TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Alvar Valtna
179280IAIB

ENTERPRISE ARCHITECT CASE VAHENDI UML KLASSISKEEMIDEST INIMKEELSETE LAUSETE GENEREERIMISE TARKVARA

Bakalaureusetöö

Juhendaja: Erki Eessaar
PhD

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Alvar Valtna

17.12.2021

Annotatsioon

Töö eesmärgiks on luua Enterprise Architect (EA) CASE vahendi (modelleerimisvahendi) täiendus, mis võimaldaks UML (*Unified Modeling Language*) klassiskeemidest e klassidiagrammidest genereerida loomuliku (inimkeele) keele lauseid.

Lauseid peab olema võimalik genereerida eesti keeles ja inglise keeles. Genereerimise tulemus pakub visuaalse skeemi alternatiivse esitusviisi ning genereerimine on tegelikult tõlkimine ühest keelest (UML) teise (loomulik keel). Genereerimise käigus peaks infokadu olema võimalikult väike. Tarkvara ülesanne ei ole tõlkida mudelielemente ühest inimkeelest teise. Tarkvara kasutab lausete moodustamiseks olemasolevaid mudelielementide nimesid. Tarkvara kasutaja valib kasutatava keele sõltuvalt mudelielementide nimedes kasutatavast keelest.

Bakalaureusetöö dokumendi alguses kirjutatakse teema teoreetilisest taustast, teema olulisusest ning töös kasutatud arendusmetoodikast. Seejärel esitatakse nõuded tarkvarale. Järgnevalt tuuakse välja, mis lahendusviise autor proovis ning kirjeldatakse loodud tarkvara ülesehitust ja tehnilist realisatsiooni. Lõpuks kirjeldatakse töö tulemuste kontrollimist, loodud tarkvara puudujääke ning selle edasiarendamise võimalusi.

Töö tulemusena valmis töötav tarkvara, mis on võimeline EA UML klassiskeemidest genereerima arusaadavaid loomuliku keele (inimkeele) lauseid. Lauseid on võimalik genereerida vaikimisi inglise ja eesti keeles. Kasutades tarkavas loodud malli saab luua uusi keelefaile, millega saab lisada tarkvarasse uute keelte toe. Keelefailide abil genereerib tarkvara laused uues keeles. Töö alamtulemuseks on klassiskeeme kirjeldavate lausete eestikeelne versioon.

Tarkvara on vaba ja avatud lähtekoodiga, millel on MIT litsents. Nii lähtekood kui ka kompileeritud tarkvara on leitav järgmisest GitHub-i salvest: https://github.com/alvaltna/NaturalLanguageGenerator.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 96 leheküljel, 10 peatükki, 66 joonist, 2 tabelit.

Abstract

A Software to Generate Human Language Sentences from Enterprise Architect CASE Tool's UML Class Diagrams

The goal of this bachelor thesis is to create an extension to Enterprise Architect (EA) modeling tool (CASE tool) that generates natural language (human language) sentences from UML (*Unified Modeling Language*) class diagrams. By default, the software should be capable of generating sentences in Estonian and English. Moreover, the system must allow adding support for new languages. The system provides an alternative representation of visual models (diagrams). Thus, the generation of sentences is in essence a translation from one language (UML) to another (natural language). Information loss during the generation should be as small as possible. The software does not have a task to translate the names of model elements from one human language to another. It uses the existing names of software elements. The users of the software choose a language based on the language of the model element names.

The current document starts with theoretical background about the topic, explanation as to why the topic is important, and overview of the used development process. Next, the author presents functional and non-functional requirements to the software. After that the author explains the first idea to implement the system, which was based on using the built-in template mechanism of the CASE tool. The author found out that this cannot be used to implement the system. Next, the author explains design and implementation of the actually used solution. Finally, the document describes checking of the software, which was done to make sure that the software satisfies requirements and that it is usable for the future users. For the checking the author generated statements based on various diagrams. Moreover, the author gave to three fellow students a task to draw diagrams based on the sentences that the software had generated. In addition, the author writes about the limitations of the software and future work with the software.

The result of this work is a software that generates natural language sentences from EA UML class diagrams. By default, one can generate sentences in Estonian and English. By

using the software's user interface, one can create new language files through the language template. If a new language file is created, then the software can use that file to generate sentences in that language. A subresult of the work is translation of sentences that describe class diagrams in Estonian.

Software is free and open-source with MIT License. Source code as well as compiled software can be found at the following GitHub link: https://github.com/alvaltna/NaturalLanguageGenerator.

The thesis is in Estonian language and contains 96 pages of text, 10 chapters, 66 figures, 2 tables.

Lühendite ja mõistete sõnastik

ANSI encoding	Kodeering, mis seab hulgale Windowsi süsteemides kasutatavatele märkidele vastavusse mingi koodi. See märkide hulk sisaldab lisaks ASCII kodeeringu märkidele veel 128 märki.
AWT	*Abstract Window Toolkit*, Java programmeerimiskeelele omane akendamise, graafika ning kasutajaliideste loomise teek [1].
CASE	*Computer-Aided Software Engineering*, tarkvara raalprojekteerimine.
CASE vahend	CASE vahend on tarkvara info- ja tarkvarasüsteemide projekteerimiseks. CASE vahend võimaldab koostada ühes või mitmes keeles süsteeme kirjeldavaid mudeleid ning kasutada neid mudeleid erinevate ülesannete lahendamiseks nagu näiteks süsteemi töö simuleerimine, dokumentatsiooni ja lähtekoodi genereerimine. Seega on CASE vahend modelleerimisvahend.
DDL	*Data Definition Language*, Andmete määratlus keel, andmekirjelduskeel. Selle keele lausete abil luuakse, muudetakse ja kustutatakse andmebaasi andmestruktuure ning teisi andmebaasiobjekte.
EA	*Enterprise Architect*. CASE vahend, mille täiendusena loodi käesolev tarkvara. Tarkvara arendamiseks kasutati EA ver 12, kuid tarkvara testiti ka EA ver 15 põhjal.
Gradle	*Gradle* on koodist valmis tarkvara loomise tööriist, mis juhib arendusprotsessi järgmiste ülesannete puhul: kompileerimine, pakendamine, testimine, kasutuselevõtt, avalikustamine [2].
HashMap	Andmestruktuur Java programmeerimiskeeles.
HTML	Standardne märgistuskeel, mis on disainitud dokumentide kuvamiseks veebilehitsejas [3].
Inimkeel	Vt loomulik keel
JavaScript	*JavaScript* on objektorienteeritud programmeerimiskeel, mida kasutatakse peamiselt veebilehtede skriptimiseks[4].
JDK	*Java Development Kit*, Java tehnoloogia kogum, mille levitajaks on *Oracle Corporation* [5]
JSON	*JavaScript Object Notation*, lihtsustatud andmevahetusvorming, mis põhineb Javascriptil [6]

Loomulik keel	Inimkeel. „Kokkuleppeline, piiramatu hääliksümbolisüsteem, mille õppimiseks on sünnipärased eeldused vaid inimesel.“ [7] Loomuliku keele näited on eesti keel ja inglise keel.
MDA	*Model Driven Architecture*, lähenemisviis tarkvara disainile, arendamisele ja teostusele, mis seisneb mudelitest töötava tarkvarani jõudmises[8].
MIT litsents	*Massachusetts Institute of Technology* poolt välja töötatud vaba tarkvara litsents, mis seab lähtekoodi ning tarkvara kasutamisele vähe piiranguid. [9]
OMG	*Object Management Group*, arvutitööstuse standardite konsortsium[10].
Skeem	Diagramm. Maailma mingit alamosa mingites aspektides kirjeldav joonis. Üks ja sama mudelielement võib olla esitatud mitmel sellisel joonisel.
SQL	*Structured Query Language*, Struktuurpäringukeel. Populaarne andmebaasikeel nii andmete otsimiseks, muutmiseks, tehingute juhtimiseks õiguste haldamiseks kui ka andmebaasiobjektide haldamiseks.
Swing	*Swing* on graafiliste kasutajaliideste loomiseks mõeldud teek Java programmeerimiskeelele[11].
UML	*Unified Modeling Language*, Ühtne modelleerimiskeel. Selles keeles saab luua visuaalseid mudeleid (diagramme), keel näeb ette hulga üldiseid skeemi tüüpe ning keelt saab kasutada väga erinevate valdkondade kirjeldamiseks.
UTF-8	*(Universal Coded Character Set*) *Transformation Format*, sümbolite kodeerimisviis.
VB	*Visual Basic*, Visual Basic on Microsofti poolt loodud programmeerimiskeel[12].
Ülikool	Tallinna Tehnikaülikool.
XMI	*XML Metadata Interchange*, metaandmete vahetus XML formaadis
XML	*Extensible Markup Language*, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Mudel on reaalse maailma lihtsustatud kirjeldus. Mudelikeskne arhitektuur (*Model Driven Architecture*) on lähenemisviis tarkvara disainile, arendamisele ja teostusele, mille eestvedajaks on OMG (*Object Management Group*)[8]. Mudelikeskse arhitektuuri ning sellega haakuva mudelite poolt juhitud arenduse (*Model Driven Development*) idee seisneb selles, et süsteemi arendamise käigus luuakse mudelid, mis kirjeldavad süsteemi erinevaid aspekte – nii staatilist struktuuri kui ka käitumist. Nendest mudelitest luuakse teisenduste kaudu uusi – tehnilisemaid – mudeleid ja lõpuks programmikoodi või siis on mudel sisendiks interpretaatorile, mis mudelis oleva info alusele pakub kasutajatele nende poolt soovitud funktsionaalsust. Tänapäeval on populaarsed paindmetoodikad. Ka paindmetoodikad ja modelleerimine ei ole üksteist välistavad [13] ning mudelite koostamine võib olla osa selliste metoodikate järgi toimuvast arendustööst. Üks mudelite koostamist võimaldav modelleerimisvahend e CASE (*Computer-Aided Software Engineering*) vahend on Enterprise Architect (EA). Tegemist on CASE vahendiga, mida kasutatakse Tallinna Tehnikaülikooli õppetöös ning mis on ka ettevõtetes kasutusel. Selles bakalaureusetöös käsitletakse EA funktsionaalsuse laiendamist nii, et seda vahendit oleks lihtsam kasutada laiemal hulgal inimestel.

1.1 Töö eesmärk

Töö eesmärgiks on luua EA CASE vahendile täiendus, mille ülesandeks oleks genereerida UML (*Unified Modeling Language*) klassiskeemide põhjal grammatiliselt võimalikult korrektsed loomuliku keele (inimkeele) laused nii, et graafiline (visuaalne) ja tekstiline mudelite esitus oleksid tähenduselt võimalikult samasugused. Sisuliselt on tegemist tõlkerakendusega, mis tõlgib klassiskeemi abil esitatud väited loomulikus keeles esitatud väideteks. Infokadu sellise tõlkimise juures peaks olema võimalikult väike.

Täienduse eesmärgiks on aidata modelleerijatel jooksvalt kontrollida mudelite õigsust, pakkuda mudelitega tutvumise võimalust neile, kes UML-i hästi või üldse ei tunne ning lihtsustada modelleerimise õppimist. Muuhulgas leiaks see kasutust ülikooli

andmebaaside õppeainetes. UMLi ja objektorienteeritud programmeerimist hästi tundvad inimesed ei ole selle vahendi sihtgrupp, mis muidugi ei tähenda, et nad seda kasutada ei võiks.

1.2 Töö tingimused ja nõuded

Täiendus oleks eeskätt mõeldud UML klassiskeemide põhjal loodud (analüüsi täpsusega) olemi-suhte skeemide tekstiliseks väljendamiseks, kuid oleks kasutatav ka teistel eesmärkidel loodud klassiskeemide puhul. Täiendus peab kindlasti töötama EA 12 versiooniga (mis on kasutusel ülikoolis) ning soovitavalt ka hilisemate versioonidega, sh värskeim versioon (2021. aasta sügise seisuga EA 15), millega plaanitakse tarkvara samuti testida.

Loodud täiendus peab olema avatud lähtekoodiga. Tarkvara peaks toetama eesti- ja inglise keelt, kuid ka teiste keelte kasutuselevõtt peaks olema kergesti lisatav.

Tarkvara ei tegele mudelielementide nimede tõlkimisega valitud keelde. Kasutaja peab valima, millises keeles on mudelielementide nimed ja vastavalt sellele, millises keeles lauseid ta soovib. Valitud keelt kasutatakse selleks, et moodustada lauseid.

Töös täiendatakse just EA vahendit järgmistel põhjustel.

- See on kasutusel ülikoolis ja õppetöö vajaks sellist täiendus.
- Autor on seda vahendit ise kasutanud.
- Vahendil on laiendusvõimalused.
- Tegemist on populaarse vahendiga.

Populaarsuse hindamiseks kasutati Google otsingut. 03.01.2022 käivitati järgmisel põhimõttel päringud:

(intitle:CASE OR intitle:modeling OR intitle:modelling OR intitle:UML) AND intitle:"**siin CASE vahendi nimi**". Tulemuseks saadud vastuste arv on esitatud Tabel 1.

Tabel 1. Kümne erineva CASE vahendi Google otsingu vastuste arv.

CASE vahendi nimi	Vastuste arv
DIA	23300
Enterprise Architect	3430
PlantUML	1510
Poseidon for UML	1280
ArgoUML	1080
DBDesigner	57
pgModeler	26

Kuigi EA vahendil on mitmeid kolmandate osapoolte loodud laiendusi [14][13], siis pole hetkel ühtegi laiendust, mis oleks juba selle probleemi ära lahendanud.

1.3 Töö teoreetiline taust

Käesolev töö tugineb bakalaureusetööle [15], mille tulemusel loodi nii-öelda kontseptsiooni tõestav (*Proof-of-Concept*) lahendus lihtsakoelise veebipõhise modelleerimisvahendi näol. Töös [15] on põhjalikult uuritud kuidas tõlkida UML klassiskeem ingliskeelseteks lauseteks. Seal on välja toodud kindlad grammatilised reeglid, kuidas kirjeldada klassiskeemi erinevaid aspekte nii, et võimalikult paljud erinevad klassiskeemide aspektid oleksid kaetud.

Autor kasutab töös välja toodud grammatilisi reegleid ka oma töös lausete genereerimiseks ja viitab sellele tööle. Viidatud lõputööd saab kasutada teoreetilise põhjana lausete genereerimiseks. Autor mõistab, et eestikeelsete ja teiste keelte lausete puhul peab tõenäoliselt mitmeid reegleid muutma ja juurdegi lisama. Samuti ei pruugi kõik kasutusjuhud olla kaetud ning täielikult töötada.

1.4 Töö struktuur

Töö teises peatükis antakse teoreetiline lühiülevaade UMLi klassiskeemidest, EA CASE vahendist ja klassiskeemidest inimkeelsete lausete genereerimisest. Samuti kirjutatakse töö olulisusest ja töö metoodikast.

Kolmandas peatükis esitatakse funktsionaalsed ja mittefunktsionaalsed nõuded tarkvarale.

Neljandas peatükis kirjeldatakse probleemi lahendamise strateegiaid – üldlahendus *vs.* konkreetse tarkvara spetsiifiline lahendust

Viiendas peatükis kirjeldatakse, kuidas autor üritas lahendada lõputöö probleemi kasutades EAsse sisseehitatud malle, mis on loodud erinevat tüüpi väljundite genereerimiseks EA mudelite põhjal.

Kuues peatükk kirjeldab EAs loodud mudelitest andmete kättesaamist kasutades selleks skripti. Samuti selgitatakse töös loodud tarkvara kasutajaliidese disaini ning tagarakenduse arhitektuuri.

Seitsmendas peatükis on õpetus tarkvara installeerimiseks ja kasutamiseks.

Kaheksas peatükk kirjeldab kuidas kontrolliti lõpptulemust kasutades ühikteste, mahukamaid klassiskeeme ja paludes tagasisidet Tallinna Tehnikaülikooli üliõpilastelt. Samuti esitatakse seal tarkvara kasutamise näide.

Üheksandas peatükis toob autor välja tarkvara praegused puudused ning millises suunas võiks toimuda edasine arendus.

2 Teoreetiline taust

Käesoleva peatükk esitab bakalaureusetöö teoreetilise tausta. Autor selgitab, mis on UML (*Unified Modeling Language*), klassiskeemid, EA CASE vahend, miks on autori töö oluline ning millist metoodikat rakendatakse töö käigus.

2.1 UML ja UMLi klassiskeem

UML on ühtne modelleerimiskeel. See võimaldab koostada visuaalseid mudeleid, kus mudelielemendid on esitatud graafiliste kujunditena skeemidel e diagrammidel. Põhjuseid just UML keele kasutamiseks on mitmeid. See annab inimestele hea võimaluse luua ja jagada visuaalseid mudeleid, mille abil vajalikku infot edasi anda. UML on universaalne selles mõttes, et see pole seotud konkreetse programmeerimiskeele või arendusmetoodikaga. Laialdaselt kasutatav ning standardiseeritud keel võimaldab erinevatel mudelite loojatel ning kasutajatel üksteise mudelitest aru saada, julgustab looma modelleerimisvahendeid ning loob ka võimaluse, mudelite ülekandmiseks erinevate modelleerimisvahendite vahel.

UML 2.5 näeb ette 14 diagrammi e skeemi tüüpi [16] UML koosneb hulgast erinevatest skeemitüüpidest, mis võimaldavad näidata süsteemi erinevatest perspektiividest ning erinevate huvitatud osapoolte (äriinimesed, nõuete kogujad, arendajad, hooldajad) jaoks. Neid skeemitüüpe saab grupeerida kahte suuremasse kategooriasse – struktuuriskeemid (*Structure Diagrams*) ja käitumuslikud skeemid (*Behavior Diagrams*) [17].

„UML klassiskeem on staatiline struktuuriskeem, mis kirjeldab süsteemi struktuuri, näidates süsteemi klasse, nende atribuute, operatsioone ja suhteid klasside vahel“[18]. Klassiskeemide peamine ülesanne on anda arendajatele kiire ja graafiline ülevaade süsteemi ülesehitusest. UML mudeleid saab kasutada väga erinevate ülesannete jaoks. Sealhulgas saab UML klassiskeeme kasutada näiteks nii valdkonna analüüsiks (domeenimudel), nõuete esitamiseks andmebaasi kohta (kontseptuaalne andmemudel), tarkvara klasside kirjeldamiseks (disaini klassiskeem) kui ka andmebaasi tehnilise kirjelduse esitamiseks (andmebaasi disaini mudel). Objektorienteeritud analüüsi ja disaini

käigus on klassiskeemid ühed peamised mudelid, mida tuleks koostada. Disaini täpsusega klassiskeemidest ja andmebaasi disaini mudelitest saab genereerida vastavalt programmikoodi ja andmebaasikeele koodi. Klassiskeem kuulub kõige populaarsemate ja rohkem kasutatavate UML skeemi tüüpide hulka [19].

2.2 EA CASE vahend

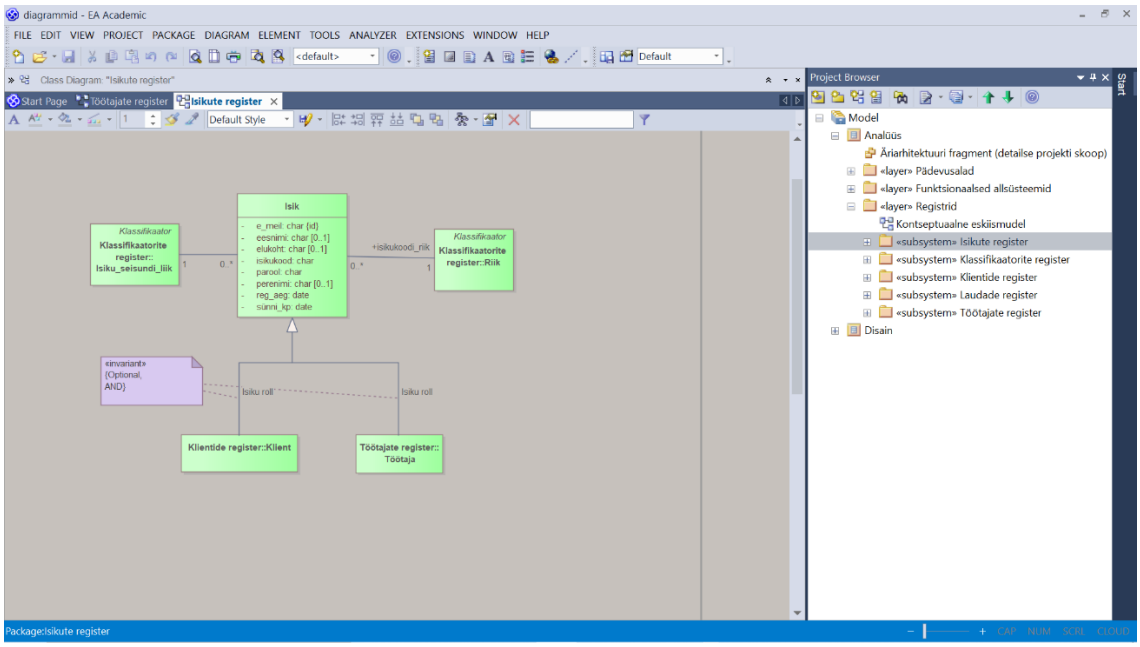
„Enterprise Architect CASE (*Computer-Aided Software Engineering*) vahend on visuaalse modelleerimise ja disaini tööriist. Platvorm toetab tarkvarasüsteemide disaini ja ehitamist, äriprotsesside modelleerimist ning ka teiste valdkondade (nt. teadus, haridus) süsteemide kavandamist ja kirjeldamist. Ettevõtted ei kasuta seda tarkvara mitte ainult neile vajalike süsteemide modelleerimiseks, vaid nende süsteemide kogu elutsükli kaardistamiseks ning realiseerimiseks“ [20]. EA pakub võimalust joonistada erinevat tüüpi UML skeeme. Lisaks sellele on võimalik luua kohandatud skeeme ja elemente, et kasutada UMLi ka teistes valdkondades väljaspool tarkvaraarendust [21]. EAs on võimalik veel lisaks näiteks genereerida SQL andmebaasi disaini esitavatest klassiskeemidest SQL andmekirjelduskeele (DDL, *Data Definition Language*) lauseid, disaini klassiskeemidest programmikoodi (nt Java ja C keeltes) ning mudelitest ka dokumentatsiooni. Iga eelmainitud väljundi genereerimiseks on EAs omad mallid. Kasutaja saab muuta olemasolevaid malle, et väljund vastaks tema nõutele või luua hoopis enda kohandatud mall. Lisaks UML keelele saab modelleerimiseks kasutada ka mitmeid teisi keeli nagu näiteks SysML ja Archimate.

