



**BRNO UNIVERSITY OF TECHNOLOGY**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

**FACULTY OF INFORMATION TECHNOLOGY**

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

**DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA**

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

**VOICE DIALOG SYSTEM IN WEB BROWSER  
FOR DEMONSTRATION PURPOSES**

HLASOVÝ DIALOGOVÝ SYSTÉM VE WEBOVÉM PROHLÍŽEČI PRO DEMONSTRAČNÍ ÚČELY

**MASTER'S THESIS**

DIPLOMOVÁ PRÁCE

**AUTHOR**

AUTOR PRÁCE

**Bc. PAVOL VLČEK**

**SUPERVISOR**

VEDOUCÍ PRÁCE

**Ing. PETR SCHWARZ, Ph.D.**

BRNO 2021

# Master's Thesis Specification



Student: **Viček Pavol, Bc.**

Programme: Information Technology and Artificial Intelligence

Specialization: Computer Networks

n:

Title: **Voice Dialog System in Web Browser for Demonstration Purposes**

Category: Speech and Natural Language Processing

Assignment:

1. Verify the functionality of the automatic dialog system supplied by Phonexia in combination with Phonexia Speech Engine, Phonexia Voicebot Suite, Asterisk and SIP softphone
2. Study the possibilities of creating a voice communication channel between a web browser and an Asterisk virtual telephone exchange using WebRTC. Focus on signaling using the SIP protocol. Address the issue of communication through NAT on the client side.
3. Create a plugin that allowing to insert the presentation part of the automatic dialog system into web pages. The plugin will communicate with the virtual telephone exchange via WebRTC. Address the context of the transition between web pages.
4. Create a demo dialog. The topic can be a presentation of your own dialogue system, a presentation of some speech technologies (for example, automatic transcription, language identification, speaker identification), a presentation of the speech group or of Phonexia. Focus on the combination of voice and graphic forms of presentation, especially their synchronization.
5. Test the system on a user test group. Discuss the proposed solution, the results of user tests and the possibilities of extension of the solution.

Recommended literature:

- Based on consultation with the advisor.

Requirements for the semestral defence:

- Items 1 and 2.

Detailed formal requirements can be found at <https://www.fit.vut.cz/study/theses/>

Supervisor: **Schwarz Petr, Ing., Ph.D.**

Head of Department: Černocký Jan, doc. Dr. Ing.

Beginning of work: November 1, 2020

Submission deadline: July 30, 2021

Approval date: October 30, 2020

## Abstract

This thesis describes how to prepare and design a voice-controlled assistant(voicebot), which can be deployed on any website as a modern way to communicate with customers using internet browsers. The main emphasis is put on synchronization between voice dialog and the graphical interface of the website. The synchronization can be achieved by transferring bidirectional voice and text commands between client and server. This is achieved by using WebRTC technology with SIP as a signaling protocol. The thesis deals with a wide range of protocols and technologies as well as interconnecting VoIP telephony, computer networks, and Phonexia speech technologies based on machine learning. As a result, deployment of the voicebot can reduce costs on outgoing calls, ease agents of a FAQ burden, and increase customers' interest in the product/company.

## Abstrakt

Cieľom práce je navrhnúť a vytvoriť hlasom ovládaného asistenta(voicebota), ktorý bude ľahko nasaditeľný na webovú stránku. Používateľom tak bude poskytnutý moderný spôsob, ako prirodzene komunikovať cez internetový prehliadač. Hlavný dôraz je kladený na synchronizáciu medzi hlasovým asistentom a obsahom na webovej stránke. Synchronizácia je dosiahnutá obojsmerným prenosom hlasu a textových príkazov medzi klientom a serverom. Na to je použitá technológia WebRTC v kombinácii so signalizačným protokolom SIP. Práca sa zaoberá oblasťami ako VoIP telefonovanie, počítačové siete a strojové učenie(proprietárne rečové technológie od Phonexie). Benefitom nasadenia hlasového asistenta je zníženie nákladov na odchádzajúce hovory pre klientov, odľahčenie agentov na call centrách pri odpovedaní na často kladené otázky a zvýšenie záujmu zákazníkov vďaka použitiu nových technológií.

## Keywords

voicebot, WebRTC, SIP, RTP, SDP, Asterisk, voice dialogue system, NAT traversing, Phonexia, speech recognition, internet browser, IBM Watson, WebSocket, Phonexia Speech Engine

## Klíčové slová

voicebot, WebRTC, SIP, RTP, SDP, Asterisk, hlasový dialógový systém, prechod NAT, Phonexia, rozpoznávanie reči, internetový prehliadač, IBM Watson, WebSocket, Phonexia Speech Engine

## Reference

VLČEK, Pavol. *Voice Dialog System in Web Browser for Demonstration Purposes*. Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Petr Schwarz, Ph.D.

## Rozšírený abstrakt

Technológie na spracovanie prirodzeného jazyka sa vďaka dátam a algoritmom umelej inteligencie neustále zlepšujú. Rozpoznávanie svetových jazykov je na vynikajúcej úrovni a postupne sa doťahujú aj menej rozšírené jazyky ako čeština a slovenčina. Prirodzeným dôsledkom je využitie týchto technológií v praxi, napríklad v prostredí call centier. Už dnes sa môžeme stretnúť s hlasovým asistentom (voicebotom) na zákazníckej linke.

Cielom tejto práce je posunúť využitie voicebotov ešte o krok ďalej a navrhnúť takého hlasového asistenta, ktorý bude synchronizovaný s webovou stránkou a spolu budú tvoriť jeden celok. Na prenos audio-textových informácií sme použili digitálne technológie z telekomunikačnej praxe (VoIP). Tento problém je veľmi komplexný, preto sme prácu rozdelili na niekoľko podčastí a komponentov. Medzi hlavné komponenty patria: webový klient, telefónna ústredňa a voicebot. Telefónna ústredňa má podobnú funkciu ako router na počítačovej sieti – manažovať účastníkov, čiže nadväzovať, ukončovať a smerovať hovory medzi klientami. Ako telefónnu ústredňu sme použili open-source program Asterisk.

V prvej časti riešime ako preniesť zvuk a textové správy medzi web klientom v prehliadači a softvérovou telefónnou ústredňou. Na ich spoločnú komunikáciu sme použili technológiu WebRTC. Skratka WebRTC (Web Real-Time Communication) označuje komunikáciu v reálnom čase prostredníctvom webu. Technológia WebRTC rieši okrem prenosu dát aj nadviazanie end-to-end spojenia. Na internete sa zvyknú nachádzať zariadenia za NATom (obojsmerný preklad lokálnych IP adries na verejné pomocou portov) alebo majú nakonfigurované špecifické pravidlá vo firewalli. Na tento účel sa využíva technológia ICE (Interactive Connectivity Establishment). ICE zabezpečuje vzájomnú výmenu IP adries a otvorenie portov na nadviazanie komunikácie. Proces hľadania ICE kandidátov je zautomatizovaný a priamo integrovaný do WebRTC. Najskôr sa strany pokúsia spojiť na priamo. Ak sa im to nepodarí, oslovia preddefinovaný STUN server, ktorý je verejne dostupný a jeho úlohou je odpovedať na žiadosti verejnou IP adresou žiadateľov. Ak nie je z internetu dosiahnuteľný ani web klient, ani voicebot, technológia ICE vie presmerovať komunikáciu cez ďalší uzol tzv. TURN server. TURN server musí byť verejne dostupný a celá komunikácia prechádza cez neho. Samotná komunikácia pozostáva zo signalizačnej časti a mediálnej časti. V signalizačnej časti sa vymieňajú stavové správy, napríklad na inicializáciu alebo ukončenie hovoru. Na signalizáciu a neskôr aj na prenos textových správ sme použili SIP protokol. V mediálnej časti sa prenáša zvuk vo forme SRTP datagramov. Webové prehliadače sú však založené iba na protokole HTTP, resp. HTTPS. Signalizačné SIP pakety je potrebné zapuzdriť cez šifrovaný WebSocket(WSS) a preniesť na telefónnu ústredňu. O zapuzdrenie sa stará knižnica JsSip.js ktorá je na webovom klientovi. Telefónna ústredňa Asterisk má možnosť zapnúť šifrovaný WebSocket(WSS) a tak dosiahneme podporu na oboch stranách. Protokol Secure WebSocket na začiatku funguje na protokole HTTPS, kde sa vymenia inicializačné správy a po inicializácii sa komunikácia prenáša cez Secure WebSocket.

V druhej časti sa zaoberáme prenášaním zvuku a signalizačných správ medzi telefónnou ústredňou a tzv. voicebot backendom. Voicebot backend je program, na ktorý sa dá “dovolať”. Asterisk rozbalí potrebné informácie zo Secure WebSocket protokolu a ďalej ich smeruje podľa SIP hlavičiek. Voicebot backend spája vrstvu rozpoznávania reči, vrstvu na spracovanie prirodzeného jazyka, dialógovú vrstvu a vrstvu prevodu textu na reč. Najdôležitejšou časťou voicebot backendu je vrstva rozpoznávania reči - SpeechEngine. Je to nástroj, ktorý dokáže spracovať zvukový tok a prepísať reč na text v reálnom čase. SpeechEngine je vyvinutý českou firmou Phonexia. Tento nástroj dokáže detekovať pauzy v reči a priebežne ju posielat na spracovanie do ďalších vrstiev. Ako dialógovú vrstvu

sme použili externú službu IBM Watson. IBM Watson je služba, ktorá ponúka nástroj na tvorbu dialógov a spracovanie prirodzeného jazyka. Táto služba je navrhnutá najmä na tvorbu chatbotov, pričom voicebot funguje na rovnakom princípe, akurát potrebuje navyše 2 vrstvy: prepis reči na text a prevod textu na reč. Vrstvu prevodu textu na reč zabezpečuje ďalšia externá služba poskytnutá firmou SpeechTech. Posledná vrstva je skôr doplnková a slúži na dodatočné spracovanie prirodzeného jazyka. Je napísaná v jazyku Python, pretože väčšina NLP nástrojov vyžaduje programovací jazyk Python a s Voicebot backendom komunikuje cez svoj vlastný HTTP server. Jazyk voicebota je nastavený na češtinu. Ako už bolo spomenuté, voicebot integruje všetky tieto vrstvy do jedného celistvého riešenia.

Ďalšia časť celého riešenia je demonštračná internetová stránka, z ktorej je možné na voicebota zavolať. Táto stránka obsahuje dve hry a analýzu hovoreného textu (analýza slovných druhov cez knižnicu Stanza a získavanie znalostí cez Google Knowledge Graph API). Jedna z hier na stránke je kartová hra Pexeso, ktorá sa dá ovládať pomocou hlasu. Na tejto hre je najlepšie vidieť vzájomnú synchronizáciu medzi hlasom a grafickou časťou stránky. Posledná časť popisuje návrh dialógu v službe IBM Watson tak, aby bol prepojený s obsahom na webovej stránke. Cez SIP správy prenášame príkazy a stavové premenné, ktoré sú vopred dohodnuté ako vo webovom klientovi, tak aj na službe IBM Watson. Prostredníctvom tohto dialógu je možné hrať na stránke dve demonštračné hry.

Táto práca splnila zadanie a jej prínos je nespochybniteľný. Nielenže ukázala možnosť nasadenia hlasového asistenta priamo na internetovej stránke, ale ukazuje aj potenciál do budúcnosti ako spraviť hlasových asistentov intuitívnejších v kombinácii s grafickým rozhraním. V budúcnosti je priestor aj na ďalšie vylepšenia. Jedno zo zaujímavých možností je aj využitie rôznych funkcionalít SpeechEnginu, napríklad nielen podpora pre identifikáciu jazyka, ale aj hovoriaceho, či pohlavia alebo odhad veku hovoriaceho.

# Voice Dialog System in Web Browser for Demonstration Purposes

## Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Mr. Petr Schwarz. The supplementary information was provided by Mr. Evzen Polenka. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....  
Pavol Vlček  
July 22, 2021

## Acknowledgements

I would like to express thanks to my supervisor Mr. Schwarz and the Phonexia company for providing an opportunity to letting me explore such an interesting area. Special thanks to my fiancée Anna for being patient while finishing this thesis just a few days before our wedding.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	A communication and modern technologies . . . . .	2
1.2	Current trends . . . . .	2
1.3	Motivation . . . . .	3
<b>2</b>	<b>Technical background</b>	<b>5</b>
2.1	WebRTC . . . . .	6
2.2	Telephony . . . . .	8
2.3	Voicebot systems . . . . .	15
<b>3</b>	<b>System design</b>	<b>18</b>
3.1	General concept . . . . .	18
3.2	Custom voicebot system . . . . .	18
3.3	Voicebot deployment . . . . .	20
<b>4</b>	<b>Implementation</b>	<b>23</b>
4.1	Initial approach . . . . .	23
4.2	JavaScript's SIP client . . . . .	23
4.3	ICE support in Asterisk . . . . .	26
4.4	Asterisk's configuration files . . . . .	27
4.5	Security . . . . .	30
4.6	Phonexia voicebot suite configuration . . . . .	31
4.7	STUN and TURN server . . . . .	32
4.8	Web presentation . . . . .	34
4.9	Synchronization of voicebot and web client using instant SIP messages . . . . .	35
<b>5</b>	<b>Dialog system and testing</b>	<b>38</b>
5.1	TTS . . . . .	38
5.2	Dialog system . . . . .	39
5.3	Demonstration dialog design . . . . .	40
5.4	Testing . . . . .	43
<b>6</b>	<b>Conclusion</b>	<b>45</b>
	<b>Bibliography</b>	<b>47</b>
<b>A</b>	<b>Demonstration website</b>	<b>50</b>
<b>B</b>	<b>Description of the digital part of the work</b>	<b>52</b>

# Chapter 1

## Introduction

Revolutionary inventions do not appear overnight. They are a combination of small gradual innovations. Our motivation was to bring voice technologies closer to people and thus make lives easier. Our assumptions are the following: People need to communicate, they have internet access, capable web browsers, hardware, and batteries are still improving, and speech technologies understand us very well already. It seems that now is the right time to start with automated dialog systems — voicebots. The goal of this thesis is to supplement a voice dialog with visual information. Web-based voicebots are not widely spread yet, but as almost 95% of all browser installations currently support WebRTC technology for real-time audio transfer, it was a natural choice for us to go.

### 1.1 A communication and modern technologies

During the Covid-19 epidemy, the industries replaced live meetings with online meetings. Businesses had to transform from classical paper to digital processes quickly. Fortunately, the WebRTC technology brought the teleconferencing clients to web browsers and made the attendance of online meetings more convenient. One does not need to install various desktop applications because we can do everything right from the browser. We can see that voicebots became popular as a way to communicate with customers and save some costs. Observing the huge progress of voice technologies in recent years, developing voicebots and bring them closer to the customers is natural. The WebRTC technology opens us a possibility to insert voicebots to websites, removing the need to install specific programs on our computers. Companies might want to deploy voicebots to improve their customer support and to attract more customers. Voicebots can help, especially when there are peaks in the number of calls when the agents are unavailable. The main benefits of deploying voicebots are scalability of the service and 24/7 availability.

### 1.2 Current trends

Thanks to improved Bluetooth chips for audio transmission with increased battery life, wireless headphones have become very popular. The technology is so advanced that the mobile phone manufacturers stopped installing jack connectors to their devices. Wireless headphones have built-in high-quality microphones and often support voice assistants as well. Also, laptops are equipped with high-quality microphone arrays used for high-quality distant speech recognition.



The future is in voice and visual communication, in our opinion. Answers to simple questions will be provided by voice, and answers to more complex questions will be displayed on a screen, on a smartwatch, smartphone, or on a computer screen. We noticed that some websites (e.g., Decathlon<sup>1</sup>) use a voice search through Web Speech API<sup>2</sup> already. Most of the devices where the web page is displayed have an inbuilt microphone. This API uses a default speech recognition system from the device - most modern operating systems have a speech recognition system. The most popular ones are Dictation on macOS, Siri on iOS, Cortana on Windows 10, and Android Speech on Android. This feature can be very convenient considering devices without any physical keyboard or any keyboard at all.

### 1.3 Motivation

The primary motivation for this work was to find such a solution where Phonexia, a company developing voice solutions, can present its product on the web live. When anyone wants to try a voice dialog with a voicebot now, he/she need to call a phone number. The users, especially from abroad, are worried about paying high fees to the mobile network operator. Also, the possibility to present visual information together with the voice answer is exciting. My personal motivation was to help this exciting market with voice technologies to grow and be a part of it. In recent years, voice technologies became a major part of general-purpose voice assistants (Google Assistant, Siri, Amazon Alexa). Through third-party voice applications (skills), these assistants can be extended, mainly for entertainment and automation of everyday life-oriented tasks.

