Metody i narzędzia rozpoznawania mowy w zastosowaniach niekomercyjnych

Jarogniew Rykowski

Streszczenie: W artykule przedyskutowano problem zaprojektowania systemu rozpoznawania mowy. W tekście pokazano, jak stosunkowo tanim kosztem, zarówno z punktu widzenia programisty (koszty przygotowania programu), jak i końcowego użytkownika (koszty nauki korzystania z systemu), uzyskać efektywną możliwość komunikacji głosowej z komputerem.

Artykuł pokazuje, po pierwsze, że rozpoznawanie mowy na potrzeby amatorskie i półprofesjonalne jest obecnie jak najbardziej możliwe i wymaga minimalnych nakładów na sprzęt audio (w zasadzie wystarczy mikrofon i głośnik wbudowany praktycznie w każdy komputer). Po drugie, dla języka programowania Java są dostępne co najmniej dwa rozwiązania niekomercyjne, za które nie trzeba bezpośrednio płacić: biblioteka Sphinx oraz usługa Google Voice Translator dostępna za pomocą biblioteki Jarvis. Google oferuje ponadto możliwość syntezy (i odtworzenia w głośniku komputera) dowolnego komunikatu głosowego na podstawie przesłanego tekstu. Po trzecie, rozpoznanie mowy jest dostępne dla dowolnego mówcy (systemu nie trzeba uczyć wymowy, intonacji, akcentu, definiować słownika itp.) i przebiega z bardzo dużą skutecznością, która w zupełności wystarcza w większości zastosowań amatorskich.

METHODS AND TOOLS FOR AUTOMATIC VOICE RECOGNITION FOR NONCOMMERCIAL USE

Abstract: *In this paper a problem is discussed of effective development of an automatic voice-recognition system for noncommercial use. The text shows how, at a relatively cheap cost, both from the point of view of the programmer (the cost of preparing the program), as well as the end-user (the cost of learning how to use the system), to obtain effective voice communication with the computer.*

The goal of the paper is threefold. First, it is shown that automatic voice recognition for amateur usage is at the moment quite easy to achieve – it is enough to utilize standard microphone/speaker of any computer. Second, there are at least two noncommercial software solutions based on popular Java programming language: Sphinx library and Google Voice Translator to be accessed via JARVIS library. In addition, Google offers a possibility of voice synthesis based on any text, in almost any national language. Third, the system easily recognizes voice commands of any speaker, regardless pronunciation, intonation, accent, etc., and the overall effectiveness is sufficient for most amateur applications.

1. Wstęp

Umiejętność porozumiewania się za pomocą mowy jest cechą, która odróżnia człowieka od zwierząt. Od początków rozwoju cywilizacji to właśnie mowa jest podstawą kontaktów międzyludzkich. Posługujemy się nią praktycznie wszędzie, wyrażając uczucia, myśli, wydając polecenia itp. W ostatnich latach położono szczególny nacisk na przeniesienie tego podstawowego sposobu komunikacji człowieka na grunt komputerów – umożliwienie interakcji z maszyną nie za pomocą klasycznego tandemu klawiatura/ekran, ale za pomocą wypowiedzi słownych, które maszyna będzie rozumieć i na które będzie odpowiednio reagować. Sterowanie komputerem z wykorzystaniem mowy ma wiele zalet w stosunku do tradycyjnych metod, takich jak myszka czy klawiatura. Mowa dla ludzi jest rzeczą naturalną, która nie wymaga specjalnego treningu (pomijając oczywiście naukę w niemowlęctwie). Wykorzystanie tego typu komunikacji pozwala na jednoczesne wykonywanie innych czynności, przy użyciu na przykład, rąk. Ponadto jest to szybsza metoda komunikacji niż pisanie przy wykorzystaniu klawiatury.

Mowa to nie tylko komunikacja werbalna, czyli wypowiadanie i rozumienie słów wypowiedzianych w konkretnym języku. Z mową wiąże się nierozerwalnie komunikacja niewerbalna – tak zwana „mowa ciała" – sposób wypowiadania się, intonacja, akcent, mimika, gesty (szczególnie wykonywane rękami). Komunikacja niewerbalna nie zawsze jest w pełni kontrolowana przez rozmówcę – przekaz niewerbalny jest często generowany i odbierany podświadomie jako uczucia (niechęć, zainteresowanie, znudzenie) i ogólne wrażenie. Komunikacja niewerbalna odgrywa kluczową rolę w procesie przekazywania informacji u ludzi. Ocenia się, że zaledwie kilka procent przekazu stanowią słowa, reszta to intonacja (kilkadziesiąt procent) oraz mimika i gesty (ponad połowa całości przekazu).

Naukowcy od dłuższego czasu zastanawiają się nad najbardziej efektywną metodą analizy mowy przez komputer. Ze względu na niejednoznaczność komunikacji niewerbalnej, jej kontekstowość (ukryte znaczenie wypowiedzi), intuicyjny i podświadomy odbiór przekazu, a także kwestie czysto techniczne, jak ograniczone zasoby sprzętowe i programowe, w pierwszej kolejności skupiono badania nad prostszym i bardziej poznanym od strony teorii elementem – komunikacją werbalną. Badania te zaowocowały wieloma systemami automatycznego rozpoznawania mowy (ASR, ang. *Automatic Speech Recognition*). Celem systemu ASR jest jak najdokładniejsze przetworzenie sygnału akustycznego na tekst (ciąg słów) w jak najszybszym czasie, z jak najmniejszym błędem, a tym samym zwiększenie możliwości interakcji między użytkownikiem a komputerem.

Systemy ASR [1] dzielą się na dwa typy: systemy zależne od mówcy, które najpierw uczą się wymowy danej osoby w celu zwiększenia skuteczności rozpoznawania wypowiadanych słów, oraz systemy niezależne, które rozpoznają mowę dowolnej

osoby, ale ze znacznie mniejszą skutecznością. To właśnie tej drugiej klasie systemów ASR jest poświęcony niniejszy artykuł. W tekście skupiono się na dwóch głównych podejściach do rozwiązania problemu zamiany mowy na tekst – polegających na wykorzystaniu łańcuchów Markowa i tzw. n-gramów – oraz dwóch głównych propozycjach dostępnych w zastosowaniach niekomercyjnych: bibliotece Sphinx oraz nieoficjalnej usłudze Voice Translator firmy Google. Dodatkowo krótko przedyskutowano drugi etap analizy mowy, czyli zrozumienie (interpretację) wypowiadanego tekstu za pomocą techniki zwanej chatterbotem.