EA on tasuline ja suletud lähekoodiga tarkvara.

EA tarkvara on võimalik täiendada. Tarkvarale on loodud ka palju lisamooduleid, mis lisavad sinna funktsionaalsust ja kasutusmugavust [14]. Ühe täiendusega [22], mis parandas kontseptuaalsest andmemudelist andmebaasi disaini mudelite loomise kvaliteeti, oli autor tuttav ka ülikoolis õppimise ajast ning selle täienduse kohta tehtud lõputöö oli üheks sisendiks, mille alusel EA vahendi täiendamisega tutvust teha.

EA kasutajaliides koosneb põhiliselt kolmest osast: tööriba, lõuend kuhu joonistada skeem ning *Project Browser*, kus on kogu projekti struktuur jagatud pakettideks, mis sisaldavad omakorda kas teisi pakette või skeeme. Iga skeem sisaldab elemente. Klassiskeemide puhul on kõige rohkem kasutatavad elemendid *Class*, *Inteface*, *Note* ja

Constraint. Need on ka peamised elemendid, mida kasutatakse Andmebaasid I ja II aines andmebaasi registrite analüüsis klassiskeemide joonistamisel. Joonisel 1 on näidatud, milline näeb välja EA kasutajaliides skeemide joonistamisel.



Joonis 1. EA 12 kasutajaliidese näide

2.3 Klassiskeemidest lausete genereerimine

EA tarkvaral on palju kolmandate osapoolte loodud lisamooduleid [14], kuid nende hulgas pole ühtegi, mis genereeriks skeemidest loomuliku keele lauseid. Sellised laused peaksid olema mõistetavad ka õppuritele, algajatele tarkvaraarendajatele, neile mitte IT-spetsialistidele, kes koostavad mudeleid millekski muuks kui tarkvaraarendus ning tarkvaraarenduses osalevatele ja arendajatega koostööd tegevatele kliendi esindajatele. Loomuliku keele laused suurendaksid mudelite ligipääsetavust ning aitaksid inimestel paremini mõista, mida nad skeemidele on üles joonistanud. Samuti aitaksid need leida loogikavigu, mis ei pruugi joonistelt nii selgelt välja paista.

Selle töö eesmärgiks ongi luua tarkvara, mis genereerib lauseid, mille mõistmiseks ei ole vaja palju tarkvaralisi teadmisi ning mis on universaalselt arusaadavad võimalikult paljude eluvaldkondade inimestele. Tarkvara eeskujuks on objekti-protsessi modelleerimist toetav CASE vahend OPCAT [23], kus sama mudelit saab vaadata nii graafilisel kui tekstilisel kujul.

2.4 Töö olulisus

Autor leidis teadusartikli, kus on uuritud kuidas tarkavara disaini puhul automatiseeritud tagasiside kiirendab õppimisprotsessi [24]. Näiteks viidi läbi eksperiment, kus 20 õpilast jagati kahte gruppi. Õpilastele anti ülesanne joonistada klassiskeemid. Ühe grupi õpilased said automatiseeritud tagasisidet enda klassiskeemide kohta, samas kui teine grupp ei saanud tagasisidet. Peale skeemide joonistamist hindasid kaks eksperti õpilaste tööd ning andsid hinde skaalal 1–5, kus 1 on madalaim hinne. Tööst selgus, et automatiseeritud tagasisidel on statistiliselt märkimisväärne positiivne mõju õpilaste võimele täita tarkvara disaini ülesandeid.

Ühes teises teadusartiklis [25] on väljatoodud, et süsteemi nõuete spetsifikatsiooni dokument on kirjutatud tihti loomuliku keele lauseid kasutades, sest see on ainus keel, mida klient mõistab ning klient on loomulikult nõus allkirjastama vaid dokumenti, millest saab ise aru. Artikkel väidab, et 20%–25% projekti ajast kulutatakse süsteemi nõuete defineerimisele. Tarkvara, mis genereerib analüüsi mudelitest loomuliku keele laused, võimaldab alustada varem modelleerimist ning genereerida hiljem mudelist dokumendi.

2.5 Töö metoodika

Käesoleva töö metoodikaks on disainiteadus [26]. Disainiteaduse põhimõte on kavandada ja realiseerida tehis (algoritm, programm, kasutajaliides) ning seejärel kontrollida mingil viisil, et loodud lahendus on hea. Tehist on mõtet luua ainult juhul kui varem on tehtud eeltööd, veendumaks, et seda on päriselt ka kellelegi vaja.

Autor tugines disainiteaduslike juhiste saamiseks teadusartiklile [27] ning valis tulemuste hindamise strateegiaks *Human Risk & Effectiveness*. See strateegia on kõige parem kui saab tulemusi hinnata päris kasutajatega tegeliku toote peal, sest siis saab veenduda, et tulemusest on pikas perspektiivis kasu ka tegelikus kasutamise olukorras.

Järgmise sammuna tuleb defineerida üldine hulk omadusi mida hinnata disaini puhul. Valitud omadustest sõltub, milline on projekti lõppeesmärk. Autori valitud omadusteks on kasutatavus, tõhusus, korrektsus, usaldusväärsus, kui lihtne üle viia teisele süsteemile, arusaadavus ning laiendatavus.

Viimaks määratleb autor hindamise etapid. Esimeseks etapiks on analüüsida, milliseid funktsionaalsed ja mittefunktsionaalseid nõudeid peab esmane tarkvara täitma. Teises etapis tuleb testida igat uut lausete genereerimise funktsionaalsuse osa ühiktestidega ja kasutajaliidese funktsionaalsust ning mugavust tuleb autoril peale iga funktsionaalsuse lisamist enda peal testida. Samuti tuleb küsida pidevat tagasisidet juhendajalt lausete korrektsuse ja arusaadavuse kohta ning saada nõu kasutajaliidese disaini ja kasutusmugavuse kohta. Tarkvara laiendatavuse kohta saab tagasisidet jooksvalt töö käigus, sest programmi tuleb jooksvalt laiendada, et lisada uut lausete genereerimise funktsionaalsust.

Kolmandas etapis, kui kogu lausete genereerimise funktsionaalsus on realiseeritud, tuleb testida selle korrektsust suure ja tervikliku EA projekti peal ning lausete arusaadavust teiste üliõpilaste peal. Samuti tuleb testida kasutajaliidese kasutatavust ja arusaadavust teiste üliõpilaste peal.

Neljandas etapis tuleb testida tarkvara üleviidavust teistesse süsteemidesse, paludes juhendajal ja teistel üliõpilastel proovida installeerida tarkvara nende süsteemile.

3 Analüüs

Selles peatükis esitatakse probleemi põhjalikum analüüs ning kirjeldatakse, millised on funktsionaalsed ja mittefunktsionaalsed nõuded tarkvarale.

3.1 Eesmärk

Töö eesmärgiks on luua tarkvaraline lahendus, mis oleks võimeline genereerima loomuliku keele lauseid EA CASE vahendis loodud UML klassiskeemidest. Lauseid peaks olema võimalik genereerida nii skeemist, kogu paketist (mis sisaldab null või rohkem skeemi) kui ka suuremast projekti alamosast, kus pakettide sees on omakorda paketid. Laused kuvatakse kasutajale kasutajaliideses. Laused peavad olema loomuliku keele laused ning need peavad sisaldama võimalikult vähe tarkvaraarenduse termineid ja üldse tehnilisi termineid. Lausetes kasutatav keel peab olema muudetav, st muutes kasutatavat keelt tõlgitakse laused teise keelde ümber, jättes küll mudelielementide nimed tõlkimata. Järgnevas näites on rasvasega muutumatu osa, milleks on mudelielementide nimed ja mitterasvasega muutuv osa, mis sõltub kasutatavast keelest.

Näide lausetest eesti keeles.

Klass **Isik**
Isik iseloomustab üks **isikukood**, mis on tüüpi char.
Isik iseloomustab üks **e_meil**, mis on tüüpi char.
Isik iseloomustab üks **parool**, mis on tüüpi char.

Näide sama mudeli põhjal genereeritud lausetest inglise keeles.

Class **Isik**
Isik is characterized by one **isikukood** that has type char.
Isik is characterized by one **e_meil** that has type char.
Isik is characterized by one **parool** that has type char.

Lõputöö tugineb teoreetilise poole pealt 2011. aasta bakalaureusetööle, mille pealkirjaks on *Generation of Sentences Based on UML Class Diagrams* [15]. Selles töös uuriti, milline peaks olema genereeritud lausete ülesehitus ning kuidas lauseid grupeerida ja neid lõppkasutajale esitada. Viidatud töö tulemusena valmis kontseptisooni tõestav (*Proof of Concept*) veebipõhine näiterakendus. Käesolev töö üritab realiseerida reaalselt kasutatavat tarkvara. Lisas 2 on näha viidatud lõputöös loodud lausete struktuurid, millele autor enda realisatsioonis tugines. Viidatud lõputöös esitati ingliskeelsete lausete struktuur. Üheks käesoleva töö oodatavaks tulemuseks on nende lausete eestikeelse tõlke väljapakkumine. Eestikeelsete lausete näited esitatakse jaotises 3.4.

Tarkvara peab suutma eristada seda, kas mudelielement on tüüpi *Class* (Klass), *Interface* (Liides), *Note* (Märge), *Constraint* (Piirang), *UseCase* (Kasutusjuht) või *Artifact* (Tehis). Kui lauseid genereeritakse paketist, siis kõige esimesena kuvatakse paketi nimi. Seejärel kuvatakse skeemi (diagrammi) nimi ning elementide nimed, mis on sellel skeemil. Skeemi siseselt kuvatakse esmalt elemendid tüüpi *Note*, mis on skeemi alguses ning pole ühendatud ühegi teise elemendiga. Need elemendid kuvatakse kasutajaliideses alajaotuses *Introductory comments* ehk sissejuhatavad märkmed. Neile järgnevad elementide tüüpi *Artifact* kirjeldused alajaotuses *Artifacts* ehk tehised. Seejärel kuvatakse kasutajaliideses alajaotuses *Element descriptions* ehk elementide kirjeldused laused elementide kohta, mis on tüüpi *Class*, *Interface* ja *UseCase*.

Tüüpi *Class*, *Interface* ja *UseCase* elementide kohta genereeritakse väljundisse kõigepealt selle tüüp ja nimi. Seejärel esitatakse atribuudid koos võimsustike ja andmetüüpidega ning algväärtusega, kui see eksisteerib.

Atribuudile lisatud võimsustik peab olema EA mudelis kirjeldatud ühel järgmisel kujul.

- *n...m*, kus *n* on täisarv, millele järgneb kaks punkti ning *m*, kus *m* on samuti täisarv. Eeldatavalt *n* on väiksem võrdne kui *m*.
- *n..**, kus *n* on täisarv, millele järgneb kaks punkti ning sümbol *.
- *n*, kus *n* on täisarv.
- *.

Peale seda esitatakse kõik operatsioonid, mida element teha saab. Kui elemendiga (sh atribuudi ja operatsiooniga) on seotud märkmed (*notes*), siis need kuvatakse elemendi kirjelduse juures. Elemendid on järjestatud elementide kirjelduse alajaotuses elemendi nimede järgi tähestikulises järjekorras. Vastavate elementide lausete struktuur on välja toodud viidatud lõputöös jaotises 2.4. Autori poolt lisati peale juhendajaga arutamist elemendi *UseCase* kohta lausete genereerimine, leides, et see on oluline täiendus.

Järgmisena kuvatakse laused, mis lähevad alajaotusesse *Connectors* ehk ühendajad (konnektorid). Ühendajad kirjeldavad elementide vahelisi sidemeid e suhteid e seoseid. Kui skeemil olevate elementide vahel on üldistusseosed mis kuuluvad üldistushulka, siis need kuvatakse kõige esimesena. See on autoripoolne täiendus, mida viidatud töös ei olnud välja toodud, aga mille autor otsustas peale juhendajaga konsulteerimist lisada.

Seejärel kuvatakse elementide nimede järgi tähestikulises järjekorras iga elemendiga seotud ühendajate põhjal genereeritud laused. Tarkvara suudab luua lauseid järgnevat tüüpi ühendajate kohta: üldistus (*Generalization*), realisatsioon (*Realisation*), sõltuvus (*Dependency*), kasutus (*Usage*), seos (*Association*), agregatsioon (*Aggregation*) ja kompositsioon (*Composition*).

Kui ühendaja on tüüpi *Association, Aggregation* või *Composition,* siis kirjeldatakse ka nende võimsustikke. Tarkvara suudab kasutada võimsustikke, mis on EAs kirjeldatud ühel järgmisel kujul.

- *n...m*, kus *n* on täisarv, millele järgneb kaks punkti ning *m*, kus *m* on samuti täisarv. Eeldatavalt *n* on väiksem võrdne kui *m*. Vastasel juhul tarkvara genereerib laused, mille sisu pole loogiliselt korrektne.
- *n..**, kus *n* on täisarv, millele järgneb kaks punkti ning sümbol *.
- *n*, kus *n* on täisarv.
- *.

Ühendajate lausete struktuur on välja toodud viidatud lõputöös jaotistes 2.5 ja 2.6.

Autor lisas funktsionaalsuse, et tarkvara eristaks *Aggregation* ja *Composition* tüüpi ühendust ning *Composition* tüüpi ühenduse puhul genereeritakse lisaks lause, mis ütleb,

et kui element, mis on tervik kustutatakse, siis tuleks kustutada ka element, mis on selle osa. Samuti on autoripoolne muudatus võrreldes viidatud tööga see, et märkeid, mis on seotud mitme ühendajaga ei kuvata mitte peale kõikide ühendajate kirjelduste kuvamist, vaid peale iga ühendaja kirjeldust, millega need märkmed on seotud. Lisaks sellele lisas autor funktsionaalsuse, et kui elemendil tüüpi *UseCase* on ühendaja tüüpi *Realisation*, mis loob seose elemendiga tüüpi *Class,* siis genereerib tarkvara lause kirjeldamaks just nende kahe elemendi tüübi vahelist suhet.

Järgmisena genereeritakse laused kasutajaliideses alajaotusse *Additional comments* e täiendavad märkmed. Sinna lisatakse laused kõikide *Note* tüüpi elementide kohta, mis pole seotud ühegi teise elemendiga ja asuvad skeemil allpool kõiki *Class, Interface* ja *UseCase* tüüpi elemente.

Lausete esituse üldine ülesehitus on võetud viidatud lõputöö põhitekstist ning realiseeritud mõningate autoripoolsete täiendustega. Viidatud lõputöös kirjeldatud üldine lausete jaotus on näha Lisas 3.

3.2 Funktsionaalsed nõuded

Järgmisena kirjeldab autor süsteemi funktsionaalseid nõudeid kasutades selleks kasutuslugusid (*user stories*) [28]

- Mina kui kasutaja tahan genereerida oma joonistatud klassiskeemist loomuliku keele lauseid, et mõista paremini joonistatut ja leida võimalike loogikavigu.
- Mina kui kasutaja tahan genereerida tervest paketist loomuliku keele lauseid, et saada parem ülevaade kogu enda projektist.
- Mina kui kasutaja tahan näha genereeritud lausete alguses paketi nime, et paremini mõista, millise paketi kohta on järgnevad laused genereeritud.
- Mina kui kasutaja tahan näha genereeritud lausete alguses skeemi nime ja seda sisaldavate elementide nimesid, et paremini mõista, millise skeemi kohta on laused genereeritud.
- Mina kui kasutaja tahan juhul kui lauseid genereeritakse mitme paketi kohta eristada, kust algavad konkreetse paketi kohta käivad laused, selleks, et suures infohulgas paremini orienteeruda.

- Mina kui kasutaja tahan näha ühe skeemi põhjal genereeritud lausete puhul tutvustavaid märkmeid, et mõista paremini kogu skeemi sisu.
- Mina kui kasutaja tahan näha skeemi kohta käivaid elemente tüüpi *Artifact*, et näha millised dokumendid on seotud antud skeemiga.
- Mina kui kasutaja tahan näha tüüpi *Class*, *Interface* ja *UseCase* elementide kirjeldusi, mis sisaldavad elemendi tüüpi, nime, atribuute, atribuutide võimsustikke, operatsioone ning märkmeid, et saada parem ülevaade nendest elementidest.
- Mina kui kasutaja tahan, et elemente kirjeldavad laused oleksid grupeeritud elementide kaupa tähestikulises järjekorras elemendi nimede järgi, et mind huvitav info oleks organiseeritud ja kiiresti leitav.
- Mina kui kasutaja tahan näha skeemi elementide ühendajate kirjeldusi, et saada parem ülevaade, mis viisil elemendid omavahel suhestuvad.
- Mina kui kasutaja tahan näha skeemi üldistushulkade kirjeldusi, et saada täpsemat infot kokku kuuluvate üldistuste kohta.
- Mina kui kasutaja tahan, et ühendajaid kirjeldavad laused oleksid grupeeritud elementide kaupa tähestikulises järjekorras, et mind huvitavad laused oleks organiseeritud ja kiiresti leitavad
- Mina kui kasutaja tahan näha täiendavaid märkmeid, et saada täpsemalt aru skeemi peensustest.
- Mina kui kasutaja tahan genereerida lauseid endale valitud keeles, et saada lausetest paremini aru.
- Mina kui kasutaja tahan selgeid juhiseid, kuidas lisada uut keelt, et ma ei peaks raiskama aega saamaks aru, kuidas see funktsionaalsus töötab.
- Mina kui kasutaja tahan uue keele lisamiseks kasutada keelemalli, et olla kindel, et minu koostatud laused ühilduvad tarkvaraga.
- Mina kui kasutaja tahan võimalust muuta genereeritavate lausete sõnastust, et teha laused loetavamaks.
- Mina kui kasutaja tahan eemaldada tarkvarast keelte toe, mida ma enam ei vaja.
- Mina kui kasutaja tahan, et keelte toe eemaldamine toimiks läbi kasutajaliidese, sest see lisab juurde kasutusmugavusele.
- Mina kui kasutaja tahan kopeerida genereeritud lauseid, et neid jagada oma kolleegidega ning erinevate teiste huvitatud osapooltega.

- Mina kui kasutaja tahan valida, millisesse faili sisendandmed kirjutatakse, et mul oleks ülevaade kogu oma arvuti failisüsteemist.
- Mina kui kasutaja tahan, et sisendandmete faili valimine töötaks läbi kasutajaliidese, sest see on kiirem ja mugavam.
- Mina kui kasutaja tahan, et tarkvara jätaks meelde ka peale sulgemist, millisest failist lugeda sisendandmeid, et sisendandmete faili ei peaks iga kord uuesti valima.

3.3 Mittefunktsionaalsed nõuded

Käesolevas jaotises toob autor välja tarkvara mittefunktsionaalsed nõuded.

- Tarkvara töötamiseks ei ole vaja internetiühendust. Tarkvara töötab igas masinas, millel on Windows 10 operatsioonisüsteem.
- Tarkvara peab olema lihtne allalaadida ja installeerida isikul, kel pole palju tarkvaralisi teadmisi. Samuti peab tarkvara olema lihtne ja intuitiivne kasutada.
- Tarkvara peab töötama EA 12-ga ja EA 15-ga.
- Kui kasutaja genereerib lauseid skeemidest, mis pole klassiskeemid siis kuvatakse kasutajaliideses, et skeem on tühi või ei kuvata üldse lauseid. Juhul kui klassiskeemil on ühendajaid ja elemente, mille põhjal lausete genereerimist autori tarkvara ei toeta, siis nende kohta lauseid ei genereerita.
- Tarkvara peab olema uue funktsionaalsuse lisamiseks kergesti laiendatav.
- Tarkvara peab olema avatud lähtekoodiga vaba tarkvara, mida võib igaüks kasutada ja täiendada vastavalt enese vajadusele. Tarkvara lähtekoodi litsentseerimiseks kasutatakse MIT litsentsi, sest see lubab inimestel tarkvara kasutada, kopeerida, muuta, liita teiste tarkvaradega, avaldada, levitada, ja müüa koopiaid tarkvarast. Ainukeseks tingimuseks on see, et iga koopia või tarkvara, mille kood koosneb enamjaolt selle tarkvara koodist, peab samuti omama MIT litsentsi.
- Lähtekood ja kompileeritud tarkvara peavad olema hoiustatud ning avalikustatud koodi haldamise platvormil GitHub.

