Tallinna Ülikool
Informaatika Instituut

Eesti vahekeele korpuse vigade märgendusmooduli optimeerimine

Bakalaureusetöö

Autor: Harry Nõmmann
Juhendaja: Jaagup Kippar

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina _____ (sünnikuupäev: _____)
(*autori nimi*)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
_____
_____
_____
(*lõputöö pealkiri*)

mille juhendaja on _____,
(*juhendaja nimi*)

säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, _____
(*digitaalne*) *allkiri ja kuupäev*

Sisukord

Sissejuhatus

Bakalaureusetöö pealkirjast lähtuvalt on teemaks Eesti vahekeele korpuse vigade märgendusmooduli optimeerimine - teha see kasutajasõbralikumaks, mugavamaks ning tõsta lehe laadimiskiirust.

Olles juba seminaritöö läbi Eesti vahekeele korpusega kokku puutunud ja tutvunud, oli soov sellega edasi tegeleda ning puudustega kohti paremaks teha. Eesti vahekeele korpuses esineb mitmeid probleeme, millele keeleteadlased ja teised korpusega seotud inimesed lahendust soovivad. Valikut soosis ka tahe midagi praktilist teha ning seeläbi oma programmeerimisalaseid oskusi parendada. Keelekorpuste ja üleüldse keele uurimise aktuaalsus ning vajadus lahenduse järele andis veelgi enam kinnitust ja motivatsiooni.

Eelnevalt on Eesti vahekeele korpuse arendusega tegelenud mitu inimest. Teiste seas näiteks Jaagup Kippar, Siim Medijainen, Vahur Rebas.

Käesoleva töö vormist tulenevalt viisin uurimuse läbi esmalt püstitades koos spetsialistidega lahendamist vajavad probleemid ning seejärel uuringuga alustades. Eesmärgiks sai suurendada märgendusmooduli lehe laadimiskiirust ning muuta märgendusmooduli kasutaja jaoks mugavamaks ja intuitiivsemaks. Teemast johtuvalt on tegu arendusuuringuga.

Arendusuuringu tulemusena suurenes mitmekordselt mooduli lehe laadimiskiirus, liites kaks lehte parenes kasutusmugavus ning muutes märgendamisloogikat toimub märgendamine kiiremini, mugavamalt, lihtsamalt.

Uuringu tähtsus ja väärtus seisneb vajaduses parema vigade märgendusmooduli lahenduse järele.

Töötavat tulemust saab vaadata ja katsetada aadressil http://greeny.cs.tlu.ee:18186/korpus/korpus/Documents/doc_213943984937_item/Siim.html. Küll aga peab olema sisselogitud, et kasutada kõiki võimalusi. Sisselogimiseks võib kasutada kasutajanime „harry“ ja selle parooli „harry“.

1. Keelekorpused

Eesti keele seletava sõnaraamatu kohaselt on korpus teatud tunnuste järgi süstematiseeritud andmekogu veebis. Keeleteaduses on sõna korpus all tavaliselt mõeldud keeleainese kogumikku, mida kasutatakse uurimistöös materjalina. Korpustesse kuuluvad tekstid on valitud eesmärgipäraselt, nii, et nendest koosnev tervik annaks tõepärase pildi kogu keelest (Muischnek, kuupäev puudub).

1.1. Eesti vahekeele korpuse tutvustus

EVKK ehk Tallinna Ülikooli eesti vahekeele korpus (evkk.tlu.ee) on õppurite kirjalike tööde kogu, mis on koostatud ning elektrooniliselt töödeldud eelkõige uurimistöö eesmärgil. See on avatud ehk monitorkorpus, mis tähendab, et sellesse saab tekste pidevalt lisada.

EVKK on õppekeskkond eesti keele kui teise keele õpetajakoolituses ja täiendõppes; lingvistika magistrantide ja doktorantide jaoks on EVKK töökeskkond. EVKKs sisalduvat materjali saab kasutada teadusliku uurimistöö eesmärgil. Soovi korral võib korpuses luua oma töökeskkonna ning arendada korpust vastavalt kitsamale uurimisteemale, lisades uusi märgendusi ja täpsustades üldist veaklassifikatsiooni (EVKK, kuupäev puudub).

Korpust saab kasutada:

- Empiirilist ja rakenduslikku laadi uurimistöös
- Tulevaste õpetajate ja lingvistide koolitamisel
- Tegevõpetajate täiendõppes

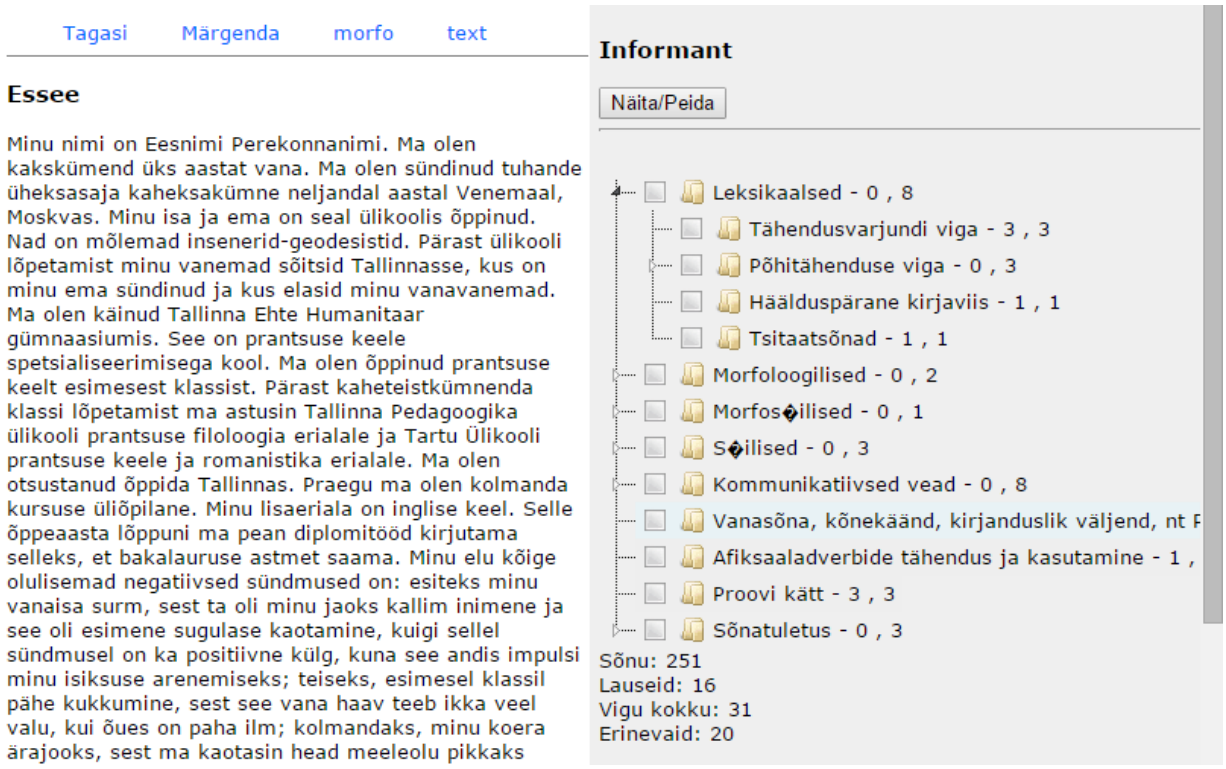
EVKKs on viga määratud kui grammatikareeglile mittevastav keelekasutus. Vigade hulka ei kuulu väsimusest, hooletusest, müradest kommunikatsioonikanalis jms põhjustel tekkinud eksimused ning keelevääratused.

EVKKd saab kasutada igaüks, kel on huvi õppijakeele vastu. Eriõigused antakse registreeritud kasutajale, andmehaldurile ning programmeerijale, aga korpuse funktsioone saavad kasutada kõik.

Praegune versioon on valminud Tallinna Ülikooli filoloogide, haridustehnoloogide ja informaatikute koostööna. Jälgida saab ka mingi sõna vigaseid esinemisnäiteid ning sama vealiiki ühes või kõigis korpuses olevates tekstides. On võimalik seostada infot keelevigade kohta õppuri sotsiaalse päritolu, staatuse, hariduse, soo, vanuse ja keele valdamise tasemega. (Eslon, 2014, 438)

2. Olemasolev märgendusmoodul

Lehe parempoolsemas osas on informandi sektsioon, kus on loetletud info teksti autori kohta, ning veapuu, kus on kõik tekstis märgendatud vealiigid (vt **Joonis 1.**). Iga vealiigi järel on kaks numbrit: vasakpoolne näitab vastava vealiigi esinemist tekstis, parempoolne selle vealiigi alamvigade esinemist. Kui teha märge tekstis oleva vealiigi märkeruutu, siis tõmmatakse tekstis joon alla kõigile selle vealiigiga märgendatud vigadele.