2. Metody zamiany mowy na tekst

Jak wspomniano wcześniej, w chwili obecnej w dziedzinie rozpoznawania mowy prym wiodą dwie metody. Pierwsza z nich wykorzystuje łańcuchy Markowa, natomiast druga opiera się na tak zwanych n-gramach. Metody te obrazowo przedstawiono poniżej, uzupełniając opisy dyskusją na temat ograniczeń oraz możliwych obszarów zastosowań.

Obie metody wykorzystują podobną analizę fonetyczną, czyli wyodrębnienie z przekazu słownego zgłosek, a następnie pogrupowanie tych zgłosek w zrozumiałe słowa. Na potrzeby dalszej części tekstu przyjmijmy intuicyjnie, że jest to zagadnienie stosunkowo proste[1] od strony technicznej (jakkolwiek dość skomplikowane od strony językowej), z tego względu jego szczegółowy opis nie jest potrzebny do zrozumienia całego rozwiązania. Załóżmy, że program analizujący potrafi wyodrębnić z przekazu mowy zgłoski oraz ciszę (przerwy między wypowiadanymi słowami), a następnie pogrupować zgłoski w słowa i porównać takie słowa ze znanym sobie słownikiem. W wyniku porównania system określa to słowo, które z największym prawdopodobieństwem jest odpowiednikiem zarejestrowanego fragmentu wypowiedzi. Jeśli system ma wątpliwości co do wyniku, to w odpowiedzi może podać listę możliwych słów, uszeregowaną pod kątem malejącego prawdopodobieństwa poprawnego rozpoznania. Produktem wyjściowym do dalszej analizy jest zatem ciąg wypowiedzianych słów, przy czym dla każdego słowa określa się prawdopodobieństwo jego wystąpienia oraz ewentualne słowa alternatywne.

2.1. Wykorzystanie łańcuchów Markowa

W metodzie tej zakładamy, że znamy z góry wszystkie możliwe kombinacje wypowiadanych ciągów słów (zdań, poleceń), czyli gramatykę języka. Jeśli potrafimy określić te kombinacje, możemy też próbować „dopasować" rozpoznawane słowa do konkretnej kombinacji. Kombinacja o najlepszym stopniu takiego „dopasowania" będzie wynikiem analizy. Zatem w tej metodzie kluczowe jest określenie gramatyki języka, który będzie rozpoznawany przez analizator mowy. W celu przybliżenia procesu definiowania i analizy gramatyki posłużmy się przykładem „inteligentnego domu", w którym chcemy za pomocą poleceń głosowych otwierać okna i drzwi. Możemy z góry przewidzieć, jakie polecenia będą w tym celu używane[2]:

```
otwórz okno
zamknij okno
otwórz drzwi
zamknij drzwi
```

Powyższy zapis jest nieformalną definicją gramatyki języka komunikacji z komputerem. Już na pierwszy rzut oka jest też widoczne, że jest on nadmiarowy – należy zatem pomyśleć o jego formalizacji oraz skróceniu. W tym celu możemy się posłużyć notacją BNF (ang. *Backus-Naur Form*, od nazwisk autorów tego rozwiązania) [2], wykorzystywaną niemal od początku rozwoju komputerów. Po pierwsze, wprowadźmy elementy formalizacji języka, dzieląc zapis na elementy terminalne (odpowiadające zanalizowanym słowom) oraz elementy pośrednie, które zwiększają czytelność zapisu języka oraz usprawniają jego analizę. Elementy pośrednie są podczas analizy rozwijane do kombinacji innych elementów, pośrednich lub terminalnych. Elementy te odróżnijmy specjalną metodą zapisu – ujmując je w znaki ‚<' oraz ‚>'. Wprowadźmy też znak przypisania[3], który pokazuje możliwe kroki analizy, zamieniając element pośredni (lewa strona przypisania) na wyżej opisaną kombinację (prawa strona). Nasza gramatyka przyjmie postać:

```
<polecenie> = otwórz okno
<polecenie> = zamknij okno
<polecenie> = otwórz drzwi
<polecenie> = zamknij drzwi
```

Powyższy zapis jest bardzo nadmiarowy. W celu jego uproszczenia posłużmy się operatorem alternatywy (wyboru), symbolizowanym znakiem ‚|', który określa, że dane możliwości (kroki) analizy są sobie równoważne:

```
<polecenie> = otwórz okno | zamknij okno | otwórz
drzwi | zamknij drzwi
```

Jeśli wprowadzimy dwa dodatkowe symbole nieterminalne, jeszcze bardziej zwiększymy czytelność zapisu:

```
<polecenie> = <czynność> <obiekt>
<czynność> = zamknij | otwórz
<obiekt> = okno | drzwi
```

W pierwszym wierszu nie wykorzystujemy operatora alternatywy – jest to niejawny operator koniunkcji (połączenia), który oznacza, że w poleceniu musi wystąpić zarówno słowo opisujące czynność, jak i wskazanie na obiekt, w takiej dokładnie kolejności. Jeśli chcemy wprowadzić dowolność w kolejności rozpoznawania tych słów, musimy zdefiniować możliwe alternatywy polecenia (w tym przypadku są tylko dwie):

```
<polecenie> = <czynność> <obiekt> | <obiekt>
<czynność>
<czynność> = zamknij | otwórz
<obiekt> = okno | drzwi
```

Musimy także przyjąć, od jakiego elementu nieterminalnego zaczynamy analizę. Dla uproszczenia załóżmy, że jest to pierwszy z definiowanych elementów pośrednich w gramatyce – w powyższym przykładzie będzie to element <polecenie>.

