TAMPEREEN TEKNILLINEN YLIOPISTO

JARI-PEKKA RYYNÄNEN

AVAINSANOJEN TUNNISTUSRATKAISUJEN TESTAUS

Diplomityö

Tarkastajat: TkT Heikki Huttunen
TkT Tuomas Virtanen
Ohjaaja: DI Janne Mansikkamäki
Tarkastajat ja aihe hyväksytty tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 5. lokakuuta 2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
JARI-PEKKA RYYNÄNEN : Avainsanojen tunnistusratkaisujen testaus
Diplomityö, 45 sivua, 1 liitesivu
Kesäkuu 2012
Pääaine: Signaalinkäsittely
Tarkastajat: Heikki Huttunen ja Tuomas Virtanen
Avainsanat: Avainsanojen tunnistus, puheentunnistus, testaus

Tämä diplomityö käsittelee avainsanojen tunnistuksen testaamista. Avainsanojen tunnistus on eräs puheentunnistuksen osa-alue, jossa tunnistetaan yksittäisiä sanoja puhevirrasta. Tunnistuksen testaaminen tarkoittaa tässä työssä tunnistustarkkuuden ja laskennallisen tehokkuuden mittaamista. Työssä esitellään avainsanojen tunnistuksen testaamiseen käytettävän järjestelmän rakenne, sekä järjestelmään toteutetut yksittäiset testit.

Puheentunnistuksen ja avainsanojen tunnistuksen teoriaa esitellään lyhyesti, jonka jälkeen käsitellään erilaisia tapoja mitata tunnistustarkkuutta ja laskennallista tehokkuutta. Työhön liittyvän testausmateriaalin, ja sen käsittelyyn liittyvien seikkojen läpikäynnin jälkeen, esitellään työn toteutuksen kannalta oleelliset ohjelmistotekniikan teoriat. Tämän jälkeen esitellään varsinainen testausjärjestelmä. Esiteltyä testausjärjestelmää on onnistuneesti käytetty avainsanojen tunnistimien testauksessa, ja käytössä tehtyjä havaintoja ja huomioita esitellään työn lopuksi.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
JARI-PEKKA RYYNÄNEN : Evaluation of Keyword Spotting Systems
Master of Science Thesis, 45 pages, 1 Appendix page
June 2012
Major: Signal Processing
Examiner: Heikki Huttunen and Tuomas Virtanen
Keywords: Keyword spotting, speech recognition, testing

This thesis is about evaluation of keyword spotting systems. Keyword spotting is a subdivision of speech recognition, where individual words are identified from the speech flow. In this thesis, evaluation of keyword spotting systems means measuring the detection accuracy and the computational efficiency. This thesis presents the structure of the system which was designed for the evaluation, and the implemented individual tests.

Speech recognition and keyword spotting theories are presented briefly, after which various ways to measure the detection accuracy and the computational efficiency are considered. The testing material and its handling are considered next. After this the actual evaluation system is presented. The presented evaluation system has been used successfully in the evaluation of keyword spotting systems, and findings and observations made during the use are presented in the end of the thesis.

ALKUSANAT

Tämä diplomityö on tehty Insta DefSec Oy:lle vuosien 2011–2012 aikana. Mahdollisuudesta tämän diplomityön tekoon, sekä työllistymisestä, suurin kiitos kuuluu työn ohjaajalle Jannelle. Tuomakselle ja Heikille kiitokset suostumisesta työn tarkastajiksi. Lisäksi Heikille kiitos siitä että päädyin valitsemaan pääaineekseni juuri signaalinkäsittelyn; peruskurssien hyvät luennot vakuuttivat että signaalinkäsittely on mielenkiintoisempaa kuin ”pelkkä” ohjelmistotekniikka. Jälkikäteen voin todeta että valinta myös oli oikea.

Tampereella 6. päivänä kesäkuuta 2012

Jari-Pekka Ryynänen

SISÄLLYS

1. JOHDANTO

Puheentunnistus on ihmiseltä vaivattomasti onnistuva prosessi, jossa ilmanpaineen vaihteluista tulkitaan puhujan jakama informaatio. Tämän tapahtuman näennäinen yksinkertaisuus perustuu ihmisen hyvin pitkälle kehittyneisiin neuro- ja fysiologisiin valmiuksiin puheen tuottamiseksi ja ymmärtämiseksi. Kielen avulla ihmiset voivat käsitellä abstrakteja kokonaisuuksia, jotka eivät visuaalisessa maailmassa konkretisoidu.

Koneen tekemä puheentunnistus (jota tästä eteenpäin tarkoitetaan termillä puheentunnistus) on ollut jo pitkään muun muassa tekoälytutkijoiden mielenkiinnon kohteena. ”Puheentunnistus”-sanan merkitys on periaatteessa kontekstisidonnaista, sillä se voidaan ymmärtää sateenvarjoterminä johon sisältyvät erilaiset puheen luokitteluongelmat. Näistä esimerkkeinä ovat seuraavat:

- Avainsanojen tunnistus (engl. *Keyword Spotting, KWS*), jonka tavoitteena on tunnistaa tietyt yksittäiset sanat puheesta. Avainsanojen tunnistusta voidaan käyttää esimerkiksi puhelinpalveluissa tunnistamaan soittajan puheesta ennalta määriteltyjä sanoja, ja ohjaamaan puhelu oikealle asiakaspalvelijalle.

- Laajan sanaston jatkuvan puheen tunnistus (engl. *Large-Vocabulary Continuous Speech Recognition, LVCSR*), jonka tavoitteena on luoda tekstitys jatkuvasta puhevirrasta. Tässä diplomityössä käytetään paikoin termiä ”litterointi” tämän sijasta. Litterointia voidaan hyödyntää muun muassa videoiden automaattisessa tekstityksessä.

Lisäksi puheentunnistusta sivuavat siitä poikkeavat hahmontunnistusongelmat, kuten kielentunnistus ja puhujantunnistus. Näissä käytetään yleensä samantyyppisiä hahmontunnistustekniikoita, sovelluskohteen ollessa kuitenkin erilainen.

Tässä diplomityössä tutkitaan avainsanojen tunnistuksen mittaamista ja pyritään vastaamaan seuraavaan kysymykseen: miten ja millaisella ratkaisulla mitata annetun avainsanojen tunnistimen tunnistustarkkuutta ja laskennallista tehokkuutta? Työssä käytetään jonkin verran ohjelmiston testaamiseen liittyvää sanastoa. Tässä yhteydessä testillä kuitenkin tarkoitetaan mittausta, eikä ohjelmistotestauksessa käytössä olevaa tapaa ymmärtää esimerkiksi testitapaus ohjelmiston ominaisuuden tai toiminnallisuuden vertaamiseksi sen määrittelyyn [5, s. 287], jolloin testitapaus

voi olla joko hyväksytty tai hylätty.

Kuten puheentunnistusta, myös sen testausta on tutkittu jo vuosikymmeniä [17]. Kuitenkaan mitään kattavaa testauskehystä/-mallia ei ole julkaistu. Puheentunnistuksen ja avainsanojen tunnistuksen testaamiseen liittyvät artikkelit keskittyvät pääasiassa joko uusien menetelmien kehittämiseen (kuten [18]), tai testausjärjestelyjen esittelyyn, joissa käytetään vain yksittäisiä mittausmenetelmiä (kuten [19]). Tässä työssä käsitellään laajemmin testauksen käytännön toteutusta, ja pyritään koostamaan useampia mittausmenetelmiä yhden järjestelmän alle.

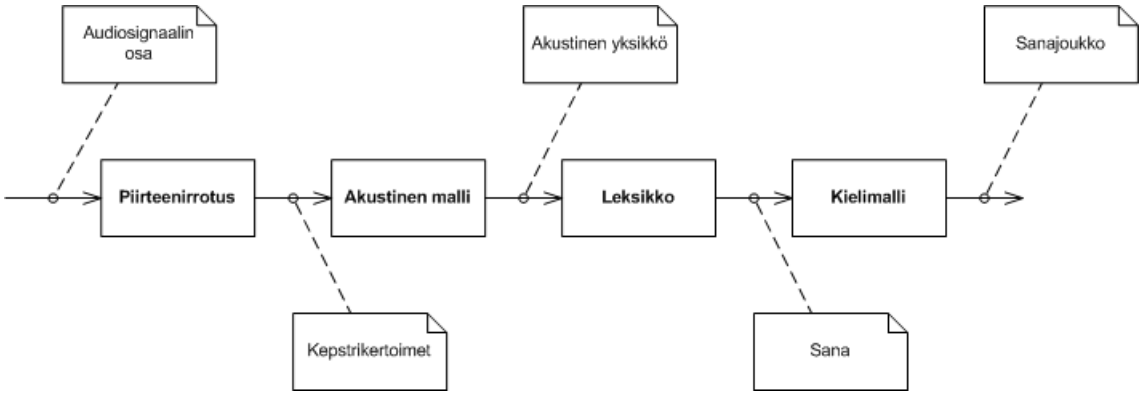
Työ noudattaa seuraavaa rakennetta: luvussa kaksi esitellään lyhyesti puheentunnistuksen ja ennen kaikkea avainsanojen tunnistuksen teoriaa. Luvussa kolme käydään läpi avainsanojen tunnistuksen mittaamiseen liittyviä asioita. Luvussa neljä käsitellään toteuttavan ohjelmiston ohjelmistoteknistä taustaa. Luvussa viisi käydään läpi käytössä olevaan testimateriaaliin liittyvät seikat. Luku kuusi sisältää varsinaisen testausohjelmiston toteutuksen kuvauksen, jota arvioidaan luvussa seitsemän. Luvussa kahdeksan tehdään yhteenveto työstä ja esitellään näkökulmia jatkokehitykseen.

2. PUHEENTUNNISTUKSEN TEORIAA

Tässä luvussa annetaan yleiskuva puheentunnistukseen ja avainsanojen tunnistukseen liittyvästä teoriasta, sekä selvennetään niihin liittyvää termistöä. Aiheisiin liittyvän teorian kattava läpikäynti on tässä työssä mahdotonta, eikä työn kannalta edes kovinkaan tarkoituksenmukaista.

2.1 Yleinen ongelmanasettelu

Puheentunnistustehtävät ovat usein tilastollisin menetelmin tarkasteltavia ongelmia, joissa näin käsiteltynä pyritään etsimään todennäköisin lause/sana/symboli havaintojen perusteella. Varsinaisena havaintona on akustinen signaali, josta jatkokäsittelyä varten lasketaan *piirteitä*. Piirteiden tarkoitus on kuvata signaalin sovellusalueen kannalta olennainen informaatio tehokkaammin ja yksinkertaisemmin. Signaalia käsitellään kehyksittäin, mikä tarkoittaa sen käsittelyä ajallisesti lyhyissä (mahdollisesti osittain päällekkäisissä) ikkunoissa. Kehyksiä käytetään, koska luonnollisestikaan muuten reaaliaikainen tunnistus ei olisi mahdollista, ja toisaalta koska puheen ajallisesti muuttuva taajuussisältö voidaan olettaa lyhytaikaisesti melko stabiiliksi (ihmisen ääntöväylä ei muuta muotoaan välittömästi [2, s. 37]). Puheentunnistuksessa käytettävinä piirteinä käytetään tavallisesti matalan asteen *kepstrikertoimia*, jotka käytännössä kuvaavat spektrin verhokäyrää. Kuvatunlainen tapa (kuva 2.1) hahmottaa puheentunnistin tarjoaa yhteisen käsiteympäristön aiheen käsittelyyn.



Kuva 2.1. *Puheentunnistimen yleinen malli. Mukailtu lähteestä [1, s. 618], jättäen semantiikkaan ja pragmatiikkaan liittyvät osat pois.*

Piirteenirrotuksesta saatavaa piirrejoukkoa verrataan *akustiseen malliin*, jonka perusteella piirrejoukkoon liittyvän kohdan signaalissa voidaan mahdollisesti todeta sisältävän tietyn akustisen yksikön, esimerkiksi tietyn äänteen. *Leksikko* määrää näiden akustisten yksiköiden koostamisen sanoiksi tai osasanoiksi, joista edelleen muodostetaan *kielimallin* perusteella sanayhdistelmiä ja lauseita. [1, s. 616–618.]

2.2 Grafeemit, foneemit ja foonit

Kirjoitettu teksti koostuu kirjoitusmerkeistä eli *grafeemeista*. Grafeemiesityksen muodostaminen on tavallinen tavoite puheentunnistuksessa, ja se on myös luonteva tapa ilmoittaa tunnistusjärjestelmälle mitä sanoja halutaan puheesta löytää. Grafeemien kytkeytyminen puhuttuun kieleen ei kuitenkaan ole aina kovinkaan yksinkertaista; käytännössä tämä nähdään erikielisten, mutta samaa kirjoitusmerkistöä käyttävien ihmisten taipumuksesta ääntää vieraskielisiä sanoja virheellisesti [9].

Askel kohti puhutun sanan muotoista kuvausta tehdään käyttämällä *foneemeja*. Foneemit ovat kielen pienimpiä merkityksiä erottavia äänneyksikköjä. Ne ovat idealisaatioita *fooneista*, jotka ovat varsinaisesti puheessa esiintyviä äänteitä [2, s. 36].

Puheen kuvaamista äänteenmukaisesti sanotaan *foneettiseksi transkriptioksi*, jonka tavoitteena on kuvata puhetta mahdollisimman tarkasti. Tästä yksinkertaisempana tapauksena on *foneeminen transkriptio*, jossa tietystä sanasta muodostetaan sen foneemiesitys. [9] Esimerkkinä näiden erosta on sanan ”saha” foneeminen (//-merkkien välissä) ja eräs foneettinen ([]-merkkien välissä) transkriptio alla

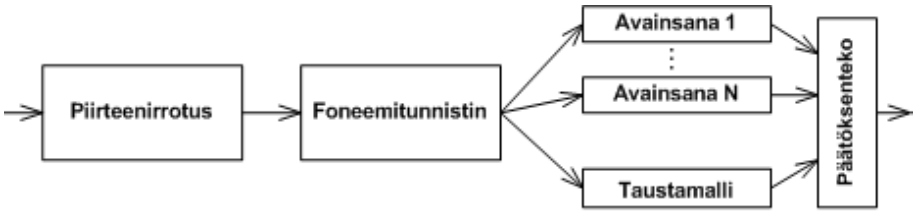
- /saha/
- ['safia]

Yllä olevassa tapauksessa foneemiesitys listaa vain sanassa olevat foneemit (joiden symbolit vastaavat tässä tapauksessa grafeemeja), kun taas foneettinen esitys sisältää myös tiedon ensimmäisen äänteen painottamisesta (['s]) sekä tässä sanassa olevan foneemin /h/ esiintymisestä "soinnillisena"([fi]) [15, s. 23–28].