We believe there are two ways how voicebots can spread in the future: The first one is to extend general-purpose voice assistants with custom voicebots and to achieve some interface between them. There is a big potential in the future in a way that custom voicebots can be integrated with voice assistants like Siri, or Google Assistant. Just imagine the following use case. A user is traveling in a car and asks Siri to get a voicebot of a website cityparking.com. The general voice assistant transfers the call to the website's hosted voicebot. The user can book and confirm the payment for the parking with his voice straight away then. The thing is that every human voice is to a large extent unique, analog to a fingerprint and thus the voice can be used as a digital signature. The service is not only limited to parking but there are many use cases. E.g: cinema (which movies are playing in cinema bestcinema.com), hotel (is it possible to bring pets to superhotel.com), or interactive sightseeing of the city (go to coolcity.com, tell me the history of this monument). We assume that a general voice assistant can't be good in every particular area. Also, the voice recognition's accuracy could be improved when the custom voicebot will be trained on a specific domain.

Another opportunity to expand the voice technology market is to combine the voice dialog with visual information, for example, presented through a web browser. Let us assume that a customer has a question about how to assembly some IKEA furniture. If the answer is complicated, it can not be provided to a user in a simple voice response. It needs to be split into more steps, and a graphical step-by-step tutorial is a huge benefit. Everybody confirms it is better to see the diagram with spoken instructions rather than hear the instructions many times. Especially for the customers, such systems are more convenient. They can ask questions immediately and do need to call technical support. Both customers and company save some money - companies do not need to pay for people

---

<sup>1</sup><https://www.decathlon.cz/>

<sup>2</sup>[https://developer.mozilla.org/en-US/docs/Web/API/Web\\_Speech\\_API/Using\\_the\\_Web\\_Speech\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API)

on the support lines, customers do not need to pay some phone costs, and the answer is provided immediately so that the customer experience can be higher. Still, if the question is too complicated, the question can be redirected to a human agent. In this case, the voicebot collects the necessary information from the customer and passes it to an agent (maybe with a proposed answer) that speeds up the process. The commercial use of voicebot systems offers better services (always available, no waiting), reduces the load on call centers, and saves some money.

## Chapter 2

# Technical background

This chapter pays attention to a necessary technical background. We turn our attention to WebRTC technology, which is the substance of the solution. Besides that, we examine telephony and its protocols, and we also take a closer look at the process of how can two sites reach each other on Internet - NAT traversal.

### A replacement of multiple applications by a web browser

Web browsers are probably the most used applications on desktops and smartphones. The significant advantage of a web application is availability. It does not mean only an always-on service, but also an everywhere-with service. Almost every computer has a pre-installed web browser with an internet connection. And the devices are often portable. Internet browsers and their APIs are being developed continuously while adding new features for users. Web technologies are so popular that even some native mobile applications use a simplified encapsulated browser. Nobody doubts that web services, together with a thin web browser-based client, are a trend. These days we can do most of the everyday tasks on computers through a web browser. The following list shows popular activities done on computers online through a browser together with desktop applications that replaced:

1. Document reading (native support of PDF format in browser) – previously Adobe reader
2. Music listening (Spotify/native support of MP3 and other formats in browser) – previously Windows Media Player
3. Video watching (YouTube, native support of MP4 in browser) – previously Windows Media Player
4. Work with documents (Office 365 online, Google Docs) – previously Microsoft Office
5. Email reading (online Mail clients) – previously Microsoft Outlook/Thunderbird
6. File access (Cloud technologies) – previously 500GB/1TB HDDs full of movies/photos/applications/songs

### Taking advantage of WebRTC technologies one can additionally:

7. Play games (Google Stadia<sup>1</sup>) – previously CD/DVD installed games/Steam

---

<sup>1</sup><https://stadia.google.com/>

8. Make calls (Facebook Messenger, Microsoft Teams, Discord) – previously Skype
9. Share your desktop – previously TeamViewer

## 2.1 WebRTC

Web Real-Time Communication (WebRTC) [17] is a web technology connecting multiple Application Programming Interfaces (APIs), allowing to interchange audio, video, and data, or any combination of them with other WebRTC client. We can program these web-based applications in the browser using JavaScript and HTML5.

This technology became very popular because of its cutting-edge features: Firstly, it supports direct peer-to-peer (P2P) communication and offers low latency real-time communication. Secondly, it is convenient for both users and developers. The users do not have to install any additional plugins or download native apps. The developers can take advantage of APIs and get high-performance and meaningful applications with few lines of code. Thirdly, the data and control logic is not tied firmly together. While WebRTC does not offer any signaling protocol, it provides excellent flexibility to implement either proprietary or standard signaling protocols.

### Historical context of WebRTC

The idea for WebRTC started in late 2009 [9]. It was a period when browsers were trying to fill the functionality gap between desktop and internet applications. One of such challenges was the ability to communicate in real-time. There were some Real-Time Communication (RTC) implementations available already. Adobe was offering its famous Flash player, and there were also some third-party plugins. The main disadvantages were, among others, the requirement of servers, low-quality communication, and a necessity to update these plugins by users often. Google has invested in web browsers a lot and launched Chrome in 2008. They saw an opportunity in the Global IP Solutions (GIPS) company, with low-level components for RTC. Google acquired GIPS in January 2011. Having both employees and technology, Google formed the WebRTC Chrome team. The WebRTC support was released officially in Chrome on 1st June 2011.

### Specification

This technology is still under development since the initial release in 2011, continually adding new features and keeps up-to-date<sup>2</sup>. Google created the technology itself as an open-source project. However, to give the project some direction, two organizations were founded to guide WebRTC standards.

- **Internet Engineering Task Force (IETF)** [2] concentrates on the specification of the network communication protocols. For example, how two or more peers can communicate. The protocols define a set of rules that determine how the system functions to exchange data. One of these protocols is JavaScript Session Establishment Protocol (JSEP).
- **World Wide Web Consortium (W3C)** [6] deals with API specification in JavaScript to access the WebRTC protocols. This specification ensures compatibility across

---

<sup>2</sup><https://webrtc.github.io/webrtc-org/release-notes/>

browsers. The browser developers should follow these rules. In the end, the APIs make the job of web developers easier. The developers need to do nothing more than to call the universal API methods in any browser.

In September 2020, the World Wide Web Consortium released WebRTC 1.0 specification, which is still in Candidate Recommendation status. There are another 2 phases on W3C Recommendation Track. The document must be published as Proposed Recommendation and finally as W3C recommendation. It means that W3C recommends deployment of its Recommendations as standards for the Web [4].

## Usage

Figure<sup>3</sup> 2.1 shows the broad-range reach of this technology. All major browsers (Google Chrome, Mozilla Firefox, Apple Safari, and Microsoft Edge) implement WebRTC nowadays. If we count mobile phone users, the potential reach is 93,46% of all browser users and this number is, apparently, still growing.

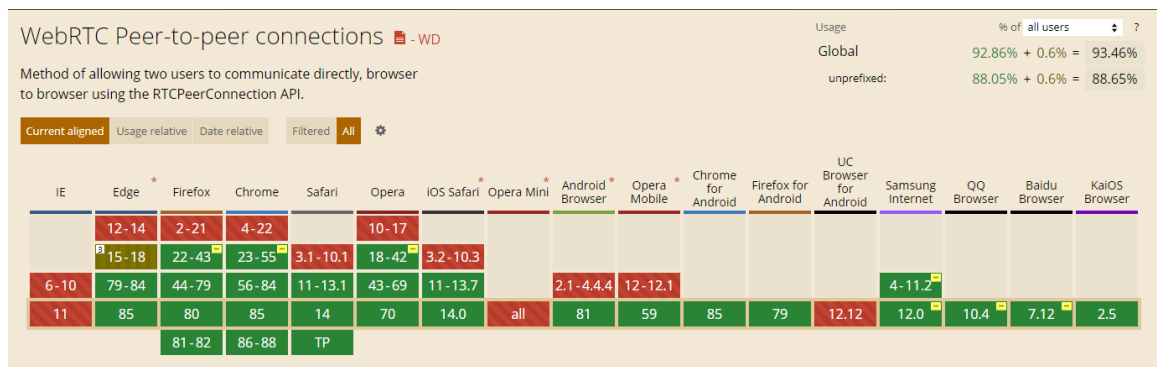


Figure 2.1: RTCPeerConnectionAPI and its compatibility across browsers.

## APIs and Interfaces

As mentioned above, the WebRTC technology is a set of APIs. The interfaces are interconnected. The three main tasks of WebRTC based on Google [25] are:

1. Access to local media devices (microphone, web camera) and processing of local streams in the MediaStream interface.
2. Connection of two WebRTC endpoints across the internet, keeping the connection alive, and stream processing in real-time.
3. Synchronized transfer of supplementary data with the stream over the same data channel.

A **MediaStream** is an abstract data type responsible for managing both local and remote streams. It provides stream related methods such as recording, viewing/playing, or data transfer to a remote peer. Each stream consists of several tracks. The stream can be empty (initialization). The track can be either video or audio. When a website

<sup>3</sup>Screenshot was taken from <https://caniuse.com/?search=webrtc> [2020-09-29]

requests access to a hardware device, such as a microphone, or webcam, the user must grant permission. The hardware devices are accessed by the `getUserMedia()` method from the `MediaDevices` interface [10] [14].

The **PeerConnection** interface is designed for peer-to-peer (P2P) communication between local and remote peers. An important role of this interface is signaling. The necessary data for the establishment of connection and session are interchanged. It includes used signal processing, codec handling, the establishment of data packets route over the network, security, and bandwidth management. The data flows directly from browser to browser has many advantages [10] [16].

The **DataChannel** interface enables data exchange in a synchronized manner. A media stream and data stream synchronization is achieved by transferring both in the same `RTCPeerConnection`. Every `DataChannel` has to be bound to `RTCPeerConnection`. The communication is bidirectional and P2P. One `RTCPeerConnection` can contain up to 65534 data channels. The exact number is given by the browser implementation. A `DataChannel` may be created by calling the `RTCPeerConnection`'s `createDataChannel()` method [15].

## 2.2 Telephony

This section presents the structure of a voice-over-IP (VoIP) call. The subsection Protocols provides details about the call control and signaling protocol - Session Initialization Protocol (SIP), protocol transferring metadata about the call - Session Description Protocol (SDP), and protocol transferring voice to the other site - Realtime Protocol (RTP). In reality, the establishment of a peer-to-peer connection is not an easy task. Many internet users are behind a device doing network address translation (NAT) or are using firewalls. The subsection NAT traversal describes several techniques about how to bypass these obstacles.

### Protocols

Even though WebRTC does not have a strictly defined signaling protocol, there are common approaches for some use cases. In the VoIP telephony, the most used signaling protocol is Session Initiation Protocol (SIP), which is *de facto* a standard.

### SIP

Session Initiation Protocol (SIP) [23] [21] is an application-layer signaling protocol. It was designed to create, modify, and terminate sessions with one or more participants. A session may be a voice over IP telephone call, multimedia distribution, or multimedia conference. It contains information about participants and their streams. Since its initial release in 1999 and second version in 2002 (still in a standard proposal), SIP has achieved significant popularity. The SIP message syntax is similar to HTTP/1.1 messages. Both SIP and HTTP are request/response-oriented client-server protocols and human-readable. The request messages are used to invoke some method on a server. The methods are the following:

- REGISTER to register a user of telephony on the server
- INVITE, ACK, and CANCEL - to set up sessions
- BYE to terminate sessions

Response class	Response Description	Action Taken or To Be Taken	Examples
1xx	Provisional	Request received, continuing to process the request	100 Trying, 180 Ringing, 182 Queued...
2xx	Success	The action was successfully received, understood, and accepted	200 OK
3xx	Redirection	Further action needs to be taken in order to complete the request	301 Moved Permanently, 302 Moved Temporarily, 305 Use Proxy...
4xx	Client Error	The request contains bad syntax or cannot be fulfilled at this server	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found...
5xx	Server Error	The server failed to fulfill an apparently valid request	500 Server Internal Error, 502 Bad Gateway, 503 Service Unavailable...
6xx	Global Failure	The request cannot be fulfilled at any server	600 Busy Everywhere, 603 Decline, 604 Does Not Exist Anywhere...

Table 2.1: An overview of SIP response codes, response classes and examples. Source: [23]

- OPTIONS to query servers about their capabilities
- MESSAGE for chat sessions
- REFER to transfer a call to someone else
- SUBSCRIBE and NOTIFY for event management
- PUBLISH to publish event state
- UPDATE to update session parameters
- INFO to transfer information during sessions
- PRACK to acknowledge provisional requests

These methods are not used equally often. Some of them are used multiple times per session, some of them only once, some of them occasionally. The methods for setting up and terminating sessions (INVITE, ACK, CANCEL, BYE) are always used. There is an example of a SIP request message with the method INVITE in the table 2.2. There is also shown how the SIP message is divided into three main parts - Request Line, Headers, and Message Body. The Request Line appears only in a request message. There is always an empty line in between the last header and the message body. The message body is optional. It can be empty, but it often contains the **Session Description Protocol(SDP)** part, which is also seen in the table.

The SIP response message is sent back from the server to the client. It contains a result of the method. The difference to the IP request message is that there is a Status-Line instead of the Request-Line. Otherwise, it uses the same schema. The method result status is encoded into a 3-digit response code, which is consistent with the HTTP/1.1 codes. Only codes relevant to SIP are used. There are 6xx response codes as an extension. The codes are listed in the table 2.1.

## SDP

Session description protocol (SDP) [23] [5] is used for media description. It describes the media type transferred in the multimedia session (through RTP protocol). The parties that are willing to communicate need to negotiate details about the call. Each party needs to know the audio or video encoding method (codec), source/destination IP addresses, and ports. The sides have to arrange these parameters. All these data are transferred in an SDP

Request Line	INVITE sip:phonexia-client@192.168.2.100:57952;ob SIP/2.0
Headers	Via: SIP/2.0/UDP 192.168.2.103:5060;branch=z9hG4bK1b0d0bd9;rport Max-Forwards: 70 From: sip:122@192.168.2.103;tag=as6d339018 To: sip:phonexia-client@192.168.2.103;tag=83a69dca02e54a00b5975ee962261d97 Contact: sip:122@192.168.2.103:5060 Call-ID: 804287b274c74ca3b546ca7723f2811f CSeq: 102 INVITE User-Agent: Asterisk PBX 13.18.3~dfsg-1ubuntu4 Session-Expires: 1800;refresher=uac Min-SE: 90 Allow: INVITE ACK CANCEL OPTIONS BYE REFER SUBSCRIBE NOTIFY INFO PUBLISH MESSAGE Supported: replaces timer Content-Type: application/sdp Content-Length: 248
Empty Line	
Message Body	v=0 o=root 70253909 70253910 IN IP4 192.168.2.103 s=Asterisk PBX 13.18.3~dfsg-1ubuntu4 c=IN IP4 127.0.0.1 t=0 0 m=audio 10001 RTP/AVP 0 101 a=rtpmap:0 PCMU/8000 a=rtpmap:101 telephone-event/8000 a=fmtp:101 0-16 a=maxptime:150 a=sendrecv

Table 2.2: An example of SIP request message



message. The SDP protocol does not enforce any specific transport protocol, but the authors of the RFC suggest to use one of these: Session Announcement Protocol (SAP)(RFC 2974), Session Initiation Protocol (SIP) (RFC 3261), Real-Time Streaming Protocol (RTSP)(RFC 2326), electronic mail using the Multipurpose Internet Mail Extensions (MIME) extensions, and the Hypertext Transport Protocol (HTTP).

