UNIWERSYTET IM. ADAMA MICKIEWICZA

WYDZIAŁ MATEMATYKI I INFORMATYKI

Krystian Sołtysik

Nr albumu: 362680

Wykorzystanie otwartych narzędzi rozpoznawania mowy do tworzenia kodu źródłowego

Implementation of open-source tools for speech recognition in program coding

Praca magisterska na kierunku:

INFORMATYKA

Specjalność:

OGÓLNA

Promotor:

prof. UAM dr hab. Krzysztof Jassem

Poznań 2016

OŚWIADCZENIE

Ja, niżej podpisany Krystian Sołtysik, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt. *Wykorzystanie otwartych narzędzi rozpoznawania mowy do tworzenia kodu źródłowego* napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom, ani nie odpisywałem tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej.

Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

Wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM oraz na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich.

……..

(podpis)

STRESZCZENIE

Niniejsza praca opisuje proces projektowania, implementacji oraz analizę wyników systemu głosowego tworzenia kodu źródłowego w języku Python.

Przedstawiono w niej przegląd istniejących rozwiązań służących do programowania przy pomocy głosu, omówiono podstawy teoretyczne systemów rozpoznawania mowy, a także opisano architekturę i sposób użycia otwartego narzędzia Sphinx4, o które oparty jest system autorski.

Uzyskane w ramach pracy wyniki zostały szczegółowo omówione, przedstawiono wpływ wykorzystanego modelu akustycznego i modelu języka na jakość działania systemu autorskiego.

Powstały system, działając w oparciu o model akustyczny zaadaptowany do konkretnego użytkownika pozwala na poprawne rozpoznanie około 84% dyktowanych przez niego linii kodu.

ABSTRACT

This thesis describes the process of design, implementation and analysis of the voice system of creating source code in Python.

It presents an overview of existing solutions for programming using human voice. Moreover, it describes the theoretical basis of speech recognition systems, and also depicts the architecture and the way to use open source tools such as Sphinx4, which the author's program was structured upon.

The final results were discussed in detail in the course of work and the effects of acoustic and language models were presented with a link to the quality of the author's program.

The final system, which uses the acoustic model adapted to the specific user, produces the correct diagnosis of approximately 84% of dictated code lines.

Spis treści

1. Wstęp

Programowanie jest obecnie czynnością wymagającą wprowadzania dużej ilości tekstu przy pomocy klawiatury, co nie tylko wymaga od użytkownika znacznych umiejętności manualnych, ale też wiąże się ze znacznym wysiłkiem, który może doprowadzić do przeciążenia mięśni i ścięgien, a w konsekwencji – do urazów. Dlatego też zasadne wydaje się poszukiwanie alternatywnych metod tworzenia kodu źródłowego. Niniejsza praca poświęcona jest programowaniu za pośrednictwem mowy.

Celem niniejszej pracy jest zbadanie, czy jest możliwe, w oparciu o otwarte narzędzia rozpoznawania mowy, stworzenie systemu umożliwiającego głosowe tworzenie kodu źródłowego w sposób efektywny. Praca opisuje proces projektowania, implementacji i ewaluacji autorskiego systemu dedykowanego językowi Python, opartego o framework Sphinx4.

Praca składa się z kilku rozdziałów, stopniowo wprowadzających czytelnika w omawiane zagadnienia. Rozdział drugi poświęcony jest teoretycznym podstawom systemów rozpoznawania mowy. Szczegółowe omówienie jednego z takich systemów znajduje się w rozdziale trzecim. Kolejny rozdział opisuje specyfikę tworzenia kodu źródłowego w kontekście systemu rozpoznawania mowy. Rozdział piąty poświęcony jest procesowi tworzenia rozwiązania autorskiego, którego jakość i perspektywy dalszego rozwoju opisano w rozdziale szóstym.

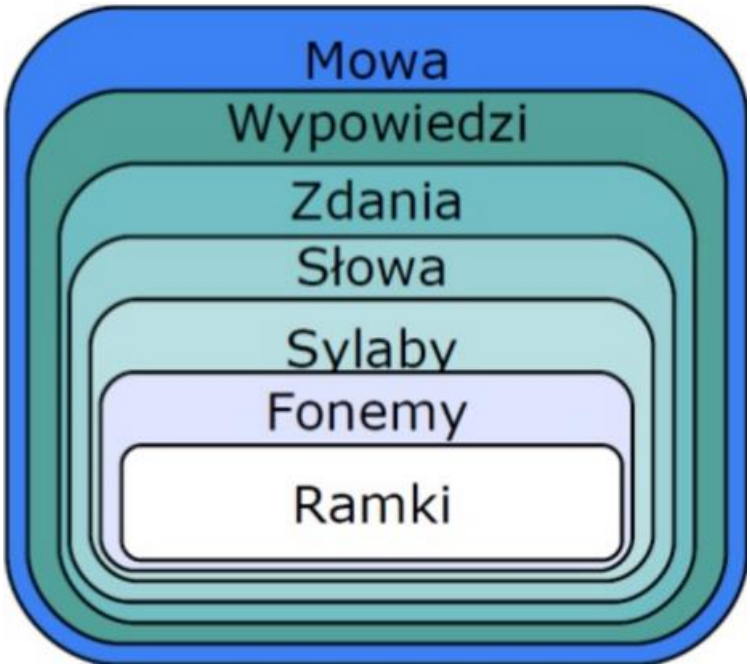
2. Podstawy teoretyczne systemów rozpoznawania mowy

2.1. Pojęcie mowy

Definicja 2.1

Mową nazywamy używanie języka w procesie porozumiewania się, czyli konkretne akty użycia systemu językowego (złożonego ze znaków i reguł).

Rysunek 2.1 ilustruje strukturę mowy, od całości wypowiedzi, poprzez zdania, słowa i sylaby, aż po ramki fonemów.



Rys 2.1. Struktura mowy. Źródło: [14]

Definicja 2.2

Widmem akustycznym nazywamy rozkład natężenia składowych dźwięku.

Wśród metod wytwarzania mowy wyróżnić można artykulację (mowę naturalną), odtwarzanie (modę rekonstruowaną) i generację (mowę syntezowaną).

Artykulacja odbywa się przy wykorzystaniu traktu głosowego, w którego skład wchodzą: płuca, dostarczające powietrza do procesu artykulacji, oskrzela i tchawica, które prowadzą strumień powietrza do krtani zawierającej struny głosowe będące źródłem dźwięku dla dźwięcznych fragmentów mowy. Głoski nosowe emitowane są przez jamę nosową i nozdrza. Modelowanie dźwięku odbywa się we wnękach

rezonansowych tworzonych przez podniebienie, język, wargi i zęby. Ważną rolę w procesie formowania tych wnęk pełnią ruchy żuchwy i policzków. Rezonanse kształtujące dźwięk powstają zarówno w wymienionych wnękach, ale też w tchawicy, krtani i klatce piersiowej.

Tonem podstawowym (krtaniowym) nazywamy dźwięk powstały w wyniku drgań strun głosowych pobudzonych przez przepływ powietrza. Widmo głoski dźwięcznej powstaje jako nałożenie charakterystyki traktu głosowego na widmo tonu podstawowego.

Ton podstawowy zmienia swą częstotliwość, wpływając na intonację wypowiedzi. Zakres tych zmian zależny jest od płci, wieku i indywidualnych cech osobniczych.

Definicja 2.3

Fonemem nazywamy najmniejszą jednostkę mowy rozróżnialną dla użytkowników danego języka.

Definicja 2.4

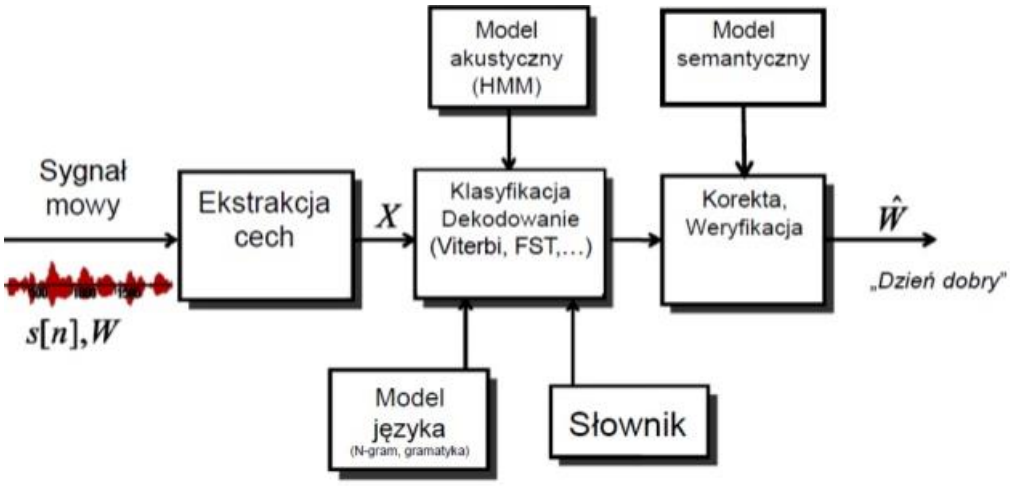
Segmentacją nazywamy podział sygnału mowy na krótkie fragmenty. Stosowane są dwa podejścia – ramkowanie, czyli podział na segmenty o stałej długości, zwykle 23,2 ms i podział na fragmenty reprezentujące fonem lub jednostkę. Ze względu na niską skuteczność algorytmów umożliwiających podział sygnału na jednostki mowy, w rzeczywistych systemach zwykle wykorzystywane jest ramkowanie.