3.4 Eestikeelsete lausete struktuur

Selles jaotises asuvas Tabelis 2 tuuakse välja elementide kirjeldamise lausete struktuur eesti keeles. Need laused põhinevad Lisa 2 esitatud ingliskeelsetel lausetel, kuid mõned laused on ka autori poolsed täiendused. Tõlgete tegemisel on lähtutud standardipõhise tarkvaratehnika sõnatiku [29] ja e-teatmiku [30] soovitustest.

Tabel 2. Elementide kirjeldamise lausete struktuur eesti keeles.

Mille kohta on laused?	**Laused**
Skeemi kirjeldus.	Skeem *nimi* kirjeldab *Klass A, Klass B…* ja suhteid nende vahel.
Elemendi tüüpi (*Class, Interface, UseCase*) nimega *A* atribuudi kirjeldus.	*A* iseloomustab *N* kuni *M* atribuuti *atribuudi nimi*, mis on tüüpi *atribuudi andmetüüp* algväärtusega *atribuudi algväärtus* (kui eksisteerib). *A* võib iseloomustada null kuni *M* atribuuti *atribuudi nimi*, mis on tüüpi *atribuudi andmetüüp* algväärtusega *atribuudi algväärtus* (kui eksisteerib).
Elemendi tüüpi (*Class, Interface, UseCase*) nimega *A* operatsiooni kirjeldus.	*A* saab *operatsiooni nimi*.
Element *A* ja *B* vahel on ühendaja tüüpi *Generalisation*, kus A on ülemtüüp.	*A* võib olla *B.* *B* on *A.*
Element *A* ja *B* vahel on ühendaja tüüpi *Realisation*, kus *B* realiseerib *A*d.	*A* võib olla *B.* *B* on *A.*
Element *A* ja *B* vahel on ühendaja tüüpi *Dependency*, kus *A* sõltub *B*st.	*A* sõltub *B.*
Element *A* ja *B* vahel on ühendaja tüüpi *Usage*, kus *A* kasutab *B*d.	*A* kasutab *B.* *B* on kasutatud *A* poolt.
Element *A* ja *B* vahel on ühendaja tüüpi *Association*.	*A* on seotud *B.* *B* on seotud *A.*

Mille kohta on laused?	**Laused**
Element *A* ja *B* vahel on ühendaja tüüpi *Association* koos võimsustikuga 1) *. *2) 0..*.* *3) 1..*.* 4) *0..1*. 5) *0..n*. 6) *1..n*. 7) *n*. 8) *n..m*. 9) *n..**.	1) Iga *A* võib olla seotud mitme *B*. 2) Iga *A* võib olla seotud mitme *B*. Mõni *A* ei ole seotud ühegi *B*. *3)* Iga *A* võib olla seotud mitme *B*. Iga *A* peab olema seotud vähemalt ühe *B*. 4) Iga *A* võib olla seotud kõige rohkem ühe *B*. Mõni *A* ei ole seotud ühegi *B*. 5) Iga *A* võib olla seotud kõige rohkem *n B*. Mõni *A* ei ole seotud ühegi *B*. 6) Iga *A* võib olla seotud kõige rohkem *n B*. Iga *A* peab olema seotud vähemalt ühe *B*. 7) Iga *A* peab olema seotud täpselt *n B*. 8) Iga A võib olla kõige rohkem seotud *m B* Iga *A* peab olema seotud vähemalt *n B*. 9) Iga *A* võib olla seotud mitme *B*. Iga *A* peab olema seotud vähemalt *n B*.
Element *A* ja *B* vahel on ühendaja tüüpi *Aggregation*, kus *A* on tervik.	*A* omab *B*. *B* kuulub *A*.
Element *A* ja *B* vahel on ühendaja tüüpi *Composition*, kus *A* on tervik.	*B* on osa *A* ja ei saa eksisteerida ilma selleta. Kui *A* on kustutatud, siis *B* peaks samuti olema kustutatud. *A* omab *B*. *B* kuulub *A*.
Element tüüpi *UseCase* nimega *A* ja element tüüpi *Class* nimega *B* vahel on ühendaja tüüpi *Realisation*.	*B* on vajalik, et teostada *A*.
Elemendile *A*, mis on ühendatud elemendiga *B*, on antud rollid *C* ja *D* vastavalt.	*A* kui *C* on seotud *B* kui *D*.

Mille kohta on laused?	**Laused**
Element *B* ja *C* kuuluvad elemendi *A* üldistushulka nimega „*A* roll“ ning 1) osaluskohustus on (*Is Covering = True*). 2) osaluskohustust ei ole (*Is Covering = False*).	1) Iga *A* peab omama vähemalt ühte *A* roll tüüpi järgnevatest: *B, C.* 2) Iga *A* ei pea omama *A* roll tüüpi järgnevatest: *B, C*.
Ainult element *B* kuulub elemendi *A* üldistushulka „*A* roll“ ning osaluskohustus on (*Is Covering = True*)	Iga *A* peab omama *A* roll tüüpi *B.*
Element *B* ja *C* kuuluvad elemendi *A* üldistushulka nimega „*A* roll“ ning *A* 1) võib kuuluda kuni ühte alamtüüpi. (*Is Disjoint = True*). 2) võib kuuluda mitmesse alamtüüpi (*Is Disjoint =* False).	1) Iga *A* võib samal ajal omada ainult ühte *A* roll tüüpi järgnevatest: *B, C*. 2) Iga *A* võib samal ajal omada rohkem kui ühte *A* roll tüüpi järgnevatest: *B, C*.
	3)

4 Lahenduse strateegia valik

UML mudeleid saab üle kanda UML tööriistade vahel kasutades XMI (*XML Metadata Interchange*) formaati. Sellest lähtuvalt võiks ka olla võimalik luua üldlahendus, mis võimaldaks genereerida lauseid erinevate XMI formaati toetavate CASE vahendite jaoks. Täpsemalt oleks tegemist tarkvaraga, mis loeb XMI formaadis faile ja loob selle põhjal loomuliku keele laused. Käesolevas töös otsustati seda lahendust mitte kasutada kuna mudeli XMI formaadis salvestamine ja siis lausete generaatorile etteandmine oleks mitmesammuline ja seega aeganõudev tegevus, mis pole kasutusmugav. Probleemiks võib ka osutuda erinevate CASE vahendite XMI failide ühildamatus, sest need on loodud erinevate XMI standardi versioonide alusel. Sellistele probleemidele viitavad ka mudelivahetuse pendeltestimise tulemused [31]. Seega lahendatakse käesolevas töös ülesanne selliselt, et luuakse täiendus konkreetsele CASE vahendile (Enterprise Architect), kasutades ära selle konkreetse CASE vahendi võimalusi.

5 Lahenduse katsetus EA mallide abil

Selle peatükis kirjeldatakse esimest viisi, mille abil üritas autor EA tarkvarast genereerida väljundfaili loomuliku keele lauseid. Selleks muudeti tarkvaras juba olemasolevaid malle skeemidest väljundi genereerimiseks. Autor mõtles, et kuna mallide kasutamise mehhanism on EA vahendisse sisse ehitatud, siis enne täiesti uue lahenduse realiseerimist tuleks proovida, kas ülesande saab lahendada EA olemasolevaid võimalusi kasutades.

5.1 Skeemidest info kättesaamine

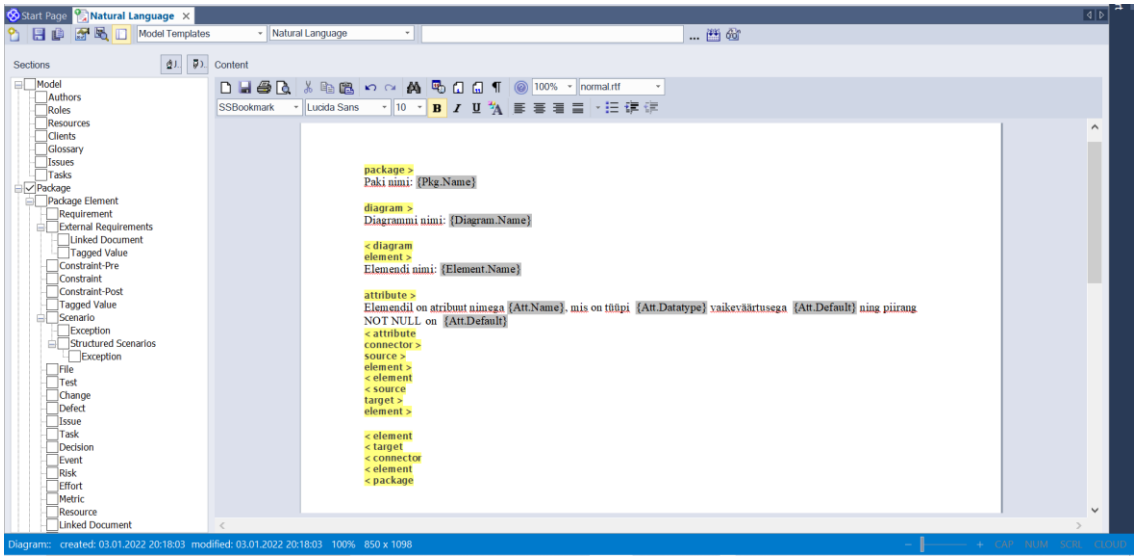
Esimeseks katsumuseks oli EA tarkvarast skeemidel esitatud info kättesaamine, et sellest saaks hiljem luua struktureeritud väljundfaili koos loomuliku keele lausetega. Kuna lõputöö teemaga ei olnud kaasatud täpseid juhiseid ega nõuandeid, mis viisil seda probleemi tuleks lahendada, siis pidi autor ise leidma viisid, kuidas EAst lähteandmeid kätte saada.

Peale esmast uurimist leidis autor, et EAsse on sisse ehitatud kolme erinevat tüüpi malli tugi, mis lasevad klassiskeemidest genereerida dokumentatsiooni, SQL andmekirjelduskeele lauseid ja programmikoodi mõnedes tuntumates programmeerimiskeeltes. Siit tekkiski idee, et kuna kõiki sisseehitatud malle oli võimalik muuta ning isegi luua täiesti uus kasutaja vajadustele kohandatud mall, siis on võimalik kasutada mallide mehhanismi väljundfaili loomuliku keele lausete genereerimiseks.

Järgmine ülesanne oli leida kolmest malli tüübist kõige sobivam, mida saaks uue malli loomisel aluseks võtta ja muuta, et lõpptulemuseks oleksid väljundis loomuliku keele laused. Seega lõi autor iga malli tüübi kohta uue malli ja proovis seda muuta.

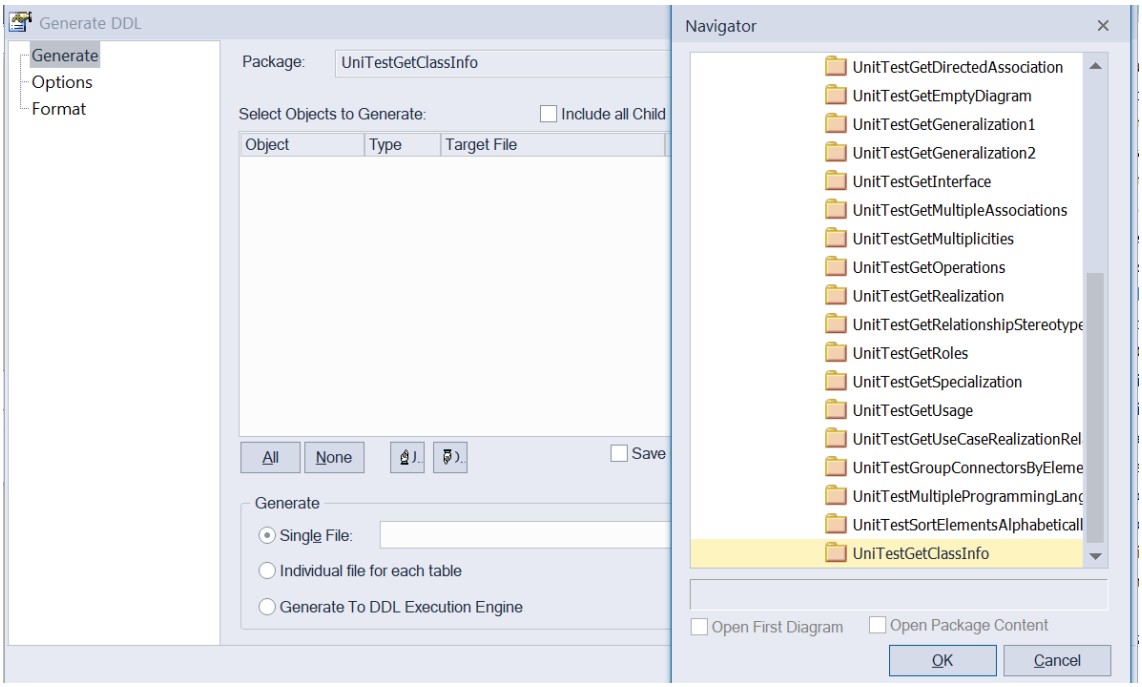
Esiteks proovis autor kasutada dokumendi genereerimise malli, mis on näha Joonisel 2. Mall koosnes vasakul olevast hierarhilisest struktuurist, kus oli võimalik linnukesega märkida millist tüüpi EA tarkvaras defineerituid objekte dokumendis kuvatakse ning nende kuvamisel kasutatav taane vastab vasakul olevale hierarhiale. Peale katseid malli

muuta avastas autor, et mallil on väga kindel struktuur, mida pole võimalik palju muuta ning mall ei luba ligipääsu kõigile vajalikele lähteandmetele.



Joonis 2. Näide EA 12 dokumentatsiooni genereerimise mallist.

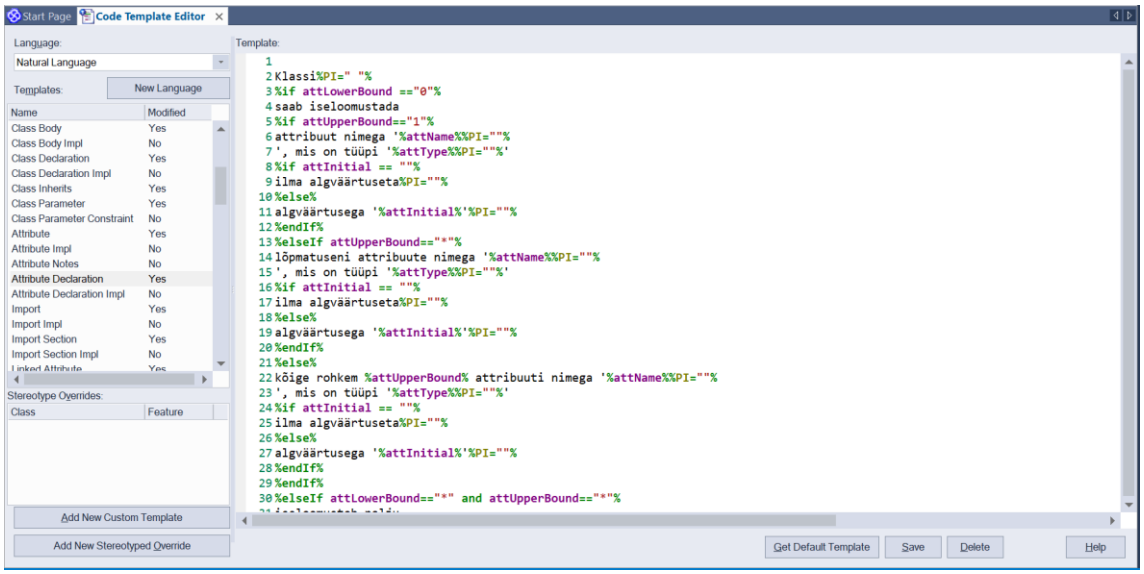
Järgmisena otsustas autor uurida lähemalt võimalusi, mida pakub DDL malli tüüp. Seda tüüpi mallid on mõeldud SQL (*Structured Query Language*) andmekirjelduskeele lausete (CREATE, ALTER) genereerimiseks. Nende lausete andmebaasisüsteemis käivitamisel luuakse andmebaasis andmebaasiobjektid. EA pakub ka võimalust luua täiesti uus enda mall, kuid autor avastas, et mall genereerib väljundit ainult elementidest, mis on tüüpi *Table*, nagu on näha Joonisel 3. Kui autor üritas genereerida väljundit paketist nimega *UnitTestGetClassInfo*, mis sisaldas skeemi ühe elemendiga tüüpi *Class*, siis ei pakutud EA DDLi genereerimise vaates valikut, sellest väljundi genereerimiseks.



Joonis 3. EA DDLi genereerimise vaade. Autor proovib genereerida väljundit paketist nimega *UnitTestGetClassInfo*.

EA pakub ka kolmandat malli tüüpi, mille mallid genereerivad skeemidel olevast infost programmikoodi tuntud programmeerimiskeeltes nagu näiteks Java, C, C#, Python. EA pakub ka sel puhul võimalust luua täiesti uus kasutaja vajadustele kohandatud mall. Seekord oli uue malli loomise järel võimalik ka hiljem sellest väljundit genereerida. Edasi otsustas autor võtta aluseks Java koodi genereerimise malli ning luua uus mall, muutes Java malli nii, et väljund oleks loomuliku keele laused.

Koodimallid said mudelist vajalikud lähteandmed kasutades EAsse sisseehitatud makrosid, mis kujutasid endast protsendimärkide vahel olevaid kindlaid termineid. EA kodulehel on väljatoodud nimekiri kõikidest võimalikest koodimalli makrodest. Koodimall koosneb omakorda paljudest alammallidest, millest igaühel on oma ülesanne. Näiteks alammalli fail nimega *Attribute Declaration* vastutab selle eest, millise struktuuri ja sõnastusega kuvatakse väljundfailis iga elemendi tüüpi *Class* atribuutide kirjeldused. Joonis 4 näitab, milline on koodimalli muutmise vaade. Aknas *Template* on näha autori poolt muudetud *Attribute Declaration* alammall, mida on muudetud nii, et see vastaks inimkeelsele struktuurile ja sõnastusele.



Joonis 4. EA koodimalli muutmise vaade. Avatud on alammall nimega *Attribute Declaration*.

Jätkates uue malli loomist hakkasid ilmsiks tuleb puudused, tänu millele ei saa luua jaotises 3.1 esitatud nõuetele vastavad väljundfaili.

Esiteks genereerib EA iga mudeli elemendi kohta eraldi faili. Kui genereerida väljundit lauseid juba sisaldavasse faili, siis EA lisab genereeritud laused faili lõppu vaid juhul kui failis eelnevalt olevad laused on samas programmeerimiskeeles kui see mida genereeritakse. EAsse on sisseehitatud olemasolevate mallide programmeerimiskeelte parser e sõeluja e süntaksianalüsaator. Kuna EAl puudub loomuliku keele parser, siis faili lisatav väljund ei ole kunagi samas keeles kui juba failis sisalduv info, mistõttu iga elemendi kohta genereeritud laused pannakse eraldi faili.

Teiseks on olemasolev koodimall mõeldud vaid *Class* tüüpi elementide kirjeldamiseks. Ülejäänud tüüpi elementide kirjeldamiseks tuleb igas alammallis luua tingimuslausetega eraldi omad reeglid erinevat tüüpi elementide vastavate osade kirjeldamiseks. Näiteks kui tahta genereerida koodimallist infot elemendi tüüpi *Interface* kohta, siis tuleb igas alammallis tingimuslausetega määrata, millist infot genereerida kui elemendi tüüp on *Interface* mitte *Class*. Teine võimalus on valida koodimalli vaates *Add New Stereotype Override*, mis lubab igale erinevale elemendi tüübile luua oma hulga alammalle.

Kolmandaks ei ole võimalik ligi pääseda kõigile olulistele andmetele, mis on skeemil esitatud. Näiteks kui on kahe elemendi vahel ühendaja, mille küljes on element tüüpi *Note* või *Constraint*, siis ei leidu makrot, mis annaks infot, et just see *Note* element on selle ühendaja küljes. Samuti on jaotises 3.1 välja toodud, et laused, mis kirjeldavad *Note* tüüpi

elemente, mis on skeemi alguses ning pole ühendatud ühegi teise elemendiga, tuleks väljundfailis esitada esimesena. Paraku puudub EAs makro, mille abil saada kätte vajalik info elemendi asukoha kohta skeemil.

Neljandaks, kuna igas mallis saab muuta ainult elemendi sisest infot, siis pole võimalik malli abil genereerida väljundit, kus kõigepealt on kõikide klasside kirjeldused ning seejärel nende klasside ühendajate kirjeldused grupeerituna klasside kaupa klassi nimede järgi tähestikulises järjekorras nagu oli üheks nõudeks jaotises 3.1. Samuti ei saa genereerida lauseid mille koostamiseks on vaja infot paljude elementide kohta nagu on vaja näiteks üldistushulki kirjeldavate lausete loomisel.