Joonis 1. Olemasolev märgenduste vaatamise leht.

Vajutades päises olevale lingile "Märgendus" avaneb märgendusmoodul (vt **Joonis 2.**), kus on võimalik vigu lisada ja kustutada. Lehe paremas osas on nimekiri tekstis esinevatest vealiikidest ning salvestamisnupp. Selekteerides mingi osa teksti, avaneb dialoogikast (vt **Lisa 1.**), kus saab valitud tekstiosale määrata vealiik või vealiigid. Tehes topeltklikk nimekirjas oleva vealiigi peale avaneb samuti dialoogiaken, aga nüüd on seal võimalik märgendust muuta.

Joonis 2. Olemasolev märgenduste lisamise leht.

2.1. Probleemid

- Vea näitamisel tekstis saab näidata vaid kõiki antud vealiigi alla kuuluvaid vigu. Eraldi ei saa.
- Vea kustutamiseks, lisamiseks või muutmiseks tuleb minna märgendusmoodulisse.
- Märgendusmoodulis tuleb vea baasi lisamiseks vajutada nuppu, mis viib tagasi eelmisesse moodulisse.
- Korraga saab efektiivselt näidata vaid ühte märgendust.
- Eksisteerib märgendusi, mida püüdes näidata hangub kogu moodul või mida lihtsalt ei näidata.
- Mõningate vigade puhul tekib olukord, kus seda tekstis küll näidatakse, kuid kui seejärel see tekstiosa selekteerida, siis seda veapuus enam ei näidata (linnukest pole).

3. Märgendamisprotsessi kirjeldus

Selles peatükis kirjeldan EVKK märgendamisprotsessi enne siinse bakalaureusetöö alustamist.

Märgendamine algab teksti märgendamiseks valimisega. Sellest hetkest alates, kui on alustatud teksti märgendamisega pole võimalik seda edaspidi enam muuta.

3.1. Märgendamine

Teksti märgendamiseks minnakse märgendusmoodulisse. Märgendusmoodulis on täpselt sama tekst, mis märgenduste vaatamise lehel, kuid funktsionaalsused on erinevad.

3.1.1 Lisamine

Märgendusmoodulis avaneb tekstiosa selekteerimise peale dialoogikast, kus on kõiki vigu sisaldav veapuu. Sellest veapuust otsib märgendaja vealiigi (võib ka mitu), mille soovib selekteeritud tekstiosale liita. Kui vealiigid on valitud, siis lisamiseks tuleb vajutada nuppu „Save and close“.

3.1.2 Kustutamine

Kustutamiseks tehakse topeltklikk listis oleva vealiigi peale, mille järel avaneb dialoogikast, kus saab keskkonna kõigi vigade seast välja otsida soovitud viga ning selle eest linnuke eemaldada. Seejärel lisatakse antud veale "DELETEME" ning hiljem kustutatakse funktsiooniga "deleteError", mis vajab sisendiks vaid antud vea id-d. Igale veale määratakse automaatselt selle lisamisprotsessi käigus unikaalne id.

Viimane liigutus märgendusmoodulis on vigade lõplik lisamine - vajutatakse "Salvesta" nuppu, misjärel käivitub funktsioon **saveMarksNG**, mis vastavalt tehtud muudatustele (kustuta, lisa) muudab ka andmebaasi kirjeid ning Zope failisüsteemis olevaid andmeid vigade kohta.

3.2. Märgenduste vaatamine

Seminaritöös lisasin märgenduste vaatamise lehele veapuu, mis grupeerib tekstile lisatud vealiigid ning laseb neid märkida. Märke tegemise peale tõmmatakse tekstis joon alla kõigile märgitud märgenditele vastavatele tekstiosadele.

Puu loomisel otsustasin kasutada jsTreed. JsTree on avatud lähtekoodiga, tasuta jQuery plugin, mis võimaldab luua interaktiivseid puusid. Seda on lihtne konfigureerida, kasutada ning teatud osasid omale meelepärasemaks muuta. JsTreel on alternatiividega võrreldes suurem kogukond ja parem dokumentatsioon. Täpsemalt olen kirjeldanud jsTreed ja selle kasuks otsustamist oma seminaritöös “Vigade märgendus Eesti vahekeele korpuse tekstides”.

4. Muudatuste kava

Selles peatükis kirjeldan muudatusi ja täiendusi, mida oleks vaja töö käigus teha ning tehnoloogiaid, mida selle jaoks kasutan.

4.1 Vajaminevad muudatused ja täiendused

Vajaminevad muudatused ja täiendused selgusid arutelu tulemusena koos spetsialistide Jaagup Kippari ja Pille Esloniga.

Hetkel saab vigu eraldi näidata vaid juhul, kui konkreetset viga on tekstis märgitud ainult ühe korra. Enamikel juhtudel esineb tekstis üht vealiiki korduvalt. Seetõttu sai otsustatud, et veapuule tuleks juurde lisada viimane tase, mille abil saaks näidata tekstis esinevaid vigu individuaalselt.

Hetkel laeb märgendusmoodul veebilehe kohta üpris kaua ning funktsionaalsuste juurde lisamisega võib oodata kiiruse veelgi enamat langemist.

Pikema teksti ja väiksema ekraani puhul võib juhtuda olukord, kus teksti peab kerima ning soovitud vea näitamisel on vea asukoht ekraanist "väljas" ning õigesse kohta kerimisel pole enam kogu veapuud näha. Kasutamismugavus suureneks, kui vea märkimisel oleks koheselt selle asukoht tekstis näha ning veapuu oleks alati nähtaval. Seega peab märke tegemisega tekst liikuma nii, et vea asukoht tuleks kasutajale nähtavale ning lehekülge või teksti kerides peaks veapuu jääma nähtavale.

Hetkel tuleb vigade lisamiseks, muutmiseks või nende eemaldamiseks vigade vaatamise lehelt minna märgendamise lehele, mis on tülikas, tüütu, ebamugav. Oleks parem, kui seda kõike saaks teha ühes kohas.

Kui muudatus on tehtud, siis tuleb vajutada "Salvesta" nuppu, mis saadab muudatused ka andmebaasi. Hetkel on see tegevus ebamugav, kuna peale nupu vajutust viiakse kasutaja tagasi mooduli esilehele. Kasutajal peaks olema võimalus jätkata tööga peale igat lisamist, muutmist ja kustutamist.

4.2 Kasutatavad tehnoloogiad

- JavaScript - on Netscape'i poolt välja töötatud skriptikeel, mis võimaldab veebiautoritel luua interaktiivseid veebisaite (Vallaste, kp puudub). Märgendusmoodulit arendas enne mind Siim Medijainen, kes kasutas selle loomiseks suuresti just JavaScripti. Käesoleva uurimuse käigus teen mitmeid muudatusi tema poolt tehtule ning lisan juurde.
- Python - Interpreteeritavas objektorienteeritud programmikeeles Python on kirjutatud palju funktsioone, millele hakkan ise juurde lisama ning milledest mõnda muutma. Kõige enam hakkan tegelema failiga Document.py.
- ZPT - Pythoni pakett, mis teostab lehe malle nagu Zope, aga töötab Zopest väljaspool (ZPT, kp puudub).
- jQuery - Kompaktne ja paljude võimalustega JavaScripti teek, mis lihtsustab JavaScriptis programmeerimist. Varem on märgendusmoodulis kasutatud teeki MooTools. Otsustasin kasutada ka jQueryt selle mitmete programmeerimist lihtsustavate võimaluste tõttu. JQueryt kasutan näiteks Dialoogikasti, veapuu loomisel ning nendest andmete kätte saamisel.
- jQuery UI - jQuery teegile ehitatud kasutajaliidese interaktsioonide, vidinate, efektide ja mallide kogum (jQuery UI, kp puudub). JQuery UI-d hakkan kasutama dialoogikasti loomisel.
- jsTree - JsTree on avatud lähtekoodiga, tasuta jQuery plugin, mis võimaldab luua interaktiivseid puid.

5. Märgendusmoodulis tehtud muudatuste kirjeldus

Selles peatükis kirjeldan uuringu käigus tehtud muudatusi. Arendus toimus testserveris, suuremas osas kasutades skriptikeelt JavaScript ja programmikeelt Python. Vähemal määral kasutasin ZPTd, HTMLi ning CSSi.