Definicja 2.5

Parametryzacją mowy nazywamy reprezentację jej widma w sposób, który ma przedstawiać najistotniejsze informacje na temat opisywanej mowy w sposób możliwie efektywny. Najpopularniejsze metody parametryzacji to cepstralne melowe współczynniki częstotliwościowe (MFCC) [1] i percepcyjna predykcja liniowa (PLP) [2].

2.2. Elementy systemu ASR

System rozpoznawania mowy złożony jest z wielu współdziałających ze sobą elementów, takich jak model akustyczny, słownik i model języka. Elementy te zostały opisane w kolejnych podrozdziałach. Na rysunku 2.1 widoczny jest schemat systemu rozpoznawania mowy.



Rys 2.2. Schemat systemu rozpoznawania mowy. Źródło: [14]

2.2.1. Model akustyczny

Załóżmy, że sygnał mowy zawiera nieznaną wiadomość, możliwą do zapisania za pomocą ciągu symboli, a celem jest odczytanie tej wiadomości. Fragmenty (ramki) sygnału zostają przetworzone na wektor parametrów. Proces rozpoznawania polega na powiązaniu ciągu parametrów i nieznanych symboli zakodowanych w procesie mówienia. Odwzorowanie to nie jest wzajemnie jednoznaczne, gdyż ta sama wiadomość może zostać wymówiona w różny sposób. Nie jest też możliwe precyzyjne wyznaczenie granic między głoskami, sylabami czy wyrazami, co wynika ze zjawiska koartykulacji (narządy mowy rozpoczynają wymowę kolejnej głoski przed zakończeniem wymawiania poprzedniej).

Niech fonem będzie sekwencją segmentów, zwanych stanami $f_1, \dots, f_t, \dots, f_T$ o zadanej długości T. Przejście ze stanu i do stanu j ma charakter probabilistyczny i zależy od dyskretnego prawdopodobieństwa

$$a_{ij} = p(f_i|f_j),$$

(Wzór 2.1)

gdzie $\sum_{j=1}^{T} a_{ij} = 1$ dla każdego i.

Na podstawie wzoru 2.1 można utworzyć niesymetryczną macierz prawdopodobieństwa przejść między stanami (wzór 2.2).

$$A = [a_{ij}]$$

(Wzór 2.2)

Procesem Markowa nazywamy ciąg zdarzeń, w którym prawdopodobieństwo każdego zdarzenia zależne jest wyłącznie od wyniku zdarzenia poprzedniego. Własność ta została przedstawiona we wzorze 2.3.

$$p(f_i|f_1, \dots, f_{i-1}) = p(f_i|f_{i-1})$$

(Wzór 2.3)

Do modelowania procesu Markowa można wykorzystać Ukryty Model Markowa (HMM) - statystyczny model Markowa, w którym stany są niewidoczne (ukryte) dla obserwatora.

Obserwacje są matematyczną reprezentacją jednostek akustycznych, powstałych w wyniku segmentacji i parametryzacji sygnału mowy. Dla każdej ramki dobierany jest najodpowiedniejszy wektor obserwacji (wzór 2.4).

$$o_t \in \{o_j\}_{j=1}^{V}$$

(Wzór 2.4)

Niech $b_i(o_j)$ będzie prawdopodobieństwem obserwacji o_j wygenerowanym przez stan f_i. Na podstawie tych prawdopodobieństw tworzona jest macierz prawdopodobieństw obserwacji (wzór 2.5).

$$B = [b_i(o_j)]$$

(Wzór 2.5)

Prawdopodobieństwo obserwacji o_t zależy wyłącznie od obecnego stanu f_i, nie jest zależne od wcześniejszych stanów czy obserwacji. Własność ta przedstawiona jest we wzorze 2.6.

$$p(o_t|f_1, \dots, f_i, o_1, \dots, o_{t-1}) = p(o_t|f_t)$$

(Wzór 2.6)

Ukryty model Markowa definiowany jest przez macierze A i B.

Dzięki zastosowaniu Ukrytego Modelu Markowa możliwe jest wygenerowanie na podstawie obserwacji i stanów macierzy A i B. Macierze te pozwalają wyznaczyć prawdopodobieństwo sekwencji obserwacji O w oparciu o statystyki, jak i znaleźć najbardziej prawdopodobną sekwencję stanów przy ustalonej obserwacji (rozpoznać fonem).

2.2.2. Słownik

Słownik to element systemu rozpoznawania mowy, który zawiera słowa rozpoznawane przez system oraz ich zapis fonetyczny. Słownik ten wykorzystywany jest zarówno w procesie tworzenia czy adaptacji modelu akustycznego, jak i w procesie rozpoznawania mowy. Jego jakość ma istotny wpływ na dokładność

działania systemu – błędna transkrypcja fonetyczna może sprawić, że tworzony model akustyczny przypisze niepoprawne obserwacje pewnym fonemom.

Ze względu na niejednoznaczność reprezentacji fonetycznej wyrazu, konieczne jest uwzględnienie wszystkich sposobów wymowy słowa. Może mieć to szczególne znaczenie, gdy grono odbiorców systemu nie jest ograniczone do pojedynczego użytkownika.

Definicja 2.7:

ARPAbet to alfabet transkrypcji fonetycznej stworzony przez ARPA (Advanced Research Projects Agency), będący standardowym sposobem zapisu fonetycznego języka angielskiego. Składa się on z 39 fonemów, reprezentowanych przez jedną lub dwie wielkie litery. Akcenty wyrazowe opisywane są przy pomocy cyfr (0 – brak akcentu, 1 – akcent główny, 2 – akcent poboczny), które są umieszczone przy samogłoskach.

Listing 2.1 zawiera kompletną listę fonemów alfabetu ARPAbet (w pierwszej kolumnie) wraz z przykładowymi słowami je zawierającymi (w kolumnie drugiej) i ich transkrypcją (w trzeciej kolumnie).

```
AA          odd          AA D
AE          at           AE T
AH          hut          HH AH T
AO          ought        AO T
AW          cow          K AW
AY          hide         HH AY D
B           be           B IY
CH          cheese       CH IY Z
D           dee          D IY
DH          thee         DH IY
EH          Ed           EH D
ER          hurt         HH ER T
EY          ate          EY T
```

```
F           fee         F IY
G           green       G R IY N
HH          he          HH IY
IH          it          IH T
IY          eat         IY T
JH          gee         JH IY
K           key         K IY
L           lee         L IY
M           me          M IY
N           knee        N IY
NG          ping        P IH NG
OW          oat         OW T
OY          toy         T OY
P           pee         P IY
R           read        R IY D
S           sea         S IY
SH          she         SH IY
T           tea         T IY
TH          theta       TH EY T AH
UH          hood        HH UH D
UW          two         T UW
V           vee         V IY
W           we          W IY
Y           yield       Y IY L D
Z           zee         Z IY
ZH          seizure     S IY ZH ER
```

(Listing 2.1)

Przykładem słownika transkrypcji fonetycznych dla słów języka angielskiego jest CMUDict, wykorzystujący ARPAbet. Do celów rozpoznawania mowy wykorzystywana jest wersja o podobnym zestawie fonemów, pozbawiona jednak informacji o akcentach wyrazowych.

