Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jure Frantar

Samodejno razpoznavanje govora v bolnišnicah

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Robert Rozman
Somentor: Andraž Krajnc, univ. dipl. inž.

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Podobno kot sodobne javne in zasebne ustanove uporabljajo tudi bolnišnice, porodnišnice ter domovi za starejše občane vedno boljšo tehnologijo s ciljem zagotavljanja čim bolj učinkovite in cenovno vzdržne oskrbe. Prav tako se želi zdravnikom, medicinskim sestram ter ostalim zaposlenim v omenjenih ustanovah olajšati delo ter jim omogočiti čim boljše delovne pogoje. V zadnjem času se zato v nekatere ustanove vgrajujejo sobni terminali na dotik in uporabljajo prenosni telefoni zaposlenih za učinkovitejšo oskrbo bolnikov. Pri svojem delu obstoječo aplikacijo za sobne terminale nadgradite z možnostjo samodejnega razpoznavanja govora in s tem izvajanje še učinkovitejše in pacientom prijaznejše oskrbe. Omogočite zaposlenim in bolnikom učinkovito komunikacijo s sistemom tudi s pomočjo najnaravnejše oblike komunikacije - govora. S tem načinom uporabe ni več potrebno ročno upravljanje terminalov, kar poenostavi delovni proces v teh ustanovah. Zagotovite tudi možnost nadaljnjih izboljšav in nadgradenj sistema.

Največja zahvala gre mentorju Robertu Rozmanu, ki mi je bil kot svetovalec za pridobivanje novih idej in kot pomoč pri izdelavi ter končnemu pregledu diplomske naloge.

Poleg tega bi se zahvalil tudi mojemu somentorju Andražu Krajncu.

Vsekakor pa bi se zahvalil tudi staršem, ki so mi dajali motivacijo tekom celotnega študija ter izdelavi diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application programming interface	Vmesnik za programiranje aplikacij
G2P	Grapheme to phoneme / Tool for adding words into dictionary	Orodje za dodajanje besed v slovar
NurseApp	Smart phone application	Aplikacija za pametni telefon
LSTM	Type of neural network	Vrsta nevronske mreže
NLA	NurseLog Server	NurseLog strežnik
NT	NurseTab	Sobni terminal na dotik
WIFI	Wireless internet connection	Brezžična internetna povezava

Povzetek

Naslov: Samodejno razpoznavanje govora v bolnišnicah

Avtor: Jure Frantar

Zaradi lažje organizacije dela ter večje preglednosti nad delom zaposlenih v medicinskih ustanovah že nekaj časa vgrajujemo Android sobne terminale, občutljive na dotik. Slednji imajo nameščene prilagojene aplikacije, ki služijo kot pomoč pri delu v medicinskih ustanovah. Njihova glavna naloga je prikaz aktivnih dogodkov, informacij ter beleženje storitev, ki jih zaposleni opravijo bolnikom. Zaposleni pa med opravljanjem storitev velikokrat potrebujejo pomoč dodatne sestre ali zdravnika. Zaradi lažjega opravljanja storitev brez prekinitev dela ob klicu dodatnega zaposlenega smo funkcionalnost terminalov nadgradili z razpoznavanjem govora, ki ga podrobneje opisujemo v tem delu. Za zaznavanje ključne besede smo uporabili orodje CMUSphinx, za razpoznavanje ukazov uporabnikov pa orodje Android.SpeechRecognizer. Postopek za samodejno tvorbo govornih odzivov na ukaze uporabnikov pa smo zasnovali na orodju TextToSpeech. Poleg zaposlenih pa lahko terminal uporabljajo tudi bolniki. V angleškem jeziku lahko izrazijo svoje želje zaposlenim, kateri lahko pridejo v sobo že ustrezno pripravljeni. Z dodano funkcionalnostjo smo poenostavili delo zaposlenih in izboljšali delovne procese v zdravstvenih ustanovah.

Ključne besede: sobni terminal na dotik, mobilna aplikacija NurseApp, razpoznavanje govora, medicina.

Abstract

Title: Automatic Speech Recognition in Hospitals

Author: Jure Frantar

To achieve the easier work organization and greater transparency over the work of medical staff, we have been installing touch-sensitive Android terminals in medical institutions for quite some time. The terminals have installed customized applications that serve as an assistance to the working process. Their main task is to present actual events, information and record the services that employees provide to patients. However, when providing services, employees often need the help of an additional employee. In order to facilitate the provision of services without the interruption due to the call of an additional employee, the functionality of the terminals was upgraded with the speech recognition, which is described in more detail in this work. We have based our work on the CMUSphinx tool to identify the spoken keywords and the Android.SpeechRecognizer tool to recognize commands and random speech of terminal users. We have implemented the automatic spoken responses to user commands with the TextToSpeech tool. In addition to employees, the terminals can also be used by patients. They can express their wishes in the spoken English language, and employees can come to the room already well prepared. With added functionality, we have simplified the work of employees and therefore improved work processes in medical institutions.

Keywords: room touch terminals, mobile application NurseApp, speech

recognition, medicine.

Poglavje 1

Uvod

Kot povsod drugje po svetu se tudi v bolnišnicah, porodnišnicah in ostalih medicinskih ustanovah uporablja vse boljša tehnologija. Le ta nam veliko pripomore k pomoči pri delu zaposlenih, poleg tega pa želijo imeti zaposleni na višjem nivoju čim boljši nadzor nad delom ostalih zaposlenih. Zato v zgoraj omenjene ustanove že nekaj časa vgrajujemo sobne terminale na dotik. Omogočajo nam, da uporabnik sproži klic sestre, zaposlen pa lahko shrani storitve, katere je opravil bolniku ali pa sproži klic dodatnega zaposlenega, v kolikor ga potrebuje. Klici se tako vidijo na ostalih terminalih, na katerih so z identifikacijskim geslom prijavljeni zaposleni v sobi, prav tako pa si zaposleni lahko na terminalu ogleda zgodovino opravljenih storitev. Več podrobnosti o delovanju sistema si lahko preberemo v 2. poglavju.

Nekaj časa nazaj smo prejeli izziv, da bi programsko opremo sobnih terminalov zaradi lažjega dela zaposlenih nadgradili z govornim razpoznavanjem. Izziv smo sprejeli kot izdelavo nove funkcionalnosti. Nova funkcionalnost precej izboljša delo zaposlenih, kateri opravljajo storitve bolnikom, pri tem pa opazijo, da potrebujejo pomoč dodatnega zaposlenega. Pred implementacijo razpoznavanja govora so se zaposleni morali sprehoditi do terminala, nameščenega ob postelji ter klikniti na sprožitev klica dodatnega zaposlenega. Sedaj lahko zaposlen to opravi med opravljanjem storitev, saj terminal lahko pokliče s pomočjo razpoznave govora. Poleg tega bolnik lahko v angleškem

jeziku pove, kaj želi od zaposlenega, sporočilo bolnika pa se skupaj s t.i. sestrskim klicem pošlje na druge naprave. S tem smo zaposlenemu omogočili neprekinjeno opravljanje dela, poleg tega pa smo bolniku omogočili, da pove zaposlenemu, kaj od njega želi. Tako zaposleni že ve, kaj želimo od njega, zato bo lahko prišel v sobo že pripravljen.

Ko pomislimo na razpoznavo govora (Speech recognition), se kot uporabniki mobilnih telefonov in drugih sodobnih naprav najprej spomnimo na Googlov Speech recognition (OK Google) [13]. Zelo znana je tudi Microsoftova Cortana [8]. Prav tako so precej znani tudi nekateri drugi sistemi za razpoznavo govora, kot so Microsoftova naprava Kinect [16], ki je imela za svoje čase implementiran zelo dober sistem za razpoznavo govora, prav tako je precej napreden tudi Amazonov Echo zvočnik (Alexa) [1].

Na spletu lahko najdemo nekaj najbolj znanih sistemov za razpoznavo govora. Pri tem smo pozorni na možnost implementacije na Android napravah (sobnih terminalih), katere uporabljamo v medicinskih ustanovah.

Vsi uporabniki pa želimo imeti sisteme čimbolj preproste in avtomatizirane, zato smo naš sistem nadgradili z novo funkcionalnostjo - razpoznavo govora. Uporabili smo 3 orodja. Za razpoznavanje ključne besede smo uporabili orodje CMUSphinx. Vsebuje več orodij, ki služijo za različne namene. Uporabili smo orodje imenovano PocketSphinx [23]. Le ta nam omogoča, da terminal lahko pokličemo s ključnimi besedami, katere razpozna s pomočjo že implementiranih metod ter nam vrne rezultat besed, katere je razpoznal. Besede lahko primerjamo z našo ključno besedo in če se besedi ujemata pomeni, da je terminal prepoznal svoj klic.

Zaposleni pa med opravljanjem storitev nimajo časa gledati na zaslon terminala, če je orodje PocketSphinx pravilno razpoznal ključno besedo. S tem namenom smo uporabili orodje TextToSpeech, s katerim se s pomočjo umetno tvorjenega govora terminali odzovejo na uporabnikove ukaze.

