UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nejc Bernot

GLASOVNI NADZOR INTERPRETERJA ZPLET Z UPORABO SISTEMA SPHINX-4

DIPOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2012

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Nejc Bernot

GLASOVNI NADZOR INTERPRETERJA ZPLET Z UPORABO SISTEMA SPHINX-4

DIPOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor:
prof. dr. Dušan Kodek, univ. dipl. ing.

Ljubljana, 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01783/2011
Datum: 05.10.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **NEJC BERNOT**

Naslov: **GLASOVNO UPRAVLJANJE INTERPRETERJA ZPLET Z UPORABO SISTEMA SPHINX-4**

VOICE CONTROL OF ZPLET INTERPRETER USING SYSTEM SPHINX-4

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Govor je za človeka najbolj naraven način komunikacije. Zato se že dalj časa razmišlja o uporabi avtomatskega razpoznavanja govora za upravljanje računalniških iger. Med temi igrami so posebej zanimive tako imenovane tekstovne avanture. Zanje je bil že v 1970-tih letih razvita navidezni stroj Z-machine in interpreter ZPlet, ki naredita delovanje iger neodvisno od strojne opreme. Interpreter ZPlet povežite s programskim orodjem Sphinx-4 in s tem omogočite glasovno upravljanje pri igranju teh iger. Uporabite akustične modele in pripadajoče slovarje, ki so na voljo na domači strani razpoznavalnika Sphinx-4. Glasovno upravljanje preizkusite na igri Zork1 in izmerite uspešnost njegovega delovanja.

Mentor:

prof. dr. Dušan Kodek

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a **Nejc Bernot**,

z vpisno številko **63000327**,

sem avtor/-ica diplomskega dela z naslovom:

Glasovni nadzor interpreterja ZPlet z uporabo sistema Sphinx-4

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek) **prof. dr. Dušana Kodeka**
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 13.4.2012 Podpis avtorja:

Zahvala

Rad bi se zahvalil mentorju prof. dr. Dušanu Kodeku za njegovo pomoč ter vodenje pri izdelavi diplomske naloge. Zahvaljujem se tudi asistentu dr. Robertu Rozmanu za njegove ideje ter nasvete.

Še posebej bi se rad zahvalil mojim staršem za podporo ter potrpežljivost v času študija.

Kazalo

Seznam uporabljenih kratic

ARPA – Advanced Research Projects Agency
BNF – Backus-Naurjeva oblika (angl. Backus-Naur Form)
DFT – diskretna Fourierjeva tranformacija (angl. Discrete Fourier Tranformation)
HMM – prikriti Markovski modeli (angl. Hidden Markov Models)
JSGF – Java Speech API Grammar Format
MFCC – mel-frekvenčni kepstralni koeficienti (angl. Mel. Frequency Cepstral Coefficients)
WER – napaka razpoznavanja besed (angl. Word Error Rate)
XML – razširljiv označevalni jezik (angl. Extensible Markup Language)
ZIL – Zork Interpreter Language

Povzetek

Govor je človekova najbolj naravna oblika komunikacije z okolico. Zaradi tega dejstva se je že v štiridesetih letih prejšnjega stoletja začel razvoj sistemov, ki bi omogočali računalniško razpoznavo govora. Namen te diplomske naloge je predstaviti trenutno stanje računalniške razpoznave govora, javanski sistem Sphinx-4 za razpoznavo govora ter povezavo tega sistema z že obstoječim interpreterjem ZPlet, namenjenim igranju iger, napisanih po standardu Z-machine. Uvod vsebuje pregled osnov strukture govora, opis delovanja in različnih oblik sistemov za razpoznavo govora ter kratek opis prednosti razpoznave govora. V nadaljevanju se diplomsko delo nekoliko bolj podrobno posveti teoriji prikritih Markovskih modelov, ki predstavljajo jedro večine sistemov za razpoznavo govora, ki so trenutno v uporabi. V tretjem in četrtem poglavju sta predstavljena sistem Sphinx-4 ter ZPlet, njuna zgradba ter osnova,na kateri delujeta. V sledečih poglavjih je opisan način povezave ter delovanje interpreterja ZPlet, povezanga s sistemom Sphinx-4. Ta poglavja vsebujejo tudi korake, potrebne za zagon poljubnega Z-machine programa v tako razširjenem sistemu ter rezultate praktičnega preizkusa natančnosti razpoznave sistema Sphinx-4.

Ključne besede:

razpoznava govora, Sphinx-4, Z-machine, ZPlet

Abstract

For humans speech represents the most natural form of communication with their environment. Due to this fact research began as early as the 1940s into systems that would enable computers to recognize speech. The purpose of this diploma thesis is to present an overview of the current state of computer speech recognition, the java speech recognition system Sphinx-4 and the integration of this system with the Zplet interpreter, which can be used for playing games written according to the Z-machine standard. The introduction contains an overview of the basics of the structure of speech, a description of the process of speech recognition, a list of the different types of speech recognition systems and a description of the basic advantages of using speech recognition. The following chapter is dedicated to the theory behind hidden Markov models, which form the core of most of the speech recognition systems currently in use. The third and forth chapters contain an overview of the structure and operation of the Sphinx-4 system and the Zplet interpreter. The last few chapters contain the descriptions of the process of integrating Sphinx-4 and Zplet, the steps needed to run any of the Z-machine programs on this integrated system and the results of practical tests of the accuracy of the speech recognition performed by Sphinx-4.

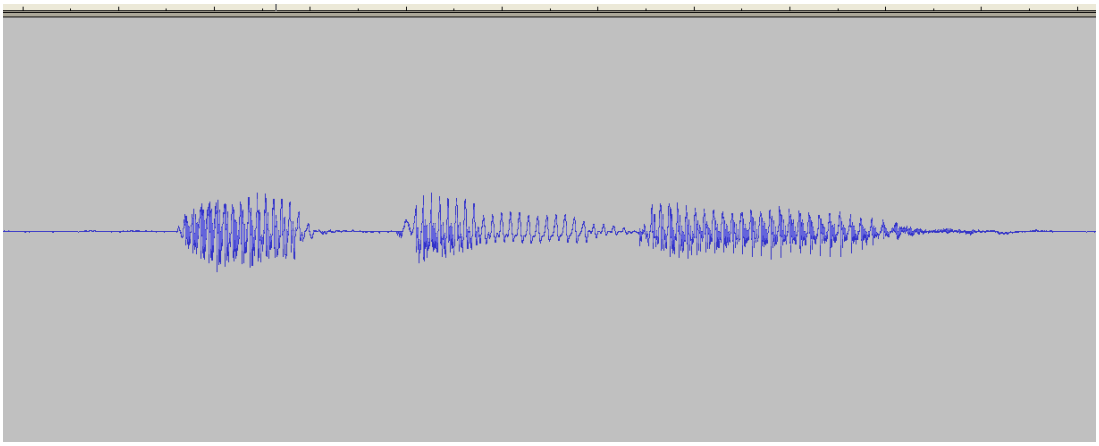
Key words:

speech recognition, Sphinx-4, Z-machine, Zplet

1. Uvod

S hitro naraščajočo priljubljenostjo prenosnih naprav[1] se vedno bolj kaže potreba po novem načinu vnosa besedila, ki bi nadomestil klasično uporabo tipkovnice, bodisi fizične ali pa le navidezne (na primer tipkovnice, realizirane z zaslonom, občutljivim na dotik). Rešitev, ki je zadnje čase vse bolj priljubljena, je uporaba razpoznave govora, procesa, pri katerem preslikamo vhodni signal – govor – v tekstovni zapis zaporedja razpoznavnih enot, ki so lahko fonemi, besede, besedne zveze, stavki ali povedi. Namen te diplomske naloge je predstaviti problem povezave že obstoječe aplikacije (v tem primeru interpreterja ZPlet) ter orodja za razpoznavo govora Sphinx-4.

Preden se spustimo v problem razpoznave govora, se moramo vprašati, kaj govor pravzaprav je [1]. V nasprotju z mnenjem večine ljudi je govor zelo zapleten pojav. Naivna predstava o govoru kot zaporedju besed, sestavljenih iz fonemov, je na žalost napačna. Govor je dinamičen proces brez strogo ločenih delov, kot je mogoče videti iz spodnje slike posnetka govora.



Slika 1: Posnetek ukaza “Open door!”

Vsi sodobni opisi govora so delno verjetnostni, kar pomeni da ni jasnih mej med enotami govora oziroma med besedami. Razpoznava govora zato ni nikoli popolnoma natančna, kar lahko povzroči težave pri njeni uporabi v programski opremi, ki je sicer zelo deterministična.

Strukturo govora pojasnimo tako:

Govor je zvezni zvočni tok, v katerem se bolj stabilna stanja mešajo z dinamično spremenljivimi stanji. V tem zaporedju stanj je mogoče določiti bolj ali manj podobne razrede zvokov, ki jih poimenujemo fonemi. Fonemi nato gradijo besede, vendar je pri tem koraku potrebno veliko pazljivosti. Akustične lastnosti valovanja, ki ustreza določenemu fonemu, so namreč odvisne od mnogih dejavnikov, na primer od konteksta fonema, lastnosti govorca ali od okolja govora.

Vsi ti dejavniki povzročijo, da se fonem lahko močno razlikuje od svoje kanonične reprezentacije. Eno izmed področij velike variabilnosti so tako imenovani di-fonemi, regije

[1] tabličnih računalnikov, pametnih telefonov, bralcev elektronskih knjig ...

med dvema zaporednima fonemoma. Drugi pristop k obravnavi fonemov je z upoštevanjem njihovega konteksta. Tu naletimo na tri-foneme ali celo pet-foneme, katerih lastnosti so odvisne od predhodnih in sledečih fonemov. Zavedati pa se moramo, da obstaja velika razlika v razmerju med fonemi ter di-fonemi ali tri-fonemi. Medtem ko se pojma fonem in di-fonem nanašata na različni regiji zvočnega valovanja, se regiji ki ju pokrivata fonem in tri(ali več)-fonem pokrivata. Razlikuje se le kontekst, ki ga uporabimo pri določanju njune oblike. Zato je bil uveden koncept senona, objekta, katerega odvisnost od konteksta je lahko zelo zapleteno določena, na primer z odločitvenimi drevesi.