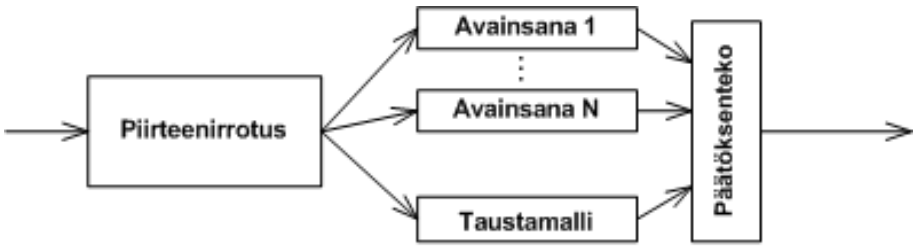
2.3 Avainsanojen tunnistus

Avainsanojen tunnistus on yksi puheentunnistuksen osa-alue. Avainsanojen tunnistuksessa pyritään tunnistamaan haluttuja avainsanoja jatkuvasta tai ”diskreetistä” puhevirrasta. Jälkimmäinen tarkoittaa puhetta, jossa sanat lausutaan selkeästi toisista erillään. Jatkuvasta puhevirrasta tapahtuva avainsanojen tunnistus on selkeästi hankalampi tehtävä johtuen sanojen epämääräisistä rajoista ja *koartikulaatiosta*, eli vierekkäisten äänteiden vaikutukseta yksittäiseen äänteeseen.

Vertaamalla kuvan 2.2 tunnistusprosessia kuvan 2.1 yleisempään tapaukseen, avainsanojen tunnistuksessa ei olla kiinnostuneita sanojen yhdistelmistä (yleisesti; tästäkin löytyy poikkeuksia [4, s. 17–18]). Tunnistimen toteutuksesta riippuen myöskin leksikkoa vastaava osa prosessista saattaa puuttua, jolloin kustakin avainsanasta on muodostettu suoraan oma akustinen mallinsa [8] (kuva 2.3).



Kuva 2.2. *Käsitteellinen kaavio avainsanojen tunnistimesta, joka etsii sanoja foneemijonosta. Avainsanoista on muodostettu foneeminen transkriptio.*



Kuva 2.3. *Käsitteellinen kaavio avainsanojen tunnistimesta, joka etsii sanoja kepstrikertoimista. Avainsanoista on muodostettu akustinen malli.*

Avainsanojen tunnistuksessa tarvitaan tunnistettavien avainsanojen lisäksi jonkinlainen taustamalli, jonka tehtävänä on kuvata kaikki muut puheessa esiintyvät sanat [4, s. 16]. Mikäli kaikkia puheessa esiintyviä sanoja ei ole mahdollista tietää etukäteen, kyseisen mallin pitäisi sisältää kaikki ennalta tuntemattomat sanat [4, s. 27]. Taustamallin toimintaa voi mahdollisesti joissain tunnistimissa parantaa antamalla *antiavainsanoja*, eli sanoja, jotka ilman erillistä määrittelyä sekoittuisivat helposti varsinaisten avainsanojen kanssa.

Avainsanojen tunnistusta voitaisiin myös tehdä käyttämällä jotain litterointiin kykenevää puheentunnistinta, ja etsimällä haluttu avainsana saadusta tekstinnöksestä. Käytännön ongelmaksi tälle kuitenkin muodostuvat litteroinnin heikohko toimivuus kun puheelle ei aseteta mitään rajoituksia, ja vaaditun tunnistuksen suurempi monimutkaisuus ja siitä johtuva kokonaisajankäytön kasvaminen (tekstinnöksen teon jälkeen haku olisi kuitenkin tietysti nopeaa). [8]

2.3.1 Käyttökohteet

Avainsanojen tunnistimen käyttökohteina voidaan periaatteessa nähdä melkein kaikentyyppiset puheentunnistustehtävät, mutta alla esitellään lyhyesti joitakin tunnistimelle luontevimmin soveltuvia käyttötapauksia.

- *Äänidokumenttien indeksointi* tarkoittaa äänidokumentteihin (eli äänitallenteisiin) kohdistuvaa hakua mielenkiinnon kohteina olevilla sanoilla.
- *Äänivirran monitorointi* tarkoittaa jatkuvaa reaaliaikaista äänisignaalin tarkkailua haluttujen avainsanojen osalta. Esimerkkinä radiolähetyksien tarkkailu mielenkiintoisten uutisten tallentamista varten.
- *Puhekäyttöliittymät* esimerkiksi DVD-soittimen ohjausta (eli muun muassa toiston käynnistämistä, levykelkan avaamista) varten.

Luetteloituna olevat ovat vain esimerkinomaisia tapauksia, mukailtuna lähteestä [4, s. 14–15].

2.3.2 Avainsanojen syöttö

Avainsanojen syöttö voi tapahtua avainsanojen tunnistimesta riippuen antamalla avainsana

- puheena,
- foneemisena transkriptiona tai
- grafeemiesityksenä.

Ensimmäinen vaihtoehto tarkoittaa puheentunnistinta jolle lausutaan haluttu avainsana. Tämä toimii opetusvaiheena, jonka jälkeen tunnistin vertailee varsinaisessa käytössä laskettavia piirteitä tämän opetusnäytteen pohjalta tehtyyn malliin avainsanasta. Toinen vaihtoehto vaatii avainsanojen valmistelun etsimällä niiden äänteen mukaisen esityksen käyttäjän toimesta. Viimeinen vaihtoehto tulee kyseeseen kun käytössä on litterointipohjainen avainsanojen tunnistin tai transkription automatisointi.

Mikäli tunnistin toimii kuten kuvassa 2.2, käytetään foneemista transkriptiota suoraan avainsanan mallina. Kuvan 2.3 tyyppisillä tunnistimilla foneemisesta transkriptiosta tai annetusta puhesyötteestä koostetaan akustinen malli. Edellisessä tapauksessa tämä tapahtuu yhdistelemällä ennalta opittuja foneemien akustisia malleja. [8] Jälkimmäisessä tapauksessa on luontevaa käyttää puhesyötteestä laskettuja

piirteitä mallin pohjana; tässä tapauksessa puheesta ei muodostu mitään esitystä joka olisi ihmisen tulkittavissa. Nämä osat prosessista tapahtuvat kuitenkin loppukäyttäjältä piilossa.

3. TUNNISTUSTEHOKKUUDEN MITTAAMINEN

Tunnistusprosessista voidaan mitata tunnistustarkkuuden lisäksi laskennallista tehokkuutta. Tässä luvussa käsitellään aiheiseen liittyvää yleistä teoriaa.

3.1 Tunnistustarkkuus

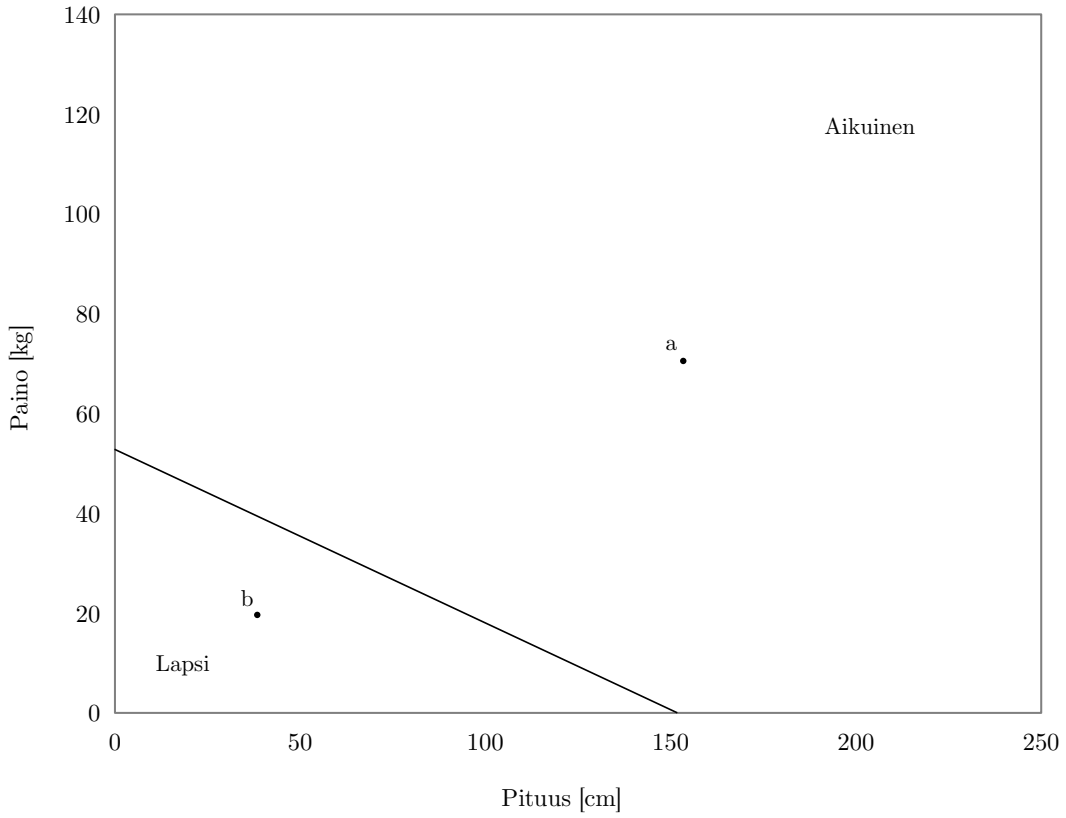
Avainsanojen tunnistuksen tuloksena voi olla jokin taulukon 3.1 soluista. Oikea positiivinen tarkoittaa äänisignaalissa todella esiintyvää avainsanaa, jonka myös tunnistin havaitsee, kun taas väärässä negatiivisessa tunnistin ei havaitse kyseistä avainsanaa. Väärää negatiivista kutsutaan myös *tyypin 2 virheeksi*. Oikea negatiivinen tarkoittaa tilannetta, jossa signaalissa ei esiinny mitään avainsanaa, eikä tunnistin signaalista myöskään mitään havaitse; väärässä positiivisessa tunnistin havaitsee jonkin avainsanan, jota ei signaalissa kuitenkaan esiinny. Väärää positiivista kutsutaan myös *tyypin 1 virheeksi*. Tilanne, jossa jokin äänisignaalissa todella esiintyvä avainsana havaitaan jonain toisena avainsanana, on tulkittavissa sekä tyypin 1 että 2 virheeksi. Taulukosta usein laskettavia arvoja ovat seuraavat [12, s. 155, 162]:

- Herkkyys (engl. *sensitivity/recall*), eli oikein tunnistettujen avainsanojen määrä suhteessa puheessa esiintyvien avainsanojen määrään ($S = \frac{\mathrm{op}}{\mathrm{op+vn}}$). Suuri herkkyysarvo tarkoittaa tunnistimen löytävän tehokkaasti puheessa esiintyvät avainsanat (pieni tyypin 2 virhe).

- Tarkkuus (engl. *precision*), eli oikein tunnistettujen avainsanojen määrä suhteessa kaikkien tunnistettujen avainsanojen määrään ($P = \frac{\mathrm{op}}{\mathrm{op+vp}}$). Suuri tarkkuusarvo tarkoittaa tunnistimen tekevän vähän vääriä havaintoja (pieni tyypin 1 virhe).

Taulukko 3.1. *Avainsanojen tunnistuksen mahdolliset tulokset (mukailtuna lähteestä [12, s. 155]).*

	Todellinen avainsana	Ei avainsana
Tunnistettiin avainsana	*Oikea positiivinen (op)*	*Väärä positiivinen (vp)*
Ei tunnistettu avainsanaa	*Väärä negatiivinen (vn)*	*Oikea negatiivinen (on)*

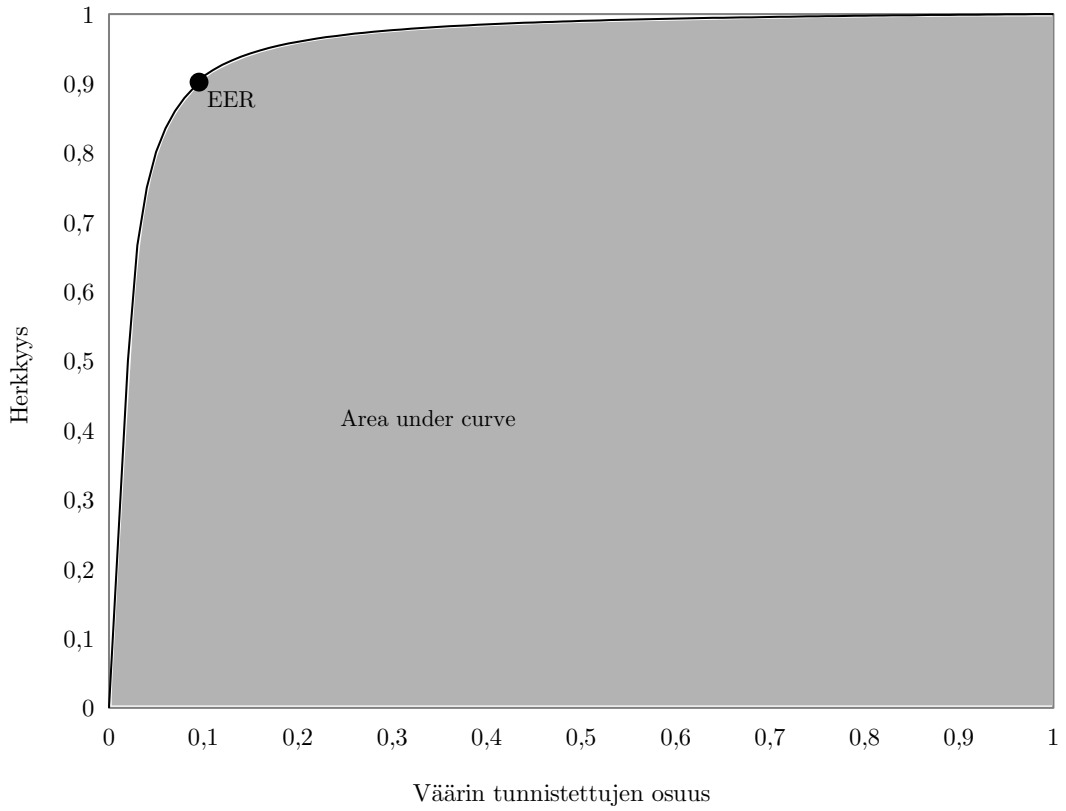


Kuva 3.1. *Esimerkki päätöspinnasta. Kuvitteellinen luokitin jakaa ihmiset aikuisiin ja lapsiin heidän painonsa ja pituutensa perusteella. Kuvaan piirretty viiva toimii päätöspintana, joka määrää pisteen a luokittuvan "Aikuinen"-luokkaan ja pisteen b "Lapsi"-luokkaan.*

- Väärin tunnistettujen osuus (engl. *False Positive Rate, FPR*), eli väärin tunnistettujen avainsanojen määrä suhteessa puheessa esiintyvien ei-avainsanojen määrään ($\mathrm{FPR} = \frac{\mathrm{vp}}{\mathrm{vp+on}}$). Suuri FPR-arvo tarkoittaa tunnistimen tekevän paljon vääriä havaintoja (suuri tyypin 1 virhe).

Yllä olevan listan mitat ovat riippuvaisia tunnistimen päätöspinnan sijainnista. Päätöspinta on tunnistimen rajanveto eri luokkien välillä [10, s. 4-5] (esimerkki kuvassa 3.1, johon on valittu yksinkertainen tapaus selvyyden vuoksi). Käytännössä tätä voidaan usein kontrolloida jollain säätöparametrilla, jolloin tutkittavaksi saadaan joukko herkkyys/tarkkuus/FPR-arvoja.

