TALLINNA ÜLIKOOL

Informaatika Instituut

Statistikapõhise tarkvara loomine morfoloogiliste kollokatsioonide eraldamiseks eesti keele tekstidest

Bakalaureusetöö

Autor: Sander Ots

Juhendaja: Erika Matsak

Autor: ..... „ ..... „2012

Juhendaja: ..... „ ..... „2012

Instituudi direktor: ..... „ ..... „2012

Tallinn 2012

Autori deklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole keegi teine varem kaitsmisele esitanud. Kõik töö koostamisel kasutatud autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

…… ……

(kuupäev) (autor)

Sisukord

Sissejuhatus

Õppides võõrkeeli tuleb ette raskusi. Raskused on tihti seotud reeglitega, mis õppija emakeeles puuduvad. Keele õppimise juures on tähtsal kohal lauseehitus. Bakalaureusetöö raames luuakse tarkvara lauseehituse (morfoloogia ja süntaks) ja keelekasutusmustrite (morfosüntaks) uurimiseks.

Arendatav tarkvara võimaldab leida morfosüntaktiliselt analüüsitud eesti keele tekstidest klastreid. Morfosüntaktiliste klastrite otsimise tulemusena saab leida grammatilisi malle, mida uurides saab näiteks lihtsustada eesti keele, kui võõrkeele, õpetamist. Rakenduse tulemuste alusel saavad lingvistid esile tuua reegleid, mis annab võimaluse näha keelekasutuse eripära ja keelestruktuuride tüüpilist rakendamist.

Loodud tarkvara hakkab asendama Erika Matsaku VBA[1]'s kirjutatud *excel*i makrosid (Metslang & Matsak, 2010). Lahendus võimaldab kasutajal leida morfosüntaktiliselt analüüsitud tekstidest klastreid mugavamalt ja kiiremini. Tarkvara on arendatud koostöös professor Pille Esloniga, kellega sai välja töötatud tarkvara disain.

Töö eesmärgiks on luua rakendus, mis võimaldaks lingvistidel leida eesti keele tekstidest morfoloogilised, süntaktilised ja morfosüntaktilised klastrid. Tulemuseni plaanin jõuda inkrementaalse ja iteratiivse arendusmudeliga (Abrahamsson, Ronkainen, Warsta, & Salo, 2002). Rakendust on pidevalt testitud valge kasti (*white box)* meetodit kasutades.

Töö sisaldab kuut peatükki. Esimeses peatükis selgitan tarkvara loomise vajadust (mõistete selgitused on valminud seminaritöö raames). Teises peatükis on välja toodud kuidas lingvistikas klastreid rakendatakse. Kolmandas peatükis on kirjutatud tarkvara disainist. Neljandas peatükis on loodud rakenduse ülevaade: missugune näeb välja klassistruktuur, kuidas seminaritöös leitud algoritm sai rakenduse ja mis laadi muudatused on saanud seminaritöös välja pakutud kasutajaliides. Viies peatükk on loodud tarkvara testimistulemuste alusel ja kuuendas pakutakse välja võimalusi rakenduse edasiarendamiseks.

[1] Visual Basic for Applications

1 Mõistete selgitus ja vajadus rakenduse loomiseks

Antud peatükis selgitan töös kasutatud mõistete tähendust.

1.1 Mõistete selgitused

Klaster - grupp sarnaseid elemente, mis on lähestikku. Bakalaureusetöös on selle all mõeldud järjest paiknevate sõnavormide märgendite järjendit.

Morfoloogiline analüüs – sõnavormide ja lemmade määramine. Morfoloogilised märgendid tähistavad sõnaliiki ja grammatilist vormi.

Süntaktiline analüüs – lauseehituse analüüs formaalsete märgendite järjendina. Märgendid tähistavad sõnade erinevaid rolle lauses, näiteks öeldis jt lause põhja märgendeid nagu subjekti, objekt ja erinevad laiendid. Kokku on 27 märgendit. (Müürisep, süntaktilised märgendid)

Morfosüntaktiline analüüs - ühendab endas kaks analüüsi: morfoloogilist ning süntaktilist. (Kaalep & Muischnek)

Morfosüntaktiline märgend – tähistab sõnavormi(de) kirjeldust.

N-gram märgib järjest paiknevate elementide jada. Tavalises kontekstis on elementideks sõnavormid või üksikud tähed. Selle abil saab vaadata, millised teksti osad esinevad koos teistega ja kui sageli.

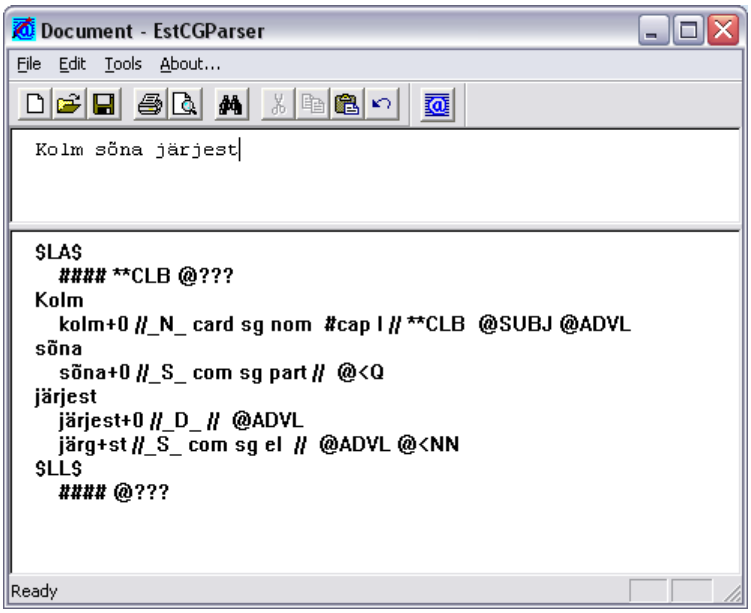
1.2 Vajadus rakenduse loomiseks

Vajadus rakenduse loomiseks on aktuaalne. Loodav rakendus võimaldab kasutajal vähese vaevaga leida pikkadest tekstidest sageduse alusel järjestatud märgendite jadasid ehk klastreid.

Seminaritöös oli toodud välja, et olemasoleva eesti keele tarkvara abil ei ole võimalik leida morfoloogilisi, süntaktilisi või morfsüntaktilisi klastreid. Uuritud sai kolme keeleuurimise tarkvara, mis toovad klastreid esile statistikapõhiselt: WordSmith Tools 5, kfNgram ja Xaira. Eesti keele süntaksianalüsaatorist (EstCGParser) saadud morfosüntaktilised märgendid said käesolevas töös koostatud programmide sisendiks.

1.3 Süntaksianalüsaatorist

EstCGParser on kirjutatud Tartu Ülikoolis Tiina Puolakaineni ja Kaili Müürisepa poolt. Rakendus kasutab VISL[2] kitsenduste grammatika parserit. Tarkvara on mõeldud akadeemiliseks uurimistööks ning abivahendiks keeleõppel. Rakendus võimaldab väljundi märgendeid tõlkida kasutajale arusaadavale kujule[Joonis 1]. (Müürisep, Eesti keele süntaksianalüsaator).