Kot uporabniki pa lahko terminalu podamo različne ukaze, katere lahko izvrši z razpoznavo govora. Zato mu moramo povedati, kaj želimo od njega. Tukaj nam je prav prišlo orodje Android.SpeechRecognizer [5]. Le ta je na-

menjen poslušanju naključno dolgega besedila, katerega uporabnik kot ukaz poda terminalu. Naključno besedilo nato pretvori v tekstovno besedilo in iz njega poskuša razbrati, kaj uporabnik želi povedati oziroma ukazati. Orodje nam vrne seznam možnih besednih zvez, katere se ujemajo z izgovorjenim besedilom. Za orodje SpeechRecognizer je značilno, da je programsko določen začetek poslušanja uporabnika.

V 2. poglavju si bomo pogledali delovanje terminalov ter naloge strežnika, omenili pa bomo tudi uporabo mobilnih telefonov s prilagojeno aplikacijo imenovano NurseApp, katero zaposleni uporabljajo za prikaz aktivnih dogodkov.

V glavnem delu si bomo pogledali implementacijo ter uporabo zgoraj omenjenih orodij za razpoznavo govora ter prednosti/slabosti orodij. Delo z orodjem CMUSphinx bomo opisali v 3. poglavju, z orodjem Android. SpeechRecognizer pa v 5. poglavju. Poleg tega bomo v poglavju 4 opisali tudi naše delo z orodjem TextToSpeech, katerega bomo uporabili za tvorbo samodejnega govornega odziva terminala. V 6. poglavju pa si bomo pogledali vse pomanjkljivosti projekta ter težave, na katere smo naleteli med delom. Ne bomo pa pozabili omeniti tudi rezultatov testiranja aplikacije ter idej za nadaljnje delo s področja razpoznave govora.

Poglavje 2

Delovanje sistema in povezanih naprav

V tem poglavju bomo opisali delovanje ter trenutno že implementirane naloge našega terminala, da si bomo lahko lažje predstavljali zastavljen cilj, katerega bomo opisali v diplomski nalogi. Poleg tega bomo omenili tudi aplikacijo za prenosne telefone, katero uporabljajo zaposleni ter naloge našega strežnika.

2.1 Opis delovanja ter povezave naprav v sistemu

Sobni terminal na dotik je posebna naprava, namenjena za uporabo v bolnišnicah, porodnišnicah in ostalih medicinskih ustanovah, ki je nameščena na steni blizu postelje bolnika. Deluje na osnovi operacijskega sistema Android. Uporabljajo ga večinoma zaposleni, posredno pa tudi bolniki. Preko dodatnih vhodov so žično povezani tudi s klicnimi vrvicami v toaletnih prostorih ali kopalnicah ter poteznimi gumbi pri postelji, katere skrbijo za sprožitev klica medicinske sestre (t.i. sestrskega klica). S pomočjo klicne vrvice ter poteznega gumba se bolniku ni potrebno sprehoditi do terminala, da bi sprožil sestrski klic.

Zaposleni terminal lahko uporabljajo kot klic dodatne sestre (t.i. klic

asistence) ali zdravnika. Uporablja se tudi za beleženje storitev, katere zaposleni opravijo bolnikom. Vsak terminal ima shranjeno zgodovino opravljenih storitev za vsakega bolnika, tako imajo zaposleni možnost pregleda zgodovino narejenih storitev. Na terminalu so zapisane tudi vse pomembne informacije bolnika, kot so alergije, informacije o prehrani, težave bolnika, njegovo zdravstveno stanje ter ostale pomembne informacije, katere zaposlen želi vedeti, da lahko trenutno zdravstveno stanje primerja s prejšnjim.

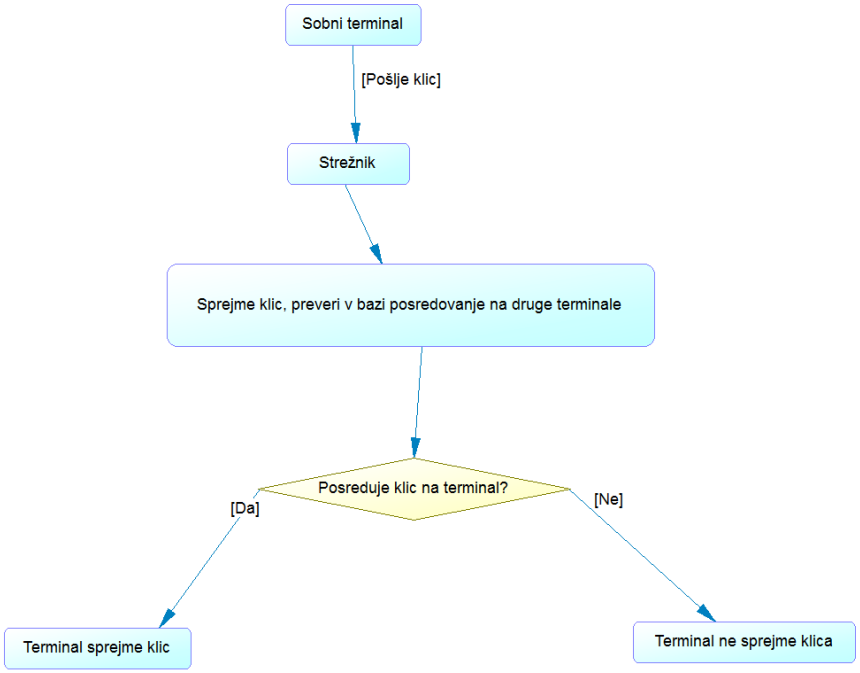
Slika 2.1: Nameščen sobni terminal v bolnišnici ter potezni gumb pri postelji.

Vsak terminal je z žično ali brezžično povezavo povezan s strežnikom, kateri v podatkovno bazo zapisuje zabeležene sestrske klice ter njihove odzivne čase, vse opravljene storitve ter meritve, kot so temperatura bolnika, izmerjen sladkor, krvni pritisk,... Velikokrat so terminali žično povezani tudi z LED diodami pred sobami bolnikov, da zaposleni lahko takoj vidijo, iz katere sobe je prišel sestrski klic, uporabniki pa takoj vidijo, v kateri sobi se trenutno nahaja zaposlen.

Sestrski klic deluje tako, da bolnik ali zaposlen sproži klic na terminalu (lahko tudi s poteznim gumbom ob postelji ali s potegom klicne vrvice v toaletnih prostorih). Terminal zahtevo za klic pošlje strežniku ter prižge LED diodo, v kolikor je ta nameščena in povezana pred sobo bolnika.

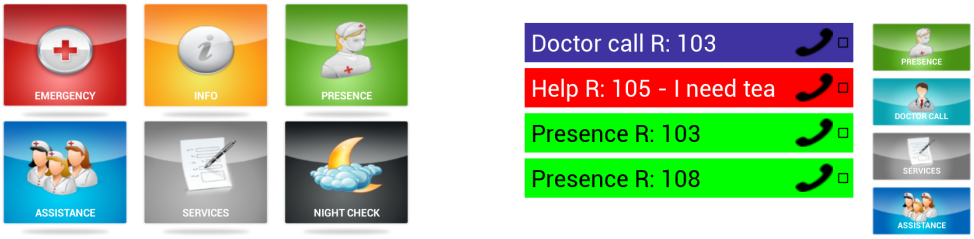
Strežnik ima v podatkovni bazi shranjene podatke vseh naprav, ki so v medicinski ustanovi. V njej preveri, na katere naprave posreduje klic (prenosni telefoni z NurseApp aplikacijo ter terminali). Strežnik naredi seznam vseh naprav, katerim posreduje klic, nato pa napravam pošlje zahtevo za prikaz se-

strskega klica. Seznami (plani) posredovanja klicev se posredujejo dinamično glede na praznike, dan v tednu, trenutnim časom, nadstropjih ter oddelkih v ustanovah. Tako si lahko popolnoma prilagodimo posredovanje sestrskih klicev ter ostalih dogodkov po naših željah. Ko je zaposlen v določeni sobi, se mora prijaviti na terminal z identifikacijskim geslom, terminal pa s strežnika začne pridobivati vse aktivne dogodke sosednjih sob, katere so določene v zgoraj omenjenih planih. Tako lahko vsak zaposlen, ki je trenutno prijavljen v eni izmed sosednjih sob vidi, da nekdo v bližini njega potrebuje pomoč.



Slika 2.2: Potek posredovanja klicev.

Na terminalu lahko izbiramo med dvema različnima prikazoma programa. Glavni zaslon (Slika 2.3a) služi za vpisovanje storitev ter sprožitev sestrskega klica, sestrski zaslon (Slika 2.3b) pa se uporablja v sestrskih sobah in je namenjen prikazu vseh aktivnih dogodkov.



(a) Glavni zaslon (b) Sestrski zaslon

Slika 2.3: Vrsti prikaza zaslonov na terminalu.