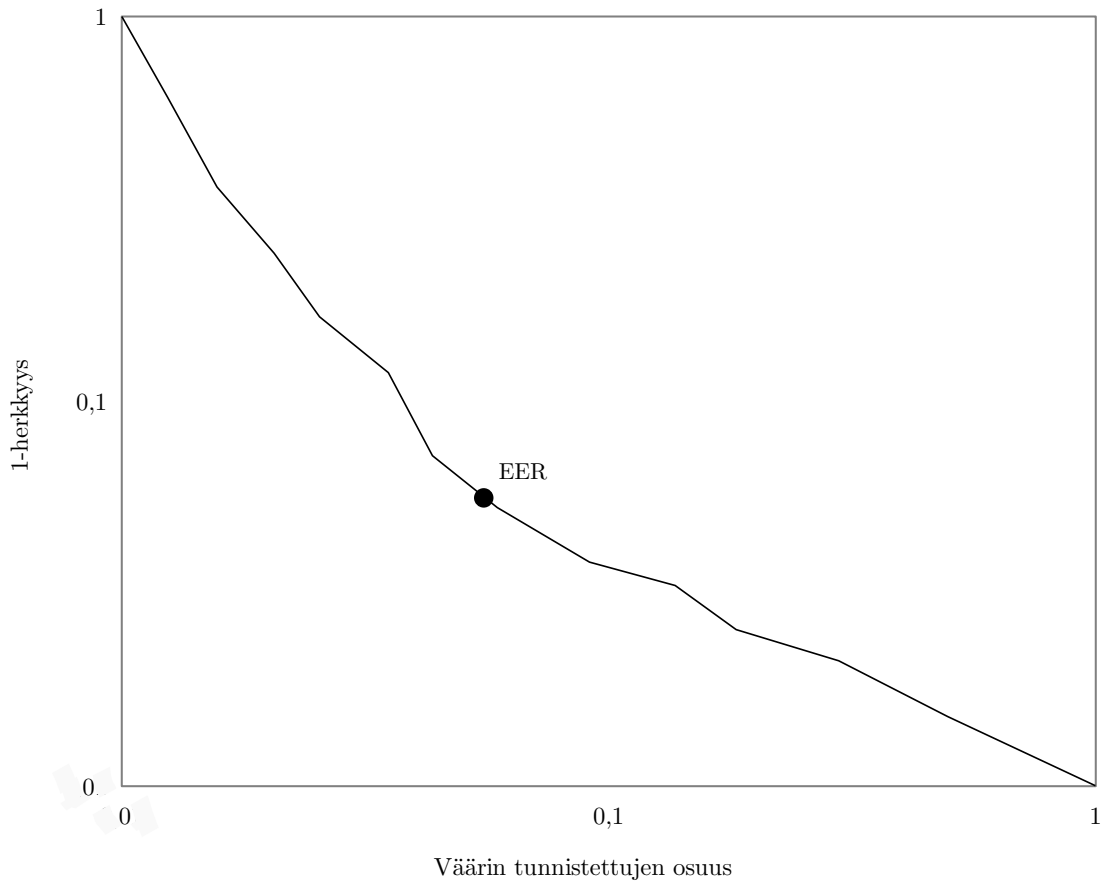
Herkkyyden kuvaamista väärin tunnistettujen osuuden (joka vastaa tyypin 1 virhetodennäköisyyttä) funktiona kutsutaan *ROC-käyräksi* (*Receiver Operating Characteristics*, kuva 3.2) [12, s. 162]. Usein kiinnostuksen kohteena olevat pisteet käyrissä sijaitsevat pienillä virheosuuksilla, ja niiden tarkempaa tarkastelua varten voidaan käyttää ROC-käyrään läheisesti liittyvää *DET-käyrää* (*Detection Error Trade-*



Kuva 3.2. *Esimerkki ROC-käyrästä. Tunnistin on sitä parempi mitä lähempänä vasenta yläkulmaa sen ROC-käyrä kulkee.*

off, kuva 3.3), joka käyttää logaritmista asteikkoa, tarkoituksena parantaa kuvaajan resoluutiota pienillä virhearvoilla. DET-käyrässä käytetään herkkyyden S sijasta sen ”komplementtia” $(1 - S)$ (joka vastaa tyypin 2 virhetodennäköisyyttä). [4, s. 25.] Nämä käyrät antavat hyvän käsityksen tunnistimen toiminnasta suhteellisen tasaisesti jakautuneissa luokissa, mutta mikäli luokkajakauma on selvästi vinoutunut, voi parempana vaihtoehtona olla *PR-käyrä* (*Precision-Recall*, kuva 3.4), jossa tarkkuus kuvataan herkkyyden funktiona. [14] Kaikkien käyrien tarkoituksena on kuvata kahden virhetyypin keskinäistä suhdetta.

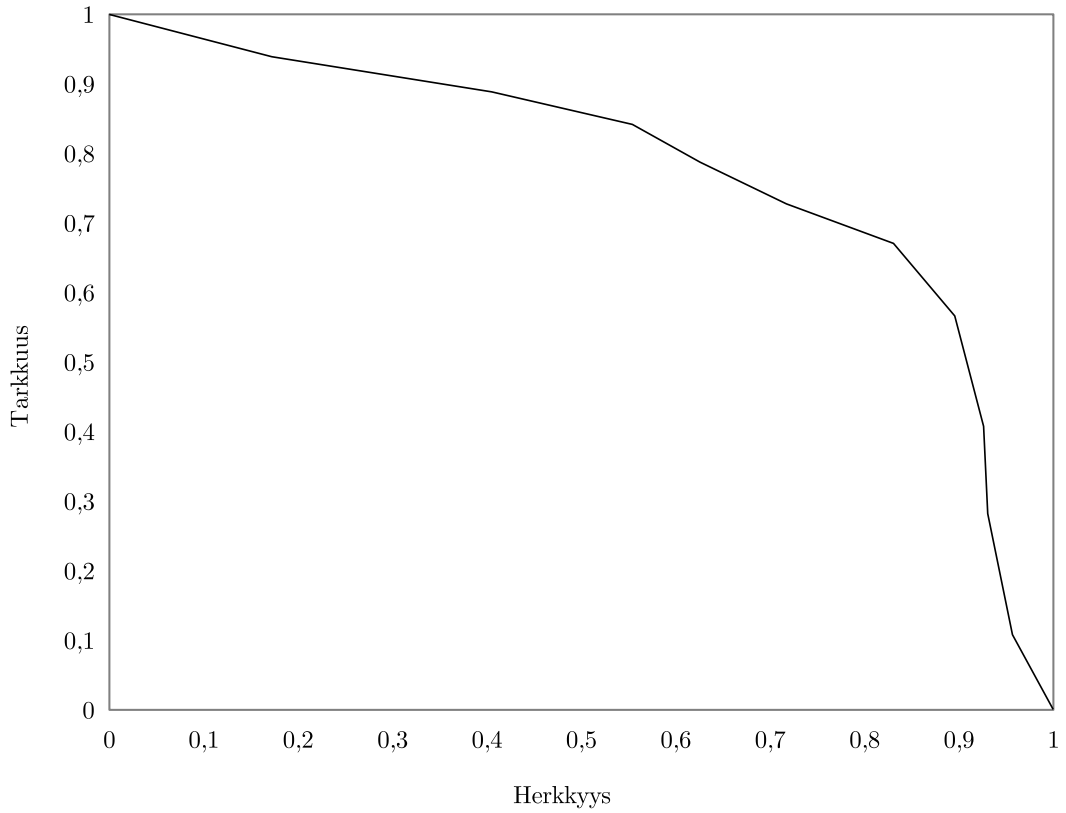
Mikäli käytössä on ollut sama testimateriaali, eri tunnistimien toimintaa voidaan vertailla halutulla kuvausmenetelmällä tutkimalla käyrien keksinäistä sijaintia tasossa. Paremman tunnistimen käyrä kulkee lähempänä kyseisen tason tavoitepistettä (joka vastaa virheetöntä tunnistusta). ROC-käyrässä se on piste (0,1) (”vasen yläkulma”), DET-käyrässä piste (0,0) (”vasen alakulma”) ja PR-käyrässä piste (1,1) (”oikea yläkulma”). Valitusta kuvaustavasta riippumatta tunnistimien keskinäinen vertailu tuottaa samat tulokset; eroa syntyy arvioitaessa kuinka paljon yksittäinen tunnistin



Kuva 3.3. *Esimerkki DET-käyrästä. Tunnistin on sitä parempi mitä lähempänä vasenta alakulmaa sen DET-käyrä kulkee. Akselit ovat logaritmisia.*

poikkeaa täydellisestä tunnistuksesta. Vinoutuneessa luokkajakaumassa ROC-käyrä voi osoittaa tunnistuksen jo olevan erittäin lähellä täydellistä, vaikka PR-käyrä kulkee vielä kaukana tavoitepisteestään. [14] Avainsanojen tunnistuksessa materiaalin luokkajakauma on usein vino, sisältäen huomattavasti enemmän ei-avainsanoja kuin varsinaisia avainsanoja. Tunnistimien vertailu käyrien avulla sisältää myös selkeän ongelman, kun käyrät leikkaavat; tällöin tunnistimien paremmuudesta voi puhua vain sovelluskohtaisesti.

Käyrien sijasta onkin usein mielekkäämpää saada yksittäinen tunnistimen tarkkuutta kuvaava tunnusluku. Yhtenä vaihtoehtona on laskea käyrän rajaaman alueen pinta-ala, tunnetuimpana tapauksena ROC-käyrän *AUC* (*Area Under Curve*), joka on merkitty kuvaan 3.2. AUC vastaa todennäköisyyttä, että tunnistin tunnistaa satunnaisesti valitun avainsanan oikein, kun päätöspinta asetetaan satunnaisesti jakamaan puheessa olevat sanat. [13] AUC:n laskenta vaatii käytännössä ROC-käyrän evaluointia halutulla tarkkuudella.



Kuva 3.4. *Esimerkki PR-käyrästä. Tunnistin on sitä parempi, mitä lähempänä oikeaa yläkulmaa sen PR-käyrä kulkee.*

Yksinkertaisempi tunnusluku on *EER* (*Equal Error Rate*), joka esiintyy kuvissa 3.2 ja 3.3. EER vastaa ROC/DET-käyrän pistettä, jossa virhetyypit ovat yhtä suuret, formaalisti $\mathrm{FPR} = (1 - S)$.

Kuvista poiketen käyristä ei useinkaan muodostu jatkuvia, vaan ne koostuvat erillisistä tasoista. Käyrät eivät myöskään välttämättä kulje kuvatunlaisesti pisteestä (0,1) pisteeseen (1,0).

3.2 Laskennallinen tehokkuus

Avainsanojen tunnistimen käyttökelpoisuus eri tapauksissa on pitkälti riippuvainen tunnistusprosessin raskaudesta. Puheohjatun reaaliaikajärjestelmän tapauksessa voidaan esimerkiksi hyväksyä heikompikin tunnistustarkkuus (tämä on tietysti sovelluskohtaista), mutta tunnistuksen ajankäytön kasvaminen yli reaaliaikavaatimusten on ongelmallisempaa. Tunnistusprosessin tehokkuutta mitataan reaaliaikakertoimella (*Real Time Factor, RTF*), joka on tunnistuksessa kuluneen ajan $t_{\text{tunnistus}}$

suhde äänisignaalin kestoon t_{audio} (aikayksiköiden oltava samat muuttujissa):

$$\text{RTF} = \frac{t_{\text{tunnistus}}}{t_{\text{audio}}} \tag{3.1}$$

Reaaliaikakerroin yksi tarkoittaa tunnistuksen olevan tarkasti reaaliaikaista; reaaliaikakertoimen ollessa suurempi kuin yksi tunnistus ei onnistu reaaliajassa.

Laskennallista tehokkuutta on myös mahdollista mitata normalisoimalla reaaliaikakerroin tunnistettavien avainsanojen määrällä [4, s. 22]. Näin saatua tunnuslukua voidaan käyttää reaaliaikakertoimen laskentaan mielivaltaiselle määrälle avainsanoja, mikäli ajankäytön oletetaan käyttäytyvän lineaarisesti suhteessa avainsanojen määrään.

Tehokkuusmitoille saatavat arvot ovat luonnollisesti riippuvaisia käytetystä testausympäristöstä, joten niiden tarkastelussa täytyy aina huomioida käytössä olleet laskentaresurssit. Käytännössä tämä tarkoittaa suorittimien tehokkuutta ja lukumäärää sekä mahdollisesti fyysisen muistin määrää. Lisäksi muut ajossa olevat prosessit vaikuttavat siihen kuinka paljon laskentaresursseja on käytössä. Koska muiden prosessien vaikutusta on vaikea arvioida, on laskennallista tehokkuutta mitatessa syytä minimoida muiden ajossa olevien prosessien määrä.

3.3 Työssä käytettävät mittarit

Tässä työssä käytetään tunnistustarkkuuden ensisijaisena mittana ROC-käyrästä laskettavaa EER-tunnuslukua. Tilanteet, joissa äänisignaalissa oleva sana havaitaan jonain toisena sanana tulkitaan sekä tyypin 1 että 2 virheeksi. Koska EER kuvaa tunnistimen toimintaa kuitenkin vain yhdellä säätöparametrin arvolla, voi se antaa harhaanjohtavan kuvan tunnistimen toimintatarkkuudesta [4, s. 23]. Tätä pyritään kompensoimaan laskemalla myös EER-pisteen kahdelta puolelta käyrän pisteet virhesuhteiden dynamiikan arvioimiseksi. Näiksi pisteiksi on tässä työssä valittu kohdat joissa toisen virhetyypin osuus on kaksinkertainen verrattuna toiseen. Jatkossa näitä kohtia kutsutaan lyhyesti 1:2- ja 2:1-virheiksi. Syitä ROC-käyrästä laskettavien pisteiden käyttöön on kaksi:

1. Käytettävän tason pisteiden tulkinta on intuitiivista.
2. Mikäli virheet kasvavat käytettävän säätöparametrin suhteen monotonisesti, voidaan käyrän tietty piste etsiä N-pisteen ROC-käyrästä muokatulla puolitushaulla (katso kohta 6.4.1).

Lisäksi tehokkuusmittarina käytetään reaaliaikakerrointa (3.1). Syy tämän valintaan on RTF-arvosta suoraan nähtävä soveltuvuus reaaliaikaprosessointiin.

Alla olevassa listassa on koottuna työssä käytettävät mitat ja niiden määritelmät. Esityksen selkeyttämiseksi määritellään *tunnistamatta jääneiden osuus* (engl. False Negative Rate, FNR), joka on jo DET-käyrässä nähty herkkyyden komplementti: $\mathrm{FNR} = (1 - S)$.