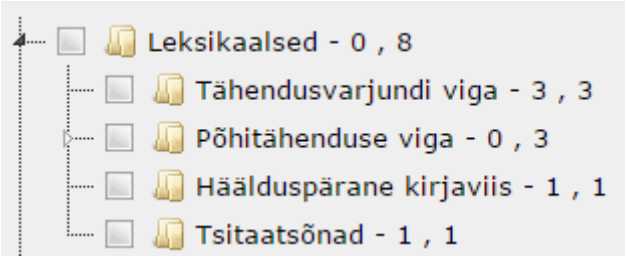
5.1. Veapuule viimase alamtaseme lisamine

Informaatikas nimetatakse puuks sellist andmestruktuuri, kus iga element on ühendatud ühe või mitme temast allpool asuva elemendiga (Vallaste, kp puudub).
Puus enamesinevad elemendid:

- juur - kõige ülemine element puus
- vanem - mingi elemendi otsene ülemelement
- laps - mingi elemendi otsene alamelement
- leht - lasteta element

Hetkel loob veapuu makro **codesTreePrinterDocument**, mis käib läbi kogu vigade massiivi käsuga `tal:repeat="code cont/getCodes"`. Sealjuures kasutatakse kahte **Document.py** (failis tehtud muudatusi vaata kolmandast lisast) failis asuvat funktsiooni **countUsedCodeTree** ja **countUsedCode** - esimene annab teada kui palju on seda viga või selle vea alamvigu tekstis, teine kui palju on viga ennast tekstis. Rakendatakse tingimus `tal:condition="python:context.countUsedCodeTree(code, request)"`, mis jätab välja need vealiigid, mis antud teksti puhul annavad funktsiooni **countUsedCodeTree** tulemuseks ühest väiksema numbri, mis tähendaks, et tekst ei sisalda antud vealiiki või selle mõnda alamvealiiki. Ühest väiksema korral võetakse kõikide vigade massiivist järgmine viga ja korratakse. Ühe või suurema puhul lisatakse viga andmepuusse ning lisatakse atribuudid, mille järel võetakse samuti massiivist ette järgmine viga ning korratakse kogu protsessi.

Väga sageli tuleb ette, et tekstis esineb mitu ühe ja sama vealiigiga märgendit (vt Joonis 3.). Sellistel puhkudel oleks hea, kui saaks näha või valida vaid ühte veakohta. Juba kasvõi kustutamisele ette mõeldes ei saaks seda teha, sest kustutatakse sellisel juhul kõik vastavad vead.



Joonis 3. Olukord, kus tekstis esineb ühte vealiiki rohkem kui üks kord.

Üheks lahenduseks on puu nendele lehtedele, mille vealiiki esineb tekstis enam kui ühel korral, laste loomine, ehk alamtaseme loomine, kus iga element on puu leht ja üksik viga.
Esiteks on vaja kätte saada vastava vealiigi tekstis esinevate vigade andmed - unikaalne vea identifikaator ning täpne asukoht tekstis XPath kujul. Selleks lõin funktsiooni nimega **allDiffUsedCodes** (vt Koodinäide 1.), mis saab sisendiks vealiigi identifikaatori ning väljastab antud tekstis oleva vealiigi kõik vead eraldi.

```
def allDiffUsedCodes(self, code_name,  REQUEST):
    codes=self.Errors.getDocumentMarks(self.getId(), str(REQUEST.AUTHENTICATED_USER))
    res = []
    append=res.append
    prettyCode=self.Marks.prettyCodeTitle
    counter = 1
    for code in codes:
        name = code.getProperty('code').encode('utf-8')
        if code_name == name:
            xpointer = code.getProperty('pointer').encode('utf-8')
            append([name, xpointer, counter, code.getId()])
            counter+=1
    return res
```

Koodinäide 1. Tekstis esineva vealiigi erinevate märgendite saamine

Kasutades loodud funktsiooni, sain muuta olemasolevat veapuu makrot nii, et veapuule lisanduks vastavatesse kohtadesse uus tase. Lisasin tingimuse (vt Koodinäide 2.), mis jätkab vaid nende vealiikide puhul mida esineb enam kui ühe korra - kui üht vealiiki esineb tekstis vaid ühel korral, siis ta juba ongi veapuus eraldi, mistõttu pole alamtaseme lisamine vajalik. Küll aga on vaja anda sellistele elementidele samasugused atribuudid nagu alamtaseme omadele - mõlemad on veapuus lehed.

```
tal:condition="python:context.countUsedCodeTree(code,request))>1 and
context.countUsedCode(code.getCodeId(), request))>1"
```

Koodinäide 2. Vealiikide arvu kontrolliv tingimus.

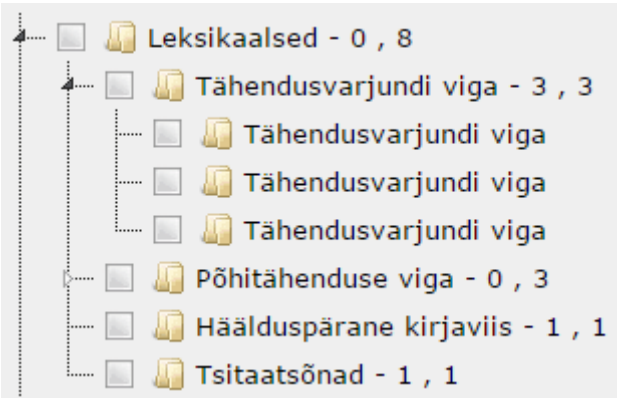
Selleks, et välja selgitada, kas hetkel makroga seotud element on leht (antud olukorras selline vealiik, mida esineb tekstis vaid ühe korra ning millel kas pole alamliike või neid ei esine

tekstis), otsustasin kirjutada funktsiooni **countUsedOnlyOne** (vt Koodinäide 3.), mis lehe korral annab vajaminevad atribuudid, teiste elementide puhul aga nullid.

```
def countUsedOnlyOne(self, codeobj, REQUEST):
  codes = self.Errors.getDocumentMarks(self.getId(), str(REQUEST.AUTHENTICATED_USER))
  res = []
  extend=res.extend
  codeobjID=codeobj.getCodeId()
  if((self.countUsedCodeTree(codeobj, REQUEST)==1) and (self.countUsedCode(codeobjID,
REQUEST)==1)):
    extend((self.allDiffUsedCodes(codeobjID,
REQUEST)[0][1],self.allDiffUsedCodes(codeobjID, REQUEST)[0][3],1))
  else:
    extend((0,0,0))
  return res
```

Koodinäide 3. Funktsioon, mis kontrollib kas element on leht või mitte

Tulemuseks on veapuu, kus vastavates kohtades on nüüd alamtase (vt Joonis 4.).



Joonis 4. Veapuus on nüüd alamtase.

Selleks, et uue alamtaseme elemendid ka eeldustekohaselt käituksid st, et ühe elemendi märkimisel näidataks tekstis vaid ühte viga, tuli teha mõned muudatused olemasolevates funktsioonides failis **document_viewSiim.js** (kõiki failis tehtud muudatusi vaata viiendast lisast). Märkeruudu märkimisel kontrollitakse elemendi atribuudi **leafnode** järgi, kas vastav element on leht ning kui on, siis loetakse ka elemendi vea asukoht tekstis (XPoint). Seejärel käivitub äsjaloetud sisenditega funktsioon **updateMarkupSchema**.
Kogu täiendatud veapuu loomise makrot vaata lisast 2.

5.2. Laadimiskiiruse tõstmine

Juba alguses, enne veapuu muutmist, oli lehe laadimiskiirus ebanormaalselt madal ning peale muutmist alanes see veelgi. Ilmnes, et mingi tegevus võtab väga palju aega. Kuna raske oli

öelda, kas aeglustaja asub serveripooles või kliendipooles, siis paigaldasin probleemikoha leidmiseks Chrome veebilehitsejale laienduse "Page load time". Selgus, et suur enamus ajast kulub serveripooles (vt Joonis 5.).

Load timings (ms)

Event	When	How long	Sum
Redirect	0	0	0
DNS	0	0	0
Connect	0	0	0
Request	3	15202	15202
Response	15205	406	15608
DOM	15223	1257	16865
Interactive	16437	0	-
Content loaded	16437	18	-
Load event	16480	68	16933

Joonis 5. Lehe laadimiskiirus millisekundites.

Arvasin ekslikult esialgu, et asi on veapuud genereerivas makros, kuid edasi uurides ja testides tuli välja, et asi on selles, et mitmed funktsioonid **Document.py** failis teevad igal laadimisel päringu andmebaasi, et saada vigade massiiv, millega seejärel edasi tegeletakse. Kuna sesssioonist lugemine on kiirem kui andmebaasist pärimine, siis otsustasin seda kasutada. Andmebaasi tehakse nüüd päring vaid esimesel korral ning tulemus salvestatakse sessiooni, kust seda igal järgmisel korral loetakse (vt Koodinäide 4.).

```
codes=self.REQUEST.SESSION.get('sessioncodes', "")
if len(codes)<1:
  codes = self.Errors.getDocumentMarks(self.getId(), str(REQUEST.AUTHENTICATED_USER))
  self.REQUEST.SESSION.set('sessioncodes', codes)
```

Koodinäide 4. Sessiooni kasutamine Document.py failis

Sessiooni kasutusele võtmisega vigade andmete salvestamisel suurenes lehe laadimiskiirus umbes neli korda (vt Joonis 6.).