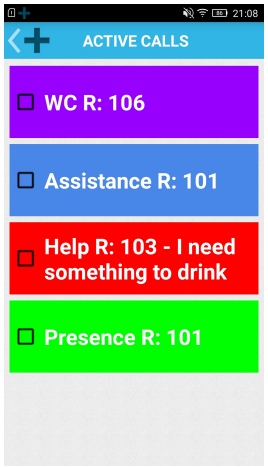
Na sliki 2.3b lahko vidimo, da je stanovalec v sobi 105 uporabil razpoznavo govora, katerega smo v naši diplomski nalogi implementirali kot novo funkcionalnost. V angleškem jeziku je povedal terminalu, da potrebuje čaj. Tako lahko z novo funkcionalnostjo zaposlen pride v sobo že pripravljen (tokrat z vročim čajem). Poleg dogodka v sobi 105 so zaposleni prijavljeni na terminalu tudi v sobi 103 ter 108. V sobi 103 zaposleni potrebuje pomoč zdravnika.

2.2 Opis postopka delovanja sestrskega klica

Ko zaposleni vidi aktiven klic, pride do sobe bolnika in se s pomočjo identifikacijskega gesla prijavi v terminal. S tem se aktiven klic spremeni v neaktivnega, zaposleni pa nato nudi pomoč bolniku. V kolikor zaposlen potrebuje pomoč še enega zaposlenega, lahko sproži klic asistence (klic dodatne sestre) ali pa klic zdravnika. Z novo funkcionalnostjo lahko uporabnik klic sproži tudi oddaljeno s pomočjo razpoznave govora.

Zaposleni pa za prikaz aktivnih dogodkov uporabljajo tudi mobilne telefone s prilagojeno aplikacijo, imenovano NurseApp, katera je povezana s

strežnikom preko WIFI povezave. Strežnik telefonu ob prijavi zaposlenega pošlje podatke o dogodkih (klicih), kakor je prikazano tudi na terminalu v sestrski sobi. Zaposleni klica sicer ne morejo sprejeti preko pametnega telefona, pomaga pa jim k hitrejšemu vpogledu na vse aktivne dogodke v ustanovi. Na sliki 2.4 si lahko ogledamo prikaz aktivnih dogodkov na pametnem telefonu.



Slika 2.4: Prikaz klicev na pametnih telefonih.

Na strežnikih se izvaja Windows storitev v ozadju sistema, katera nam med drugimi omogoča tudi:

- Komunikacijo in pošiljanje nalog terminalom ter prenosnim telefonom.
- Shranjevanje klicev preko SQL strežnika, vpogled na odzivne čase zaposlenih glede na klice in čase prihodov ter odhodov po sobah.
- Shranjevanje opravljenih storitev bolnikom.
- Sestavo poročil glede dela zaposlenih.
- Samodejno dnevno, tedensko ali mesečno posredovanje sestavljenih poročil storitev preko e-pošte.
- Natančen pregled nad delom zaposlenih.
- Pregled nad delovanjem / nedelovanjem naprav.
- Evidenco plačljivih storitev bolnikov z dodatnim delom zaposlenega.

Poglavje 3

Zaznava ključne besede z orodjem CMUSphinx

Kot smo že v uvodu omenili, smo za razpoznavanje govora uporabili orodje CMUSphinx. Je napredno orodje, ki se uporablja za neprekinjeno razpoznavanje že v naprej znanih besed in stavkov. Uporabili ga bomo za razpoznavo ključnih besed [18], za katero Googlov Home zvočnik [12] uporablja "OK Google". Amazonov Echo zvočnik [1] pa za ključne besede uporablja ime "Alexa".

Ključna beseda je beseda ali besedna zveza, katero uporabimo za klic terminala, le ta pa potem začne poslušati uporabnikov ukaz.

CMUSphinx vsebuje dve različni orodji. Razlikujeta se predvsem v številu nastavitev, katere lahko ročno nastavljamo ter v implementaciji. Poleg tega lahko spreminjamo natančnost razpoznave besed [17]:

- **PocketSphinx** orodje je napisano v C programskem jeziku. Je trenutno najhitrejše in najpogosteje uporabljeno CMUSphinx orodje za razpoznavo govora. Uporabljamo ga lahko v namiznih aplikacijah, Android in Apple napravah, starejših Nokia dlančnikih ter na malo zastarelem sistemu Windows mobile [20]. Uporablja se za razvijalce z manj izkušnjami glede razpoznavanja govora in modeliranju zvočnih posnetkov, tako lahko na enostaven način z manj nastavitvami imple-

mentiramo dovolj dober produkt z implementacijo razpoznave govora. Prav tako pa želimo, da je uporaba le tega možna na različnih napravah (hitrost in prenosljivost) [9].

- **Sphinx4** orodje je napisano v Java programskem jeziku. Uporablja se za zahtevnejše in kompleksnejše namene. Tukaj je na voljo veliko več nastavitev za razpoznavo govora. Namenjeno je predvsem zahtevnejšim programerjem, ki imajo veliko izkušenj z razpoznavo govora in modeliranjem zvočnih posnetkov, saj vsebuje veliko parametrov, katere moramo ročno nastaviti pri razpoznavi govora (fleksibilnost in upravljanje ter prilagodljivost in spremenljivost) [9].

Za začetek smo poskusili uporabiti orodje PocketSphinx, ki je zaradi manj nastavitev, nekompleksnosti in lažje prenosljivosti med napravami v našem primeru boljše za implementacijo ter uporabo. Prav tako smo sčasoma ugotovili, da orodje PocketSphinx vsebuje dovolj nastavitev in parametrov, katere potrebujemo za naše nadaljnje delo. Orodje je dovolj napredno, da nam omogoča spreminjanje naše ključne besede, prav tako pa lahko orodju dodajamo besede v naš osebni slovar, omogoča pa tudi implementacijo novega slovarja z drugim jezikom. O podrobnostih dodajanja in izdelave novega slovarja bomo govorili v poglavju 3.3.

3.1 Testiranje orodja PocketSphinx

Kot pri vsaki novi funkcionalnosti, katero želimo implementirati, si je tudi tukaj na začetku potrebno pogledati, kako natančno je izdelana dokumentacija, nato pa pogledati nekaj vodičev. V kolikor vidimo, da je virov učenja veliko in so nam razumljivi, lahko začnemo z delom ter testiranjem demonstracijskega primera. CMUSphinx nam omogoča, da si preko Github spletnega strežnika prenesemo demonstracijski projekt, v katerem si lahko pogledamo osnovno delovanje PocketSphinx orodja [19]. Spoznamo lahko osnovne metode ter nastavitve parametrov za razpoznavo naše ključne besede. Na začetku si ogledamo osnovno delovanje demonstracijskega programa ter samega orodja.

Ko uvozimo demonstracijski projekt v Android studio in se projekt uspešno naloži, imamo na začetku možnosti razpoznavanja govora, števil, vremenskih napovedi ali telefonskih številk. Tukaj nas najbolj zanima iskanje besed ter besednih zvez. Ostale možnosti razpoznavanja nas trenutno ne zanimajo, zato jih lahko zakomentiramo.

```
private static final String KWS_SEARCH = "wakeup";
// private static final String FORECAST_SEARCH = "forecast";
// private static final String DIGITS_SEARCH = "digits";
// private static final String PHONE_SEARCH = "phones";
// private static final String MENU_SEARCH = "menu";
```

V naslednjem odstavku demonstracijskega programa imamo spremenljivko poimenovano KEYPHRASE. V spremenljivki je shranjena ključna beseda s katero pokličemo terminal. Po želji ključno besedo lahko spremenimo. Odločili smo se, da bomo terminalu dali ime Carmen, zato spremenimo spodnjo vrednost v spremenljivki.

```
private static final String KEYPHRASE = "Wakeup Carmen";
```

Pri spreminjanju ključne besede moramo biti pozorni, da se vse besede v nizu poimenovanem KEYPHRASE nahajajo v slovarju. Odpremo ga lahko v Android projektu v mapi *assets/sync/cmudict-en-us.dict*. Slovar je po abecednem redu urejena zbirka besed, katero sistem lahko prebere in jo primerja z našim govorom. O dodajanju besed v obstoječi ter izdelavi novega slovarja si bomo ogledali v poglavju 3.3.

Poleg tega moramo v programu spremeniti parameter dolžine ključne besede. Določen je v spremenljivki *.setKeywordThreshold*, ki se nahaja v metodi *setupRecognizer*. Več o tem si lahko pogledamo v poglavju 3.2.4.

Spremenimo vse zgornje lastnosti ter zaženemo program. Ko našo napravo pokličemo ter program prepozna govor podobne dolžine, se nadaljuje s klicem metode *onPartialResult*. Znotraj te metode lahko primerjamo našo izgovorjeno besedo s ključno besedo. V kolikor sta besedi enaki, se izvajanje

programa nadaljuje, v kolikor pa različni, se program pripravi na ponovno poslušanje ključne besede s klicanjem metode *switchSearch*, kateri kot parameter podamo ključno besedo.

3.2 Vključitev orodja PocketSphinx

Kot smo že na začetku 3. poglavja povedali, CMUSphinx vsebuje dve orodji. Za začetek nam je pomembno, da orodje uspešno integriramo v naš program in preizkusimo uporabo orodja CMUSphinx v našem programu, kateri se uporablja na terminalih. Za začetek smo se odločili, da preizkusimo orodje PocketSphinx, saj nam mogoče že ta ponuja dovolj natančno razpoznavo govora za implementacijo nove funkcionalnosti. Glede nastavitev razpoznave govora se predvsem osredotočimo na motnje zvokov v ozadju ter raznih drugih šumov naprav, katere so nameščene zraven mikrofona v medicinskih ustanovah.