Listing 2.2 zawiera fragment słownika CMUDict w wersji kompatybilnej z systemem rozpoznawania mowy CMUSphinx. W pierwszej kolumnie znajduje się wyraz oraz ew. zawarty w nawiasach numer alternatywnej formy jego wymowy, w drugiej zaś znajduje się jego transkrypcja fonetyczna.

```
DICTATE       D IH K T EY T
DICTATED      D IH K T EY T AH D
DICTATED(2)   D IH K T EY T IH D
DICTATES      D IH K T EY T S
DICTATING     D IH K T EY T IH NG
DICTATION     D IH K T EY SH AH N
DICTATOR      D IH K T EY T ER
DICTATORIAL   D IH K T AH T AO R IY AH L
DICTATORS     D IH K T EY T ER Z
DICTATORSHIP  D IH K T EY T ER SH IH P
DICTATORSHIPS D IH K T EY T ER SH IH P S
DICTION       D IH K SH AH N
DICTIONARIES  D IH K SH AH N EH R IY Z
DICTIONARY    D IH K SH AH N EH R IY
```

(Listing 2.2)

2.2.3. Model języka

Model języka jest narzędziem pozwalającym na określenie prawdopodobieństwa wystąpienia sekwencji słów. Niniejsza praca opisuje statystyczny model n-gramowy, a także gramatykę JSGF.

Model n-gramowy

Model n-gramowy jest prostym modelem statystycznym pozwalającym przewidzieć kolejny element sekwencji. Tworzenie n-gramowego modelu języka wymaga zliczenia wystąpień ciągów wyrazów o ustalonej długości n w korpusie trenującym. Liczby wystąpień są następnie zamieniane na znormalizowane prawdopodobieństwa.

Przykład:

Liczymy prawdopodobieństwo wystąpienia sekwencji wyrazów „Ala ma kota".

Dla n=2 obliczenia wyglądają następująco:

P(‘Ala', ‘ma', ‘kota') = P(‘Ala'|'<s>')P(‘ma'|'Ala')P(‘kota'|'ma')P(‘</s>'|'kota'),

gdzie znaczniki <s> i </s> oznaczają ciszę przed rozpoczęciem i zakończeniem mówienia. P(‘kota'|'ma') równe jest ilorazowi liczby wystąpień wyrazu ‘kota' po wyrazie ‘ma' i liczby wystąpień wszystkich wyrazów po ‘ma'.

W rzeczywistych zastosowaniach stosowane są logarytmy prawdopodobieństw, a mnożenie zastąpione jest dodawaniem, co pozwala uniknąć problemu niedomiaru zmiennoprzecinkowego i poprawia wydajność obliczeń.

Zasadniczym problemem przy wykorzystaniu modelu n-gramowego są zerowe liczebności n-gramów niewystępujących w korpusie. Możliwą metodą obejścia problemu jest wykorzystanie n-gramów niższego rzędu, gdy statystyki rzędu wyższego nie zawierają wystarczających danych (ang. *back-off*). Wprowadza się także wygładzanie modelu, polegające na zmniejszeniu liczebności n-gramów występujących w korpusie, na rzecz tych, które w nim nie wystąpiły.

Listing 2.3 zawiera fragment standardowego n-gramowego modelu języka angielskiego CMUSphinx w formacie ARPA.

```
-2.9224 speech recognition -0.1358
-4.1782 speech represented -0.1666
-2.6424 speech right -0.0350
-2.6047 speech rights -0.0681
-3.5752 speech saddam -0.1088
-3.1153 speech saturday -0.0608
-2.8838 speech saying -0.0149
-3.3978 speech scheduled -0.2749
-2.6229 speech she -0.1155
-3.4314 speech short -0.1654
-2.8348 speech should -0.2405
-3.7040 speech simply -0.2141
-2.9546 speech since -0.0275
-2.5169 speech so -0.0065
-3.2838 speech sounds -0.0741
-2.8206 speech speech -0.0162
-3.4526 speech such -0.0489
-3.6757 speech sunday -0.1221
```

(Listing 2.3)

Pierwsza kolumna zawiera logarytmy prawdopodobieństwa 2-gramów wymienionych w drugiej kolumnie. Trzecia kolumna zawiera natomiast wartości prawdopodobieństw

użycia krótszego n-gramu do obliczania prawdopodobieństwa (wartość wagi *back-off*).

Gramatyka JSGF

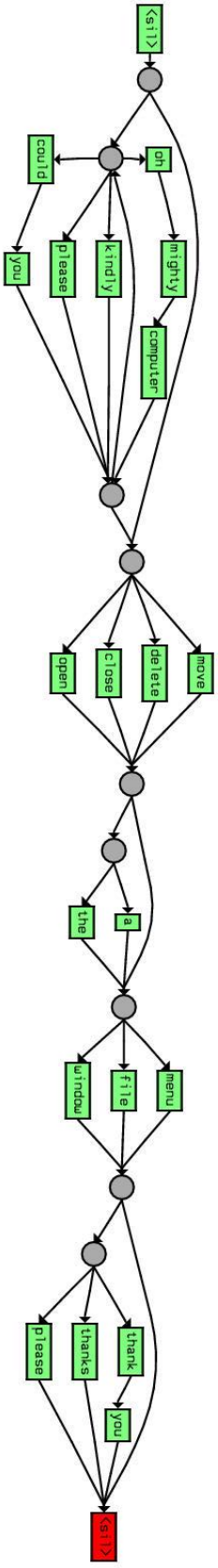
Gramatyka JSGF (ang. *Java Speech Grammar Format*) jest typem modelu języka kompatybilnym z systemem Sphinx. Nie jest to model statystyczny, służy do definiowania komend rozpoznawanych przez system. Każda akceptowana przez system komenda musi zostać precyzyjnie opisana.

Gramatyka JSGF nie jest więc dobrym narzędziem do tworzenia systemów akceptujących polecenia o skomplikowanej składni.

Listing 2.3 zawiera przykład prostej gramatyki służącej do kontrolowania komputera za pośrednictwem komend. Graf pozwalający odczytanie wszystkich akceptowanych przez gramatykę komend przedstawiono na rysunku 2.2.

```
public <basicCmd> = <startPolite> <command> <endPolite>;

<command> = <action> <object>;

<action> = /10/ open |/2/ close |/1/ delete |/1/ move;

<object> = [the | a] (window | file | menu);


<startPolite> = (please | kindly | could you | oh  mighty  computer)
*;

<endPolite> = [ please | thanks | thank you ];
```

(Listing 2.3)



Rys. 2.2. Graf przedstawiający gramatykę JSGF. Źródło:
http://cmusphinx.sourceforge.net/wiki/sphinx4:jsgfsupport

2.3. Wybór technologii

Na etapie planowania pracy nad systemem autorskim rozważane było wykorzystanie jednego z frameworków: Sphinx4, Pocketsphinx lub HTK.

HTK (ang. *Hmm ToolKit*) to oprogramowanie powstałe na uniwersytecie w Cambridge. Jest to zestaw narzędzi napisanych w C, przeznaczonych do różnych zadań, z których korzystanie nie jest łatwe. Licencja ogranicza komercyjne wykorzystanie systemu. Ostatnia wersja stabilna pochodzi z roku 2009, podczas gdy najnowsza wersja beta z 2015 roku. Dostęp do kodu źródłowego wymaga rejestracji.

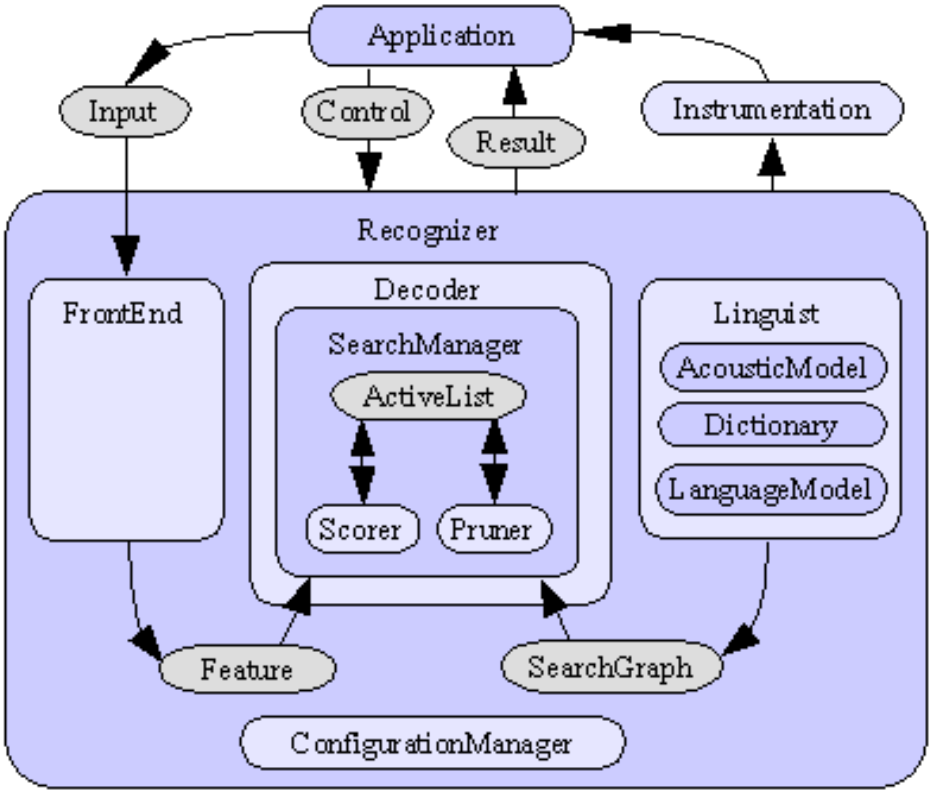
Sphinx4 i Pocketsphinx to narzędzia powstałe na Carnegie Mellon University, należące do zestawu narzędzi CMUSphinx. Sphinx4 jest napisanym w całości w Javie frameworkiem o dużych możliwościach, natomiast Pocketsphinx jest lekką biblioteką napisaną w C, przeznaczoną dla urządzeń mobilnych i wbudowanych. Wszystkie elementy zestawu narzędzi CMUSphinx udostępniane są na przyjaznej licencji w stylu BSD. Projekt jest aktywnie rozwijany, repozytorium kodu źródłowego jest publicznie dostępne, udostępniana jest też obszerna dokumentacja.