Joonis 1 - EstCGParser ekraanipilt

Jooniselt on näha, mis kujul on EstCGParseri väljund. Antud väljund on loodava rakenduse sisendiks. Sõna väljund koosneb kolmest osast: Sõna tüvi, morfoloogiline märgend ja süntaktiline märgend. Iga osa on eraldatud kahe kaldkriipsuga.

[2] Visual Interactive Syntax Learning

2 Morfoloogiliste ja süntaktiliste klastrite kasutamine lingvistikas

Selles peatükis seletatakse millised klastrid pakuvad lingvistikas huvi ning tuuakse esile, mida on võimalik klastreid uurides leida.

2.1 Morfoloogilised klastrid

Morfoloogilised klastrid kirjeldavad keelekasutusele tüüpilisi vormijärjendeid. Näiteks vene emakeelega gümnaasiumiõpilaste eesti keele olümpiaaditööde (esseed) kümme sagedamat vormijärjendit on [Näide 1][Tabel 1 lk 10]:

tegusõna + määrsõna + määrsõna (nt on veel vara)
eitus + tegusõna + määrsõna (nt ei tule enam)
määrus + määrsõna + määrsõna (nt juba kusagilt mujalt)
tegusõna + määrsõna + omadussõna (nt on väga huvitav)
määrsõna + määrsõna + nimisõna (nt väga palju emotsioone)
asesõna + nimisõna + nimisõna (nt oma õnne sepp)
tegusõna + omadussõna + nimisõna (nt on halb asi)
määrsõna + sidesõna + määrsõna (nt iseenesest ja muretult)
asesõna + tegusõna + määrsõna (nt see on praegu)
sidesõna + määrsõna + määrsõna (nt ja nii edasi)

Näide 1 - Kümme sagedasemat vormijärjendit

Võrreldes sama keele erinevate keelekasutusvariantide (nt õppijakeel, kirjakeel, murdekeel, teaduskeel jne) morfoloogilisi klastreid on võimalik esile tuua erinevatele keelekasutusvariantidele omased sõnaliigilised ja morfoloogilised (grammatiliste vormide kasutamise) tunnusjooned. Näiteks võrreldes õppijakeele morfoloogilisi klastreid kirjakeeles sageli kasutatud klastritega ilmneb see, mille poolest keeleõppija ja emakeelekasutaja loodud tekstid erinevad. Kui aga võrrelda erinevate keeleoskustasemete tekste omavahel, saame ettekujutuse sellest, mille poolest need tasemed morfoloogiliselt erinevad. Nii saame teada, missugused morfoloogilised klastrid ja vormikasutus üht või teist keeleoskustaset omavahel eristavad.

2.2 Süntaktilised klastrid

Süntaktilised klastrid näitavad tüüpilist lauseliikmete järjendit (Müürisep, süntaktilised märgendid) ning annavad informatsiooni sõnajärjele iseloomulikest joontest ühes või teises keelekasutusvariandis või siis keeleoskustasemetel. Näiteks samade vene emakeelega gümnaasiumiõpilaste eesti keele olümpiaaditööde (esseed) kümme sagedamat lauseliikmejärjendit koos näitega on toodud tabelis [Tabel 2]

2.3 Morfosüntaktilised klastrid

Nendes uuringutes, kus on kavas teada saada missuguseid morfosüntaktilisi klastreid keelekasutuses esineb, rakendatakse morfo- ja süntaksianalüsaatorit koos (Kaalep & Muischnek). Nii leitakse tekstidest need konstruktsioonid, mis on keelekasutusele iseloomulikud ning vormistikku vaadeldakse vaid sedavõrd, kuivõrd vormivalik on oluline seoses leitud konstruktsioonitüübi kasutussagedusega. Samade vene emakeelega gümnaasiumiõpilaste eesti keele olümpiaaditööde (esseed) kümme sagedamat morfosüntaktilist konstruktsiooni on toodud joonisel [Näide 2].

Selle valimi sagedasem morfosüntaktiline konstruktsioon langeb kokku sageduselt teisel kohal oleva morfoloogilise klastriga eitus + tegusõna + määrsõna (nt ei tule enam) ja sageduselt kuuenda sõnajärjemalliga @NEG @+FMV @ADVL. Inimene eelistab kasutada teatud tüüpi morfosüntaktilisi konstruktsioone. Nende tüübist oleneb, missuguseid vorme konstruktsioonis kasutada saab. Selle põhjal tekivad keelekasutusmustrid, mis võivad erinevates keelekasutusvariantides erineda, varieeruda või üldse puududa. Sama kehtib ka erinevate keeleoskustasemete kohta.

ei tule enam
on veel vara
juba kusagilt mujalt
oma õnne sepp
on ju ebanormaalne
on oma õnne
on tänapäeval nii
et see jätkub
toetus on aga
tule enam kunagi

Näide 2 - Kümme sagedasemat morfosüntaktilist konstruktsiooni

Tabel 1 - Eesti õppijakeele kümme sagedasemat morfoloogilist klastrit

Sagedus	Morfoloogiline märgend	Morfoloogiline märgend	Morfoloogiline märgend
64	_V_ main indic pres ps3 sg ps af #FinV #Intr	_D_	_D_
57	_V_ aux neg	_V_ main indic pres ps neg #FinV #Intr	_D_
52	_D_	_D_	_D_
48	_V_ main indic pres ps3 sg ps af #FinV #Intr	_D_	_A_ pos sg nom
33	_D_	_D_	_S_ com pl part
33	_P_ pos det refl sg gen	_S_ com sg gen	_S_ com sg nom
31	_V_ main indic pres ps3 sg ps af #FinV #Intr	_A_ pos sg nom	_S_ com sg nom
30	_D_	_J_ crd	_D_
30	_P_ dem sg nom	_V_ main indic pres ps3 sg ps af #FinV #Intr	_D_
28	_J_ crd	_D_	_D_

Tabel 2 - Eesti õppijakeele kümme sagedasemat süntaktilist klastrit

Sagedus	Süntaktiline märgend	Süntaktiline märgend	Süntaktiline märgend	Näide
451	**CLB @J	@SUBJ	@+FMV	et autor tahab
446	@SUBJ	@+FMV	@ADVL	autor kirjeldab mitte
415	@+FMV	@ADVL	@ADVL	on tänapäeval nii
306	**CLB @SUBJ	@+FMV	@ADVL	mis annab paljudele
261	@SUBJ	@NEG	@+FMV	Dima ei tea
241	@NEG	@+FMV	@ADVL	ei tule enam
205	**CLB @J	@ADVL	@+FMV	ja aina lähevad
199	@ADVL	@+FMV	@ADVL	tänapäeval leiagi vist
189	@NN>	@SUBJ	@+FMV	Mirja isa ostis
178	@+FMV	@ADVL	@PRD	on rahvuselt eestlane

3 Tarkvara disain

Selles peatükis toon välja kuidas arendatava tarkvara esialgne disain välja nägi. Antud disain on tulenenud tarkvara põhieesmärgist (Zhu, 2005). Loodav tarkvara peab võimaldama kasutajal pikkadest tekstidest leida klastreid morfosüntaktiliste märgendite järgi.