PocketSphinx je orodje, katerega lahko uvozimo s pomočjo demonstracijskega primera, katerega smo opisali v prejšnjem podpoglavju. Orodje najprej testiramo na malo spremenjenem demonstracijskem programu, nato pa ga uvozimo v naš projekt, kar si bomo pogledali v naslednjem podpoglavju.

3.2.1 Vključitev slovarja

V Android studiu odpremo naš projekt. Pregledamo dokumentacijo CMUSphinx ter PocketSphinx za uvoz orodja v projekt [15].

Začeli bomo s pravicami, katere naša aplikacija potrebuje, da bo PocketSphinx deloval pravilno ter brez težav. Za shranjevanje slovarja na naš terminal potrebujemo pravice za pisanje datotek. To storimo tako, da v datoteko AndroidManifest.xml dodamo pravice za shranjevanje datotek.

```
<uses-permission android:name="android.permission.
WRITE_EXTERNAL_STORAGE" />
```

Ker je pa na terminalih določena verzija operacijskega sistema Android, moramo poleg pravic za shranjevanje datotek dodati tudi pravice za snemanje zvoka(omogočimo pravice mikrofona). To storimo enako kot zgoraj, vendar uporabimo naslednjo pravico:

```
<uses-permission android:name="android.permission.
RECORD_AUDIO" />
```

V prejšnjem odstavku smo dodelili pravice za dostop do mikrofona ter dostop do slovarja, sedaj pa lahko povemo nekaj več o uvozu slovarja v naš projekt.

Standardno imamo v Android projektih narejeno Assets mapo, znotraj mape imamo slovarje. V našem primeru pa to ni dovolj, saj mora naš projekt zaradi uvoženega orodja PocketSphinx vedno imeti navedeno fizično pot do slovarja. To pomeni, da mora biti slovar shranjen nekje v zunanjem pomnilniku, da lahko orodje PocketSphinx dostopa do njega. V projektu najprej ustvarimo mapo *Assets*.

Mapa *Assets* nam omogoča samodejno kopiranje slovarja v zunanji pomnilnik. Kot navajajo na CMUSphinx spletni strani, nam razred *pocketsphinx.Assets#syncAssets* omogoča branje virov z datoteke *assets.lst*, katera se nahaja v korenski mapi [24]. Razred *Assets* poskrbi, da se datoteke s slovarjem kopirajo le, če so se spremenile ali pa datoteke v zunanjem pomnilniku še ne obstajajo. V kolikor se skripta kopiranja datotek ni pravilno prevedla, moramo sami poskrbeti za ustvarjanje datoteke *md5* in *assets.lst* v zunanjem pomnilniku [24].

Slovar uvozimo tako, da z demonstracijskega PocketSphinx projekta kopiramo *models/assets.xml* datoteko v naš projekt v enako mapo *app*. Nato prenesemo mapo *sync* z demonstracijskega projekta v mapo *assets* v našem projektu.

V datoteki *app/build.gradle* dodamo nastavitve za prevajanje slovarja [23].

```
ant.importBuild 'assets.xml'
preBuild.dependsOn(list, checksum)
clean.dependsOn(clean_assets)
```

Sedaj lahko preverimo, če se je *assets.lst* datoteka kreirala in *md5* datoteka posodobila.

3.2.2 Uvoz orodja PocketShinx v projekt

V demonstracijskem primeru imamo že priloženo datoteko imenovano *pocketsphinx-android-5prealpha-nolib.jar*. Datoteko najprej skopiramo v naš projekt (mapa *app/libs/*). V Android Studiu v menijski vrstici izberemo možnost "datoteka"(file), nato "struktura projekta"(project Structure) in izberemo še "odvisnost"(dependencies). Tu lahko dodamo jar datoteko. Sedaj lahko uporabljamo orodje *PocketSphinx* v našem projektu.

Ko v naš projekt uvozimo orodje z *Assets* mapo, imamo na razpolago že slovar z angleškimi izrazi. Med besedami lahko izberemo skupino besed, katero bomo uporabili za ključno besedo ali pa v projektu v angleškem slovarju dodamo svojo besedo, v kolikor še ni dodana. V slovarju je sicer že zbrana precej velika množica besed, katera nam lahko pomaga pri izbiri ključnih besed. S pomočjo že napisanih besed pa v slovarju lahko ustvarimo kakšno novo besedo, v kolikor bi to potrebovali. Več o tem si lahko pogledamo v poglavju 3.3.1.

3.2.3 Prilagoditev programa za delovanje v ozadju

Ker imamo v našem primeru že narejeno aplikacijo za terminal, katera deluje neprekinjeno, je zelo nesmiselno, da bi naredili še eno aplikacijo, katera bo odgovorna le za razpoznavo govora. Zato v glavnem programu naredimo še podprogram, ki se izvaja v drugem procesu in pri tem obremenjuje zelo maj-

hen del glavnega programa. Tukaj govorimo o Android background service (Androidovi funkciji, ki se izvaja v ozadju programa) [14].

Android background service ustvarimo preko našega glavnega programa. Najprej v datoteki AndroidManifest dodamo vrstico, kjer povemo ime našega razreda s PocketSphinx programom.

```
<service android:name="com.something.PocketSphinx" />
```

V naš glavni program uvozimo naše orodje oziroma razred ter dodamo zagon našega programa za razpoznavo govora.

```
import com.something.PocketSphinx;
startService(new Intent(this, PocketSphinx.class));
```

V našem programu moramo spremeniti še nekaj kode, da bo le ta nemoteno deloval v ozadju. Tako razredu spremenimo prvo vrstico ter mu dodamo vse potrebne metode za zagon razreda:

```
public class PocketSphinx extends Service implements
RecognitionListener {
@Override
    public int onStartCommand(Intent intent, int flags,
        int startId) {
        return START_STICKY;
    }
}
```

Spremenljivka START_STICKY omogoča operacijskemu sistemu, da prekine izvajanje programa, v kolikor nima na razpolago dovolj pomnilnika ter se znova zažene, ko je določen odstotek pomnilnika zopet prost. Sedaj imamo vse, kar je potrebno za zagon programa, ki deluje v ozadju.

Program pa zaradi delovanja v ozadju ne more direktno dostopati do zaslona na naši napravi, tako ne moremo izpisati Toast pojavnih oken Android sistema. To so pojavna okna, katera se prikažejo za določen čas. Aplikaciji

smo dodali izpise v obliki dnevnika (log files) in videli, kako deluje aplikacija brez potrebe dostopa do glavnega zaslona.

Da bi izpisali Androidova pojavna okna s pomočjo razreda Toast, preko aplikacije v ozadju, je najbolje uporabiti razred Handler [2]. Le ta nam omogoča pošiljanje sporočil, ki se odvija neodvisno od programa. Uporablja svojo procesorsko nit, preko katere izpisuje vsa sporočila, ki morajo biti prikazana na zaslonu.

Ker kot uporabniki želimo, da se sporočila prikazujejo po določenem vrstnem redu, je razred zgrajen tako, da mu lahko naenkrat pripada le ena nit. Razred je na voljo, dokler mu ne zmanjka sporočil, katere mora izpisati na zaslon. Ko se na zaslonu prikaže zadnje sporočilo, se instanca razreda Handler uniči.

3.2.4 Preizkus demonstracijskega primera v našem projektu

Sedaj, ko imamo na voljo PocketSphinx orodje z njegovimi metodami, lahko preizkusimo funkcionalnost izvajanja programa v ozadju z demonstracijskega programa, katerega smo v preteklosti dogradili v naš projekt. Tako lahko preizkusimo tudi praktično, če program deluje.

Ko se program uspešno prevede, si lahko pogledamo angleške besede v slovarju ter si izberemo našo ključno besedo. V mapi *assets/sync* odpremo datoteko *cmudict-en-us.dict* in si izberemo nekaj besed za našo ključno besedo. Na demonstracijskem primeru smo si za primer izbrali ključno besedo "Wakeup Carmen", katero seveda lahko kasneje tudi spremenimo. V dokumentaciji orodja PocketSphinx preberemo, da moramo ob menjavi ključne besede spremeniti tudi parameter dolžine naše izgovorjene ključne besede. To spremenimo v spremenljivki imenovani setKeywordThreshold, ki se nahaja v metodi setupRecognizer. V dokumentaciji imamo nekaj primerov dolžin besed [10]:

```
oh mighty computer /1e-40/
```

```
hello world /1e-30/
other phrase /1e-20/
```

Med preizkušanjem programa za čim bolj optimalno dolžino naše ključne besede izberemo dolžino 1e-35, katera tudi po dolžini ključne besede izgleda malo daljša kot "Hello World". Implementacija dolžine besed deluje med intervaloma le-1 in le-50, kjer del za minusom pomeni dolžino besede.