Fonemi gradijo podbesedne strukture, ki nato gradijo besede. Besede so pri razpoznavi govora zelo uporabne, saj je njihovo število zelo omejeno v primerjavi z vsemi možnimi kombinacijami posameznih fonemov. Besede in drugi glasovi, ki jih poimenujemo mašila[2], gradijo izjave[3]. Le-te so določene kot kosi zvoka med daljšimi obdobji tišine.

1.1 Razvrstitev sistemov za razpoznavo govora

Sisteme za razpoznavo govora v grobem delimo po sledečih merilih [2]:

a) glede na število govorcev

Poznamo sisteme za prepoznavo govora majhnega števila govorcev (ali samo enega) ter sisteme za prepoznavo govora velikega števila govorcev.

Sistemi za prepoznavo govora majhnega števila govorcev so omejeni na prepoznavo govora oseb, katerih govor je bil prisoten v učni množici sistema. Njihova prednost je njihova visoka natančnost, ter dejstvo, da za njihovo učenje potrebujemo precej manjšo učno množico posnetkov govora (le nekaj ur posnetkov), kot za bolj splošne sisteme.

Sistemi za prepoznavo govora velikega števila govorcev so bolj splošni, saj omogočajo tudi prepoznavanje govora neznanih govorcev, vendar ima ta splošnost svojo ceno. Sistemi so v povprečju manj natančni od sistemov za razpoznavo majhnega števila govorcev, za njihovo učenja pa potrebujemo zelo velike učne množice (več sto ur posnetkov dvesto in več govorcev).

Kompromisno rešitev pomeni prilagoditev[4] sistema za prepoznavo govora, tako da splošni sistem prilagodimo nekemu govorcu z uporabo manjšega števila posnetkov tega govorca. Ta pristop lahko občutno izboljša natančnost razpoznave v primerjavi z neprilagojenim sistemom, obenem pa zahteva manj časa in napora kot izdelava celotnega sistema, namenjenega le enemu govorcu.

b) glede na prepoznavo ločenih besed ali tekočega govora

Prepoznava ločenih besed je preprostejši primer razpoznave govora. Izgovorjene besede morajo biti jasno ločene, z razmikom med zaporednima besedama vsaj 200 ms. Ta

[2] zajem sape, kašelj ali drugi nepomembni glasovi
[3] angl. utterance
[4] angl. adaptation

oblika razpoznave govora je primerna predvsem za uporabo v sistemih, katerih odzivni čas ni kritičen ter v katerih posamezna beseda že pomeni ukaz.

Razpoznava tekočega govora pa je precej težji problem, saj mora sistem med drugim tudi določiti število izgovorjenih besed ter poiskati meje med njimi. Dobro mora obvladati tudi vse koartikulacijske pojave ter slabo izgovarjavo, prisotno v tekočem govoru.

c) glede na velikost slovarja (množice besed, ki so jih sposobni prepoznati)

Sistemi z majhnim slovarjem so omejeni na prepoznavo manj kot 100 besed ter so ponavadi uporabljeni za prepoznavo številk ter v avtomatskih telefonskih odzivnikih.

Sistemi s srednje velikimi slovarji prepoznajo med 100 in 1000 besed ter so ponavadi uporabljeni v raziskovalne namene, pri razvoju novih metod razpoznave govora.

Sistemi z velikimi slovarji prepoznajo nad 1000 besed ter so večinoma dostopni kot del večjih, komercialno dostopnih produktov za razpoznavo govora.

1.2 Uporaba in prednosti razpoznave govora

Avtomatska razpoznava govora se počasi širi na vsa področja našega življenja, na primer:

a) izobraževanje
- – ocenjevanje pravilnosti izgovarjave pri učenju tujih jezikov
- – trening na področjih, kot je na primer kontrola zračnega prometa

b) transport
- – nadzor nad napravami za statelitsko navigacijo

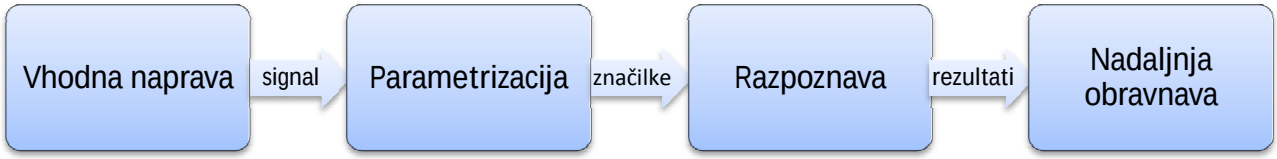
c) delo
- – avtomatsko pisanje po nareku
- – avtomatsko generiranje podnapisov

d) zabava
- – glasovno upravljanje mobilnih telefonov
- – daljinski nadzor nad hišnimi napravami
- – upravljanje robotskih hišnih ljubljencev
- – glasovni nadzor video iger

e) drugo
- – komunikacija z osebami s prizadetim sluhom
- – interakcija z okolico za invalidne osebe

Razpoznava govora ima mnoge prednosti pred drugimi oblikami nadzora, saj zanjo ne potrebujemo velikih vhodnih naprav, kot je na primer tipkovnica. Njena uporaba je mogoča tudi, ko se ukvajamo z drugimi opravili, obenem pa je pogosto tudi hitrejša kot ročni vnos teksta. Še vedno pa se moramo zavedati, da uporaba razpoznave govora ni mogoča na področjih, kjer je natančnost kritična. Vedno je namreč možna napačna razpoznava.

1.3 Osnovni potek razpoznave govora



Slika 2: Shema poteka razpoznave govora

Prvi korak za razpoznavo govora je parametrizacija vhodnega signala [3]. Postopek, ki ga za parametrizacijo uporabimo, je odvisen od sistema razpoznave govora, najpogosteje pa se uporabljajo mel-frekvenčni kepstralni[5] koeficienti, ki jih pridobimo na sledeči način. Signal z uporabo kratkočasne diskretne Fourierjeve transformacije pretvorimo v močnostni spekter. Ta spekter nato preslikamo v tako imenovano mel-merilo. Le-to poskuša aproksimirati sposobnost človeškega sluha pri razlikovanju frekvenc. Človeško uho namreč bolje razloči frekvence, nižje od 1000 Hz, kot višje frekvence, zato to merilo hitro narašča pri nižjih frekvencah, pri visokih pa vedno počasneje. Tako preoblikovane podatke nato logaritmiramo ter nad njimi izvedemo diskretno kosinusno transformacijo. Amplitude tako pridobljenega spektra imenujemo mel-frekvenčni kepstralni koeficienti. Njihove vrednosti predstavljajo vektor značilk, ki jih uporabljamo v naslednjih korakih.

V drugem koraku z značilkami določimo verjetne besede v vhodnem signalu. To izvedemo na podlagi predhodno sestavljenih modelov, naučenih na podlagi velikih množic učnih podatkov – posnetkov jezika, ki ga želimo prepoznati. Najpogosteje uporabljeni tipi modelov za razpoznavo govora so nevronske mreže ter statistični modeli, med katere štejemo tudi prikrite Markovske modele[6]. Ker so le-ti uporabljeni tudi v orodju Sphinx-4, se bo ta diplomska naloga posvetila le njim.

Rezultate drugega koraka lahko nato sprejmemo kot končno rešitev našega problema ali pa jih uporabimo le kot vhodne podatke za nadaljnjo obravnavo, vendar tu že zapuščamo področje razpoznave govora ter se selimo na področje semantike naravnega jezika.

[5] angl. mel-frequency cepstrum coefficients (MFCC)
[6] angl. Hidden Markov Models (HMM)

2. Markovski modeli

2.1 Markovske verige prvega reda

Markovske verige prvega reda definiramo z množico n možnih stanj X={x_1,x_2,….x_n}, ter z matriko verjetnosti prehoda med temi stanji.

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \quad (1)$$

$$a_{ij} = p(X_t = x_j | X_{t-1} = x_i) \quad (2)$$

$$\sum_i a_{ij} = 1 \quad (3)$$

V 2. formuli $p(X_t = x_j | X_{t-1} = x_i)$ predstavlja verjetnost, da se bo model v času t nahajal v stanju x_j, pri pogoju, da se je v času t-1 nahajal v stanju x_i.

Potrebujemo tudi vektor začetnih verjetnosti stanj π.

$$\pi = (\pi_1, \pi_2, \ldots . \pi_n) \quad (4)$$

$$\pi_i = p(X_1 = x_i) \quad (5)$$

$$\sum_i \pi_i = 1 \quad (6)$$

Verjetnost zaporedja stanj dolžine k v takšnem modelu je mogoče izračunati kot produkt verjetnosti začetnega stanja ter verjetnosti vseh sledečih prehodov med stanji.

$$p(zaporedja) = p(X_1) * \prod_{i=2}^{k} p(X_i | X_{i-1}) \quad (7)$$

Določanje parametrov takšnega modela iz učnih podatkov je precej preprosto, saj lahko vektor verjetnosti začetnih stanj določimo po formuli 8.

$$\pi_i = \frac{število\ zaporedij\ stanj,\ \ kjer\ \ X_1 = x_i}{skupno\ število\ zaporedij\ stanj} \quad (8)$$

Vrednosti matrike prehodnih verjetnosti pa določimo po formuli 9.

$$a_{ij} = \frac{\sum_{k=1}^{število\ zaporedij} \sum_{l=1}^{\infty} C(X_l^k = x_i, X_{l+1}^k = x_j)}{\sum_{k=1}^{število\ zaporedij} \sum_{l=1}^{\infty} C(X_l^k = x_i)} \quad (9)$$

V tej formuli $\boldsymbol{C(X_l^k = x_i, X_{l+1}^k = x_j)}$ predstavlja število primerov v k-tem zaporedju, ko stanju s_i sledi stanje s_j.

Markovske verige prvega reda so sicer uporabne pri modeliranju nekaterih preprostih sistemov, vendar so nezmožne upoštevati daljšo zgodovino, saj je izbira naslednjega stanja vezana izključno na trenutno stanje sistema.

2.2 Markovske verige višjih redov

Prva možna nadgradnja našega modela so Markovske verige k-tega reda, pri katerih za razliko od verig prvega reda upoštevamo ne le zadnje stanje sistema,

$$p(X_i|X_{i-1}, X_{i-2}, \dots . X_1) = p(X_i|X_{i-1}) \qquad (10)$$

temveč tudi k preteklih stanj.

$$p(X_i|X_{i-1}, X_{i-2}, \dots . X_1) = p(X_i|X_{i-1}, X_{i-2}, \dots . X_{i-k}) \qquad (11)$$

Takšni modeli so sicer precej bolj prilagodljivi, vendar imajo tudi več resnih pomanjkljivosti. Velika težava pri njihovi uporabi je, da potrebujemo zaporedje zahtev dolžine k, preden lahko naš model začne napovedovati prihodnje zahteve. Druga težava teh modelov pa je njihovo število stanj ter s tem povezana prostorska zahtevnost, ki eksponentno narašča z redom modela, saj nova množica stanj vsebuje vsa možna zaporedja dolžine k. Tako velika množica stanj pomeni tudi to, da bodo nekatera stanja slabo podprta v množici učnih primerov, ali pa se v njej sploh ne bodo pojavila.