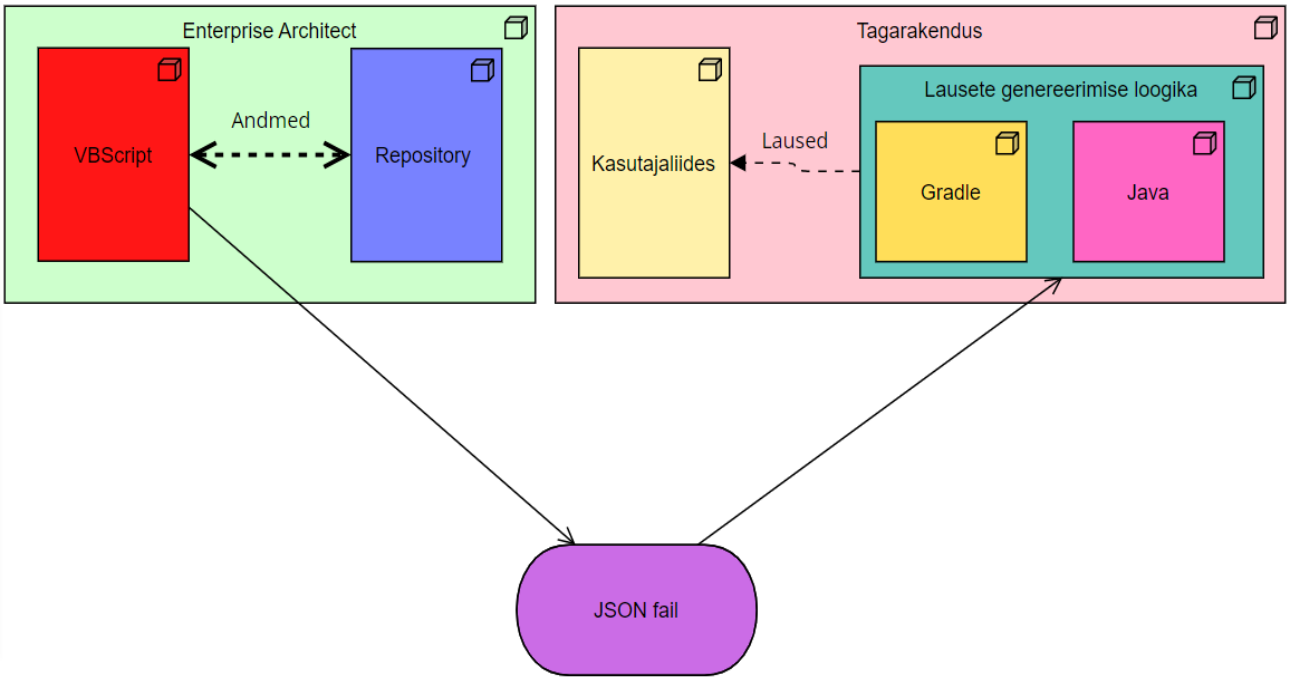
Viiendaks, kuna genereeritavate lausete keel peab olema muudetav ja peab saama lisada uute keelte tuge, siis peavad eksisteerima globaalsed muutujad, mille väärtusteks on sõnad ja väljendid, mida saab kõikides alammallides kasutada. Kui on vaja muuta või täiendada keelt, siis peab vaid nende muutujate väärtuseid muutma ning seejärel saab genereerida uued laused uues keeles. EA koodimallis ei saa deklareerida globaalseid muutujaid, mis teeb keele lisamise ja muutmise äärmiselt vaevaliseks, sest igas alammallis tuleks selleks muuta vajalikke sõnu ja termineid.

Seega alguses paljulubav tundunud mallide abil lausete genereerimise meetod ei pakkunud piisavalt võimalusi, et selle abil oleks võimalik lõputöös püstitatud probleem piisavalt hästi lahendada.

6 Rakenduse disain ja arhitektuur

Käesolevas peatükis kirjeldab autor teist meetodi EAst lähteandmete kättesaamiseks, mis hõlmab endas EAs skripti loomist ning saadud info JSON (*JavaScript Object Notion*) formaadis faili kirjutamist. Samuti kirjeldab peatükk kasutajaliidese disaini ning funktsionaalsust ja tagarakenduse arhitektuuri.

Kogu tarkvara koosneb skriptist, mis kirjutab kasutaja EA projektist JSON vormis andmed faili. Tagarakendus on kirjutatud Java programmeerimiskeeles ning kasutab *Gradle*-i Java teekide haldamiseks ning koodi kompileerimiseks. Rakendus loeb info JSON vormis failist sisse ning kasutab sisendit, et luua loomuliku keele laused. Seejärel kuvab kasutajaliides laused kasutajale. Kasutajaliides pakub ka uue keele lisamise ja keele kustutamise funktsionaalsust, samuti võimalust tõlkida lauseid nendesse keeltesse, mis on kasutaja pool lisatud. Tarkvara kogu arhitektuur on näha Joonisel 5.



Joonis 5. Tarkvara arhitektuur.

Skripti füüsiliste koodiridade arv 04.01.2022 aasta seisuga on 261.

Tagarakenduse füüsiliste koodiridade arv 04.01.2022 aasta seisuga on 3009.

6.1 EAst lähteandmete kättesaamine kasutades skripti

EAs on võimalik luua skripte kolmes skriptimiskeeles *VBScript*, *JScript* ning *JavaScript*. Nende skriptimiskeelte abil on EAs võimalik pääseda ligi kogu infole mudelite kohta, mis on kasutaja avatud EA projektides. Kasutades neid skriptimiskeeli saab kasutaja teha päringuid EA sisesesse andmebaasi läbi sisseehitatud *Repository* klassi meetodite. See on põhiline konteiner projekti mudelitele, pakettidele, skeemidele ning elementidele [32].

Järgmise meetodina EAst vajalike lähteandmete kättesaamiseks otsustas autor kasutada *VBScript*-i, et teha päringuid EA sisesesse andmebaasi läbi *Repository* klassi meetodite ning kirjutada saadud andmed JSON formaadis objektidena tekstifaili.

Autor valis skriptimiskeeleks *VBScript*-i, sest *VBScript*-is on võimalus teha uute muutujate deklareerimine enne kasutamist kohustuslikuks. See on kasulik, sest kui nüüd muutuja nime kirjutamisel koodis teha viga, siis see tuleb skripti käivitamisel välja ja põhjustab erandi. Kui muutujate deklareerimine pole kohustuslik, siis eeldab skripti käivitav programm, et kirjaveaga muutuja nimi tähistab uut muutujat. See aitab parandada koodi kvaliteeti. Samuti leiab palju abistavaid koodinäiteid EA kohta *VB* ja *VBScript* keeltes.

Raskendav asjaolu *VBScript*-i puhul oli see, et autor ei olnud sellega varem kokku puutunud ning pidi selle skriptimiskeele võimalusi ja eripärasid iseseisvalt õppima.

EAs saab valida enne skriptifaili loomist, millisesse skriptigruppi see kuulub. Autor valis *Project Browser Group*-i, sest selle grupi skriptid on käitatavad projekti failipuus olevate skeemide ja pakettide korral, mille põhjal on vaja lauseid genereerida.

EAs on klassil *Repository* meetod, mille abil on võimalik saada EA objekt, millele kasutaja *Project Browser*-is vajutas. See objekt on seda tüüpi (pakett või skeem), millele kasutaja *Project Browser*-is vajutas. Nendel objektidel on omakorda meetodid saamaks infot neid sisaldavate EA objektide kohta (paketid, skeemid, elemendid, ühendajad).

Skriptis on funktsioon nimega *generateJsonFromAnDiagram()*. Kui kasutaja vajutab *Project Browser*-is näiteks skeemile, siis küsitakse läbi *Repository* klassi EA skeemi objekt. See objekt antakse sisendina *generateJsonFromADiagram()* funktsioonile. Funktsiooni sees rakendatakse EA skeemi objektile meetodit, millega saadakse kätte kõik EA elemendi objektid, mis on selle skeemi objekti sees. Kõik EA elemendi objektid

käiakse iteratiivselt läbi ning neile rakendatakse omakorda meetodeid EA atribuudi ja EA ühendaja objektide saamiseks selle EA elemendi objekti seest. Seejärel käiakse iteratiivselt läbi iga EA ühendaja tüüpi objekt ning EA atribuuti tüüpi objekt, mis on seotud EA elemendi tüüpi objektiga. Kõik info, mis on lausete genereerimiseks vajalik, kirjutatakse JSON vormingus faili.

Skriptis on lisaks veel funktsioon nimega *generateJsonFromAPackage()*, mis saab sisendina EA paketi objekti. EA paketi objektile rakendatakse meetodit, mis annab kõik EA skeemi objektid, mis on selles EA paketi objektis. Seejärel antakse iga skeemi objekt sisendina *generateJsonFromADiagram()* funktsioonile, et funktsioon kirjutaks iga paketis oleva skeemi andmed JSON vormis faili. Järgnevalt kirjutatakse JSON vormis faili sisendina saadud paketi nimi. Seejärel rakendatakse sisendina saadud EA paketi objekti peal meetodit, mis annab väljundina iga EA paketi objekti, mis on sisendobjekti sees. Seejärel käiakse iteratiivselt läbi iga väljundina saadud EA paketi objekt ning antakse see objekt rekursiivselt *generateJsonFromAPackage()* funktsioonile sisendiks. Rekursiivne funktsiooni kutsumine laseb sisendina saadud EA paketi objektist saada kätte kõikide seda sisalduvate skeemide ja pakettide info olenemata kui suur on üksteise sees olevate pakettide ja skeemide tasemete arv.

Järgmisena kirjeldab autor JSON faili skeemi.

JSON failis kõrgeima taseme objekt on EA juurpaketi objekt, millest kasutaja lauseid soovis genereerida. EA juurpaketti iseloomustab JSON failis objekt, millel on väli *RootPackage*, mille väärtuseks on EA juurpaketi nimi ning väli *Packages*, mille väärtuseks on nimekiri EA juurpaketis olevaid EA pakette JSON objektidena ning ka EA juurpakett ise JSON objektina.

EA pakette iseloomustab JSON failis objekt, millel on väli *packageName*, mille väärtuseks on selle EA paketi nimi, väli *note*, mille väärtuseks on EAs sellele paketile lisatud märge ning väli *diagrams*, mille väärtuseks on nimekiri EA skeeme JSON objektidena, mis on selles EA paketis. Visuaalselt on seda kujutatud Joonisel 6.

Joonis 6. JSON faili objektide näide EA juurpaketi tasandist EA skeemi tasandini.

EA skeemide kohta on JSON failis objekt, millel on väli *diagram*, mille väärtus on selle EA skeemi nimi. Väli *note*, mille väärtuseks on sellele EA skeemile lisatud märge. Väli *elements*, mille väärtuseks on nimekiri EA elemente JSON objektidena, mis on sellel EA skeemil. Väli *smallestClassYCoord*, mille väärtuseks on EA skeemil kõige väiksema Y-koordinaadiga elemendi, mis on tüüpi *Class*, *Interface* või *UseCase* Y-koordinaat. Väli *generalizationSetString*, mille väärtuseks on *String*, mis sisaldab terve projekti üldistushulkade infot. EA skeem võib olla JSON failis kõrgeima taseme JSON objekt, kui kasutaja soovib genereerida lauseid ainult EA skeemist. Visuaalselt on seda kujutatud Joonisel 7.

Joonis 7. JSON objekt failis EA skeemi objekti kohta.

EA elemente, mis on tüüpi *Class*, *Interface*, *UseCase* iseloomustab JSON failis objekt, millel on väli *elementId*, mille väärtuseks on selle EA elemendi unikaalne identifikaator. Väli *type*, mille väärtuseks on selle EA elemendi tüüp (*Class*, *Interface*, *UseCase*). Väli *name*, mille väärtuseks on selle EA elemendi nimi. Väli *note*, mille väärtuseks on sellele EA elemendile lisatud märge. Väli *connectors*, mis on nimekiri selle elemendi EA ühendajatest JSON objektidena. Väli *attributes*, mille väärtuseks on nimekiri selle

elemendi EA atribuutidest JSON objektidena. Väli *methods*, mis on nimekiri selle elemendi EA operatsioonidest JSON objektidena. Visuaalselt on seda näha Joonisel 8.

EA elementide, mis on tüüpi *Note* või *Constraint* ning ühendatud EA ühendajaga iseloomustab JSON failis objekt, millel on väli *type*, mille väärtuseks on EA elemendi tüüp. Väli *note*, mille väärtuseks on selle EA elemendi kirjalik sisu. Väli *connectedTo*, mille väärtuseks on selle EA ühendaja unikaalne identifikaator, millega see element on ühendatud. Seda on näha Joonisel 8.

EA elementide, mis on tüüpi *Note*, kuid ei ole ühendatud ühegi EA elemendiga ega EA ühendajaga iseloomustab JSON failis objekt, millel on väli *type*, mille väärtuseks on alati '*diagramNote'*. Väli *note*, mille väärtuseks on selle EA elemendi märge. Väli *y_coord*, mille väärtuseks on selle EA elemendi Y-koordinaadi väärtus. Seda on näha Joonisel 8.

EA ühendajaid iseloomustab JSON failis objekt, millel on väli *sourceId*, mille väärtuseks on EA elemendi, mis on selle ühendaja alguses, unikaalne identifikaator. Väli *endId*, mille väärtuseks on EA elemendi, mis on selle ühendaja lõpus, unikaalne identifikaator. Väli *sourceName*, mille väärtuseks on selle EA ühendaja algus elemendi nimi. Väli *endName*, mille väärtuseks on selle EA ühendaja lõpu elemendi nimi. Väli *type*, mille väärtuseks on selle EA ühendaja tüüp (nt *Association*, *Aggregation*). Väli *stereotype*, mille väärtuseks on selle EA ühendaja stereotüüp. Väli *startMulitplicity*, mille väärtuseks on selle EA ühendaja alguse võimsustik. Väli *endMultiplicity*, mille väärtuseks on selle EA ühendaja lõpu võimsustik. Väli *connectorId*, mille väärtuseks on selle EA ühendaja unikaalne identifikaator. Väli *connectorName*, mille väärtuseks on sellele EA ühendajale antud nimi. Väli *targetRole*, mille väärtuseks on selle EA ühendaja lõpu elemendi roll. Väli *sourceRole*, mille väärtuseks on selle EA ühendaja algus elemendi roll. Väli *note*, mille väärtuseks on sellel EA ühendajale lisatud märge. Seda on näha Joonisel 8.

Kui EA ühendaja tüüp on *NoteLink*, siis on JSON objektis veel lisaks ka väli *connectedNote* või *constraint*, mille väärtuseks on EA elemendi tüüpi *Note* või EA elemendi tüüpi *Constraint* sisu vastavalt, olenevalt sellest milline element on selle ühendaja küljes. Seda on näha Joonisel 9.

EA atribuute iseloomustab JSON failis objekt, millel on väli *name*, mille väärtuseks on selle EA atribuudi nimi. Väli *type*, mille väärtuseks on selle EA atribuudi andmetüüp. Väli *initialValue*, mille väärtuseks on sellele EA atribuudile antud algväärtus. Väli

lowerBound, mille väärtuseks on selle EA atribuudi võimsustiku alampiir. Väli *upperBound*, mille väärtuseks on selle EA atribuudi võimsustiku ülempiir. Väli *note*, mis on selle EA atribuudile lisatud märge. Seda on näha Joonisel 8.

EA operatsioone iseloomustab JSON failis objekt, millel on väli *name*, mille väärtuseks on selle operatsiooni nimi. Väli *note*, mis on sellele EA operatsioonile lisatud märge. Seda on näha Joonisel 8.

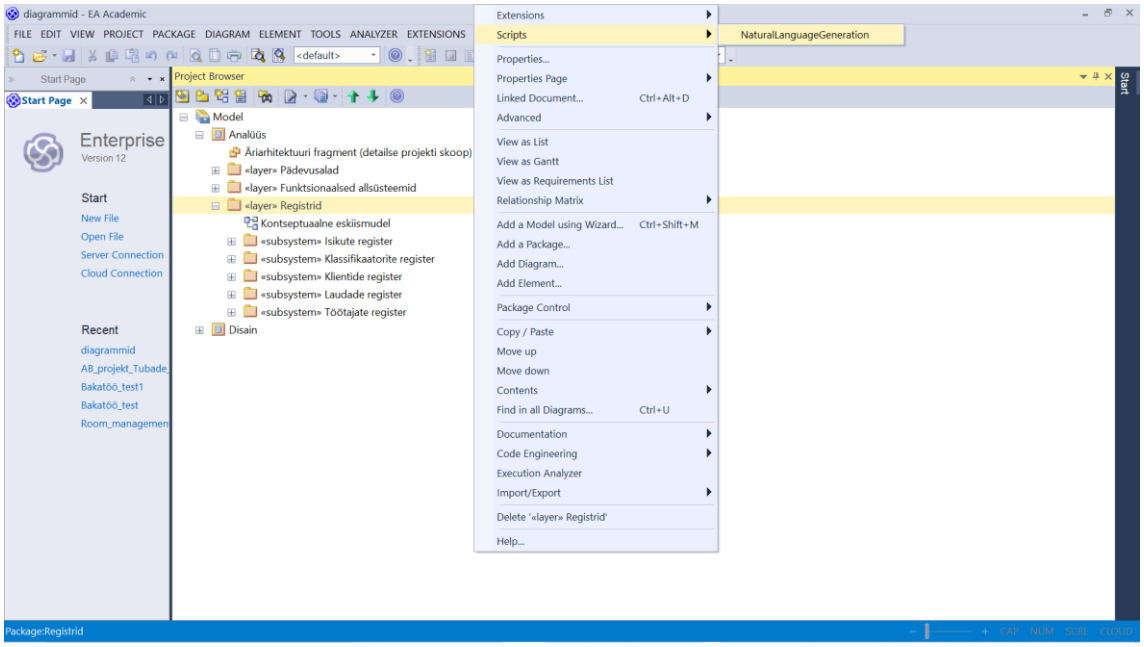

```
{"elementId": "416", "type" : "Class", "name": "Person", "note": "Class note",
"connectors" : [
{
"sourceId": "417", "endId": "416", "sourceName": "Car", "endName": "Person", "type": "Association",
  "stereotype": "", "startMultiplicity": "0..*", "endMultiplicity": "1..*", "connectorId": "275",
  "connectorName": "", "targetRole": "Driver", "sourceRole": "","note":""
}
],
"attributes" : [
{"name":"ID-code", "type": "int", "initialValue":"", "lowerBound": "1", "upperBound": "1","note":"Attribute note"
},
{"name":"firstname", "type": "String", "initialValue":"", "lowerBound": "1", "upperBound": "1","note":""
},
{"name":"lastname", "type": "String", "initialValue":"", "lowerBound": "1", "upperBound": "1","note":""
},
{"name":"children", "type": "int", "initialValue":"", "lowerBound": "0", "upperBound": "*","note":""
}
],
"methods" : [
{"name":"driving","note":"Method note"
```


Joonis 8. EA tüüpi *Class* elemendi objekti esitus JSON objektina failis.


```
{
"sourceId": "264", "endId": "242", "sourceName": "", "endName": "Class1", "type": "NoteLink",
  "stereotype": "", "startMultiplicity": "", "endMultiplicity": "", "connectorId": "251", "connectorName": "",
  "targetRole": "", "sourceRole": "","note":""
, "constraint": "Constraint 3"
},
{
"sourceId": "320", "endId": "242", "sourceName": "", "endName": "Class1",
  "type": "NoteLink", "stereotype": "", "startMultiplicity": "", "endMultiplicity": "",
  "connectorId": "274", "connectorName": "", "targetRole": "", "sourceRole": "","note":""
, "connectedNote": "Note pointing to Class 1"
}
```


Joonis 9. EA ühendajate tüüpi *NoteLink* esitus JSON objektidena.

Kui kasutaja vajutab *Project Browser*-is olevale juurpaketile, ning paneb skripti tööle nagu on näha Joonis 10, siis kirjutatakse selle juurpaketi nimi JSON vormis faili, antakse see EA paketi objekt sisendina *generateJsonFromAPackage()* funktsioonile, mis kutsub end rekursiivselt välja, kuni kogu info juurpaketi ja selles sisalduva kohta on JSON vormis failis.



Joonis 10. Näide EAs skripti käivitamisest ühel juurpaketil.