3.1 Koostöö filoloogidega

Tarkvara arendus on protsess, mis vajab eeltööd. Seminaritöös sai suur osa sellest ära tehtud, kuid tehtud töö vajas ühtlustamist filoloogidega. Praegune disain sai välja töötatud koos professor Pille Esloniga.

Alustasime tarkvara protsessi kirjeldamisest. Algselt plaanitud sammud olid teksti lisamine ja klastrite otsimine. Selgus, et nende kahe sammu vahel on vajalik üks lisasamm. Selleks on teksti parandamine [Joonis 2]. Kuna morfoanalüsaatorist võib tulla mitmese analüüsiga tekst, peab korrektsete tulemuste saamiseks sisestatud teksti käsitsi ühestama.

Joonis 2 - Protsessi visualiseering

Teksti lisamist on võimlik teha kahel viisil, kas tekstivälja teksti kopeerimisega või juba parandatud teksti avamisega.

Teksti parandamise juures peab kasutajal olema võimalus valida morfoanalüsaatori pakutavat märgendit. Kui lause kontekstile vastav märgend puudub, peab olema kasutajal võimalus märgendit muuta.

Klastrite otsimise juures peab kasutajal olema valik, kas otsida morfoloogilisi, süntaktilisi või morfosüntaktilisi klastreid.

3.2 Klassistruktuuri disain

Kuna tarkvara on kirjutatud objektorienteeritult, siis tuleb esialgsed objektid paika saada. Kõige väiksem objekt oleks sõna märgend. Kuna ühel sõnal võib olla mitu märgendit, on vajadus uue objekti järele, mis hoiaks endas sõna informatsiooni (sõna ja kõiki tema märgendeid). Kuna peab sõnade objekte tekstist leidma, siis läheb tarvis sõnade haldamise klassi. Sõnade haldurklassilt saaks näiteks küsida, mitu sõna antud tekstis on või lasta tal leida kasutaja lisatud tekstist uued sõnad.

Vajadus on ka klastri objekti järele. Klastri objekt hoiaks endas informatsiooni n sõna märgendi kohta, kus n on klastri suurus. Kui klastri suuruseks on kolm, hoiab klastri objekt endas kolme tekstis järjestikku paiknevat sõna märgendit. Näiteks peab olema võimalus kontrollida, kas antud klaster on teise klastriga samasugune või erineb. Sellise kontrolli tegemiseks peab olema objekt, mis omaks endas hetkel olevaid klastreid. Klass peaks suutma uusi klastreid tekitada ja kontrollima, ega olemasolevate klastrite juures samasugust klastrit ole. Vastavalt kontrolli tulemusele peaks kas tekitama uue klastri või olemasolevasse klastrisse märkima, et kordus on toimunud. Sarnaselt sõnadele peab olema klastritel klass, mis oskaks nendega majandada. Klastrihalduri klassi käest saaks küsida näiteks kõiki olemasolevaid klastreid, otsida tekstist uute parameetrite järgi uued klastrid või kõik olemasolevad klastrid eemaldada.

Lisaks sõnadele ja klastritele oleks vaja failihaldurit. Failihalduri klass omaks endas oskust olemasolevaid sõnu või klastreid kas salvestada või eksportida. Loomulikult peab ta ka salvestatud faile avama.

Tarkvara toimima panekuks on ainult üks klass puudu. Tarvis on kõike programmi koodi siduvat klassi. Antud klass, mis on ühendatud kasutajaliidese peaaknaga. Peaakna loomisel luuakse haldur klassidehaldurklasside objektid, mis saab antud tarkvara osaga suhelda. Käsud on ära seotud kasutajaliidese valikutega. Nendeks valikuteks oleks näiteks faili salvestamine, teksti lisamine või klastrite otsimine. [Lisa 2]

Kasutajaliidese juures on samuti vaja toetavaid klasse. Antud klassid tegeleksid näiteks hüpikakna või teise vahekaardi kuvamisega. Nendeks oleks näiteks teksti lisamise aken, klastrite vaatamise tabel ja klastrite otsingu jaoks valikute tegemise aken.

Mainitud klassidega peaks saama tarkvara oma eesmärkki täita. Olen jaganud antud klassid nelja paketi vahel, milleks on:ui, words, clusters ja filemanager. [Lisa 3]

4 Loodud rakendus

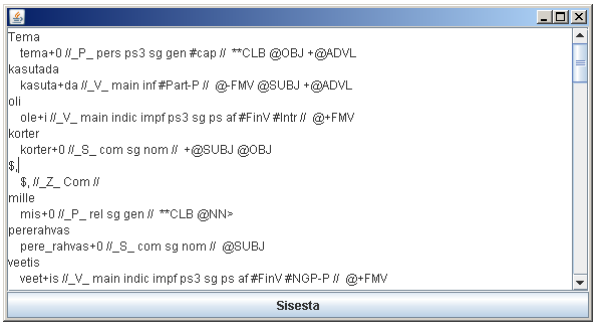
Selles peatükis on kirjeldatud valminud rakendust tavakasutaja ja arendaja poolt vaadates. Lahti on seletatud, kuidas soovitakse tarkvara kasutada ja mis toimub tarkvara sees.

4.1 Tarkvara kasutajaliides

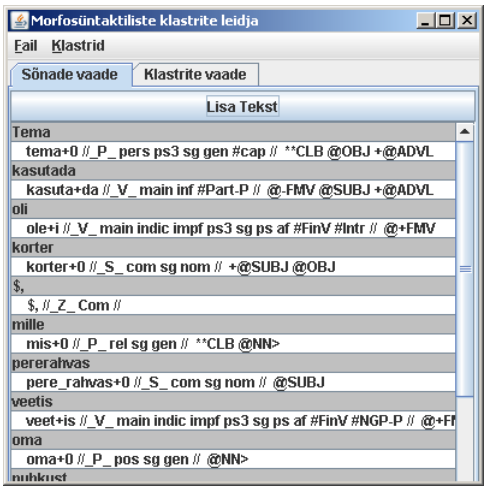
Käivitades tarkvara ilmub ekraanile aken, kus asuvad menüü ja kaks vahekaarti. Esimesel vahekaardil on “Sõnade vaade” ja teisel “Klastrite vaade”. [Joonis 3]

Joonis 3 - Tarkvara esmavaade

Alguses on sõnade vaatel nupp, mis avab uue akna teksti väljaga. Sinna sisestada morfoanalüsaatorist saadud morfosüntaktiline tekst [Joonis 4]. Nupp “Sisesta” lisab tabelisse uued read [Joonis 45]. Tabelist on näha, kas analüsaatorist on tulnud ühestatud tekst või mitte. Erisustega sõnade puhul on hetkel aktiivne sõna märgend märgitud rohelise värviga (klastrite otsimisel kasutatakse aktiivset sõna märgendit). Tabeli teises tulbas tekivad märkeruudud, kus on võimalik aktiivset sõna märgendit vahetada. Uue teksti puhul on alati aktiivne esimese sõna märgend. Kasutajal on võimalus käsitsi ja topeltklõpsuga sisenedes muuta sõna märgendit, kui see on vale. Salvestamiseks tuleb Fail-menüüst valida “Salvesta”. Tulemused salvestatakse teksti faili, mida on võimalik avada Faili-menüüst “Ava”.