## NAT Traversal

**Network address translation (NAT)** Network address translation, defined in RFC 3022 [24], is a widely used method to map IP address from one to another together with a port number on the transport layer. An example of how NAT works is shown in Figure 2.2. The

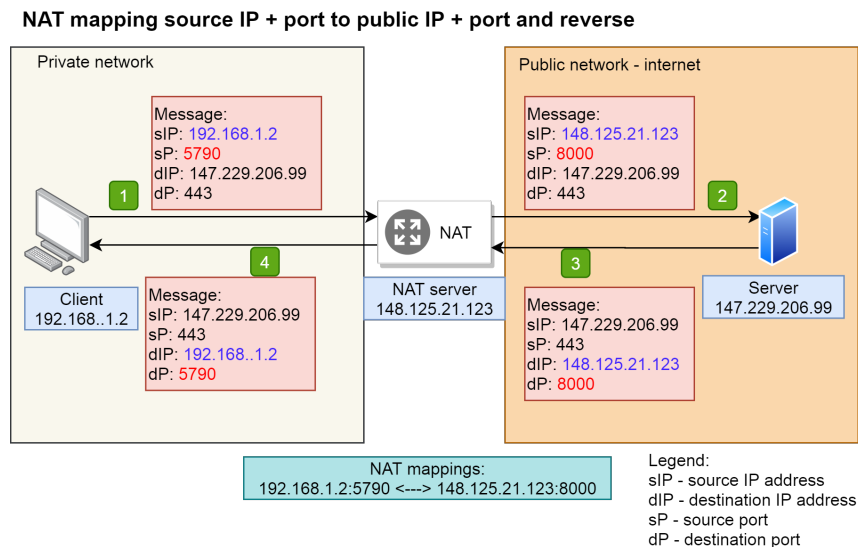


Figure 2.2: Simple example of NAT mapping

limited number of IPv4 addresses is the main reason for using NAT. Because the number of Internet users is continuously increasing, multiple users share the same public address. Another reason to use NAT is security. It is a good idea to separate public and private networks. There are four basic implementations of NAT<sup>4</sup> [22]:

- full cone - all packets from the same internal client's source IP address and port are mapped to the same external IP address and port. In other words, if a client sends a message with a source address and port to a device on the internet, the address and port are mapped to a NAT external IP address and port. Any device can pass a message to the client using the client's NAT IP address and port. The NAT will forward this message to the specific client in a local network. Figure 2.3 demonstrates that if server 2 knows the combination of NAT IP and port, the server can send data to the client, although it has never communicated with the client before.
- address restricted cone - in this implementation, NAT passes only packets from IP addresses that received some data from the client before. For example, in Figure 2.4 Server 2 did not receive any data before. Therefore the communication is blocked by

<sup>4</sup>Figures 2.3 - 2.6 are taken from [https://en.wikipedia.org/wiki/Network\\_address\\_translation](https://en.wikipedia.org/wiki/Network_address_translation)

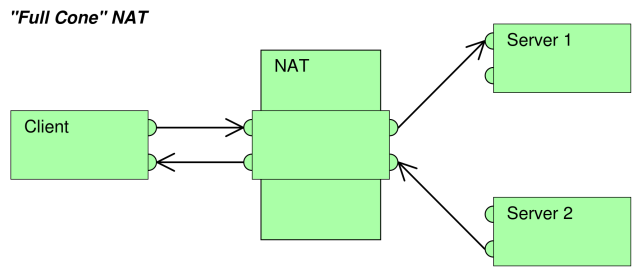


Figure 2.3: Full Cone NAT

NAT. But Server 1 has been contacted. Therefore Server 1 can send data to the client from any source port.

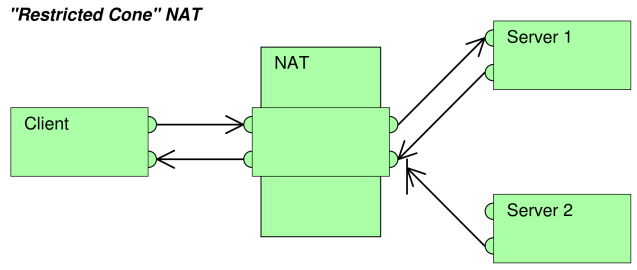


Figure 2.4: Restricted Cone NAT

- port restricted cone - here, the same rules as for the address restricted cone are applied, but a port number is checked too. NAT passes data only from the destination IP address and port that received some data before. In Figure 2.5 is illustrated that neither message from Server 1 with a different port, nor Server 2 with a different address can traverse through NAT.

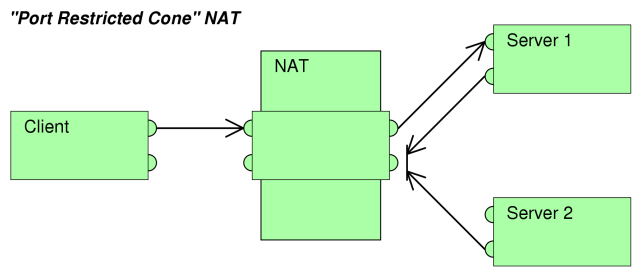


Figure 2.5: Port Restricted Cone NAT

- symmetric - This implementation is the most restrictive. In Figure 2.6, the communication between two peers is possible only if Server 1 responds on the same IP address

and port, which it received as a source address and port. Both IP addresses and ports must match the same conditions as by the port restricted cone. The difference is that symmetric NAT does not guarantee to assign any specific IP and port. A unique mapping is used for every new connection - the combination of source and destination IP addresses and ports.

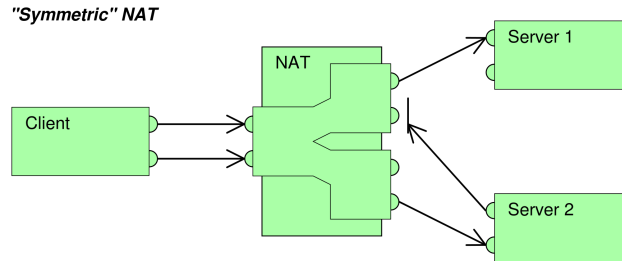


Figure 2.6: Symmetric NAT

For all of the mentioned implementations apply that the client's outgoing messages open NAT and are always first. Only then the destination side can start the communication. Besides this recommendation, there is no standard implementation. Vendors can implement variations of the above NAT approaches. NAT brings significant difficulties to an end to end communication. Therefore WebRTC engineers implement a solution that combines various NAT traversal techniques.

**Session Traversal Utilities for NAT (STUN)** [19] is a protocol serving as a tool for other protocols handling a client's visibility outside the NAT. An application supporting this protocol can discover an external IP address and port allocated to it by a NAT. It is necessary to know the address of a STUN server. Additionally, this protocol can check connectivity between the client and the outside world, and as a keep-alive protocol, it can maintain NAT bindings. The protocol starts with the client's message to the STUN server. This message passes through one or more NATs. Every NAT assigns a tuple of the external IP address and the port. The last NAT exposes a public address and port. The STUN server puts the public address and port into a new message and sends it back to the client. If the client gets the STUN server's message successfully, it knows that the client can communicate with the external world. The types of NATs where a client can reach the STUN server are the followings:

- full cone
- address restricted cone
- port restricted cone

**Explanation of some communication setups:**

Two hosts, both behind port restricted cone NATs:

1. In the first step, each side contacts a STUN server it knows (has STUN server IP address and port).
2. STUN server informs them what their public IP address and ports (behind NAT) are.

Caller \ Callee	FC	AR	PR	SM
Full Cone (FC)	pass	pass	pass	pass
Address Restricted (AR)	pass	pass	pass	pass
Port Restricted (PR)	pass	pass	pass	fail
Symmetric (SM)	pass	pass	fail	fail

Table 2.3: NAT Traversal possibilities using STUN. Preverit z tohoto linku <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5620993>

- Both sides send each other their public IP addresses and ports.
- The sides use the public source address and port obtained from the STUN server and the destination public address and port obtained from the opposite side to communicate.
- The first packets might be discarded by the opposite side's NAT, but the opening of a specific IP address and port in local NAT is achieved.
- After some period, both sides can communicate peer to peer.

The first behind symmetric NAT (host A) communicate with a host behind address restricted cone NAT (host B): We need to understand the **symmetric RTP** technique[27] for an explanation of this communication setup. A device supporting Symmetric RTP uses the same IP address and port for both ingress (listening) and egress (transmitting) communication. Symmetric RTP can be used for transmission through a pair of symmetric NAT and full cone or a pair of symmetric NAT and address restricted NATs on the path. The port can be adjusted for full cone NAT or address restricted NAT by knowing the other side's public IP address. Let us assume that both hosts A and B support symmetric RTP. The communication steps are the following:

- Steps 1-3 are the same as in the example above.
- The answer from the STUN server to host A is useless because of different mapping, yet the host A will receive a valid mapping from host B.
- Host B opens a hole in NAT for the host's A IP address. Host A establishes a one-way communication channel with host B.
- Host B can listen to the data of host A, while host's A NAT drops backward communication. Host B also inspects a source port number and tries to create a communication channel to this newly discovered port. Fortunately, this is possible because host A accepts communication at this port.
- Host B changes the destination port of its RTP packets. Hence the bidirectional communication starts.

In the end, let us explain why symmetric NAT (host A) and port restricted NAT (host B) on the path brake the communication:

- Steps 1-3 are the same as in the setup with the port restricted cone NAT - port restricted cone NAT communication.

2. The answer from the STUN server to host A is useless because of different mapping. Host A will get public IP and port for B.
3. The communication will fail because symmetric NAT (host A) assigns a different public source address than expected by port restricted NAT (host B).

It is impossible to solve all the communication setups via mapping IP addresses and ports using STUN servers. On the other hand, there is still a possibility to connect two peers, however not directly. The idea is to relay messages through an intermediate node.

**Traversal Using Relays around NAT (TURN)** [12] is a protocol extending the STUN protocol with the possibility to use a relay server halfway. TURN is critical when there is no possibility to establish direct P2P communication. In addition to that, TURN allows a client to connect with more than one endpoint using a single relay server. This is an advantage for users with a slow internet connection when doing group calls. Instead of sending the same data to multiple P2P connections, the voice is sent to the relay server only once. On the other side, this behavior brings a high load to the TURN server. It needs a high-bandwidth connection to the internet and good performance. TURN server is typically used when there is no other connection way possible.

**Interactive Connectivity Establishment (ICE)** [8] is a technique to find routes between two parties to communicate peer-to-peer over the network. The more straightforward, the better. ICE is defined in standalone RFC 8445, although it is linked with presented STUN and TURN protocols. ICE makes the decision which is used. It is worth noting that ICE was designed for NAT for the media only. It is supposed that the signaling path is already established via another mechanism. The ICE agents must be able to share proposed candidates. The ICE candidate is a self-generated entry generated by an agent, which contains a combination of an IP address and port. Such a combination is called a transport address. The more candidates an agent generates, the higher is the probability that at least one is accessible from the outside. These candidates are ordered by priority and then sent to a remote peer. There are several kinds of candidates:

- *host candidates* - A transport address is obtained directly from a local interface, for example, Ethernet, Wifi, or VPN. Unless there are other restrictions from the underlying operating system, the port number can be chosen freely.
- *server-reflexive candidates* - These candidates are collected after sending a request and receiving a response from STUN servers and thus having a public IP address and port.
- *relayed candidates* - Candidates acquired from TURN servers. Used in the case when previously gathered candidates have failed. In this case, the connection is established with a relay server.

The ICE also operates during a call in the background. If some connection fails, or a more reliable path between two peers is found, a new route is selected.

## 2.3 Voicebot systems

A bot (robot) is a computer program, which can handle requests autonomously. A voicebot can understand and process requests provided by voice. The output is typically in a voice form too. It is not an easy task to make computers understand a human language. It was

a big deal of telling computers what they should do and how to get a response back when they were invited. Computer designers came with a very simple subset of natural language (commands) to communicate with computers. The most reliable way to transfer these commands to a computer was to be physical by keyboard to minimize misunderstanding. A keyboard, a timeless invention, is still a very efficient way to control the computer because commands are interpreted precisely according to the programmer's intention. People had to adapt to this new technology. However, the most efficient way of communication for people is talking. It is even more natural for people than writing because kids usually start talking several years earlier than writing. That is why the voice interface is believed to be the future of human-computer interaction (HCI). Every voicebot system typically contains at least three main components: automatic speech recognition (ASR), natural language processing (NLP), and speech synthesis, also called text-to-speech (TTS).

### **Automatic speech recognition (ASR)**

Automatic speech recognition transcribes speech to text. A typical approach uses machine learning (ML) nowadays. The ASR consists of an acoustic model, language model, and decoder evaluating some transcription hypotheses. The ASR has always been a challenging task for multiple reasons:

- There are many different words, which sound similar, but have different meanings. E.g.: ware - wear - where, cite - sight - site, rain - reign - rein...
- The vocabulary might be highly dependent on the domain (a „regular“ conversation uses different dictionary than a conversation between doctors in hospital)
- There is also a lot of variety in speech among people and listening devices (microphone quality, dialect, pitch, gender, emotions, other speakers, background noise...)

Modern ASR systems can boost, or in other words increase the probability of some words, based on the context. Modifying a language model is also denoted as Language Model Customization (LMC).

### **Natural language processing (NLP), natural language understanding (NLU) and command execution**

After the ASR system outputs a plain text, it is crucial to understand the meaning of words and the whole sentence. Based on the content, the right intent can be recognized. The intent is typically to retrieve data from a database or to execute a service. Some of the NLP tasks are following:

- Named entity recognition (NER): determining the parts of a text that can be identified and categorized into preset groups.
- Document classification: classify document based on the used vocabulary e.g. research paper vs. newspaper article
- Spelling and grammar checking and correction: this can be done based on language rules
- Summary of a document: To capture the most important things from the document.

- Question answering: given a question and a document. The NLP's task is to find an answer either in that text (closed domain), or from other sources (open domain).
- Sentiment analysis: given a document, the task is to find positive, neutral, or negative sentiment in that document.

Even if it seems that these tasks have no applications in voicebots, we need to take into account that we work with a text all the time. The document classification, for example, doesn't mean that a spoken utterance is classified (one or two sentences). It can be used as a step in the voicebot's pipeline. If the query is: „Show me all master thesis dealing with WebRTC this year“. The NLP layer, after extracting the intent, will analyze all documents and return the names of the thesis.

### **Text-to-speech (TTS)**

When people ask a question to other people, they expect a voice answer. While speech production is standard for humans, it is not so apparent for computers. The process of transforming text into speech is called a speech synthesis. Although it is probably the easiest task in the voicebot system process, to achieve human-like natural-sounding speech is complicated. A human ear is well trained by listening to hundreds of conversations every day, and so even a small variation in the TTS process can be understandable but disturbing. In ordinary human discussions, people change voice characteristics like tone, speed, and emphasis based on the content or their feelings. The TTS system should ideally do the same to make the voice more natural.

## Chapter 3

# System design

There are many technologies and programming languages in the IT industry, and their number increases all the time. Each software architect must build on top of some time-proven technologies to design a well-working system. In this chapter, a design of a dialog system working with voice and visual information is presented. The design starts with Phonexia Voicebot Suite, which is extended with additional components that enable the use of the dialog system on the web and the connection of voice and visual information.

### 3.1 General concept

Figure 3.1 presents the schema of the proposed system. There are voicebot related components and some additional components needed for a deployment. The additional components include a Private branch exchange (PBX) - a server enabling the voicebot to connect to a public telephone network (PSTN) or connect a web telephony client through WebRTC. Then there is a web server that hosts a website, HTTP relay server for transferring text messages between client and voicebot, and a STUN (or TURN) server to pass the telephony traffic through NATs.

### 3.2 Custom voicebot system

A commerce voicebot solution used in this thesis uses Phonexia Voicebot Suite, a set of components for building voicebots. The Phonexia voicebot solution can be installed as a standalone application, docker image, or a VirtualBox image. In the case of deployment to a Unix-based operation system, the Ubuntu Linux distribution is preferred. It is very important to have at least 4GB of memory because just SpeechEngine takes approximately 2GB. The amount of needed memory also depends on the language, average phrase length, and the number of served users in parallel. If the system does not have enough memory, the service runs properly for about 5 minutes, then it crashes, and the operating system restarts the service again. This is repeating in a loop. It is tough to debug problems like this because most of the time, the memory consumption does not exceed 50%, but then the process is killed due to a memory peek. In the following sections, we describe components and their responsibility.



## SIP connector

SIP connector is an endpoint for the telephony system. It translates between SIP messages and HTTP REST calls and vice versa. SIP connector's configuration includes IP address and port of the telephony SIP server (here PBX) and IP address and port of a REST interface offered by the SIP connector to a Voicebot Backend. The Voicebot Backend uses the REST interface to initialize or terminate calls or react to call states. The Phonexia Voicebot Suite implementation of the SIP connector does not support the SIP MESSAGE message. Thus not text messages can not be transferred between client and voicebot over SIP. Another server (HTTP Relay for text messages) must transfer text messages between the client and voicebot.

## Phonexia Speech Engine (SPE)

Phonexia Speech Engine performs speech transcription and handles speech synthesis via an external service. The engine offers multiple pre-trained languages for speech transcription. The required one can be specified in a configuration file. Also, the maximum number of processing instances should be specified there. This controls the maximal number of simultaneous phone calls. Next to speech transcription, the SpeechEngine also offers other technologies:

- age estimation
- denoiser
- diarization (identification which speaker speaks when)
- gender identification
- keyword spotting
- language identification
- speaker identification
- voice activity detection

These features can be used out-of-the-box, just with a bit of configuration. The speech synthesis connector is capable of working with external TTS providers Acapela, Google, and SpeechTech. In our demo, we use SpeechTech, where the speech is synthesized through SpeechTech's online service. The SPE supports RTP streams natively. During the writing of this thesis, several SpeechEngine versions were released. The accuracy of the speech transcription improved, and new features were added. In the newest release, some RTP streams issues were fixed, too, such as proper RTP packet length to minimize latency, new packets for hole punching of NATs, a unique ID for each stream. Since SpeechEngine is a commercial product, it is likely that updates will continue, and customer support will be available in the long term. Updating SpeechEngine is a straightforward process. Step-by-step instructions and changelog are included in the docs folder of the SpeechEngine package. SpeechEngine can be updated separately without the need to reinstall all the voicebot components.