- EER, ROC-käyrän piste jossa $\mathrm{FPR} = \mathrm{FNR}$
- 1:2-virhe, ROC-käyrän piste jossa $2 \cdot \mathrm{FPR} = \mathrm{FNR}$
- 2:1-virhe, ROC-käyrän piste jossa $\mathrm{FPR} = 2 \cdot \mathrm{FNR}$
- Reaaliaikakerroin, $\mathrm{RTF} = \frac{t_{\text{tunnistus}}}{t_{\text{audio}}}$

Mittareiden tarkkuutta voidaan parantaa laskemalla usean mittauksen keskiarvoja. Tämä voi tulla kyseeseen, mikäli esimerkiksi testimateriaalia poimitaan satunnaisesti.

4. TESTAUSJÄRJESTELMÄSSÄ KÄYTETTÄVIÄ OHJELMISTOTEKNIIKOITA

Koska työn varsinaisena sisältönä oli mittausjärjestelmän toteuttaminen, sen toteutuksessa käytettyjä ohjelmistoteknisiä perustuksia esitellään tässä luvussa. Järjestelmän toteutus käyttää olioita, johtuen niiden tarjoamista tehokkaista mekanismeista jotka mahdollistavat sekä tunnistimen että suureiden vaihtamisen helposti tarpeen mukaan (jos esimerkiksi jokin luvussa 3 esitellyista suureista todetaan huonoksi ratkaisuksi ja halutaan korvata jollain toisella). Lisäksi keskenään riippumattomien testitapausten suorittamiseen kuluva kokonaisaika pienenee käyttämällä rinnakaisprosessointia.

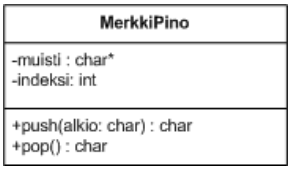
4.1 Olio-ohjelmointi

Olio-ohjelmointi on nykyään käytännössä *de facto* -tapa toteuttaa ohjelmistoja; monet nykyaikaiset ohjelmointikielet jopa pakottavat käyttämään olioparadigmaa (Java, C#). Olio-ohjelmoinnin peruskäsitteitä ovat luokka ja olio, joista jälkimmäinen voidaan ymmärtää edellisen realisaationa [6, s. 47].

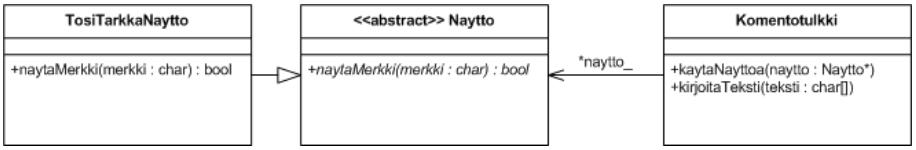
”Tavallinen” luokka sisältää aina vähintää *rakentajan*, jota kutsutaan kun luodaan kyseisen luokan olio ja *purkajan*, jota kutsutaan kun olio tuhotaan. Rakentajasta on mahdollista tehdä parametrillisia versioita, mikäli oliota luotaessa on tarpeellista saada jotain tietoja siihen liittyen. Hiukan erikoisempi ratkaisu olioiden luontia varten on käyttää *nimettyä rakentajaa*, jolloin kaikki rakentajat siirretään luokan sisäiseen toteutukseen, ja julkisen rajapinnan kautta voidaan kutsua vain *staattista* operaatiota (operaatio on olemassa riippumatta olioista) joka palauttaa halutunlaisen olion. Tällä menetelmällä saadaan suurempi kontrolli kyseisen luokan olioiden luontiin ohjelmassa.

4.1.1 Abstrahointi

Abstrahointi tarkoittaa jonkin käsitteen yksinkertaistamista karsimalla sen yksityiskohtia. Olioiden tapauksessa se tarkoittaa yksinkertaisimmillaan sitä, ettei olion käyttäjän tarvitse tietää olion sisäistä toteutusta, joka on *kapseloitu* olion sisään



Kuva 4.1. *Esimerkki sisäisen toteutuksen kapseloinnista.* MerkkiPino-*luokan käyttäjän ei tarvitse tietää mitään luokan sisäisestä muistinhallinnasta.*



Kuva 4.2. *Esimerkki periytymisestä.* Komentotulkki-*luokka ei tiedä mitä* Naytto-*luokan perillistä se todellisuudessa käyttää.*

(kuva 4.1). [6, s. 25–28.] Hiukan erityyppistä abstrahointia edustaa *periytymismekanismin* käyttö, jolloin oliota voidaan käyttää jopa tietämättä sen varsinaista luokkaa [6, s. 119] (kuva 4.2).

Mikäli luokalle, josta periytetään, ei ole järkevää implementoida kaikkia operaatioita, ne määritellään *puhtaasti virtuaalisiksi* eikä niille kirjoiteta toteutusta, jolloin luokkaa kutsutaan *abstraktiksi kantaluokaksi*. Luokasta ei luonnollisestikaan voida suoraan luoda oliota sen vajaan toteutuksen takia. Ääritapauksena tästä on rajapintaluokka, joka sisältää vain ja ainoastaan luokan julkisen rajapinnan operaatiot puhtaasti virtuaalisina. Tätä voidaan käyttää tapauksissa joissa kyseisen rajapintaluokan toteutuksista ei ole järkevää tehdä mitään etukäteisoletuksia. Lisäksi, koska rajapintaluokka ei sisällä mitään konkreettista toteutusta tai tietosisältöä, sitä voidaan käyttää *moniperiyttämisessä*, joka tarkoittaa luokan periyttämistä useammasta kantaluokasta.

4.1.2 Ohjelmakoodin uudelleenkäyttö

Käyttämällä edellisessä luvussa esiteltyä periytymistä voidaan ohjelma kirjoittaa käyttämään hyvin määriteltyjen kantaluokkien osoittimia luokkien välisessä toiminnassa. Näin kirjoitettuna ohjelmakoodi on helposti uudelleenkäytettävää; uuden kantaluokasta periytetyn luokan käyttäminen ei vaadi muiden luokkien toteutuksen muuttamista.

4.1.3 Poikkeukset

Ainakin C++- ja Java-kielissä esiintyy olioita sivuava virheenkäsittelymekanismi, *poikkeusten* käyttäminen. Käytännössä nämä tulevat väistämättä vastaan, mikäli edellä mainittujen kielien standardikirjastoja käytetään. Virheen sattuessa poik-

keusolio *heitetään*, jonka jälkeen ohjelma palaa funktioiden kutsuhierarkiassa takaisinpäin kunnes löytää poikkeuskäsittelijän, joka on halukas *sieppaamaan* heitetyn tyyppisen olion [6, s. 303].

Olioiden turvallista käyttämistä varten kaikki niistä mahdollisesti lentävät poikkeukset tulisi periaatteessa dokumentoida; käytännössä tämä on kuitenkin mahdotonta. Sen sijaan luokkien operaatioille voidaan antaa takuut miten ne vaikuttavat olion tilaan poikkeuksen sattuessa. Mahdollisia takuita on neljä, joista kolme ensimmäistä ovat keskenään vaihtoehtoisia ja neljäs eräänlainen erikoistakuu [6, s. 315–318]. Mahdolliset takuut ovat seuraavat:

- Perustakuu, joka takaa että poikkeuksen lentäessä resursseja ei hukata ja olio on tuhottavissa.
- Vahva takuu, joka takaa että poikkeuksen lentäessä olion tila pysyy alkuperäisenä.
- Nothrow-takuu, joka takaa ettei operaatiossa voi tapahtua mitään virheitä.
- Poikkeusneutraalius, joka takaa että olion sisäisten komponenttien poikkeukset vuodetaan muuttumattomina ulos.

Tämän työn kannalta oleellisimmat ovat ensimmäinen ja viimeinen takuu.

4.2 Rinnakkaisuus

Tietokoneen kyky tehdä samanaikaisesti useita tehtäviä voi olla joko todellista, kun käytössä on useita suoritinytimiä, tai virtuaalista, jolloin erilliset tehtävät käyttävät yksittäistä ydintä pienissä aikaviipaleissa [11, s. 88]. Virtuaalisen rinnakkaisuuden tapauksessa voidaan helposti nähdä, ettei tehtävän, joka vaatii määrän X toisistaan riippumattomia laskentaintensiivisiä vaiheita, toteutusta voida nopeuttaa ajamalla vaiheita rinnakkain. Mikäli vaiheet kuitenkin sisältäisivät esimerkiksi jonkin oheislaitteen odottamisesta aiheutuvaa tyhjäkäyntiä, voisi näinkin toteutettu rinnakkaisuus nopeuttaa tehtävää. Tässä työssä rinnakkaisuutta käytetään nopeuttamaan laskentaa ajamalla eri testitapauksia rinnakkain; testiympäristönä oletetaan olevan moniydinjärjestelmä.

Rinnakkaisuuteen liittyvä käsite *säie* tarkoittaa kevytprosessia, joka luodaan isäntäprosessin (ohjelman) sisällä toteuttamaan jotain tiettyä tehtävää. Saman isäntäprosessin säikeet jakavat saman osoiteavaruuden, jonka ansiosta säikeiden käsittely on täysimittaisia prosesseja kevyempää. [11, s. 52.]

Rinnakkaisuuden toteutuksessa kohdataan tavallisesti *poissulkemis-* ja *synkronointiongelmia*. Poissulkemisongelma tarkoittaa tilannetta, jossa säikeet pyrkivät käsittelemään samaa tietorakennetta samanaikaisesti. Mikäli säikeet vaikuttavat tietorakenteen sisältöön, voi niiden käsitys esimerkiksi muuttujien arvoista päätyä keskenään ristiriitaiseksi. Tämänkaltaisten tilanteiden välttämiseksi säikeille yhteisiä tietorakenteita muokkaavat osat ohjelmakoodista merkitään *kriittisiksi alueiksi*, joiden suoritus sallitaan vain yhdelle säikeelle kerrallaan. [11, s. 90-95.] Kriittisten alueiden suojaaminen tehdään usein käyttäen *mutex*-algoritmeja (engl. *mutual exclusion*), jotka toteuttavat tämän rajoituksen.

Synkronointiongelma käsittelee säikeiden keskinäistä tahdistamista [11, s. 89]. Tälläinen tilanne voi esiintyä esimerkiksi silloin kun kaikkien säikeiden vaaditaan suorittavan tietty tehtävä ennen kuin mikään säie voi jatkaa suoritustaan. Näissä tilanteissa voidaan käyttää *semafori*-rakenteita, joita käytetään myös poissulkemisongelman kriittisten alueiden toteutuksessa [11, s. 95].

5. TESTIMATERIAALI

Testimateriaalina käytettiin yhden kielen puheäänitteiden kokoelmaa (*korpusta*), joka sisältää yli 9900 uniikkia sanaa, jotka esiintyvät yhteensä yli 120000 kertaa. Äänitteissä esiintyy 2500 eri puhujaa, kukin tavallisimmin 8-9 äänitteessä. Äänitteet ovat tallennettu lankapuhelinverkosta ja ovat kestoltaan 1-20 sekuntia. Niiden sisältönä on eri lähteistä luettua tekstiä (esimerkiksi osia sanomalehtiartikkeleista). Kukin äänite sijaitsee omassa tiedostossaan, ja lisäksi kuhunkin äänitteeseen liittyy metadatatiedosto.

Metadatatiedosto sisältää tiedon äänitteen *signaali-kohinasuhteesta* (engl. *Signal-to-Noise Ratio, SNR*, kaava (5.3)) sekä subjektiivisen arvion sen laadusta. Jälkimmäinen arvio on ihmisen tekemä luokitus yhteen neljästä seuraavasta joukosta:

- puhe on ymmärrettävää,
- puhe ei ole kaikilta osin ymmärrettävää,
- äänite sisältää voimakasta kohinaa tai
- äänite sisältää pelkkää kohinaa.

Lisäksi tiedostossa on äänitteessä esiintyvästä puhujasta listattuna sukupuoli, ikä, ja *aksentti* (aksentilla neljä vaihtoehtoa, joihin viitataan luvuilla 1-4). Tässä yhteydessä aksentilla tarkoitetaan saman kielen puhujien keskuudessa esiintyviä alueellisia variaatioita ääntötavoissa.

Tässä luvussa esiteltävät ratkaisut materiaalin valinnassa ja muokkauksessa eivät nojaudu mihinkään yleisempään teoriaan, vaan ovat pääosin seurausta tämän diplomityön asiakkaan tarpeista.

5.1 Akustinen data

Käytetyn korpuksen äänitteet ovat *A-laki-kompandoituja*. Kompandointi on yhdistelmä sanoista kompressio-ekspandointi. Sen tarkoituksena on kuvata kvantisoitaessa pienet amplitudit suuremmalla tarkkuudella kuin suuret. Tähän löytyy kaksi syytä puhesignaalin tapauksessa: Ihmiskorva on vähemmän tarkka kohinalle (tai yleisemmin muutoksille signaalin arvoissa) signaalin tason ollessa suuri, ja toisaalta

puhesignaali sisältää pääasiassa pieniä amplitudeja. Kompressio toteutetaan logaritmifunktiolla ennen kvantisointia, A-laki-kompandoinnin tapauksessa kaavan (5.1) mukaisesti. [2, s. 4-5.]

$$F(x) = \begin{cases} \frac{Ax}{1+\log_{10} A} & , \quad \text{kun } |x| < \frac{1}{A} \\ \text{sign}(x)\frac{1+\log_{10}(A|x|)}{1+\log_{10} A} & , \quad \text{kun } \frac{1}{A} \leq |x| \leq 1 \end{cases} \tag{5.1}$$

Kompression jälkeen signaali kvantisoidaan halutulla tarkkuudella. Käänteinen operaatio, ekspandointi palauttaa signaalin alkuperäiseen muotoon:

$$F^{-1}(y) = \begin{cases} \frac{y(1+\log_{10} A)}{A} & , \quad \text{kun } |y| < \frac{1}{1+\log_{10} A} \\ \text{sign}(y)\frac{\exp(|y|(1+\log_{10} A)-1)}{A} & , \quad \text{kun } \frac{1}{1+\log_{10} A} \leq |y| \leq 1 \end{cases} \tag{5.2}$$

Vakiolle A käytetään arvoa $87,56$. A-laki-kompandointi on käytössä Euroopan lankapuhelinverkossa. Pohjois-Amerikassa on käytössä tästä hiukan poikkeava μ-laki-kompandointi, joka perustuu myös logaritmiseen kompressointiin.

Korpuksessa olevat äänisignaalit ovat kompressoituja ja 8 bitillä kvantisoituja (vastaa noin 12 bitin käyttöä tasavälisesti kvantisoitaessa [3, s. 6]). Näytteenottotaajuutena on käytetty 8000 hertsiä, ja signaalit ovat kaistarajoitettu välille 300 - 3400 Hz (johtuen lankapuhelinverkosta).

5.1.1 Esivalinta

Testimateriaalista on syytä karsia pois korpuksessa olevat huonolaatuisimmat äänitteet, jotta materiaali olisi mahdollisimman tasalaatuista. Tämä pienentää mittaustulosten varianssia valittaessa äänitteitä ajonaikaisesti. Tässä työssä materiaalista valitaan vain ne jotka on merkitty kuuluvaksi joukkoon ”puhe on ymmärrettävää” (kuva 5.1).