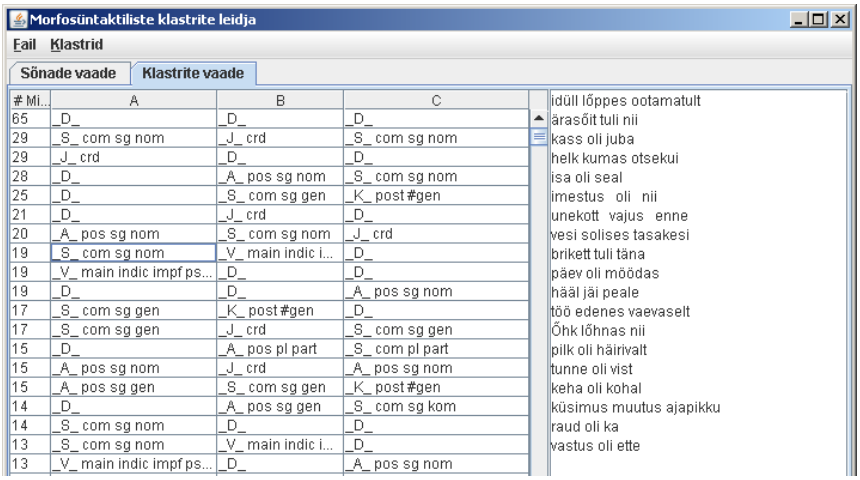


Joonis 4 - Teksti lisamine



Joonis 5 - Sõnade vaade

Klastrite vaate vahekaardil on hetkel ainult tühi tabel. Klastrite leidmiseks peab “Klastrite” menüüst valima “Otsi klastrid”. Avanenud aknas on võimalik valida, milliseid klastreid otsida. Saab otsida morfoloogilisi, süntaktilise või morfosüntaktilisi klastreid. “Lülita ‘_Z_’ märgendid sisse” laseb kasutajal kas kirjavahemärgid kaasata oma otsingus või mitte. Klastri suurus määrab mitmesõnalisi klastreid programm peaks otsima. Klõpsates nuppu “Otsi”, leiab tarkvara eelnevate valikute põhjal klastrid (üle 100 000 sõna puhul võtab tegevus veidi aega). Tekkinud tabelis on näha korduste arvu ja sõnade märgendeid [Joonis 6]. Märgendid on toodud vastavalt otsingu valikutele. Klõpsates ühe rea peal, näeb antud märgenditele vastavaid sõnu.



Joonis 6 - Klastrite vaade

Leitud tulemusi on võimalik eksportida CSV[3] faili. Seda saab teha valides "Klastrite" menüüst "Ekspordi klastrid". Tekkinud faili on võimalik avada enamikes tabelarvutuse programmides. Uue otsingu tegemiseks on võimalik otsitud klastrid kustutada. Selleks valida "Klastrite" menüüst "Kustuta klastrid".

4.2 Tarkvara klassistruktuur

Tarkvara on arendatud vabavaralist Integreeritud Arenduskeskkonda[4] Eclipse kasutades. Võrreldes tavalise tekstiredaktoriga on arenduskeskkonna eelised suured. Eclipse's on palju tööriistu, mis hoiavad aega kokku. See on peapõhjus miks antud tarkvara kasutan.

Programmi kood on kirjutatud objektorienteeritult ja jälgides koodimise häid tavasid. Kood on jagatud nelja paketi vahel. Nendeks on kasutajaliides (ui), failihaldur (*filemanager*), sõnad (*words*) ja klastrid (*clusters*).

Kasutajaliidese paketis asuvad klassid, millega saab ekraanile kuvada komponente. *MainWindow* on klass mis kutsutakse esimesena välja ja kuvab esimese akna. Tegemist on tarkvara tuumaga. Läbi selle klassi suhtlevad omavahel teised paketid.

TextInsert klass kuvab teksti lisamise akent. Lisamise nuppu vajutades saadetakse aknas olev tekst *FileManager* klassile, kus fail salvestatakse.

Tekitatud fail avatakse lisades iga sõna järel eraldaja ja saadetakse klassile *WordTagExtractor*. Antud klass on *words* paketi haldur klass. Klass tekitab iga sõna kohta *WordTags* klassi objekti, mis omakorda tekitab enda alla *WordTag* klassi objekte. *WordTag* klass on kõige väiksem objekt, kus hoitakse ühe sõna ühte märgendit. Mitme sõna märgendid annavad kokku *WordTags* klassi objekti, kust on näiteks võimalik küsida kõikide tema all olevat objektide morfosüntaktilisi märgendeid või hetkel aktiivse sõna märgendit.

Pärast objektide tekitamist teksti põhjal lisatakse andmed tabelisse. Tabelis on kaks tulpa. Esimesse tulpa kirjutatakse sõna märgendid stringina ja teise tulpa märkeruut, kui seda on vaja. Teises tulbas on tõeväärtus, kuid visualiseeritud on see tabelis märkeruuduna. Hiirevajutus tabeli teises tulbas muudab selle tõeväärtuse tõeseks ja kasutaja saab määrata ühe sõna aktiivset märgendit.

[3] *Comma-Separated-Values*
[4] IDE – *Integrated Development Environment*

Klastreid otsima asudes tervitab kasutajat teateaken. Tegemist on *Options* klassiga, mis laiendab *JDialog* akent. *JDialog* pakub kõik vajaliku hüpikakna tekitamiseks. Hüpikaknas on kasutajal võimalik määrata seadeid, mille järgi klastreid otsitakse. "Otsi" nuppu vajutades saadetakse kogutud seaded tõeväärtuste massiivina *ClusterManager* klassile.

Klastrite paketi haldur klass on *ClusterManager*. *ClusterManager* klass annab edasi *MainWindow* klassist tulnud sõnade n-grammid *Clusters* klassi objektile, kus viimane hakkab neid analüüsima. Iga n-grammi põhjal tekitatakse *Cluster* klass, mida võrreldakse olemasolevate klastrite vastu. Kui klaster on juba olemas, lisatakse olemasolevasse objekti n-grammid, mille põhjal see klaster tekitati.

Cluster on kõige väiksem objekt selles paketis. Tema hoiab informatsiooni ühe klastri kohta. Nendeks on näiteks sõnade morfoloogilised märgendid, mille põhjal klaster tekitati, sõnad mis vastavad antud klastrile, mitu sellist klastrit, tekstis on olnud ja klastri suurust. Klaster saab kontrollida, kas selline klaster on juba olemas meetodiga *checkIfExists*().

Clusters klass hoiab enda sees *Arraylisti Cluster* klassi objektidest. Ta tekitab iga n-grammi kohta *Cluster* klassi objekti ja võrdleb seda kõikide juba tekitatud objektide vastu. Kui leitakse samade märgenditega klaster, võetakse järgmine n-gram ja tehakse see protsess uuesti. See tähendab, et kui tekstis on palju erinevaid klastreid võtab nende leidmine rohkem aega. Ennem klastrite lisamist klastrite tabelisse sorteeritakse nimekiri korduste põhjal ära.