Možna rešitev prvega problema je uporaba tako imenovanih All K-th order modelov, pri katerih množico stanj sestavljajo vsa možna zaporedja zahtev z dolžino k ali manj. Tako pridobljeni model je sposoben napovedovati prihodnje zahteve tudi, preden imamo na voljo k predhodnih zahtev, obenem pa je sposoben pri napovedi upoštevati daljšo zgodovino zahtev. Na žalost ta rešitev le še poslabša drugo težavo, saj še bolj poveča našo množico stanj in s tem povezano prostorsko zahtevnost.

Tega problema se lahko vsaj delno rešimo z uporabo obrezovanja stanj, pri katerem z več metodami postopoma zmanjšujemo velikost našega modela. Najbolj očitna metoda je odstranjevanje stanj, ki se v naših učnih podatkih ne pojavljajo dovolj pogosto, saj je malo verjetno, da bodo napovedi, dosežene na njihovi podlagi, zanesljive. Množico stanj lahko nato še bolj obrežemo, če odstranimo vsa stanja višjih redov, kadar so njihove napovedi manj natančne kot napovedi stanja nižjega reda. S tem ne le povečamo zanesljivost našega modela, temveč lahko tudi drastično zmanjšamo njegovo prostorsko zahtevnost.

2.3 Prikriti Markovski modeli

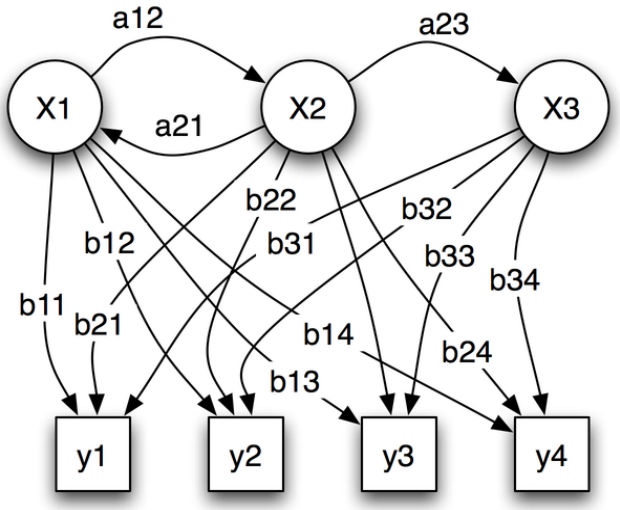
Zadnja vrsta tu predstavljenih modelov so prikriti Markovski modeli [4]. Delovanje teh modelov je v osnovi enako kot delovanje Markovskih verig, vendar se razlikuje v dejstvu, da stanja sistema ne moremo neposredno opazovati. Namesto tega sistem ob vsakem prehodu

med stanji odda enega izmed možnih izhodnih simbolov iz množice $Y=\{y_1,y_2,\ldots y_m\}$. Verjetnost posameznega simbola, oddanega v nekem stanju, je določena z matriko emisijskih verjetnosti B.

$$B = \begin{bmatrix} b_1(y_1) & \cdots & b_n(y_m) \\ \vdots & \ddots & \vdots \\ b_n(y_1) & \cdots & b_n(y_m) \end{bmatrix} \qquad (12)$$

$$b_i(Y_j) = p(Y_t = y_j | X_t = x_i) \qquad (13)$$

Celoten prikriti Markovski model je tako določen s trojčkom $\lambda=(A,B,\pi)$.



Slika 3: Primer prikritega Markovskega modela s tremi stanji ter štirimi izhodnimi simboli

Pri uporabi prikritih Markovskih modelov se srečamo s tremi osnovnimi problemi, in sicer s problemom ocenjevanja, problemom razpoznavanja ter problemom učenja. Preden se lotimo reševanja teh problemov moramo definitrati izhodno zaporedje O, ki ga generira naš model, pri čemer o_i ustreza oddanemu simbolu v času i.

$$O = \{o_1, o_2, o_3, \ldots o_T\} \qquad (14)$$

a) Problem ocenjevanja:
Kakšna je verjetnost, da naš model $\lambda=(A,B,\pi)$ generira zaporedje O?

$$p(O|\lambda) = ? \qquad (15)$$

Naivno gledano je problem preprosto rešljiv, saj od nas zahteva le to, da izračunamo verjetnost vseh mogočih zaporedij zaporedij po formuli 17.

$$S=\{s_1,s_2,s_3,\ldots s_T\} \qquad (16)$$

$$p(S|\lambda) = \pi(s_1) * \prod_{i=2}^{T} a_{si-1,si} \qquad (17)$$

Simbol $a_{si-1,si}$ predstavlja verjetnost prehoda iz stanja, v katerem se sistem nahaja v času i-1, v stanje, v katerem se nahaja v času i.

Tako pridobljene verjetnosti zaporedij nato pomnožimo z verjetnostjo, da posamezno zaporedje generira želeno izhodno zaporedje O.

$$p(O|S) = \prod_{i=1}^{T} b_{si}(o_i) \qquad (18)$$

Simbol $b_{si}(o_i)$ predstavja verjetnost, da sistem v stanju, v katerem se nahaja v času i, odda simbol s_i.

Obe tako pridobljeni verjetnosti nato pomnožimo ter tako pridobimo verjetnost, da je določeno zaporedje notranjih stanj generiralo želeno izhodno zaporedje. To vrednost nato seštejemo za vsa možna zaporedja stanj, ter tako pridobimo verjetnost izhodnega zaporedja.

Na žalost ta pogled ne upošteva računske zahtevnosti celotnega procesa. Zavedati se namreč moramo, da je treba za vsako izhodno zaporedje dolžine T pregledati N^T zaporedij notranjih stanj (pri čemer je N število notranjih stanj sistema), kar z naraščajočo dolžino zaporedja hitro preseže vse razumne meje. Boljšo rešitev prvega problema najdemo v algoritmu naprej-nazaj[7]. Ta algoritem izkoristi dejstvo, da nam ni treba ločeno slediti izvajanju modela po vseh možnih zaporedjih notranjih stanj, saj lahko po vsakem časovnem koraku združimo vsa zaporedja, ki se končajo v določenem stanju. Zato je treba v vsakem koraku i izračunati le N različnih vrednosti, in sicer $p(O_i|S_i)$ za vsako možno vrednost S_i, kjer O_i pomeni zaporedje prvih i simbolov izhodnega zaporedja, S_i pa stanje sistema v času i. Začetne vrednosti teh verjetnosti je lahko določiti, saj preprosto pomnožimo verjetnosti, da se sistem nahaja v posameznem stanju, z verjetnostmi, da sistem odda prvi simbol izhodnega zaporedja, kot je razvidno iz formule 19.

$$p(O_1|S_1) = \pi(S_1) * B_{S1}(O_1) \qquad (19)$$

V vsakem naslednjem koraku nato preprosto izračunamo verjetnost prehoda v vsa možna stanja ter verjetnost oddaje prihodnjega simbola izhodnega zaporedja. Poti, ki se končajo v istem stanju, nato ponovno združimo in proces nadaljujemo. Ko dosežemo zadnji korak, združimo vse verjetnosti neodvisno od stanja, v katerem se naš sistem nahaja, ter tako pridobimo skupno verjetnost oddanega zaporedja.

b) Problem razpoznave:

Če poznamo izhodno zaporedje O ter model λ, katero zaporedje notranjih stanj S nam da najvišjo verjetnost P(O,S|λ). Tako kot pri prvem problemu bi se tudi v tem primeru pristop z grobo silo izkazal za neobvladljivega, zato za določitev rešitve uporabimo Viterbijev postopek.

c) Problem učenja:

Do zdaj smo predpostavljali, da so komponente našega modela poznane. Vendar pa je ravno določitev teh komponent najtežji problem prikritih Markovskih modelov. V nasprotju z

[7] forward-backward

do zdaj predstavljenimi problemi namreč ne poznamo analitične metode, po kateri bi določili optimalne koeficiente modela, tako da bi maksimizirali verjetnost, da je naš model generiral predhodno znano množico učnih podatkov (izhodnih zaporedij). Namesto tega moramo uporabiti iterativne metode, kot je postopek Baum-Welch.

Preden lahko začnemo postopek izvajati, se moramo odločiti o topologiji našega modela. Na podlagi predhodnega znanja moramo določiti tako število stanj kot tudi omejitve pri prehodih med njimi. V nekaterih primerih, na primer tudi pri razpoznavi govora, je namreč bolje uporabiti tako imenovane levo-desne HMM modele, pri katerih ni mogoče prehajati iz stanja z višjim indeksom v stanje z nižjim. S takšnimi omejitvami lahko že predhodno izboljšamo končno natančnost našega sistema.

Ko je topologija modela določena, nad njim poženemo postopek Baum-Welch, ki postopoma izboljšuje naš model, dokler bodisi ne dosega več opaznih izboljšav natančnosti bodisi doseže predhodno določeno maksimalno dovoljeno število iteracij. Tako pridobljeni rezultati pomenijo lokalni optimum, vendar se je treba zavedati, da so dosežene vrednosti odvisne od začetnega izbora parametrov (A,B,π), zato je postopek pametno pognati večkrat z različnimi začetnimi vrednostmi.