Kuva 5.1. *Akustisen materiaalin esivalinta.*

5.1.2 Ajonaikainen valinta ja muokkaus

Ajonaikaisesti pyritään tässä työssä hallitsemaan äänitteissä esiintyvien puhujien aksenttijakaumaa ja äänitteiden signaali-kohinasuhdetta. Näiden rajausten toteutus tapahtuu yksinkertaisesti vertaamalla äänitteeseen liittyviä tietoja haluttuihin rajoihin. Aksenttien vaikutus tunnistustarkkuuteen kertoo tunnistimen kyvystä toimia kattavasti tutkittavana olevan kielen eri puhealueilla; signaali-kohinasuhteen vaikutus tunnistustarkkuuteen kertoo tunnistuksen käyttökelpoisuudessa vaihtelevissa olosuhteissa.

Signaali-kohinasuhdetta voidaan myös varioida rajaamisen lisäksi muokkaamalla signaaleja. Jotta korpuksen akustista dataa voidaan muokata hallitusti, täytyy se ensin ekspandoida kaavan (5.2) mukaisesti. Tämän jälkeen signaalin laatua voidaan huonontaa summaamalla siihen kohinaa. Tässä työssä käytettyä kohinan tyyppiä ja sen lisäämiseen liittyviä käytännön seikkoja käsitellään kohdassa 6.4.2. Signaali-kohinasuhteen uudelleenlaskentaan käytettävä menetelmä johdetaan kuitenkin alla.

Signaali-kohinasuhde määritellään desibeleissä

$$SNR = 10 \log_{10} \frac{P_S}{P_\varepsilon} \ , \tag{5.3}$$

jossa P_s on puhtaan signaalin teho ja P_ε kohinan teho. Tehon mittana käytetään signaalin neliöityjen näytteiden keskiarvoa.

Yhtälön (5.3) evaluointi vaikeutuu huomattavasti mikäli puhtaan signaalin tehoa ei ole tiedossa. Käytännössä tässä tapauksessa näin onkin, koska käytössä oleva alkuperäinen signaali on äänite, jossa jo itsessään on kohinaa. Kohinat oletetaan additiivisiksi.

Mikäli jo alkuperäiseen signaaliin sisältyvän kohinan ε_1 tehoa merkitään symbolilla P_{ε_1}, signaaliin lisättävän kohinan ε_2 tehoa symbolilla P_{ε_2} ja kohinat oletetaan toisistaan riippumattomaksi ja ainakin toisen odotusarvoksi nolla, voidaan kohinoiden kokonaisteholle $P_\varepsilon = P_{\varepsilon_1+\varepsilon_2}$ käyttää harhattomana estimaattorina niiden tehojen summaa ($\mathbb{E}[x]$ on x:n odotusarvo):

$$\begin{aligned}
\mathbb{E}[P_{\varepsilon_1+\varepsilon_2}] &= \mathbb{E}[\frac{1}{N}\sum_{n=0}^{N-1}(\varepsilon_1[n]+\varepsilon_2[n])^2] \\
&= \mathbb{E}[\frac{1}{N}\sum_{n=0}^{N-1}\varepsilon_1^2[n] + \frac{1}{N}\sum_{n=0}^{N-1}\varepsilon_2^2[n]] + \mathbb{E}[\frac{2}{N}\sum_{n=0}^{N-1}\varepsilon_1[n]\varepsilon_2[n]] \\
&= \mathbb{E}[P_{\varepsilon_1} + P_{\varepsilon_2}] + 0
\end{aligned}$$

Käyttäen tätä ja kaavaa (5.3), saadaan

$$\begin{aligned} SNR_{\text{uusi}} &\approx 10\log_{10}\frac{P_S}{P_{\varepsilon_1}+P_{\varepsilon_2}} = 10\log_{10}\left(\frac{P_{\varepsilon_1}+P_{\varepsilon_2}}{P_S}\right)^{-1} \\ &= -10\log_{10}\left(\frac{P_{\varepsilon_1}}{P_S}+\frac{P_{\varepsilon_2}}{P_S}\right) \end{aligned} \tag{5.4}$$

Mikäli SNR-arvon muuntelu tehdään vain alkuperäisille signaaleille joilla $P_S \gg P_{\varepsilon_1}$ (esimerkiksi jos signaalin SNR-arvo on 30 dB, on puhtaan signaalin teho tuhatkertainen suhteessa kohinan tehoon), voidaan tehdä approksimaatio $P_S \approx P_{S+\varepsilon_1}$ ja koska

$$SNR_{\text{orig}} = 10\log_{10}\frac{P_S}{P_{\varepsilon_1}} ,$$

saadaan (5.4) muotoon

$$SNR_{\text{uusi}} \approx -10\log_{10}\left(\frac{1}{10^{\frac{SNR_{\text{orig}}}{10}}}+\frac{P_{\varepsilon_2}}{P_{S+\varepsilon_1}}\right) , \tag{5.5}$$

joka voidaan laskea, kun tiedetään äänitteen SNR-arvo SNR_{orig}, sen teho $P_{S+\varepsilon_1}$ sekä lisättävän kohinasignaalin teho P_{ε_2}. Näistä kaksi jälkimmäistä saadaan laskettua summattavista signaaleista.

5.2 Avainsanat

Tunnistusprosessin varsinaisina kohteina toimivat avainsanat. Materiaalissa esiintyvistä 9900 sanasta luodaan tunnistamista varten pienempiä osajoukkoja.

5.2.1 Esivalinta

Sanajoukko ei ole täysin homogeeninen esimerkiksi esiintymisfrekvenssin suhteen, ja tätä pyritään kompensoimaan jättämällä kaikki vain yhden kerran ja yli sata kertaa materiaalissa esiintyvät sanat pois. Myös kaikkein lyhyimmät sanat (1-3 grafeemia) jätetään pois. Nämä ovat tavallisesti partikkelisanoja, jotka eivät tunnistimen hypoteettisessa loppukäytössä ole merkityksellisiä (yleisesti; avainsanojen tunnistimen käyttökohteeksi ei oleteta apusanojen paikallistamista), vaan antavat todellisuutta huonomman kuvan tunnistimen tunnistustarkkuudesta [4, s. 43]. Valinta on esitetty kuvassa 5.2. Esivalinnan läpäisivät noin 9000 sanaa.

5.2.2 Ajonaikainen valinta

Avainsanat valitaan testimateriaalista löytyvistä sanoista satunnaisotannalla. Tunnistettavien avainsanojen määrä vaikuttaa merkittävästi tunnistustarkkuuteen. Mi-

Kuva 5.2. *Avainsanojen esivalinta.*

tä useampi vaihtoehto tunnistustulokselle on, sitä todennäköisemmin löydetty sana myös tunnistetaan väärin.

5.3 Antiavainsanat

Tyypin 2 virhettä on mahdollista pienentää käyttäen antiavainsanoja, eli sanoja joilla on suuri riski sekoittua kiinnostuksen kohteina oleviin avainsanoihin. Näiden generointi täytyy kuitenkin jollain tavalla automatisoida, koska niiden käsin valitseminen kattavasta testimateriaalista on käytännössä mahdotonta käsiteltäessä hiukankaan suurempia sanamääriä. Järkevälle antiavainsanojen joukolle voidaan asettaa seuraavat sanakohtaiset reunaehdot:

1. Sanan pitää esiintyä äänitteissä.
2. Sanan pitää muistuttaa jotain avainsanaa.
3. Sana ei saa esiintyä avainsanoissa.

Kaksi ensimmäistä kohtaa voidaan toteuttaa etukäteen, viimeinen testiajon aikaisesti, avainsanojen valinnan jälkeen.

5.3.1 Esivalinta

Kuvan 5.1 mukaisesti rajatun äänitallennejoukon sisältämät kaikki sanat ovat lähtökohtaisesti potentiaalisia antiavainsanoja. Niiden soveltuvuus antiavainsanaksi on kuitenkin riippuvainen niiden samankaltaisuudesta johonkin avainsanaan (joiden rajaus kuvassa 5.2).

”Samankaltaisuus” on melko epämääräinen termi, mutta tässä yhteydessä on sille kuitenkin löydettävissä merkityksellinen tunnusluku. *Levenšteinin* etäisyys (joka

Taulukko 5.1. *Esimerkki Levenšteinin etäisyyden laskennasta, kun sana "kassa" muunnetaan sanaksi "kisa".*

		k	i	s	a
	0	1	2	3	4
k	1	0	1	2	3
a	2	1	1	2	2
s	3	2	2	1	2
s	4	3	3	2	2
a	5	4	4	3	**2**

kuuluu yleisempään *edit distance*-mittojen joukkoon) on kahden symbolijonon poikkeavuuden mitta [4, s. 217]. Etäisyys on pienin vaadittujen operaatioiden määrä jolla jono $p_{1..N}$ saadaan muunnettua jonoksi $q_{1..M}$. Sallitut operaatiot ovat sijoitus, poisto ja korvaus. Operaatioiden kustannukset on tämän työn puitteissa sidottu arvoon yksi, paitsi symbolin korvaus on ilmaista jos korvattava symboli on sama kuin korvaava symboli. Korvausoperaation kustannuksen laskentaan varten määritellään

$$korvaa(a,b) = \begin{cases} 0, \text{ kun } a = b \\ 1, \text{ muulloin} \end{cases}$$

Etäisyys voidaan laskea luomalla $(M+1) \times (N+1)$ kokoinen taulukko Ω ja täyttämällä se:

1. Aseta taulukon ensimmäiseksi riviksi luvut $0..N$.
2. Aseta taulukon ensimmäiseksi sarakkeeksi luvut $0..M$.
3. Täytetään taulukon tyhjät solut alkaen vasemmasta yläkulmasta

$$\forall i \neq 0, j \neq 0 : \Omega_{i,j} = \min \begin{cases} \Omega_{i,j-1} & +1 \\ \Omega_{i-1,j} & +1 \\ \Omega_{i-1,j-1} & + korvaa(p_i, q_j) \end{cases}$$

Tämän jälkeen Levenšteinin etäisyys voidaan lukea taulukon oikeasta alakulmasta. [4, s. 218–220] Esimerkki etäisyyden laskennasta on taulukossa 5.1. Käyttäen tätä menetelmää voidaan kullekin avainsanalle etsiä lähimmät vastaavuudet muista sanoista, käyttäen kullekin avainsanalle vuorollaan kaikkia sanoja kohdesanana, ja valitsemalla kullekin avainsanalle joukon lähimpiä sanoja Levenšteinin etäisyyden perusteella.

5.3.2 Ajonaikainen valinta

Ajonaikaisesti tehdään päätös kuinka monta antiavainsanaa pyritään käyttämään kutakin avainsanaa kohti. Koska antiavainsanaksi ei ole aiheellista valita jo avainsanajoukossa olevaa sanaa, täytyy avainsanaa kohti olla useampi antiavainsanaehdokas listattuna, jotta vaihtoehdoista voidaan varmemmin valita antiavainsana.

Osassa äänitiedostoja saattaa tulla vastaan tilanne, jossa valittu antiavainsana sisältyy samaan äänitiedostoon kuin siihen liittyvä varsinainen avainsana. Tässä tapauksessa antiavainsanan käyttämisestä ei ole edes teoriassa hyötyä. Antiavainsanan käyttö voi tällöin jopa näennäisesti heikentää tunnistustarkkuutta, jos aiemmin tehty havainto avainsanasta onkin kohdistunut virheellisesti vain kyseiseen antiavainsanaan. Tämäntyyppinen virheellinen tulkinta kuitenkin hyväksytään tässä työssä: näiden tilanteiden voidaan arvioida olevan melko harvalukuisia, ja koska materiaalissa on sanoja tuhansittain, ei näiden virheiden oleteta vääristävän tulosta merkittävästi. Toisaalta tälläisia tilanteita (eli tilanteita jossa avainsana ja sen antiavainsana ovat toistensa välittömässä läheisyydessä) tulee vastaan myös avainsanojen tunnistimien ”oikeassa” käytössä, joten niiden poistaminen testaustilanteesta ei välttämättä olisi täysin perusteltuakaan.

5.4 Testimateriaalin sisällön hallinta

Valitut äänitiedostot ja osa metatiedoista (kuva 5.1) listataan tiedostoon `info.txt`, ja niiden sisältämät esivalitut sanat (kuva 5.2), listataan tiedostoon `wordlist.txt`. Lisäksi jälkimmäiseen lasketaan kullekin sanalle kymmenen antiavainsanaa kohdan 5.3.1. `info.txt`-tiedosto sisältää rivin kutakin äänitiedostoa kohti muodossa

<tiedosto> : <pituus> <sanojen lkm> <SNR> <aksentti>

jossa

- <tiedosto> on tiedoston nimi
- <pituus> on tiedoston sisältämän signaalin näytteiden määrä
- <sanojen lkm> on signaalin sisältämien sanojen lukumäärä
- <SNR> on signaalin signaali-kohinasuhde desibeleissä
- <aksentti> on luku väliltä 1-4

`wordlist.txt`-tiedosto sisältää rivin kutakin aineistossa esiintyvää sanaa kohti muodossa

<sana> !<antisana1> ... !<antisana10> : <tiedosto1> <tiedosto2> ...

jossa

- <sana> on sanan grafeemiesitys
- <antisanaX> on sanalle etsitty antiavainsana
- <tiedostoX> on tiedosto joka sisältää sanan <sana>

6. TOTEUTETTU TESTAUSJÄRJESTELMÄ

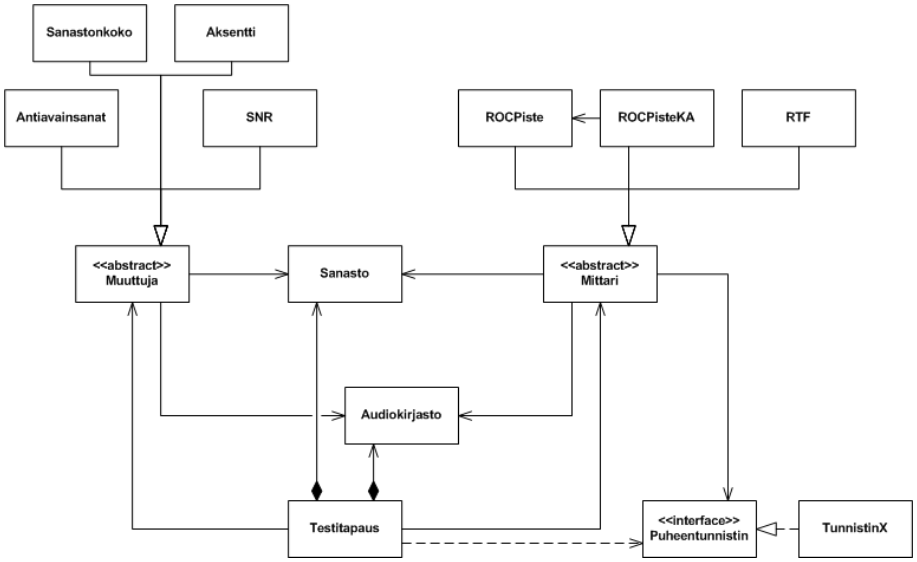
Työssä toteutettu järjestelmä suunniteltiin riippumattomaksi testattavasta avainsanojen tunnistimesta ja mahdollistamaan yksinkertainen mitattavien ja kontrolloitavien suureiden muuttaminen kajoamatta muuhun toteutukseen. Tunnistimen avainsanojen syötön oletettiin tapahtuvan grafeemimuodossa. Lisäksi järjestelyn täytyi taipua rinnakkaiseen prosessointiin soveltuvilta osin voidakseen käsitellä tehokkaasti suurta datamassaa. Järjestelmä toteutettiin C++-kielellä.