W analizie wykorzystujemy aparat matematyczny zwany ukrytymi łańcuchami Markowa (ang. *Hidden Markov Models* HMM). Aparat HMM [3] umożliwia statystyczną analizę ciągu

zdarzeń, w tym przypadku – ciągu wypowiadanych słów[4]. Jeśli nauczymy komputer, jakie jest prawdopodobieństwo wystąpienia danego słowa po zanalizowaniu ciągu poprzednich słów, to możemy drogą kolejnych kroków rozpoznać całe polecenie. Do „nauczania" służy właśnie wcześniej zdefiniowana gramatyka. Na przykład, jeśli analizę powyższej gramatyki przeprowadzamy dla ciągu wypowiedzianych słów „otwórz okno", to w pierwszej kolejności program sprawdza, czy pierwsze słowo to opis czynności (pierwszy element nieterminalny elementu <polecenie> to <czynność>). Rozpoznanie słowa „otwórz" jako jednej z alternatyw elementu <czynność> powoduje, że w kolejnym kroku analizy przechodzimy do elementu „obiekt", dla którego sprawdzamy wystąpienie słów „okno" lub „drzwi". Ponieważ w ciągu wejściowym w tym momencie analizy występuje słowo „okno", uzyskaliśmy pełne dopasowanie do elementu <polecenie> i tym samym możemy zakończyć cały proces, raportując rozpoznanie ciągu dwóch słów „otwórz" i „okno".

Rozpoznanie ciągu „okno otwórz" (zapis nie jest naturalny dla człowieka, ale prawidłowy z punktu widzenia gramatyki) nie przebiega tak bezproblemowo. Podobnie jak w poprzednim przykładzie, w pierwszym podejściu próbujemy zanalizować element <czynność>, który nie ma zdefiniowanego słowa „okno" jako elementu terminalnego. Ponieważ nie jest to możliwe i analiza na tym etapie nie zakończy się sukcesem, wracamy do kroku, w którym rozpoczęliśmy analizę elementu <polecenie>, ale tym razem kontynuujemy analizę na podstawie drugiej alternatywy, w której zdefiniowano możliwość odwrócenia kolejności słów określających czynność i obiekt. Tym razem analiza zakończy się sukcesem, a w jej wyniku otrzymamy ten sam zestaw słów co poprzednio, aczkolwiek w innej kolejności.

Jeśli dla powyższej gramatyki spróbujemy zanalizować ciąg znaków „otwórz, proszę, okno", to niestety taka analiza nie zostanie zakończona powodzeniem. Gramatyka nie uwzględnia żadnych słów poza słowami określającymi czynność oraz obiekt. Zatem napotkanie słowa „proszę" uniemożliwi poprawną analizę całego polecenia, nawet jeśli pozostałe słowa są poprawne. Można to ograniczenie ominąć, dodając w gramatyce możliwość pomijania w analizie pewnych słów:

```
<polecenie> = <czynność> <cokolwiek> <obiekt>
<czynność> = zamknij | otwórz
<obiekt> = okno | drzwi
<cokolwiek> = | proszę
```

W powyższym przykładzie ostatni wiersz definiuje alternatywę, która zakłada możliwość wystąpienia elementu pustego (braku słowa). W ten sposób można uelastycznić język komunikacji, wprowadzając elementy opisu gramatyki, które są w pewnym sensie nadmiarowe, ale jednocześnie likwidują skutki ewentualnych błędów i niejednoznaczności języka naturalnego.

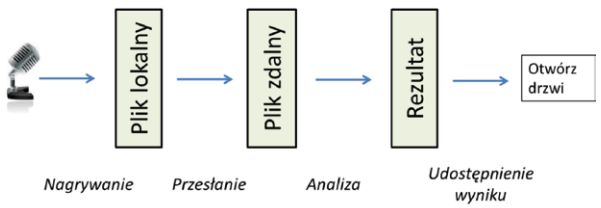
Przedstawiony powyżej opis jest bardzo uproszczony – nie pokazano, jak definiować wielokrotne wystąpienie danego słowa lub grupy słów, wystąpienie dowolnego słowa (pomijanego w analizie), ciszy itp. Zainteresowanych odsyłamy do dokumentacji tego systemu, który będą wykorzystywać w swoim programie – niestety składnia tych rozszerzeń bardzo mocno zależy od konkretnej implementacji i trudno ją opisać w jeden zunifikowany sposób.

Podsumowując – metodę analizy mowy wykorzystującą gramatykę języka oraz aparat HMM możemy wykorzystać tylko wtedy, gdy potrafimy z góry przewidzieć wszystkie możliwe kombinacje wypowiadanych słów (na przykład są to proste polecenia sterujące pracą pewnej maszyny), a także dysponujemy odpowiednim słownikiem, specyficznym dla każdego języka narodowego. Słownik taki umożliwia dopasowanie reprezentacji tekstowej słowa do wypowiadanego ciągu zgłosek, a gramatyka – do całościowej wypowiedzi (zdania, polecenia). Przedstawiona metoda ma stosunkowo małe wymagania, jeśli chodzi o moc obliczeniową procesora, ale niestety, nawet w przypadku mało skomplikowanych tekstów, intensywnie wykorzystuje ona pamięć operacyjną komputera (zwłaszcza w przypadku rozbudowanych gramatyk oraz wielowariantowych kroków analizy).

2.2. Wykorzystanie analizy n-gramowej

Innym podejściem do analizy mowy jest wykrywanie w wypowiadanym tekście zbitek liter lub słów, czyli tak zwanych n-gramów [4] (*n* oznacza liczbę liter/słów w wykrywanej sekwencji). Najczęściej w analizie języka wykorzystuje się 2, 3 oraz 4 gramy. Na potrzeby takiej analizy należy najpierw utworzyć tak zwany korpus, czyli statystycznie zanalizować jak najwięcej tekstów, dzieląc je na zbitki literowe/słowne o określonej długości. W praktyce korpusy są tworzone na podstawie milionów różnorodnych tekstów, zatem ich przygotowanie (dla danego języka narodowego) zajmuje wiele lat pracy i wymaga bardzo dużych nakładów. Korpusy te są też bardzo dużymi bazami danych – dla przykładu, typowy korpus języka polskiego zawiera około dziesięciu milionów słów (część z nich to nazwy własne, duża część wynika z odmiany gramatycznej innego słowa, część jest zapożyczona z innych języków, pewna część jest także wynikiem błędów językowych). Przetworzenie tak olbrzymiej ilości danych w jednym komputerze w rozsądnym czasie jest praktycznie niemożliwe. Z tego względu analizę n-gramową mogą wykonywać jedynie wyspecjalizowane farmy serwerów.