Ostatecznie, system autorski został oparty o narzędzie Sphinx4 – łatwość wykorzystania i jakość dokumentacji były ważniejsze niż ograniczone wymagania sprzętowe.

3. Framework Sphinx4

3.1. Elementy systemu

Sphinx4 jest frameworkiem służącym do rozpoznawania mowy w całości napisanym w języku Java. Ma on więc specyficzną dla języka obiektowego architekturę, której diagram przedstawiono na rysunku 3.1.



Rys 3.1. Architektura frameworka Sphinx4. Źródło: [11]

Na rysunku 3.1 widoczne są następujące moduły:

- Recognizer – zawiera główne komponenty frameworka takie jak FrontEnd, Linguist, Decoder. Interakcja pomiędzy aplikacją a frameworkiem Sphinx4 odbywa się głównie przy użyciu klasy Recognizer.
- FrontEnd – przetwarza sygnał wejściowy mowy
- Linguist – łączy model akustyczny, słownik i model języka
- Acoustic Model – reprezentuje model akustyczny

- Dictionary – jest to klasa odpowiedzialna za obsługę słownika
- Language Model – reprezentuje model języka, obsługuje m.in. modele n-gramowe i gramatyki
- Search Graph – stanowi grafową strukturę tworzoną przez moduł Linguist, na podstawie wiedzy ze słownika, modelu akustycznego i modelu języka
- Decoder – jest do główna klasa dekodera
- Search Manager – wykonuje wyszukiwania korzystając z algorytmów, m.in. przeszukiwania wszerz, przeszukiwania w głąb; zawiera moduły Scorer i Pruner.
- Active List – zawiera łańcuchy rozpoznań
- Feature – reprezentuje cechy sygnału mowy używane przez dekoder
- Scorer – tworzy prawdopodobne łańcuchy rozpoznań
- Pruner – usuwa mało prawdopodobne łańcuchy rozpoznań z Active List
- Result – reprezentuje wynik działania systemu
- Configuration Manager – zarządza konfiguracją systemu, ładuje dane konfiguracyjne i kontroluje cykl życia obiektów

3.2. Korzystanie z systemu

Korzystanie z frameworka Sphinx4 nie wymaga dużego doświadczenia programistycznego ani szerokiej wiedzy. Najprostszą metodą by rozpocząć pracę z frameworkiem jest utworzenie projektu wykorzystującego system budowania Maven w środowisku programistycznym takim jak Eclipse czy NetBeans. Następnie do pliku pom.xml dodajemy repozytorium (listing 3.1).

```
<project>
...
    <repositories>
        <repository>
            <id>snapshots-repo</id>
```

```
                <url>https://oss.sonatype.org/content/repositories/s
                napshots</url>
            <releases><enabled>false</enabled></releases>
        <snapshots><enabled>true</enabled></snapshots>
        </repository>
    </repositories>
...
</project>
```

(Listing 3.1)

Kolejnym krokiem jest dodanie zależności do projektu. Listing 3.2 zawiera fragment pliku pom.xml odpowiedzialny za dodanie do projektu frameworka Sphinx, standardowego słownika języka angielskiego, a także modelu akustycznego i języka.

```
  <dependencies>
   …
   <dependency>
  <groupId>edu.cmu.sphinx</groupId>
  <artifactId>sphinx4-core</artifactId>
  <version>5prealpha-SNAPSHOT</version>
</dependency>

<dependency>
  <groupId>edu.cmu.sphinx</groupId>
  <artifactId>sphinx4-data</artifactId>
  <version>5prealpha-SNAPSHOT</version>
```

```
</dependency>

…

  </dependencies>
```

(Listing 3.2)

Od tej chwili możemy korzystać w projekcie z frameworka Sphinx4. Listing 3.3 zawiera kod źródłowy odpowiedzialny za utworzenie obiektu rozpoznającego mowę nagrywaną przez mikrofon, wykorzystującego domyślne modele i słownik i wypisywanie w nieskończonej pętli rozpoznanych słów.

```
Configuration configuration = new Configuration();

configuration.setAcousticModelPath("resource:/edu/cmu/sphinx/models/
en-us/en-us");

configuration.setDictionaryPath("resource:/edu/cmu/sphinx/models/en-
us/cmudict-en-us.dict");

configuration.setLanguageModelPath("resource:/edu/cmu/sphinx/models/
en-us/en-us.lm.bin");

LiveSpeechRecognizer recognizer = new
LiveSpeechRecognizer(configuration);

recognizer.startRecognition(true);

while (true){

String utterance = rcognizer.getResult().getHypothesis();

          System.out.println(utterance);

}
```

(Listing 3.3)

Oprócz klasy LiveSpeechRecognizer, odpowiedzialnej za rozpoznawanie mowy z mikrofonu, dostępna jest także klasa StreamSpeechRecognizer korzystająca z pliku dźwiękowego lub np. gniazda sieciowego, a także klasa SpeechAligner pozwalająca na dopasowywanie w czasie tekstu z mową.

4. Specyfika tworzenia kodu źródłowego w kontekście systemu rozpoznawania mowy

4.1. Sposoby wykorzystania systemu rozpoznawania mowy do tworzenia kodu źródłowego

Obecnie głównymi narzędziami wykorzystywanymi do tworzenia kodu źródłowego są klawiatura i mysz. Pomimo coraz bardziej zaawansowanych możliwości nowoczesnych zintegrowanych środowisk programistycznych, programowanie wciąż wymaga od użytkownika wpisywania znacznej ilości tekstu przy pomocy klawiatury, co wiąże się ze znacznym wysiłkiem oraz wymaga od użytkownika pewnych umiejętności manualnych.

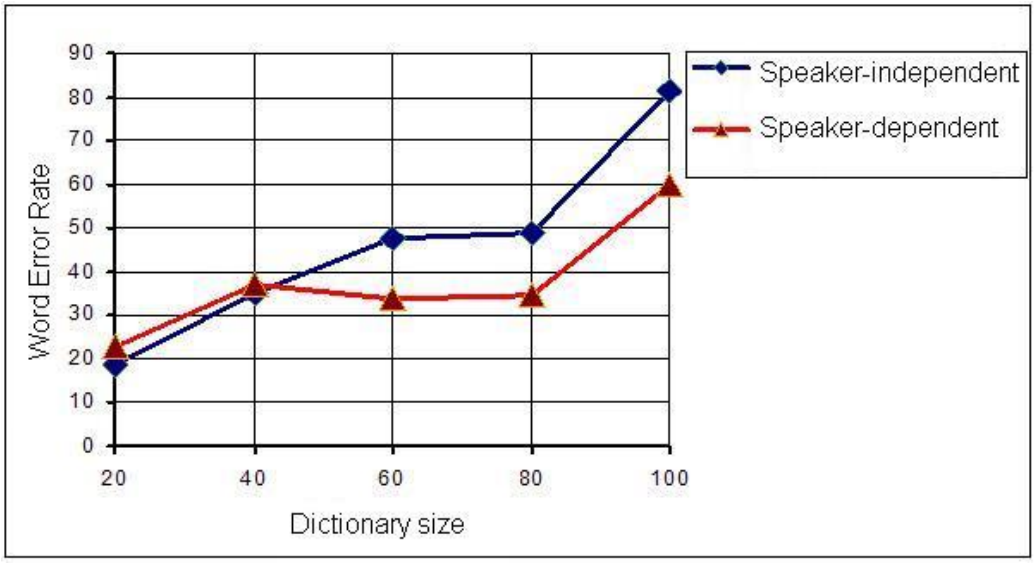
Kompletne rozwiązanie, eliminujące konieczność wykorzystania klawiatury i myszki, wymagałoby stworzenia środowiska programistycznego pozwalającego nie tylko na dyktowanie kodu źródłowego, głosową nawigację po pliku tekstowym i projekcie, ale też obsługę takich funkcji przyspieszających programowanie, jak np. inteligentne uzupełnianie kodu.

