CSE310 – JAVA

Submission Report

**Personal Assistant**

## School of Computer Science & Engineering

**LOVELY PROFESSIONAL UNIVERSITY**

March 2023



## SUBMITED TO – Mrs. Shruti

|  |
| --- |
| Name - Aditya Sharma  Registration Number - 12111592 |
| Name - Amit Kumar Dansena  Registration Number - 12111643 |
| Name - Saquib Nadeem  Registration Number - 12105826 |
| Name - Abhishek Kumar  Registration Number - 12106473 |
| CLASS NAME - K21DP  TEAM NUMBER - |

**Abstract**

This paper tries to give a comprehensive introduction to state-of-the-art Text-To-Speech (TTS) synthesis by highlighting its Digital Signal Processing (DSP) and Natural Language Processing (NLP) components. As a matter of fact, since very few people associate a good knowledge of DSP with a comprehensive insight into NLP, synthesis mostly remains unclear, even for people working in either research area. Section 2.1 begins with a presentation of the many practical NLP problems which must be solved by a TTS system. We then examine, in Section 2.2, how synthetic speech can be obtained by simply concatenating elementary speech units, and what choices must be made for this operation to yield high quality. We finally give a word on existing TTS solutions, with special emphasis on the computational and economical constraints which have to be kept in mind when designing TTS systems.

**Introduction**

A Text-To-Speech (TTS) synthesizer is a computer-based system that can do this. Read text aloud regardless of whether the operator typed it directly into the computer or scan it and send it to his OCR (Optical Character Recognition) system. Let’s clarify there is a fundamental difference between the system we are trying to have for other talking machines (such as cassette players) discussed here, we feel that you are interested in automated production of new sets of the definition still needs some refinements. A system that is just isolated and linked only words or phrases labeled Voice Response System apply if you need a limited vocabulary (usually a few hundred words) and when the pronounced sentence follows a very restricted structure. For announcements of arrivals at stations. as part of TTS. In synthesis it is impossible (and fortunately useless) to collect and store all the words, language. Therefore, we recommend defining Text-To-Speech as automatic. Speech generation by transcribing sentences from graphemes to phonemes to pronounce. At first glance, this task does not seem too difficult to accomplish. That's not a man after all we may be able to pronounce unfamiliar sentences correctly, even if they are our own childhood? We all have a deep knowledge of the rules of reading, almost subconsciously in our native language. sent a simple version when we were in elementary school. The school and we have improved it year after year. but that would be a bold claim. In fact, to say that it's just a small step before computers can handle it. Human in that respect. Despite the current state of our knowledge and technology, Recent advances in signal processing and artificial intelligence. The reading process draws from the deepest, often unthoughtful depths of human beings’ intelligence.

**Litrature review**

Since the advent of deep learning techniques and the accessibility of big voice datasets, text-to-speech (TTS) systems have been the focus of research for several decades. The production of speech that sounds natural and has appropriate prosody and intonation is one of the main issues in TTS.

TTS systems employ a number of approaches, such as unit selection TTS, parametric TTS, and concatenative TTS. Concatenative TTS combines pre-recorded voice fragments to create new words or phrases, whereas parametric TTS creates speech using mathematical models. By choosing the most relevant pre-recorded speech segments to provide high-quality speech output, unit selection TTS combines the advantages of both methods.

The widely used Java Speech API (JSAPI) is a tool for creating TTS systems. It supports various languages and offers a foundation for speech synthesis and recognition. Several TTS systems, such FreeTTS and MaryTTS, which are free and open-source programs that support numerous languages, have made use of JSAPI.

Existing TTS programs like Festival TTS and MaryTTS have demonstrated positive outcomes in terms of the clarity and intelligibility of speech. Since these systems are flexible and adaptable, they can be used for a variety of purposes, such as language acquisition and assistive technologies.

**Text Analysis**

Text parsing is a complex process consisting of several modules. These modules include:

* Preprocessor module:

This module is responsible for organizing input sentences into manageable word lists. It also identifies numbers, abbreviations, acronyms and idioms and turns them into full text if necessary. Punctuation ambiguity, including punctuation detection, is also resolved to some extent with basic regular grammars.

* Morphology module:

This module suggests all possible parts of speech for each word according to their spelling. Inflected words, derivatives, and compound words are broken down into their basic graph units using simple regular grammars that exploit root and affix vocabulary.

* Contextual analytics module:

This module considers words in their context, which allows it to reduce the list of possible speech-part categories to a limited number of highly probable hypotheses, based on the corresponding possible speech portion of the neighboring words. This can be achieved using n-grams, multi-layer perceptron’s, or non-random local grammar provided by professional linguists, or automatically inferred from the training dataset using the technique. CART.