Za daljše ključne besede je potrebno razdeliti besedno zvezo na več manjših delov, prav tako pa lahko ustvarimo tudi zvočni posnetek in ga vnesemo v program. Za dobro natančnost izgovora mora biti zvočni posnetek dolg vsaj eno uro različnih glasov ponavljajočih se besed. Zvočni posnetki so seveda lahko uvoženi s filmov, nekaj jih lahko posnamemo sami ali pa jih prenesemo iz spletnih virov. Ukaz za prevod datoteke je naslednji [25]:

```
pocketsphinx_continuous
-infile <your_file.wav> -keyphrase <your keyphrase>
\-kws_threshold <your_threshold> -time yes
```

To funkcionalnost CMUSphinx orodja smo zaradi dolgotrajnega iskanja besednih zvez ter snemanja različnih glasov različnih ljudi z različnimi narečji preskočili, saj snemanje ter iskanje različnih govorov ni glavni namen našega dela.

Predvsem pa je pomembno omeniti, da je za najboljšo natančnost razpoznave ključne besede potrebno imeti 3-4 zlogovno besedo. Prekratke besedne zveze lahko povzročijo slabše delovanje sistema (slabše zaznavanje ključne besede). Pri predolgih besednih zvezah pa je večja možnost napačnega zaznavanja posamezne besede.

3.3 Uporaba slovarja

Orodje CMUSphinx je zelo prilagodljivo. V njem lahko spreminjamo ključne besede, s katerimi pokličemo napravo ali pa dodamo svoje besede za razpoznavanje govora. Najprej omenimo, da CMUSphinx orodje že omogoča razpoznavo nekaterih tujih jezikov, vendar ne podpira slovenščine. Tu bi lahko ustvarili slovar slovenskega jezika, vendar se bomo zaenkrat raje posvetili že ustvarjenemu angleškemu slovarju, saj je količina dodatnega dela preobsežna za naše delo.

PocketSphinx trenutno podpira naslednje jezike [22]:

- nemški jezik
- grški jezik
- indijsko-angleški jezik
- francoski jezik
- nizozemski jezik
- španski jezik
- italijanski jezik
- Mandarin jezik
- Hindijščino
- kazaški jezik
- ruski jezik

3.3.1 Razširitev obstoječega slovarja

V kolikor želimo dopolniti obstoječ slovar ali pa ustvariti nov slovar čisto od začetka obstaja veliko orodij. CMUSphinx v svoji dokumentaciji priporoča naslednje tri [21]:

- Phonetisaurus
- Sequitur
- G2P-Seq2Seq

CMUSphinx za ustvarjanje ali dodajanje besed obstoječemu slovarju najbolj priporoča G2P-Seq2Seq model. Je preprost model orodja za dodajanje besed v angleškem jeziku.

Model G2P-Seq2Seq uporablja LSTM globinske nevronske mreže, katere so se izkazale za zelo učinkovite pri modeliranju jezikov, razpoznavanju govora ter ostalih področjih, ki so povezane z obdelavo zaporedij. Pri tem modelu se uporabljata dve LSTM nevronski mreži. Prva za kodiranje zaporedja znakov, druga pa za dekodiranje zaporedja. Program preprosto zazna besedo, katero smo mu kot vhod posredovali izgovorjeno besedo. Preko G2P prevajalnega modela besedo pretvori v računalniško razumljiv jezik ter nato izpiše izgovorjeno besedo, zapisano v slovarju kot zaporedje posameznih govornih enot imenovanih fonemi. Fonemi se sicer tvorijo samodejno, vendar program še vseeno potrebuje veliko človeškega dela, saj ga moramo "naučiti", da bo fonem, katerega smo spregovorili, pravilen. Tako modele naučimo s pomočjo primerov izgovorjav fonemov, katerim točnost razpoznave besed narašča s primeri izgovorjav. Učenje modela je počasno ter dolgotrajno dejanje, zato je tudi preko spleta na voljo le omejena množica jezikov za razpoznavanje človeškega govora. Vendar se v podrobnosti kreiranja slovarja ter dodajanja besed zaenkrat ne bomo spuščali, poleg tega nam bo zaenkrat dovolj uporaba že narejenega angleškega slovarja. Angleški slovar trenutno vsebuje skoraj 150 000 besed, katere lahko uporabimo.

Del angleškega slovarja izgleda takole:

```
the DH AH
the(2) DH IY
thea TH IY AH
theall TH IY L
theano TH IY N OW
theater TH IY AH T ER
theater's TH IY AH T ER Z
theatergoer TH IY T ER G OW ER
theatergoer(2) TH IY IH T ER G OW ER
theatergoers TH IY T ER G OW ER Z
theatergoers(2) TH IY IH T ER G OW ER Z
theaters TH IY AH T ER Z
theatre TH IY AH T ER
theatre's TH IY AH T ER Z
theatres TH IY AH T ER Z
```

V delu zgoraj omenjenega slovarja vidimo, da ima beseda **THE** dve izgovorjavi. To sta zgoraj označeni besedi **THE** ter **THE(2)**. V primeru, da ima beseda lahko dva ali več izgovorjav, to slovarju povemo tako, da brez presledka za besedo napišemo zaporedno številko izgovorjave v oklepaju.

V že narejenem slovarju nam ni potrebno odstranjevati neuporabljenih besed, razen če želimo prihraniti prostor na pomnilniku naprave. Dodatne besede v slovarju ne vplivajo na točnost.

Nastavitev iskanja ključne besede v programu, kjer je *keyphrase* iskana beseda je v kodi implementirana:

```
recognizer.setKws('keyphrase', kwsFile);
recognizer.startListening('keyphrase')
```

3.4 Delovanje programa z orodjem PocketSphinx

Ko pokličemo našo napravo s ključno besedo in se naprava začne odzivati na naš govor, potem je čas, da naš program dopolnimo. Naš program ima do sedaj le eno nalogo. Ko terminal pokličemo, se odzove bodisi preko Toast prikaznega okna, katerega uporablja Android sistem za časovno omejen prikaz sporočila, bodisi napiše v kakšen gradnik izgovorjene besede. Kot programerji pa za testiranje največkrat uporabimo izpis LogCat. LogCat je tekstovna datoteka, kamor naprava zapisuje težave, sporočila, informacije,... Datoteka hrani določeno število zapisov. Zapise briše od zgoraj navzdol, ko doseže določeno število vrstic, da so zadnja sporočila kljub brisanju vrstic še vedno vidna.

Da se terminal le odzove na naš klic nam v bolnišnici ne pomaga veliko, zato bi program radi nadgradili, da lahko razpozna več kot le točno določeno (ključno) besedo.

3.5 Analiza uporabnosti orodja

PocketSphinx orodje je do sedaj naše najboljše testirano orodje za neprekinjeno poskušanje. Namenjen je poslušanju točno določene besede ali besedne zveze. Orodje je zelo prilagodljivo in dogradljivo iz strani razvijalcev. Vendar pa nam orodje zaradi nekaterih omejitev, kot so vnaprej definirana dolžina besedne zveze, poslušanje le določene besedne zveze ter slabšega posodabljanja slovarja v našem programu ne bo dovolj. Poleg tega se orodje osredotoča le na točno določene izgovorjene besede, zato vrstnega reda klicanja terminala ne smemo spremeniti med izgovorjavo.

Zaradi zgoraj omenjenih omejitev bomo poleg PocketSphinx orodja kasneje uporabili tudi orodje Android.SpeechRecognizer, kateremu bomo lahko povedali naključen stavek, le tega pa bo pretvoril v besedilo. Poleg tega bomo v naslednjem poglavju uporabili tudi Android.TextToSpeech orodje. Le ta

nam bo prišel prav za tvorbo samodejnega govornega odziva, katerega bomo nato predvajali zaposlenemu. Več o implementaciji orodij si bomo pogledali v naslednjih poglavjih.

Poglavje 4

Tvorba samodejnih govornih odzivov z orodjem TextToSpeech

Ko PocketSphinx ugotovi, da smo terminal poklicali, se izvede metoda *onPartialResult* orodja PocketSphinx. Vendar pa je poleg izvedene metode pomembno vedeti, kdaj je terminal pripravljen na poslušanje našega ukaza. Uporabnik v bolnišnici velikokrat nima časa gledati na zaslon terminala, kdaj bo terminal izpisal, da je pripravljen na naslednji ukaz, katerega mu bomo povedali. Preko metode *onPartialResult* pokličemo našo na novo implementirano metodo imenovano *sayNurse*. V tej metodi uporabimo novo orodje, ki ga ponuja Android imenovano *TextToSpeech* [6].

Orodje *TextToSpeech* deluje nasprotno, kot SpeechRecognizer. Prebere besedilo, zapisano v programu v angleščini, ga pretvori v govor ter predvaja na terminalu preko zvočnikov.

Android.TextToSpeech orodje dinamično podpira jezike glede na verzijo Android sistema na terminalu ali drugi android napravi. Na našem terminalu podpira naslednje jezike: kitajski, češki, danski, nizozemski, angleški, estonski, filipinski, finski, francoski, nemški, grški, madžarski, indonezijski, italijanski, japonski, korejski, nepalski, norveški, poljski, portugalski, španski,

švedski, tajski, turški, ukrajinski ter vietnamski jezik.