Tworzenie tak skomplikowanego systemu może jednak nie mieć uzasadnienia, biorąc pod uwagę niewielkie grono odbiorców takiego rozwiązania. Liczba błędów popełnianych przez systemy rozpoznawania mowy znacznie przewyższa liczbę błędów popełnianych w trakcie pisania na klawiaturze. Należy również pamiętać, iż dyktowanie znacznej ilości tekstu również wiąże się z wysiłkiem, a dźwięki wydawane przez użytkownika programującego przy pomocy głosu mogą być trudne do zaakceptowania dla innych osób w biurze. Trudno więc oczekiwać, by osoby, które nie mają zdrowotnych przeciwskazań do pisania na klawiaturze, były zainteresowane wykorzystywaniem na co dzień rozwiązania tego typu.

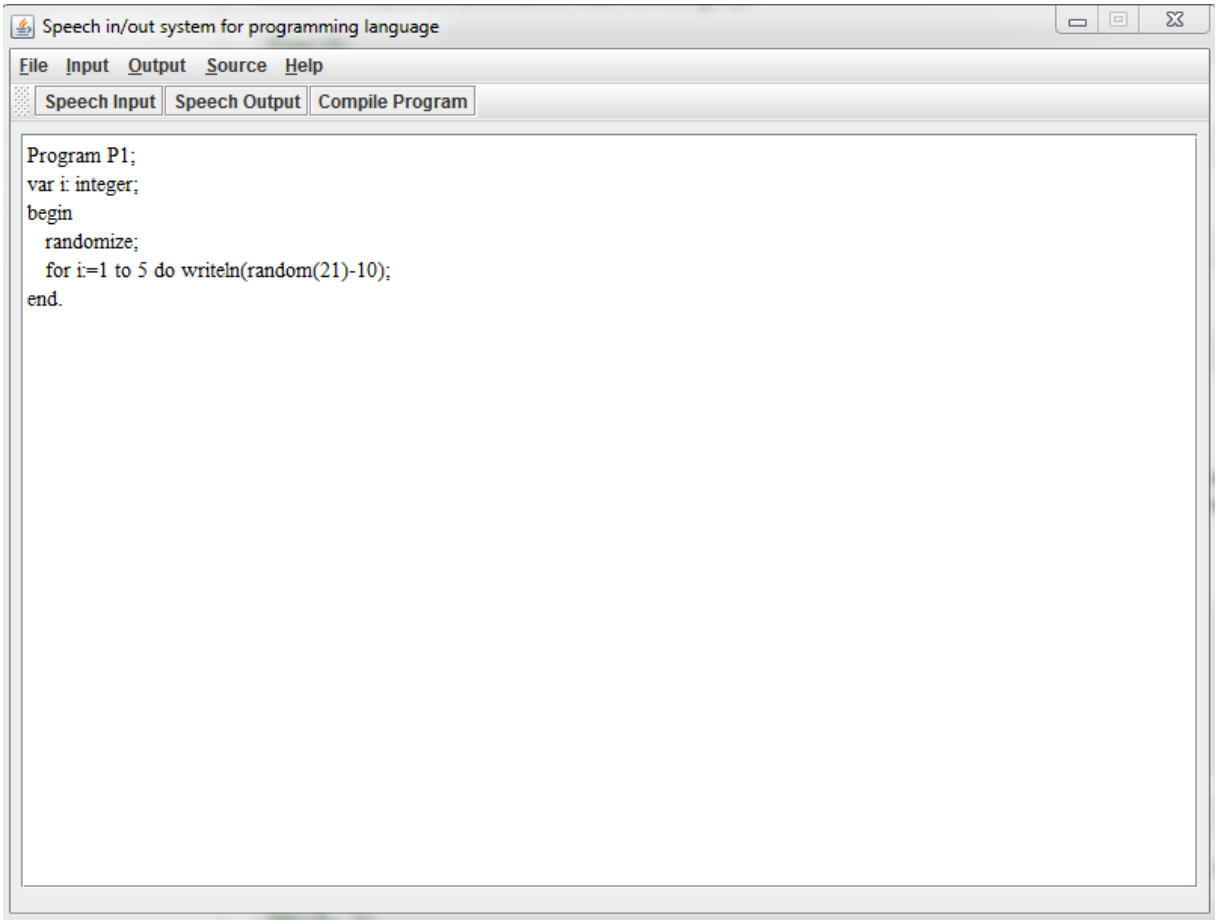
Z tych powodów istniejące systemy umożliwiające głosowe tworzenie kodu są stosunkowo proste, a ich funkcjonalność – ograniczona.

4.2. Istniejące rozwiązania

Rozwiązaniami najbardziej zbliżonymi do systemu autorskiego są projekty stworzone przez studentów Donieckiego Narodowego Uniwersytetu Technicznego – Darię Savkovą i Valery'ego Bakelenko. Z informacji podanych na stronie internetowej uczelni można wywnioskować, iż powstałe aplikacje (niedostępne publicznie) oparte są o framework Sphinx4 i służą do dyktowania kodu w języku Pascal, a dokładność ich działania jest bardzo niska. Rysunek 4.1 przedstawia zależność WER od rozmiaru słownika w projekcie autorstwa Darii Savkovej. Rysunek 4.2 przedstawia interfejs aplikacji Valery'ego Bakelenko.



Rys 4.1. Zależność WER od rozmiaru słownika w projekcie Darii Savkovej.



Rys 4.2. Interfejs aplikacji autorstwa Valery'ego Bakelenko. Widoczny jest kod źródłowy w języku Pascal.

Kolejnym rozwiązaniem służącym do wspomagania tworzenia kodu źródłowego przy pomocy głosu jest plugin *idear* do środowiska IntelliJ IDEA, oparty o Sphinx4. Obsługuje on niektóre funkcje środowiska, pozwalając m.in. na głosową nawigację po pliku tekstowym czy obsługę debugowania.

Istnieją również narzędzia i makra oparte o komercyjny system rozpoznawania mowy Dragon NaturallySpeaking, takie jak nierozwijany od wielu lat VoiceCode stworzony przez National Research Council Canada. Ze względu na wysoką dokładność zamkniętego rozwiązania do rozpoznawania mowy, narzędzia te prezentują wystarczającą jakość dla niektórych użytkowników.

4.3. Wymagania wobec systemu

Użyteczna aplikacja pozwalająca na głosowe tworzenie kodu źródłowego powinna minimalizować konieczność użycia urządzeń wskazujących. Polecenia dyktowane przez użytkownika muszą być krótkie, by ograniczyć wysiłek użytkownika i zwiększyć wydajność wprowadzania. Powinny być też one również możliwie oczywiste, pozwalając użytkownikowi skupić się na programowaniu. Reakcja systemu na polecania głosowe musi następować szybko. Dokładność działania systemu powinna być możliwie najwyższa, a negatywne skutki błędnego zrozumienia polecenia – ograniczone.

4.4. Czynniki wpływające na jakość działania systemu

Głównym czynnikiem wpływającym na jakość działania systemu jest głos użytkownika – nie należy się spodziewać wysokiej jakości, jeśli model akustyczny nie zostanie zaadaptowany do głosu konkretnej osoby. Inne czynniki to m.in. jakość mikrofonu, szumy i dźwięki tła (np. rozmowy w biurze), a także zgodność wprowadzanego przez użytkownika kodu z modelem języka.

5. Projektowanie i ewaluacja systemu autorskiego

Celem projektu było stworzenie systemu umożliwiającego głosowe tworzenie kodu źródłowego w języku Python, opartego o otwarte narzędzie do rozpoznawania mowy – framework Sphinx4.

Najważniejszym etapem pracy nad projektem było ustalenie takiego sposobu dyktowania kodu źródłowego przez użytkownika, by skuteczność działania mechanizmu rozpoznawania mowy była możliwie najwyższa, a jednocześnie by sposób ten był możliwie oczywisty w działaniu i naturalny dla człowieka. Szczegółowe informacje na temat tego zagadnienia są opisane w podrozdziale 5.1, a model języka jest opisany w części 5.3. Podrozdział 5.2 poświęcony jest natomiast metodom ewaluacji systemu autorskiego.

Część 5.4 zawiera opis i wyniki prac nad adaptacją modelu akustycznego do konkretnego mówcy, a podrozdział 5.5 opisuje interfejs aplikacji.