## Dialog/NLP component

The dialog layer gives meaning to the pronounced and transcribed text using natural language processing (NLP). It controls a human-voicebot conversation flow. Both the input and the output of this layer are texts retrieved/provided through a REST call. The dialog layer can be entirely an external service, e.g., IBM Watson, or a custom solution. An IBM Watson service connector is provided. One must add credentials into a configuration file.

## Voicebot Backend

Voicebot Backend is the central part that interconnects all the individual components into a suite. It initializes speech technologies for each new call. It sends a text from Speech Engine (STT) to the dialog platform and back (to TTS). Next, it controls the dialog flow (stops TTS when the user speaks, terminates the call at the end of a dialog, etc.). We set IP addresses and ports of Voicebot Backend and individual components in a config file. We can specify language and TTS providers too. Voicebot Backend is written in JavaScript as an open-source and can be modified by anyone.

## 3.3 Voicebot deployment

The voicebot integration relies on real-time communication. The implementation of the SIP signalization protocol into a web browser expects a duplex communication. A WebSocket protocol can reliably encapsulate and transfer SIP messages over the network [3]. One of the critical features of a voicebot is the ability to scale in parallel calls. All components - PBX, SIP connector, SpeechEngine, speech synthesis, and the dialog layer are ready for scaling.

### Private branch exchange (PBX)

Private branch exchange (PBX) is a gateway for incoming and outgoing calls to a local telephony network. It has a similar role as a router in computer networks. PBX keeps records about users in a local network, and it can mediate calls to a final destination. The key features of PBX based on SIP are handling SIP signaling messages (in our case, transferred over the WebSocket protocol) and audio streams (in our case transferred over the SRTP protocol). Web-based clients (softphones) can be connected to a PBX using WebRTC. In some PBXs, e.g., in Asterisk, WebRTC clients are supported natively. The support includes a WebSocket interface to the outside world that transfers signalization. The signalization is not standardized. We use the SIP protocol. Then it supports real-time communication between clients to transfer audio payload (RTP/SRTP streams) and NAT traversal protocols. Furthermore, PBX enables easy interconnection of calls to a public telephone network (PSTN) via a SIP Trunk. The SIP Trunk links the public telecommunication network operated by telephony providers. SIP trunk usage might bring an engaging scenario where the same voicebot could be accessed from both the Internet and public telephone network.

### HTTP Web Server

A web server is a computer that hosts websites. It communicates with clients through Secure Hypertext Transfer Protocol (HTTPS) and delivers content requested by a client. In our case, we expect that every user, willing to use voicebot, connects to a website and loads a

web page with an embedded JavaScript SIP client. The user is a web-based SIP client from the telephony point of view. These clients connect to the SIP server (PBX), which brings essential call control options. Besides that, it is a regular website distributing HTML, CSS, and JavaScript code. The web server also hosts a complete web-based application with our demo dialog.

### **HTTP Relay server**

This server acts as a mediator that transfers non-telephony-related control messages between the client and the voicebot backend. It ensures synchronization of the voicebot and the graphical interface on the client. It must be a publicly accessible server. We propose that each call has a unique ID used as an identifier used to pair both sides.

### **STUN (+TURN) Server**

A STUN server helps with NAT traversing. Even though there are some free public STUN servers, running a private STUN server increases reliability. As it was explained earlier in subsection 2.2, a STUN server addresses some connectivity cases, but only a configured TURN server guarantees the connectivity. Running the TURN server is not mandatory, but it is highly recommended.

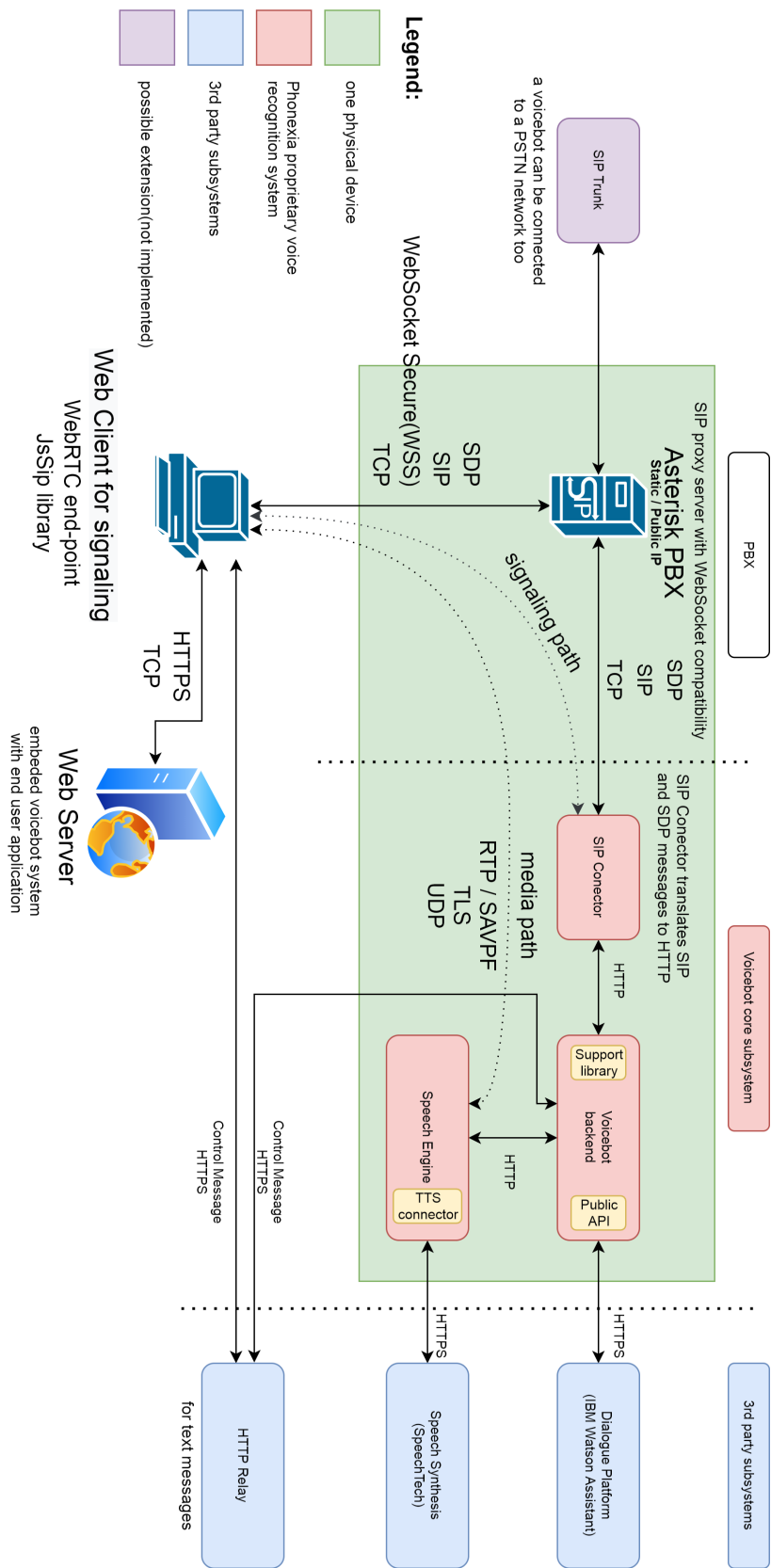


Figure 3.1: System design with components and communication protocols

# Chapter 4

## Implementation

This chapter describes how the solution is implemented. Both the final solution and dead-end attempts are presented.

### 4.1 Initial approach

At first, we installed Phonexia voicebot solution on a server containing a fresh installation of Ubuntu 18.04. To make the application run, one needs to have a valid license file for Phonexia SpeechEngine stored in the root directory of SpeechEngine. The license is periodically checked by a license server. Afterward, we installed an open-source PBX Asterisk in version 13.18.3 dfsg-1ubuntu4. Asterisk was configured according to the description presented in the following sections. Before doing anything with a WebRTC client, we tried to make a call from a softphone installed on the local computer to the voicebot backend through Asterisk. Two popular VoIP softphone clients were tested. The clients are: Zoiper<sup>1</sup> and MicroSIP<sup>2</sup>. Both are free for non-commercial use. The Zoiper softphone was tried at first, but we encountered troubles in the testing phase. In Wireshark, we found out that the RTP stream goes in one way only. It was possible to hear the voicebot voice, but the other side was silent. Later we found that Zoiper sends a small packet with payload type 95 in the RTP stream to open a NAT binding if there is a NAT between Zoiper and the PBX. This mechanism is called „Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows“ from RFC 6263[11]. The SpeechEngine could not handle this packet and closed the communication. Support of this RFC was implemented to SpeechEngine later. Incoming packets with payload type are ignored, and SpeechEngine reports warnings in its log file only. The MicroSIP client worked fine, except we had an issue with an acoustic echo between the speaker and microphone (the voicebot reacted on its own voice). So far, we tested everything on a single machine with the VoIP client on the local computer hosted in VirtualBox.

### 4.2 JavaScript’s SIP client

We decided to use the SIP protocol for signaling between the voicebot client and PBX. The next step was to find a JavaScript SIP library, which implements the SIP protocol. There are two popular SIP libraries, which are widely used and regularly updated. The

---

<sup>1</sup><https://www.zoiper.com/>

<sup>2</sup><https://www.micosip.org/>

first one is SIP.js library<sup>3</sup>. Unfortunately, we could not find any working example either in documentation or on the Internet. Therefore we selected JsSip.js<sup>4</sup>, where we found more examples. The JsSip.js library is licensed under the MIT license with no restriction of commercial use. This is interesting for the Phonexia company and others who would like to build on this work. Also, the debugging of code that uses the JsSip library is convenient thanks to the detailed log output in the browser debug console showing all the SIP/SDP messages and their parsing. Next to desktop clients, it is also possible to debug and optimize the client for smartphones. Chrome enables you to see the smartphone's browser console on the desktop. Google Chrome integrates a plug & play solution<sup>5</sup>, which makes the debugging convenient. In general, documentation of both JavaScript SIP libraries is not very intuitive for beginners. It is updated often with some changes to APIs, but examples in the documentation are not always updated. It isn't easy to start using the newest version without good examples.

In Listing 4.1 we show a client code based on the JsSip.js library with the configuration part. Note that it is just a skeleton and not a complete frontend code. The variable `config` is in a separate file for easier modifications. An interesting part is the `callOptions` configuration specifying the `iceServers`. It is a part where STUN and TURN servers are defined. With this configuration, a call from a smartphone was established immediately, but a call establishment from a notebook took about 40 seconds initially (both smartphone and notebook were connected to the same WiFi router). A smartphone usually has the only network interface, but a notebook can have many (localhost, ethernet, wlan, virtual box network interfaces, and others). The delay occurred when the notebook gathered ICE candidates on every interface. Since there is no response from some interfaces, there is a timer, and the process was waiting for timeouts.

```
1 var config =
2 {
3   "sendMessageTo": "sip:sip-connector@127.0.0.1",
4   "socket": "wss://20.52.138.109:8089/ws",
5   "uri": "sip:webrtc@20.52.138.109",
6   "password": "webrtc-273",
7   "contact_uri": "sip:petr@127.0.0.1",
8   "pcConfig": {
9     'iceServers': [
10      {
11        'urls': [
12          'stun:stun.internetcalls.com',
13          'stun:stun1.l.google.com:19302'
14        ]
15      },
16      // here a TURN server might be added
17      { 'urls':
18        'turn:example.com',
19        'username': 'foo',
20        'credential': '1234'
```

<sup>3</sup><https://sipjs.com/>

<sup>4</sup><https://jssip.net/>

<sup>5</sup><https://developer.chrome.com/docs/devtools/remote-debugging/>

```

21         }
22     ]
23 },
24     "callNumber": "200"
25 }
26 const socket = new JsSIP.WebSocketInterface(config.socket);
27 const configuration = {
28     'sockets': [socket],
29     'uri': config.uri,
30     'password': config.password,
31     'contact_uri': config.contact_uri,
32 };
33 // Register callbacks to desired call events
34 var eventHandlers = {
35     'progress': function(data){ /* Your code here */ },
36     'failed': function(data){ /* Your code here */ },
37     'confirmed': function(data){ /* Your code here */ },
38     'ended': function(data){ /* Your code here */ }
39 };
40 const callOptions = {
41     'eventHandlers': eventHandlers,
42     'mediaConstraints': {audio: true, video: false},
43     'pcConfig': config.pcConfig
44 };

```

Listing 4.1: A configuration of JsSIP.js library

To limit the delay before the call is established, we tried two approaches. The first one stopped the gathering of ICE candidates after the first one arrived. The second approach limits the maximal waiting time for each candidate. A disadvantage of the second solution is that the process always waits some non-negligible time, so we implemented the first one. In Listing 4.2 is the second part of voicebot client.

```

1 // a remoteAudio is a container for the remote stream and it is played
  locally, in a browser
2 const remoteAudio = new window.Audio();
3 remoteAudio.autoplay = true;
4 let phone;
5 let session;
6 if(configuration.uri && configuration.password){
7     JsSIP.debug.enable('JsSIP:*'); // more detailed debug output
8     phone = new JsSIP.UA(configuration);
9     phone.on('registrationFailed', function(ev){
10         alert('Registering on SIP server failed with error: ' + ev.cause);
11         configuration.uri = null;
12         configuration.password = null;
13     });
14     phone.on('newRTCSession',function(ev){
15         const newSession = ev.session;
16         if(session){ // hangup any existing call

```

```

17     session.terminate();
18 }
19 session = newSession;
20 var completeSession = function(){
21     session = null;
22 };
23 // this part limits the delay of getting srflx candidates
24 session.on("icecandidate", function (event) {
25     if (event.candidate.type === "srflx" &&
26         event.candidate.relatedAddress !== null &&
27         event.candidate.relatedPort !== null) {
28         event.ready();
29     }
30 });
31 session.on('ended', completeSession);
32 session.on('failed', completeSession);
33 session.on('accepted',function(){
34     const remoteStream = session.connection.getRemoteStreams()[0];
35     remoteAudio.srcObject = remoteStream.clone();
36 });
37 });
38 phone.start();
39 }
40 // make a call when pressing a button, also jQuery library is used
41 $('#connectCall').on("click",function () {
42     const dest = config.callNumber;
43     phone.call(dest, callOptions);
44 });

```

Listing 4.2: Use of JsSIP.js library for connecting to Asterisk via secured WebSocket

### 4.3 ICE support in Asterisk

When we replaced the localhost's SIP client (softphone) with the browser WebRTC solution for the first time, the browser SIP client was able to connect to Asterisk but was not able to make any calls. It always failed with error:

Reason: SIP ;cause=488; text="Not Acceptable Here"

As it emerged from the debugging later, the ICE support in Asterisk was disabled by default. There are two possible SIP implementations in Asterisk - chan\_sip and pjsip. At first, we used chan\_sip, and thus the following variable needs to be set in the configuration file `/etc/asterisk/sip.conf`

```
icesupport = yes
```

Enabling the ICE support made the establishment of bi-directional calls possible. Replacing MicroSIP with the JavaScript library handling SIP (JsSip) solved a problem with the echo cancellation, too, because WebRTC implements echo cancellation, and it is enabled by default. In the first version of Phonexia Voicebot Suite, an older Asterisk SIP



implementation (driver) - chan\_sip was used. Voice communication between the browser and the voicebot worked as expected.

However, we wanted to use text messages transferred between the client and the voicebot backend through the SIP protocol (SIP MESSAGE) for synchronization of these two parts to simplify the implementation. Text communication using SIP messages did not work. The text communication worked in WebSocket (ws) only used as a transfer layer but did not work in WebSocket Secure (wss). Asterisk shows the following error:

```
[Feb 11 22:36:04] ERROR[5469] [C-00000000]:
chan_sip.c:4274 __sip_reliable_xmit:
Serious Network Trouble; __sip_xmit returns error for pkt data
```

This was unacceptable, and we solved this problem by using a newer Asterisk PJSIP implementation. The sip.conf configuration was rewritten to the newer pjsip.conf configuration. The second textual channel is very important to achieve synchronous of the voicebot backend and client and fusion of the voice and visual presentation in the browser. The SIP messages can be used even without call establishment.

If there is no need to use IPv6 address family, it is recommended to have this interface turned off. Otherwise, the ICE generates additional candidates, which leads to longer connection times.