Riippumattomuus tunnistimesta sekä suureista saavutettiin käyttämällä olio-ohjelmoinnin periytysmekaniikkaa ja abstrakteja kantaluokkia, rinnakkaisprosessointi käyttäen OpenMP (*Open Multi-Processing*) -kirjaston säikeitä ja tunnistamalla ohjelman kriittiset osiot. Kuvassa 6.1 on kuvattuna järjestelmän luokkarakenne. Toteutuksen keskeisenä komponenttina on *Testitapaus*-luokka. Se toteuttaa yksittäisen testitapauksen, eli mittauksen, jossa varioidaan ajonaikaisesti testimateriaalia (kohdat 5.1.2, 5.2.2 ja 5.3.2) luokan *Muuttuja* avulla ja käytetään yhtä aliluvun 3.3 mittaria luokan *Mittari* avulla. Luotavat testitapaukset ovat liitteen taulukossa A.1. Järjestelmä käyttää testimateriaalia luokkien *Sanasto* ja Audiokirjasto kautta, jotka edelleen käyttävät kohdassa 5.4 määriteltyjä tiedostoja. Järjestelmän osia tarkennetaan seuraavissa kohdissa (pl. *TunnistinX*, joka toimii paremman kokonaiskäsityksen antamiseksi, ja vain paikanpitäjänä todelliselle avainsanojen tunnistimen integraatiolle).

Järjestelmään kuuluu myös apuohjelmia jotka toteuttavat luvun 5 esivalinnat. Näitä apuohjelmia ei kuitenkaan erityisemmin käsitellä tässä niiden suhteellisen triviaalin luonteen vuoksi.

6.1 Testitapaukset

Testitapaus on järjestelmän keskeisin luokka. Se omistaa muista luokista *Sanasto*- ja *Audiokirjasto*-luokan oliot, eli se vastaa näiden olioiden elinkaaresta. Rinnakkaisuuden vuoksi kuhunkin testitapaukseen pitää voida liittää uniikki tunnistenumero, jota se käyttää *Audiokirjasto*-oliolle annettavan työhakemiston nimeämisessä. Tämä tehdään nimetty rakentaja-idiomilla. Nimettynä rakentajana toimiva staattinen jäsenfunktio sisältää paikallisen staattisen muuttujan joka pitää kirjaa luotujen olioiden määrästä, ja koko funktio merkitään kriittiseksi alueeksi (sijoittaminen paikalliseen



Kuva 6.1. *Testausjärjestelmän luokkien keskinäiset suhteet*

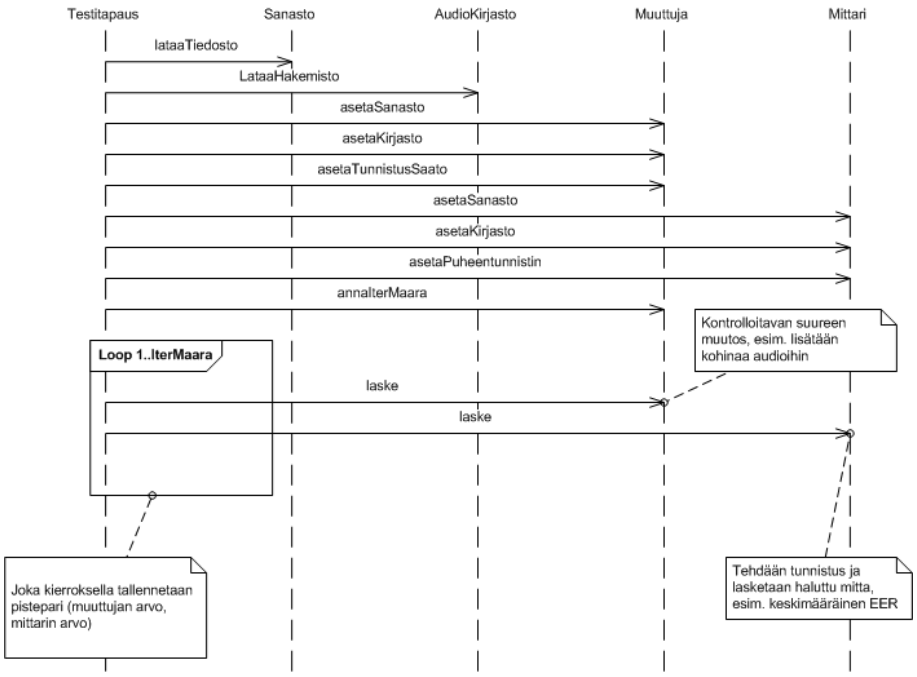
```
Testitapaus* tapaus = Testitapaus::luoTestitapaus();
Puheentunnistin* tunnistin = new TunnistinX();
Muuttuja* muuttuja = new RTF();
Mittari* mittari = new Sanastonkoko();
tapaus->asetaKorpusHakemisto(HAKEMISTO);
tapaus->asetaTulostiedosto(TULOSTIEDOSTO);
tapaus->asetaPuheentunnistin(tunnistin);
tapaus->asetaMuuttuja(muuttuja);
tapaus->asetaMittari(mittari);
tapaus->aja();
```

Listaus 6.1. *Esimerkki testitapauksen luonnista ja ajamisesta*

staattiseen muuttujaan ei ole säieturvallista). Koska testitapausta ei ole tarkoitettu periytettäväksi, ei varsinaisen rakentajan kapseloiminen private-näkyvyydelle aiheuta ongelmia.

Vaihtoehtoisia toteutuksia testitapausten identifioimiseksi olisi ollut staattisen jäsenmuuttujan käsittely rakentajassa tai this-osoittimen numeerisen arvon käyttäminen tunnisteena. Edellisen toteutuksen heikkoutena voidaan kuitenkin nähdä luokan sisäisen toteutuksen valuminen esille: vaikka staattinen jäsenmuuttuja olisi määritelty private-näkyvyydellä, se täytyisi alustaa ohjelman alussa globaalilla tasolla. Jälkimmäisen tapauksessa ohjelman virheiden jäljittäminen ja seuranta taas hankaloituisi verrattuna testitapauksen luomisjärjestyksen mukaan juoksevan numeroinnin käyttöön.

Käytännössä yksittäinen testitapaus luodaan ja ajetaan listauksen 6.1 mukaisesti koodin tasolla (muistinvaraus tehdään dynaamisesti, jotta polyformismia voidaan



Kuva 6.2. *Testitapauksen ajamisen sekvenssikaavio*

hyödyntää). Listauksessa oleva `aja()`-metodin kutsu toteuttaa sekvenssikaavion 6.2. Tämän metodin suorituksen jälkeen `new`-operaattorilla varattu muisti vapautetaan.

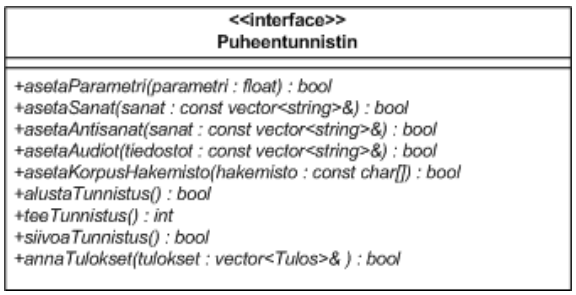
Taulukossa A.1 ovat työssä toteutetut testitapaukset. Taulukon toinen sarake kertoo, mitä testitapauksella mitataan, ja seuraavat sarakkeet (3. ja 4.) sisältävät testitapaukseen liittyvät *Mittari*- ja *Muuttuja*-luokan perilliset, jotka on esitelty kohdissa 6.4.1 ja 6.4.2. Testitapauksessa käytetty avainsanojen ja antiavainsanojen määrä on riippuvainen käytetyistä *Mittari*- ja *Muuttuja*-luokista.

Ensimmäistä kolme testitapausta on tarkoitettu tutkimaan virhemittojen muutosta tunnistettavien avainsanojen määrän kasvaessa (alkaen määrästä 100 ja loppuen määrään 1500). Testitapaukset neljästä kuuteen mittaavat virhemittojen muutosta antiavainsanojen määrän kasvaessa (antiavainsanojen määrä per avainsana, alkaen nollasta ja loppuen viiteen). Testitapaus seitsemän mittaa onko tunnistimen toiminta ”tasalaatuista”, eli toimiiko tunnistin samalla tarkkuudella eri aksenteilla. Testitapaukset kahdeksasta kymmeneen tutkivat miten tunnistustarkkuus heikkenee kohinan kasvaessa. Testitapaus 11 mittaa tunnistimen ajankäyttöä.

6.2 Avainsanojen tunnistimen abstrahointi

Testausjärjestelmä käsittelee tunnistinta käyttäen rajapintaluokkaa *Puheentunnistin*. Tunnistimelle käytetään rajapintaluokkaa, koska käytettävästä tunnistimen integraatiorajapinnasta ei voida tehdä mitään oletuksia. Lisäarvona rajapintaluokka

antaa mahdollisuuden turvallisempaan moniperiytymiseen (käytännössä tästä hyödytään, mikäli puheentunnistimen toimittajan integraatiorajapinnan toteutus tapahtuu periyttämällä).

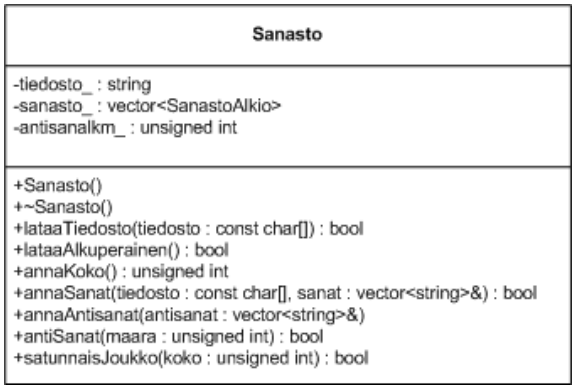


Kuva 6.3. *Avainsanojen tunnistimen rajapintaluokka*

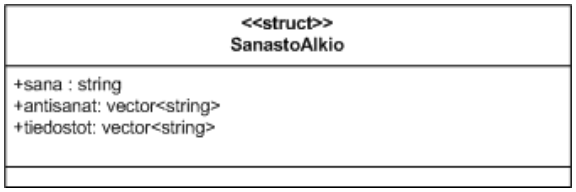
6.3 Sanaston ja äänitiedostojen käsittely

Avainsanoja ja antiavainsanoja käsitellään *Sanasto*-luokan kautta. Luokka lukee tietosisältönsä luvussa 5.4 esitellystä `wordlist.txt`-tiedostosta. Sanastoa voidaan muokata pienentämällä sitä satunnaisotannalla osajoukokseen, ja antamalla suositus sanakohtaisesta antiavainsanamäärästä. Kyseessä on suositus, koska tässä toteutettavan luvun 5.3 kolmannen ehdon täyttyminen saattaa estää tämän suosituksen mukaisen antiavainsanojen määrän löytymisen kaikille sanoille (poikkeuksena tietenkin tapaus joilloin antiavainsanojen haluttu määrä on nolla). Luokan sisäisenä toteutuksena toimivat `vector`-säiliöön tallennetut *SanastoAlkio*-tietueet, joiden tietosisältö on kuvassa 6.5.

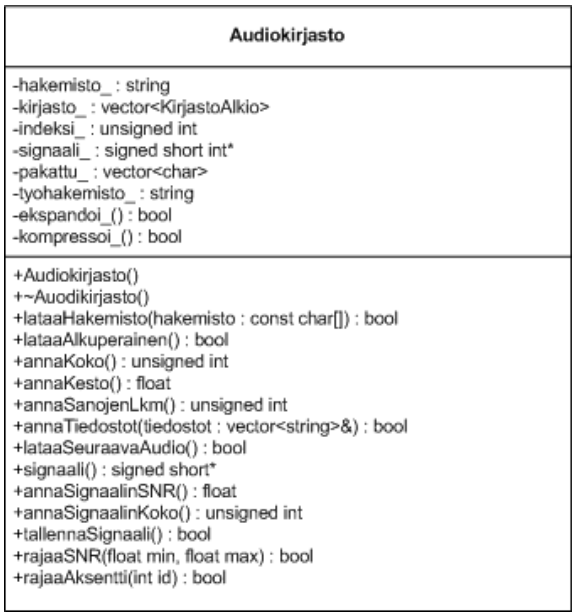
Äänitiedostoja valikoidaan ja muokataan *Audiokirjasto*-luokan kautta. Luokka lukee tietosisältönsä luvussa 5.4 esitellystä `info.txt`-tiedostosta. Luokan toteutus noudattelee lukua 5.1.2. Luokan sisäisenä toteutuksena toimivat *vector*-säiliöön tal-



Kuva 6.4. *Sanaston käsittelyyn tarkoitettu luokka*

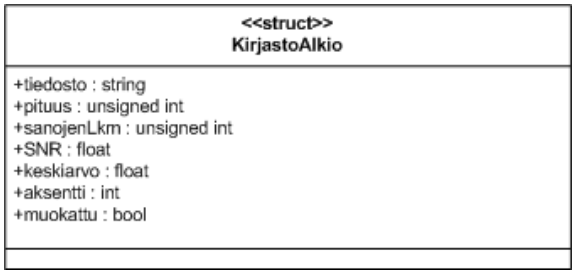


Kuva 6.5. *Sanaston sisäisessä toteutuksessa käytetty tietorakennealkio*



Kuva 6.6. *Äänisignaalien käsittelyyn tarkoitettu luokka*

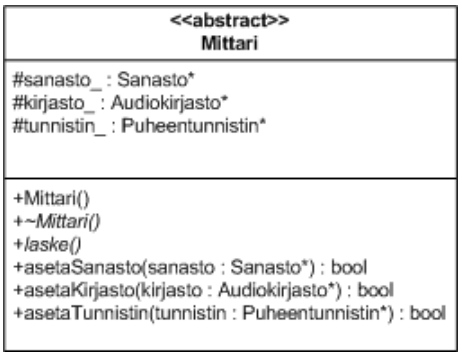
lennetut *KirjastoAlkio*-tietueet, joiden tietosisältö on kuvassa 6.7. Signaalin kompandointi pidetään myös luokan sisäisenä asiana: Luokan käyttäjä näkee äänitiedostojen sisällöt vain puretussa muodossa. Koska kaikkia äänitteitä ei luonnollisestikaan voida säilyttää muistissa, ja ne täytyy myös saada välitettyä tunnistimelle, täytyy niiden muokkaamisessa huomioida rinnakkaisuudesta johtuvat seikat. Kunkin testitapauksen omistaessa *Audiokirjasto*-olionsa, ne asettavat niille yksilölliset työhakemistot joihin muokatut äänitteet tallennetaan.