2.4 Zvezni prikriti Markovski modeli

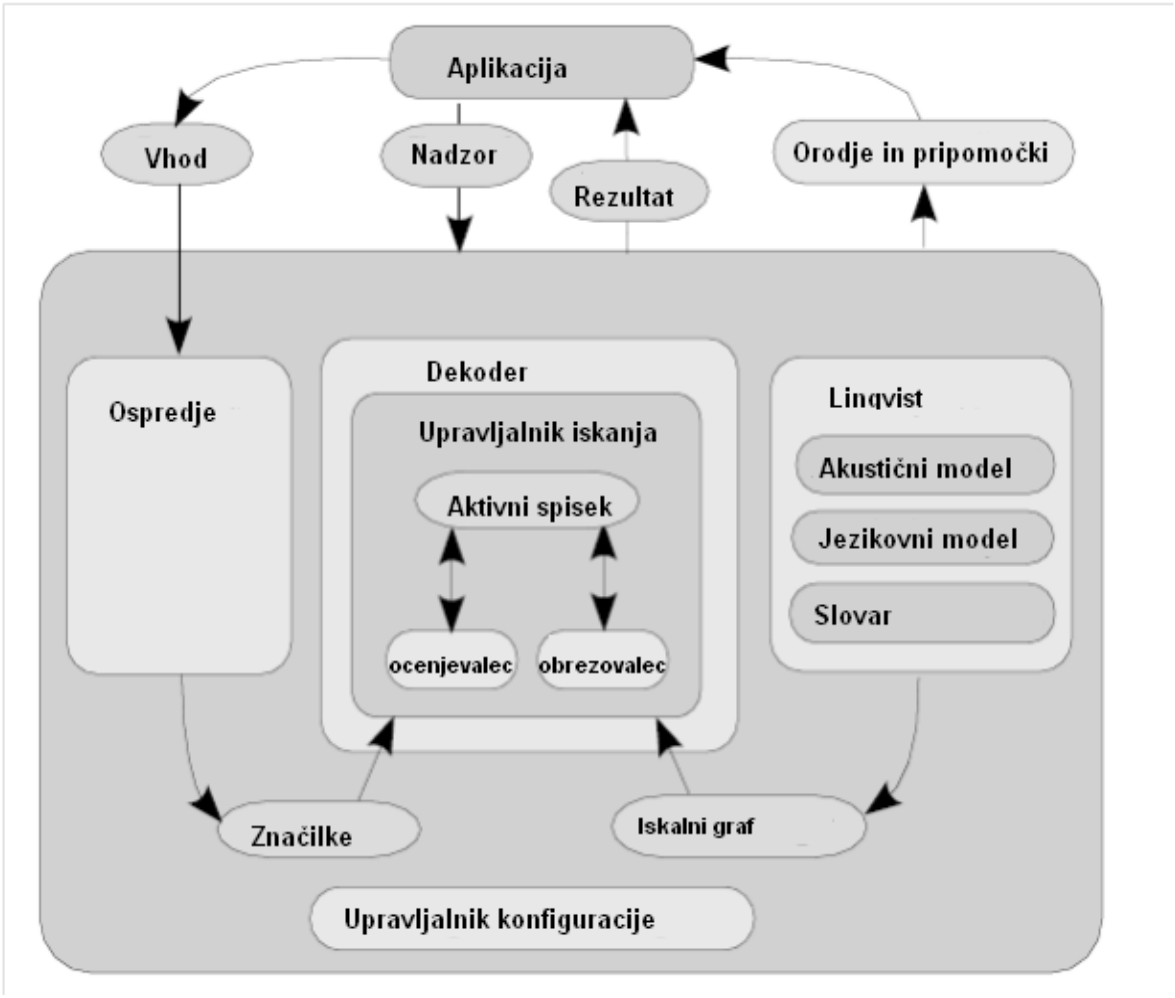
Z običajnimi (diskretnimi) prikritimi Markovskimi modeli je s postopkom vektorske kvantizacije sicer mogoče modelirati tudi zvezne procese, vendar s tem v sistem vnesemo kvantizacijsko napako. Boljša rešitev je sprememba definicije sistema, po kateri možni izhodni simboli niso več diskretni, ampak so predstavljeni z zveznimi vrednostmi, oziroma bolj splošno z vektorji zveznih vrednosti. V teh sistemih verjetnostno gostoto $b_i(x)$ nadomestimo z zvezno gostoto $b_i(x)dx$, ki predstavlja verjetnost, da opazovani vektor leži med x in x+dx. Pogosto uporabljena oblika teh gostot so Gaussove M-komponentne mešanice gostot oblike, razvidne iz formule 20.

$$b_i(x) = \sum_{k=1}^{M} c_{ik}\, \mathrm{N}[x, \mu_{ik}, U_{jk}] \qquad (20)$$

V tej formuli je c_{ik} utež mešanice, N je normalna gostota in μ_{ik},U_{jk} sta srednji vektor ter kovariančna matrika, povezana s stanjem i, mešanice k.

3. Sistem Sphinx-4

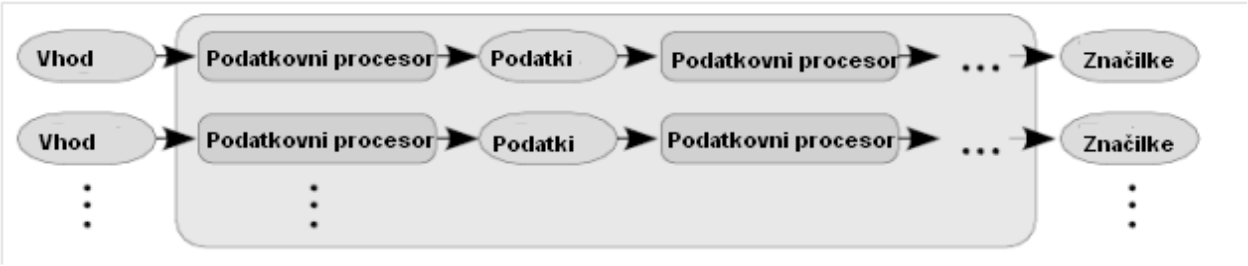
Sphinx-4 je odprtokodni sistem za razpoznavo govora univerze Carnegie Mellon[8], v celoti napisan v programskem jeziku Java [5]. Medtem ko so druga orodja za razpoznavo govora[9] osredotočena na le en pristop k razpoznavi govora, Sphinx-4 uporabnikom ponuja izredno prilagodljivost ter modularnost. To dejstvo, v povezavi z veliko prenosljivostjo javanskih programov, omogoča uporabo sistema Sphinx-4 na širokem razponu sistemov ter nalog. Sistem sestavljajo trije glavni moduli, in sicer ospredje, lingvist ter dekoder, celotni sistem pa sestavimo z uporabo upravljalnika konfiguracije, katerega nastavitve je mogoče bodisi določiti ob zagonu programa, bolj pogosto pa se zanj uporaljajo XML konfiguracijske datoteke.



Slika 4: Struktura sistema Sphinx-4

[8] Carnegie Mellon University, http://www.speech.cs.cmu.edu/
[9] HTK, ISIP, AVCRS in starejše verzije sistema Sphinx

3.1 Ospredje



Slika 5: Struktura ospredja

Naloga ospredja je parametrizacija vhodnega signala (zvoka) v zaporedje izhodnih značilk. Sestavljeno je iz ene ali več vzporednih verig podatkovnih procesorjev. Ta struktura omogoča vzporedni izračun več vrst značilk na podlagi enega ali celo več vhodnih signalov. Vsak posamezni podatkovni procesor ima vhod in izhod, ki sta lahko povezana na druge podatkovne procesorje, kar omogoča gradnjo poljubno dolgih verig. Med seboj si izmenjujejo generične podatkovne[10] objekte, ki enkapsulirajo obdelane vhodne podatke ali pa zaznamke, kot je na primer zaznan konec govora. Zadnji procesor v vsaki verigi je zadolžen za generiranje značilk, ki jih ospredje posreduje dekoderju.

Podatkovni procesorji nad vhodnimi signali izvajajo širok razpon operacij, kot so okenske funkcije, DFT, mel-frekvenčno filtriranje, diskretna kosinusna transformacija ... Komunikacija med posameznimi podatkovnimi procesorji temelji na principu 'povleci', pri katerem vsak procesor generira svoj izhodni podatek šele takrat, ko to zahteva procesor, ki mu v verigi sledi. Ta struktura omogoča tako medpomnjenje kot tudi pregled rezultatov skozi čas. To dekoderju omogoča uporabo širokega razpona iskalnih algoritmov, od iskanja v globino, algoritma A* do okensko sinhroniziranega Viterbijevega iskanja.

3.2 Lingvist

Lingvist izdela **iskalni graf**[11], ki ga pri iskanju uporablja dekoder. Tipična realizacija lingvista sestavi iskalni graf na podlagi strukture jezika, vsebovanega v jezikovnem modelu[12], ter topološke strukture akustičnega modela, za preslikavo med besedami, vsebovanimi v jezikovnem modelu, ter enotami akustičnega modela[13] pa uporablja slovar[14].

Jezikovni model vsebuje strukturo jezika na ravni besed. Obstaja veliko različnih predstavitev te strukture, ki pa večinoma sodijo v eno izmed dveh kategorij: poznamo gramatike v obliki grafov, pri katerih vsako vozlišče predstavlja besedo, usmerjene povezave

[10] Data
[11] SearchGraph
[12] LanguageModel
[13] AcusticModel
[14] Dictionary

pa predstavljajo verjetnost prehoda med besedami. Druga predstavitev pa so stohastični N-gramski modeli, ki določajo verjetnost naslednje besede na podlagi predhodnih N-1 besed.

Trenutno Sphinx-4 med drugim vsebuje tudi sledeče implementacije jezikovnega modela:

- SimpleWordListGramar, ki določa gramatiko, sestavljeno iz preprostega spiska besed. Dodatni parameter določa, ali se lahko gramatika ponavlja ali ne. Če ponavljanja ne dovolimo, je model primeren le za razpoznavo ločenih besed, če ponavljanje dovolimo, pa je primeren za razpoznavo poljubnega zaporedja besed našega jezika.
- JSGFGrammar uporablja gramatike oblike JSGF[15], ki vsebujejo gramatike BNF oblike.
- LMGrammar, ki določa gramatike, osnovane na statističnih modelih jezika.
- SimpleNGramModel, ki podpira ASCII N-gramske modele v ARPA formatu. Ta implementacija ne poskuša optimizirati porabe spomina, zato je primeren le za manjše jezikovne modele.
- LargeTrigramModel, ki podpira prave N-gramske modele, ustvarjene z orodjem za statistično jezikovno modeliranje CMU-Cambridge. Implementacija optimizira porabo spomina ter je zato primerna tudi za uporabo z večjimi jezikovnimi modeli.