## 4.4 Asterisk's configuration files

In this part, we describe the most important Asterisk configuration files. Asterisk configuration is located in /etc/Asterisk by default. These are essential commands to debug and maintain Asterisk:

```
sudo asterisk -r ;run Asterisk's command-line interface (CLI) from shell
pjsip set logger on ;to show sip messages in real-time
rtp set debug on ;to show the rtp traffic in real-time
core restart now ;to restart Asterisk's service
```

Listing 4.3: Essential commands to debug and maintain Asterisk

### WebSocket component

Asterisk has an in-built HTTP server that enables to use WebSockets as the transfer layer for the SIP protocol. The server must be configured in the /etc/asterisk/http.conf configuration file:

```
[general]
enabled=yes
; HTTPS support. In addition to enabled=yes, you need to
; explicitly enable tls, define the port to use, and have a certificate.
; We describe this in section Certificates.
tlsenable=yes ; enable tls - default no.
tlsbindaddr=0.0.0.0:8089
```

Listing 4.4: Configuration of Asterisk builtin HTTP server. Configuration file — /etc/asterisk/http.conf.

The WebSocket' port number can be chosen, but one needs to be careful to open the port in the firewall and/or to add port forwarding rules in the router. The IP address and port number must be accessible from the outside network. The port number must match the one used on the second line of the voicebot client code [4.1](#).

## Dialplan

Asterisk dial plan specifies how the calls are handled and routed to devices (in Asterisk terminology endpoints). Asterisk uses its own macro language for these plans. The code [4.5](#) redirects all the incoming calls from the WebRTC endpoint to the sip-connector (our gateway to voicebot). The dial plan enables routing of text messages too. Both directions — from the web client to voicebot and from voicebot to the web client need to be specified.

```
; only a client can initiate a call
[from_webrtc]
    exten => _X.,1,Dial(PJSIP/sip-connector)
; messages can be sent bi-directionally
[messages]
    exten => sip-connector,1,NoOp(MESSAGE ${MESSAGE(from)}, ${MESSAGE(to)})
    exten => sip-connector,n,MessageSend(pjsip:sip-connector,
                                        ${MESSAGE(from)})
[messages_from_voicebot]
    exten => webrtc,1,NoOp(MESSAGE ${MESSAGE(from)}, ${MESSAGE(to)})
    exten => webrtc,n,MessageSend(pjsip:webrtc,${MESSAGE(from)})
```

Listing 4.5: Asterisk dial plan. Configuration file `/etc/asterisk/extensions.conf`.

## Asterisk channel driver

Asterisk channel drivers enable your devices (endpoints) to communicate via some protocol (SIP, IAX, Skinny, etc). As mentioned already earlier, there are 2 SIP channel drivers in Asterisk — `chan_sip`, and `chan_pjsip`.

An older, `chan_sip` was the only available driver in Asterisk version 11 and lower. It was widely used because it is stable, time-tested, and supports all of the features needed for SIP-telephony. However, due to new feature requirements, it is hard to ensure back-compatibility, and therefore `chan_pjsip` was introduced. Starting in Asterisk version 12, one can choose between `chan_sip` and `chan_pjsip`. [\[1\]](#) Both channel drivers can work simultaneously, but it is highly recommended to enable only one, preferably the newer one. In order to disable the older one, the following line should be added to the configuration file `/etc/asterisk/modules.conf`:

```
noload => chan_sip.so
```

Listing 4.6: Disabling of the `chan_pjsip` Asterisk channel driver. Configuration of file `/etc/asterisk/modules.conf`

The SIP configuration file is named after the channel driver - `pjsip.conf`. This is the most important file since it contains settings of all endpoints and user authorization. NOTE: It is normal for multiple objects in `pjsip.conf` to have the same name as long as their types differ.

```

; endpoint for web clients (WebRtc)
[webrtc]
    type=endpoint
    transport=transport-wss
    aors=webrtc
    auth=webrtc
    context=from_webrtc
    disallow=all
    allow=ulaw

; WARNING: dtls_auto_generate_cert parameter is available in Asterisk v18+
    dtls_auto_generate_cert=yes
; in older versions, this needs to be replaced for manually created certs
    ; dtls_cert_file=/home/certs/cert.pem
    ; dtls_ca_file=/home/ca/cert_authority.crt

; WARNING: webrtc parameter is available in Asterisk v15+
    webrtc=yes
    message_context=messages

[webrtc]
    type=aor
    max_contacts=5
    remove_existing=yes

[webrtc]
    type=auth
    auth_type=userpass
    username=webrtc
    password=webrtc-273

; transport protocol between Asterisk and web client
[transport-wss]
    type=transport
    protocol=wss
    bind=0.0.0.0

; endpoint for voicebot (SIP connector)
[sip-connector]
    type = endpoint
    context = from_voicebot
    message_context=messages_from_voicebot
    disallow = all
    allow = ulaw
    auth = sip-connector
    outbound_auth = sip-connector
    aors = sip-connector

```

```
[sip-connector]
  type = aor
  max_contacts = 5

[sip-connector]
  type = auth
  username = sip-connector
  password = sip-connector-273
```

Listing 4.7: SIP setting in asterisk using the pjsip channel driver. Configuration file `/etc/asterisk/pjsip.conf`

## 4.5 Security

Several certificates are needed. One certificate is needed for the webserver, where the web client HTML page is hosted. In our case, the web hosting provider offers an automatic generation and installation of a free Let's Encrypt certificate. Let's Encrypt is a nonprofit Certificate Authority providing TLS certificates for free. The second certificate is needed for the build-in Asterisk's HTTP server to ensure secure WebSocket (WSS) for SIP. Since this server is maintained by us, we need to install the certificate manually. For testing purposes, we generated self-signed certificates and enabled them in the web browser. It is much easier for real deployment to use the internet domain than to use a private IP address. A third certificate is needed to secure the audio/RTP traffic.

To simplify the certificate generating process, we used a tool called *certbot*<sup>6</sup>. It generates certificates recognized on the Internet. It uses the Let's Encrypt services. The certbot verifies that the specific domain runs a service on port 80 (HTTP). We needed to open this port on the firewall and run a web service (a dummy installation of Apache2 was used). After that, the tool generated the certificate. The certificate files were copied to a directory accessible by the Asterisk user (Asterisk runs under its user to increase security), and paths were added to the particular configuration file `/etc/asterisk/http.conf`:

```
; path to the certificate file (*.pem) only.
tlscertfile=/home/palo/certifikat/fullchain.pem
; path to private key file (*.pem) only.
tlsprivatekey=/home/palo/certifikat/privkey.pem
```

Listing 4.8: Setting of certificates in Asterisk. Configuration file `/etc/asterisk/pjsip.conf`

Phonexia Voicebot Suite package contains Asterisk in version 13.18.3 dfsg-1ubuntu4. It is an old version from the year 2014, and many changes have been made since then. It required manually created DTLS certificates for the pjsip endpoint. In the newest version, Asterisk 18, an option for auto-generate DTLS certificates was introduced. The DTLS certificate is crucial for SRTP communication. Secure media is enforced by WebRTC, and SRTP is required. Fortunately, Asterisk takes care of it.

<sup>6</sup><https://certbot.eff.org/>

## 4.6 Phonexia voicebot suite configuration

There is Asterisk preinstalled and pre-configured in the Phonexia Voicebot Suit package already. It has an Asterisk user account for voicebot and other of a caller. The voicebot can be called through any three-digit number. We must add a new user account for the caller that uses WebRTC. It is essential to turn on the WebRTC support for this user to enable to use of the SIP signaling over the WebSocket protocol. The STUN and TURN servers need to be defined and configured to ensure that the solution works well with all types of NATs. The coTURN server was installed on another computer with a public IP address.

A website with JavaScript code using the SIP library was developed for testing purposes and deployed on a web server. This website acts as a SIP client with the ability to call any number through Asterisk. The SIP client configuration requires a SIP URI and a password for the Asterisk account, a WebSocket URI (Asterisk WebSocket address as an entry point to telephony), STUN server URI, and TURN server URI. The testing scenario is shown in Figure 4.1. All devices in Figure have different IP addresses. Any web hosting company can run the webserver. The client was a mobile phone connected to the Internet using mobile data (e.g., 3G), which led to an IP address behind NAT or firewall. The server running Phonexia Voicebot Suite (Virtualbox image) was located in a private home network with a WiFi router. The WebSocket access point (Asterisk) had to be visible from the Internet, so a rule NAT mapping was added. After all the configurations mentioned above, the web client could make a call to the voicebot with bi-directional audio.

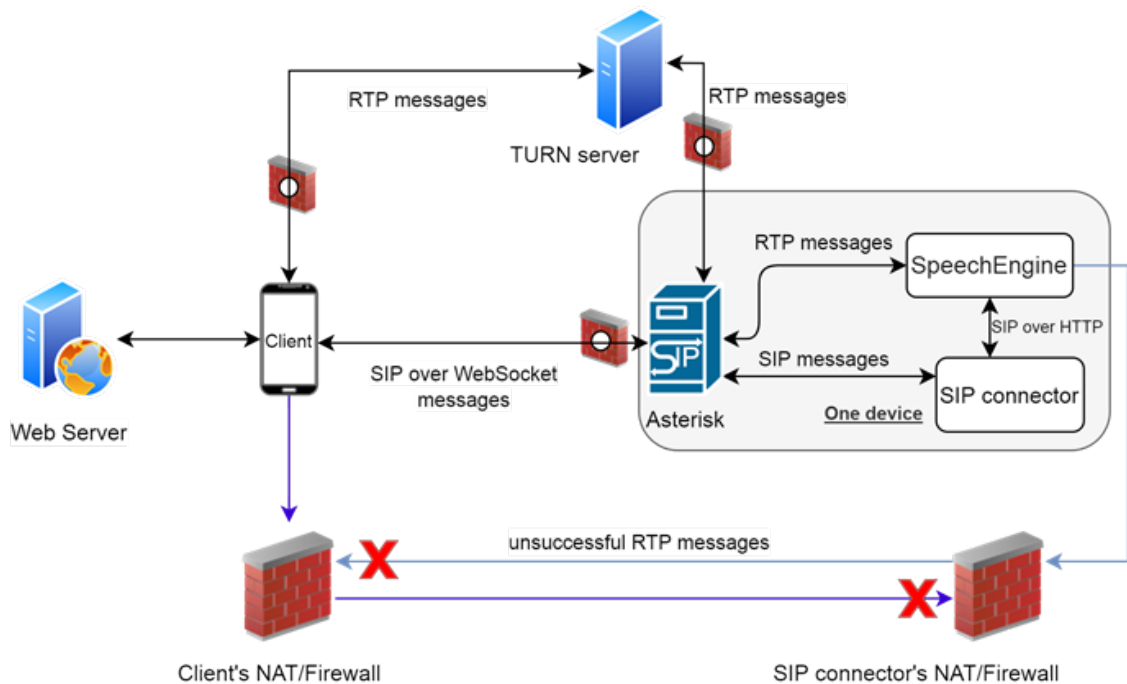


Figure 4.1: Worst-case testing scenario with each device in a different network address space and symmetric NATs.

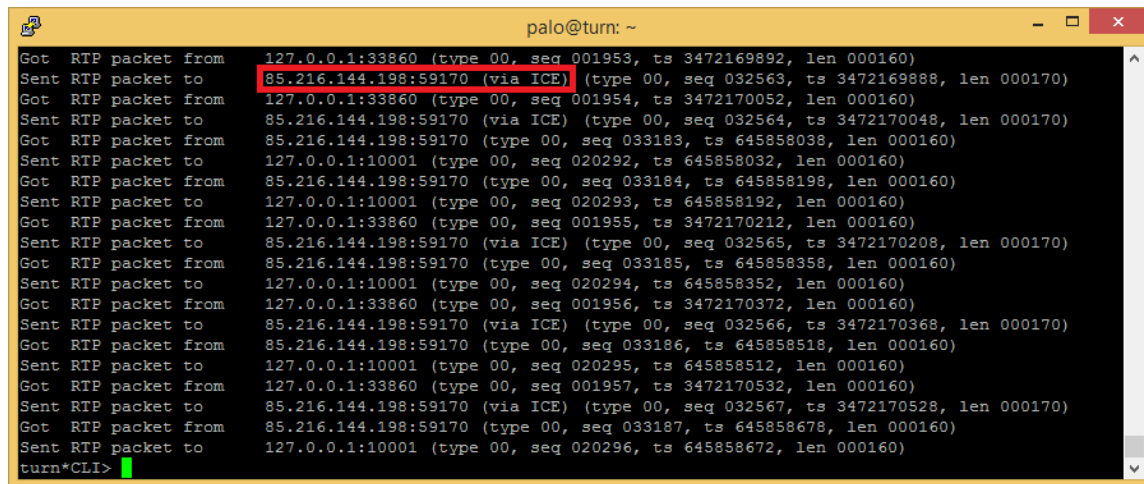
## 4.7 STUN and TURN server

As it was mentioned before, in some specific cases (e.g., caller and called behind NAT), a STUN or a TURN server is needed. It is possible to find and use some free STUN servers on the Internet, but it is often unreliable and unsafe. Public STUN servers are recommended to be used for experimenting only. The Asterisk's configuration enables to specify only one STUN server. We encountered behavior where one STUN server worked properly for 1-2 days, and afterward, it stopped working. It was not possible to establish the call because only some private addresses were exchanged. When we exchanged the server to another one from the list below and restarted Asterisk, it worked again. The missing possibility to specify multiple STUN servers reduces the reliability of the service. A list with free public STUN servers is here <sup>7</sup>. We can verify that the STUN server works by inspection of the SDP part in the SIP INVITE or SIP OK messages. It works if we can gather a candidate type „srflx“ having a public IP address, as you can see below. The first IP address on the line is public, and the second is private).

```
a=candidate:S14348a6d 1 UDP 1694498815 20.52.138.109 17660 \\  
                                typ srflx raddr 10.0.0.4 rport 17660
```

Listing 4.9: A working STUN signalling with public IP address

To verify if ICE is used, we can also connect to the Asterisk's CLI during a call and investigate the RTP stream. We can clearly see in Figure 4.2 that the candidate was gathered via ICE.



```
palo@turn: ~  
Got RTP packet from 127.0.0.1:33860 (type 00, seq 001953, ts 3472169892, len 000160)  
Sent RTP packet to 85.216.144.198:59170 (via ICE) (type 00, seq 032563, ts 3472169888, len 000170)  
Got RTP packet from 127.0.0.1:33860 (type 00, seq 001954, ts 3472170052, len 000160)  
Sent RTP packet to 85.216.144.198:59170 (via ICE) (type 00, seq 032564, ts 3472170048, len 000170)  
Got RTP packet from 85.216.144.198:59170 (type 00, seq 033183, ts 645858038, len 000160)  
Sent RTP packet to 127.0.0.1:10001 (type 00, seq 020292, ts 645858032, len 000160)  
Got RTP packet from 85.216.144.198:59170 (type 00, seq 033184, ts 645858198, len 000160)  
Sent RTP packet to 127.0.0.1:10001 (type 00, seq 020293, ts 645858192, len 000160)  
Got RTP packet from 127.0.0.1:33860 (type 00, seq 001955, ts 3472170212, len 000160)  
Sent RTP packet to 85.216.144.198:59170 (via ICE) (type 00, seq 032565, ts 3472170208, len 000170)  
Got RTP packet from 85.216.144.198:59170 (type 00, seq 033185, ts 645858358, len 000160)  
Sent RTP packet to 127.0.0.1:10001 (type 00, seq 020294, ts 645858352, len 000160)  
Got RTP packet from 127.0.0.1:33860 (type 00, seq 001956, ts 3472170372, len 000160)  
Sent RTP packet to 85.216.144.198:59170 (via ICE) (type 00, seq 032566, ts 3472170368, len 000170)  
Got RTP packet from 85.216.144.198:59170 (type 00, seq 033186, ts 645858518, len 000160)  
Sent RTP packet to 127.0.0.1:10001 (type 00, seq 020295, ts 645858512, len 000160)  
Got RTP packet from 127.0.0.1:33860 (type 00, seq 001957, ts 3472170532, len 000160)  
Sent RTP packet to 85.216.144.198:59170 (via ICE) (type 00, seq 032567, ts 3472170528, len 000170)  
Got RTP packet from 85.216.144.198:59170 (type 00, seq 033187, ts 645858678, len 000160)  
Sent RTP packet to 127.0.0.1:10001 (type 00, seq 020296, ts 645858672, len 000160)  
turn*CLI>
```

Figure 4.2: Asterisk's log with enabled `rtp set debug on` showing target IP address negotiated via ICE.

```
stunaddr=stun.internetcalls.com  
;stunaddr=stun.l.google.com:19302  
;stunaddr=stun1.l.google.com:19302  
;stunaddr=stun4.l.google.com:19305
```

Listing 4.10: STUN server setting. Configuration file `/etc/asterisk/rtp.conf`.