* Parser-Prosodic:

This module examines the remaining search space and finds the structure of the text, i.e., organization of the text into clauses and sentence-like elements, which is most closely related to tonal performance its expected.

A screenshot of a computer

Description automatically generated with medium confidence

**Automatic Phonetization**

The Letter-To-Sound (LTS) module automatically determines the phonetic transcription of incoming text. At first glance, the LTS task may seem as simple as performing a dictionary lookup, but in reality, there are some challenges that must be overcome.

First, pronunciation dictionaries usually refer only to roots and do not consider morphological variations such as plurals, conjugations, and inflections. This requires incorporating a specific component of phonology, called morphology, into the LTS module.

Second, some words have multiple dictionary entries or morphological analyses, resulting in different pronunciations. Heterophonic, words spelled the same but pronounced differently, cause the most ambiguity in pronunciation.

Third, pronunciation dictionaries provide transcriptions, not transcripts. For example, consonants can reduce or remove clusters. This is a phenomenon known as consonant cluster simplification.

Fourth, compound words in sentences are not pronounced like isolated words. Variations in phonetic association, phonetic stretching, and other phonological processes can occur by organizing sentences into non-lexical units. Finally, new words and proper nouns are not in phonetic dictionaries, so their pronunciation must be inferred from known similarly sounding words.

LTS engines can be organized in many ways, but dictionary-based and rule-based strategies are the most common. Dictionary-based solutions need to store phonological knowledge in vocabularies and consider inflection, derivation, and complex phoneme-morphological rules that describe the phonetic transcription of morphological constituents when combined into words. I have. Morphemes not in the vocabulary are written according to rules. A rule-based phonetic system translates most phonological abilities from a dictionary into an alphabetical set of sound rules. Only words that form their own rules are stored in the exception dictionary. Phonetic post-processing is often applied to consider joint smoothing after the phonetic transcription of each word is obtained.

**DSP Component**

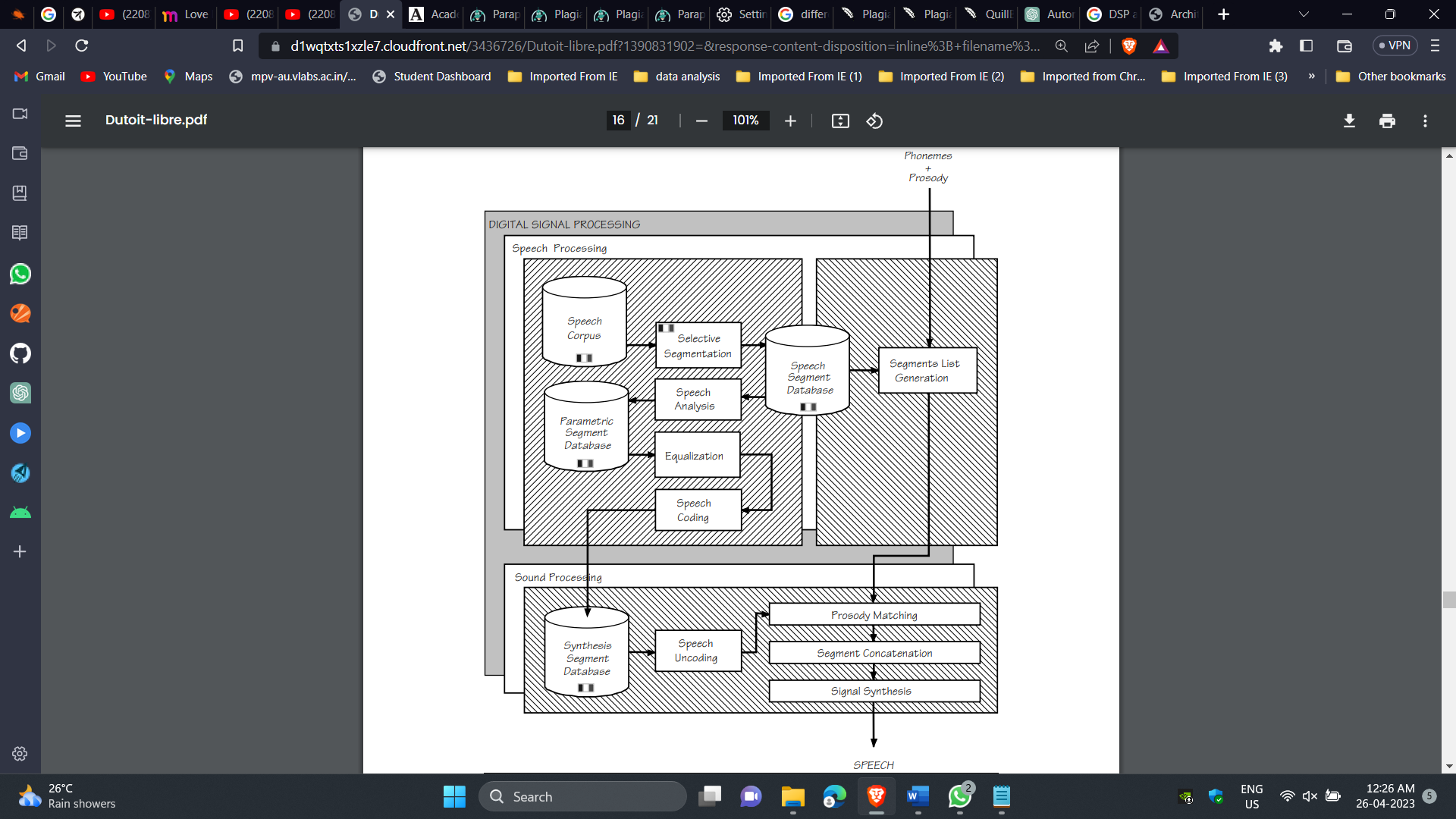
Synthesis-by-rule is a TTS approach that uses a set of pre-defined rules to generate speech. These rules describe the relationships between phonemes and their contextual variations, such as allophones and co-articulation effects. The synthesis process involves breaking down the text into phonetic units, applying the rules to generate a sequence of speech units, and finally concatenating them to form the synthesized speech signal. This approach relies on expert knowledge and linguistic expertise to create accurate and natural-sounding speech.