Z punktu widzenia końcowego użytkownika przetwarzanie mowy z wykorzystaniem n-gramów i przetwarzaniem po stronie serwera jest procesem wieloetapowym (rys. 1). W pierwszym etapie użytkownik, korzystając z lokalnego sprzętu, nagrywa dźwięk do pliku przez określony czas. W drugim etapie nagrany plik, jako strumień danych audio, jest przesyłany do serwera. Trzeci etap obejmuje analizę zawartości przesłanego pliku przez serwer. W ostatnim, czwartym etapie użytkownikowi odsyłany jest rezultat analizy, w postaci najbardziej prawdopodobnego zestawu słów będącego odpowiednikiem przekazu albo też listy alternatywnych zestawów, z określonym współczynnikiem prawdopodobieństwa („sukcesu") dla każdego zestawu.



Rys. 1. Cztery etapy analizy n-gramowej z punktu widzenia końcowego użytkownika

Podsumowując – metoda analizy mowy z wykorzystaniem n-gramów ma praktycznie zerowe wymagania, jeśli chodzi o moc obliczeniową lokalnego procesora oraz zajętość pamięci, gdyż całość przetwarzania odbywa się po stronie serwera. Z punktu widzenia końcowego użytkownika jest to metoda bardzo elastyczna, gdyż nie trzeba definiować gramatyki języka i w ten sposób ograniczać zakresu rozpoznawanych słów i zdań. Metoda ta wymaga jednak bardzo szybkiego łącza do sieci rozległej – nagrywane pliki mają wielkość wielu MB, nawet dla kilkusekundowych nagrań. Dodatkowo dostawca usługi musi dysponować bazą danych milionów, jeśli nie miliardów wypowiedzi i tekstów, które służą do zdefiniowania korpusu języka, a także bardzo wydajną farmą serwerów. Na świecie jest niewiele firm lub organizacji (włączając całe państwa), które spełniają te warunki, wśród nich na pierwszy plan wybija się amerykańska firma Google.

3. Chatterbot – uproszczona metoda analizy znaczenia tekstu

Rozpoznane ciągi słów, po zamianie na tekst, trzeba jeszcze zinterpretować pod kątem ich znaczenia. Na przestrzeni ostatnich lat dokonano istotnego postępu w dziedzinie analizy języka naturalnego, jednakże analiza taka wiąże się z ekstremalnym wykorzystaniem zasobów komputera lokalnego (procesor, pamięć), a także z reguły długim czasem oczekiwania na wynik. Jednocześnie, w wielu sytuacjach sprawdzenie poprawnej budowy gramatycznej zdania, wyodrębnienie podmiotu, orzeczenia i dopełnienia itp. nie jest potrzebne – do celów na przykład sterowania inteligentnym budynkiem wystarczy posługiwanie się kilkunastoma słowami-kluczami. Dlatego można zastąpić pełną analizę gramatyczną i znaczeniową wypowiadanych zdań analizą uproszczoną, polegającą na wyodrębnieniu z treści wypowiedzi tylko znanych systemowi słów kluczowych, z pominięciem reszty. Do tych celów służą programy zwane chatterbotami, rozwijane od lat siedemdziesiątych ubiegłego wieku i cyklicznie rywalizujące w konkursie o Nagrodę Loebnera [5]. Oczywiście do prostych celów sterowania (a wydaje się, że właśnie taki jest główny powód stosowania niekomercyjnych systemów rozpoznawania mowy) instalowanie rozbudowanego chatterbotu nie jest konieczne, można za to wykorzystać ten sposób analizy w swoim programie. W najprostszym przypadku analiza taka polega na podziale tekstu uzyskanego z analizy mowy na poszczególne słowa, usunięciu duplikatów oraz porównaniu każdego słowa z zestawem słów kluczowych. Po wykryciu danego słowa kluczowego (lub kilku) program analizujący ma za zadanie podjąć wskazane czynności. Analizę taką można poprowadzić wieloetapowo, na każdym etapie analizując inny zestaw słów, co umożliwi kontekstową parametryzację poleceń głosowych [6].

4. Środowisko CMU Sphinx

4.1. Geneza i architektura środowiska

CMU Sphinx, zwykle zwany Sphinx, jest zespołem systemów, których celem jest rozpoznawanie mowy [7]. Zawiera on dwa główne elementy odpowiedzialne za właściwie rozpoznawanie mowy (Sphinx 2-4) oraz nauczanie (trening) modelu akustycznego (SphinxTrain). Sphinx został zaprojektowany na Uniwersytecie Carnegie Mellon w Pittsburghu. W 2000 roku komponent Sphinx v.2 otrzymał prawo rozpowszechniania na zasadach wolnego oprogramowania (ang. *open source*), rok później prawo to otrzymał również Sphinx v.3. W Sphinx v.4 silnik został całkowicie przepisany w języku programowania Java, tak aby opracować narzędzie znacznie elastyczniejsze w rozpoznawaniu mowy. Odmianą systemu jest PocketSphinx, przygotowany specjalnie dla urządzeń przenośnych oraz mobilnych.

Środowisko Sphinx ma budowę modułową. Architektura systemu jest tak pomyślana, żeby była możliwa zamiana jednego modułu na inny oraz rozbudowa systemu o dodatkowe funkcje, realizowane przez nowe moduły. Przykładem mogą być dostępne różne implementacje modelu języka (np. JSGFGrammar).