<sup>7</sup><https://www.voip-info.org/stun/>

The use of TURN servers is more complicated than STUN servers. A free TURN server is difficult to find because the whole telephony traffic goes through the server, and the server needs to be well prepared for the expected load. It is highly recommended to operate our own TURN server to ensure a stable connection. Fortunately, there is a free open source implementation of TURN and STUN servers named coTURN<sup>8</sup>. The installation is easy and fast. Our TURN server requires rules for forwarding specific ports in NAT and firewall and a public IP address. Text 4.11 shows a basic configuration of the TURN server. There are three input parameters:

\$1 username  
\$2 password  
\$3 server name

```
# automatically detect and set internal IP address
internalIp="$(ip a | grep -Eo 'inet (addr:)?([0-9]*\.){3}[0-9]*'
| grep -Eo '([0-9]*\.){3}[0-9]*' | grep -v '127.0.0.1')"
# automatically detect and set external IP address
externalIp="$(dig +short myip.opendns.com @resolver1.opendns.com)"

echo "
# TURN listener port for UDP and TCP (Default: 3478)
listening-port=3478

# TURN listener port for TLS (Default: 5349)
tls-listening-port=5349

listening-ip="$internalIp"
relay-ip="$internalIp"
external-ip="$externalIp"

# Note: If the default realm is not specified,
# then realm falls back to the host domain name.
# If the domain name string is empty, or set to '(None)',
# then it is initialized as an empty string. Realm as a parameter.
realm=$3

# Using fingerprints in the TURN messages.
fingerprint
server-name=$3

# Using long-term credential mechanism.
lt-cred-mech

# Username and password as a parameters
user=$1:$2

# A password to the client console.
```

<sup>8</sup><https://github.com/coturn/coturn>

```
cli-password=strongPassword123

# These ports need to be allowed in NAT/firewall.
min-port=12000
max-port=20000
" | tee /etc/turnserver.conf
```

Listing 4.11: TURN server configuration script `run_turnserver.sh`

Here is an example how to run the TURN server:

```
./run_turnserver.sh john P@sSvvorD!1 example.com
```

A TURN server guarantees that the audio traffic can pass all NATs. The coTURN simplifies the deployment for everybody. If the web client and SpeechEngine want to communicate directly, the communication is often blocked by NATs. The TURN server respects the rules of each side NAT and routes the audio traffic through itself.

## 4.8 Web presentation

The presentation layer uses using web technologies (HTML, CSS, and JavaScript). The design of the presentation webpage was created in a website builder software - Nicepage<sup>9</sup>. We chose this software because it generates responsive code for many types of screens. It is possible to customize graphical elements for every resolution or do not display some elements at all. Since this website is planed mainly for devices with microphones (notebooks, tablets, and smartphones), one of our goals was to prepare a modern-looking website working on both big screens (Figure 4.3) as well as on smartphones (Figure 4.4).

We decided to use the so-called one-page website. Such a simple website contains only one HTML page. It is a modern approach using anchors for navigation inside a single page. An anchor is a part of the hyperlink after the hashtag („#“) character. An example of such a link is here:

```
https://domain.com#section5
```

Even if the **section5** anchor is at the bottom of a page, the browser scrolls to this element immediately after the page is loaded.

The main advantage of this solution is that there is no cross-page redirection. Once a voice call is established, and dialog started, the web page can not be reloaded. Else the phone call is terminated and needs to be restarted. More elaboration would need to be done to integrate the voicebot to an existing multi-webpage website and crosslinks among them. One solution to this could be to have a voicebot in one frame and the presentation layer in another frame. Another possibility is to have the voicebot in one tab of the browser and web presentation in another. The tabs can communicate via a broadcast channel [18]. A drawback of this approach might be a lack of some features in browsers like Safari.

---

<sup>9</sup><https://nicepage.com/>



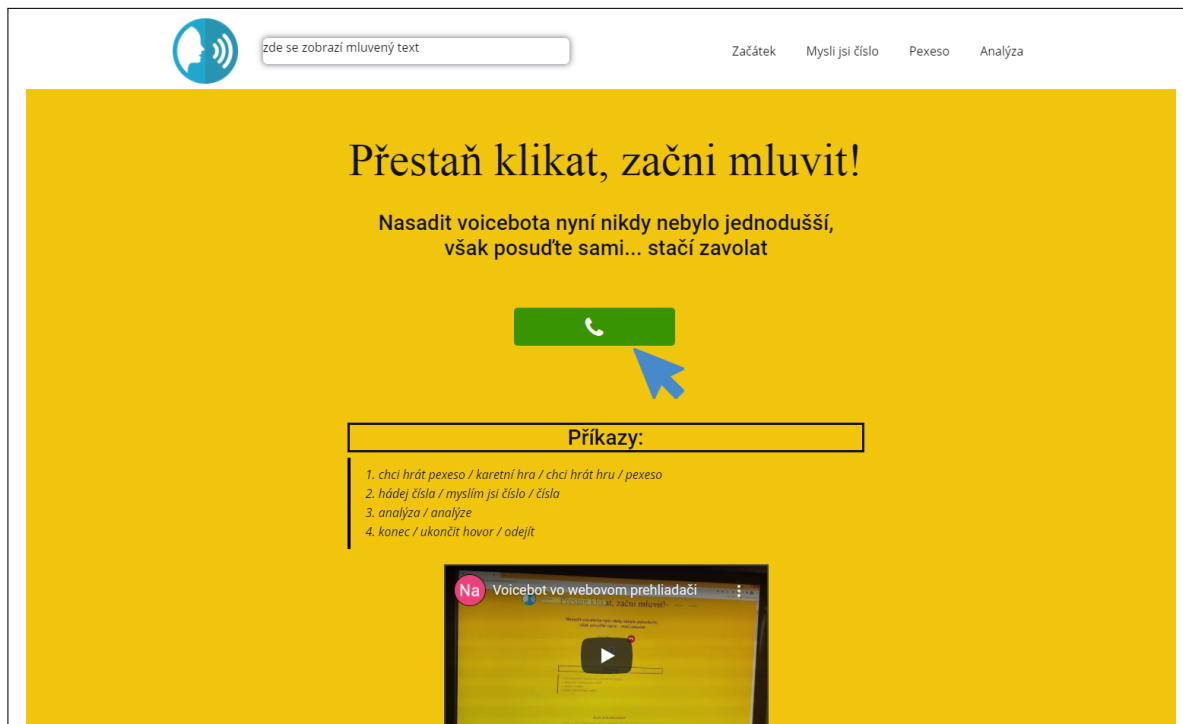


Figure 4.3: Demonstration website, desktop.

## 4.9 Synchronization of voicebot and web client using instant SIP messages

One of the main goals of this thesis is to achieve good synchronization of voice dialog and visual presentation. The synchronization is done through control messages sent between the web client and the voicebot backend. Our former idea was to have a public server with a REST interface as a mediator handling such messages. We used the HTTP Relay service. [relay](https://httprelay.io/)<sup>10</sup>. It is a lightweight, open-source, and cross-platform solution for delivering text messages. The communication was designed to work in the following way: A site willing to send a message sends the message on the HTTP Relay server under a specific call ID. Another site checks periodically (e.g., every second) the presence of a new message. If the message is consumed, it is deleted from the server queue. While working on the implementation of HTTP Relay to our solution, we came up with an idea to use SIP messages instead. This means that we can have one synchronization channel for both phone calls and control messages between the web client and the voicebot backend.

We tested the communication between 2 JsSip clients run in browsers through SIP MESSAGE messages, and this approach worked well. Unfortunately, the version of the SIP connector used in Phonexia Voicebot Suite could not handle the SIP MESSAGE extension of the SIP protocol. We decided to replace the SIP connector with a JavaScript instance to have the same SIP implementation on both web client and voicebot backend. Using this approach simplifies the whole solution significantly. The communication is straightforward, without the need for an additional HTTP server. The ability to exchange the synchronization text messages through one channel shared with the telephony is also beneficial. The

<sup>10</sup><https://httprelay.io/>

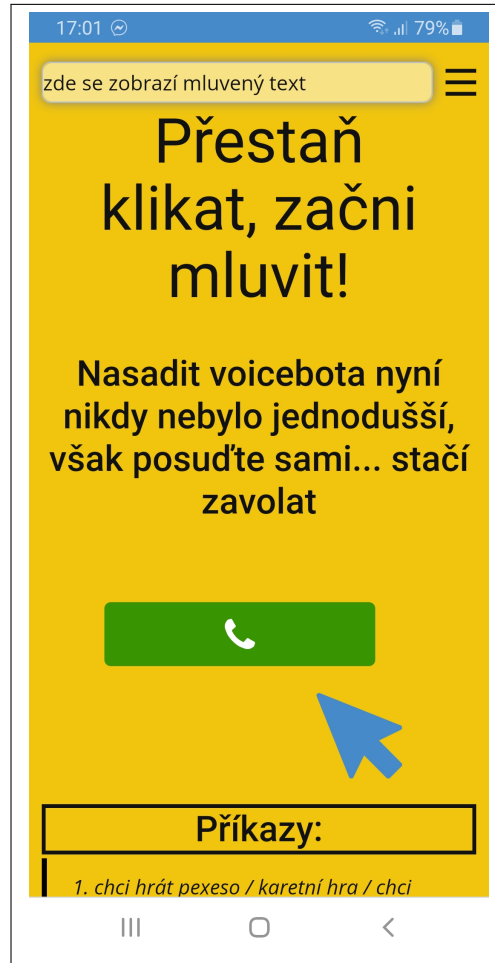


Figure 4.4: Demonstration website, smartphone.

control messages are JSON objects. The structure of messages is not strict. A message consists of keys and values. Both sites — the web client and voicebot backend — have some controllers to process these messages. If there is a defined action for a specific key, the action is executed. We need to transfer this information:

1. Voicebot -> Web client:

- textual transcripts of voice (to display on the website)
- command (an action that is executed in the browser)
- some context variables (a dialog state)
- focus (which HTML element should be emphasized)
- additional data (lemmatized text, search queries, etc.)

2. Web browser -> Voicebot:

- actions done by the user in the graphical part of user interface (for example, a click to some browser element that leads to change of the dialog state)
- additional data (use case specific)

This is a general concept, and the transferred information may vary by uses cases.

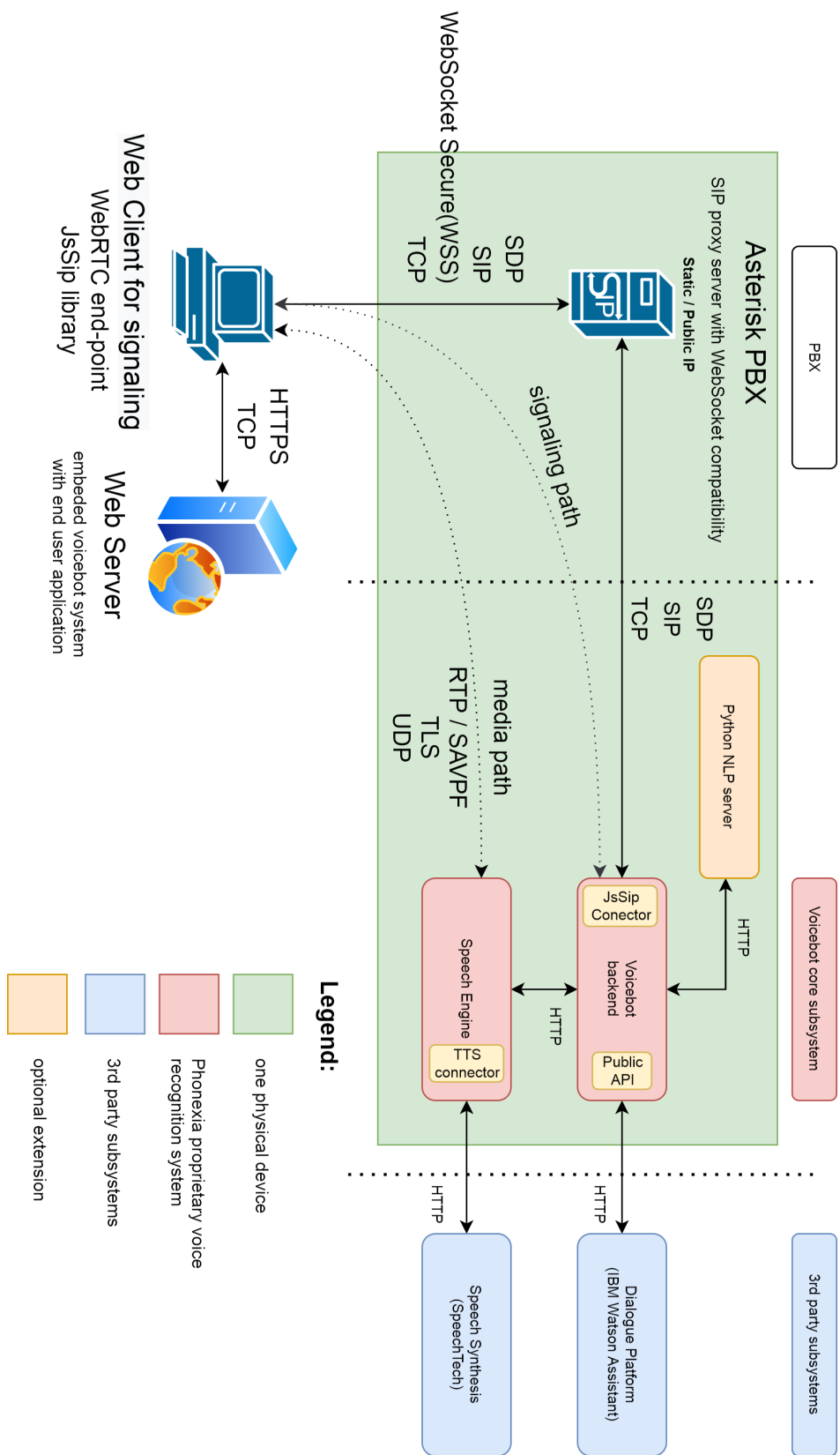


Figure 4.5: Upgraded system design with replaced SIP Connector, including components and used protocols

## Chapter 5

# Dialog system and testing

*„Language is the mark of humanity and sentience, and conversation or dialogue is the most fundamental and specially privileged arena of language“ [7].*

Speaking is one of the skills that distinguish humans from animals. A dialog is a sequence of speaker turns, a sequence of pronounced phrases contributed by each participant. Each party contributes to the dialog alternately. There may be short pauses between speaker turns. People can detect when the other person is about to finish the phrase easily. It is still challenging for automatic systems, especially when there is noise in the background. [7] See the example of a dialog in Listing 5.1.

```
V: Welcome, I am your voice assistant and I will accompany you on this site.
  : What is your name?
H: My name is $name.
V: Nice to meet you $name. This page can be controlled by voice.
  : I would like to begin by telling you that you can interrupt me at
  : any time and stop talking. Where do you live?
H: I live in $city.
V: Great, I'm from there too. $city is a beautiful city.
...

```

Listing 5.1: An example of a voicebot-human dialog, V-voicebot, H-human

### 5.1 TTS

Text-to-speech (or speech synthesis) is an important component of dialog systems important for delivering a convenient user experience. There are three connectors to TTS systems implemented in the Phonexia Voicebot Suit solution. One can choose between providers Acapela, SpeechTech, and Google TTS. SpeechTech and Google are online services. Acapela offers on-premises deployment. The systems need some credentials to work. For experimental use, the easiest way is to use Google TTS, because they offer many languages, and a limited usage is for free. The first 1 million characters for *WaveNet* voices (better quality than *Standard*). For *Standard* voices, the first 4 million characters are free each month. It is necessary to create a Google Cloud account and activate the text-to-speech service. The retrieved credentials in a JSON file need to be put into the voicebot configuration file. We use SpeechTech voice demonstration for purposes as it was pre-configured in Phonexia Voicebot Suite, and it seems to sound better.