Load timings (ms)

Event	When	How long	Sum
Redirect	0	0	0
DNS	0	0	0
Connect	0	0	0
Request	4	2678	2678
Response	2682	267	2945
DOM	2689	980	3925
Interactive	3616	0	-
Content loaded	3616	19	-
Load event	3669	70	3995

Joonis 6. Lehe laadimiskiirus millisekundites peale vigade andmete salvestamist sessiooni.

5.3. Märgenduste lisamine ning jQuery UI kasutamine

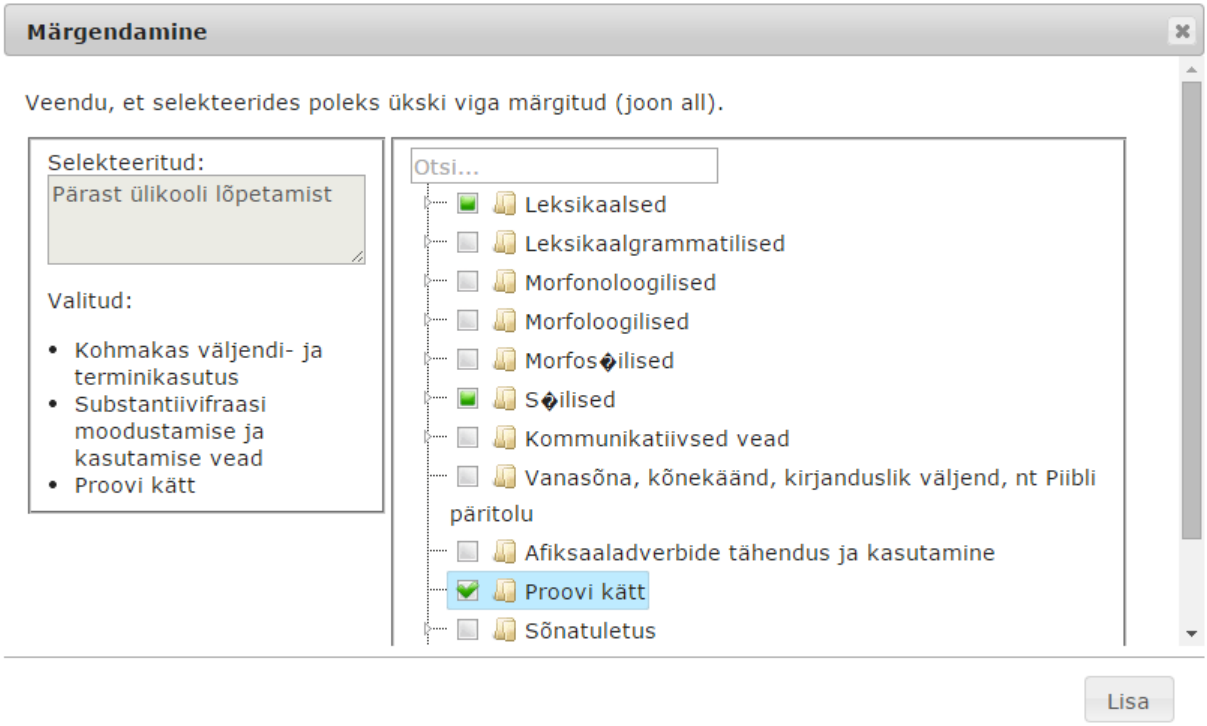
Märgenduse lisamiseks tuleb kasutada **Errors.py** failis asuvat funktsiooni **addNewError**, mis võtab seitse sisendit:

1. pointer - vea täpne asukoht tekstis
2. content - tekstiosa, millele soovitakse viga lisada
3. document - dokumendi ehk teksti number
4. code - vealiik
5. author - vea märkija (kasutaja)nimi
6. pre - märgendatavale tekstiosale eelnev tekstiosa
7. post - märgendatavale tekstiosale järgnev tekstiosa

Vea täpse asukoha saamiseks kasutatakse XPathi. Asukoht koosneb vea esimese ja viimase sümboli asukohtadest. http://www.w3schools.com/XPath.

Uue märgenduse lisamisprotsessi otsustasin kasutaja jaoks sarnaseks jätta - tuleb selekteerida tekstiosa, mida soovitakse märgendada, mille järel avaneb dialoogikast (vt Joonis 7.), kus on kõiki vealiike sisaldav veapuu. Vajutades nuppu "Lisa", lisatakse selekteeritud tekstiosale kõik veapuus märgitud vealiigid: esmalt pannakse märgitud vealiigid kokku massiivi ning seejärel liidetakse kõik massiivi elemendid kokku üheks sõneks (elemendid eraldatakse kaldkriipsuga). Valminud sõne, selekteeritud tekstiosa ning selle täpne asukoht tekstis ja veale eelnev ning järgnev tekstiosa saadetakse seejärel Ajaxi kaudu serveripoolsele funktsioonile **prooviAdd**.

Kasutades saadud andmeid, sisestatakse ükshaaval vead andmebaasi, ainus erinevus igal sisestamisel on vealiik.



Joonis 7. Selekteerimisel avanev dialoogikast, mille kaudu saab vigu lisada ja muuta.

Selekteerimisel avaneva dialoogikasti loomiseks otsustasin kasutada jQuery UI vidinat "Dialog". Valiku tegemisel lähtusin asjaoludest, et märgendusmoodulis on jQuery juba kasutusel, jQuery on äärmiselt laialt kasutust leidev teek ning olen sellega ise varem palju kokku puutunud.

Selleks, et jQuery UI-d kasutada, tuleb esmalt lisada veebilehe päisesse kolm viidet, mis võimaldavad juurdepääsu jQuery UI funktsioonidele (vt Koodinäidet 5.).

```
<skript
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<link   rel="stylesheet"   href="//code.jquery.com/ui/1.11.4/themes/smoothness/jquery-ui.css">
<script src="//code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
```

Koodinäide 5. JQuery UI kasutamiseks vajaminevad viited.

Selleks, et seda rakendada, peab `<div>` märgendil olema id. Näiteks `<div id="proov">` korral saab skriptiosas dialoogikasti vidina rakendada käsuga `$( "#proov" ).dialog();`. Interaktiivsed näidised leiab aadressilt https://jqueryui.com/dialog/.

5.4. Veapuu täiendamine kasutades jsTree võimalusi

Seminaritöös kasutasin jsTreed ning selle märkeruutude laiendit. Bakalaureusetöös võtan kasutusele ka laiendid otsing ja märgenduste lisamiseks ning kustutamiseks hüpikmenüü.

5.4.1 Vealiikide otsing

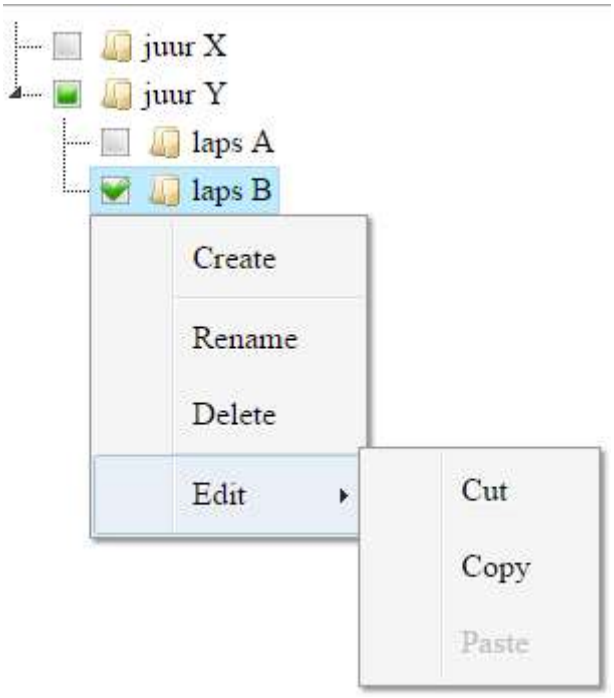
Veapuu kiiremaks kasutamiseks lisasin otsingulahtri. JsTreel on kümme laiendust - teiste seas märkeruudud, mis on juba kasutusel, otsing ja kohandatav menüü.
Otsingu laiendi kasutamiseks, tuleb laiendite loetellu lisada "search". JsTree kodulehelt võtsin õpetuse järgi funktsiooni, mis teostab otsingu (vt Koodinäide 6.).

```
var to = false;
  jQuery("#puuvaade0_q").bind("keyup change", function(e) {
    if(to) { clearTimeout(to); }
    to = setTimeout(function () {
      var v = jQuery('#puuvaade0_q').val();
      jQuery('#puuvaade0').jstree('search', v);
    }, 250);
  });
```

Koodinäide 6. Otsingut teostav funktsioon.