5.1. Metoda dyktowania

Prace nad ustaleniem metody dyktowania kodu źródłowego przez użytkownika wymagały określenia elementów kodu źródłowego, które będą przez system rozpoznawane, a także tych, które będą wprowadzane przy użyciu klawiatury. Przykładowo, komentarze nie są elementem obsługiwanym przez moduł rozpoznawania mowy, ponieważ, biorąc pod uwagę konieczny rozmiar słownika pozwalającego na napisanie dowolnego tekstu w języku angielskim, nie można by było oczekiwać, aby skuteczność rozpoznawania mowy przy wykorzystaniu frameworka Sphinx4 była satysfakcjonująca.

Najważniejszymi elementami wspieranymi przez system są słowa kluczowe, operatory, liczby, separatory, nazwy zmiennych, znaki (tabulatora, nowej linii). Niezbędne było również zdefiniowanie komend informujących system, czy dyktowane polecenie zostało rozpoznane poprawnie, czy też konieczne jest jego powtórzenie.

Choć słowa kluczowe języka Python są, w większości, słowami języka angielskiego, zamiana niektórych z nich na ich odpowiedniki, łatwiejsze do rozpoznania, znacząco podnosi jakość działania systemu.

Listing 5.1 zawiera fragment słownika wstępnej wersji systemu autorskiego. Widać na nim, że jedna z transkrypcji fonetycznych słowa kluczowego *for* jest identyczna z transkrypcją słowa *four*. Istnieje więc duże ryzyko błędnego rozpoznawania tych słów. Dlatego też, w dalszych pracach nad systemem, słowo *for* zostało zastąpione słowem *foreach*.

```
FOR         F AO R
FOR(2)      F ER
FOR(3)      F R ER
FOUR        F AO R
```

(Listing 5.1)

Tabela 5.1 zawiera listę słów kluczowych języka Python, które w systemie zostały zastąpione łatwiejszymi do rozpoznania odpowiednikami.

Słowo kluczowe języka Python	Słowo lub sekwencja słów dyktowana przez użytkownika
def	define
del	delete
elif	else if
for	foreach

(Tabela 5.1)

Słowa kluczowe znajdujące się na listingu 5.2 użytkownik wymawia w sposób standardowy.

```
and

as

assert
```

```
break
class
continue
else
except
exec
finally
from
global
if
import
in
is
lambda
not
or
pass
print
raise
return
try
while
with
yield
```

(Listing 5.2)

Ponieważ nazwy zmiennych mogą być ciągiem dowolnych, niekoniecznie istniejących w jakimkolwiek języku, słów, w których wielkość liter ma znaczenie, wymawianie ich byłoby dla użytkownika zajęciem uciążliwym, a skuteczność ich rozpoznawania przez system byłaby niska. Dlatego też sposób wprowadzania zmiennych do kodu źródłowego w systemie autorskim wymaga uprzedniego wpisania ich przy użyciu klawiatury do odpowiedniego pola tekstowego w programie. Następnie, dyktując linię kodu źródłowego, użytkownik dyktuje nazwę pola tekstowego zawierającego nazwę zmiennej.

Definicja 5.1

Alfabet fonetyczny ICAO znany również jako Alfabet fonetyczny NATO, a także jako Międzynarodowy alfabet fonetyczny, to najbardziej rozpowszechniony system literowania wyrazów w pewnych specyficznych zastosowaniach, np. w lotnictwie podczas prowadzenia komunikacji radiowej, gdzie każdej literze alfabetu odpowiada ustalone słowo.

Podstawową zaletą tego kodowania jest jego prostota, jednoznaczność i odporność na zakłócenia. Ponieważ żadna z sylab tworzących poszczególne słowa kodu nie powtarza się, nawet fragmentaryczna transmisja może być zrozumiana.

Nazwy pól tekstowych w opisywanym projekcie należą do zbioru słów kodowych alfabetu fonetycznego ICAO, co redukuje liczbę błędów rozpoznawania mowy.

Listing 5.3 zawiera linie słownika systemu autorskiego odpowiedzialne za rozpoznawanie nazw pól tekstowych.

```
ALPHA      AE L F AH
BRAVO      B R AA V OW
CHARLIE    CH AA R L IY
DELTA      D EH L T AH
ECHO       EH K OW
```

```
FOXTROTT   F AA K S T R AH T
GOLF       G AA L F
GOLF(2)    G AO L F
HOTEL      HH OW T EH L
JULIETT    JH UW L IY T
```

(Listing 5.3)

W sposób analogiczny do nazw zmiennych obsługiwane są także literały łańcuchowe oraz dekoratory. Interfejs użytkownika zawiera jedno pole o nazwie *string* i jedno pole o nazwie *decorator*. Zawartość pól tekstowych wykorzystywana jest wyłącznie w momencie wprowadzania rozpoznanej linii kodu do pola edycyjnego, późniejsza modyfikacja zawartości pola nie wprowadza zmian w kodzie źródłowym. Limitowana jest więc liczba różnych od siebie zmiennych, literałów łańcuchowych i dekoratorów które można wprowadzić w danym momencie (w danej linii kodu) ale nie w całym kodzie źródłowym programu.

Dyktowanie liczb polega na podyktowaniu ciągu cyfr wraz z ewentualnym separatorem dziesiętnym (kropką) i znakiem.

Listing 5.4 zawiera listę słów potrzebnych do głosowego wprowadzania liczb. Bezpośrednie wprowadzanie liczb w formacie niestandardowym, np. zapisanych w postaci szesnastkowej lub w notacji naukowej, nie jest możliwe. Jeśli jednak występuje konieczność wielokrotnego użycia takiej liczby, można wpisać ją przy użyciu klawiatury w wybrane pole tekstowe przeznaczone dla nazw zmiennych i używać jej w sposób analogiczny do nazw zmiennych.

```
DOT        D AA T
ONE        W AH N
ONE(2)     HH W AH N
TWO        T UW
THREE      TH R IY
FOUR       F AO R
FIVE       F AY V
```

```
SIX         S IH K S
SEVEN       S EH V AH N
EIGHT       EY T
NINE        N AY N
ZERO        Z IY R OW
MINUS       M AY N AH S
```

(Listing 5.4)

Wprowadzanie operatorów i separatorów sprowadza się do podyktowania odpowiedniego słowa lub sekwencji słów. Nazwy nadane niektórym elementom (np. nawiasom kwadratowym) mogą wydawać się nieintuicyjne, jednak ze względu na wysoką częstość występowania tych elementów w kodzie źródłowym, konieczne było nadanie im krótkich, jednowyrazowych nazw.

Tabela 5.2 zawiera kompletną listę separatorów i operatorów występujących w języku Python wraz ze sposobem ich wprowadzania.

Separator lub operator	Słowo wypowiadane przez użytkownika
+	plus
-	minus
*	star
**	star star
/	slash
//	slash slash
%	percent
<<	less less
>>	greater greater
&	ampersand
|	bar
^	caret
~	tilde
<	less

>	greater
<=	less assign
>=	greater assign
==	equals
!=	unequal
(	open
)	close
[	left
]	right
{	first
}	second
,	comma
:	colon
.	dot
;	semicolon
@	at
=	assing
->	minus greater
+=	add
-=	subtract
*=	multiply
/=	divide
//=	slash divide
%=	percent assign
@=	at assign
&=	ampersand assign
|=	bar assign
^=	caret assign
>>=	greater greater assign
<<=	less less assign
**=	star multiply

(Tabela 5.2)

Listing 5.5 zawiera listę komend obsługiwanych przez system. Wypowiadając polecenie *ok*, użytkownik potwierdza poprawność rozpoznanego wcześniej ciągu elementów języka Python, a system umieszcza ten ciąg w polu edycyjnym, w miejscu wskazywanym przez kursor oraz dodaje znak nowej linii. W szczególności, gdy rozpoznany ciąg jest pusty, efektem wykonania polecenia jest utworzenie pustej linii.
Skutkiem komendy *indent* jest wprowadzenie znaku tabulacji w pole edycyjne i przesunięcie kursora.
Wypowiadając polecenie *line* użytkownik dodaje znak nowej linii i przemieszcza do niej kursor.

```
OK                OW K EY
WRONG             R AO NG
LINE              L AY N
```

(Listing 5.5)

5.2. Testy systemu

Ewaluacja systemu rozpoznawania mowy jest konieczna na każdym etapie jego tworzenia. Dzięki testom możliwe jest ustalenie skutków zmian wprowadzonych w systemie, co pozwala na podjęcie decyzji dotyczących dalszego jego rozwoju. Niezwykle istotne jest więc, by testy były prowadzone w sposób wiarygodny, jak najlepiej odzwierciedlając rzeczywiste warunki pracy systemu.