Sposób, w jaki działa rozpoznawanie mowy, jest następujący. Cały proces rozpoczyna się od fal dźwiękowych, które reprezentują odpowiednie dźwięki, w tym przypadku mowę. Fale te dzielone są na części oddzielone ciszą, tak aby każda część reprezentowała jedno słowo. Następnie system rozpoczyna właściwie rozpoznawanie. Aby tego dokonać, pobierane są wszystkie możliwe kombinacje słów tak, aby dopasować je do zarejestrowanego dźwięku. Moduł analizy języka tworzy graf rozbioru semantycznego wypowiedzi, wykorzystując strukturę języka (gramatykę) oraz strukturę topologiczną modelu akustycznego. Dodatkowo możliwe jest wykorzystanie słownika, którego celem jest mapowanie słów modelu języka na sekwencję elementów modelu akustycznego. Moduł analizy korzysta z dwóch modeli: językowego i akustycznego, oraz słownika. Model języka zawiera listę słów i prawdopodobieństwo ich wystąpienia. Wykorzystuje się go, aby ograniczyć wyszukiwanie w dekoderze, ograniczając liczbę możliwych słów, które mają być brane pod uwagę w dowolnym momencie wyszukiwania.

Model akustyczny zapewnia odwzorowanie między jednostką mowy a aparatem HMM. Model ten zawiera reprezentację każdego z odrębnych dźwięków, które tworzą całe słowo. Każda z tych reprezentacji nazywana jest fonemem. Model akustyczny jest tworzony z wykorzystaniem nagrania oraz treści danego nagrania, po czym tworzona jest statyczna reprezentacja każdego dźwięku, z którego składa się pojedyncze słowo. Model ten odgrywa bardzo ważną rolę w rozpoznawaniu mowy. Aby dobrze działał, musi wcześniej zostać odpowiednio wyćwiczony.

4.2. Prosty chatterbot dla środowiska Sphinx

Wykorzystanie biblioteki Sphinx pokażemy na przykładzie programu napisanego w języku programowania Java. Program taki piszemy wieloetapowo. W pierwszej kolejności należy określić gramatykę analizowanego języka, tak jak to wspomniano wcześniej. W tym celu należy utworzyć plik zawierający definicję tej gramatyki w formacie JSGF[5] [8] i zapisać w dowolnym katalogu:

```
#JSGF V1.0;
/**
 * JSGF Grammar for Sphinx Voice Gateway
*/
grammar sample;
public <command> = <action> <object> | <end> ;
<action> = open | close ;
<object> = doors | door | windows | window ;
<end> = stop | end;
```

Należy także utworzyć i zapisać na dysku plik konfiguracyjny środowiska Sphinx – może to być dokładna kopia pliku załączonego do dokumentacji tego środowiska, nie będziemy w jego zawartość ingerować, z jednym wyjątkiem – w wierszu definiującym element „JSGFGrammar" należy podać dokładną lokalizację pliku z wcześniej przygotowaną gramatyką języka.

W programie należy przewidzieć dwie główne funkcje: inicjacji środowiska (wykonywana jednokrotnie podczas startu programu) oraz wywołania procesu analizy mowy (ta z kolei może być wywoływana wielokrotnie). W poniższym opisie podamy przykładową definicję tych funkcji bez sposobu ich wywołania w kodzie programu – odkrycie, jak to ostatnie wykonać, pozostawiamy Czytelnikowi.

```
public boolean initializeVoiceGateway (String
configurationFile)
   throws IOException,
   JSGFGrammarParseException,
   JSGFGrammarException{
   boolean ok=true;
   // sprawdzenie dostępności do pliku konfigura-
   cyjnego środowiska
   File f = new File(configurationFile);
   if (!f.exists())
      return !ok;
   if (f.isDirectory())
      return !ok;
   // zainicjowanie środowiska
   ConfigurationManager configManager =
   new ConfigurationManager(configurationFile);
   // zainicjowanie zmiennych pomocniczych
   (globalnych dla całego programu)
   recognizer = (Recognizer)
   configManager.lookup(„recognizer");
   if (recognizer==null)
      return !ok;
   microphone = (Microphone)
   configManager.lookup(„microphone");
   if (microphone==null)
      return !ok;
   grammar = (JSGFGrammar)
   configManager.lookup(„jsgfGrammar");
   if (grammar==null)
      return !ok;
   // rezerwacja zasobów sprzętowych na
   potrzeby programu
   recognizer.allocate();
   // rozpoczęcie nasłuchu przez mikrofon
   ok=microphone.startRecording();
      return ok;
}
public synchronized String invokeOnce(){
   String resultText=null;
   Result result = recognizer.recognize();
   if (result != null) {
      // wynik jest niepusty, jeśli analiza się za-
      kończyła sukcesem
      resultText = result.getBestFinalResultNo-
      Filler();
   }
   return resultText;
}
```

Jako ostatni element można zaproponować prosty chatterbot, który wypisuje na ekranie uzyskane w wyniku analizy słowa, a po wykryciu polecenia „end" lub „stop" kończy działanie programu:

```
public void activate (String text){
   if (text==null)
      return;
   // jeśli analiza mowy się nie powiodła, nie ma
   danych wyniku - wyjście
   System.out.println(„Polecenie: „+text);
   // test końca pracy programu
   text=text.toLowerCase();
   if (text.contains(„end") || text.contains
   („stop") )
      System.exit(0);
}
```

Czytelnikowi należy się w tym miejscu wyjaśnienie, dlaczego w powyższym przykładzie pojawiają się anglojęzyczne polecenia. Niestety, środowisko Sphinx nie oferuje jak dotąd korpusu dla języka polskiego, korpus taki nie jest też dostarczany niekomercyjnie przez inną firmę zewnętrzną lub organizację. Wielu polskich użytkowników tego systemu pokusiło się o przygotowanie własnych korpusów z ograniczonym zestawem słów, jednakże są one bardzo specjalizowane (na przykład do sterowania robotami lub postaciami w grze komputerowej) i trudno dostępne (najczęściej udostępniane tylko na życzenie, bezpośrednio przez Autorów). Z tego względu podstawowym językiem narodowym dla tej biblioteki w zastosowaniach niekomercyjnych pozostaje ciągle język angielski.