5.4.2 Märgenduste muutmine ja kustutamine

Järgmisena lisasin jsTree hüpikmenüü laiendi. Selleks lisasin laiendite loendisse "contextmenu". Hüpikmenüü avaneb, kui teha paremklikk puu elemendi peal. Vaikimisi on menüüs võimalused "Create", millega saab puusse uue elemendi lisada, "Rename" ja "Delete" vastavalt nime muutmiseks ja kustutamiseks ning "Edit", mille alt on võimalik teostada toiminguid lõika, kopeeri, kleebi (vt Joonis 8.).



Joonis 8. JsTree vaikimisi seadetega hüpikmenüü näide.

Olemasolevate märgendite muutmise ja kustutamise vajadusest lähtuvalt kirjutasin kohandatud menüü, kus on kaks valikut: "Muuda" ja "Kustuta".

Kustutamine

Andmebaasist märgenduse kustutamiseks on vaja teada selle unikaalset identifikaatorit. Täiendasin makrot nii, et igale veapuu lehele lisatakse atribuut "errorid", mis ongi seesama unikaalne identifikaator.

Märgenduse kustutab **Errors.py** failis asuv funktsioon **deleteError**. Kirjutasin **Document.py** faili funktsiooni **prooviDelete**, mis saab kliendi poole pealt, kui kasutaja vajutab “Kustuta”, Ajaxiga saadetud vea identifikaatori kätte ning käivitab seejärel **deleteError** funktsiooni, misjärel märgendus kustub. Kui andmebaasist kustutamine õnnestub, siis käivitub koodiosa, mis kustutab vastava vea ka veapuust ära (vt Koodinäide 7.).

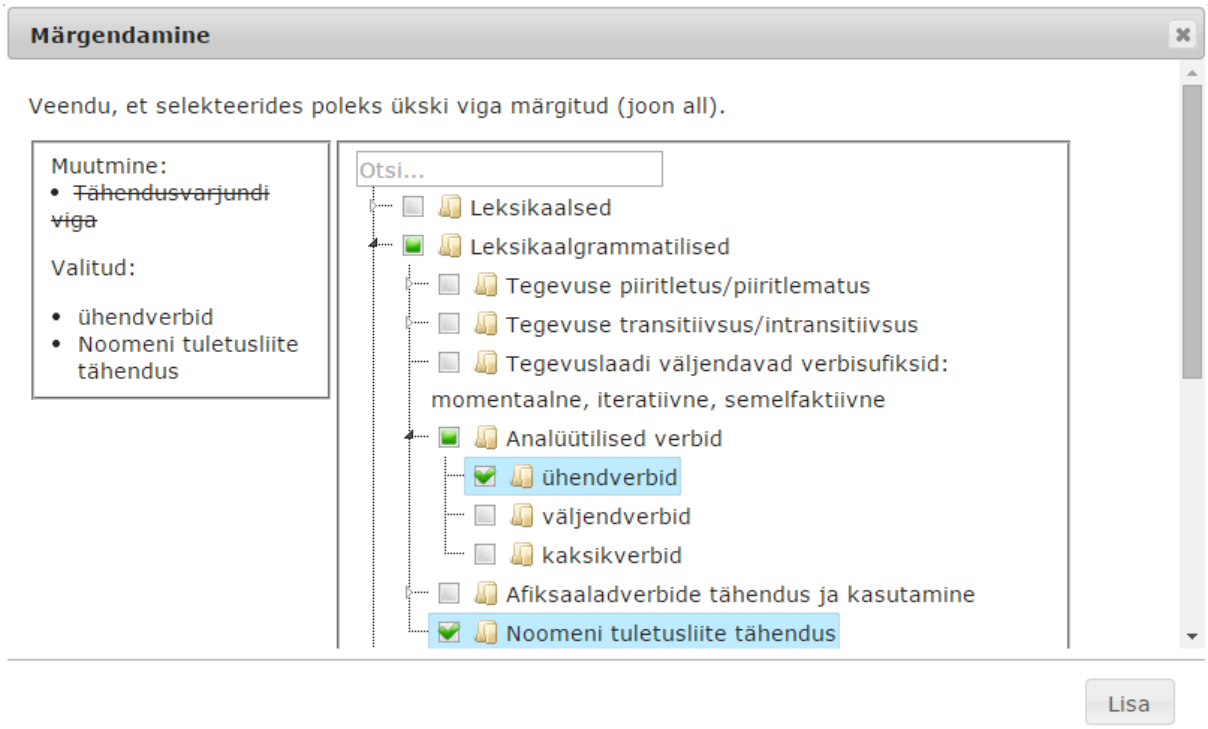
```
var inst = jQuery.jstree.reference(data.reference),
obj = inst.get_node(data.reference);
if(inst.is_selected(obj)) {
  inst.delete_node(inst.get_selected());
} else { inst.delete_node(obj); }
```

Koodinäide 7. Elemendi kustutamine kliendi poolel veapuust.

Kasutades JavaScripti **confirm()** meetodit, lisasin kontrolli, mis küsib kasutaja käest üle, kas ollakse kindel, et soovitakse kustutada. Lisasin tingimuse, mis kontrollib, et kustutatakse vaid lehti.

Muutmine

Kui vajutada menüüs "Muuda", siis avaneb eelpool mainitud dialoogikast, mille mõned väljad eelnevalt ära muudetakse: ei näidata selekteeritud teksti, näidatakse hetkel olevat vealiiki ning tekib loetelu uutest vealiikidest, mille kasutaja veapuust valib (vt Joonis 8.). Muutmise tehniline pool on üldiselt kustutamise ja lisamise summa st muutes mingit konkreetset viga tekstis see esialgu kustutatakse ning siis luuakse sama asukohaga (asukoht tekstis) uus või uued vead.



Joonis 8. Ühe tekstis oleva vea muutmine (antud juhul asendatakse üks vealiik kahe teisega).

5.5. Valitud vea esiletoomine

Teise probleemi lahendamiseks kasutasin JavaScripti "scrollIntoView" meetodit, mis liigutab lehte selliselt, et etteantud element tuleb nähtavale. On mainitud, et teatud situatsioonides pole see meetod soovitatav lahendus, kuna tegevus toimub momentaanselt, mistõttu võib kasutajale segaseks jääda mis toimus: kas mindi uuele lehele, kas keriti üles, kas keriti alla. Sellest vaatepunktist lähtudes peaks tegevus toimuma sujuvamalt, et kasutaja näeks, et keritakse üles

või alla (Koritnik, 2012). Arvan, et praeguses olukorras on selle meetodi kasutamine siiski õigustatud, kuna mooduli kujundusest ja elementidest tingituna ei saa kasutajal tekkida kahtlus kas ta on viidud mõnele teisele lehele. Kuna keritakse teksti, siis ei teki tõenäoliselt segadust ka arusaamisega kas keriti üles või alla.

6. Edasiarendus

Märgendusmoodul pole veel täiesti valmis ning edasiarendusvõimalusi jätkub. Selles peatükis nimetan toon välja kaks asja, mille ellu viimist võiks selle koha pealt jätkaja kaaluda.

Esiteks teeks lisamise, kustutamise ja muutmise veelgi mugavamaks võimalus teksti selekteerimisel avanevas dialoogikastis näha selekteeritud tekstiosas juba olemasolevaid vealiike ning neid ka veapuus märgituna näidata. Ning sõltuvalt sellest, kas mõni vealiik on juurde märgitud või mõne eest märge ära võetud, teha lisamised ja kustutamised.

Teiseks, märgendamist mugavdaks ka see, kui oleks võimalik kõik tekstis olevad vead nähtavale tuua ning märgendatud tekstiosadele hiir viies avaneks spikker (*tooltip*), mis näitaks millised vealiigid sellel tekstiosal juba olemas on. Mingil määral kõigi vigade näitamine juba õnnestus, kuid teatud probleemidega. Nimelt mõnede tekstide puhul esineb märgendusi, mille märkimisel/selekteerimisel kogu moodul hangub. Hangumine on seotud teksti korpusesse kopeerimisel kaastulnud sobimatutest kujunduskäsklustest. Kui nimetatud probleemile lahendus leida ning igale märgendusele sellele vastavad vealiigid spikrina kuvama panna, mugavneks märgendamine veel enamgi.

Kokkuvõte

Esimeses peatükis tutvustasin korpuste kasutamist keeleteaduses üldiselt ning Eesti vahekeele korpust.