## 5.2 Dialog system

Dialog systems are programs that communicate with users in natural language (text, speech, or both). There are two kinds of conversational agents. **Task-oriented dialog agents** use a conversation with users to complete some tasks. The voice assistants like Siri, Alexa, Google Now/Home, Cortana are in this category. They fulfill user requirements like answering questions about weather, booking tickets, or recommending restaurants. Conversational agents can answer questions and even control some devices or robots. According to Jurkovsky, the second category are **chatbots**. Chatbots are systems designed for unstructured conversations, mainly for entertainment purposes. You can see „task-oriented“ chatbots too, but their authors consider them as task-oriented dialog agents. [7]

There is a big difference between human dialog and a dialog with a computer system so far. While human dialogs are limited by the knowledge of participants only and change the topics very often, dialogs with a computer work well in some specific narrow areas so far. Since the dialog system is limited, we can define the flow of a conversation and visualize it as a logic tree. Nodes in the tree may have other sub-trees, also called sub-dialogs. The system recognizes intents (what users would like) and matches them with responses (what your virtual assistant says back). Each node is triggered by a condition matched against the user input. In this thesis, IBM Watson was used for handling dialogs. The Watson natural language processing supports pre-trained intents and predefined named entities. It can match the intents even when the input is not an exact representation of what was seen during the dialog design. [13]

Voicebot dialogs operated on the phone are limited to voice input and voice output. A voicebot enhanced with a graphical user interface has more options for both input and output. IBM Watson Assistant supports multiple channels. A channel connected with a medium, from which/where the information is delivered. Channels enable to use of bots in more places. Every integration (voice, chat, Slack, Facebook) can have a slightly different response. Obviously, in voice communication, reading a URL address pointing to a picture is unwanted. On the other hand, in a chat, it is convenient to display the picture. Then if we speak about chatbots, the chatbot clients may use different text formatting. Some clients use the `<strong>` tag, and some use the `<b>` tag to make the text bold. Also, a voice user prompt can be displayed on the screen in a textual form to prevent cases where it is transcribed to text incorrectly. The user can repeat the prompt in this case, which makes the user experience better than showing wrong answers. Often the user even does not need to repeat the prompt. It is enough to select the right query from a list.

### IBM Watson

A simple dialog was created using the IBM Watson web service. An IBM Watson account needs to be created first. Then it is possible to design the dialog in a web user interface. After that, it is possible to generate credentials for integration with third-party services. These credentials need to be added to the voicebot’s configuration file. There are some limitations in using IBM Watson as an NLP component. The service offers three pricing plans: Lite, Plus, and Enterprise. The Lite plan is for free. The others plans are paid. We use the Lite plan with the following limitations: In case of user inactivity, the Watson Assistant closes the session after 5 minutes. The number of monthly active users is limited to 1000. The dialog version control is disabled for this pricing plan. There is a limitation for a number of skills (dialog size) too. But the Lite plan is enough for our simple dialogs.

The main advantage of IBM Watson over other NLP systems is the support of the Czech language. It can automatically extract built-in (system) entities as currency, date, time, and numbers from the input text. More spread languages like English support even more system entities. It is possible to add custom languages too. Either manually in GUI or through import from a CSV file. A dialog can be created in the graphical interface or imported from a CSV file as well. The system does not offer such variability as when some programming languages are used. Thus complicated dialogs must be done in combination with other services integrated to Watson through a REST call. On the other hand, Watson supports variables, basic mathematical functions, dialog call flow control like jumps to other dialog nodes, and conditional answers for specific user intents. Slots are another handy feature. The slot is a set of information pieces that need to be collected. For example, when collecting information such as a name, a place of birth, and a date of birth, the user can share everything in one utterance, no matter what is the order of the entities. But if some entities are missing or not recognized, the system asks additional questions until all the slot values are collected. Slots guarantee that the full information was provided.

### 5.3 Demonstration dialog design

The demonstration dialog is linked with a web page and application code. It demonstrates voicebot capabilities, the capabilities of graphical presentation, and its synchronization. When a user makes a call from the web client, the voicebot introduces itself. There are three demonstration use cases — Pexeso game, Think a number game, and text analysis. Every section on the website has some commands, which control the use case and related webpage components. The visual synchronization of the voice part of the dialog and the graphical presentation is achieved by a jump to a particular section of the web page. If a user pronounces an utterance related to the Pexeso game, the website scrolls to the Pexeso game too.

The demonstration dialog 5.1 asks a user for his/her name to be able to create personalized answers. It is more convenient for users to call them by their names. If the name is not recognized (the name is not on the predefined list of names), the system will not use personalized sentences and will continue with a general salutation. It is not worth bothering users by asking for the name multiple times when the piece of information is not essential for the dialog. In uses cases, such as booking a flight ticket, the name is essential, and also people would be more cooperative in repeating or spelling their names.

Figure 5.1 shows our demonstration dialog. Commonly, the dialogs contain some repeated patterns. As you can see, some blocks are used more times. For example, *yes* in Think of a number and *yes* in Pexeso. Then it is possible to express yes in many ways. Therefore we created intent for yes to cover multiple possibilities like yes, true, okay, indeed, sure, yea. IBM Watson uses all the examples to train a machine learning model for intent recognition. It is possible to recognize the intent even from unseen sentences then. Every time we need to understand a positive answer, we reuse this block.

#### Think a number game

The rules of the game are easy. The user must think of a number in the range of 1 - 100. Afterward, cards shown in Figure 5.2 are displayed to the user sequentially on the website, and he/she says whether the picked number is present on the card or not. After seven cards, the system tells the user which number he/she was thinking on. Although the game

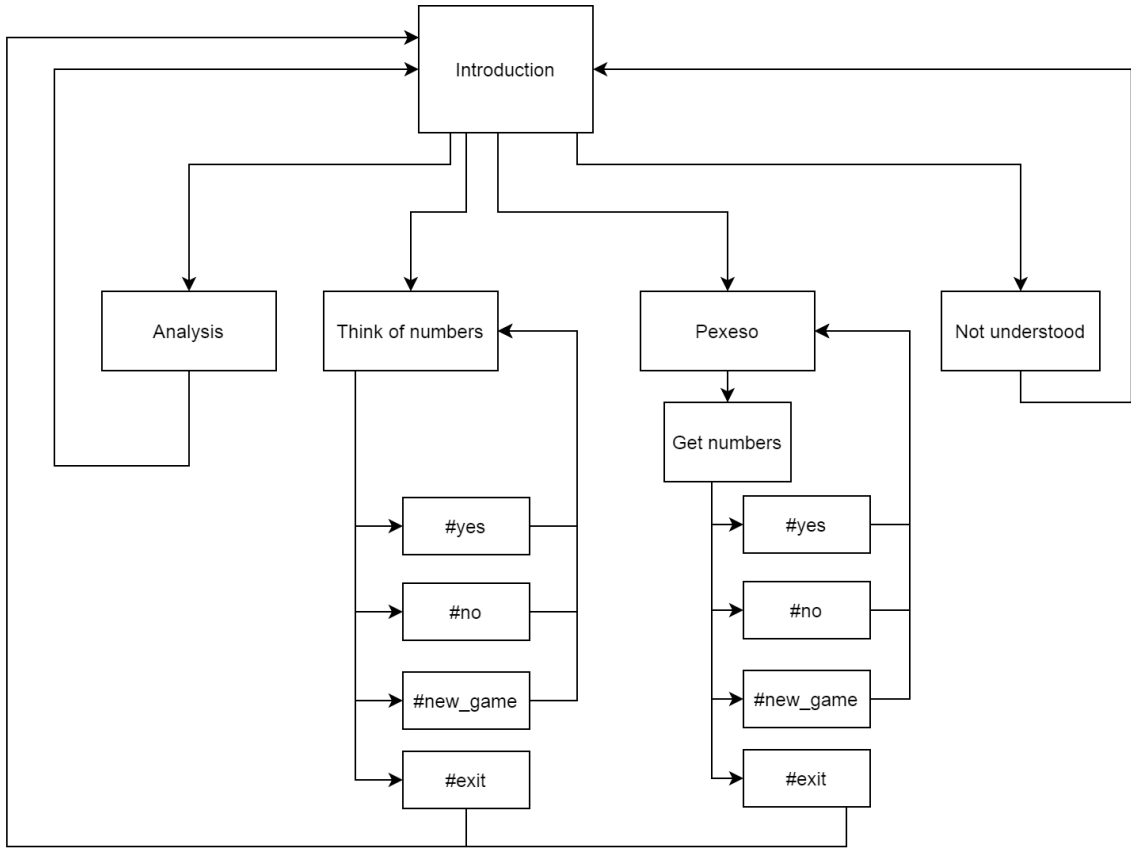


Figure 5.1: Dialog design.

is simple, it demonstrates the possibilities of synchronization between voice and graphical information well. There are many use cases when the user can not use their hands. For example, a cook that follows a recipe on a screen, her/his hands are often dirty. The cooking could be much more convenient without washing and drying hands after each cooking step to list in a cooking book. The user can move next or back in the recipe just with basic commands. Other use cases might be IKEA furniture assembling or repairing a washing machine.

1	3	5	7	9	11	13	15	17	19
21	23	25	27	29	31	33	35	37	39
41	43	45	47	49	51	53	55	57	59
61	63	65	67	69	71	73	75	77	79
81	83	85	87	89	91	93	95	97	99

2	3	6	7	10	11	14	15	18	19
22	23	26	27	30	31	34	35	38	39
42	43	46	47	50	51	54	55	58	59
62	63	66	67	70	71	74	75	78	79
82	83	86	87	90	91	94	95	98	99

4	5	6	7	12	13	14	15	20	21
22	23	28	29	30	31	36	37	38	39
44	45	46	47	52	53	54	55	60	61
62	63	68	69	70	71	76	77	78	79
84	85	86	87	92	93	94	95	100	

8	9	10	11	12	13	14	15	24	25
26	27	28	29	30	31	40	41	42	43
44	45	46	47	56	57	58	59	60	61
62	63	72	73	74	75	76	77	78	79
88	89	90	91	92	93	94	95		

16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	48	49	50	51
52	53	54	55	56	57	58	59	60	61
62	63	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95		

32	33	34	35	36	37	38	39	40	41
42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61
62	63	96	97	98	99	100			

64	65	66	67	68	69	70	71	72	73
74	75	76	77	78	79	80	81	82	83
84	85	86	87	88	89	90	91	92	93
94	95	96	97	98	99	100			

Figure 5.2: Think of a number (game) cards

## Concentration Memory Game — Pexeso

Concentration Memory Game [26], also called Pexeso in Czech, is a card game in which all of the cards are laid face down on a surface, and two cards are flipped face up over each

turn. The goal of this game is to find pairs of matching cards. Two players play against each other usually — a human plays against the voicebot in our case. There are two game difficulties in our demonstrator. A smaller one consists of 16 cards in 4 rows and 4 columns, and a more difficult one consists of 36 cards in 6 rows and 6 columns. The only reason for choosing sizes 4 and 6 is that they create a nice square. Other sizes are too easy to play, or the cards are too small on a smartphone screen. The voicebot selects cards randomly for simplicity. The user can select the cards by pronouncing two numbers. The game options are either to select cards, play a new game, and exit the game. We chose this game because it combines voice and visual presentation well, and there is a limited set of words that are necessary for the control (only numbers and control commands). The game logic is implemented in the dialog layer (IBM Watson), and the GUI is implemented in the web client. At the beginning of the game, the game parameters are initialized and sent to IBM Watson as context variables. When IBM Watson gets card numbers, it can quickly verify whether selected numbers are in the correct range. If not, the user can be asked for new numbers. If everything is correct, IBM Watson sends a command to flip the cards to the web client.

This demo is straightforward. But it could be extended easily. For example, a user could choose a specific theme of cards, or when the right card pair is flipped, the voicebot could present detailed information about the card picture. There is a possibility to offer a multiplayer mode too. The multiplayer mode could be a good demonstrator for the speaker identification technology. His/her voice would identify the player. The visualization of Pexeso can be seen in Figure A.3.

### Analysis of spoken text

This part is demonstration what can be retrieved from the spoken text in the Czech language. It shows state-of-the-art technologies what is possible nowadays. Since Python is a very popular programming language among NLP libraries, we implement a server on localhost, where Python code can be run. The communication between the voicebot’s backend and the Python server is made via REST calls (flask library).

We used the Stanza [20] Python library for natural language analysis. It enables (among other things) to convert a human language text into sentences and words with some annotation. It generates base forms of the words (lemmatization), their parts of speech tags, morphological features, and a syntactic structure. Features as lemmatization might be useful for new use cases with many ways, how to give a command.

We recommend Stanza because there are not many toolkits supporting the Czech and Slovak languages. Next to these languages, Stanza has pre-trained neural models supporting 66 languages in total. The other models are easily downloadable by calling a library’s download method. The library can work without an internet connection after downloading the models. You may encounter difficulty when installing Stanza due to memory consumption. The Stanza library installs PyTorch automatically. During our installation, everything passed fine. The related packages were installed using via *apt-get* with no error. However, then Stanza did not work. As we found later, we ran out of memory. The torch package has more than 700MBs, and the system killed some installation processes. We were able to install Stanza after adding the *-no-cache-dir* to the *pip install* command. An example of the Stanza output displayed on our website is seen in Figure A.1.



Another information retrieval tool for Czech is Google Knowledge Graph Search API <sup>1</sup>. It shows the same structured information as Google shows in the right panel during standard Google Search. This might be interesting for getting additional information about famous persons or objects. The Czech-speaking people can not get this information via voice yet because Google Assistant does not support the Czech language yet. One needs to enable this API in the Google Developer console and follow a tutorial on their website to use it. A limited number of queries is for free. An example of the retrieved information for the utterance „pražský hrad“ (Prague castle) is shown in Figure A.2.

## 5.4 Testing

### Call establishment

We tested to set up a call with several devices (notebooks, smartphones), as well as with several internet service providers (cable, mobile) and two internet browsers (Google Chrome, Mozilla Firefox). In one case, the call did not work as expected. The call was established, but there was no sound. It happened because of the enabled Block fragmented IP packets option in the local router COMPAL CH7465LG under firewall settings at home. After disabling this option in the router, the call worked as expected. This observation confirms that for most of the calls, just STUN is enough for NAT traversal, but in some cases, also TURN server is needed. It is easier to operate our own TURN server than to force users to change their firewall settings.

### Testing wireless headphones

Bluetooth headphones connected to a mobile worked as well. It used a headphone microphone, and thus the voice input was cleaner and the transcription event even more accurate than with the mobile phone only. The mobile phone had a turned-off display. Later, we discovered that the RTP stream with the turned-off display works only on WIFI. It did not work on the cellular network with mobile data turned on. It seems that the browser in the tested smartphone Samsung Galaxy S9 optimizes battery consumption and stops the RTP stream while the browser is not on screen. When we switched the display on again, it started to work. This behavior might vary by a web browser, smartphone, and operating system. This outcome is very important as it needs to be addressed to bring practical voicebots to users.

### Testing dialog on users

Users have to confirm access to the microphone in the web browser for the voicebot client to make a call after loading the demonstration website users. The confirmation is not difficult for an average computer user. The first call establishment takes about 5 seconds till a user got the first voice response. The delay is very individual. The second call establishment is faster, and there is no delay. The users were confused after the call was established at first. They didn't know what to say. What is the voicebot capable of. Based on this observation, some example commands and a tutorial video explaining how to use the voicebot were added to the website's landing page. Also, it was seen that the speech recognizer understands better native speakers than foreign people, even if the languages

---

<sup>1</sup><https://developers.google.com/knowledge-graph>

are very similar, like Czech and Slovak. There is a limitation that the voicebot can serve one person simultaneously (due to the pairing of call ids and message-ids that needs to be solved for production). If multiple web client instances are opened simultaneously, the solution does not work properly. It modifies the web page of the firstly loaded web client. We built the new SIP connector based on the JsSIP JavaScript library into the voicebot backend and proved that this synchronization channel works. But there was no time to finish the issue with the pairing of ids. Either the *Contact* field of the SIP INVITE message could be used, or a new field could be introduced to the SIP message headers.

## Chapter 6

# Conclusion

The main idea of this work is to complement an automatic voice dialog between user and machine with a graphical presentation through a web browser. Automatic voice dialogs are a hot topic nowadays because the technology is ready for practical deployments and can save a lot of money for companies. On the other hand, it is common that each company has its web page, a web presence. It is just a matter of time when the voice presence through some automatic voice assistants becomes standard too. It is convenient for a user to speak. The speech might be the only possibility for users with disabilities. Joint voice presence and web presence bring even higher convenience, new use cases, and higher benefits to users. Therefore we decided to explore this area.

The work was done in collaboration with the speech-oriented company Phonexia. One of the aimed goals was to be able to present their voice products on their website live. The work intersects several knowledge areas such as computer networks, telecommunications, speech processing, natural language processing, computer security, and software development. The work has an exploration and implementation character.

The Introduction chapter [1](#) describes current trends in telecommunication and our motivation for the work. We describe the purpose of WebRTC technology and discuss its enormous benefits for the web. In the section [2.1](#), we discuss how voicebots can be integrated with this technology.

The second Technical Background chapter [2](#) describes the theoretical part where we analyzed the real-time communication technology over the internet from a web browser (details of WebRTC). In the Telephony section [2.2](#) describe how a VoIP telephony connection can be established for various network configurations (NAT traversing, firewalls). Also, the telecommunication protocols SIP and SDP are described. The systems section [2.3](#) describes typical components of a voicebot system.