Leitud klastrid, nagu ka sõnad, lisatakse *JTable'isse*. Klastrite tabel asub klassis *ClusterView*, mis laiendab *JPanel* konteinerit. *ClusterView* klassi kuvatakse akna teisel vahekaardil. *JPanel* on jagatud kaheks. Vasakul pool on tabel klastritega. Tabeli esimesel veerul on klastri korduste arv. Järgnevas n veerus on sõnade klastrid, kus n on otsitava klastri suurus. Kui otsida nelja sõnalisi klastreid, on tabelis kokku 5 veergu. Tabelile on lisatud ka hiire kuular arusaamaks millise klastri sõnu peaks akna parempoolsem osa kuvama. Kuular edastab klõpsatud rea numbri, mida kasutatakse vastava klastri otsimisel klastrite ArrayListis. Tulemus lisatakse paremal pool olevasse *JTextArea'sse*, kuhu paigutatakse kõik ühes klastris olevad sõnad.

4.3 Salvestamine ja eksportimine

Plaanitud oli teha rakenduse hetkeseisu salvestamine tavalisse tekstifaili. Fail oleks koosnenud programmi laetud tekstist ja leitud klastritest. Kuna sama sisendi puhul leiab

tarkvara samad klastrid, ei pea leitud klastreid salvestama. Leitud klastreid peab saama eksportida.

Plaanitud juhul ei oleks algteksti salvestamine midagi teinud, kuna ei olnud võimalik sisendteksti muuta. Kuna nüüd see võimalus peab olema on muudetud algteksti salvestamine äärmiselt oluline. Programmis hetkel oleva teksti salvestamine on endiselt tekstifaili. See annab võimaluse kasutajal avada fail tekstiredaktoriga ja sealt otse muudatusi teha. Salvestamist vajavad sõnad, sõnade märgendid ja aktiivne sõna märgend. Antud andmete põhjal on võimalik salvestatud hetkeseis taastada. Salvestatakse sellisel kujul:

tööd
töö+d //_S_ com pl nom // @ADVL ## 1
töö+d //_S_ com sg part // @OBJ ## 0

Näide 3 - Lõik salvestatud failist

Selline string tuleb morfoanalüsaatori väljundist. Lisaks on lõppu lisatud eraldajaks "##" ja number. Number näitab, kas see rida on aktiivne, kui fail avada. Number ühe puhul on aktiivne, nulli puhul mitte. Soovitan salvestatud faile avada kas tarkvara endaga või *Wordpad'iga*. Kui tungiv soov on *NotePad'iga* avada, tuleb rea lõpu märgid teisendada *NotePad'ile* arusaadaval kujul.

Klastreid saab eksportida CSV-faili, mida on võimalik avada tekstiredaktori või tabelarvutus programmiga. Eksporditud fail on kujul.

"korduste arv ; esimese sõna märgend ; ... ; n-inda sõna märgend ; ; sõnad"*
Iga kordus välja toodud ja eraldatud semikooloniga. Näiteks:

"3 ; @SUBJ ; @+FMV; @NN> ; ; pererahvas veetis oma ; sügisnäitus toimus sel ; need polnud meie"

Näide 4 - Lõik eksporditud failist

4.4 Algoritmi rakendamine klastrite otsimiseks

Kuna seminaritöös oli algoritm eelnevalt välja töötatud, ei tekkinud probleeme selle rakendamisega. Suurim muutus on see, et kasutajal ei ole võimalik klastrite suuruse vahemikku määrata - kaob ära kõige välimine tsükkel algoritmis. Tegemist on tsükliga, mis oleks iga klastri suuruse korral terve teksti läbi käinud.

Algoritmi rakendus saab alguse *ClusterManager* klassi *getClusters*() funktsioonist. Funktsioon kutsutakse välja kahe parameetriga. Esiteks on tõeväärtuste massiiv kasutaja tehtud valikutest ja teiseks on otsitavate klastrite suurus. Funktsiooni sees on esimene tsükkel, kus kutsutakse välja *Clusters* klassi *addCluster*() iga võimaliku klastri kohta.

Clusters klassi *addCluster*() funktsioon kontrollib ega sellist klastrit juba tekitatud ei oleks. Ta kontrollib seda massiivis olevate *Cluster* klassi objektide vastu *checkIfExists*() meetodiga. See on teine tsükkel, kus käiakse kõik juba tekitatud klastrid läbi. Kui sellist klastrit ei leita, lisatakse klaster klastrite massiivi.

checkIfExists() meetod tagastab tõeväärtuse. Ta saab üheks parameetriks kaasa klastri, mille vastu ta kontrollib. Siin asub kolmas tsükkel, mis kontrollib igas klastris oleva sõna märgendit parameetrina kaasa tulnud sõna märgendite vastu. Esimese erinevuse tekkimise järel tagastatakse väär väärtus. Kui nad on samad, tagastatakse tõene väärtus.

4.5 Kasutajaliidese muudatused.

Esialgsest plaanist on praegune kasutajaliides palju muutunud. Enamus muudatusi on filoloogide poolt tulnud uuest informatsioonist.

Üks suurematest muudatustest on sõnade tabeli ümbertegemine. Eelnevalt plaanitud tabel, kus kahe rea kaupa asuvad sõna ja sõna märgendid, jäi ära. Kasutajal oleks olnud mugav selliselt ülesehitatud tabelist sisestatud teksti lugeda [Tabel 3]. Sellises tabelis oleks olnud raske kuvada ühe sõna morfosüntaktilisi mitmesusi. Kaasnema pidi ka võimalus määrata käsitsi õige märgend, mida klastrite otsimisel kasutatakse. Uue tabeli esimeses veerus asuvad sõnavormid ja selle võimalikud märgendid. Teine veerg on märkeruudu jaoks, millega on võimalik valida hetkel aktiivset märgendit. [Tabel 4]

Tabel 3 - Planeeritud sisestatud teksti tabel

Sõna 1	Sõna 2	Sõna 3	Sõna 4	Sõna 5	**Kerimisriba**
Sõna 1 märgend	Sõna 2 märgend	Sõna 3 märgend	Sõna 4 märgend	Sõna 5 märgend	
Sõna 6	Sõna 7	Sõna 8	Sõna 9	Sõna 10	
Sõna 6 märgend	Sõna 7 märgend	Sõna 8 märgend	Sõna 9 märgend	Sõna 10 märgend	
...	...	...	...	...	

Tabel 4- Kohandatud sisestatud teksti tabel

Sõna 1		Kerimisriba
Sõna 1 esimene märgend	☐	
Sõna 1 teine märgend	☒	
Sõna 2		
…	…	

Teiseks suureks muudatuseks on tabelite vahetamine. Eelnevalt oli kasutajal rippmenüüs valik, millega oleks saanud hetkel nähtavat tabelit vahetada. See sai asendatud kahe vahekaardiga [Joonis 7]. Vahekaartide puhul näeb kasutaja kohe, millise tabeli juures ta asub ja mida on kindlasti palju mugavam kasutada kui rippmenüüd.

Joonis 7 - Vahekaardid

Varem plaanitud uue teksti valimine ja teksti lisamine on ka muutunud. Uue teksti valimise nupp on ära kaotatud. Kasutajal on võimalik valida Fail-menüüst “Uus” ja see kaotab kõik hetkel mälus olevad objektid ära. Siis saab kasutaja hakata teksti lisama. Teksti saab lisada valides Fail-menüüst “Ava”, mis avab eelsalvestatud faili, või klõpsates nupule “Lisa tekst”. Muudatus said tehtud silmas pidades kasutajaliidese mugavust.