Razredu najprej ustvarimo TextToSpeech spremenljivko:

```
TextToSpeech talkNurse;
```

Paziti moramo, da je pred začetkom samodejnega odgovora, ki je v tem primeru metoda *sayNurse*, potrebno ustaviti (Stop, Destroy) vsa prejšnja orodja za razpoznavo govora. Ob neustavitvi le teh bi lahko prišlo do nepotrebne razpoznave.

TextToSpeech orodje že vsebuje metodo *setOnUtteranceCompletedListener*, ki se izvede, ko poklicano orodje zaključi z govorom. Tako naprava ve, kdaj se je samodejni govor končal.

Vendar pa pri preizkušanju TextToSpeech orodja naš projekt ni deloval pravilno, saj se je metoda klicala večkrat vzporedno, česar pa nismo želeli. Zato smo metodi dodelili še funkcijo *synchronized* [4], katera skrbi, da se metoda ne more klicati večkrat vzporedno (deluje podobno kot razred *Handler*, opisan v poglavju 3.2.3).

Metoda *onUtteranceCompletedListener* [7] deluje pravilno, ko na koncu metode *speak* v orodju TextToSpeech, kot zadnji parameter metode dodamo še HashMap, v katerem je kot prva vrednost zapisana:

```
private HashMap<String, String> mapSpeak;

mapSpeak = new HashMap<String, String>();
mapSpeak.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID,
"EndOfTextSpeaking");
```

```
talkNurse = new TextToSpeech(getApplicationContext(),
new TextToSpeech.OnInitListener() {
    @Override
    public void onInit(int status) {
       if (status == TextToSpeech.SUCCESS) {
       recognizer.stop();
       String NurseBackChoice = "Hi, Carmen here, how
       can I help you with?";
       talkNurse.setLanguage(Locale.US);
       talkNurse.setOnUtteranceCompletedListener(new
       TextToSpeech.OnUtteranceCompletedListener() {
           @Override
           public void onUtteranceCompleted(String s) {
               UtteranceCompleted();
           }
       });
    }
}
```

Po implementaciji in uporabi orodja TextToSpeech terminal sedaj posluša uporabnika, kdaj ga bo poklical, nato pa se terminal odzove z besedilom: *Hi, Carmen here, how can I help you with?* Ko metoda TextToSpeech zaključi s svojim delom, terminal sporoči uporabnikom, da je po koncu TextToSpeech govora pripravljen na ponovno poslušanje naslednjega ukaza. O razpoznavi naslednjega ukaza si bomo podrobneje pogledali v naslednjem poglavju.

Poglavje 5

Razpoznava ukazov z orodjem SpeechRecognizer

Do sedaj se nam terminal odziva na naš klic ter nam zvočno odgovarja. Omenili smo že, da terminalu lahko določimo več kot le eno nalogo, poleg tega pa programu še manjka funkcionalnost razpoznave govora z naključno vsebino. To so uporabnikovi ukazi. Zato bomo poleg CMUSphinx ter TextToSpeech orodja uporabili tudi orodje Android.SpeechRecognizer. Orodje se zelo dobro posodablja s strani Androida, poleg tega pa je že zelo napredno in nam zna zelo dobro pretvoriti naše glasovno sporočilo v besedilo. Orodju lahko povemo kakršnokoli govorno besedilo v več jezikih, ta ga bo nato pretvoril v besedilo ter poslal našemu strežniku.

5.1 Kratka zgodovina orodja

Android Speechrecognizer se je začel razvijati leta 2010 [3] (API 8, Android verzija: 2.2). Takrat je Android dodal razpoznavanje govora glasovnemu iskanju v napravah Android. S tem namenom naj bi omogočil uporabnikom hitrejšo uporabo telefonov. Leta 2011 je bila dodana funkcionalnost tudi v brskalnik, kjer jo lahko uporabimo med brskanjem, pisanjem besedila v okvirje ter menjavo zavihkov. Po tem letu se orodje posodablja z novimi

verzijami Android sistema.

Android SpeechRecognizer smo testirali na veliko napravah različnih znamk in cenovnih razredov ter različnimi verzijami Android sistema. Orodje je od verzije 5.0 (Lollipop) naprej že zelo dobro implementirano in zajame precej točne podatke. Pred to verzijo je imel Android v svoji bazi podatkov precej manj nedopolnjenih slovarjev ter algoritmov. Testiranje Android verzij sicer ni potekalo na našem prvotnem projektu, vendar na dodatno implementirani aplikaciji, katero smo naredili le za testiranje orodja Android.SpeechRecognizer.

5.2 Uporaba orodja Android.SpeechRecognizer

Na koncu izvajanja TextToSpeech orodja (v prejšnjem poglavju omenjene metode *onUtteranceCompletedListener*) ustvarimo spremenljivko SpeechRecognitionListener razreda Android.SpeechRecognizer. Orodje *SpeechRecognizer* omogoča razpoznavo človeškega govora, kjer je programsko določen začetek, posluša pa vse do konca govora. Za razliko od PocketSphinx se Android orodje prilagaja dolžini govora, pri PocketSphinx pa se dolžina določi programsko. Ko Androidovo orodje glasovno sporočilo pretvori v besedilo, nam vrne nekaj najboljših zadetkov pretvorbe v obliki seznama.

Ker nam terminal vrne seznam možnosti, kar smo mu ukazali, se moramo domisliti algoritma, kako bi razčlenili posamezne besedne zveze v seznamu. Ker smo z zaposlenimi v ustanovah določili točno določene fraze, kako poklicati terminal ter s katerimi besednimi zvezami od njega zahtevamo nalogo, bi se najprej sprehodili po seznamu ter preverili, če se mogoče točno določena fraza nahaja v tem seznamu.

Ljudje lahko velikokrat pozabimo, kako točno pokličemo terminal, da nam bo prisluhnil ali pa obrnemo vrstni red besed v stavku. Na žalost si pri PocketSphinx orodju ne moremo preveč pomagati, saj se orodje osredotoča le na točno določene izgovorjene besedne zveze. V Androidovem orodju pa nam zelo prav pride, ko lahko uporabniki uporabimo več podobnih izrazov

za klicanja SOS klica, asistence ter doktorskega klica.

V kolikor točno določene fraze ni v seznamu, obstaja veliko načinov, kako lahko gledamo na rezultat seznama razpoznave govora. Lahko uporabimo kar prvi rezultat seznama, ki ga vrne Android SpeechRecognizer. Tukaj mora terminal pravilno prevesti izgovorjeno besedo in jo vrniti kot prvi rezultat v seznamu, da bo program lahko nadaljeval pravilno izvajanje, kot je napisano v programu.

Naslednji način je, da se sprehodimo čez seznam in nekako ugotovimo, kaj je uporabnik želel od terminala. Predvsem nam bo prav prišla metoda *String.contains*. V kolikor en rezultat v seznamu vsebuje več besed, ki se skladajo z besedami napisanimi v programu naj terminal naredi želeno. V kolikor pa se besedilo ne sklada z nobenim klicem, naj terminal pošlje prvi zadetek iz seznama strežniku. Strežnik nato pridobljeno besedilo prikaže na drugih terminalih in zaposleni lahko vidijo, kaj od njih želi bolnik.

5.2.1 Prva uporabna funkcionalnost našega projekta

Ker smo kot izziv dobili pred začetkom nadgrajevanja terminalov, da zaposleni brez sprehajanja po sobi sredi dela z bolnikom lahko pokliče še enega zaposlenega, si bomo najprej pogledali, kako pokličemo asistenco. Asistenca v našem projektu pomeni klic dodatne medicinske sestre. Na aplikaciji, ki se izvaja na terminalu moramo najprej vedeti, ali je kdo od zaposlenih trenutno prijavljen v tej sobi. V kolikor je, lahko pokličemo asistenco ali klic zdravnika na pomoč. V kolikor v sobi ni prijavljen nihče, lahko sprožimo le sestrski klic. Ko pride sestra po klicu v sobo, ugotovi ali potrebuje pomoč dodatne sestre ali zdravnika. Potek razpoznavanja si lahko ogledamo na sliki 5.1.