Synthesis-by-concatenation, on the other hand, is a TTS approach that uses a database of pre-recorded speech segments as the building blocks for generating synthesized speech. The speech segments are typically recorded by a human speaker and cover a wide range of phonetic variations and co-articulation effects. During synthesis, the text is first converted into a sequence of phonetic units, which are then matched to the closest speech segments in the database. The selected speech segments are then concatenated to form the synthesized speech signal. This approach can generate highly natural-sounding speech with good prosodic features, but it requires a large database of high-quality speech recordings and advanced signal processing techniques to ensure smooth concatenation.

Both synthesis-by-rule and synthesis-by-concatenation have their advantages and disadvantages, and their choice depends on the specific requirements of the TTS application. Some modern TTS systems use a hybrid approach that combines elements of both techniques to achieve high-quality and flexible synthesis.

**Rule Based Synthesizer**

Rule-based synthesizers are a cognitive, generative approach to speech synthesis that use a series of rules to describe the influence of phonemes on one another. They are primarily favored by phoneticians and phonologists due to their ability to study the characteristics of natural speech and investigate physiological constraints. Rule-based synthesizers typically appear in the form of formant synthesizers, which describe speech as the dynamic evolution of up to 60 parameters related to formant and anti-formant frequencies and glottal waveforms. However, the large number of coupled parameters complicates the analysis stage and tends to produce analysis errors. Despite these challenges, rule-based synthesizers remain a potentially powerful approach to speech synthesis, with applications in studying speaker-dependent voice features and changes in speaking styles. They have been widely integrated into TTS systems for various languages, including English and French.



**Methodology**

1. Import necessary libraries:

The code starts with importing the necessary libraries like Voice and WebDriver from the com.sun.speech.freetts and org.openqa.selenium packages respectively.

2. Create a class:

The code has two classes. The first class is the base class, which initializes the voice and greets the user. The other class is the child class that extends the first class and contains the main method.

3. Initialize the voice:

In the constructor of the first class, an instance of Voice is created using the third class. The "mechanical" voice is allocated and the volume is set to 1.0f. The style of the voice is set to "robotic". The voice then greets the user depending on the time of the day.

4. Accept user input:

The program accepts user input using a Scanner object.

5. Execute user requests: The user's input is then used in a switch statement that executes the requested action. For example, if the user says "open chrome", a ChromeDriver instance is created and the Google website is opened.

6. Terminate the program: If the user says "exit", the program terminates with a farewell message.

7. Error handling: If the user's input does not match any of the cases in the switch statement, the program outputs "Cannot Understand" and prompts the user to try again.

Overall, the code uses the Java programming language and its libraries to implement voice-controlled actions using Selenium WebDriver. The code structure follows object-oriented programming principles, with classes and methods organized logically to achieve the desired functionality.

**Conclusion**

The code in a Java program need to use various libraries and APIs to interact with the user through voice input and output, and performs various tasks based on the user's commands. The program consists of two classes, base class and derived class, where a base class and a derived class that extends the base class.