Klastrite vaate juures sai muudetud “Otsi” nupu asukohta. Algselt plaanitud nupu asemel on ta “Klastrite” rippmenüüs “Otsi klastreid” valiku all. See muudatus võimaldas kasutajaliidese puhtamana hoida. Otsingute valikud sai liigutatud hüpikaknasse. Sai ka muudetud korduvate sõnade väljatoomine. Plaanitud tabeliga hüpikaken sai asendatud klastritest vasakul pool asetseva tekstiväljaga. Tekstiväljaga on kasutajal lihtsam läbi käia, näiteks kõikide samalaadselt vormistatud sõnade kopeerimise võimalus.

5 Testimine

Tänapäeval on tarkvara ja operatsioonisüsteemid palju kõrgemal tasemel kui vanasti. Objektid, mida kasutan näiteks graafilise kasutajaliidese juures, on juba eelsilutud ja testitud. Seetõttu ei ole nii pingsalt vaja testida tarkvara, mis on kirjutatud kasutades keeles olevaid objekte. Küll aga on vaja testida tarkvara sisu. (Myers, Badgett, & Sandler, 2012)

Testimise põhieesmärgiks on leida üles tarkvara kitsaskohad. Oluline on tarkvara stabiilsus. Kui tarkvara iga natukese aja tagant kokku jookseb on kasutajal seda ebameeldiv kasutada. Tarvis oleks leida, kui palju kasvab tarkvara klastrite otsimise tööaeg olenevalt sisendi suurusest. Samuti kontrollida, kas tarkvara valmidus leida saja tuhande sõnalise teksti seest klastrid, toimub mõistliku aja piires. Oluline on, et tarkvara oleks platvormist sõltumatu. Seetõttu testisin seda erinevate operatsioonisüsteemide peal.

Testimisel kasutasin valge kasti meetodit (*white box testing*). Valge kasti testimise meetodi puhul on testijal selge arusaam tarkvara arhitektuurist ja koodist. See testimise meetod sisaldab endas andmevoo (*data flow*), käsuvoo (*control flow*), kodeerimise tavade (*coding practices*) ja erandite ja veakäsitlust (*exception and error handling*) (Janardhanudu & Wyk, 2005, 2009). Samuti kasutasin jõudlusteste (*performance testing*). (Gopalaswamy & Srinivasan, 2006)

5.1 Tarkvara stabiilsus

Tarkvara stabiilsuse testimise jaoks üritasin programmi kokku jooksutada. Sai proovitud erinevaid sisendeid teksti lisamise osas. Sai proovitud erinevate failide avamist lootuses, et programm ei saa sisendist aru ja jookseb kokku.

Tarkvara erinevate sisenditega kokku jooksutada ei suutnud, küll aga mittestandardse sisendi puhul ei saa tarkvara aru, kui märgendid puuduvad. Sõnade tabelis on kohe näha, et sisend on vale ja pole mõtet klastreid otsida.

Keelekorpused on suured ja nende analüüs võtab aega. Tihti tuleb ette valimeid, mille maht on üle saja tuhande või üle mitmekümne ja mitmesaja miljoni sõna. Kasutajale sellest kuidagi märku ei anta ja ei näidata, kui palju aega veel läheb, et analüüs kuvada. Samuti ei hoiatata kasutajat uue faili tegemisel, et kõik siiani tehtud töö kustub. Klastreid otsides salvestatakse hetkeseis. See võimaldab vea tekkimisele eelnev olukord taastada.

Failide salvestamine ja eksportimine ei lisa laiendit ise juurde. Ei saa kasutajatelt oodata, et nad ise iga kord kirjutaksid lõppu ’.txt’ või ’.csv’.

5.2 Klastrite leidmise ajakulu

Klastrite otsimisel kasutasin Tartu Ülikooli eesti keele korpuse morfoloogiliselt ühestatud tekste: valimis oli kokku 96574 sõnet. Aeg on mõõdetud klastrite leidmise algoritmi algusest töö lõpuni. Sinna sisse ei kuulu aeg, kui lisatakse leitud klastrid tabelisse. Otsingus kasutasin morfosüntaktilist otsingut. Aeg on tabelis millisekundites. [Tabel 5]

Tabel 5 - Klastrite leidmise ajakulu

Sõnade arv	5 000	10 000	20 000	40 000	96 574
Aega kulus (ms)	797	2 922	11 015	48 265	314719
Ajakulu ühe sõna kohta (ms)	0.159	0.292	0.550	1.206	3.258

Väikeste sõnade arvude puhul on väike ajakulu. Küllaga, kui sõnade arv kasvab, kasvab ajakulu ühe sõna leidmiseks. See ei kasva lineaarselt, kuna algoritmi keerukus läheneb ruutkeerukusele. Kuna tekstis on rohkem sõnu, leiab tarkvara järjest rohkem klastreid, mida esineb ainult ühe korra. Kuna kõik klastrid lisatakse klastrite massiivi, läheb kauem aega kontrollimaks kas uus klaster on juba massiivis olemas. Suurima teksti analüüsiks kulus aega ligikaudu viis minutit ja viisteist sekundit.

Testimiseks kasutatud arvuti parameetrid:

- Operatsioonisüsteem: Microsoft Windows XP professional (5.1, Build 2600)
- Protsessor: Intel(R) Core(TM)2 Duo CPU E6750 @ 2.66GHz (2 CPUs)
- Mälu: 2048MB RAM

5.3 Tarkvara teiste operatsioonisüsteemide peal

Eesmärk oli luua rakendus, mis oleks operatsioonisüsteemist sõltumatu. Siin on tulemused analüüsides morfoloogiliselt ühestatud tekstist võetud osa eri platvormide peal. Valitud platvormideks osutusid Windows 7, Windows XP, Mac os ja Linux(Ubuntu).

Tabel 6 - Operatsioonisüsteemid, mille peal rakendus käivitus.

Window 7 (SP 1)	Windows XP (SP 3)	Mac OSX 10.5 – Leopard	Ubuntu 12.04
Käivitus	Käivitus	Ei käivitunud	Käivitus

Tabelist on näha, et rakendus läks tööle kõikide operatsioonisüsteemide peal välja arvatud Mac OSX Mac OSX 10.5 – Leopard'i uusim java versioon - on 1.5, mis ei jooksuta uue java versiooniga arendatud tarkvara. Tarkvara kompileerides tuleks lisada java SDK 1.5 klassi teegid.

5.4 Tarkvara optimiseerimine tulenevalt testidest.

Liigne ajakulu testide tulemuste juures oli tingitud klastrite massiivi sorteerimisest. Sorteerimine toimus iga uue klastri lisamise juures. Kui massiiv suuremaks muutus, kasvas ka vastavalt ühe klastri lisamise tsükkel. Kui klastri sorteerimine klastrite otsimise algoritmist välja võtta, siis muutus algoritmi tööaeg mitmekordselt vähemaks.