Zaradi pravic uporabnika terminala smo naredili nov javno dostopni razred ter vsem uporabniško vidnim zaslonom dodelili, da bodo spreminjali javno spremenljivko na odvisnost, kje v aplikaciji se trenutno nahajamo. To sicer ni primarno delo našega projekta, zato bomo novo funkcionalnost opisali zelo na kratko. Vsakič, ko se kliče metoda za prikaz novega okna v Android aplikaciji (new Intent), želimo, da se spremeni tudi spremenljivka v

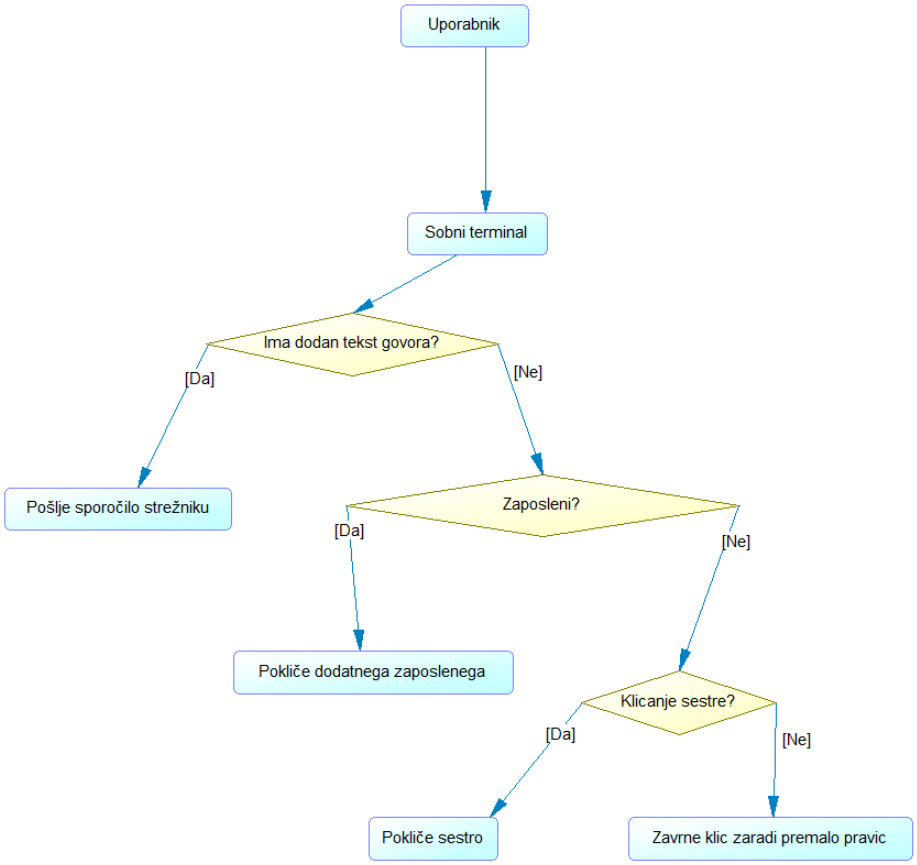
zgoraj opisanem javnem razredu. Spremenljivka se mora posodobiti tudi, ko pritisnemo na tipko nazaj ter vse ostale možnosti spreminjanja zaslona. Ko bomo pri SpeechRecognizerju spremenljivko prebrali, bomo lahko ocenili, ali imamo pravice za sprožitev klica asistence ali ne. V kolikor je zaposleni v sobi že prijavljen v sistem, lahko sprožimo asistenčni ali doktorski klic. Tako bolnik ne more poklicati zdravnika brez prijave zaposlenega v sobi.

V spodnji kodi lahko preverimo, ali je zaposlen prijavljen v sistem s pomočjo pogleda, kje v aplikaciji se trenutno nahajamo. V kolikor je, se lahko sproži asistenčni ali doktorski klic.

```
if(DataHandler.currentScreen.name().equals("PRESENCE") ||
DataHandler.currentScreen.name().equals("SERVICES"))
```

Ko se spremenljivka zanesljivo spreminja ob menjavi prikaznega okna na zaslonu, lahko naredimo prvo funkcionalnost glede razpoznave govora, ki jo bodo lahko uporabili v bolnišnici. Le ta je namenjena osnovnemu sporazumevanju terminala z govorom. Terminal bo poklical klic asistence, ko bo zaposlen v sobi potreboval pomoč dodatne sestre. Opis implementacije klicanja dodatne sestre bo podrobneje opisan v poglavju 5.2.3.

Program sedaj posluša neprekinjeno na uporabnikov klic ter se glasovno odzove, ko ga pokličemo. Terminal sedaj ve, kje v aplikaciji se trenutno nahaja in ima to shranjeno v javni spremenljivki.



Slika 5.1: Potek razpoznavanja govora in pravice.

5.2.2 Opis razreda Android.SpeechRecognizer in vmesnika RecognizerListener

Zaradi enostavnejše implementacije Android.SpeechRecognizerja bomo uporabili Androidov vmesnik RecognizerListener. Vmesnik vsebuje vse metode, katere potrebujemo za uspešno pretvorjeno besedilo.

Vsebuje naslednje metode [11]:

- onBeginningOfSpeech() - Metoda se izvede samodejno, ko uporabnik prične z govorom - Metodo predvsem uporabimo na testiranju orodja, da preverimo, če je razred PocketSphinx sprostil mikrofon, da lahko razred SpeechRecognizer začne s poslušanjem govora.

- onBufferReceived(byte[] buffer) - Metoda za nas sicer ni zanimiva, kliče se takrat, ko orodje sliši več različnih govorov. Pri tem težje razloči, kaj je zaposlen oziroma bolnik povedal.

- onEndOfSpeech() - Metoda se kliče, ko uporabnik preneha z govorjenjem.

- onError(int error) - Metoda ki se izvede ob nastanku težav, katere so opisane spodaj:

```
ERROR_NETWORK_TIMEOUT - Predolg omrežni odzivni čas.
ERROR_NETWORK - Težava z omrežno nastavitvijo.
ERROR_AUDIO - Težava z zvokom.
ERROR_SERVER - Napaka pri povezavi s strežnikom.
ERROR_CLIENT - Težava z odjemalcem.
ERROR_SPEECH_TIMEOUT - Predolg odzivni čas za
govor uporabnika.
ERROR_NO_MATCH - Orodje ne more vrniti rezultatov
razpoznavanja glasu.
ERROR_RECOGNIZER_BUSY - Orodje za razpoznavanja glasu
zasedeno.
ERROR_INSUFFICIENT_PERMISSIONS - Težava z dostopom
ali pravicami - v večini primerih dostop do mikrofona.
```

- onEvent(int eventType, Bundle params) - Rezervirana metoda, katera bo namenjena za dodajanje novih dogodkov (v prihodnosti).

- onPartialResults(Bundle partialResults) - Metoda se pokliče, ko je delni rezultat na voljo.

- onReadyForSpeech(Bundle params) - Metoda se kliče, ko je SpeechRecognizer na voljo, da začne poslušati uporabnika. Tu lahko nastavimo naprimer pisk ali kaj podobnega, da uporabniku povemo, da lahko začne z govorom.

- onResults(Bundle results) - Za nas najpomembnejša metoda, katera se pokliče, ko je rezultat poslušanja na voljo.
- onRmsChanged(float rmsdB) - Metoda se pokliče, ko nastane sprememba glasovne jakosti govora / zvoka.

  Na začetku nad vsemi zgoraj opisanimi metodami naredimo dnevnik (Log). Tako kot razvijalci najlažje vidimo, kako točno program deluje, kakšne so lahko napake, kaj metode kličejo ter kako se odzovejo na določene rezultate.

5.2.3 Klicanje asistence

V zgoraj omenjenih metodah dobimo seznam z rezultati govora v metodi *onResults*. Tukaj imamo en sam argument metode, shranjen v razredu Bundle. Zato naredimo spremenljivko tipa seznama nizov. Seznam bomo napolnili z seznamom rezultatov prejetih v argumentu metode.

```
ArrayList<String> resultArray =
  results.getStringArrayList(
    android.speech.SpeechRecognizer.RESULTS_RECOGNITION);
```

Pri seznamu rezultatov bomo zaenkrat naredili enostaven način iskanja najprimernejšega rezultata uporabniškega govora. Lahko rečemo, da naš program lahko sprejme le določene fraze oziroma rezultate govora. Zato se lahko sprehodimo le po seznamu rezultatov imenovanem *resultArray*. Najprej preverimo, če resultArray vsebuje frazo *call assistance* ali *send assistance*. V spremenljivki imenovani *DataHandler.currentScreen.name* za preverjanje, kje trenutno se nahajamo moramo preveriti, če je zaposlen prijavljen v sobi, da lahko pokliče asistenco. To pa naredimo tako, da pokličemo enako metodo, kot če bi pritisnili na gumb za asistenco.

V spodnjem primeru lahko vidimo, kako pokličemo metodo za klicanje dodatne sestre (ustvarimo dogodek asistence v določeni sobi). Uporabili smo tudi TextToSpeech orodje kot samodejno sporočilo terminala ob uspešno poslanem klicu asistence strežniku. Poleg tega moramo preveriti, če je zaposleni prijavljen na terminalu, katero smo opisali v poglavju 5.2.1.

```
if (result.contains("call assistance") ||
    result.contains("send assistence")) {
       MainScreen.getInstance().onAssistanceClick(null);
       talkNurse.speak("Assistance sent",
       TextToSpeech.QUEUE_FLUSH, null);
```

Na zgoraj opisan način lahko tudi bolnik pokliče sestro, sestra pa asistenco ali zdravnika. Zdravnik lahko pokliče dodatnega zdravnika ali sestro. Klici se razlikujejo le v klicanju različnih funkcij, katere pošljejo strežniku različne podatke.

5.2.4 Pošiljanje osebnega sporočila medicinski sestri

Kot zadnjo implementacijo, ki jo bomo opisali v diplomski nalogi je pošiljanje našega lastnega sporočila medicinski sestri.