Pierwszym etapem pracy nad korpusem testowym było ustalenie takich sekwencji testowych, by w jak największym stopniu przypominały one rzeczywiste dane dyktowane przez użytkownika. Każda spośród spisanych sekwencji była przykładową linią kodu źródłowego lub komendą dla systemu. Sekwencje te zostały następnie nagrane przez tego samego użytkownika, w takich samych warunkach i z użyciem tego samego sprzętu.

Metryki

Jednymi z najczęściej stosowanych metryk służących do ewaluacji systemów rozpoznawania mowy są Accuracy, WER (Word Error Rate) oraz SER (Sentence Error Rate).

Metryka Accuracy obliczana jest wg wzoru 5.1.

$$\text{Accuracy} = \frac{N-D-S}{N} * 100\% \tag{5.1}$$

gdzie:
N - liczba słów
D - liczba usunięć
S - liczba zastąpień

Metryka ta określa procent słów rozpoznanych poprawnie. Nie uwzględnia ona wyrazów wstawionych.

Przykład:

Użytkownik podyktował systemowi ciąg słów *‘print open alpha close’* który został rozpoznany jako *‘print open bravo close’*. W takiej sytuacji: N = 4, D = 0, S = 1.

A zatem:

$$\text{Accuracy} = \frac{4-0-1}{4} * 100\% = 75\%$$

WER (Word Error Rate) jest metryką pozwalającą określić, jaka część słów została rozpoznana w sposób błędny. Metryka WER obliczana jest wg wzoru 5.2.

$$\text{WER} = \frac{I+D+S}{N} * 100\% \tag{5.2}$$

gdzie:

N - liczba słów

I - liczba słów wstawionych

D - liczba słów usuniętych

S - liczba słów zastąpionych

Przykład:

Użytkownik podyktował ciąg słów *'charlie assign true'*, który został rozpoznany jako *'if charlie assign true'*.

W takiej sytuacji: N = 3, I = 1, D = 0, S = 0.

A zatem:

$$\text{WER} = \frac{1+0+0}{3} * 100\% = 33,(3)\%$$

Kolejną metryką jest SER (Sentence Error Rate), określająca, jaka część zdań (a w przypadku systemu autorskiego – linii kodu) została rozpoznana w sposób błędny. Obliczana jest ona wg wzoru 5.3.

$$\text{SER} = \frac{S_e}{S} * 100\% \tag{5.3}$$

gdzie:

S_e - liczba zdań błędnie rozpoznanych

S- liczba wszystkich zdań

Sposób korzystania z systemu, polegający na dyktowaniu pełnych linii kodu źródłowego i ich akceptowaniu lub odrzucaniu przez użytkownika sprawia, że najważniejszą metryką określającą użyteczność systemu dla użytkownika jest Sentence Error Rate.

Pierwsze testy

Punktem odniesienia dla dalszego rozwoju systemu jest test z wykorzystaniem modelu akustycznego będącego częścią projektu CMU Sphinx (model EN-US, odmiana PTM, wersja 5.2) i prostego modelu języka opartego o gramatykę JSGF. Gramatyka ta nie zawierała informacji o prawdopodobieństwach wystąpienia poszczególnych słów ani ich sekwencji. Testy przeprowadzono z wykorzystaniem programu Sphinx4 w wersji 5prealpha.

Uzyskano w ten sposób następujące wyniki:

```
WER = 70,64%

Accuracy = 72,25%

SER = 86,96%
```

(Listing 5.6)

Ponieważ wyniki przedstawione w listingu 5.6 uniemożliwiają stworzenie funkcjonalnego systemu tworzenia kodu źródłowego przy użyciu mowy, niezbędne okazało się wykonanie dodatkowej pracy mającej na celu zwiększenie dokładności działania systemu. Ciekawostką jest względnie wysoka wartość Accuracy pomimo niskiej jakości systemu potwierdzonej innymi metrykami, która wynika z pominięcia w tej metryce błędu polegającego na wstawianiu wyrazów.

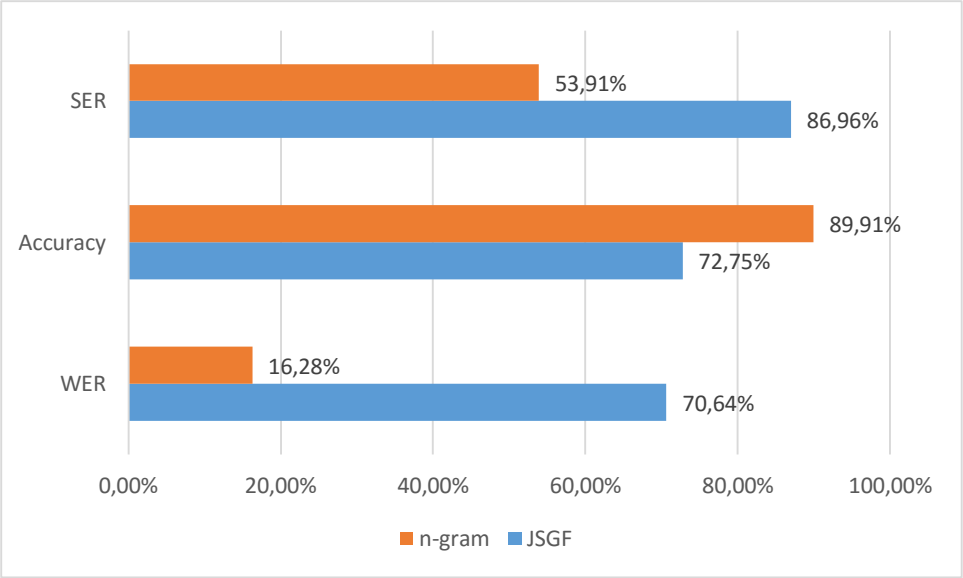
5.3. Model języka

System autorski wykorzystuje statystyczny model języka typu n-gram. Pierwszym krokiem do stworzenia takiego modelu było zdobycie dużego, liczącego kilkaset tysięcy linii kodu, korpusu kodu źródłowego języka Python. Powstał on z połączenia plików kodu otwarto-źródłowych projektów dostępnych w serwisie GitHub. Projekty te nie zostały wybrane na podstawie tematyki, jedynym kryterium była wysoka pozycja w wyszukiwarce Google.

Kolejny etap tworzenia modelu polegał na przetworzeniu korpusu – usunięto z niego wszystkie komentarze, zaś elementy języka Python, takie jak słowa kluczowe, operatory, liczby zastąpiono odpowiednikami opisanymi w podrozdziale 5.1. Wszystkie zmienne występujące w korpusie zostały tymczasowo zastąpione takim samym słowem.

Na podstawie tak przetworzonego modelu został następnie, za pomocą narzędzia lmtool, wygenerowany model 3-gramowy. Do wynikowego modelu w postaci pliku ARPA zostały dopisane dobrane arbitralnie prawdopodobieństwa komend obsługiwanych przez system. Każda z linii pliku ARPA zawierająca słowo oznaczające wystąpienie zmiennej zastąpiona została liniami zawierającymi nazwy pól tekstowych do wpisywania zmiennych w systemie. A zatem, prawdopodobieństwo sekwencji wyrazów zawierającej nazwę konkretnego pola (np. *‘alpha’*) jest równe prawdopodobieństwu sekwencji, w której nazwę tego pola zastąpiono inną (np. *‘bravo’*).

Wykres 5.1 zawiera porównanie jakości systemu opartego o statystyczny model n-gramowy z systemem opartym o gramatykę JSGF. Każda z użytych metryk wskazuje na znaczny wzrost dokładności rozpoznawania mowy.