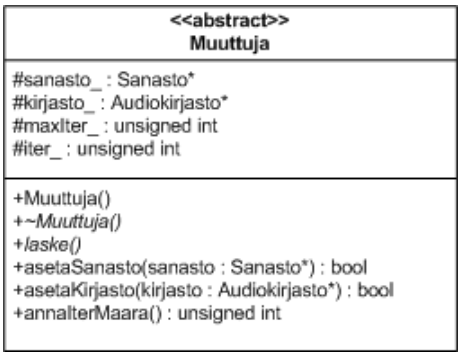
Kuva 6.7. *Audiokirjaston sisäisessä toteutuksessa käytetty tietorakennealkio*

6.4 Suureiden käsittely

Mitattavien ja kontrolloitavien suureiden käsittelyn yksinkertaistamiseksi luodaan molemmille kantaluokat, joista halutut suureet periytetään. Kantaluokkia ei ole tarvetta jättää täysin abstrakteiksi, koska suureiden toiminnan perustana oleva sanaston ja äänitiedostojen käsittely vaatii kaikille suureille näiden yhteyksien luomiseksi tarvittavat *mutaattorit*, eli jäsenfunktiot joilla muokataan olion tietosisältöä. Lisäksi tarvitaan mutaattori tunnistimen rajanpinnan saamiseksi mittaussuureluokalle. Mitattavien suureiden kantaluokan nimenä on *Mittari* ja kontrolloitavien suureiden kantaluokan nimenä on *Muuttuja*. Luokkien tietosisältö määritellään protected-näkyvyydelle, jotta se on periytettyjen luokkien käsiteltävissä.



Kuva 6.8. *Mitattavien suureiden kantaluokka.*



Kuva 6.9. *Kontrolloitavien suureiden kantaluokka.*

6.4.1 Mittarit

Tarvittavat mittarit luodaan periyttämällä ne kuvan 6.8 kantaluokasta ja kirjoittamalla niiden implementointi luvun 3 mukaisesti.

ROCPiste

Luokkaa *ROCPiste* käytetään laskemaan aliluvussa 3.1 esitellyn ROC-käyrän haluttu piste. Käytännössä luokka toimii etsimällä parametrilliselle rakentajalle

```
ROCPiste(float painoFPR, float painoFNR)
```

annettujen suhdelukujen mukaisen ROC-käyrän pisteen käyttäen puolitushakua. Oletusrakentaja käyttää molempina suhdelukuina ykköstä, joka vastaa EER-kohdan mittaamista. Luokka ei muuta *Sanasto*- ja *Audiokirjasto*-luokan olioiden tilaa.

Työssä tulkitaan tunnistuksessa tapahtuvat virheet äänitallenteen tarkkuudella

- Tyypin 1 virhe: äänitiedostosta tunnistetaan avainsana, joka sinne ei kuitenkaan sisälly
- Tyypin 2 virhe: äänitiedostosta ei tunnisteta avainsanaa, joka siihen sisältyy

Nämä määritelmät ovat järkeviä, kun yksittäisen äänitiedoston ajallinen kesto ei kasva liian suureksi; työssä käytettyjen äänitiedostojen enimmäispituus oli 20 sekuntia, joka tässä tapauksessa hyväksyttiin riittävän lyhyeksi löydettävien avainsanojen kohdistamiseksi.

Haluttujen pisteiden laskenta vaatii ROC-käyrän arvojen evaluointia eri päätöspinnan sijainneilla. Jos tämän sijainnin oletetaan vaikuttavan ennustettavasti virhemittoihin, voidaan EER, 1:2-, ja 2:1-virhe etsiä käyttämällä puolitushakua. Koska käytännössä ROC-käyrä koostuu diskreeteistä tasoista, ei haettua pistettä välttämättä ole löydettävissä. Näin ollen puolitushaulle täytyy asettaa yläraja iteraatioiden määrälle (`MAX_SEARCH_N`). Listauksessa 6.2 pseudokoodissa oletetaan säätöparametrin arvojoukon olevan väliltä $[0, 1]$, jossa tunnistimen toiminta on ”liberaaleinta” (suurin väärin tunnistettujen osuus) arvolla 0 ja ”tiukinta” arvolla 1 (suurin tunnistamatta jääneiden osuus).

Puolitushaun perustapaus etsii järjestetystä taulukosta haluttua alkiota [16, s. 193]. Listauksessa 6.2 on perustapauksen kokonaislukuindeksin sijaan liukulukuparametri `saatoParametri` ja taulukon alkion sijaan `poikkeama`. Jälkimmäisen oletetaan käyttäytyvän monotonisesti, ts. vastaavan järjestettyä taulukkoa. Perustapauksessa etsittiin eksaktisti oikeaa taulukon alkiota, mutta sovelletussa käytössä tyydytään riittävään tarkkuuteen (vakio `MIN_DEV`).

ROCPisteKA

Luokka *ROCPisteKA* laskee *ROCPiste*-luokan avulla ROC-käyrän pisteitä, mutta rajaten itse halutun kokoisen avainsanajoukon ja laskemalla virhemitan usean

```
MAX_SEARCH_N = 20;
saatoParametri = 0.5;
vasenParametri = 0;
oikeaParametri = 1;
while (k < MAX_SEARCH_N)
{
 k++;
 // Tee tunnistus, kerää tulokset muuttujaan tulokset
 tulokset = TeeTunnistus(saatoParametri);

 FPR = VaarinTunnistetut(tulokset);
 FNR = TunnistamattaJaaneet(tulokset);

 poikkeama = painoFPR*FPR - painoFNR*FNR;

 // Jos poikkeama halutusta kohdasta alle minimin,
 // palautetaan se
 if (abs(poikkeama)) < MIN_DEV)
 {
        break;
 }

 // Jos erotus samanmerkkinen, haluttu piste ei ole
 // näiden välissä
 if (sign(vasenPoikkeama) == sign(poikkeama))
 {
        vasenPoikkeama = poikkeama;
        vasenParametri = saatoParametri;
 }
 else
 {
        oikeaParametri = saatoParametri;
 }
 // Etsitty parametri on välissä
 saatoParametri = oikeaParametri/2 + vasenParametri/2;
}
// Palauta pari FPR, FNR
```

Listaus 6.2. *Muokattu puolitushaku*

mittauksen keskiarvona. Keskiarvottaminen parantaa saatavan ROC-käyrän pisteen luotettavuutta. Luokka siis muuttaa käytössänsä olevan *Sanasto*-luokan olion tilaa. Luokan alustus tapahtuu parametrisella rakentajalla

```
ROCPisteKA(float painoFPR, float painoFNR, float osajoukko,
           unsigned int iteraatiot) ,
```

jonka kaksi ensimmäistä parametria toimivat kuten *ROCPiste*-luokassa, kolmas määrittää käytettävän osajoukon suhteellisen koon väliltä $[0, 1]$, ja viimeinen määrittää kuinka monen iteraation yli keskiarvo lasketaan. Luokan oletusrakentaja käyttää kaikkina arvoina ykköstä.

RTF

Luokka *RTF* laskee luvun 3.2 mukaisen ajankäyttömitan. Käytännössä tunnistukseen kuluva aika saadaan mittaamalla kohdan 6.2 *teeTunnistus()*-funktion suorituksen ajallinen kesto (Windows-käyttöjärjestelmän ohjelmointirajapinnan funktioilla *QueryPerformanceCounter()* ja *QueryPerformanceFrequency()*). Tunnistimen säätöparametriksi asetetaan $0,5$. Kyseinen arvo on melko mielivaltaisesti valittu säätöparametrin arvoalueen puolivälistä; säätöparametrin arvolla ei oletettu olevan merkittävää vaikutusta tunnistimen ajankäyttöön. *Audiokirjasto*-yhteyden sijaan tunnistimelle annetaan yksi äänitiedosto käsiteltäväksi, joka on luotu yhdistämällä kaikki akustinen materiaali. Akustinen materiaali haluttiin käsitellä tässä tapauksessa yhtenä tiedostona, koska tällä pyrittiin vähentämään useiden pienten tiedostojen käsittelystä johtuvaa (pääasiassa levynlukuun liittyvää) yleiskuormitusta tunnistusprosessille. Luokka ei vaikuta *Sanasto*-luokan olion tilaan.

6.4.2 Muuttujat

Tarvittavat muuttujat luodaan periyttämällä ne kuvan 6.9 kantaluokasta ja kirjoittamalla niiden toteutus kohtien 5.1.2, 5.2.2 ja 5.3.2 mukaisesti.

Sanastonkoko

Luokka *Sanastonkoko* muokkaa tunnistettavien avainsanojen määrää käyttäen *Sanasto*-luokan rajapintaa. Luokka varioi käytettävien osajoukkojen kokoa viidellätoista eri arvolla, alkaen joukon koosta 100 ja loppuen kokoon 1500. Luokka ei vaikuta äänitiedostoihin.

Antiavainsanat

Luokka *Antiavainsanat* muokkaa käytössä olevien antiavainsanojen määrää *Sanasto*-luokan rajapintaa. Luokka varioi käytettävien antiavainsanojen määrää per avainsana arvoilla nollasta viiteen. Luokka ei vaikuta äänitiedostoihin.

Aksentti

Luokka *Aksentti* muokkaa *Audiokirjasto*-luokan olion tilaa rajaamalla kullakin iteraatiolla käyttöön vain yhden aksentin vuorollaan, väliltä 1–4. Luokka ei vaikuta sanastoon.

SNR

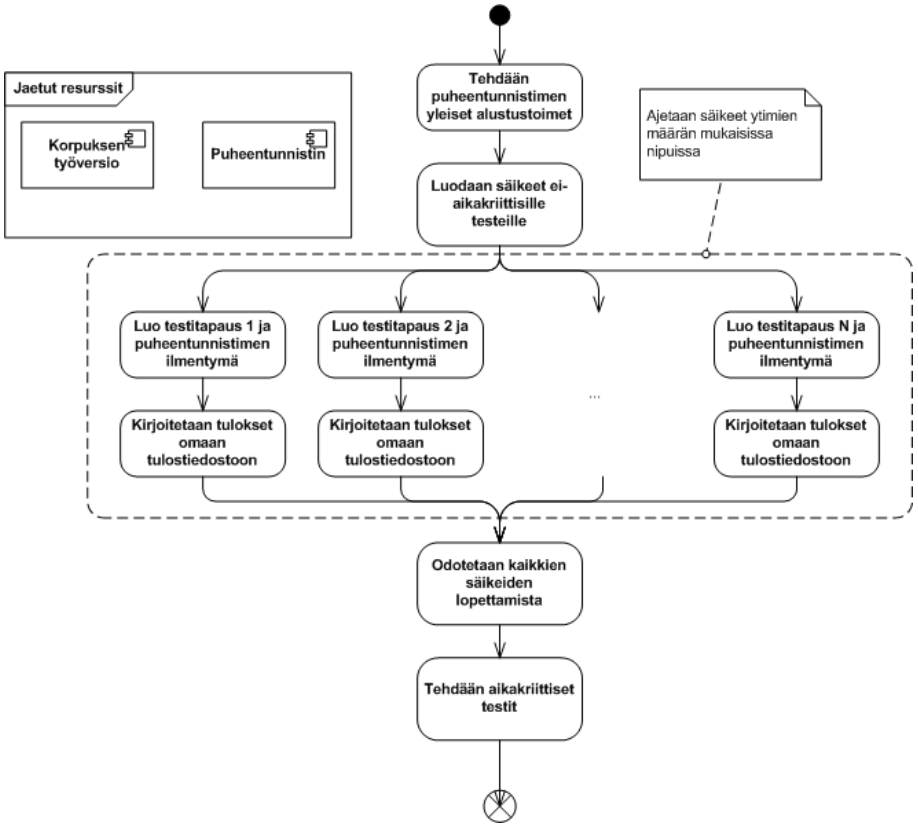
Luokka *SNR* muokkaa tunnistuksen kohteena olevien äänisignaalien signaali-kohinasuhdetta käyttäen *Audiokirjasto*-luokan rajapintaa. Luokka ei vaikuta avainsanoihin tai antiavainsanoihin. Luokka rajaa äänitiedostoista käyttöön vain ne joilla on SNR-arvo vähintään 25 dB, jonka jälkeen lisää kuhunkin näihin `iteraatio_`-muuttujansa mukaisen määrän valkoista kohinaa, ja laskee kunkin modifioidun signaalin uuden SNR-arvon. Valkoista kohinaa käytetään yksinkertaisena mallina siirtoteissä tapahtuvista häiriöistä.

6.5 Rinnakkaistaminen

Testiympäristönä toimivan palvelimen moniytimisyydestä johtuen testitapaukset ovat ajettavissa huomattavasti nopeammin rinnakkaistamalla ne. Tehokkaimmin tämä tapahtuu ajamalla ytimien verran säikeitä. Useamman säikeen käyttämisellä ei saavuteta enää tehokkuushyötyjä, vaan pikemminkin päinvastoin: tilanteessa jossa säikeitä on enemmän kuin suoritinytimiä, käyttöjärjestelmä joutuu vaihtelemaan ytimien ajamia säikeitä luodakseen virtuaalisen moniajon. Säikeiden lomittaminen lisää laskentarasitetta eikä enää nopeuta testiprosessia.

Testitapauksista voidaan rinnakkaistaa ne joissa ei mitata ajankäyttöä, eli ei-aikakriittiset testit.

Rinnakkaislaskennan implementoinnissa käytetään OpenMP-kirjastoa. Kirjaston käyttäminen tapahtuu C++-kielessä pragma-kääntäjädirektiiveillä. Oletusarvoisesti OpenMP luo rinnakkaisia säikeitä tietokoneen ytimien määrän mukaisesti, ja osaa esimerkiksi jakaa silmukan suorituksen tälläisiin säienippuihin. Listauksissa 6.3 ja 6.4 on esitetty tämän työn testausjärjestelmän tapa käyttää OpenMP:tä. Listauksen 6.3 for-silmukan loppuun sisältyy implisiittisesti synkronointi, jossa pääohjelman suoritus jatkuu vasta säikeiden lopetettua. Testitapaukset ovat liitteen A mukaiset.