Namesto klica na pomoč izgovorjeno naključno besedilo shranimo v spremenljivko, katero nato pošljemo do našega strežnika. Strežnik dogodek skupaj z izgovorjenim besedilom nato shrani v podatkovno bazo. Pred pošiljanjem dogodka na ostale terminale bo najprej pogledal, ali je v podatkovni bazi shranjeno tudi izgovorjeno besedilo, katerega imamo shranjenega v posebnem stolpcu ter ga poslal naprej na terminale. Tako lahko bolnik pove, kaj potrebuje in tako olajša delo zaposlenega, kateri s tem razlogom lahko pride v sobo že pripravljen.

Kot primer lahko dodamo bolnika, ki bi rad, da mu sestra prinese v sobo vroč čaj. Bolnik lahko pokliče terminal ter mu v angleškem jeziku pove "I need tea". Tako se sestri ni potrebno sprehajati dvakrat do sobe, saj bi tako prvič vprašala bolnika, kaj želi, drugič pa prinesla čaj.

Z novo funkcionalnostjo smo zelo olajšali delo in čas zaposlenega, poleg tega pa delo poteka bolj neprekinjeno in sestra lahko naredi veliko več stvari tekom dneva.

V spodnji kodi lahko vidimo klic bolnika z besedilom, katerega je povedal. V parametru klicanja funkcije lahko podamo izgovorjeno besedilo bolnika, nato pa preko TextToSpeech bolniku povemo, da se je klic z besedilom uspešno poslal. Na Sliki 5.2 pa si lahko ogledamo prikaz klicev z naključnim besedilom ter ostale aktivne dogodke.

```
if (matches.size() > 0)
      MainScreen.getInstance().onCall(matches.get(0));
      talkNurse.speak("OK, I will tell to Nurse: "
      + matches.get(0), TextToSpeech.QUEUE_FLUSH, null);
```

Slika 5.2: Razpoznano naključno besedilo prikazano v sestrski sobi v rdeči barvi.

Poglavje 6

Sklepne ugotovitve

V diplomskem delu je predstavljena uporaba dveh orodij za razpoznavanje govora. Orodji smo uporabili na Android operacijskih sistemih. Imenujeta se CMUSphinx ter Android.SpeechRecognizer. Slednje orodje smo uporabili s pomočjo vmesnika RecognizerListener.

- Pri CMUSphinx uporabljamo orodje PocketSphinx za brezprekinitveno poslušanje ključne besede, pri kateri čas začetka govora ni programsko določen. Pri orodju smo povedali, da ima kot prednost brezprekinitveno poslušanje, kot slabost pa, da se orodje slabo posodablja ter ima programsko določen parameter dolžine izgovorjene besede. Uporabljamo ga le za razpoznavo vnaprej znanih besednih zvez, saj se je pri testiranju pokazalo, da je orodje precej slabo pri razpoznavi naključno izgovorjenih besed.

- Orodje Android.SpeechRecognizer uporabljamo, ko programsko lahko določimo, od kdaj mora program začeti poslušati uporabnika. Uporablja se za različno dolga besedila. Preneha poslušati, ko uporabnik konča z govorom. Vrne nam seznam možnih rezultatov. Dolžina in čas izgovorjenega besedila nista programsko določena. Primeren je za poslušanje naključno dolgih povedi uporabnika.

Na podlagi zgornjih dveh orodij lahko implementiramo veliko zanesljivih aplikacij, pri kateri lahko uporabniku olajšamo delo ter prihranimo čas. Opisali smo celoten postopek implementacije omenjenih orodij ter delovanje storitev, ki se v celoti izvajajo v ozadju programa.

Kot pomoč pri aplikaciji, katera je naš govor razpoznala kot klic terminala smo uporabili orodje Android.TextToSpeech. Orodje nam pretvori tekstovno v govorno besedilo. Zapisano besedilo se nato predvaja na terminalu.

Z novimi funkcionalnosti v medicinskih ustanovah smo sedaj zaposlenim omogočili enostavnejše delo, prav tako pa smo bolnikom omogočili, da zaposlenim že v naprej povejo, kaj želijo od njih.

Zaposleni pa v medicini uporabljajo tudi pametne mobilne telefone z naprednimi aplikacijami. Kot naslednja implementacija v prihodnosti je integracija razpoznave govora tudi za telefone. Tako bo sestra lahko preko mobilnega telefona javila odgovor na klic ali pa povedala nekaj sistemu, sistem pa bo nato posredoval njeno sporočilo naprej na druge terminale. Ena izmed možnosti je naprimer, da je sestra trenutno zasedena z delom. Klicu lahko posreduje komentar, prosi drugega zaposlenega za odziv na sestrski klic.

Poleg tega bi lahko tudi poskusili implementirati razpoznavanje govora z orodjem Sphinx4. Nato bi primerjali orodji ter se odločili za boljšega.

Kot novo implementacijo bi lahko sistem tudi nadgradili z naključnim besedilom za klic asistence ter zdravnika.

Literatura

[1] Amazon echo speaker. Dosegljivo: https://developer.amazon.com/alexa, 2018. [Dostopno: 10.6.2018].

[2] Android handler class. Dosegljivo: https://developer.android.com/reference/android/os/Handler, 2018. [Dostopno: 10.6.2018].

[3] Android speechrecognizer history. Dosegljivo: https://www.itbusiness.ca/news/history-of-voice-recognition-from-audrey-to-siri/15008, 2018. [Dostopno: 10.6.2018].

[4] Android synchronized method. Dosegljivo: https://developer.android.com/reference/java/util/concurrent/locks/Lock, 2018. [Dostopno: 10.6.2018].

[5] Android.speechrecognizer main page. Dosegljivo: https://developer.android.com/reference/android/speech/SpeechRecognizer, 2018. [Dostopno: 10.6.2018].

[6] Android.texttospeech. Dosegljivo: https://developer.android.com/reference/android/speech/tts/TextToSpeech, 2018. [Dostopno: 10.6.2018].

[7] Android.texttospeech.onutterancecompletedlistener. Dosegljivo: https://stackoverflow.com/questions/4652969/android-tts-onutterancecompleted-callback-isnt-getting-called/, 2018. [Dostopno: 10.6.2018].

[8] Cortana. Dosegljivo: https://www.microsoft.com/en-us/cortana, 2018. [Dostopno: 10.6.2018].

[9] Difference between pocketsphinx and sphinx4. Dosegljivo: https://cmusphinx.github.io/wiki/tutorialoverview/, 2018. [Dostopno: 10.6.2018].

[10] Example of magic word length. Dosegljivo: https://cmusphinx.github.io/wiki/tutoriallm/, 2018. [Dostopno: 10.6.2018].

[11] Explained errors for android.speechrecognizer tool. Dosegljivo: https://developer.android.com/reference/android/speech/SpeechRecognizer, 2018. [Dostopno: 10.6.2018].

[12] Google home speaker. Dosegljivo: https://store.google.com/gb/product/google_home?hl=en-GB, 2018. [Dostopno: 10.6.2018].

[13] Google speechrecognizer voice commands. Dosegljivo: https://www.digitaltrends.com/mobile/ok-google-voice-commands-list/2/, 2018. [Dostopno: 10.6.2018].

[14] Implementing android background service. Dosegljivo: https://developer.android.com/training/run-background-service/create-service, 2018. [Dostopno: 10.6.2018].

[15] Importing pocketsphinx into project. Dosegljivo: https://cmusphinx.github.io/wiki/tutorialandroid/#building-and-running-from-android-studio, 2018. [Dostopno: 10.6.2018].

[16] Kinect. Dosegljivo: https://support.xbox.com/en-BM/xbox-360/kinect/speech-recognition, 2018. [Dostopno: 10.6.2018].

[17] Pocketsphinx and sphinx4 difference number 2. Dosegljivo: https://cmusphinx.github.io/wiki/tutorialbeforestart/#technologies, 2018. [Dostopno: 10.6.2018].

[18] Pocketsphinx continuous recognizer implementation. Dosegljivo: https://www.vladmarton.com/pocketsphinx-continuous-speech-recognition-android-tutorial/, 2018. [Dostopno: 10.6.2018].

[19] Pocketsphinx demo project download from github. Dosegljivo: https://github.com/cmusphinx/pocketsphinx-android-demo, 2018. [Dostopno: 10.6.2018].

[20] Pocketsphinx devices support list. Dosegljivo: https://cmusphinx.github.io/2010/03/pocketsphinx-0-6-release/, 2018. [Dostopno: 10.6.2018].

[21] Pocketsphinx dictionary. Dosegljivo: https://cmusphinx.github.io/wiki/tutorialdict/, 2018. [Dostopno: 10.6.2018].

[22] Pocketsphinx language support. Dosegljivo: https://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/, 2018. [Dostopno: 10.6.2018].

[23] Pocketsphinx main page. Dosegljivo: https://cmusphinx.github.io/wiki/tutorialpocketsphinx/, 2018. [Dostopno: 10.6.2018].

[24] Pocketsphinx syncassets. Dosegljivo: https://cmusphinx.github.io/wiki/tutorialandroid/, 2018. [Dostopno: 10.6.2018].

[25] Pocketsphinx wav file importing into project. Dosegljivo: https://cmusphinx.github.io/wiki/tutoriallm/#keyword-lists, 2018. [Dostopno: 10.6.2018].