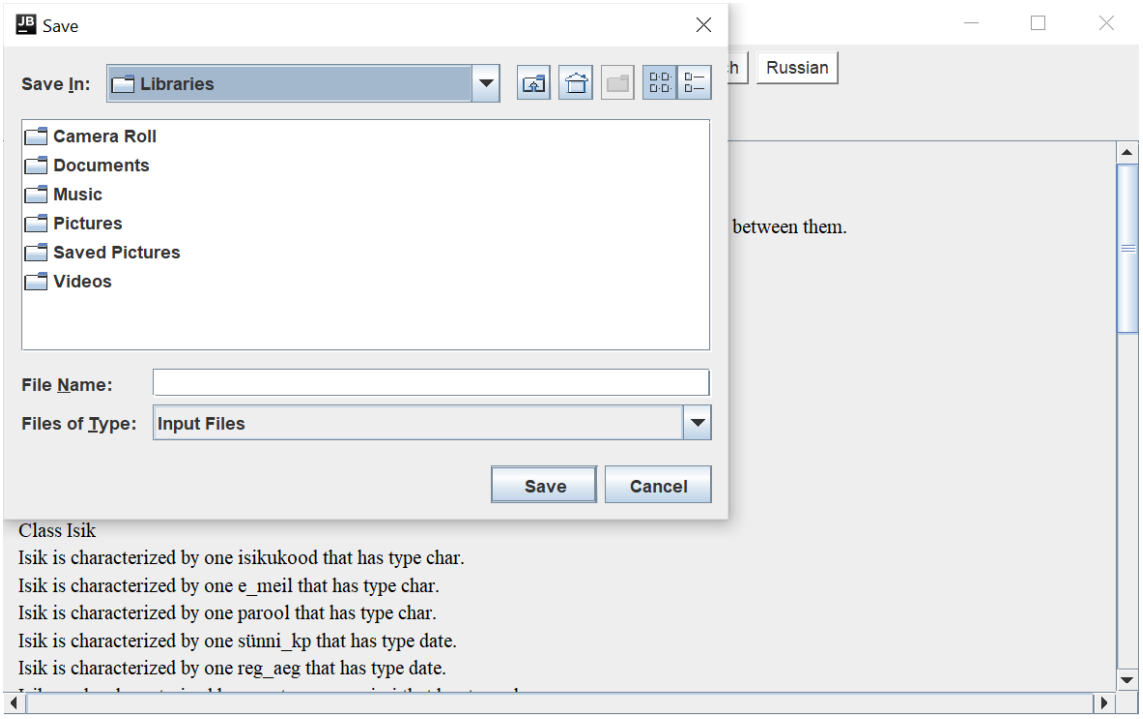
Vaikimisi kirjutab skript andmed *Windows* operatsioonisüsteemi puhul kausta nimega *Temp*, mis on omane igale masinale, kus on *Windows*-i operatsioonisüsteem ning kuhu paljud rakendused kirjutavad ajutist infot. Selles kaustas luuakse fail nimega *file.json* ning kirjutatakse sinna skripti väljund JSON vormis. Kasutaja saab skripti sees ise muuta rida, mis vastutab selle eest, millisesse faili andmed kirjutatakse. Tarkvara installeerimisest ja kasutamisest kirjutab autor täpsemalt peatükis 7. *VBScript* kirjutab, andmed ANSI kodeeringus, mis on omane *VBScript*-le

6.2 Kasutajaliides

Selles jaotises kirjeldatakse loomuliku keele lauseid kuvava kasutajaliidese funktsionaalsust ning väljanägemist. Kasutajaliides on loodud kasutades Java programmeerimiskeelega kaasa tulevat AWT-d (*Abstract Window Toolkit*) ja *Swing* teeki. Mõlemad teegid on loodud Java programmidele akendamise, graafika ja kasutajaliideste loomise jaoks.

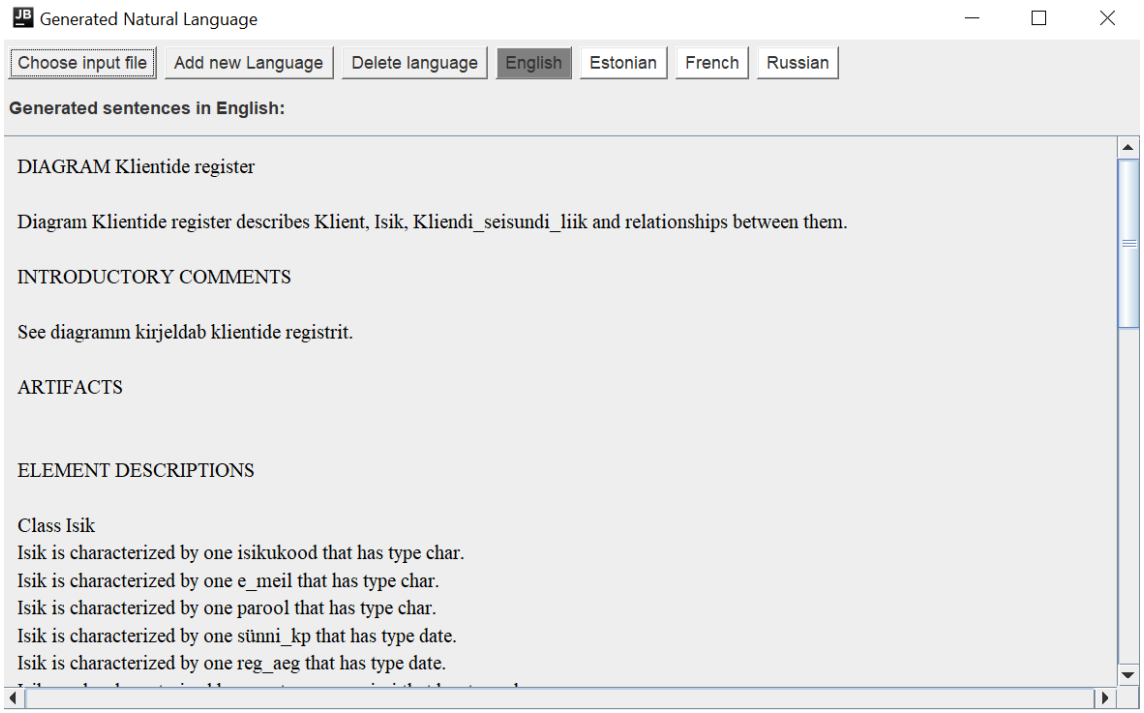
Kasutajaliidesel on lihtne disain, mis koosneb kolmest põhivaatest. Avavaade kuvatakse programmi käima panemisel. Sellel vaatel näeb kasutaja oma EA projektist genereeritud loomuliku keele lauseid. Samuti on vaate ülaservas tööriba, mis laseb kasutajal liikuda vaadete vahel ning muuta genereeritud lausete keelt.

Tööriba esimene nupp on tekstiga *Choose input file*, mis laseb kasutajal graafiliselt valida, millisest failist loetakse skripti JSON väljund nagu on näha Joonis 11. Nagu eelmises jaotises mainitud, on valiku põhjuseks see, et skriptis saab muuta JSON väljundi kirjutamise asukohta. Tarkvara töötab vaid siis kui skripti JSON väljundfaili asukoht ühtib liideses valitud faili asukohaga. See lisab tarkvarale paindlikust ja kasutaja saab ise otsustada, millise faili läbi käib tema süsteemis andmevahetus skripti ja Java tagarakenduse vahel.

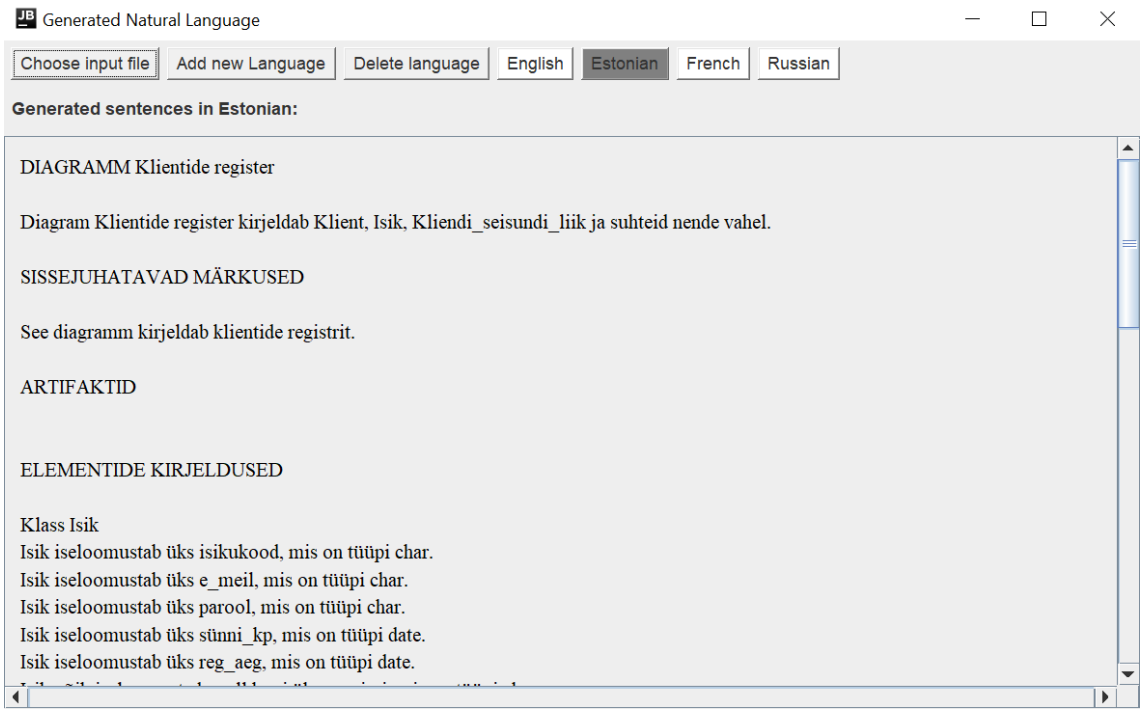


Joonis 11. Graafiline faili valimis funktsionaalsus kasutajaliideses.

Tööriba teine nupp on tekstiga *Add new language*, mis viib kasutaja järgmisesse vaatesse, kus saab lisada uue keele. Kolmas nupp tekstiga *Delete language* viib kasutaja samuti uude vaatesse. Selles vaates saab kasutaja poolt lisatud keeli kustutada. Peale neid nuppe on tööribal väljatoodud valik kasutaja pool juba lisatud keeltest, millele vajutades tõlgitakse kõik genereeritud laused just sellesse keelde. Nuppude all järgmisel real on kirjas, millises keeles on kuvatud laused. Kogu avavaate disain koos ingliskeelsete lausetega on Joonisel 12 ja eestikeelsete lausetega on Joonisel 13.

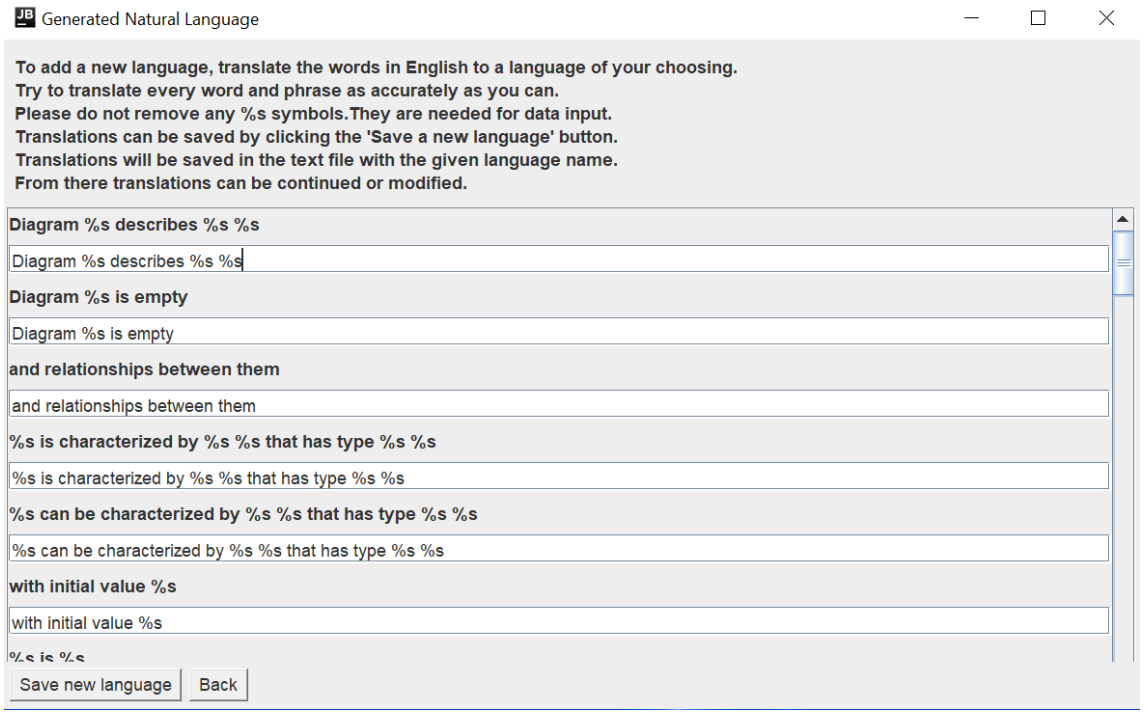


Joonis 12. Kogu kasutajaliidese avavaade. Laused on inglise keeles.



Joonis 13. Kogu kasutajaliidese avavaade. Laused on eesti keeles.

Vajutades nupule *Add a New Language*, kuvatakse kasutajale keele lisamise vaade, mida on näha Joonis 14. Vaate põhiosas on kordamööda tekst, kus on kas inglise keelne sõna termin või lause ja tekstiväli, mis sisaldab täpselt sama teksti. Vaate põhiosa on mall, mille järgi saab luua keelefaile kasutaja soovitud keeles.



Joonis 14. Kasutajaliidese keele lisamise vaade.

Tekstiväljade kohal olev tekst tuleb tõlkida võimalikult täpselt soovitud keelde. Tekst sisaldab sümboleid kujul '%s' (kohahoidjaid, parameetreid), mida ei tohi tõlkides eemaldada, sest sinna sisestatakse EAst tulevad andmed. Peale kõikide lausete tõlkimist saab keele salvestada keelefaili vajutades vaate all ääres olevale nupule *Save new language*. Seejärel kuvatakse kasutajale hüpikaken, kus ta saab määrata keele nime, mis on kujutatud Joonis 15. Tõlgitud laused salvestatakse keelefaili, Java tagarakenduse kausta – selle nimega, mille kasutaja määras keele nimeks. Keelefail on tekstifail.

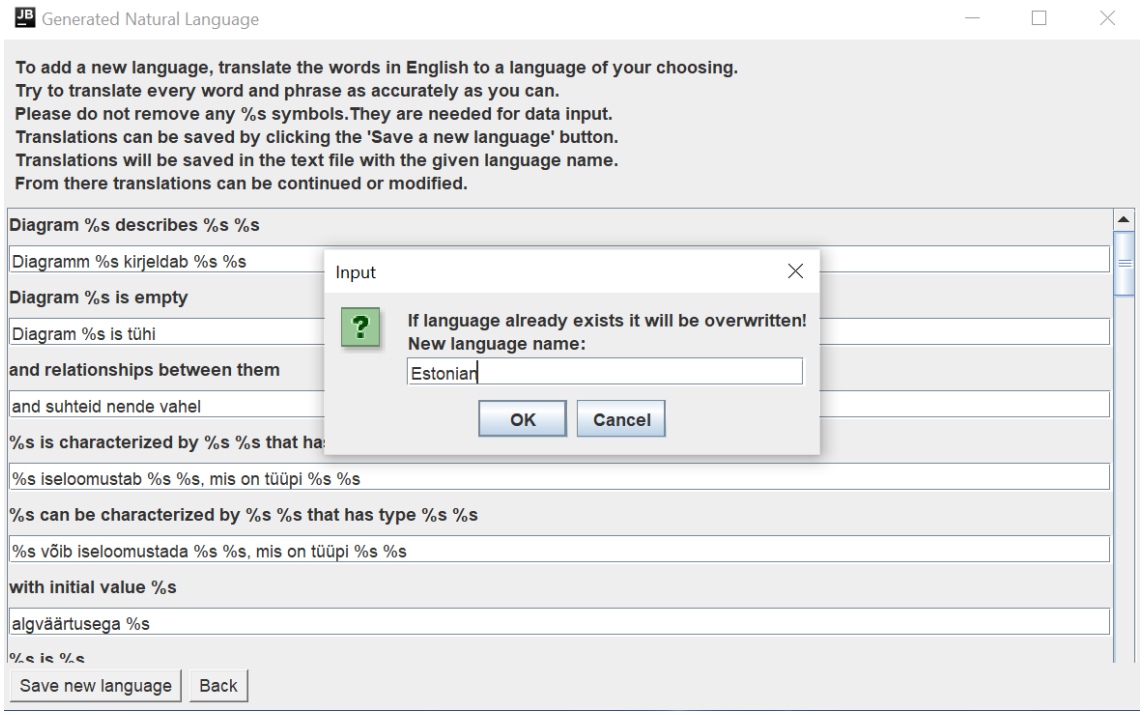
Ingliskeelsete lausete mallid on defineeritud Java klassis (lähtekoodis). Kui soovida parandada neid lauseid ilma lähtekoodi muutmata, siis tuleb luua uus keelefail koos parandatud/täiendatud lausetega. Selle keelefaili nimi ei saa olla *English* – tarkvara keelab sellise nimega keele loomise. Keelefaili (keele) nimi võib näiteks olla *English (modified)* ja see käitub rakenduses nagu iga teine keel, millel on keelefail.

Kui määratakse keelefailiga keele nimeks selline nimi, mis juba eksisteerib, siis vana keelefail kirjutatakse üle.

Vaate üleval ääres on ka täpsed juhised selle kohta, kuidas malli tuleb täita. Vaate all ääres on ka nupp tekstiga *Back.* Nupp viib kasutaja tagasi kasutajaliidese avavaatesse. Ülesse tööribale on nüüd tekkinud uus keelevalik. Vajutades sellel valikul tõlgitakse avavaates kuvatavad laused just sellesse keelde.

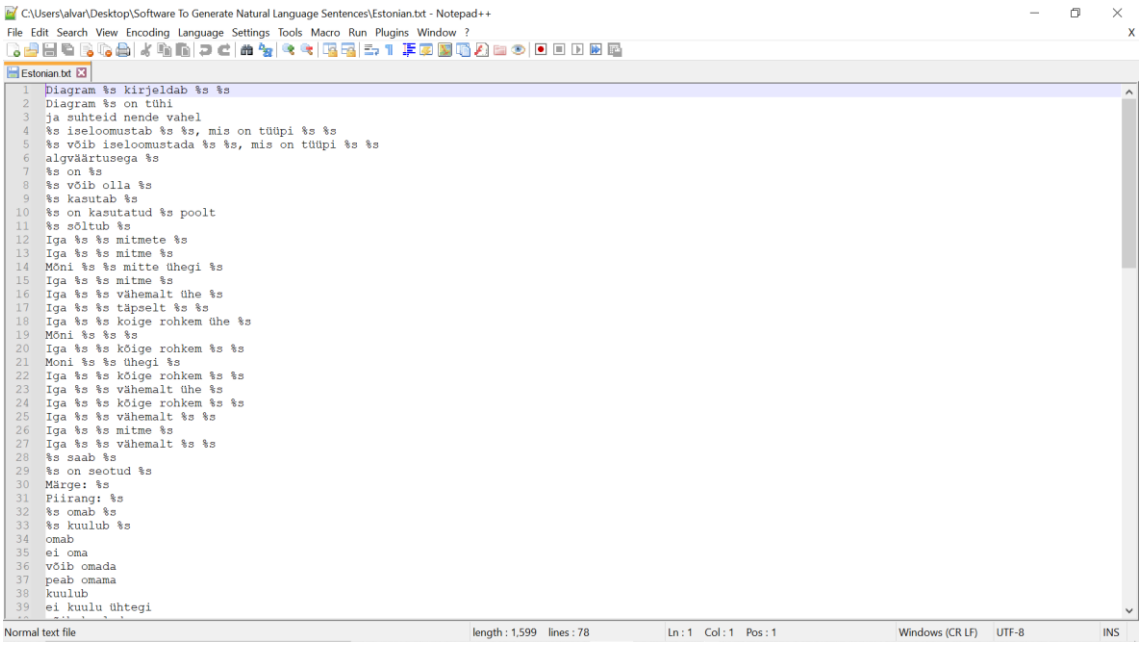
Uue keele kasutuselevõtuks võib ka kopeerida tagarakenduse kausta mõnes teises arvutis loodud keelefaili (näiteks keegi koostas soomekeelsed laused ja kasutaja tahab ka enda arvutis selles keeles lauseid genereerida). Eeldus on, et keelefail loodi sama rakenduse versiooni kasutades. Sellisel juhul ei ole graafilises kasutajaliideses vaja uue keele kasutuselevõtuks midagi teha.

Keelemallis uue keele lisamisel toetab rakendus kõiki sümboleid, mis kuuluvad UTF-8 kodeeringusse.



Joonis 15. Keelte lisamise vaade koos keele nimetamise hüpikaknaga.

Kui kasutaja on lisanud uue keele ning peale sellega töötamist avastab, et esmased tõlked ei olnud kõige paremad ning tahaks lausete struktuuri ja sõnastust muuta, siis on selleks hiljem võimalus teha muudatus keelefailis, mille sisu on näha Joonis 16. Java tagarakendus loob keelefaili alati samasse kausta, kus ise asub.



Joonis 16. Keelefaili Estonian.txt sisu.

Kasutajaliidese viimane vaade on keelte kustutamise vaade. Kasutaja saab liikuda sellele vaatele, vajutades *Delete language* nuppu. Kuvatakse vaade, kus vasakul pool on üksteise all kasutaja keelefailide nimed, ning paremal pool on üksteise all punased nupud tekstiga *Remove*. Kui kasutaja vajutab *Remove* nupule, siis kustutatakse keelefail, mille nimi on nupuga samal real vaate vasakus ääres. Keelte kustutamise vaade on näha Joonis 17.

Joonis 17. Kasutajaliidese keelte kustutamise vaade.