The third System design chapter [3](#) describes our voicebot solution integrated into the web browser environment. Phonexia provided its own product Phonexia Speech Engine (SPE), for this work. It offers speech transcription (STT) natively and speech synthesis (TTS) through some external plugins. Together with a telephony connector (SIP connector) and a JavaScript backend logic, it forms Phonexia Voicebot Suite, a set of components aimed at the building of voicebots. Phonexia Voicebot Suite can connect components like STT (SPE), TTS, and NLP into one working solution. The basic voice assistant uses VoIP telephony. It is connected with an open-source Private Branch Exchange (PBX) - Asterisk. Asterisk can be connected to a software telephone network, and thus voicebot is accessible from any connected telephony device. Our voicebot solution was built on the top

Phonexia Voicebot Suite. It added web telephony clients and a possibility to handle visual information in the dialog next to voice.

The Implementation chapter 4 describes how approached the problem in detail. The individual sections focus on Asterisk configuration, describe the source code of the web client, and describe how we exchange the synchronization messages between web client and voicebot backend. We improved the original Voicebot Suite design by removing the standalone SIP connector and implementing a new SIP agent directly to the voicebot backend. The new implementation supports to exchange of text messages between the web client and voicebot backend that were used for synchronization. This enabled us to use the same signaling/synchronization channel for both telephony and visual content. It removed the need for other components for transferring non-telephony data. In addition, we added Python libraries support for more advanced natural language processing.

The Dialog system and testing chapter 5 presents how the voice dialog and web client visual presentation is composed into one presentation. We demonstrate the synchronization of the website and the voice dialog on the Pexeso Concentration Memory Game. The demonstration dialog graph was encoded in the IBM Watson environment. The whole solution was tested and adjusted based on user feedback.

This work fulfilled the assignment. Its added value is significant. It proved the possibility of deploying a voice assistant directly on the website and opened the massive potential of making voice assistants more intuitive and convenient through a combination of voice and visual presentation. The way was mapped. All the used technologies, their configurations, and ways of integration are described in this thesis or the complimentary digital files.

Also, the space for further improvements is huge. One of the interesting options might be to add other speech technologies implemented in SpeechEngine, such as language identification, voice biometry, gender recognition, or age estimation.

# Bibliography

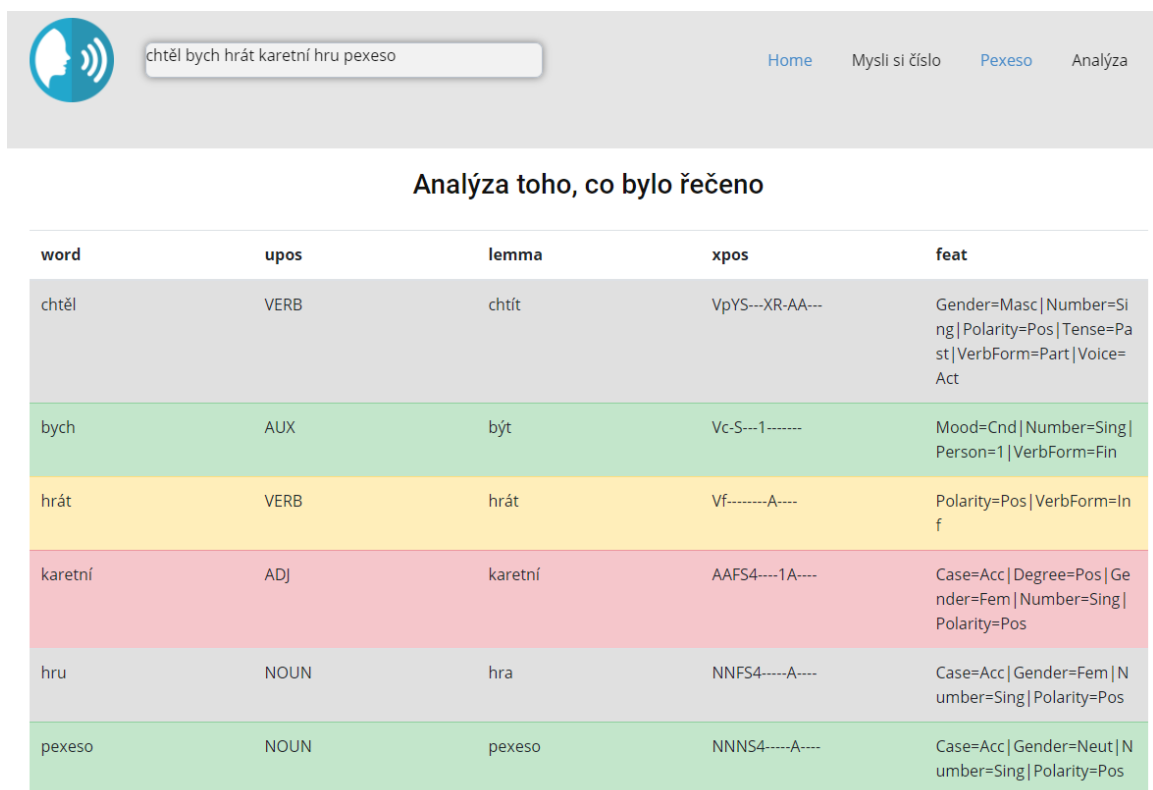
- [1] ADAMS, K. *FAQ's SIP vs. CHAN\_SIP vs. CHAN\_PJSIP* [online]. LinkedIn blog, july 2016 [cit. 2021-06-17]. Available at: <https://www.linkedin.com/pulse/faqs-sip-vs-chansip-chanpjsip-kent-adams>.
- [2] ALVSTRAND, H. *Overview: Real Time Protocols for Browser-based Applications* [online]. draft-ietf-rtcweb-overview-19. Internet Engineering Task Force, november 2019 [cit. 2020-10-06]. Available at: <https://tools.ietf.org/html/draft-ietf-rtcweb-overview-19>.
- [3] CASTILLO, I. B., VILLEGAS, J. M. and PASCUAL, V. *The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)* [online]. Internet Engineering Task Force (IETF), january 2014 [cit. 2020-12-28]. RFC 7118. Available at: <https://tools.ietf.org/html/rfc7118>.
- [4] ETEMAD, E. J., RIVOAL, F. et al. *W3C Process Document* [online]. The World Wide Web Consortium (W3C), september 2020 [cit. 2020-10-08]. Available at: <https://www.w3.org/2020/Process-20200915/>.
- [5] HANDLEY, M., JACOBSON, V. and PERKINS, C. *SDP: Session Description Protocol* [online]. Network Working Group, july 2006 [cit. 2020-10-18]. RFC 4566. Available at: <https://tools.ietf.org/html/rfc4566>.
- [6] HICKSOND, I. *WebRTC 1.0: Real-time Communication Between Browsers* [online]. Cullen Jennings and Henrik Boström and Jan-Ivar Bruaroey and others. Draft - W3C Candidate Recommendation 24 September 2020. The World Wide Web Consortium (W3C), september 2020 [cit. 2020-10-06]. Available at: <https://www.w3.org/TR/2020/CRD-webrtc-20200924/>.
- [7] JURAFSKY, D. and MARTIN, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Draft 3th ed. 2020 [cit. 2021-06-28]. Available at: [https://web.stanford.edu/~jurafsky/slp3/ed3book\\_dec302020.pdf](https://web.stanford.edu/~jurafsky/slp3/ed3book_dec302020.pdf).
- [8] KERANEN, A., HOLMBERG, C. and ROSENBERG, J. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal* [online]. Internet Engineering Task Force (IETF), july 2018 [cit. 2020-10-13]. RFC 8445. Available at: <https://tools.ietf.org/html/rfc8445>.
- [9] LACHAPELLE, S. *What was the history of WebRTC inside Google before it was released to the public?* [online]. Quora, september 2013 [cit. 2020-10-06]. Product Manager, Chrome WebRTC. Available at:

- <https://www.quora.com/WebRTC/What-was-the-history-of-WebRTC-inside-Google-before-it-was-released-to-the-public/answer/Serge-Lachapelle>.
- [10] LORETO, S. and ROMANO, S. P. *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser*. 1st Editionth ed. O'Reilly Media, 2014. ISBN 978-1449371876.
  - [11] MARJOU, X. and SOLLAUD, A. *Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows* [online]. Internet Engineering Task Force, june 2011 [cit. 2021-06-11]. RFC 6263. Available at: <https://tools.ietf.org/html/rfc6263>.
  - [12] MATTHEWS, P., ROSENBERG, J. et al. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)* [online]. Internet Engineering Task Force (IETF), february 2020 [cit. 2020-10-13]. RFC 8656. Available at: <https://tools.ietf.org/html/rfc8656>.
  - [13] MILLER, M. *Getting started with a dialog skill* [online]. IBM Cloud Docs, may 2021 [cit. 2021-06-28]. Available at: <https://cloud.ibm.com/docs/assistant?topic=assistant-gs-dialog>.
  - [14] MOZILLA DEVELOPER NETWORK CONTRIBUTORS. *Media Capture and Streams API (Media Stream)* [online]. Mozilla, february 2020 [cit. 2020-10-09]. Available at: [https://developer.mozilla.org/en-US/docs/Web/API/Media\\_Streams\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API).
  - [15] MOZILLA DEVELOPER NETWORK CONTRIBUTORS. *RTCDataChannel* [online]. Mozilla, april 2020 [cit. 2020-10-09]. Available at: <https://developer.mozilla.org/en-US/docs/Web/API/RTCDataChannel>.
  - [16] MOZILLA DEVELOPER NETWORK CONTRIBUTORS. *RTCPeerConnection* [online]. Mozilla, august 2020 [cit. 2020-10-09]. Available at: <https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection>.
  - [17] MOZILLA DEVELOPER NETWORK CONTRIBUTORS. *WebRTC API* [online]. Mozilla, september 2020 [cit. 2020-10-13]. Available at: [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API).
  - [18] MOZILLA DEVELOPER NETWORK CONTRIBUTORS. *Broadcast Channel API* [online]. Mozilla, february 2021 [cit. 2021-06-09]. Available at: [https://developer.mozilla.org/en-US/docs/Web/API/Broadcast\\_Channel\\_API](https://developer.mozilla.org/en-US/docs/Web/API/Broadcast_Channel_API).
  - [19] PETIT HUGUENIN, M., SALGUEIRO, G., ROSENBERG, J. et al. *Session Traversal Utilities for NAT (STUN)* [online]. Internet Engineering Task Force (IETF), february 2020 [cit. 2020-10-13]. RFC 8489. Available at: <https://tools.ietf.org/html/rfc8489>.
  - [20] QI, P., ZHANG, Y., ZHANG, Y., BOLTON, J. and MANNING, C. D. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2020. Available at: <https://nlp.stanford.edu/pubs/qi2020stanza.pdf>.

- [21] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G. et al. *SIP: Session Initiation Protocol* [online]. Network Working Group, june 2002 [cit. 2020-10-18]. RFC 3261. Available at: <https://tools.ietf.org/html/rfc3261>.
- [22] ROSENBERG, J., WEINBERGER, J., HUITEMA, C. et al. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)* [online]. Network Working Group, march 2003 [cit. 2020-10-10]. RFC 3489. Available at: <https://tools.ietf.org/html/rfc3489>.
- [23] ROY, R. R. *Handbook of SDP for Multimedia Session Negotiations*. First editionth ed. CRC Press, 2018. ISBN 9781138484498.
- [24] SRISURESH, P. and EGEVANG, K. *Traditional IP Network Address Translator (Traditional NAT)* [online]. Network Working Group, january 2001 [cit. 2020-10-09]. RFC 3022. Available at: <https://tools.ietf.org/html/rfc3022>.
- [25] UBERTI, J. and DUTTON, S. *Real-time communication with WebRTC: Google I/O 2013* [online]. San Francisco: Google, may 2013 [cit. 2020-10-17]. Available at: <https://www.youtube.com/watch?v=p2HzZkd2A40>.
- [26] WIKIPEDIA CONTRIBUTORS. *Concentration (card game)* — *Wikipedia, The Free Encyclopedia*. 2021. [Online; accessed 11-July-2021]. Available at: [https://en.wikipedia.org/w/index.php?title=Concentration\\_\(card\\_game\)&oldid=1031609244](https://en.wikipedia.org/w/index.php?title=Concentration_(card_game)&oldid=1031609244).
- [27] WING, D. *Symmetric RTP / RTP Control Protocol (RTCP)* [online]. Network Working Group, july 2007 [cit. 2020-10-13]. RFC 4961. Available at: <https://tools.ietf.org/html/rfc4961>.

# Appendix A


## Demonstration website



word	upos	lemma	xpos	feat
chtěl	VERB	chtít	VpYS---XR-AA---	Gender=Masc   Number=Sing   Polarity=Pos   Tense=Past   VerbForm=Part   Voice=Act
bych	AUX	být	Vc-S---1-----	Mood=Cnd   Number=Sing   Person=1   VerbForm=Fin
hrát	VERB	hrát	Vf-----A----	Polarity=Pos   VerbForm=Inf
karetní	ADJ	karetní	AAFS4----1A----	Case=Acc   Degree=Pos   Gender=Fem   Number=Sing   Polarity=Pos
hru	NOUN	hra	NNFS4----A----	Case=Acc   Gender=Fem   Number=Sing   Polarity=Pos
pexeso	NOUN	pexeso	NNNS4----A----	Case=Acc   Gender=Neut   Number=Sing   Polarity=Pos

Figure A.1: Sentence analysis using the Stanza toolkit [20] on the demonstration website. The translation of the sentence is: „I would like to play the card game Pexeso“.







[Home](#)
[Mysli si číslo](#)
[Pexeso](#)
[Analýza](#)

**Additional information**

Key	Value
name	Pražský hrad
description	Hrad v Praze, Česko
detailedDescription	Pražský hrad je nejvýznamnější český hrad stojící na skalnatém ostrohu nad řekou Vltavou v centru Prahy, na vrchu Opyš. Od 9. století býval sídlem českých knížat, později králů a od roku 1918 je sídlem prezidenta republiky.


Figure A.2: Sentence analysis using Google Knowledge Graph Search API on the demonstration website





[Home](#)
[Mysli si číslo](#)
[Pexeso](#)
[Analýza](#)

**PEKELNĚ SE SOUSTŘEĎ (Pexeso)**

0



Zámek arnoštůvká hrad Blatná

3



Zámek arnoštůvká hrad Blatná

6

7


Hrad Kokořov

8


Zámek Křivčice

9

10

11

12

13

14

15

Figure A.3: An example of Concentration Memory Game (Pexeso) on the website.

## Appendix B

# Description of the digital part of the work

The bold names are directories. The non-bold names are files. Note that this is not a complete directory tree. Only some important files are listed.

<b>Root directory</b>	<b>Description</b>
<b>/asterisk_config</b>	Asterisk configuration files
asterisk.conf	general Asterisk settings
extensions.conf	call routing setting
http.conf	Asterisk WebSocket server configuration
modules.conf	specifies which modules are loaded
pjsip.conf	PJSIP SIP channel config (endpoints, transport layer, ...)
rtp.conf	setting of RTP (ports, ICE, STUN, TURN)
<b>/nlp_backend</b>	
pythonNlp.py	a directory with Python NLP libraries
README.txt	information about additional NLP libraries
<b>/speech_engine</b>	SpeechEngine is a proprietary speech server developed by company Phonexia
...	
README.txt	information how to get the SpeechEngine
<b>/web_client</b>	web client (frontend)
<b>/js</b>	website dynamic content
config.js	configuration values
controller.js	the main part controlling call and website content
pexeso.js	Pexeso game controller
<b>/css</b>	cascade files for the website
<b>/lib</b>	
jssip-3.7.1.js	JsSip library for SIP signalling
index.html	landing page on the website

<b>Root directory</b>	<b>Description</b>
<b>/voicebot_backend</b>	layer integrating all components together
<b>/call</b>	call management code
CallService.js	A places where the SIP messages arrive. Communication
...	with the NLP Python backend and NLP providers.
<b>/nlp</b>	NLP providers can be stored here
<b>/watson</b>	
WatsonNlpProvider.js	IBM Watson connector (the context variables between
...	the Watson and the web client are interchanged here)
<b>/settings</b>	
phxvbot.yml	configuration of voicebot client components (NLP, SPE,
...	TTS...)
<b>/spe</b>	SpeechEngine
<b>/sip</b>	SIP protocol implementation
<b>/voicebot</b>	voicebot implementation
start	a script to start all components via systemctl
stop	a script to stop all components via systemctl
is-active	a script for checking components' status, if they are active
<b>/turn_server</b>	
README.txt	Instructions to install the coTurn server
deploy.sh	An example script for deploying the coTurn server
<b>/systemd</b>	configuration of systemd services
asterisk.service	
nlp-backend.service	
spe.service	
voicebot.service	
<b>/watson_dialog</b>	
demonstration_voicebot.json	IBM Watson dialog, which can be imported to the online
...	service
README.txt	Instructions to import the dialog.
demonstration_video.mp4	a demonstration video showing the system
master_thesis.pdf	Master thesis PDF file
source_text.zip	Master thesis source text with all the resources, created
	in the online Latex editor — Overleaf