Bakalaureusetöö püstitatud eesmärgid said täidetud. Suutsin märgendamismooduli laadimiskiirust suurendada ligikaudu neli korda, mis parandab tunduvalt mooduli kasutamist. Sain hakkama ka kahe lehe - märgendamismooduli lehe ning märgenduste vaatamise lehe - kokkuliitmisega, mis tähendab, et enam ei pea keeleteadlased tekstide märgendamisel liikuma nende kahe lehe vahel, vaid saavad kõike ühes kohas teha. Parandatud sai ka märgendusmooduli kasutatavust väiksemate ekraanide või suuremate tekstide puhul, kui teksti kerimisel võis veapuu nö ekraanist välja minna. Kaotasin ka salvestamise nupu, mis varem viis märgendaja märgendusmoodulist välja. Nüüd toimub tehtud märgenduste lisamine, muutmine, kustutamine koheselt.

Uuringu käigus sain uusi oskusi JavaScriptis ja Pythonis ning veel parema arusaama, kuidas EVKK erinevad osad omavahel töötavad. Sain kasuliku kogemuse veebipõhise rakenduse arendamisest.

Seitsmendas peatükis tõin välja kaks viisi, kuidas saaks märgendusmoodulit veelgi mugavamaks teha. Muudatused olid esialgu plaanis ka läbi viia, kuid ajanappuse tõttu jäid need minu poolt vaid ideedeks.

Summary

The first chapter provides an overview of corpuses in linguistics and more specifically of Estonian Interlanguage Corpus.

The goals established in the fourth chapter were completed. Error Marking Module's loading speed was improved by four times which improves the module's usage considerably. By merging two pages: Error Marking Module's page and Markings view page it isn't necessary anymore to switch between these two pages while marking a text. User interface was improved for smaller screens and/or for larger texts where some parts of the module might "go out of screen". The save button was removed because it lost its value and purpose because saving was made automatic - every marking that is made is instantly saved and stored in the database.

During research I acquired new skills in JavaScript and Python and an even better understanding of how different parts of Estonian Interlanguage Corpus work together. I got a valuable experience of developing a web-based application.

In the seventh chapter two improvements for further development were brought out that could enhance user experience even further. These improvements were at first planned to be completed as well but later dismissed due to lack of time.

Kasutatud allikad

Eslon, P. (2014, juuni). *Eesti vahekeele korpus*. Keel ja kirjandus 6, 438. Tallinn, Eesti: SA Kultuurileht.

Medijainen, S. (2011). *Eesti vahekeele korpuse tekstide märgendusmooduli arendamine* (diplomitöö). Haapsalu, Eesti: Tallinna Ülikooli Haapsalu kolledž.

Muischnek, K. (kuupäev puudub). *Korpuslingvistika kursus: 1*. Loetud 4. 05. 2015 aadressilt http://www.cl.ut.ee/kursused/korp_ling01.

Vallaste, H. (2000). *e-Teatmik*. Loetud 4. 05. 2015 aadressilt http://www.vallaste.ee/.

jQuery UI. (kuupäev puudub). *jQuery UI*. Loetud 4. 05. 2015 aadressilt http://jqueryui.com/.

Sourceforge. (kuupäev puudub). *Zope Page Template*. Loetud 4. 05. 2015 aadressilt http://zpt.sourceforge.net/.

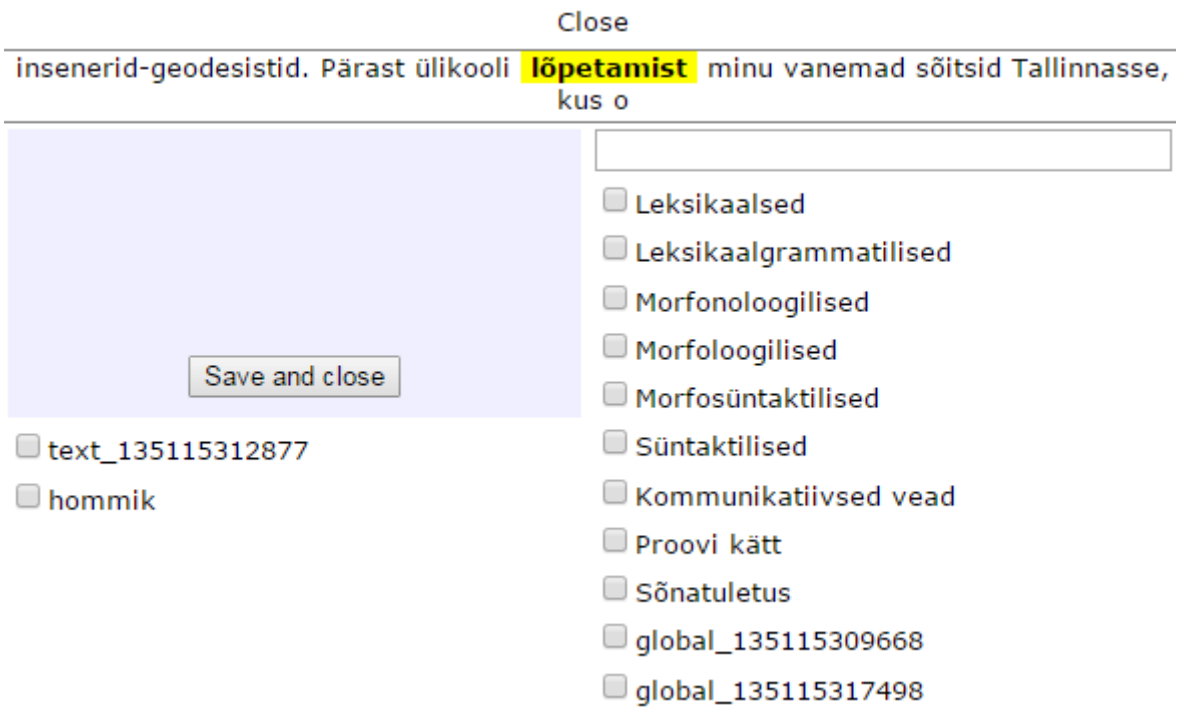
EVKK. (kuupäev puudub). *Eesti vahekeele korpus*. Loetud 4. 05. 2015 aadressilt http://evkk.tlu.ee/.

Nõmmann, H. (2014). *Vigade märgendus Eesti vahekeele korpuse tekstides* (seminaritöö). Tallinn, Eesti: Tallinna Ülikool.

Koritnik, R. (2012). *Does scrollIntoView work in all browsers?* Loetud 4. 05. 2015 aadressilt http://stackoverflow.com/questions/9445842/does-scrollintoview-work-in-all-browsers

Lisad

Lisa 1. Vana märgendusmooduli dialoogikast



Lisa 2. Macros.pt faili täiendused

```
<ul metal:define-macro="codesTreePrinterDocument">
             <span tal:omit-tag="" tal:repeat="code cont/getCodes">
                    <span tal:omit-tag="" tal:define="dele code/isDeleted; dUsed
python:context.diffUsedCodes(code, request); onlyOne
python:context.countUsedOnlyOne(code, request)">

                                  <li style="" tal:define="cont code;round
python:round+1" tal:condition="python: context.countUsedCodeTree(code, request)">
                                        <a href="markdown.html" tal:content="python:
code.getTitle()+' - '+str(context.countUsedCode(code.getCodeId(), request))

                                                                    +' ,
'+str(context.countUsedCodeTree(code, request))"
                                              tal:attributes="
                style python:view.test(dele, 'font-style:italic' );
                href string:${code/absolute_url}/markdown.html;
                name python:code.getCodeId();
                id python:code.getCodeId();
                docolor python:dUsed[2][2];
                xpoint python:onlyOne[0];
                errorid python:onlyOne[1];
                leafnode python:onlyOne[2]
             ">
                                        </a>

                                        <span metal:use-
macro="container/macros/codesTreePrinterDocument"></span>

                                        <ul>
                                          <li style="" tal:define="cont code;round
python:round+1;ADUCodes python:context.allDiffUsedCodes(code.getCodeId(), request)"
tal:repeat="nr python: range(len(ADUCodes))" tal:condition="python:
(context.countUsedCodeTree(code, request))>1 and
(context.countUsedCode(code.getCodeId(), request))>1">
                                                <a
tal:content="python:code.getTitle()" tal:attributes="name python:ADUCodes[nr][0];
docolor python:dUsed[2][2]; xpoint python:ADUCodes[nr][1]; errorid
python:ADUCodes[nr][3]" leafnode=1 ></a>
                                          </li>
                                        </ul>
                                  </li>
                    </span>
             </span></ul>
```

Tekstis esinevate märgenduste puud genereeriv makro.

```
<ul metal:define-macro="codesTreePrinterDocumentAll">
             <span tal:omit-tag="" tal:repeat="code cont/getCodes">
                    <span tal:omit-tag="" tal:define="dele code/isDeleted">

                                  <li tal:define="cont code;round python:round+1">
                                        <a tal:content="python: code.getTitle()"
               tal:attributes="
                id python:code.getCodeId();
                style python:view.test(dele, 'font-style:italic' )"></a>

                                        <span metal:use-
macro="container/macros/codesTreePrinterDocumentAll"></span>
                                  </li>
                    </span>
             </span>/ul>
```

Dialoogikastis olevat vealiikide puud genereeriv makro.