Kuva 6.10. *Ohjelmavuo rinnakkaisuuden näkökulmasta*

6.6 Virheenkäsittely

Ohjelman virheenkäsittely perustuu pääasiallisesti funktioiden paluuarvojen tutkimiseen. Ohjelmassa ei käytetä omia poikkeuksia. Käytetyt valmiskomponentit voivat kuitenkin heittää poikkeuksia, jotka luokat vuotavat muuttumattomina ulos (*poikkeusneutraalius*) sekä takaavat etteivät hukkaa resursseja ja ovat tuhottattavissa poikkeuksen vuotamisen jälkeen (*perustakuu*).

```
int main()
{
  ...

  // ei-aikakriittiset testit
  #pragma omp parallel for
  for (int k = 1; k < 11; k++)
  {
        ajaTestitapaus(k);
  }

  // aikakriittiset testit
  ajaTestitapaus(11);

  ...
}

int ajaTestitapaus(int id)
{
  Testitapaus* tapaus = Testitapaus::luoTestitapaus();
  Puheentunnistin* tunnistin = new TunnistinX;
  Muuttuja* muuttuja = NULL;
  Mittari* mittari = NULL;
  int retvalue = 0;

  switch (id)
  {
        // luodaan kunkin testitapauksen mittari ja muuttuja
        ...
  }

  tapaus->asetaKorpusHakemisto(HAKEMISTO);
  tapaus->asetaTulostiedosto(TULOSTIEDOSTOIDNMUKAAN);
  tapaus->asetaPuheentunnistin(tunnistin);
  tapaus->asetaMuuttuja(muuttuja);
  tapaus->asetaMittari(mittari);
  retvalue = tapaus->aja();

  delete mittari;
  delete muuttuja;
  delete tunnistin;
  delete tapaus;

  return retvalue;
}
```

Listaus 6.3. *Testitapausten rinnakkaistaminen OpenMP-kirjastolla*

```
Testitapaus* Testitapaus::luoTestiTapaus()
{
  ...
  #pragma omp critical
  {
        // testitapauksen dynaaminen luominen ja
        // tunnistenumeron käsittely
        ...
  }
  ...
 }
```

Listaus 6.4. *Kriittisen osion suojaaminen OpenMP-kirjastolla*

7. TOTEUTUKSEN ARVIOINTI

Tässä luvussa arvioidaan toteutuksen arkkitehtuuria ja toteutettuja testitapauksia (taulukko A.1).

7.1 Ohjelmiston arkkitehtuurin arviointi

Aiemmassa luvussa esitetty testausjärjestelmä mahdollistaa vapaasti valittavan avainsanojen tunnistimen testaamisen. Muu testausjärjestelmä voidaan pitää muuttumattomana, ja vaihtaa vain *Puheentunnistin*-luokan toteutus. Teoriassa rajapinnan taakse jäävä avainsanojen tunnistin voi olla lähes minkälainen tahansa. Käytännössä kuitenkin muun kuin grafeemeja syötteenä ottavan tunnistimen integrointi vaatii huomattavan suuren työmäärän, koska integraatiokomponentin täytyy tällöin tehdä esimerkiksi muunnokset grafeemijonoista foneemeiksi. Puheentunnistimen parametrointi on myös suppeaa nykyisessä toteutuksessa. Tunnistinta voidaan säätää vain yhden liukulukuparametrin kautta. Parempi ratkaisu olisi tunnistimen rajapinnan määrittely siten, että tunnistimelta voidaan kysyä sen käyttämät parametrit.

Toteutetun testausjärjestelmän liittyminen sanastoon ja äänitteisiin tapahtuu eiperiytettyjen luokkien kautta. Tämä tarkoittaa kiinteämpää sitoutumista niiden toteutukseen kuin ehkä on tarpeellista. Väljät riippuvuudet yksinkertaistavat myös järjestelmän yksikkötestaamista, mitä nyt ei ole huomioitu lainkaan. Yksikkötestien eristäminen esimerkiksi vain yhden luokan testaamiseen helpottuu, kun usein tarvittavat tynkätoteutukset tai Mock-oliot muiden luokkien osalta voidaan toteuttaa rajapintamäärittelyjä vastaan.

Mittari- ja *Muuttuja*-luokkien jäsenmuuttujien protected-näkyvyys rikkoo luokan kapseloinnin. Jäsenmuuttujat pitäisi siirtää private-näkyvyydelle, ja kirjoittaa niiden käyttöä varten tarvittavat metodit protected-näkyvyydellä.

7.2 Testitapausten arviointi

Avainsanojen tunnistimen tunnistustarkkuuden mittaaminen tehdään toteutuksessa laskemalla ROC-käyrän pisteitä. Kuitenkin parempina vaihtoehtoina voitaisiin harkita PR-käyrän (vinon luokkajakauman vuoksi) tai DET-käyrän käyttöä (parempi tarkkuus pienillä virhearvoilla). Lisäksi koska nykyinen toteutus sisältää kolmen eri

ROC-käyrän pisteen laskennan puolitushaulla, voidaan kyseenalaistaan tämän toteutuksen järkevyys. Vaihtoehtoinen keino olisi koko ROC-käyrän laskenta kerran, ja haluttujen pisteiden poiminta tuloksista. Tästä ratkaisusta saataisiin myös suoraan laskettua estimaatti AUC-tunnusluvulle.

Antiavainsanojen vaikutus ROC-käyrästä laskettuihin pisteisiin osoittautui hiukan hankalasti tulkittavaksi. Antiavainsanojen vaikutus perustuu väärien positiivisten määrän vähentymiseen. Parempi tapa niiden vaikutuksen mittaamiseen olisi ollut kiinteän FNR-arvon valinta, jota vastaavan FPR-arvon muutosta olisi tutkittu antiavainsanojen määrää muutellessa.

Äänitteiden SNR-arvojen muokkaaminen tapahtuu nyt valkoista kohinaa lisäämällä. Siirtoteihin liittyvää kohinaa voitaisiin lisätä myös kaistanrajoitetuilla satunnaiskohinoilla, jaksollisilla häiriöillä tai satunnaisilla amplitudipiikeillä aikatasossa. Järjestelmän toteutus ei myöskään sisällä minkäänlaista taustamelun, esimerkiksi automelun tai taustalta kuuluvan puheen, vaikutuksen mittausta tunnistustarkkuuteen.

Tunnistimen säätöparametriksi valitaan nyt RTF-mittauksessa etukäteen mielivaltaisesti valittu arvo, koska säätöparametrilla ei oleteta olevan suurta merkitystä ajankäytön kannalta. Olisi kuitenkin parempi jos tämä arvo saataisiin laskettua ”keskimäärin hyvän” tunnistuksen tuottavaksi. Vaihtoehtoisesti oletus säätöparametrin pienestä vaikutuksesta ajankäyttöön voitaisiin näyttää toteen, esimerkiksi testitapauksella jossa säädetään säätöparametria ja mitataan RTF-arvoa.

8. YHTEENVETO

Työssä kuvailtiin avainsanojen tunnistuksen testausjärjestelmä, joka abstrahoi käytetyn tunnistimen ja mittaussuureet. Lisäksi esiteltiin testitapauksia jotka toteutettiin diplomityön puitteissa.

Testausjärjestelmän toteutus sisältää useita yksityiskohtia joita jatkokehittämällä ohjelmiston laatua voidaan parantaa. Kohdassa 7.1 tehdyt havainnot liittyvät kaikki liian matalaan abstraktiotasoon. Testitapauksien arvioinnissa (kohta 7.2) suurimmaksi ongelmaksi osoittautui valittu tunnistustarkkuuden mittari.

Työ poikkeaa aiemmista tunnistustarkkuuden mittaukseen liittyvistä tutkimuksista käytännönläheisemmällä otteellaan. Myös tehdyn työn hyödyt ovat käytännöllisempiä kuin akateemisemmin orientoituneissa papereissa. Testausjärjestelmä tarjoaa päätöksentekoa tukevaa tietoa ohjelmistohankintoihin, joissa pohditaan hankintaa useamman avainsanojen tunnistimen kesken. Vaikka testausjärjestelmä sisältää puutteita, tarjoaa se kuitenkin yhteismitallisen vertailutuloksen eri tunnistimille. Koska testausjärjestelmään on määritelty yksinkertainen C++-integraatiorajapinta tunnistimille, saadaan testauksen yhteydessä myös erittäin tärkeää käytännön tietoa testattavan tunnistimen ohjelmistorajapinnan integroitavuudesta. Tämän diplomityön ulkopuolelle jätettiin työhön todellisuudessa liittynyt tunnistimen integrointiominaisuuksien arviointilomakkeen malli; näihin seikkoihin liittyvä tieto on kuitenkin usein ainakin yhtä tärkeää kuin mittauksilla saadut tunnusluvut.

Vaikka taulukon A.1 testitapaukset antavatkin tarkkoja lukuja tunnistimen eri ominaisuuksista, todettiin työn aikana varsinaisten yksittäisten tunnuslukujen sijaan niiden dynamiikan ja kokonaisvaltaisen tulkinnan olevan oleellisempi seikka. Testausjärjestelmästä saatavien tulosten raportoinnin pääasiallinen anti lukijan kannalta onkin seuraaviin kysymyksiin vastaamisessa:

- Testitapaukset 1, 2, 3 ja 11: Kuinka monta avainsanaa on järkevää maksimissaan käyttää?
- Testitapaukset 4, 5, 6: Voidaanko tunnistustarkkuutta merkittävästi parantaa antiavainsanoilla?
- Testitapaus 7: Onko virhemitta samalla tasolla eri aksenteilla?

- Testitapaukset 8, 9, 10: Heikkeneekö tunnistustarkkuus hallitusti kohinan lisääntyessä vai tapahtuuko jossain kohtaa äkillinen romahdus?

Testausjärjestelmää on onnistuneesti hyödynnetty suunnitellussa käyttötarkoituksessaan, eli avainsanojen tunnistimien vertailussa. Järjestelmää on kuitenkin syytä kehittää edelleen, ja sujuvan jatkokehityksen kannalta sen perusrakenne onkin onnistunut.

LÄHTEET

[1] Deller, J., Hansen, J. and Proakis, J. Discrete-Time Processing of Speech Signals. New York 2000, The Institute of Electrical and Electronics Engineers Inc. 908 p.

[2] Koppinen, K. Puheenkäsittelyn menetelmät -luentomoniste [WWW]. Tampereen teknillinen yliopisto. [viitattu 20.5.2011]. Saatavissa: http://www.cs.tut.fi/kurssit/SGN-4010/sgn4010.pdf

[3] Koppinen, K. Puheen koodaus-luentomoniste [WWW]. Tampereen teknillinen yliopisto. [viitattu 24.8.2011]. Saatavissa: http://www.cs.tut.fi/kurssit/SGN-4050/sgn4050.pdf

[4] Thambiratnam, A. Acoustic Keyword Spotting in Speech with Applications to Data Mining. PhD thesis. Brisbane 2005. Queensland University of Technology. 220 p.

[5] Haikala, I. ja Märijärvi, J. Ohjelmistotuotanto. Helsinki 2004, Talentum. 440 s.

[6] Rintala, M. ja Jokinen, J. Olioiden ohjelmointi C++:lla. Helsinki 2005, Talentum. 377 s.

[7] Pallett, D.S. A look at NIST'S benchmark ASR tests: past, present and future. Proceedings of the 2003 IEEE Workshop on Automatic Speech Recognition and Understanding. St. Thomas, Virgin Islands, 30 November-3 December 2003. The Institute of Electrical and Electronics Engineers, Inc. pp. 483–488.

[8] Szöke, I., Schwarz, P., Matejka, P. and Karafiát, M. Comparison of keyword spotting approaches for informal continuous speech. Proceedings of the INTERSPEECH 2005-Eurospeech, 9th European Conference on Speech Communication and Technology. Lisbon, Portugal, September 4-8, 2005. ISCA. pp. 633–636.

[9] Aulanko, R. Foneettinen kirjoitus. Teoksessa: Iivonen, A., Aulanko, R., ja Vainio, M. Monikäyttöinen fonetiikka. 3. painos. Helsinki 2005, Yliopistopaino. s. 21–32.

[10] Duda, R., Hart, P., Stork, D. Pattern Classification. 2nd edition. New York 2001, John Wiley & Sons, Inc. 654 p.

[11] Haikala, I., Järvinen, H-M. Käyttöjärjestelmät. Toinen painos. Helsinki 2004, Talentum. 246 s.

[12] Manning, C., Raghavan, P., and Schütze, H. Introduction to Information Retrieval. Cambridge 2008, Cambridge University Press. 544 p.

[13] Hand, D. Measuring classifier performance: a coherent alternative to the area under the ROC curve. Machine Learning 77(2009)1, pp. 103–123.

[14] Davis, J., Goadrich, M. The relationship between Precision-Recall and ROC curves. Proceedings of the 23rd international conference on Machine learning, Pittsburgh, Pennsylvania, June 25-29, 2006. New York 2006, ACM Press. pp. 233–240.

[15] Suomi, K., Toivanen, J. and Ylitalo, R. Finnish sound structure. Phonetics, phonology, phonotactics and prosody. Oulu 2008, Oulu University Press. 149 p.

[16] Pattis, R. Textbook Errors in Binary Searching. Proceedings of the nineteenth SIGCSE technical symposium on Computer science education, Atlanta, Georgia, United States, 1988. ACM. pp. 190–194.

[17] Moore, R. Evaluating speech recognizers. IEEE Trans. Acoustics Speech and Signal Processing 25(1977)2, pp. 178–183.

[18] Silaghi, M. and Vargiya, R. A new evaluation criteria for keyword spotting techniques and a new algorithm. Proceedings of the INTERSPEECH 2005-Eurospeech, 9th European Conference on Speech Communication and Technology. Lisbon, Portugal, September 4-8, 2005. ISCA. pp. 1593–1596.

[19] Yao, X., Bhutada, P., Georgila, K., Sagae, K., Artstein, R. and Traum, D. Practical Evaluation of Speech Recognizers for Virtual Human Dialogue Systems. Proceedings of the 7th International Conference on Language Resources and Evaluation. Valletta, Malta, May 19-21, 2010. European Language Resources Association.

A. TESTITAPAUKSET

Taulukko A.1. *Testitapausten tiedot*

#	Mitä testataan	Mittari	Muuttuja	Avainsanat	Antiavainsanat	Äänitallenteet
1	Tunnistustarkkuus	`ROCPiste(1,1)`	`Sanastonkoko`	Kasvava määrä	Ei	Kaikki
2	Tunnistustarkkuus	`ROCPiste(2,1)`	`Sanastonkoko`	Kasvava määrä	Ei	Kaikki
3	Tunnistustarkkuus	`ROCPiste(1,2)`	`Sanastonkoko`	Kasvava määrä	Ei	Kaikki
4	Tunnistustarkkuus	`ROCPisteKA(1,1,0.1,15)`	`Antiavainsanat`	Kiinteä määrä	Kasvava määrä	Kaikki
5	Tunnistustarkkuus	`ROCPisteKA(2,1,0.1,15)`	`Antiavainsanat`	Kiinteä määrä	Kasvava määrä	Kaikki
6	Tunnistustarkkuus	`ROCPisteKA(1,2,0.1,15)`	`Antiavainsanat`	Kiinteä määrä	Kasvava määrä	Kaikki
7	Skaalautuvuus	`ROCPisteKA(1,1,0.1,15)`	`Aksentti`	Kiinteä määrä	Ei	Yksittäinen aksentti
8	Häiriösietoisuus	`ROCPisteKA(1,1,0.1,15)`	`SNR`	Kiinteä määrä	Ei	Muokattu, tietty SNR
9	Häiriösietoisuus	`ROCPisteKA(2,1,0.1,15)`	`SNR`	Kiinteä määrä	Ei	Muokattu, tietty SNR
10	Häiriösietoisuus	`ROCPisteKA(1,2,0.1,15)`	`SNR`	Kiinteä määrä	Ei	Muokattu, tietty SNR
11	Suorituskyky	`RTF`	`Sanastonkoko`	Kasvava määrä	Ei	Koostetiedosto