Kui kasutaja vajutab keelte kustutamise vaate all ääres asuvat *Back* nuppu, siis kuvatakse kasutajaliidese avavaade, kuid kustutatud keele valik on nüüd avavaate üleval tööribal olevast keelte valikust kadunud.

Keelt saab kustutada ka ilma graafilise kasutajaliideseta – kustutades tagarakenduse kataloogist keelefaili. Keelt *English* ei ole võimalik kustutada – ei graafilise kasutajaliidese kaudu ega ka keelefaili kustutades, sest selle keele lausete struktuur on defineeritud Java klassis, mitte keelefailis.

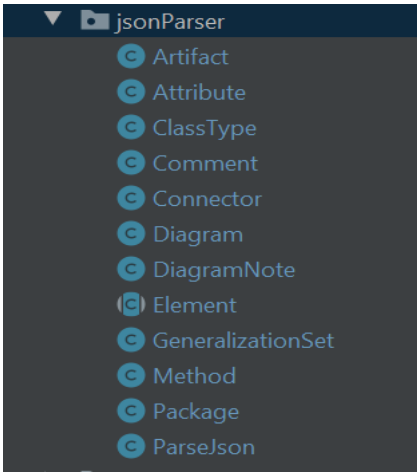
6.3 Java tagarakendus

Tagarakendus on kirjutatud Java keeles. Kuna Java keel on väga populaarne, on sellele loodud palju teeke, mis võimaldavad keerulisi ülesandeid lihtsustada. Samuti on autoril Java keelega olnud kokkupuudet ning seetõttu on ta teadlik keele pakutavatest võimalustest probleemide lahendamisel ning keele varasem tundmine tõstab arenduse kiirust. Selles jaotises on kirjeldatud lähemalt Java tagarakenduse ülesehitust ja toimimispõhimõtteid.

Tagarakendus on jagatud kolmeks paketiks nimedega *jsonParser*, *sentenceGenerator* ning *applicationUI*. Igal paketil on enda ülesanne. *jsonParser* pakett vastutab selle eest,

et *VBScript*-i poolt loodud JSON faili sisu tõlgitakse Java objektideks. *sentenceGenerator* vastutab selle eest, et loodud Java objekte kasutades luuakse kasutajale loetavad loomuliku keele laused. *applicationUI* loob kasutajaliidese, millesse kuvatakse loodud loomuliku keele laused. Samuti on eraldiseisva osana *Main.java* klass, mis loob kõik vajalikud objektid, et JSON failist sisse loetud sisendit lõpuks loomuliku keele lausetena kasutajaliideses kuvada.

Esmalt kirjeldab autor *jsonParser* paketi sisu, mis on näha Joonis 18. Projekti *build.gradle* failis, mis on *Gradle*-i konfiguratsiooni skripti fail, on lisatud teek nimega *com.fasterxml.jackson.core*. *Gradle* lisab automaatselt selle teegi failid autori projekti kausta. Teek aitab tõlkida failis oleva JSON kujul info Java objektideks.

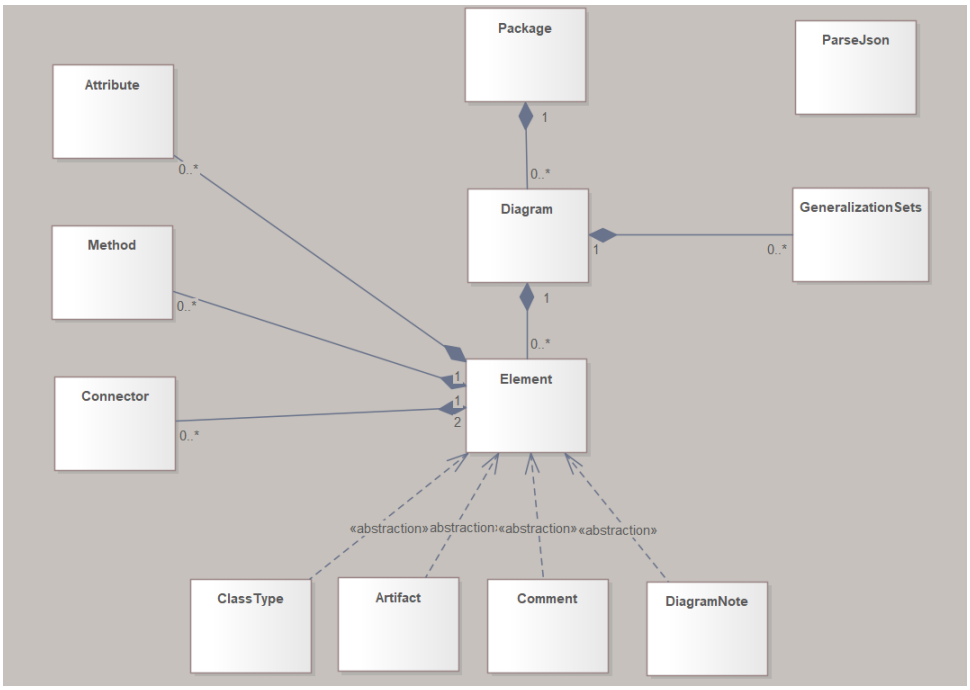


Joonis 18. Paketi *jsonParser* sisu.

Paketis *jsonParser* on klass nimega *Package*, millel on väli selle EA paketi nime jaoks ja Java *List*, mis sisaldab Java objekte *Diagram*. Klass *Diagram* on samuti defineeritud paketis, mis sisaldab EA skeemi objekti infot. Veel on defineeritud abstraktne klass *Element*, mille väljadeks on iga EA elemendi ühisosa ehk väljad, mis on igal EA elemendil nagu näiteks elemendi unikaalne *id*, elemendi tüüp, elemendi märgete väli.

Seejärel on defineeritud klassid, mis pärivad abstraktset klassi *Element*. Neid on neli nimedega *ClassType*, *Artifact*, *Comment* ja *DiagramNote*. Klassi *ClassType*, kuuluvad kõik elemendid, millele on EAs võimalik lisada atribuute, ühendajaid ning meetodeid. Seetõttu on paketis loodud ka klassid nimedega *Attribute, Connector* ja *Method*. Lisaks selle on veel defineeritud klass *GeneralizationSets*, mis hoiustab infot ühendajate kohta, mis kuuluvad sellesse EA üldistushulka ning kahte *Boolean* väärtust nimedega *isCovering* ja *isDisjoint*. *isCovering* väärtus ütleb kas EA mudelis oleva ülemklassi

eksemplar (objekt) peab kuuluma vähemalt ühte EA mudelis olevasse alamklassi. *isDisjoint* väärtus ütleb kas EA mudelis oleva ülaklassi eksemplar võib samal ajal kuuluda mitmesse EA mudelis olevasse alamklassi. Kogu *jsonParser* paketi klassiskeem on näha Joonis 19.



Joonis 19. Paketi *jsonParser* klassiskeem.

Abstraktse klassi *Element* kohale on lisatud *com.fasterxml.jackson.core* teegiga kaasatud annotatsioon, mis laseb defineerida, millised JSON objektid muuta millisteks Java objektideks ühe JSON objekti välja järgi. Autori pool valitud väli on nimega *type.* Näitena on näha Joonisel 8 väli *type*, mille väärtuseks on *Class*, sest tegu on EA *Class* tüüpi objektiga. Joonisel 20 on näha, et JSON objekt, mille välja *type* väärtus on *Class* muudetakse Java koodis *ClassType* objektiks. . Joonis 20 näitab annotatsiooni ning objektide kaardistamist JSON objektist Java objektideks..

```
@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY, property = "type
        visible = true)
@JsonSubTypes({
        @JsonSubTypes.Type(value = ClassType.class, names={"Class","Interface","UseCase"}),
        @JsonSubTypes.Type(value = Artifact.class, name = "Artifact") ,
        @JsonSubTypes.Type(value = Comment.class, names = {"Note", "Constraint"}),
        @JsonSubTypes.Type(value = DiagramNote.class, name = "diagramNote")}
```

Joonis 20. Kaardistus JSON objektide ja neist loodavate Java objektide vahel, kasutades selleks JSONis olevat välja nimega *type*.

Sellega on kõik EA projektist tulevat infot hoidvad klassid defineeritud. Viimane klass, mis on defineeritud *jsonParser* paketis, on klass *JsonParser*. Tagarakendusega samas

kaustas on tekstifail nimega *inputJsonFilePath.txt*. See on fail, kus hoiustatakse faili asukohta, kuhu skript kirjutab oma väljundi ANSI kodeeringus. Esmalt loeb *JsonParser* klass sisse kogu tekstifaili sisu UTF-8 kodeeringus. See on vajalik, sest kui EA projekti nimes on täpitähed, siis lugedes neid ANSI kodeeringus failist annab *com.fasterxml.jackson.core* teek veateate, kuna ei mõista ANSI kodeeringus täpitähti.

Järgmise sammuna kasutab *JsonParser*-i klass *com.fasterxml.jackson.core* teegist objekti nimega *ObjectMapper*, mis loob igast JSON faili objektist vastava Java objekti.

Järgmisena kirjeldab autor paketi nimega *sentenceGenerator* sisu. Paketis on neli klassi järgmiste nimedega: *FileWatcher*, *Phrases*, *SentenceBuilder* ja *SentenceWriter*.

Phrases on klass, mis sisaldab klassi väljade (*class field*) väärtustes kõiki ingliskeelseid lauseid, fraase ja sõnu. Väärtused sisaldavad *%s* kohahoidjaid (parameetreid), millede asemele sisestatakse *jsonParser* paketi objektidest andmed. *Phrases* klassi konstruktorit on näha Joonis 21.

```
public Phrases() {
    this.numberZero = "zero";
    this.numberOne = "one";
    this.numberTwo = "two";
    this.numberThree = "three";
    this.numberFour = "four";
    this.numberFive = "five";
    this.numberSix = "six";
    this.numberSeven = "seven";
    this.numberEight = "eight";
    this.numberNine = "nine";
    this.diagramDescribesSentence = "Diagram %s describes %s %s";
    this.diagramDescribeEmptyDiagram = "Diagram %s is empty";
    this.addToDiagramDescribeSentenceIfMultiple = "and relationships between them";
    this.describeClassAttributesMandatory = "%s is characterized by %s %s that has type %s %s";
    this.describeClassAttributesOptional = "%s can be characterized by %s %s that has type %s %s";
    this.addInitialValue = "with initial value %s";
    this.describeConnectorGeneralizationSubToSuper = "%s is %s";
    this.describeConnectorGeneralizationSuperToSub = "%s can be %s";
    this.describeConnectorUsage1 = "%s uses %s";
    this.describeConnectorUsage2 = "%s is used by %s";
    this.describeConnectorDependency = "%s depends on %s";
```

Joonis 21. *Phrases* klassi konstruktor.

SentenceBuilder klass on kogu tagarakenduse tuum, kus kasutatakse *jsonParser* paketi Java objekte ja *Phrases* klassi ingliskeelseid sõnu ning fraase, et luua loomuliku keele laused, mida kasutajaliideses kuvada.

SenteneBuilder klass saab sisendina ühe *jsonParser* paketi *Diagram* objekti ning ühe klassi *Phrases* objekti. *SentenceBuilder* klassil on viis põhilist meetodit lausete loomiseks ja nende enda klassiväljadele salvestamiseks.

Esimene meetod on *diagramDescribeSentence()*, mis loob lause EA skeemi nimest ja milliste EA elementide vahelisi suhteid see EA skeem kirjeldab.

Järgmine meetod on *diagramNotes()*. See meetod sorteerib, millised *Diagram* objekti *DiagramNotes* objektidest on sissejuhatavad märkmed ning millised on täiendavad märkmed. Meetod otsustab seda selle järgi, kas *DiagramNote* objektiga kaasas olev Y-koordinaadi väärtus on suurem kui *Diagram* objektiga kaasa tulev Y-koordinaadi väärtus või mitte. *Diagram* objektiga kaasa tulev Y-koordinaadi väärtus on EA skeemil oleva visuaalselt kõige allpool oleva *Class*, *Inteface* või *UseCase* tüüpi EA elemendi koordinaat. Teiste sõnadega, kui märkmed on allpool kõiki neid elemente, siis on need täiendavad märkmed.

describeArtifactsInfo() meetod käib iteratiivselt läbi iga *Diagram* objekti *Artifacts* objekti ning teeb sellest loetava lause kasutades *Phrase* klassivälja, mis on tüüpi *String* ning sisestades *%s* sümbolite asemele tehise info. Seejärel salvestades selle lause *SentenceBuilder* klassiväljale.

Klassi üks kõige olulisemaid meetodeid on *describeElement()*. Selles meetodis käiakse iteratiivselt läbi kõik *Diagram* objekti *ClassType* objektid ning neile rakendatakse meetodi sees esmalt meetodit *describeElementAttributes()*. Kuna igal *ClassType* objektil võivad olla atribuudid, siis kasutades *Attribute* objekte luuakse loomulike keele laused ja salvestatakse *SentenceBuilder* klassiväljale.

Seejärel rakendatakse igale *ClassType* objektile meetodit *addConnectorInfo()*, mis loob iga *ClassType* objekti *Connector* objektidest loomuliku keele laused. Laused salvestatakse *SentenceBuilder* objekti klassivälja.

Peale seda rakendatakse igale *ClassType* objektile meetodit *classMethods()*. See meetod loob *ClassType* objekti *Method* objektidest loomuliku keele laused, kasutades *Phrases* objekti klassiväljade väärtuseks olevaid fraase ning salvestab laused *SentenceBuilder* objekti klassiväljadele.

Viimane põhiline meetod selles klassis on *addGeneralizationSetsInfo()*. Eelnevalt mainitud *addConnectorInfo()* meetodis käiakse iteratiivselt läbi kõik *Connector* objektid. Läbi käies vaadatakse *Connector* objekti *id* välja järgi, kas see objekt asub mõnes *Diagram* objekti *generalizationSets* väljal salvestatud *GeneralizationSet* objektis. *generalizationSet* väli hoiustab kasutaja kogu EA projekti üldistushulki.

Kuigi ühel *Diagram* objektil pole otseselt vaja terve EA projekti üldisushulki hoiustada, siis polnud EAs võimalik skriptil pärida üldistushulki ainult ühe kindla EA skeemi kohta.

GeneralizationSet objektis on salvestatud kõik EAs sellesse üldistushulka kuuluvate ühendajate unikaalsed *id*-d ehk identifikaatorid. Iga *Connector* objekt hoiustab samuti endas unikaalset *id*-d, mis on vastavuses EAs oleva ühendaja *id*-ga. Kui *Connector*-i *id* väärtus on salvestatud mõnes *GeneralizationSet* objektis, siis see ühendaja asub EAs vastavas üldistushulgas.

GeneralizationSet objekt, milles on salvestunud vähemalt ühe *Connector* objekti *id*, salvestatakse *Diagram* objekti *generalizationSetsOnThisDiagram* väljale, kus hoiustatakse kõikidest üldistushulkadest just neid, mis kuuluvad sellele *Diagram* objektile. See tähendab, et autor on otsustanud, et EA üldistushulk kuulub EA skeemi juurde, kui EA skeemil leidub selline EA element, millel on ühendaja, mis kuulub sellesse EA üldistushulka.

Meetod *addGeneralizationSetsInfo()* käib iteratiivselt läbi *Diagram* objekti *generalizationSetsOnThisDiagram* välja ehk kõik EA üldistushulgad, mis asuvad sellel EA skeemil ning loob neist loomuliku keele laused ning salvestab *SentenceBuilder* objekti klassiväljale. Kogu *SentenceBuilder* objekti kasutamine on näha Joonis 22. Kasutades neid *SentenceBuilder* meetodeid koos *Diagram* objektiga ja *Phrases* objektiga luuakse kõik vajalikud loomuliku keele laused. Laused on *String* kujul salvestatud *SentenceBuilder* objekti klassiväljades Java *HashMap*-idesse.

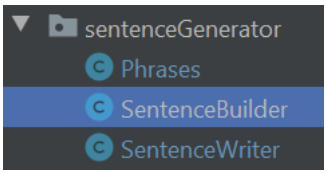
```
public void buildSentences() {

    sentenceBuilder = new SentenceBuilder(phrases, diagram);
    sentenceBuilder.diagramDescribesSentence();
    sentenceBuilder.diagramNotes();
    sentenceBuilder.describeArtifactsInfo();
    sentenceBuilder.describeElement();
    sentenceBuilder.addGeneralizationSetsInfo();
}
```

Joonis 22. Näide *SentenceBuilder* objekti loomisest ja selle kasutamisest autori lähtekoodis. *Phrases* objekt on nimega *phrases* ning *Diagram* objekt on nimega *diagram.*

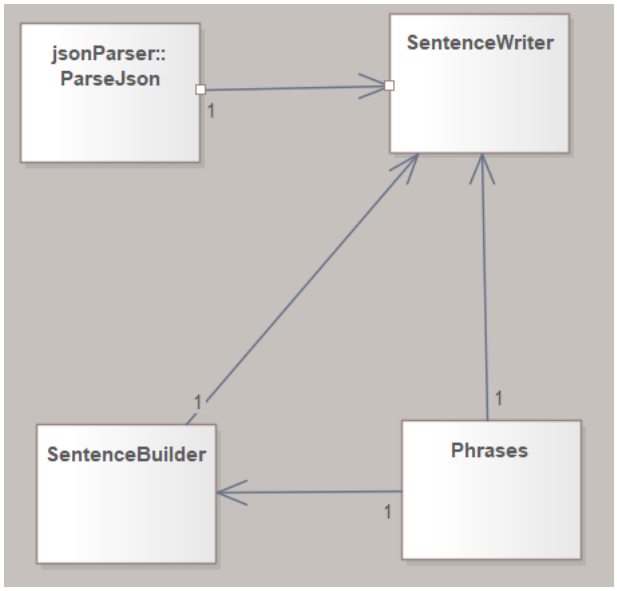
Järgmisena kirjeldab autor klassi nimega *SentenceWriter*. See klass saab konstruktoris sisendina endale *Phrases* objekti ning *ParseJson* objekti. Klassi põhiülesanne on luua üks pikk andmetüübiga *String* muutuja, mille väärtuseks on HTML (*HyperText Markup Language*) formaadis tekst, mis sisaldab kõiki kasutajaliideses kuvatavaid lauseid.

SentenceWriter objekt saab kõigepealt *ParseJson* objekti käest kõik Java objektid EAst tulnud andmetega. *SentenceWriter* loob *SentenceBuilder* objekti, andes selle konstruktorile sisendina *Phrases* objekti ning *ParseJson*-lt saadud *Diagram* objekti. Peale seda rakendab *SentenceWriter* objekt loodud *SentenceBuilder* objekti peal kõiki eelnevates lõikudes kirjeldatud meetodeid. Visuaalselt on seda näha Joonis 22.

SentenceWriter objektile on nüüd kättesaadavad kõik *SentenceBuilder*-i loodud laused, selle klassiväljadelt, millele saab ligi vastavate *Get* meetoditega. *SentenceWriter* objekt saab nüüd rakendada enda *WriteDiagramSentences()* meetodit. See meetod loob kõikidest *SentenceBuilder* objekti lausetest koos *SentenceWriter*-i enda loodud kirjeldavate lausetega ühe pika *String*-i, mis sisaldab kõiki EA skeemi kirjeldavaid loomuliku keele lauseid. Kui *SentenceWriter* saab *ParseJson* objektilt aga hoopis *Package* tüüpi objekti, siis käiakse läbi iteratiivselt iga *Package* objektile kuuluv *Diagram* objekt ning rakendatakse igale *Diagram* objektile *WriteDiagramSentences()* meetodit. Terve paketi *sentenceGenerator* sisu on näha Joonis 23 ja paketis olevate klasside vahelisis suhteid on illustreeritud Joonis 24.



Joonis 23. Paketi *sentenceGenerator* sisu.



Joonis 24. Paketi *sentenceGenerator* klassiskeem.

Järgmisena kirjeldab autor paketi nimega *applicationUI* sisu. Paketis on kaks klassi nimedega *FileWatcher* ning *ApplicationUI.*

ApplicationUI klass vastutab kasutajaliidese disaini ning funktsionaalsuse eest. Klass saab konstruktoris sisendina *SentenceWriter* objekti, milles on salvestatud tekst, mida *ApplicationUI* peab kasutajaliideses kuvama.

ApplicationUI konstruktoris käivitatakse kõigepealt meetod *UpdateGeneratedLanguage()*, mis küsib *SentenceWriter* objektilt kuvatava teksti ning salvestab *String*-ina klassiväljale. Seejärel rakendatakse meetodit *getUserLanguages()*, mis loeb programmi töökaustast sisse kõikide tekstifailide nimed väljaarvatud faili *InputJsonFilePath.txt*, sest see on fail, mis sisaldab selle faili asukohta, kuhu skript kirjutab JSON väljundi.

Konstruktoris rakendatakse viimasena *addButtonPanel()* meetodit. See meetod loob nupuriba, mis on kasutajaliidese avavaates ülevalääres ning järgmised nupud: *Choose input file, Add new Language* ja *Delete language*. Lisaks neile loob see veel nupud nende nimedega, mille *getUserLanguages()*, meetod tagastas. Igale nupule lisatakse

ActionListener() meetod, mis realiseerib kindlat funktsionaalsust kui kasutaja vajutab nupule. Kasutajaliidese avavaade on näha Joonis 12 ja Joonis 13.