Sai ka optimeeritud muutujate kasutamist tsüklite sees. Eelnevalt küsis tsükkel iga kord sõnade massiivi pikkust. Sai tehtud lisa muutuja, mis talletab endas vastava massiivi pikkust ja tsüklis seda muutujat kasutades. Tabelis on optimeeritud tulemused. [Tabel 7]

Tabel 7 - Optimeeritud algoritmi ajakulu tabel

Sõnade arv	5 000	10 000	20 000	40 000	96 574
Aega kulus (ms)	297	1000	3860	19488	130475
Ajakulu ühe sõna kohta (ms)	0.059	0.1	0.193	0.4872	1.351

Optimeeritud algoritmi jõudlustesti tulemused leiti kaks kuni kolm korda kiiremini.

Juhul, kui kasutaja kustutas olemasolevad klastrid ära, jäid tekstivälja alles viimasele klastrile vastavad sõnad. Kustutamise funktsiooni juurde sai lisatud käsk, mis määrab tekstivälja tekstiks tühja stringi.

Salvestatud ja eksporditud faililaiendeid ei kirjutatud seepärast et stringi funktsioon *concat*() ei tee päris seda, mida algselt arvasin.

6 Ettepanekud tarkvara edasiarenduseks

Klastrite ja kollokatsioonide otsimise funktsionaalsus oleks tõenäoliselt esimene samm. Klastrite otsimise juures saaks täpselt ära kirjeldada näiteks, milline peab olema teine märgend klastris.

Arenduse poole pealt ei tohiks selle teostus keeruline olla. Vaja tekitada kasutajaliides, mis võimaldaks kasutajal märgend sisse kirjutada ja määrata, kus kohas ta klastris paiknema peaks. Iga klastri juures mis tekitatakse, määratakse sisestatud märgend märgitud kohta ja otsitakse tekstist ainult sellele vastavaid klastreid.

Uurida lähemalt java ja Mac OSX-i võimalusi. Eesmärgiks saada kompileerida selline fail, mida saaks käivitada ka Mac OSX operatsioonisüsteemide peal.

Kasutajal puudub informatsioon, kaugel klastrite otsimise algoritm omadega on. Tekitada väli, mis hoiaks kasutajat toimuvaga kursis ja annaks teada kaua klastrite otsimisega veel aega läheb.

Kasutajaliides on rakenduse käivitades ilusasti paigas. Küllaga raamide suuruse muutmise juures ei püsi rakenduses olevad tabelid raamiga kaasas. Kasutajamugavust silmas pidades peaks ühendama tabeli suuruse raami enda suurusega.

Kokkuvõte.

Bakalaureusetöö eesmärk, luua rakendus, mis võimaldab lingvistel leida eesti keele tekstidest morfoloogilised, süntaktilised ja morfosüntaktilised klastrid, on saavutatud. Käesolev bakalaureuse töö annab ülevaate, kuidas eesmärgini jõuti.

Loodud tarkvara töötab Windowsi ja Linuxi operatsioonisüsteemide peal, kusjuures jre[5] 1.6 peab olema installeeritud.

Rakenduse disain, mis sai loodud seminari ja bakalaureusetöö raames, on realiseeritud. Seminaritöö raames loodud graafilise kasutajaliides juures sai tehtud muudatused, mis oli tingitud uue funktsionaalsuse lisamisest, milleks on sisestatud teksti parandamine.

Tarkvara testides tulid välja rakenduse kitsad kohad. Ajanappuse tõttu ei jõudnud parandused bakalaureusetöö esitamise tähtajaks valmis. Küllaga on plaanitud parandused, mis graafilise kasutajaliidesega välja pakutud on, rakendada pärast bakalaureusetöö valmimist.

[5] Java Runtime Environment

Summary

The topic of the bachelor's thesis is - Software for Morphosyntactic Cluster Extraction from Estonian Texts. The main purpose of this thesis is to develop software, that could extract word clusters using morphological, syntactic or morpho-syntactic tags.

This software is developed to further word order research. Linguists can take the output of this application and create new rules, which provide an opportunity to see language specifics and linguistic structures typical usage.

This thesis is further development of my seminar paper. The application requirements and graphical user interface design of the software have been done in the seminar paper. Software Architecture, design and testing have been done here.

Due to lack of time, some issues that arose from testing the application have not been fixed. Despite that, the application does what it is meant to do. The goal of this thesis has been met.

Kasutatud allikad

Abrahamsson, P., Ronkainen, J., Warsta, J., & Salo, O. (2002). *Agile software development methods.* Julkaisija - Utgivare.

Gopalaswamy, R., & Srinivasan, D. (2006). *Software Tesitng Principles and Practices.* Delhi: Dorling Kindersley.

Janardhanudu, G., & Wyk, K. (27. 07, 2009. a.). *White Box Testing*. Kasutamise kuupäev: 29. April 2012. a., allikas https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/white-box/259-BSI.html#dsy259-BSI_whatis

Kaalep, H. J., & Muischnek, K. (2009) Eesti keele püsiühendid arvutilingvistikas: Miks ja kuidas.

Metslang, H., & Matsak, E. (2010). Kesksete lausekomponentide järjestus õppijakeeles: arvutianalüüsi katse. Eesti Rakenduslingvistika Ühingu aastaraamat (175 - 193). Tallinn.

Müürisep, K. *Eesti keele süntaksianalüsaator*. Kasutamise kuupäev: 29. April 2012. a., allikas http://www.cs.ut.ee/~kaili/parser/

Müürisep, K. *Morfoloogilised märgendid*. Kasutamise kuupäev: 4. November 2011. a., allikas http://www.cs.ut.ee/~kaili/parser/demo/morftags.html

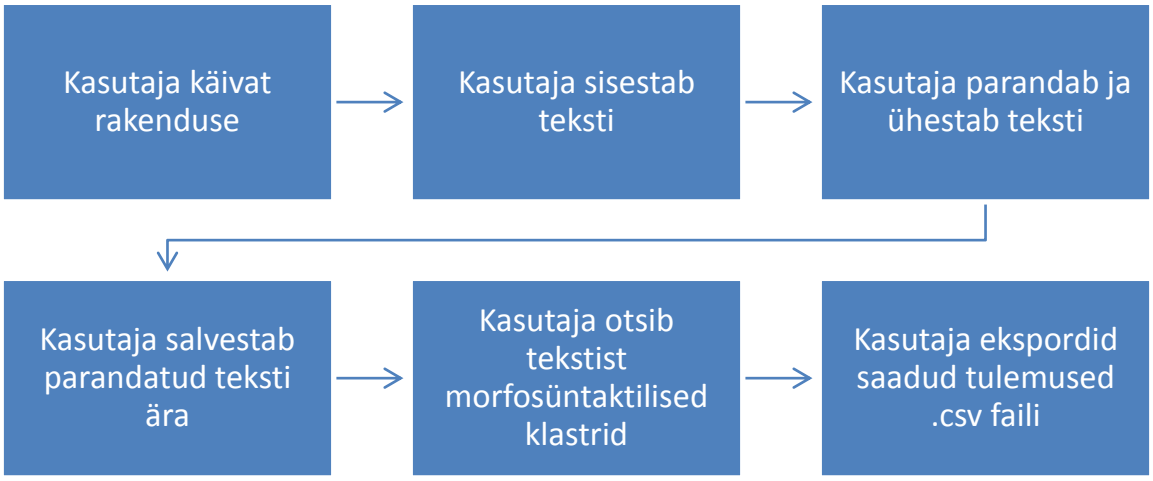
Müürisep, K. *süntaktilised märgendid*. Kasutamise kuupäev: 27. April 2012. a., allikas http://www.cs.ut.ee/~kaili/parser/demo/morftags.html

Myers, G. J., Badgett, T., & Sandler, C. (2012). *The Art of Software Testing.* New Jersey: John Wiley & Sons.