Lisa 3. Document.py faili täiendused

```
def countUsedOnlyOne(self, codeobj, REQUEST):
        "test"
        codes=self.REQUEST.SESSION.get('sessioncodes', "")
        if len(codes)<1:
          codes = self.Errors.getDocumentMarks(self.getId(),
str(REQUEST.AUTHENTICATED_USER))
          self.REQUEST.SESSION.set('sessioncodes', codes)
        res = []
        extend=res.extend
        codeobjID=codeobj.getCodeId()
        if((self.countUsedCodeTree(codeobj, REQUEST)==1) and
(self.countUsedCode(codeobjID, REQUEST)==1)):
          extend((self.allDiffUsedCodes(codeobjID,
REQUEST)[0][1],self.allDiffUsedCodes(codeobjID, REQUEST)[0][3],1))
        else:
          extend((0,0,0))
        return res
```

Funktsioon, mis kontrollib kas antud teksti puhul on etteantud märgendus veapuus leht.

```
security.declareProtected(perm_view_document, 'diffUsedCodes')
    def diffUsedCodes(self, codeobj, REQUEST):
        "kommentaar"
        codes=self.REQUEST.SESSION.get('sessioncodes', "")
        codesdoc=self.REQUEST.SESSION.get('sessioncodesdoc', "")
        if len(codes)<1 or codesdoc!=self:
          codes = self.Errors.getDocumentMarks(self.getId(),
str(REQUEST.AUTHENTICATED_USER))
          self.REQUEST.SESSION.set('sessioncodes', codes)
          self.REQUEST.SESSION.set('sessioncodesdoc', self)
        res = []
        append=res.append
        prettyCode=self.Marks.prettyCodeTitle
        truecolor="666666"
        for x in codes:
          add=1
          y = x.getProperty('code').encode('utf-8')
          for current in res:
            if current[0]==y:
             add=0
          if add==1:
             append([y, prettyCode(y), truecolor])
        return res
```

Funktsioon, mis töötleb andmebaasist saadud konkreetse teksti märgendusi nii, et tulemuseks on erinevatest vealiikidest koosnev massiiv.

```
def allUsedCodes(self, REQUEST):
        "teretulemast"
        codes=self.REQUEST.SESSION.get('sessioncodes', "")
        codesdoc=self.REQUEST.SESSION.get('sessioncodesdoc', "")
        if len(codes)<1 or codesdoc!=self:
          codes = self.Errors.getDocumentMarks(self.getId(),
str(REQUEST.AUTHENTICATED_USER))
          self.REQUEST.SESSION.set('sessioncodes', codes)
          self.REQUEST.SESSION.set('sessioncodesdoc', self)
        res = []
        append=res.append
        prettyCode=self.Marks.prettyCodeTitle
        counter=0
```

```
        for x in codes:
            y = x.getProperty('code').encode('utf-8')
            xpinter = x.getProperty('pointer').encode('utf-8')
            append([y, xpinter, counter, prettyCode(y)])
            counter+=1
        return res
```

Funktsioon, mis saab andmebaasist kätte kõik etteantud teksti märgenduste vealiigid.

```
def allDiffUsedCodes(self, code_name,  REQUEST):
        #Gets all different codes used in text - Harry
        codes=self.REQUEST.SESSION.get('sessioncodes', "")
        if len(codes)<1:
          codes = self.Errors.getDocumentMarks(self.getId(),
str(REQUEST.AUTHENTICATED_USER))
          self.REQUEST.SESSION.set('sessioncodes', codes)
        res = []
        append=res.append
        prettyCode=self.Marks.prettyCodeTitle
        counter = 1
        for code in codes:
            name = code.getProperty('code').encode('utf-8')
            if code_name == name:
                xpointer = code.getProperty('pointer').encode('utf-8')
                append([name, xpointer, counter, code.getId()])
                counter+=1
        return res
```

Funktsioon, mis saab kätte etteantud vealiigi erinevad märgendused konkreetse teksti puhul.

```
def prooviDelete(self, errorID):
        "kustutamiseks loodud abifunktsioon"
        self.Errors.deleteError(errorID)
        eorrorid = errorID
        tekst = " on nyyd kustutatud!"
        tulemus = str(eorrorid)+str(tekst)
        return tulemus
```

Funktsioon, mis saadab etteantud märgenduse unikaalse identifikaatori funktsioonile „deleteError“ kustutamiseks.

```
    def prooviAdd(self, xpoint, errorstring, content, pre, post, REQUEST):
        res=[]
        err_xpoint=xpoint
        document=self.getId()
        errors=errorstring.split('/')
        author = str(REQUEST.AUTHENTICATED_USER)
        for error in errors:
          res.append([err_xpoint, content, document, error, author, pre, post])
          self.Errors.addNewError(err_xpoint, content, document, error, author, pre,
post)
        return res
```

Funktsiooni, mis valmistab märgenduse lisamiseks etteantud andmed ette.

Lisa 4. document_viewSiim.pt faili täiendused

```
<h3>Veapuu</h3><button id="getUpdatedContent" type="button">Värskenda</button>
<div id="veapuu_sekstioon">
  <input id="puuvaade0_q" type="text" placeholder="Otsi..." />
  <div id="puuvaade0" style="width:90%" tal:define="cont python:
context.Marks.global_marks;round python:1">
    <span tal:omit-tag="" metal:use-macro="container/macros/codesTreePrinterDocument"
></span>
  </div>
</div>
```

Veapuu ning selle otsingu koodiosa.

```
<div id="dialog" title="Märgendamine">
  <p>Veendu, et selekteerides poleks ükski viga märgitud (joon all).</p>
    <fieldset class="fieldset-selection">
      <label id="selectionLabel" for="selectedText">Selekteeritud:</label><br>
      <textarea id="selectedText" rows="3" cols="21" disabled></textarea>
      <input id="selectionXpoint" value="0" disabled type="hidden" />
      <div id="currentErrorList"></div>
      <div id="errorListATM"></div>
      <div id="errorList"></div>
    </fieldset>
    <fieldset class="fieldset-tree">
      <input id="puuvaade1_q" type="text" placeholder="Otsi..." />
      <div id="puuvaade1" tal:define="cont python: context.Marks.global_marks;round
python:1">
        <span tal:omit-tag="" metal:use-
macro="container/macros/codesTreePrinterDocumentAll" ></span>
      </div>
    </fieldset>
</div>
```

Dialoogikasti ning selle sisse genereeritava kõikidest vealiikidest koosneva puu koodiosa.

Lisa 5. document_viewSiim.js failis tehtud muudatused

```
function updateMarkupSchema(error_type, action, leafNode, xpoint){
       $$(".document_markup_schema input").each(function(item){
    if(leafNode==0){//isn't leaf
      if (item.get('name') == error_type) {
        var state = item.get('value');
        var newstate = state;

        if(action == 1){newstate = 1;}
        if(action == 0){newstate = 0;}

        item.set('value', newstate);
        item.set('docolor',666666);
      }
    }else if(leafNode==1){//is leaf
      if(item.get('name') == error_type && item.get('xpoint') == xpoint) {
        var state = item.get('value');
        var newstate = state;

        if(action == 1){newstate = 1;}
        if(action == 0){newstate = 0;}

        item.set('value', newstate);
        item.set('docolor',666666);
      }
    }
       });
       displayCurrentErrors();
}
```

Funktsioon, mis sorteerib millised märgendused lähevad edasi alljoone lisamiseks või eemaldamiseks

```
function customMenu(node) {
  var items = {
    changeItem: {
        label: "Muuda",
        action: function (data) {
          var err_id = node.a_attr.errorid;
          if(parseInt(err_id) != 0){
            console.log(node);
            console.log(data);
            jQuery('#puuvaade1').jstree('uncheck_all');
            jQuery('#puuvaade1').jstree('close_all');
            jQuery('#puuvaade1').jstree('clear_search');
            jQuery('#puuvaade1_q').val('');
            jQuery("#errorList").html('');
            jQuery("#selectionLabel").text("Muutmine:");
            jQuery("#selectedText").hide();
            jQuery("#currentErrorList").show();

jQuery("#currentErrorList").html("<li><strike>"+node.text+"</strike></li>");
            jQuery("#selectionXpoint").val(node.a_attr.xpoint);

            dialog.dialog( "open" );
          }else{
            alert("Seda ei saa muuta, vali mõni alamviga.");
          }
        }
    },
    deleteItem : {
      "label"                    : "Kustuta",
      "action"                   : function (data) {
```

```
        var err_id = node.a_attr.errorid;
        if(parseInt(err_id) != 0){
          //delete from db
          var confirmation = false;
          if (confirm("Kas oled kindel, et soovid kustutada?") == true) {
              confirmation = true;
          } else {
              confirmation = false;
          }
          if(confirmation){
            jQuery(function($) {
              $.ajax({
                url : 'prooviDelete',
                data: {errorID: err_id},
                type : "POST",
                success : function(data)
                {
                  console.log("success");
                  console.log(data);
                },
                error: function(data){
                    console.log("fail");
                    console.log(data);
                }
              });
            });
            //delete from tree
            var inst = jQuery.jstree.reference(data.reference),
              obj = inst.get_node(data.reference);
            if(inst.is_selected(obj)) {
              inst.delete_node(inst.get_selected());
            }
            else {
              inst.delete_node(obj);
            }
          }
        }else{
          alert("Seda ei saa kustutada, vali mõni alamviga.");
        }
      }
    }
  };
  return items;
}
```