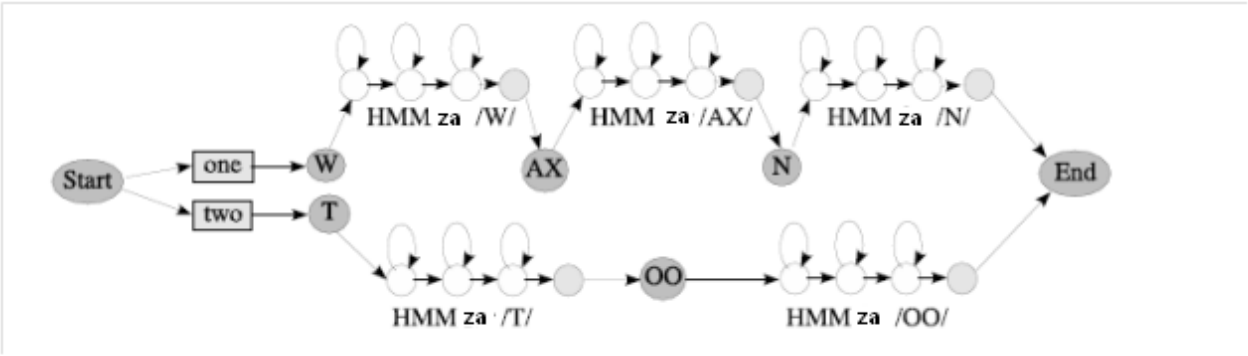
Slovar nam poda izgovarjave besed, ki jih najdemo v jezikovnem modelu. Te izgovarjave razdrobijo besede v zaporedja podbesednih enot, ki jih nato najdemo v akustičnem modelu.

Akustični model vsebuje podatke o preslikavi med enotami jezika ter prikritimi Markovskimi modeli, ki jih je nato mogoče ocenjevati v primerjavi z vhodnimi značilkami, ki nam jih posreduje ospredje. Ta preslikava je seveda lahko odvisna od konteksta ter pozicije v besedi. V primeru tri-fonemov, na primer, sta kontekst predhodni ter sledeči fonem, pozicija v besedi pa je lahko začetna, srednja ali končna. Sam Sphinx-4 ne predpisuje konteksta, ki ga mora akustični model upoštevati, zato je ta izbira prepuščena uporabnikom.

Sphinx-4 postavlja le malo omejitev pri sestavi HMM-jev, prisotnih v akustičnih modelih. Vsak HMM je predstavljen kot usmerjen graf, pri katerem vozlišča ustrezajo stanjem, povezave pa ustrezajo prehodom med stanji. Samo število stanj, povezav med stanji ter smer povezav niso predhodno določeni. Prav tako sistem ne določa metode, po kateri se izračuna prilagajanje med posameznim stanjem ter vhodnimi značilkami. To je prepuščeno akustičnemu modelu.

Iskalni graf je glavna podatkovna struktura, uporabljena med procesom dekodiranja. Je usmerjen graf, katerega vozlišča so poimenovana iskalna stanja[16]. Vsako vozlišče je lahko emisijsko ali pa neemisijsko[17]. Emisijska vozlišča je mogoče primerjati z vhodnimi značilkami, neemisijska pa pomenijo višje jezikovne konstrukte, kot so besede ter fonemi. Vse povezave v grafu imajo določene vrednosti, ki pomenijo verjetnost prehoda po posamezni povezavi.

[15] Java Speech API Grammar Format
[16] SearchState
[17] emitting ali non-emitting



Slika 6: Iskalni graf za besedi ‘one’ in ‘two’

Sphinx-4 vsebuje tri implementacije lingvista:

a) FlatLinguist, ki ob zagonu zgradi celotni iskalni graf, ter ga nato hrani v spominu. Odlikuje ga hitrost, vendar je tudi zelo prostorsko požrešen, zato ni primeren za uporabo z večjimi jezikovnimi modeli.
b) DynamicFlatLinguist, ki deluje podobno kot FlatLinguist, vendar iskalni graf dinamično gradi, ko se to od njega zahteva. To mu omogoča delo z večjimi jezikovnimi modeli, vendar tudi zmanjša njegovo hitrost.
c) LexTreeLinguist je namenjen delu z nalogami, ki imajo zelo obsežen besednjak ter uporabljajo N-gramske modele jezika. Njegova struktura omogoča uporabo teh modelov brez pretirane porabe pomnilnika.

3.3 Dekoder

Naloga dekoderja je generiranje hipotez o rezultatih na podlagi značilk, ki jih prejme od ospredja, ter iskalnega grafa, ki ga prejme od lingvista. Sestavljajo ga štirje moduli: upravljalnik iskanja ter njegovi podmoduli: aktivni spisek, ocenjevalec ter obrezovalec.

Upravljalnik iskanja na podlagi vhodnih značilk z uporabo ocenjevalca ocenjuje poti skozi iskalni graf ter jih shranjuje v aktivnem spisku. Med izvajanjem z uporabo obrezovalca omejuje število poti, ki jih mora preiskati.

Sphinx-4 vsebuje štiri implementacije upravljalnika iskanja:

- SimpleBreadtFirstSearchManager
- WordPruningBreadthFirstSearchManager
- BushderbySearchManager

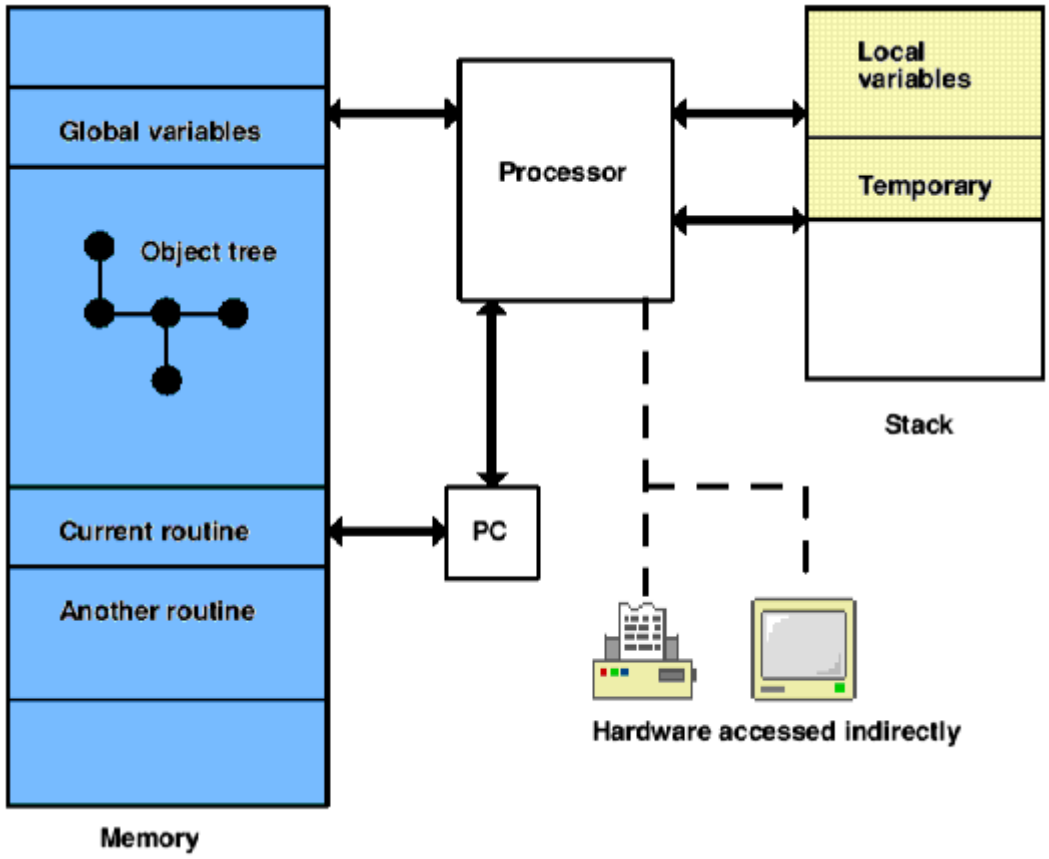
– ParallelSearchManager.

Odvisno od aplikacije, ki ga uporablja, lahko Sphinx-4 generira različne končne rezultate. Najpreprostejši rezultat je neposredna najbolje ocenjena rešitev, vendar lahko aplikacija zahteva tudi več podatkov, kot so na primer ocena zanesljivosti rešitve ter različne oblike množice najboljših rešitev.

4. Z-machine, ZIL ter ZPlet

Konec sedemdesetih let prejšnjega stoletja se je pojavila nova oblika računalniških igre: tekstovne avanture oziroma interaktivna fikcija[18]. Te igre so bile zaradi svojega preprostega uporabniškega vmesnika (tako vhod kot tudi izhod sta bila tekstovna) primerne za računalnike tistega časa, vendar so bile obenem nezdružljive z različnimi računalniškimi sistemi. Rešitev za ta problem je leta 1979 razvilo podjetje Infocom. Določilo je standard za navidezni stroj, poimenovan Z-machine oziroma Z-stroj [6], na katerega je mogoče gledati kot na duhovnega predhodnika Jave. Tako kot pri uporabi Jave je bil tudi cilj Z-stroja narediti programe neodvisne od strojne opreme, na kateri tečejo. Vsak program, pisan za Z-stroj, je bil sposoben teči na vsakem računalniku, za katerega je bil napisan interpreter jezika Z-stroja. Ta neodvisnost je podjetju omogočila, da je s svojimi igrami doseglo neprimerno večje občinstvo, kot bi ga, če bi podpiralo le manjše število platform. Zamisel se je izkazala za tako dobro, da standard Z-stroj še vedno živi, ljudje pa še vedno pišejo tako igre zanj kot tudi interpreterje za nove naprave, kot so na primer mobilni telefoni.

[18] Interactive Fiction



Slika 7: Arhitektura Z-stroja

Z-stroj je sicer dovolj splošen, da bi na njem lahko poganjali programe, pisane v poljubnem programskem jeziku (obstaja na primer prevajalnik jezika C za Z-stroj), vendar je bolje prilagojen za izvajanje programov, pisanih v Infocomovem jeziku **ZIL** [7] ter njegovih naslednikih. Ta jezik je bil razvit z namenom čim bolj preprosto, vendar tudi zelo prosto pisati tekstovne avanture. Temelji na uporabi treh struktur – objektov, akcij in rutin, ki so vezane na objekte in akcije. Objekti predstavljajo predmete, osebe in lokacije v igri, rutine pa dogodke, ki se vršijo glede na uporabnikove ukaze. Obravnava ukaza poteka tako:

a) Ukaz prevzame razčlenjevalnik[19] ter iz njega poskusi razbrati do tri parametre – akcijo, primarni objekt in sekundarni objekt. Če se mu to posreči, parametre preda preostalim rutinam, v nasprotnem primeru pa uporabniku posreduje obvestilo, da njegovega ukaza ne razume.

b) Program zaporedoma preveri rutine sekundarnega objekta ter primarnega objekta, ali so pripravljene prevzeti obravnavo parametrov. Če je obravnava sprejeta, se izvede rutina objekta, če pa oba objekta obravnavo zavrneta, se izvede privzeta rutina akcije.

c) Ko je obravnava ukaza končana, se izvedejo še rutine, vezane na potek časa v igri.