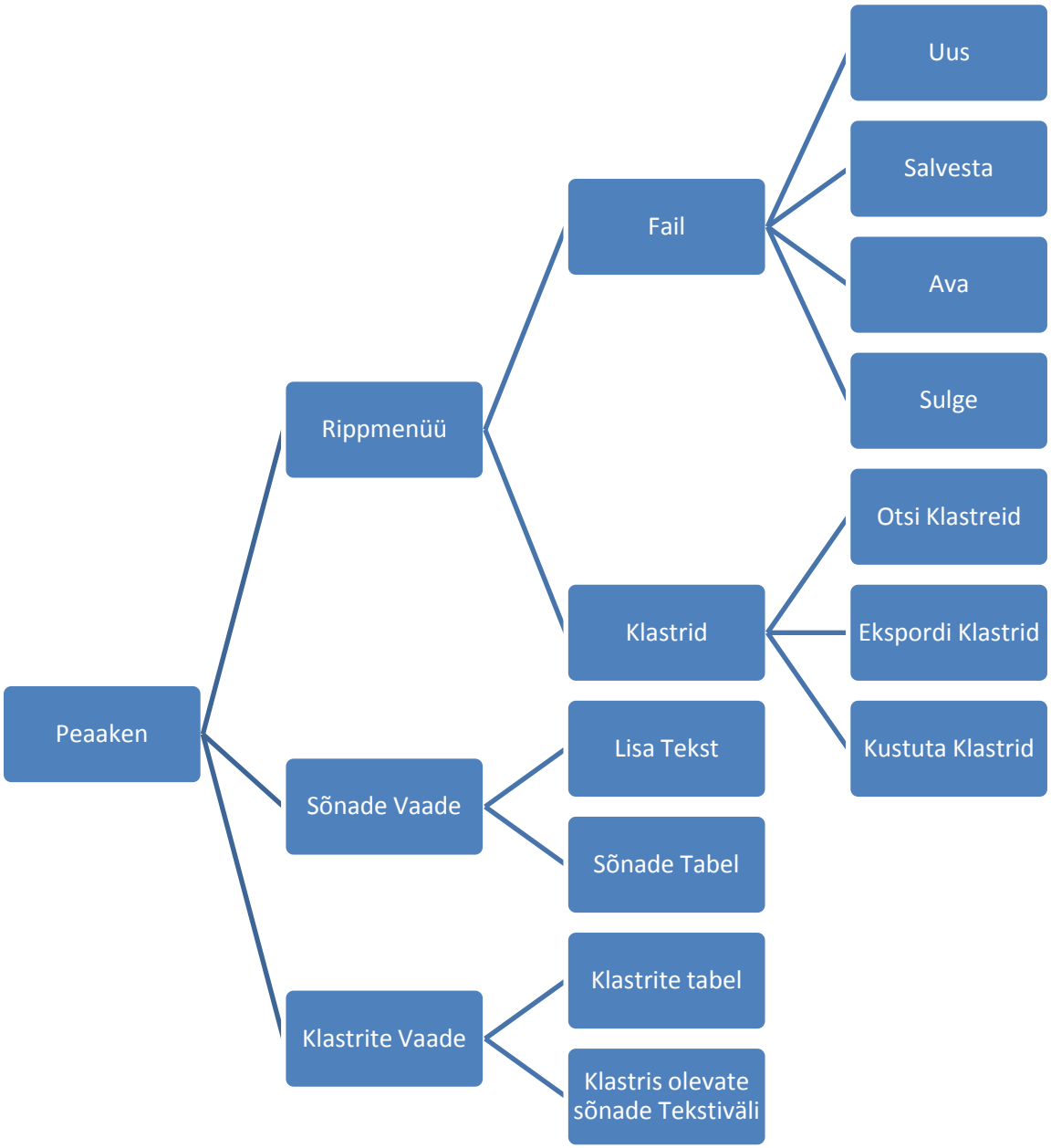
Smadja F. A & McKeown, K. R. (1990). Automatically extracting and representing collocations for language generation.

Zhu, H. (2005). *Software Design Methodology: From Principles to Architectural Styles.* Oxford: Butterworth-Heinemann.

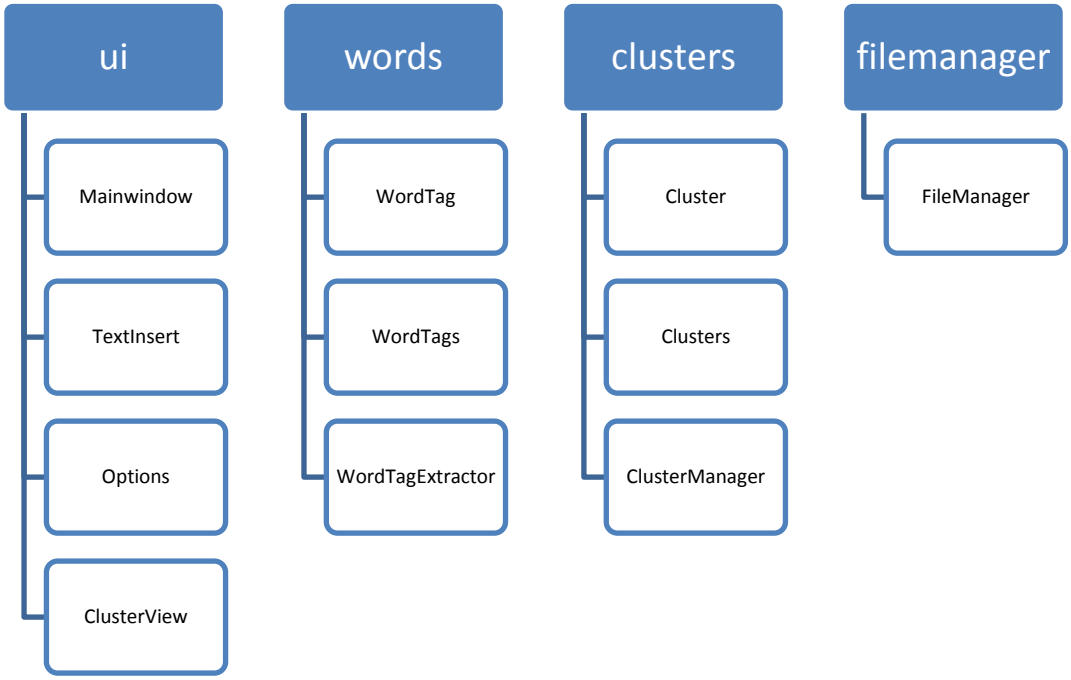
Lisad



Lisa 1 - Tarkvara kasutamise süžeetahvel



Lisa 2 - Tarkvara nõuetest tulenenud kasutajaliidese vooskeem



Lisa 3 - Pakettide ja klasside esialgne disain