Samuti on *ApplicationUI* klassis kolm meetodit nimedega *homeView()*, *changeToLanguageModificationView()* ja *changeToLanguageDeletionView()*. Meetodid loovad vastavalt kasutajaliidese avavaate (Joonis 12 ja Joonis 13), keele lisamise vaate (Joonis 14) ning keele kustutamise vaate (Joonis 17).

Vaadete loomiseks kasutatakse Java AWT ja Java *Swift* teeke. Vaadete loomiseks luuakse esmalt *JFrame* objekt, mis loob akna. Seejärel saab sellele aknale lisada teisi Java AWT teegi objekte – näiteks *JPanel* objekti akna segmenteerimiseks ning *JButton* objekti nuppude lisamiseks. Teksti kuvamiseks avavaates on kasutatud *JtextField* objekti, millele on lisatud *setText()* meetodiga kogu tekst, mille *ApplicationUI* objekt *SentenceWriter* objektilt pärib. Selleks, et tekst oleks vormindatud kujul nagu seda on näha Joonis 5, on *JTextField* objektil määratud meetodiga *setContentType(„text/html“)*. See meetod tagab, et *JTextField* objekt kasutaks sisendteksti HTML sümboleid. Sellepärast loobki *SentenceWriter* objekt kuvatava teksti HTML vormingus.

Vaate muutmisel kustutatakse *JFrame* objekti sees olevad Java AWT ja Java *Swift* objektid ning objektile antakse kuvamiseks uued sisendobjektid. Kõige lõpus rakendatakse *JFrame* objektil meetodeid *repaint()*, mis joonistab akna uue sisuga ning *setVisible(true)*, mis kuvab kasutajale akna uue sisuga.

Klass *FileWatcher* on avalik abstraktne klass, mis laiendab klassi *TimerTask* ning mille ülesandeks on vaadata, kas skript on kirjutanud faili uued JSON kujul andmed. *TimerTask* klass defineerib objekti, mille meetodeid saab perioodiliselt käivitada. Selle tegemiseks peab *FileWatcher* objekt ülekirjutama *TimerTask run()* meetodi. *FileWatcher* objekti *run()* meetodit saab perioodiliselt käivitada kui luua Java objekt *Timer*, mis on Javas juba defineeritud klass ning anda selle *schedule()* meetodile sisse *FileWatcher* objekt.

Nüüd on olemas loogika *FileWatcher* objekti *run()* meetodi perioodiliseks käivitamiseks. Saab luua *FileWatcher* objekti, mis saab sisendina *ParseJson* objekti, kuhu on salvestatud skripti väljundfaili asukoht. *FileWatcher* objekti konstruktoris saab kasutades väljundfaili asukohta luua Java *File* klassi faili objekti ning rakendada sellel meetodit *lastModified()*, mis tagastab millal faili viimati muudeti. Kui *run()* meetod perioodiliselt käivitatakse, siis meetod kontrollib, kas rakendades failile praegu meetodit *lastModified()* tagastatakse

sama tulemus kui siis kui *FileWatcher* objekt loodi. Juhul kui mitte, siis salvestatakse faili uus muutmise aeg klassiväljale ning kutsutakse välja *FileWatcher* objekti meetodit *onChange()*. Selle meetodi kehas tehakse midagi juhul kui skripti väljundfaili sisu on muudetud. Kui *run()* meetod uuesti perioodiliselt käivitatakse, siis võrreldakse uuesti, kas faili on võrreldes klassiväljale salvestatud ajaga muudetud.

Kogu tagarakenduse viimaseks kirjeldatavaks klassiks on *Main* klass. See on klass, mille ülesandeks on luua kõik vajalikud objektid programmi käivitamiseks ning käivitada programm.

Main klass loob objektid tüüpi *Phrases*, *ParseJson*, *SentenceWriter*, *ApplicationUI*, *FileWatcher* ning *Timer*. *SentenceWriter* objektile antakse konstruktoris sisendina *Phrases* ning *ParseJson* objektid. *ApplicationUI* objektile antakse konstruktoris sisendina *SentenceWriter* objekt.

ApplicationUI objektile rakendatakse meetodit *homeView()*, mis kuvab rakenduse tööle pannes kasutajaliideses avavaate. Samuti luuakse *FileWatcher* objekt nimega *task*. Luues *FileWatcher* objekti defineeritakse ka *onChange()* meetod. See meetod käivitub kui *FileWatcher* objekti *run()* meetod on tuvastanud muutuse skripti väljundfailis.

onChange() meetodi sees on defineeritud, et kui see meetod käivitatakse, siis *ApplicationUI* objektis eemaldatakse vana *JTextField* objekt, mis kuvab teksti ning see asendatakse uue *JTextField* objektiga, milles on uus tekst.

Viimaseks luuakse Java *Timer* objekt. *Timer* objektil rakendatakse meetodit *schedule()*, millele antakse sisendina *task*. See paneb perioodiliselt käima *task* objekti *run()* meetodi. Autor on valinud ajaperioodiks kaks sekundit. Kogu *Main* klassi *main()* meetodi sisu on näha Joonis 25 ning paketi *applicationUI* klassiskeem on näha Joonis 26.

```
public static void main(String[] args) {
    System.setProperty("file.encoding","UTF-8");
    Phrases phrases = new Phrases();
    ParseJson parser = new ParseJson();
    SentenceWriter sentenceWriter = new SentenceWriter(phrases, parser);
    ApplicationUI applicationUI = new ApplicationUI(sentenceWriter);
    applicationUI.homeView();

    TimerTask task = new FileWatcher(parser) {
        protected void onChange( File file ) {
            // here we code the action on a change
            System.out.println( "File "+ file.getName() +" have change !" );
            applicationUI.removeLabels();
            applicationUI.updateLabelsText();
        }
    };
    Timer timer = new Timer();
    // repeat the check every 2 seconds
    timer.schedule( task , new Date(),  period: 2000 );

}
```

Joonis 25. Java tagarakenduse *Main* klassi *main()* meetod.

Joonis 26. *applicationUI* paketi klassiskeem.

Lähtekoodis on veel palju, millest võiks kirjutada, aga see muudaks töö põhiosa teksti liiga pikaks. Seetõttu tõi autor välja lähtekoodi olulisemad osad. Lähtekoodiga on võimalik täpsemalt tutvuda sellel GitHub-i lehel: https://github.com/alvaltna/NaturalLanguageGenerator.

7 Rakenduse installeerimine ja kasutamine

Käesolevas peatükis kirjeldab autor, millised sammud ja nõudmised tuleb täita, et töö käigus loodud tarkvara arvutisse installeerida ning see kasutusvalmis seada.

7.1 Rakenduse installeerimine

Kõik tarkvarakomplektiga kaasa tulevad failid peavad asuma arvuti ühes kaustas. Kompileeritud tarkvarakomplekt on hoiustatud GitHub-is https://github.com/alvaltna/NaturalLanguageGenerator, kaustas nimega *Software To Generate Natural Language Sentences*. Seal on leitav ka inglisekeelne installeerimise ja kasutusjuhend.

1. Alla laadida JDK (*Java Development Kit*) 11 või uuem versioon.
2. Käivitada tarkvarakomplekti fail nimega *SetJdkPath* valikuga *Run as administrator*.
3. Avanenud konsooliaknasse kopeerida JDK-s kaustas oleva *bin* kausta kataloogitee.
4. Seejärel avada EA.
   EA 12-s tuleb valida kasutajaliideses ülevalt tööribalt *Project -> Data Management -> Import Reference Data*. Valida tarkvarakomplekti kaustast fail nimega *LanguageGenerationScript.xml*. Seejärel valida EAs *Select Datasets To Import -> Datasets to Import* ja vajutada nupule *Import*.
   EA 15 puhul tuleb valida kasutajaliideses ülevalt tööribalt *Configure -> Transfer -> Import Reference Data*. Valida tarkvarakomplekti kaustast fail nimega *LanguageGenerationScript.xml*. Seejärel valida EAs *Select Datasets To Import -> Datasets to Import* ja vajutada nupule *Import*.
5. Käivitada programm tarkvarakomplektis olevast *RunProgram* failist.

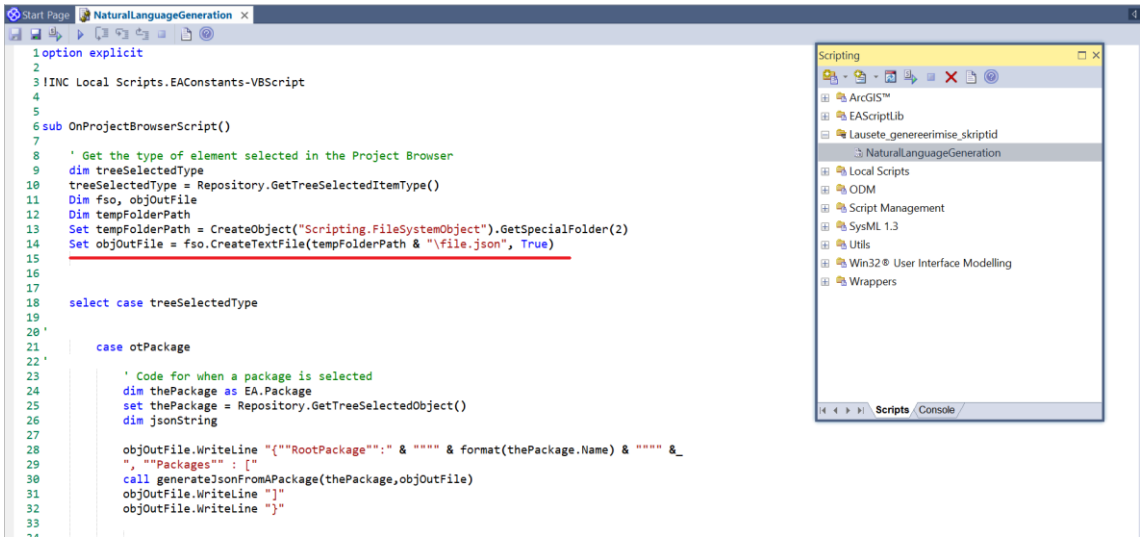
7.2 Rakenduse kasutamine

Lauseid saab genereerida ainult pakettidest ja skeemidest. Vaikimisi kirjutab skript JSON kujul väljundi masina *C:\Users\user\AppData\Local\Temp* kasuta faili *file.json*, kus *user* on masina kasutaja. Selle muutmiseks on vaja minna EA 12-s kasutajaliideses tööribal

View->Scripting. Avanenud aknas valida *Lausete_Genereerimise_skriptid* valiku alt *NaturalLanguageGeneration* skript. Tehes sellel topelt hiirevajutuse avaneb skripti sisu kasutajaliideses.

EA 15-s on vaja minna kasutajaliideses tööribal *Specialize->Tools->Script Library*. *Scripting tab* aknas valida *Lausete_Genereerimise_skriptid* valiku alt *NaturalLanguageGeneration* skript. Tehes sellel topelt hiirevajutuse avaneb skripti sisu kasutajaliideses.

Skripti sees on rida, mida on näha Joonisel 27, mille all on punane joon ning lähemalt on rida näha Joonisel 28.

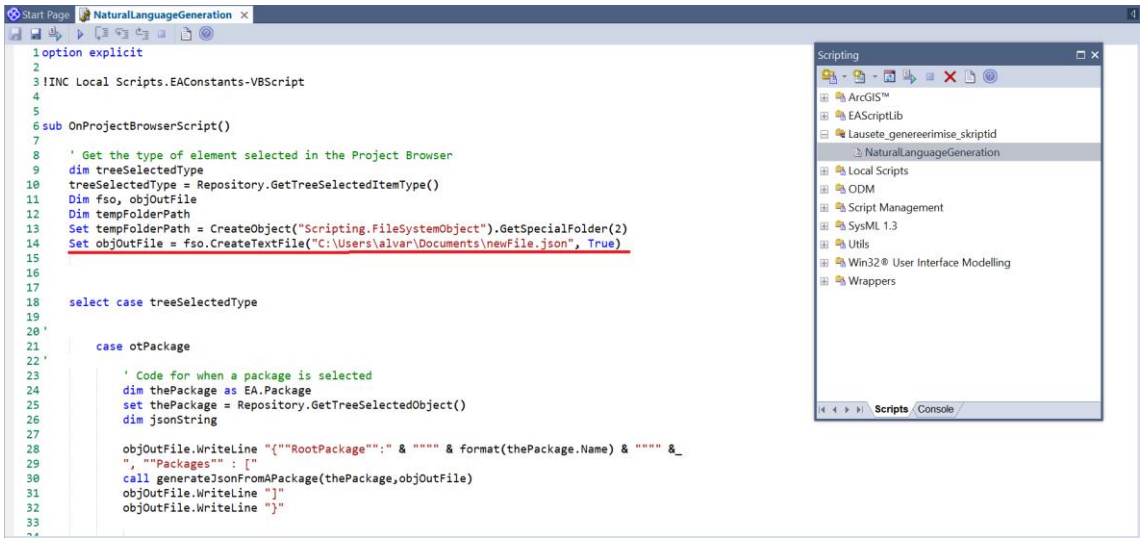


Joonis 27. Rida skriptis mida tuleb muuta, et skript genereeriks väljundi kasutaja soovitud faili.

```
Set objOutFile = fso.CreateTextFile(tempFolderPath & "\file.json",
True)
```

Joonis 28 Rida skriptis, mis määrab kuhu skript kirjutab oma väljundi.

Sellel real tuleb *CreateTextFile()* meetodi esimene sisend asendada selle faili asukohaga, kuhu tahetakse kirjutada skripti JSON kujul väljund. Näiteks autor määrab uueks skripti väljundi asukohaks *C:\Users\alvar\Documents\newFile.json*. Muudetud skripti rida on näha Joonisel 29, mille all on punane joon.



Joonis 29. Skripti rida peale uue väljundi asukoha määramist.

Tagarakenduse käivitamiseks tuleb tarkvarakomplektis käivitada fail *RunProgram*. Selleks, et tarkvara töötaks õigesti, peab tagarakenduses olema valitud andmete sisse lugemiseks skriptis määratud fail. Tarkvara esmakordsel käivitamisel skripti väljundfaili asukoht ja tagarakenduse lugemise asukoht vaikimisi ühtivad ning mõlemad viitavad eelnevalt mainitud asukohale.

Kui kasutaja muudab skriptis väljundfaili asukohta, siis peab ta sama asukoha valima tagarakenduses kasutajaliideses, vajutades selleks nuppu *Choose file input* ning valima graafilise kasutajaliidese abil sama asukoha. Nüüd loeb tagarakendus andmeid õigest failist.

Lausete genereerimiseks soovitud paketist või skeemist tuleb EAs *Project Browser*-is teha soovitud paketile/skeemile parem hiirevajutus. EA 12-s avanenud menüüst valida *Scripts* ning *NaturalLanguageGeneration*. EA 15-s avanenud menüüst valida *Specialize->Scripts* ning *NaturalLanguageGeneration.*

Vaikimis genereeritakse laused inglise keeles. Kasutaja saab lisada endale sobiva keele. Keelte lisamist, keelefailide muutmist ja keele toe kustutamist selgitatakse jaotises 8.3.

Genereerides suurel hulgal lauseid paljudest pakettidest läheb programmil rohkem aega. Autor mõõtis genereerimiseks kuluvat aega kahel juhul.

Juht 1. Pakett, milles polnud alampakette ja kus oli üks skeem, millel oli kokku 5 klassi. Paketist lausete genereerimiseks kulus 2774 ms (2.77 s).

Juht 2. Pakett, milles oli 29 otsest alampaketti, millel olid omakorda alampaketid. Kokku oli selle paketi alampakettides 130 elementi (*Class*, *Interface*, *UseCase*). Paketist lausete genereerimiseks kulus 19482 ms (19.48 s).

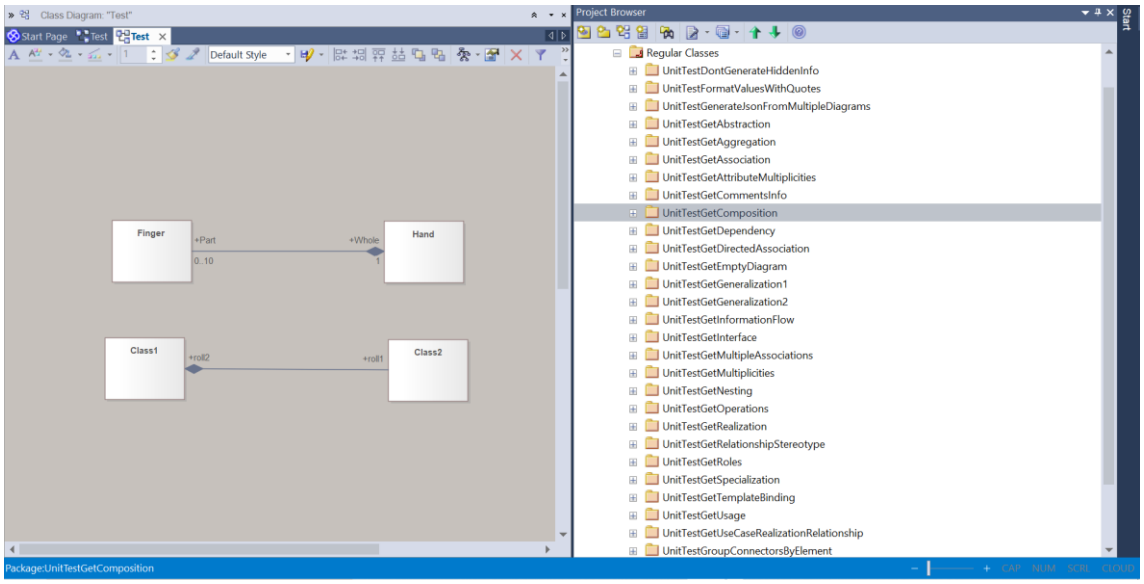
8 Tulemuste kontrollimine

Käesolevas peatükis käsitletakse töö tulemuse kontrollimist läbi testimise. Tarkvara testitakse mitmel viisil. Esimene testimise viis on kasutada EAs loodud ühikteste ning näiteprojektides olevaid suuremahulisi klassiskeeme. Teine viis on lasta Tallinna Tehnikaülikooli informaatika eriala üliõpilastel joonistada autori tarkvara genereeritud loomuliku keele lausete põhjal klassiskeeme ning hinnata, kas tulemus on samasugune kui algsed skeemid, millest tarkvara laused genereeris. Katses osalevad üliõpilased aga ei näe algseid skeeme, vaid ainult lauseid. Kõigele lisaks on peatüki viimases jaotises esitatud tarkvara kasutamise põhjalik näide.

8.1 Testimine kasutades EAd

Terve tarkvaraarenduse protsessi jooksul lõi autor EAs ühikteste, mida kasutada erinevate programmikoodi osade testimiseks. Ühiktestid kujutasid endast EA skeeme, millel oli esitatud vaid sellised elemendid, millest lausete genereerimine aitaks testida autori programmikoodis teatud klasse või meetodeid.

Joonisel 30 on näha autori EA 12 projekti ühiktestidega ning neid sisaldav fail on leitav GitHub-i salvest failist nimega *Unit_tests.eap.*



Joonis 30 Autori loodud ühiktestid EA 12-s.

Näiteks lõi autor paketi nimega *UnitTestGetClassInfo*, milles oli skeem kuhu oli joonistatud, vaid üks *Class* tüüpi element. Sellel oli nimi ja kohustuslikud atribuudid (atribuudid, mille võimsustik oli 1). Lisaks oli mõnel atribuudil ka algväärtus.

See oli üks esimesi ühikteste kontrollimaks, millised laused loob tarkvara ühest lihtsast skeemist. Arendusega jätkati peale seda kui sellest skeemist loodi soovitud laused.

Järgmine autori loodud ühiktest oli *UnitTestGetAttributeMultiplicities*. Tegu oli lihtsa skeemiga, mis sisaldas *Class* tüüpi elementi nagu eelmises ühiktestiski, ainult seekord olid atribuutidel erinevad võimsustikud.

Samuti lõi autor näiteks ühikteste järgnevate olukordade jaoks.

- *Class* tüüpi elemendi meetodite kohta lausete genereerimiseks,
- Kontrollimaks mis juhtub lausete genereerimisel siis kui mudelielement on *Interface* või *UseCase.*
- Iga jaotises 3.1 mainitud ühendaja tüübi kohta lausete genereerimiseks.
- Elementide küljes olevate märkmete ja piirangute infot sisaldavate lausete genereerimiseks.
- Elementide rolli infot sisaldavate lausete genereerimiseks.

Kokku lõi autor 29 ühiktesti.

Ühiktestide põhiidee oli luua skeem ning lisada sellele minimaalne hulk infot, mis on vajalik, et testida programmikoodi mingit osa. Ühiktestid pidid katma kõiki jaotises 3.1 välja toodud nõudeid.

Kui kõikide ühiktestide puhul suutis tarkvara genereerida selliseid lauseid nagu jaotises 3.1 on kirjeldatud ning ühiktestid katsid kõik võimalikud kasutusjuhud ning kombinatsioonid, siis liikus autor edasi testima oma tarkvara suuremate klassiskeemide peal.