[19] parser

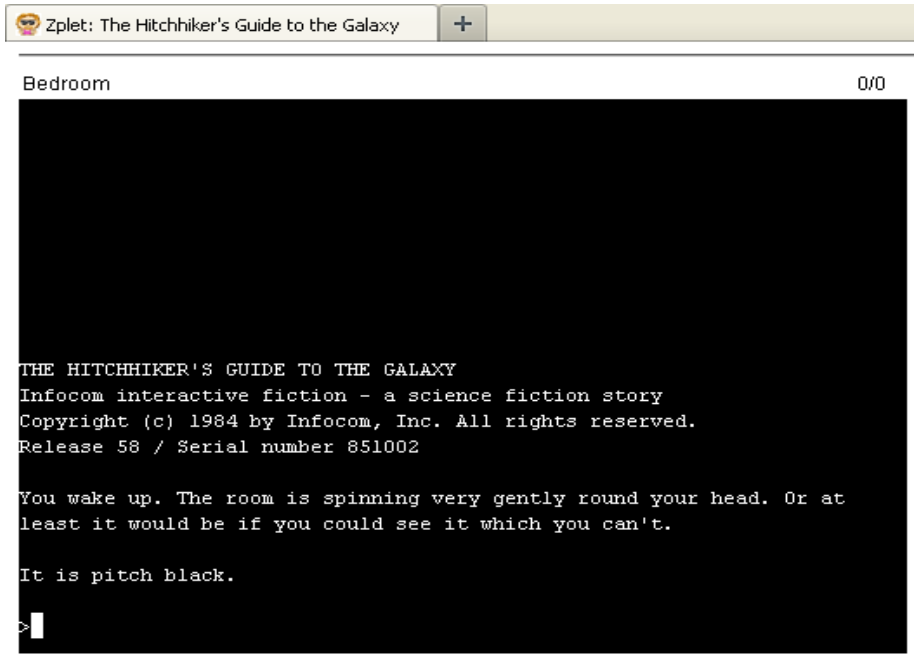
Razčlenjevalnik ukaze obravnava z dvema strukturama, ki se zgradita v procesu prevajanja v ZIL programa: slovarja in gramatike.

Slovar vsebuje vse besede, ki jih razčlenjevalnik lahko prepozna. V odvisnosti od verzije Z-stroja, za katerega je program preveden, se v slovarju besede okrajšajo. Tako se lahko na primer shrani le prvih nekaj črk vsake besede, na podlagi katerih se nato išče ujemanje z ukazi, ki jih prejme razčlenjevalnik.

Gramatika vsebuje spisek vseh mogočih akcij ter oblike ukazov, ki jih ustrezajo. Akcije namreč ne določa le glagol, prisoten v ukazu, temveč tudi njegov kontekst. Tako lahko na primer ukaz 'skrij X' pomeni drugo akcijo kot ukaz 'skrij se' ali pa ukaz 'skrij X za Y'. Po drugi strani se lahko več glagolov nanaša na isto akcijo, na primer 'poglej' ali 'preberi'.

Delovanje razčlenjevalnika je imelo pri pisanju ZIL programov vlogo črne skrinjice, saj je bila njegova koda priložena prevajalniku ter se piscu novih programov ni bilo treba ukvarjati z njo.

ZPlet je odprtokodni interpreter Z-stroja, napisan v Javi, in sicer v obliki apleta. Sicer ni najbolj napreden izmed trenutno razvitih interpreterjev, vendar ima dejstvo, da je napisan v Javi, dve veliki prednosti. Prva je prenosljivost med različnimi platformami, druga pa je preprosta povezljivost z drugimi javanskimi programi, kot je tudi sistem Sphinx-4.



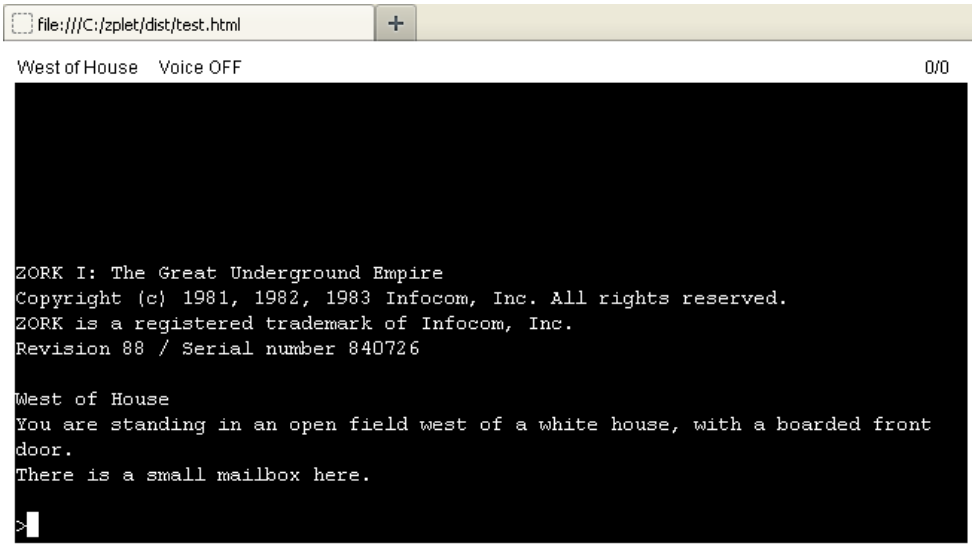
Slika 8: Interpreter ZPlet

ZPlet podpira najbolj razširjene verzije Z-machine programov (3,5,8), vhodno-izhodni del pa je od interpreterskega dela ločen. Zaradi tega je bilo mogoče implementirati skupno razpoznavo govora za vse tri verzije interpreterja.

5. Razširitev ZPlet z razpoznavo govora

Povezava interpreterja ZPlet ter sistema Sphinx-4 je imela več ciljev. Prvi je bil praktičen preizkus sistema Sphinx-4 ter njegove natančnosti zunaj umetno zastavljenih testnih primerov. Drugi cilj je bila posodobitev interpreterja ZPlet. Z napredkom tehnologije ter uporabniških vmesnikov je namreč vedno manj ljudi pripravljenih uporabljati programe, ki od njih zahtevajo uporabo ukazne vrstice. Razširitev programa z razpoznavo govora poveča njegovo dostopnost mlajšim generacijam ter jim obenem predstavi pomemben del zgodovine (in tudi sedanjosti) računalništva, ki sestoji iz interaktivne fikcije. Zadnji cilj je prikaz težav, ki se pojavijo pri govornem nadzoru interaktivne fikcije.

Dopolnjeni program se vizualno skoraj ne razlikuje od svoje nespremenjene različice, saj je edina opazna razlika indikator razpoznave govora v statusni vrstici, kot je razvidno na sliki 9.



Slika 9: Razširjeni interpreter ZPlet

Tudi samo delovanje programa se zaradi razširitve z razpoznavo govora ne spremeni. Daljši je le čas nalaganja ter povečana potreba po pomnilniku. Razpoznavo govora se vklopi in izklopi s pritiskom na tipko F1. To bo spremenilo stanje indikatorja (Voice ON) ter začelo proces razpoznave govora. Program kot vhod uporablja privzeto napravo za snemanje zvoka računalnika, na katerem teče.

Razpoznava govora poteka na ravni razpoznave posameznih ukazov, meje posameznega ukaza pa poskuša program določiti sam. Natančnost tega procesa je zelo odvisna od strojne ter programske opreme računalnika, zato je treba v skladu s tem ročno nastaviti občutljivost modula "SpeechClassifier". To je mogoče storiti v konfiguracijski datoteki, in sicer s spremembo parametra threshold:

```
<component name="speechClassifier"
  type="edu.cmu.sphinx.frontend.endpoint.SpeechClassifier">
 <property name="threshold" value="10"/>
</component>
```

Parameter določi najmanjšo razliko med močjo signala ter običajnim šumom, pri kateri bo signal prepoznan kot govor. Najboljšo vrednost parametra je mogoče določiti le na podlagi praktičnih izkušenj z opremo.

Ker je proces določitve meja govora pri določenih vhodnih napravah zelo nezanesljiv, je mogoče tudi ročno signalizirati programu, naj začne razpoznavo ukaza. To je mogoče storiti tako, da se zajem zvoka prekine s ponovnim pritiskom na tipko F1. Ko program določen ukaz prepozna, ga zapiše v vhodni medpomnilnik, kamor se drugače shranjujejo podatki o pritisnjenih tipkah. Od tu naprej se ukaz igri posreduje enako, kot da bi bil napisan s tipkovnico. Obnašanje igre se zaradi uporabe prepoznave govora v ničemer ne spremeni.

5.1 Prilagoditev igre na razširjeni interpreter ZPlet

Za zagon programa z novo igro potrebujemo sledeče datoteke:

- zplet.jar, ki vsebuje kodo programa ter sistem Sphinx-4
- zplet.html, ki vsebuje nastavitve interpreterja ZPlet
- datoteko z igro, ki jo želimo pognati
- akustični model
- datoteko z gramatiko, ki ustreza želeni igri
- zplet.config.xml, ki vsebuje nastavitve sistema Sphinx-4.

V datoteki zplet.html je mogoče prirediti videz programa, vključuje pa tudi izbor igre, ki jo želimo pognati, in sicer v obliki parametra "StoryFile".

```
<param name="Foreground" value="white" />
<param name="Background" value="black" />
<param name="StatusForeground" value="black" />
<param name="StatusBackground" value="white" />
<param name="FontFamily" value="Courier New" />
<param name="FontSize" value="14" />
<param name="StoryFile" value="zork1.z5" />
```

Pomembno je, da pri nastavitvi fonta izberemo font s fiksno dolžino znakov, saj se program pri izvajanju na to zanaša.

Datoteke z igrami so prosto dostopne na internetu. Kot izhodišče pri iskanju novih dogodivščin lahko služi stran http://www.ifwiki.org/ .