JsTree hüpikmenüü jaoks luuakse kaks valikut: Kustuta ja Muuda.

```
function jsTree(){

       jQuery("#puuvaade0").jstree({
    "core" : {
       "check_callback" : true
     },
             "checkbox" : {
                    "whole_node" : false,            //Kui false, siis mujale node
peale klikkides, ei checkita checkboxi
                    "tie_selection" : false,   //See seade kontrollib kas checkbox on
seotud üldise puu selekteerimisega või seesmise arrayga (kui false)
                    "three_state" : false
             },
    "contextmenu" : {
      "select_node" : false,   //true puhul paremklõps avab hrefi
      "items" : customMenu
    },
```

```
    "search" : {
      "show_only_matches" : false //Kui true, siis näitab ainult vasteid ja ülemaid
aga mitte alamaid
    },
              "plugins" : [ "checkbox", "contextmenu", "search" ]
       });


       jQuery("#puuvaade0").on("select_node.jstree", function(e,data){
          window.location.href = data.node.a_attr.href;
       });

       jQuery("#puuvaade0").on("check_node.jstree", function(e, data){
      isLeafNode=data.node.a_attr.leafnode;
      if(isLeafNode==0){updateMarkupSchema(data.node.a_attr.name, 1, isLeafNode, 0);
      }else if(isLeafNode==1){updateMarkupSchema(data.node.a_attr.name, 1,
isLeafNode, data.node.a_attr.xpoint);}
  });
       jQuery("#puuvaade0").on("uncheck_node.jstree", function(e, data){
      isLeafNode=data.node.a_attr.leafnode;
      if(isLeafNode==0){updateMarkupSchema(data.node.a_attr.name, 0, isLeafNode, 0);
      }else if(isLeafNode==1){updateMarkupSchema(data.node.a_attr.name, 0,
isLeafNode, data.node.a_attr.xpoint);}
  });

  var to = false;
  jQuery("#puuvaade0_q").bind("keyup change", function(e) {
    if(to) { clearTimeout(to); }
    to = setTimeout(function () {
      var v = jQuery('#puuvaade0_q').val();
      jQuery('#puuvaade0').jstree('search', v);
    }, 250);
  });

  ////jsTree in modal
  jQuery("#puuvaade1").jstree({
    "core" : {
       "check_callback" : true
     },
              "checkbox" : {
                    "whole_node" : true,             //Kui false, siis mujale node
peale klikkides, ei checkita checkboxi
                    "tie_selection" : true,    //See seade kontrollib kas checkbox on
seotud üldise puu selekteerimisega või seesmise arrayga (kui false)
                    "three_state" : true
              },
    "search" : {
      "show_only_matches" : true //Kui true, siis näitab ainult vasteid ja ülemaid
aga mitte alamaid
    },
              "plugins" : [ "checkbox", "search" ]
       });
  var to = false;
  jQuery("#puuvaade1_q").bind("keyup change", function(e) {
    if(to) { clearTimeout(to); }
    to = setTimeout(function () {
      var v = jQuery('#puuvaade1_q').val();
      jQuery('#puuvaade1').jstree('search', v);
    }, 250);
  });

  jQuery("#puuvaade1").on("select_node.jstree", function(e, data){
    selectedErrors=[];
```

```
    var list="";
    list+="<ul class='error-list'>";
    for(var i=0;i<(jQuery('#puuvaade1').jstree('get_bottom_selected')).length;i++){
      var error=jQuery('#puuvaade1').jstree('get_bottom_selected', 'full')[i];
      list+="<li name='"+error.a_attr.id+"'>"+error.text+"</li>";
      selectedErrors.push(error.a_attr.id);
    }
    list+="</ul>";
    jQuery( "#errorList" ).html(list);
  });
  jQuery("#puuvaade1").on("deselect_node.jstree", function(e, data){
    selectedErrors=[];
    var list="";
    list+="<ul class='error-list'>";
    for(var i=0;i<(jQuery('#puuvaade1').jstree('get_bottom_selected')).length;i++){
      var error=jQuery('#puuvaade1').jstree('get_bottom_selected', 'full')[i];
      list+="<li><a>"+error.text+"</a></li>";
      selectedErrors.push(error.a_attr.id);
    }
    list+="</ul>";
    jQuery( "#errorList" ).html(list);
  });
}
```

Funktsioon, mis rakendab märgendusmooduli mõlema andmepuu peal jsTree ning muudab nende elemendid interaktiivseteks.

```
window.onload=function(){
  contentArea=jQuery( "#contentDiv" ) ;
  jQuery( contentArea ).on( "mouseup", function() {
    console.log("mouseup");
    if (window.getSelection().toString().length>0) {
      var fullText=document.getElementById("contentDiv");
      var toRemoveSpans=jQuery(fullText).find('div.inlineMark');
      for(var i=0;i<toRemoveSpans.length;i++){
        jQuery(toRemoveSpans[i]).replaceWith( jQuery(toRemoveSpans[i]).text() );
      }

      fullText=document.getElementById("contentDiv");
      toRemoveSpans=jQuery(fullText).find('span.paintedMark');
      for(var i=0;i<toRemoveSpans.length;i++){
        jQuery(toRemoveSpans[i]).replaceWith( jQuery(toRemoveSpans[i]).text() );
      }
      var proov=jQuery(".ignore_xpath.beginMark").contents().unwrap();
      var proov = document.getElementsByClassName("paintedMark");
      console.log(proov);

      jQuery('#puuvaade1').jstree('uncheck_all');
      jQuery('#puuvaade1').jstree('close_all');
      jQuery('#puuvaade1').jstree('clear_search');
      jQuery('#puuvaade1_q').val('');
      jQuery("#errorList").html('');

      var selectionText = "";
      selection=window.getSelection();
      selectionText=selection.toString();

      jQuery("#selectionLabel").text("Selekteeritud:");
      jQuery("#selectedText").show();
      jQuery("#currentErrorList").hide();
      jQuery("#selectedText").val(selectionText);
      xpathData=getXPATH();
      jQuery("#selectionXpoint").val(xpathData[0]);
```

```
      var str="<p>Olemasolevad:</p>";
      str+="<ul>";
      $$(".document_markup_schema input").each(function(item){
        if(item.get('xpoint')==xpathData[0]){
          console.log(item.get('xpoint')+" vs "+xpathData[0]);
          console.log(item.get('rel'));
          str+="<li>"+item.get('rel')+"</li>";
        }
      });
      str+="</ul>";
      console.log(str);
      jQuery("#errorListATM").html(str);

      dialog.dialog( "open" );

    }
    else if (document.selection && document.selection.type != "Control") {
      alert("Internet Exploreris ei pruugi kõik korrapäraselt töötada, proovi
Chrome'i");
    }
  });

  dialog = jQuery( "#dialog" ).dialog({
    autoOpen: false,
    height: 500,
    width: 800,
    modal: true,
    buttons: {
      "Lisa": function() {

        var selectedErrorsString="";
        for(var i=0;i<selectedErrors.length;i++){
          selectedErrorsString+=selectedErrors[i]+"/";
        }
        var selectedErrorsString=selectedErrorsString.slice(0,-1);
        var errorXPATH=jQuery("#selectionXpoint").val();
        var textBefore=xpathData[1];
        var selectedTEXT=xpathData[2];
        var textAfter=xpathData[3];

        jQuery(function($) {
          $.ajax({
            url : 'prooviAdd',
            data: {xpoint: errorXPATH, errorstring: selectedErrorsString, content:
selectedTEXT, pre: textBefore, post: textAfter},
            type : "POST",
            success : function(data){
              console.log("success");
              console.log(data);
            },
            error: function(data){
                console.log("fail");
                console.log(data);
            }
          });
        });
        dialog.dialog( "close" );
      }
    },
    close: function() {
      console.log("kinni");
    }
  });
```

```
  jQuery( "#open-modal" ).button().on( "click", function() {
    dialog.dialog( "open" );
  });

}
```

Lehe laadimisel käivitatavad funktsioonid. Siin saadakse kätte selekteeritud tekstiosa ning selle täpne asukoht tekstis ning laetakse dialoogikasti jaoks jQuery UI.