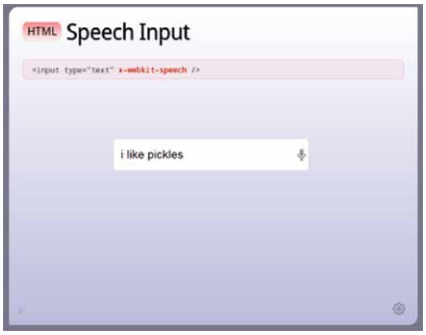
Eksperymenty przeprowadzone przez Autora niniejszego tekstu z różnymi wartościami parametrów z pliku konfiguracyjnego oraz dla różnych komputerów i mikrofonów pokazują, że analiza w środowisku Sphinx jest bardzo silnie zależna od jakości posiadanego sprzętu, w szczególności mikrofonu. Stosunkowo dobrze sprawdzają się w tym systemie mikrofony zewnętrzne, najczęściej kierunkowe, natomiast mikrofony wbudowane w komputery przenośne charakteryzują się bardzo niską przydatnością, ze względu na większą czułość (nagrywanie dźwięków otoczenia) oraz charakterystykę dookólną. Duży wpływ na stopień rozpoznawania słów ma także sposób ich wymowy (trzeba starać się maksymalnie naśladować wymowę oryginalną, np. amerykańską, szczególnie w odniesieniu do krótkich słów bogatych w samogłoski „a" oraz „o"). Mimo tych wad, jest to jedyne rozwiązanie w zakresie niekomercyjnego rozpoznawania mowy, które jest możliwe do wykorzystania przy braku dostępu do sieci rozległej i konieczności pracy offline.

5. Środowisko Google Voice Translator

Uwaga: *Materiały przedstawione poniżej nie są oficjalnymi informacjami ani firmy Google, ani też firm z nią współpracujących i powstały albo jako wynik przemyśleń oraz eksperymentów au-*

tora niniejszego opracowania, albo też jako nieoficjalny wynik badań osób niezależnych (najczęściej pracowników i studentów kilku amerykańskich uniwersytetów). Przedstawione w tekście informacje w każdej chwili mogą okazać się nieaktualne lub wręcz błędne, gdyż powstały w dużej mierze na bazie domysłów, a nie rzetelnej informacji uzyskanej od firmy Google. Autorzy opracowania oraz tekstów źródłowych dołożyli jednak wszelkich starań, aby prawdopodobieństwo takiej sytuacji było minimalne.

Wraz z wypuszczeniem wersji 11-beta przeglądarki Chrome firma Google udostępniła rozszerzenie standardu HTML5 o rozpoznawanie mowy. Wypowiedziane dźwięki, po przechwyceniu przez mikrofon komputera lub telefonu, są w tym programie nagrywane jako plik w formacie FLAC [9] oraz przesyłane do serwera Google. Tam plik dźwiękowy jest analizowany z wykorzystaniem n-gramów oraz korpusu słów i wyrażeń, a uzyskany w wyniku tej analizy tekst, odpowiadający wypowiedzianym słowom, jest odsyłany jako odpowiedź usługi w formacie zbliżonym do XML. Usługę analizy mowy można wypróbować, przechodząc do strony o adresie **http://slides.html5rocks.com/#speech-input**[6] (rys. 2). Po naciśnięciu przycisku oznaczonego symbolem mikrofonu należy nagrać swoją wypowiedź i przesłać ją do serwera. Najbardziej prawdopodobny z uzyskanych wyników analizy zostanie wyświetlony w polu w centralnej części strony. Czas trwania całego procesu silnie zależy od możliwości transmisyjnych łącza oraz czasu nagrywania – do serwera są przesyłane pliki audio, których rozmiar może być stosunkowo duży. Sama analiza dźwięku oraz generowanie i przesłanie tekstu przekładu przebiegają już błyskawicznie, dzięki wykorzystaniu olbrzymiej mocy obliczeniowej chmury Google.



Rys. 2. Strona Google z analizatorem mowy dostępna za pomocą przeglądarki Chrome

Pytanie – jak to działa, a także, jak można to wykorzystać z poziomu programu napisanego np. w języku Java? Odpowiedź na pierwszą część pytania jest stosunkowo prosta [10]. Przeglądarka nagrywa dźwięk z mikrofonu, zapisuje go w pliku tymczasowym w formacie FLAC, a następnie przesyła do serwera Google, pod ściśle określony adres. Serwer w odpowiedzi przesyła tekst w formacie JSON [11], zawierający listę najbardziej prawdopodobnych tłumaczeń wypowiedzi, uzupełnioną dla „najlepszego" tłumaczenia o prawdopodobieństwo poprawności analizy:

```
{
  „status": 0,
  „id": „b3447b5d98c5653e0067f35b32c0a8ca-1",
  „hypotheses": [
   {
     „utterance": „i like pickles",
     „confidence": 0.9012539
   },
   {
     „utterance": „i like pickle"
   }]
}
```