Akustični modeli ter pripadajoči slovarji so na voljo na domači strani sistema sphinx http://cmusphinx.sourceforge.net/, mojemu programu pa je že priložen model WSJ za angleščino. Če za jezik, ki ga želimo uporabljati, modela ni na voljo, je mogoče akustični model ustvariti z orodjem Sphinxtrain, prav tako na voljo na domači strani sistema.

Datoteko z gramatiko je treba ročno napisati za vsako igro, ki jo želimo pognati. Pri tem se lahko opiramo na orodja ztools, dostopna na strani http://www.inform-fiction.org/zmachine/ztools.html. Z njihovo pomočjo je mogoče dostopati do gramatike, vgrajene v posamezno igro, ter jo pretvoriti v obliko, ki jo lahko uporabi Sphinx-4. V prvem koraku je treba napisati pravila za vse akcije. Za to uporabimo orodje infodump s parametrom -g, katerega izhod je videti takole:

```
Story file is zork1.z5

    **** Parse tables ****

  Verb entries = 134

255. 1 entry, verb = "zork"
    [00 00 00 00 00 00 00 91] "zork"

254. 1 entry, verb = "scream", synonyms = "shout", "yell"
    [00 00 00 00 00 00 00 90] "scream"

253. 1 entry, verb = "wish"
    [00 00 00 00 00 00 00 8f] "wish"

252. 2 entries, verb = "wind"
    [01 fc 00 00 00 00 00 8e] "wind up OBJ"
    [01 00 00 00 00 00 00 8e] "wind OBJ"

251. 1 entry, verb = "win", synonyms = "winnag"
    [00 00 00 00 00 00 00 8d] "win"

250. 1 entry, verb = "wear"
    [01 00 00 00 00 00 00 67] "wear OBJ"

249. 2 entries, verb = "brandi", synonyms = "wave"
    [02 00 f3 00 00 ca 00 8c] "brandi OBJ at OBJ"
    [01 00 00 00 00 ca 00 8c] "brandi OBJ"

248. 10 entries, verb = "go", synonyms = "procee", "run", "step", "walk"
    [01 fa 00 18 00 30 00 1f] "go down OBJ"
    [01 fc 00 18 00 30 00 1e] "go up OBJ"
    [01 f1 00 00 00 00 00 8b] "go around OBJ"
    [01 ff 00 00 00 00 00 8a] "go to OBJ"
    [01 f6 00 00 00 00 00 45] "go over OBJ"
    [01 f9 00 00 00 00 00 22] "go on OBJ"
    [01 fe 00 00 00 00 00 22] "go with OBJ"
    [01 fb 00 00 00 00 00 22] "go in OBJ"
    [01 f7 00 00 00 00 00 89] "go away OBJ"
    [01 00 00 00 00 00 00 89] "go OBJ"
```

Slika 10: Del gramatike, uporabljene v igri Zork 1

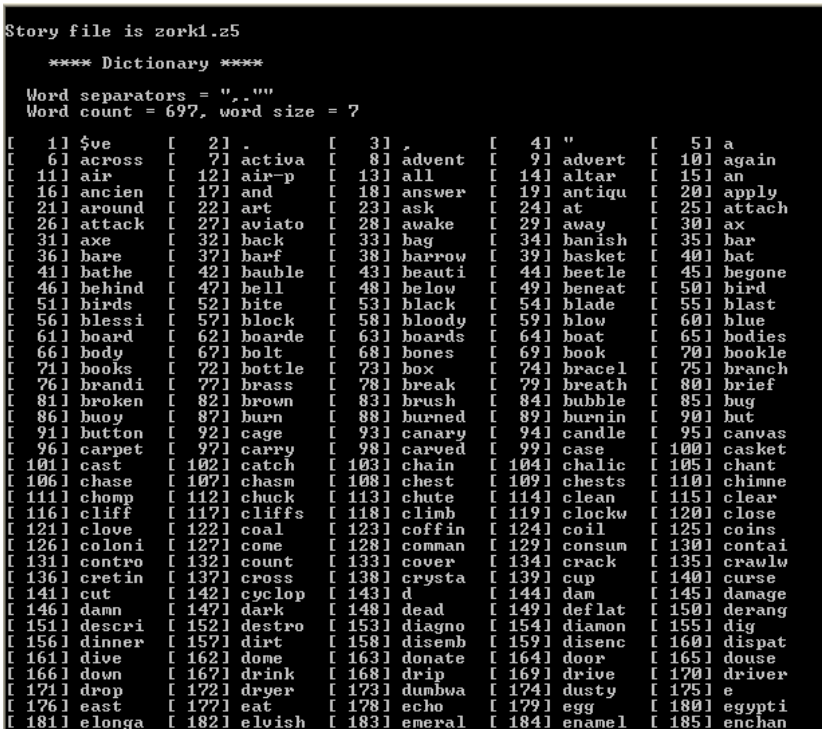
Kot primer vzemimo glagol "brandi". Vidimo lahko, da je dolžina besede v tu predstavljenem programu Zork1 omejena na le 6 črk, kar pomeni, da moramo najprej določiti celotno obliko glagola. Na podlagi sinonima "wave" lahko predpostavimo, da gre v tem primeru za okrajšavo besede "brandish". Pravilo za akcije, povezane s tem glagolom, nato pretvorimo v BNF[20]:

```
public <brandish>=((wave|brandish) <object> [at <object>]);
```

V večini primerov bo tako pridobljeni spisek pravil izredno obsežen. Ker velikost gramatike neposredno vpliva na hitrost ter spominsko potratnost našega programa, moramo pravila nato razredčiti. To storimo tako, da izpustimo manj znane sinonime akcij ter akcije, ki v igri niso pomembne. Če z igro nismo dobro seznanjeni, je za ta korak dobro uporabljati rešitev igre, ki vsebuje vse potrebne ukaze za zmago.

Zadnji korak pri izdelavi gramatike je generiranje spiska objektov. Le-tega pridobimo z uporabo orodja infodump s parametrom –d, kar nam omogoči dostop do slovarja.

[20] Backus-Naurjevo obliko (Backus-Naur form)



Slika 11: Del slovarja, uporabljenega v igri Zork 1

Iz vsebine slovarja je treba izluščiti vse samostalnike ter jih dodati v gramatiko pod skupno pravilo <object>:

```
<object> = (adventurer|pump|altar|axe|bag …….
```

Tako kot pri spisku akcij je treba tudi tu dopolniti daljše besede, saj je shranjen le njihov začetek. Ko je ta proces končan, moramo tudi spisek objektov razredčiti. Tu je proces še manj gotov kot pri redčenju spiska akcij, saj ne vemo, katere besede opisujejo isti objekt v igri.

Pred uporabo gramatike moramo še preveriti, ali so vse besede, ki so v njej uporabljene, prisotne tudi v našem slovarju. To lahko storimo tako, da preprosto zaženemo razširjeni ZPlet z našo gramatiko ter sledimo sporočilom o napakah. Če naš slovar kakšnih besed ne vsebuje, lahko zamenjamo akustični model ter slovar, popravimo našo gramatiko tako, da bo uporabljala drugačne sinonime, lahko pa slovar tudi razširimo, in sicer tako, da neznane besede in njihove izgovarjave ročno dodamo. Če izberemo zadnjo možnost, se moramo zavedati, da ta proces ni zanesljiv, saj ne vemo, ali je akustični model dovolj bogat, da bo novo besedo lahko prepoznal.

Izdelava gramatike je časovno zelo potratna, vendar dvomim, da bi bilo proces mogoče avtomatizirati. Zgradba gramatike v datotekah posameznih iger se namreč dovolj razlikuje, da bi bilo nemogoče napisati program, ki bi jo lahko avtomatično prepisal v primeren format. Prav tako se moramo zavedati, da bi bilo pri današnji stopnji umetne inteligence nemogoče napisati program, ki bi pravilno dopolnjeval nedokončane besede.

Kot zadnji korak pred zagonom ZPlet-a moramo še popraviti konfiguracijsko datoteko. Vsekakor moramo dodati pot do naše nove gramatike, in sicer tako, da podamo ime naše gramatike v sledečem parametru:

<property name="grammarName" value="**zork1**"/>

Če smo zamenjali tudi akustični model, je treba popraviti tudi nastavitve v konfiguracijski datoteki, ki se nanj nanašajo.

<property name="acousticModel" value="wsj"/>

```
<component name="dictionary"
  type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
  <property name="dictionaryPath"
    value="resource:/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/cmudict.0.6d"/>
  <property name="fillerPath"
    value="resource:/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/noisedict"/>
</component>

<component name="wsj"
  type="edu.cmu.sphinx.linguist.acoustic.tiedstate.TiedStateAcousticModel">
      <property name="loader" value="wsjLoader"/>
      <property name="unitManager" value="unitManager"/>
 </component>

 <component name="wsjLoader"
  type="edu.cmu.sphinx.linguist.acoustic.tiedstate.Sphinx3Loader">
      <property name="logMath" value="logMath"/>
      <property name="unitManager" value="unitManager"/>
      <property name="location"
        value="resource:/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz"/>
</component>
```

6. Testiranje in rezultati

Prva faza testiranja natančnosti sistema Sphinx-4 je potekala z regresijskimi testi[21]. Njihov namen je avtomatično merjenje natančnosti prepoznave govora na določeni testni množici, kar omogoča hitro primerjavo rezultatov v odvisnosti od sprememb v sistemu.

Regresijski testi so v sistemu Sphinx-4 že vgrajeni, zato za njihovo uporabo potrebujemo le sledeče datoteke:

- akustični model
- slovar
- jezikovni model
- batch datoteko
- testne podatke
- konfiguracijsko datoteko.

Uporabljeni akustični model, slovar ter jezikovni model so bili enaki kot tisti, uporabljani v interpreterju ZPlet. Prav tako so se ujemali parametri konfiguracijske datoteke. Batch datoteka vsebuje spisek testnih datotek ter opis njihove vsebine. Sphinx-4 nad vsako datoteko izvrši prepoznavo govora, nato pa rezultat primerja z opisom. Če se rezultat ter vsebina ne ujemata, te napake delimo na tri razrede:

- zamenjave[22] (S), pri katerih sistem napačno prepozna besedo
- vrinjanja[23] (I), pri katerih sistem napačno razmeji govorni signal ter razpozna neobstoječe besede
- brisanja[24] (D), pri katerih sistem ne razmeji in zato tudi ne razpozna besede.