The base class is responsible for greeting the user based on the current time of the day and initializing the voice object for further communication. It uses the `Voice` and `VoiceManager` classes from the `com.sun.speech.freetts` library to generate voice output.

The derived class is responsible for interacting with the user through voice input and performing various tasks based on the user's commands. It uses the `Scanner` class to get user input from the console, and based on the user's command, it performs various actions such as getting the current time, date, and day of the week, opening different websites on the browser, and launching different applications on the computer.

The program uses the `ChromeDriver` class from the `org.openqa.selenium.chrome` package to open different websites on the Google Chrome browser, and the `Desktop` class from the `java.awt` package to launch different applications on the computer.

Overall, the program demonstrates how different APIs and libraries can be used to build a voice-based assistant that can interact with the user and perform various tasks based on the user's commands. However, it should be noted that the program has several limitations and may not work as expected in some cases. For example, the program assumes that the user is using a Windows operating system, and it may not work on other operating systems. Additionally, the program has limited functionality and may not be able to perform more complex tasks that require advanced AI algorithms or machine learning models. However there are still some issues with the TTS. These are all important issues that need to be addressed to further improve the quality of text-to-speech (TTS) systems. Let me briefly address each of these:

1. Accounting for coarticulatory phenomena: Coarticulation refers to the phenomenon where the articulation of one speech sound affects the articulation of the adjacent sounds. TTS systems need to account for coarticulation to generate natural-sounding speech. One approach is to use unit selection, where the TTS system selects pre-recorded speech segments that match the input text and concatenate them to produce the final output. The selection of these segments can be optimized based on coarticulation.

2. Formalizing the relationship between syntax, semantics, pragmatics, and prosody: The prosody of speech refers to the melody and rhythm of speech, including intonation, stress, and duration. To generate natural-sounding speech, TTS systems need to generate prosody that is appropriate for the input text. This requires understanding the relationship between the linguistic structure of the input text and the prosodic patterns that are associated with different meanings and contexts.

3. Accounting for variability in prosodic patterns: Prosodic patterns can vary depending on factors such as emotion, emphasis, and speaker identity. TTS systems need to be able to generate speech that captures this variability, while still sounding natural and coherent.

4. Accounting for speaker and speaking style effects: Speakers have different voices and speaking styles, and TTS systems need to be able to capture these individual differences to produce speech that sounds like it was produced by a specific speaker. This requires modeling the characteristics of individual speakers and incorporating this information into the synthesis process.

Overall, addressing these issues will help to improve the quality and naturalness of TTS systems, making them more useful in a wide range of applications.

**Reference**

[Moulines & Charpentier 90] E.MOULINES, F. CHARPENTIER, "Pitch Synchronous waveform Processing techniques for Text-To-Speech Synthesis using diphones", Speech Communication, Vol. 9, n°5-6.

[O' Shaughnessy 84] D. O' SHAUGHNESSY, 'Design of a real-time French text-to-speech system', Speech Communication, Vol 3, pp. 233-243.

[O'Shaughnessy 90] D. O'SHAUGHNESSY, "Relationships between syntax and prosody for speech synthesis", Proceedings of the ESCA tutorial day on speech synthesis, Autrans, 1990, pp. 39-42. [Sproat et al. 92] R. SPROAT, J. HIRSHBERG, D. YAROWSKY, "A Corpus-based Synthesizer", Proc. ICSLP 92 Alberta, pp. 563-566.

[Stevens 90] K.N. STEVENS, 'Control parameters for synthesis by rule', Proceedings of the ESCA tutorial day on speech synthesis, Autrans, 25 sept 90, pp. 27-37.

[Traber 93] C. TRABER, "Syntactic Processing and Prosody Control in the SVOX TTS System for German", Proc. Eurospeech 93, Berlin, vol. 3, pp. 2099-2102.

[Willemse & Gulikers 92] R. WILLEMSE, L. GULIKERS, "Word class assignment in a Tes-ToSpeech system", Proc. Int. Conf. on Spoken Language Processing, Alberta, 1992, pp. 105-108.

[Withgott & Chen 93] M. M. WITHGOTT, F.R. CHEN, Computational models of American English, CSLI Lecture Notes, n°32, 143pp. [Witten 82] I.H. WITTEN, Principles of Computer Speech, Academic Press, 1992, 286 pp.

[Yarowsky 94] D. YAROWSKY, "Homograph Disambiguation in Speech Synthesis'', Proceedings, 2nd ESCA/IEEE Workshop on Speech Synthesis, New Paltz, NY, 1994.