Przechodząc do odpowiedzi na drugą część pytania, musimy określić, jak w kodzie programu (podobnie jak poprzednio, posłużymy się językiem programowania Java) wywołać procedury (1) nagrywania dźwięku, (2) przesłania nagrania do serwera Google oraz (3) interpretacji otrzymanego wyniku analizy. Wykorzystamy w tym celu dwie biblioteki Javy, dostępne na zasadach tzw. wolnego oprogramowania: bibliotekę JARVIS [12] oraz konwerter formatu FLAC [9]. Podobnie jak w przykładzie dla biblioteki Sphinx, zdefiniujemy dwie procedury: inicjacji środowiska oraz jego jednokrotnego wywołania:

```
public boolean initializeVoiceGateway(){
  boolean ok=true;
  //inicjacja zmiennych globalnych
  if (mic==null)
    mic = new Microphone(FLACFileWriter.FLAC);
  if (file==null)
    file = new File(configFileName);
    //Nazwa tego pliku jest dowolna – zostanie
    usunięty po zakończeniu analizy
  return ok;
}

public synchronized String invokeOnce(){
  if (invocationStart()==null)
    return null; // gdy nie ma modułu audio –
    wyjście bez wyniku
  // jest moduł audio - nagrywamy dźwięk przez
  kolejne 5 sekund
  try {
    Thread.sleep(5000);
  } catch (InterruptedException ex) {}
  GoogleResponse response=
  invocationStopAndDisplayResults(true);
  if (response==null)
    return null;
  return response.getResponse();
}

public synchronized String invocationStart(){
  try {
    mic.captureAudioToFile(file);
  } catch (Exception ex) {//Brak mikrofonu lub
  jest zajęty przez inny program?
    System.out.println(„Błąd: brak dostępu do
    systemu audio: „+ex.getMessage());
    return null;
  }
  return „";
}
public synchronized GoogleResponse
```

```
invocationStopAndDisplayResults(){
   GoogleResponse response=null;
   mic.close();//koniec nagrania
   // próba kontaktu z serwerem Google
   Recognizer recognizer = new Recognizer(
   Recognizer.Languages.POLISH);
   // wybór języka narodowego w analizie mowy
   try {
      int maxNumOfResponses = 4;
      response = recognizer.
      getRecognizedDataForFlac
      (file, maxNumOfResponses);
      if (response!=null){
      String resultText=response.getResponse();
        if (resultText!=null)
           if (resultText.toLowerCase()
        .contains(„koniec”))
             dialogActive=false;
        System.out.println(„Google Response: „ +
        resultText);
        String confidence=response.getConfiden-
        ce();
        if (confidence!=null)
          System.out.println(
            „Google ma pewność na „ + Double.
             parseDouble(confidence)*100 +
            „% co do poprawności tego tłumacze-
             nia.”);
        boolean possibleOtherResponses=false;
        for(String s: response.getOtherPossibleRe-
        sponses()){
          if (!possibleOtherResponses){
             possibleOtherResponses=true;
             System.out.println(„Inne możliwe tłu-
             maczenia: „);
          }
          System.out.println(„\t” + s);
        }
      }
   } catch (java.lang.Throwable ex) {
        // Obsługa błędu
        System.out.println(„Błąd kontaktu
        z Google: „+ex.getMessage());
        return null;
   }
   file.deleteOnExit();//usunięcie pliku tymcza-
   sowego
   return response;
}
```

Funkcję analizy tekstu (chatterbot) możemy zaprogramować dokładnie w taki sam sposób, jak dla przykładu z wykorzystaniem biblioteki Sphinx. Może tylko warto rozważyć możliwość analizy nie tylko najbardziej prawdopodobnego tłumaczenia, ale sumy słów wynikającej ze wszystkich możliwych tłumaczeń – w powyższym przykładzie do wyniku pamiętanego w zmiennej *response* można sukcesywnie dodać ciągi słów uzyskane w pętli *for* za pomocą wywołania funkcji *getOtherPossibleResponses()*.

Google oferuje ponadto możliwość syntezy dźwięku. Po przesłaniu tekstu serwer Google odsyła plik w formacie MP3, który może być odtworzony za pomocą dowolnego programu interpretującego ten format (w poniższym przykładzie jest to popularna biblioteka JLayer [13]):

```
public static void talk(String text){
   String language = Recognizer.Languages.POLISH.
   toString();
   // wybór języka syntezy dźwięku
   Synthesiser synth = new Synthesiser(language);
   try {
      AdvancedPlayer player;
      // pobranie strumienia danych MP3 z serwera
      Google
      InputStream is = synth.getMP3Data(text);
      // odtworzenie dźwięku nadesłanego z serwera
      player=new AdvancedPlayer(is);
      player.play();
   } catch (Exception e) {
      System.out.println(„BŁĄD: nie można odtwo-
      rzyć danych MP3: „+e.getMessage());
      return;
   }
}
```

Autor przeprowadził szereg eksperymentów dla powyższej metody syntezy oraz analizy dźwięku i był mile zaskoczony zarówno efektywnością analizy (znacznie przewyższającej możliwości środowiska Sphinx), jak i czasem reakcji (po przesłaniu danych odpowiedź jest odsyłana praktycznie natychmiast, bez jakichkolwiek opóźnień). Metoda jest stosunkowo mało wrażliwa na jakość mikrofonu oraz tło dźwiękowe (hałas w pomieszczeniu). Nie bez znaczenia jest fakt, że zarówno analiza, jak i synteza dźwięku mogą być przeprowadzone dla języka polskiego, z pełnym zachowaniem zasad ortografii i interpunkcji. Jedyną wadą podejścia jest konieczność przesłania przez sieć bardzo dużych plików audio, co może wiązać się ze sporymi opóźnieniami w przypadku dostępu mobilnego. Jednakże dla dostępu z typowej sieci stacjonarnej (kablowej lub WiFi) opóźnienia z tytułu transmisji są praktycznie niezauważalne.

6. Wnioski końcowe

Na podstawie powyższej lektury można wysnuć pewne wnioski natury ogólnej. Po pierwsze, rozpoznawanie mowy na potrzeby amatorskie i półprofesjonalne jest obecnie jak najbardziej możliwe i wymaga minimalnych nakładów na sprzęt audio (w zasadzie wystarczy mikrofon i głośnik wbudowany praktycznie w każdy komputer). Po drugie, dla języka programowania Java są dostępne co najmniej dwa rozwiązania niekomercyjne, za które nie trzeba bezpośrednio płacić: biblioteka Sphinx oraz usługa Google Voice Translator dostępna za pomocą biblioteki Jarvis. Google oferuje ponadto możliwość syntezy (i odtworzenia w głośniku komputera) dowolnego komunikatu głosowego na podstawie przesłanego tekstu. Po trzecie, rozpoznanie mowy jest dostępne dla dowolnego mówcy (systemu nie trzeba uczyć wymowy, intonacji, akcentu, definiować słownika itp.)

i przebiega z bardzo dużą skutecznością, która w zupełności wystarcza w większości zastosowań amatorskich.