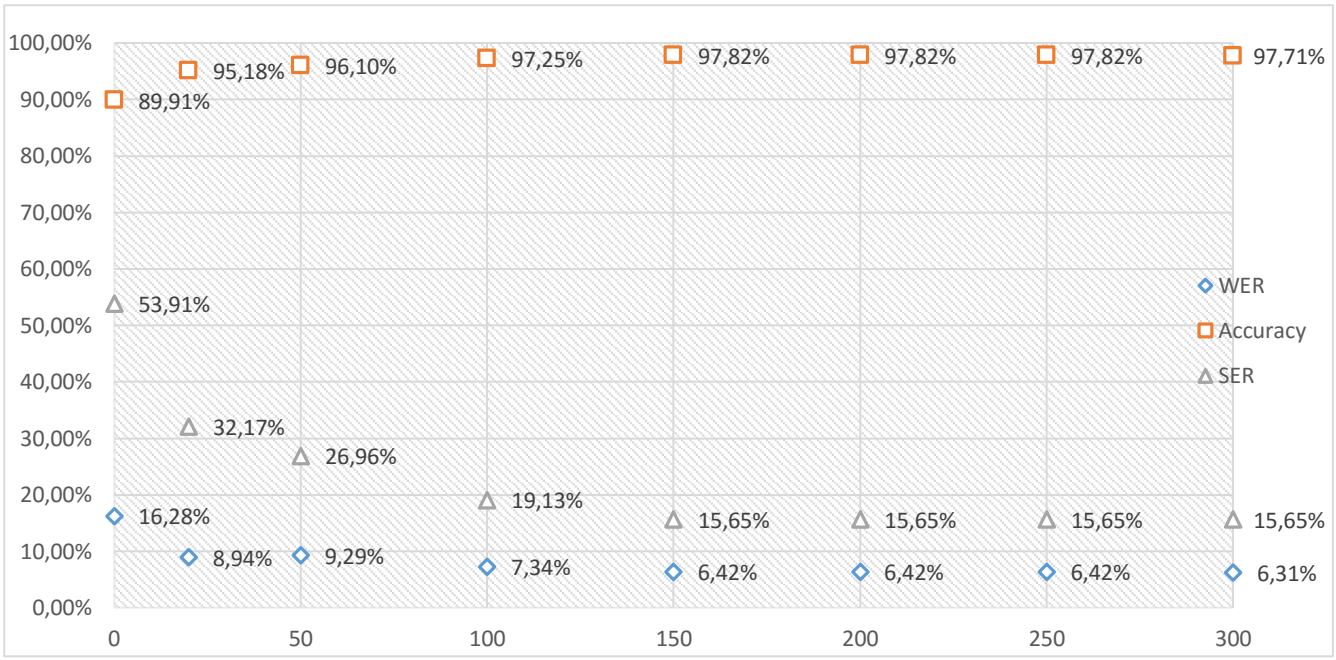


(Wykres 5.1)

5.4. Model akustyczny

Model akustyczny, odwzorowujący związek między sygnałem mowy a jednostkami akustycznymi (fonemami), ma kluczowy wpływ na jakość działania systemu. Zestaw narzędzi CMUSphinx zawiera wysokiej jakości model akustyczny języka angielskiego, a ponieważ słowa kluczowe języka Python są słowami pochodzącymi z języka angielskiego, nie było konieczności tworzenia modelu autorskiego. Wykorzystany został więc model w najnowszej, w chwili pisania projektu, wersji 5.2 i odmianie PTM (phonetically-tied mixture), który został zaadaptowany do głosu konkretnego użytkownika. Proces adaptacji nie wymaga tak dużej ilości danych i czasu, jak tworzenie nowego modelu akustycznego.

W celu adaptacji został stworzony kolejny korpus nagrań. Podobnie jak korpus testowy, korpus adaptacyjny składa się ze spisanych sekwencji wyrazów oraz ich nagrań. Wszystkie nagrania zostały wykonane przez tego samego użytkownika, model został zatem zaadaptowany dla pojedynczego mówcy.



(Wykres 5.2)

Wykres 5.2 przedstawia wyniki testów działania systemu w zależności od liczby nagrań stanowiących korpus adaptacyjny. Oś odciętych zawiera informacje o liczbie nagrań, zaś oś rzędnych o wartości procentowej metryki. Wszystkie testy zostały wykonane z użyciem opisanego w podrozdziale 5.3 n-gramowego modelu języka. Wykorzystano metodę adaptacji MAP. Do osiągnięcia najlepszych wartości SER i Accuracy wystarczył korpus liczący 150 nagrań, a więc zaledwie kilkuminutowy. Dalsze zwiększanie liczby nagrań nie prowadzi do istotnego polepszenia wartości metryk lub nawet do ich pogorszenia.

Adaptacja modelu akustycznego do pojedynczego mówcy zmniejszyła prawdopodobieństwo błędnego rozpoznania linii kodu z 53,91% do 15,65%.

5.5. Interfejs aplikacji

Rysunek 5.1 zawiera zrzut ekranowy interfejsu aplikacji systemu autorskiego.

Widoczne są na nim następujące elementy:

- rozpoznana linia kodu, którą użytkownik akceptuje komendą *ok* lub dyktuje ponownie
- pole edycyjne do którego wprowadzane są rozpoznane linie kodu
- pola tekstowe pozwalające wprowadzać nazwy zmiennych, literałów łańcuchowych i dekoratorów

Rys 5.1 Interfejs aplikacji systemu autorskiego

6. Podsumowanie

6.1. Jakość systemu

Z przeprowadzonych testów jakości działania systemu autorskiego wynika, iż tworzenie kodu źródłowego z wykorzystaniem mowy jest możliwe. Należy jednak pamiętać, iż funkcjonalność systemu autorskiego jest mocno ograniczona i nie pozwala ona na rezygnację ze stosowania tradycyjnych urządzeń wskazujących, a jedynie na ograniczenie ich użycia w trakcie tworzenia kodu.

Z testów można też wysnuć wniosek, iż zastosowanie statystycznego n-gramowego modelu języka znacząco poprawia dokładność pracy systemu w porównaniu z modelem, którego jedynym zadaniem jest ograniczenie liczby rozpoznawanych słów. Narzędzia służące do głosowego tworzenia kodu źródłowego powinny być więc dedykowane konkretnym językom programowania.

Należy również zwrócić uwagę iż stosunkowo niewielka ilość danych pozwala na skuteczne zaadoptowanie modelu akustycznego do głosu konkretnego użytkownika, znacznie zwiększając dokładność działania systemu. Narzędzia wspomagające programowanie głosowe powinny zatem udostępniać użytkownikowi możliwość adaptacji modelu.

6.2. Dalszy rozwój

System autorski jest aplikacją typu *proof of concept*. Przy założeniu, iż celem jest stworzenie użytecznego systemu wspomagającego tworzenie kodu, dalsza praca nad samą aplikacją wydaje się być pozbawiona sensu - znacznie lepszym rozwiązaniem byłoby stworzenie wtyczki do istniejącego środowiska programistycznego lub nowego środowiska programistycznego od początku projektowanego do głosowej obsługi. Jeżeli jednak celem miałoby być napisanie prostej aplikacji do celów edukacyjnych, służącej osobom niepełnosprawnym, kontynuacja prac jest możliwa. Należałoby wówczas wprowadzić możliwość

głosowego wprowadzania (np. litera po literze) nazw zmiennych, literałów łańcuchowych i dekoratorów a także zwiększyć liczbę komend głosowych służących do kontrolowania aplikacji.

Bibliografia:

[1] Bartosz Ziółko, Mariusz Ziółko, ‘Przetwarzanie Mowy’, Wydawnictwa AGH, Kraków 2011

[2] Ryszard Tadeusiewicz, ‘Sygnał mowy’, Wydawnictwa Komunikacji i Łączności, Warszawa 1988

[3] Agnieszka Mykowiecka, , ‘Inżynieria lingwistyczna. Komputerowe przetwarzanie tekstów w języku naturalnym’, Wydawnictwo Polsko-Japońskiej Wyższej Szkoły Technik Komputerowych, Warszawa 2007

[4] The CMU Pronouncing Dictionary, data odczytu: 28 kwietnia 2016, adres: http://www.speech.cs.cmu.edu/cgi-bin/cmudict

[5] ngram-format, data odczytu: 26 kwietnia 2016, adres: http://www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html

[6] ARPA Language models, data odczytu: 27 marca 2016, adres: http://cmusphinx.sourceforge.net/wiki/sphinx4:standardgrammarformats

[7] JSpeech Grammar Format, data odczytu: 28 marca 2016, adres: https://www.w3.org/TR/jsgf/

[8] JSGF Grammars in Sphinx4, data odczytu : 25 marca 2016, adres: http://cmusphinx.sourceforge.net/wiki/sphinx4:jsgfsupport

[9] HTK Speech Recognition Toolki, data odczytu: 20 maja 2016, adres: http://htk.eng.cam.ac.uk

[10] Alfabet fonetyczny ICAO – Wikipedia, wolna encyklopedia, data odczytu: 15 lutego 2016, adres: https://pl.wikipedia.org/wiki/Alfabet_fonetyczny_ICAO,

[11] Overview (Sphinx-4), data odczytu: 20 maja 2016, adres: http://www.gavo.t.u-tokyo.ac.jp/~kuenishi/java/sphinx4/overview-summary.html

[12] Abstract - The use of speech technologies for intellectualization of the input-output of the program code, data odczytu: 10 maja 2016, adres: http://masters.donntu.org/2015/fknt/bakalenko/diss/indexe.htm

[13] Resume – Daria Savkova - Speech interface for intellectualization of programming language texts input, data odczytu: 10 maja 2016, adres: http://masters.donntu.org/2013/fknt/savkova/indexe.htm

[14] Przetwarzanie mowy polskiej, data odczytu: 9 maja 2016, adres: http://www.slideshare.net/studenckifestiwalinformatyczny/1-ziko