Skupno napako[25] (WER) izračunamo na sledeči način:

$$WER = 100\% * \frac{S+I+D}{N} \qquad (21)$$

kjer je N skupno število besed v testnih podatkih.

Testi so bili izvedeni na dveh testnih množicah. Prva je obsegala ukaze, posnete z brezžičnim mikrofonom s slušalkami na glavi, druga pa z namiznim mikrofonom, oddaljenim približno pol metra. Obe testni množici sta vsebovali po 50 ukazov, ki jih je sestavljalo od ene do pet besed. Posamezni ukaz je bil pravilno razpoznan tedaj, ko so bile pravilno razpoznane vse besede, ki so ga sestavljale.

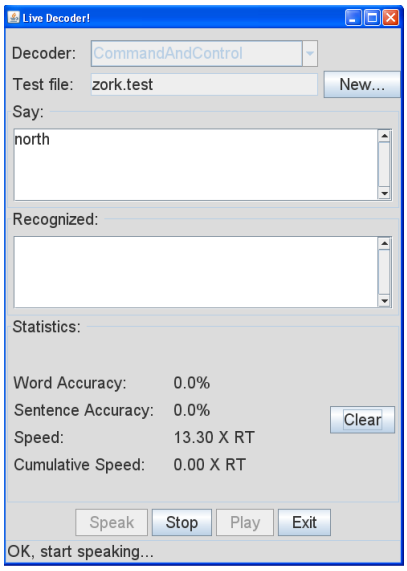
[21] angl. regression tests
[22] angl. substitutions
[23] angl. insertions
[24] angl. deletions
[25] angl. word error rate

	Mikrofon s slušalkami	Namizni mikrofon
Natančnost razpoznave besed	99 %	94 %
Natančnost razpoznave ukazov	98 %	88 %
Število napak	1	7
Zamenjave	1	6
Vrinjanja	0	1
Brisanja	0	0
Število besed	100	100
Število pravilno razpoznanih besed	99	94
WER	1 %	7 %

Tabela 1: Natančnost razpoznave posnetkov

Kot je razvidno iz rezultatov, je prepoznava posnetkov govora izredno natančna. Na žalost pa se rezultati niso ujemali s praktičnimi izkušnjami o natančnosti pri uporabi razpoznave govora v interpreterju ZPlet. Glavna razlika med regresijskimi testi ter praktično uporabo je bila v tem, da so testi kot vhodne podatke prejeli posnetke govora, ZPlet pa je govor neposredno zajemal z mikrofonom. Zato so bili potrebni še testi natančnosti v primeru neposrednega zajema govora.

Druga faza testov je kot testno orodje uporabila program Live, ki je prav tako priložen Sphinx-4. Tudi ta program za delovanje potrebuje akustični model, slovar, jezikovni model ter konfiguracijsko datoteko. Tako kot pri regresijskih testih so se tudi tu vse nastavitve ujemale s tistimi, uporabljenimi pri interpreterju ZPlet.



Slika 12: Program Live

Namesto batch datoteke program Live uporablja testno datoteko, ki vsebuje le zaporedje izjav. Med delovanjem program izjave zaporedno izpisuje na zaslonu ter od uporabnika pričakuje, da bo izjavo tudi izrekel. Prepoznani govor nato primerja z izpisano izjavo ter na podlagi tega računa natančnost.

Glede na nastavitve zajem govora poteka na tri različne načine. V prvem načinu mora uporabnik sam (s klikom na gumb) označiti začetek in konec svojega govora. V drugem

načinu uporabnik označi le začetek govora, konec pa program določi sam. Tretji način določanje začetka in konca govora prepusti izključno programu. Testi so bili izvedeni v tretjem načinu delovanja.

	Mikrofon s slušalkami	Namizni mikrofon
Natančnost razpoznave besed	30 %	74 %
Natančnost razpoznave ukazov	28 %	70 %

Tabela 2: Natančnost razpoznave govora

Tu se je izkazalo, da ima Sphinx-4 težave pri določanju delov zvoka, iz katerih sestoji govor, in zato drastično slabše prepoznava živi govor Pokazala se je tudi velika razlika med razpoznavo govora, zajetega z različnima napravama. Medtem ko je natančnost namiznega mikrofona padla le za približno 20 %, se je rezultat brezžičnega naglavnega mikrofona spustil iz skoraj popolne natančnosti na le 30 %, kar je treba pripisati večji stopnji šuma pri uporabi brezžične naprave. V prisotnosti močnega šuma se namreč pogosto zgodi, da ospredje sistema Sphinx-4 napačno označi dele govora kot negovor ter jih izloči iz signala, ki vstopa v proces razpozave govora. Zato je treba ukaze izreči precej glasneje kot pri običajnem govoru, kar spremeni tudi preostale lastnosti govora, to pa ima hude posledice za natančnost razpoznave.

Problem močnega šuma je mogoče omiliti z uporabo Wienerjevega filtra, ki omeji vpliv šuma na vhodni signal. Tako filtrirani signal je nato precej lažje klasificirati v dele, ki vsebujejo govor, ter dele, ki govora ne vsebujejo. Vendar tudi ta filter ne reši vseh težav neposredne razpoznave govora, saj filter obenem tudi spremeni posneti govor, kar zmanjša natančnost razpoznave. Kljub temu pa so se rezultati testa približali rezultatom, pridobljenim pri razpoznavi predhodno posnetega govora.

	Mikrofon s slušalkami	Namizni mikrofon
Natančnost razpoznave besed	82 %	78 %
Natančnost razpoznave ukazov	76 %	72 %

Tabela 3: Natančnost razpoznave govora z uporabo Wienerjevega filtra

Iz rezultatov je razvidno, da upravljanje z uporabo razpoznave govora še vedno ne dosega popolne natančnosti. Po drugi strani pa se moramo zavedati, da tudi pri vnosu ukazov s tipkovnico pogosto naletimo na napake, saj hitro pride do napačnega črkovanja besed. Tu se mora vsak uporabnik sam odločiti, ali je bolje, da je za napačen vnos odgovoren računalnik (v primeru uporabe razpoznave govora), ali pa njegova lastna površnost (v primeru uporabe tipkovnice).

7. Sklep

Pri izdelavi te diplomske naloge se je sistem Sphinx-4 izkazal kot precej preprost za uporabo. Sistemu je priloženih dovolj primerov uporabe, ki so tudi dobro dokumentirani, da razvoj aplikacij, ki sistem uporabljajo, ne pomeni večje težave. Težave pa se pojavijo, ko želimo sistem bolje prilagoditi posamezni nalogi. Dokumentacija mnogih nastavitev je namreč pomanjkljiva, ali pa je celo sploh ni. Prav tako so v priloženih primerih prisotne napake, kar za novega uporabnika nikakor ni prijetno.

Razširjeni interpreter ZPlet se je izkazal za zelo prilagodljivega. Z njim je brez posegov v izvorno kodo mogoče poganjati vse igre, ki jih je podpirala osnovna verzija ZPleta, če smo le pripravljeni v to vložiti nekaj časa ter napisati gramatiko, ki igri ustreza. Natančnost razpoznave govora se je izkazala kot zadovoljiva, vendar je pri uporabi nekaterih vhodnih naprav treba v programu ročno označevati konce ukazov ali pa uporabiti filtriran vhod, ki zmanjša natančnost razpoznave. Uporabnik pa mora še vedno pokazati nekaj potrpljenja, saj bodo zaradi nenatančnosti razpoznave govora nekateri njegovi ukazi še vedno napačno prepoznani.

Sistem Sphinx obsega mnoga področja, primerna za prihodnje raziskave. Najprej naj omenim uporabo sistema v kombinaciji z dinamično gramatiko, ki bi se prilagajala dogajanju v programu, ki bi jo uporabljal. Ta pristop je bil sprva mišljen tudi za ZPlet, vendar se je kmalu izkazalo, da bi ga delovanje Z-stroja naredilo nepraktičnega.

Drugo obetavno področje bi bila izdelava akustičnega modela, prilagojenega uporabi Wienerjevega filtra. Trenutno dostopni modeli namreč predpostavljajo, da vhodnega signala ni treba predhodno filtrirati, kar zmanjša natančnost razpoznave v primeru uporabe filtra.

Kot tretje, najobsežnejše območje moram omeniti še sistem Pocketsphinx, ki je zaradi nižjih strojnih zahtev bolj primeren predvsem za delo s prenosnimi napravami. Vendar je v primerjavi s Sphinx-4 slabše dokumentiran, kar lahko začetniku povzroči kopico težav.

V prihodnosti se bo računalniška razpoznava govora gotovo bolje razvijala ter širila na vedno več področij. Zato je pomembno, da se mladi računalničarji s tem področjem seznanijo, za kar je Sphinx-4 skoraj idealno orodje.

8.Priloge

8.1 Seznam slik

8.2 Seznam tabel

8.3 Seznam datotek na CD-ju

- razširjeni interpreter ZPlet ter njegova izvorna koda
- akustični model WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz
- slovar besed prisotnih v učnih podatkih akustičnega modela
- konfiguracijska datoteka sistema Sphinx-4 prirejena igri Zork 1
- konfiguracijska datoteka interpreterja ZPlet
- jezikovni model za igro Zork 1
- igra Zork 1
- izvod diplomskega dela v elektronski obliki
- naslov, povzetek in ključne besede diplomskega dela

9. Viri

[1] Basic concepts of Speech, dostopno na:
http://cmusphinx.sourceforge.net/wiki/tutorialconcepts
[2] Z. Kačič, "Komunikacija človek-stroj", Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, 1995, poglavje 5
[3] R. Rozman, "Nesimetrične okenske funkcije v sistemih za razpoznavo govora", doktorska disertacija, Fakulteta za računalništvo in informatiko, Ljubljana 2005, poglavje 2, dostopno na http://eprints.fri.uni-lj.si/772/1/doktorat_vse.pdf
[4] B. H. Juang, L. R. Rabiner, "An Introduction to Hidden Markov Models, IEEE ASSP Magazine, januar 1986
[5] Sphinx-4 Whitepaper: A Flexible Opensource Framework For Speech Recognition, dostopno na: http://cmusphinx.sourceforge.net/sphinx4/doc/Sphinx4Whitepaper.pdf
[6]Z-Machine standards document, dostopno na:
http://www.inform-fiction.org/zmachine/standards/z1point0/index.html
[7] Learning ZIL, dostopno na:
http://www.xlisp.org/zil.pdf