Future Work

Outlook...

Bibliography

[1] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[2] Peter Bradley. *The XOR Problem and Solution.* 2006. URL: http://mind.ilstu.edu/.

[3] Vanessa Wang, Frank Salim, and Peter Moskovits. "The WebSocket API". In: *The Definitive Guide to HTML5 WebSocket* (2013), 13–32. DOI: 10.1007/978-1-4302-4741-8_2.

[4] Martin Bulín. "Classification of terrain based on proprioception and tactile sensing for multi-legged walking robot". MA thesis. Campusvej 55, 5230 Odense M: University of Southern Denmark, June 2016.

[5] Martin Malý. *Protokol MQTT: komunikační standard pro IoT.* 2016. URL: https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/.

[6] Luboš Šmídl. personal communication. supervision of the thesis. 2017.

[7] Prashanth Jayaram. *When to Use (and Not to Use) MongoDB - DZone Database.* 2020. URL: https://dzone.com/articles/why-mongodb.

[8] Petr Stanislav. "Speech recognition of patients after total laryngectomy communicating by electrolarynx". PhD dissertation. Západočeská univerzita v Plzni, 2020.

[9] *Voice Kit.* URL: https://aiyprojects.withgoogle.com/voice/.

Appendix A1

Structure of the Workspace

- root
  - officials
  - literature
  - data
    - data_mnist
    - data_speech
  - py
    - examples
      - karnin
      - mnist
      - rpe
      - speech
      - train
      - xor
    - kitt_lib
    - scripts
  - results
  - progress_reports
  - thesis