Aktualnie na pierwszy plan wybijają się dwie propozycje:

- metoda offline z wykorzystaniem biblioteki Sphinx – rozpoznawanie mowy obejmuje w praktyce wyłącznie język angielski[7] i cechuje się średnią skutecznością, wykonanie programu wiąże się z dużą zajętością zasobów komputera (zwłaszcza pamięci, ale także mocy procesora); koszt napisania programu jest praktycznie zerowy, ale optymalizacja jego pracy wymaga szeregu wysiłków i eksperymentów na własną rękę (brak dokumentacji, silna zależność od lokalnego sprzętu, w tym jakości mikrofonu itp.); w podstawowym zastosowaniu Sphinx działa w oparciu o łańcuchy Markowa, zatem należy z góry przewidzieć wszystkie możliwe kombinacje wypowiadanych słów (zdań/poleceń) i zapisać je w postaci mocno sformalizowanej gramatyki języka;
- metoda online z wykorzystaniem usługi Google – usługa umożliwia wykorzystanie praktycznie dowolnego języka narodowego i cechuje się zaskakująco wysoką skutecznością; ma ona duże zapotrzebowanie na szybki kanał komunikacyjny od urządzenia mobilnego do serwera, co raczej kieruje ją w stronę sieci stacjonarnych i WiFi, a nie sieci mobilnych (szerokopasmowy kanał wyjściowy nie jest normą nawet dla najnowszego systemu LTE); dużą zaletą usługi jest bardzo niska zajętość zasobów komputera – pamięci i czasu procesora, okupiona jednak wysokim wykorzystaniem łącza do sieci rozległej, oraz bardzo niski (w zasadzie pomijalny) koszt napisania oprogramowania, nie trzeba też definiować gramatyki języka, czyli z góry ograniczać analizowanych komunikatów słownych.

Należy tutaj jednak zaznaczyć, że nie wiemy, co dokładnie firma Google robi z przesyłanymi do serwera danymi. Jeśli służą wyłącznie do wzbogacenia korpusu języka, to jest to jak najbardziej pożądane, ale jeśli dodatkowo służą one do realizacji celów biznesowych firmy bez informowania o tym użytkowników (śledzenie i profilowanie marketingowe), to już nie jest to komfortowa i pożądana przez użytkowników sytuacja. Analogia do pamiętania i procesowania prywatnych zapytań do wyszukiwarki Google oraz prywatnych wiadomości w poczcie Gmail jest tu jak najbardziej na miejscu. Ponieważ usługa ma ciągle charakter nieoficjalny, nie znamy modelu biznesowego firmy Google w zakresie rozpowszechniania narzędzia rozpoznawania mowy oraz nie potrafimy obecnie określić, na jakich zasadach (i czy w ogóle) będzie ono powszechnie dostępne.

Na podstawie porównania zalet i wad obu rozwiązań jest widoczne, że naturalnym wyborem dla większości zainteresowanych będzie raczej korzystanie z usługi Google, o ile będzie zapewniony szerokopasmowy dostęp do sieci rozległej. W przypadku pracy offline lub ograniczonego kanału dostępowego do sieci niestety jesteśmy niejako skazani na korzystanie z biblioteki Sphinx i (przynajmniej na razie) komunikację słowną z komputerem w języku angielskim.

Przypisy

1. W rzeczywistości proces analizy fonemów i składania z nich słów jest procesem bardzo skomplikowanym, a opracowanie odpowiedniego oprogramowania zajęło długie lata. Jednakże, z punktu widzenia dzisiejszego programisty, który otrzymuje gotową bibliotekę i wykorzystuje ją na zasadzie „czarnej skrzynki", proces ten przebiega w przybliżeniu tak, jak to opisano w niniejszym tekście (aczkolwiek okno czasowe analizy jednego elementu to nie pojedyncze słowo, tylko określony czas, zwykle kilkanaście ms).
2. Zakładamy, że każde polecenie zapisujemy w osobnym wierszu.
3. W rzeczywistej notacji BNF symbole znakowe mają nieco bardziej skomplikowaną postać, która jednak silnie zależy od implementacji (oprogramowania) – w tekście wykorzystujemy notację uproszczoną.
4. Jak już wcześniej powiedziano – jest to uproszczenie przyjęte dla potrzeb tego artykułu, w rzeczywistości analizujemy w ten sposób poszczególne fonemy, czyli fragmenty wypowiadanych słów.
5. Czytelnikowi pozostawiamy odkrycie różnic między formatem JSGF wykorzystanym w niniejszym przykładzie i wcześniej wprowadzonym uproszczonym formatem BNF.
6. Uwaga – zawartość strony jest w pełni dostępna wyłącznie dla przeglądarki Chrome, w przypadku innych przeglądarek na wskazanej stronie pojawia się komunikat o niemożności uruchomienia systemu analizy mowy.
7. Chyba, że ktoś zechce skorzystać z narzędzi biblioteki Sphinx i przygotować własny słownik – wiąże się to jednak z bardzo dużym nakładem pracy.

Literatura

[1] Ghai W., Singh N.: *Literature Review on Automatic Speech Recognition*. International Journal of Computer Applications, vol. 41, no. 8, 2012, pp. 42–50.

[2] What is BNF notation?, http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html.

[3] Ghahramani Z.: *An introduction to Hidden Markov Models and Bayesian Networks*. mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf.

[4] N-gram – model językowy stosowany w rozpoznawaniu mowy, http://pl.wikipedia.org/wiki/N-gram.

[5] Home Page of The Loebner Prize in Artificial Intelligence, http://www.loebner.net/Prizef/loebner-prize.html.

[6] Rykowski J.: *Using software agents to personalize natural-language access to Internet services in a chatterbot manner*. 2nd Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Lingusitics, ed. Zygmunt Vetulani, 2005, s. 269–273.

[7] Strona CMUSphinx.sourceforge.net, „Basic Concepts of Speech", http://cmusphinx.sourceforge.net/wiki/tutorialconcepts.

[8] Opis języka definiowania gramatyki JSGF, http://www.w3.org/TR/2000/NOTE-jsgf-20000605/.

[9] FLAC – free lossless audio codec, https://xiph.org/flac/features.html.

[10] Pultz M.: *Accessing Google Speech API / Chrome 11*, http://mikepultz.com/2011/03/accessing-google-speech-api-chrome-11/.

[11] ECMA-404 The JSON Data Interchange Standard, http://www.json.org/.

[12] Strona domowa biblioteki JARVIS, https://github.com/The-Shadow/java-speech-api.

[13] Oficjalna strona projektu JLayer, http://www.javazoom.net/javalayer/javalayer.html.

Jarogniew Rykowski – Katedra Technologii Informacyjnych; Uniwersytet Ekonomiczny w Poznaniu; e-mail: rykowski@kti.ue.poznan.pl

artykuł recenzowany