Autoril oli endal Andmebaasid I ja II õppeainest alles EA fail, milles oli andmebaasirakenduse ühe funktsionaalse alamosa e allsüsteemi poolt kasutatavat andmebaasi alamosa kirjeldavad olemi-suhte skeemid e diagrammid, mis olid loodud klassiskeemide põhjal. Need mudelid loodi analüüsi tulemusena ja sobisid seega testimiseks. Selles failis olid klassiskeemid 6 paketis. Skeeme oli kokku 6 ning skeemidel kujutatud klasse oli kokku 32. Lisaks sai autor juhendajalt näiteprojekte, mille põhjal tarkvara testida.

Põhiline erinevus ühiktestidest oli see, et kui ühiktestide puhul olid erinevad klassiskeemide aspektid eraldi skeemidel, siis näiteprojektides olid paljud nendest ühel skeemil. Lisaks sellele võis ühel skeemil esitatud element olla ühendatud läbi ühendaja teise elemendiga, mis oli kujutatud teisel skeemil, kuid mida ei olnud esimesel skeemil näha.

Tarkvara pidi suutma luua ühendajate kohta arusaadavad laused ka siis kui ühendaja poolt ühendatud elemendid olid kujutatud erinevatel skeemidel. Peale selle võis näiteks element, mis asus ühel skeemil, kuuluda üldistushulka, mis oli esitatud teisel skeemil. Tarkvara pidi sellisel juhul suutma esimesel skeemil oleva info põhjal genereerida korrektsed laused üldistushulga kohta ning esitama lauses kõik elemendid, mis kuuluvad sellesse üldistushulka.

8.2 Katsetus teiste üliõpilastega

Töö autor võttis ühendust kolme Tallinna Tehnikaülikooli informaatika eriala üliõpilasega, selleks, et nende abil kontrollida loodud tarkvara poolt genereeritud

loomuliku keele lausete loetavust ja arusaadavust. Autor suhtles üliõpilastega ükshaaval, kasutades MS Teamsi. Kõik nad olid kunagi õppinud UMLi, kuid kahel nendest oli õppimisest möödas rohkem kui 1,5 aastat.

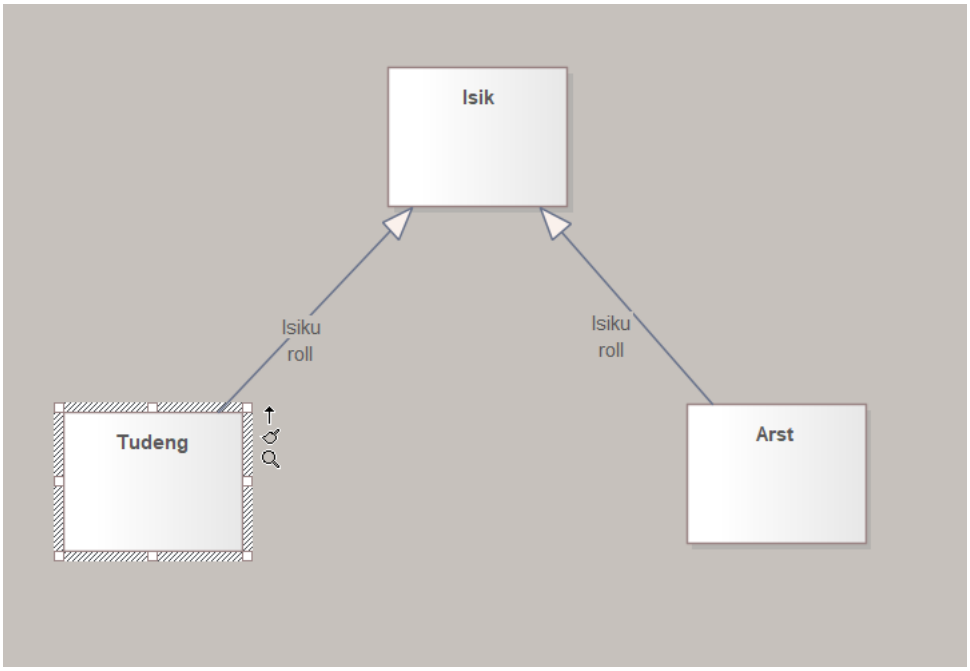
Autor koostas klassiskeemid ja genereeris loodud tarkvara abil nendest loomuliku keele laused. Ta palus üliõpilaste joonistada kas EAs või käsitsi paberile klassiskeemid ainult tarkvara loodud lausete põhjal, ilma, et üliõpilased oleksid näinud algseid skeeme, millest laused genereeriti. Kokku anti üliõpilastele joonistamiseks 4 skeemi alusel genereeritud laused. Kokku anti üliõpilastele 61 lauset. Üliõpilastele ei tutvustatud eelnevalt klassiskeeme ega lausete ülesehitust. Autor andis üliõpilasele, kes joonistas skeeme EAis infot kui ta ei teadnud, kust teatud asju kasutajaliideses leida. Näiteks, kust saab lisada rolle elementidele.

Järgmisena toob autor välja skeemid, millelt lauseid genereeris. Skeemid on näha Joonisel 31, Joonisel 33, Joonisel 35 ja Joonisel 37. Laused, mis tarkvara genereeris nendest skeemidest on näha Joonisel 32, Joonisel 34, Joonisel 36 ja Joonisel 38.

Joonis 31. Esimene skeem, millest autor genereeris laused testimise jaoks.

```
DIAGRAM Test

Diagram Test describes Töötaja.

ELEMENT DESCRIPTIONS

Class Töötaja
Töötaja is characterized by one Eesnimi that has type char.
Töötaja can be characterized by zero to many Email that has type char.
Töötaja is characterized by one Perenimi that has type char.
Töötaja is characterized by one Vanus that has type int with initial value 22.

CONNECTORS

Class Töötaja
```

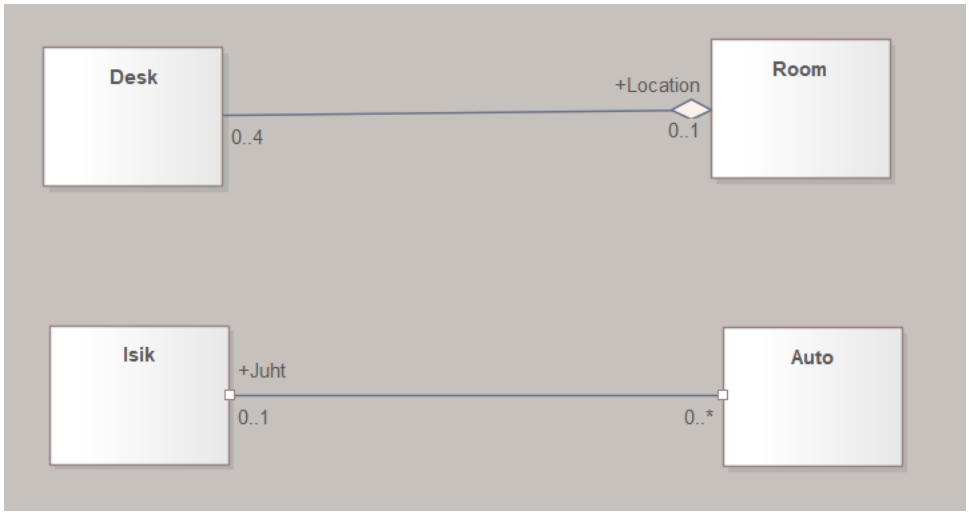
Joonis 32. Laused, mis genereeriti esimeselt skeemilt (Joonis 31).



Joonis 33. Teine skeem, millelt autor genereeris laused.

```
DIAGRAM Test

Diagram Test describes Isik, Tudeng, Arst and relationships between them.

ELEMENT DESCRIPTIONS

Class Arst
Class Isik
Class Tudeng

CONNECTORS

Each Isik doesn't have to have Isiku roll of the following type: Tudeng, Arst.
Each Isik can at same time have only one Isiku roll of the following type: Tudeng, Arst.

Class Arst
Arst is Isik.

Class Isik
Isik can be Tudeng.
Isik can be Arst.

Class Tudeng
Tudeng is Isik.
```

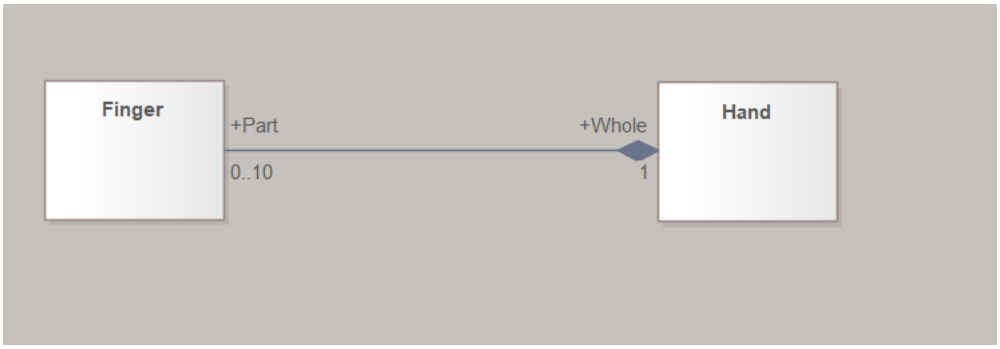
Joonis 34. Laused, mis genereeriti teiselt skeemilt (Joonis 33).



Joonis 35. Kolmas skeem, millelt autor lauseid genereeris.

```
DIAGRAM Test

Diagram Test describes Desk, Room, Isik, Auto and relationships between them.

ELEMENT DESCRIPTIONS

Class Auto
Class Desk
Class Isik
Class Room

CONNECTORS

Class Auto
Each Auto can be associated with at most one Isik as a Juht.
Some Auto is associated with no Isik as a Juht.

Class Desk
Each Desk can belong to at most one Room as a Location.
Some Desk belongs to no Room as a Location.

Class Isik
Each Isik as a Juht can be associated with many Auto.
Some Isik as a Juht is associated with no Auto.

Class Room
Each Room as a Location can have at most four Desk.
Some Room as a Location has no Desk.
```

Joonis 36. Laused, mis genereeriti kolmandalt skeemilt (Joonis 35).



Joonis 37. Neljas skeem, millelt autor lauseid genereeris.

```
DIAGRAM Test

Diagram Test describes Finger, Hand and relationships between them.

ELEMENT DESCRIPTIONS

Class Finger
Class Hand

CONNECTORS

Class Finger
Finger as a Part is a part of Hand as a Whole and can't exist without it.
If Hand as a Whole is deleted, then Finger as a Part should also be deleted.
Each Finger as a Part must belong to exactly one Hand as a Whole.

Class Hand
Finger as a Part is a part of Hand as a Whole and can't exist without it.
If Hand as a Whole is deleted, then Finger as a Part should also be deleted.
Each Hand as a Whole can have at most 10 Finger as a Part.
Some Hand as a Whole has no Finger as a Part.
```

Joonis 38. Laused, mis genereeriti neljandalt skeemilt (Joonis 37).

Autor andis üliõpilastele tarkvara genereeritud laused ning palus nende järgi joonistada klassiskeemid. Lisaks lausete loogikale ja arusaadavusele andis see tagasisidet ka genereeritud lausete järjestuse ja alajaotustesse paigutamise headuse kohta. See aitas aru saada kui hästi on võimalik klassiskeeme joonistada järgides lausete alajaotuste järjestust ning lausete järjestust alajaotuste sees.

Esimene üliõpilane otsustas joonistada klassiskeemi EAs, sest tal oli see tarkvara arvutis juba olemas. Teised kaks üliõpilast otsustasid joonistada oma klassiskeemid käsitsi paberile. Autor jälgis reaalajas joonistamise protsessi.

Üliõpilased alustasid joonistamist esimestest genereeritud lausetest ning liikusid järjest allapoole. Ühelgi üliõpilasel ei olnud kogu joonistamise protsessi vältel klassiskeemi korrektse joonistamise jaoks vaja lausete alajaotusi vahele jätta või hiljem mõne varasema lasuete alajaotuse juurde naasta.

Esimesed puudujäägid genereeritud lausetes tulid välja seoses üldistusseostega ja realisatsiooni seostega. Lausete alajaotuses *Connectors* elementide ühendajate kohta käivad laused on järjestatud tähestikulises järjekorras elementide nimede järgi, mistõttu on klassiskeemil olevad pärivate elementide ühendajad ning realiseerivate elementide ühendajad mõnikord varem kirjeldatud, kui üldistavate elementide ühendajad või realiseeritavate elementide ühendajad. Näitena võib tuua Joonisel 34 *Connectors* alajaotuses olevaid lauseid. Elemendi nimega „Arst“ ühendajad on kirjeldatud varem kui

elemendi nimega „Isik“ ühendajad, kuigi viimane element üldistab elementi nimega „Arst“.

Kaks üliõpilast andsid tagasisidet, et parem oleks kui elementide ühendajad oleksid kirjeldatud sellises järjekorras, kus elementide, mida ei üldista teised elemendid, ühendajad oleksid varem kirjeldatud ning spetsiifilisemate elementide, mis pärivad üldisemaid elemente ühendajad, on alajaotuses hiljem kirjeldatud.

Seega elemendid, mis oleks pärimiste jadas kõige allpool ehk elemendid, mille üldistavad elemendid pärivad omakorda üldistavatel elementidelt. Nende elementide ühendajad võiksid olla alajaotuses kõige hiljem kirjeldatud.

Mida allpool on pärimiste jadas element seda hiljem võiks selle elemendi ühendajad olla alajaotuse kirjeldatud.

Üliõpilased väitsid, et nii on kõige mugavam joonistada elemente, sest saab joonistada elementide ühendajaid järjest alajaotuses olevatest lausetest ning ei pea alajaotuses sees üles-alla liikuma, et leida ühendajaid, mida tuleks järgmisena joonistada.

Samuti võiksid *Element descriptions* alajaotuses olla elemendid järjestatud ka üldistuste järgi nagu ühendajad *Connectors* alajaotuses, sest siis saab kõige üldisemad elemendid joonistada kohe skeemi ülaserva, kus on nende kõige loogilisem asetus.

See tagasiside on oluline, kuid ei ole hetkel töös realiseeritud ning võiks olla osa tuleviku arendusest.

Teisena võib välja tuua, et autor pidi vahel selgitama ühendajate lausete alajaotuses, millist UMLi ühendaja tüüpi lauses kirjeldatakse. Näiteks ühendajaid kirjeldavates lausetes (Joonis 36) tähendab sõna *’can have’* UMLis valikut *Aggregation to Part.*

Klassiskeemil tähendab see klasside vahelist ühendust, kus valge teemant joonistatakse selle elemendi poole, mille objektid omavad teise elemendi objekte. Kõigile katsealustele, tuli seda joonistamise käigus selgitada.

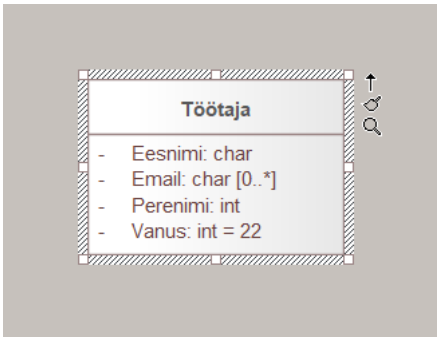
Samuti, näiteks kui ühendajate alajaotuses lausele sõnaga *’can have’* (Joonis 38) eelneb lause, mis ütleb, et kui klassi objekt kustutatakse, siis tuleks ka kustutada kõik objektid, mida see objekt omas, siis see tähendab ühendaja tüüpi *Composition*. Selle ühendaja tüübi

puhul joonistatakse must teemant selle klassi poole, mille objektid omavad teise klassi objekte. Kõigile katsealustele tuli selgitada *Composition* ja *Aggregation* tüüpi ühendaja erinevust.

Peale selgituste andmist oskasid üliõpilased joonistada korrektsed ühendajad. Lisaks sellele on ühendajate lausetes kirjeldatud ühendajate võimsustikud sõnadega. Kuigi üliõpilased said lausete loogikast aru, ei teadnud kaks nendest, kuidas neid võimsustikke õigesti joonisel üles märkida. Näiteks ei teatud , et sõna ’mitut’ tähistatakse klassiskeemil sümboliga *.

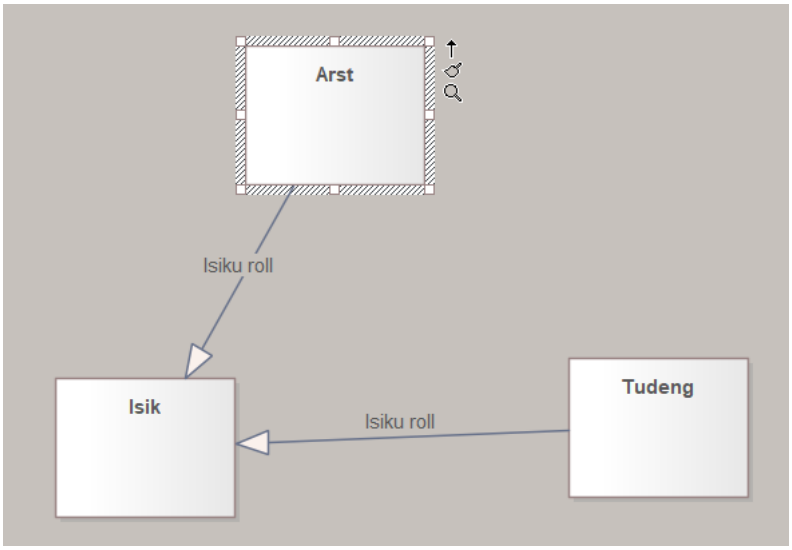
Nähes lauseid üldistushulkade kohta said kõik üliõpilased aru, mida need sisuliselt tähendasid, kuigi kaks neist ei teadnud, mis on üldistushulgad. Tarkvara genereeritud laused aitasid neil mõista üldistushulga mõistet ning millist infot need endas kätkevad.

Järgmisena toob autor välja skeemid, mille joonistas tudeng, kes kasutas EA CASE vahendit. Skeemi, mille üliõpilane joonsitas esimeste lausete järgi (Joonis 32) on näha Joonisel 39. Kõik joonistati õigesti.



Joonis 39. Esimese skeemi lausete (Joonis 32) järgi joonistatud skeem.

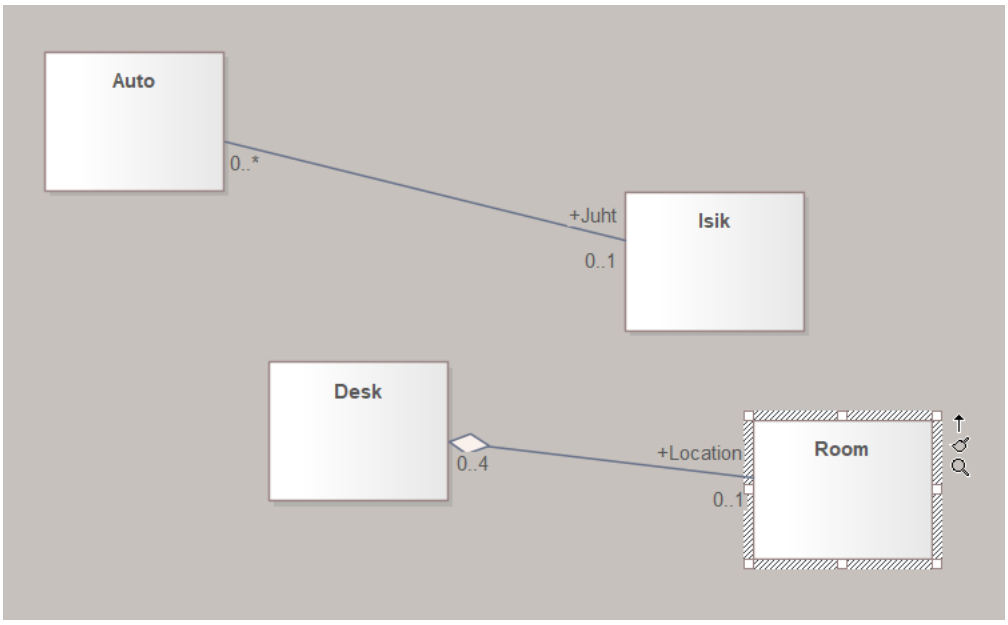
Skeemi, mille üliõpilane joonistas teiste lasuete järgi (Joonis 34) on näha Joonisel 40. Üliõpilane ei mäletanud, mis on üldistushulgad ning üldistushulkade lisamise asemel määras ainult ühendajatele nimed „Isiku roll“.



Joonis 40. Teise skeemi lausete (Joonis 34) järgi joonistatud skeem.

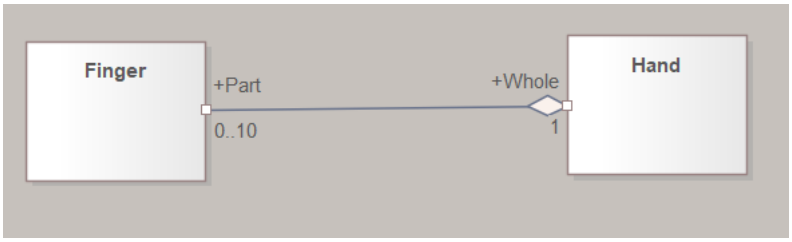
Skeemi, mille üliõpilane joonistas kolmandate lausete järgi (Joonis 36) on näha Joonisel 41. Üliõpilane joonistas kõik õigesti, ainult *Aggregation* tüüpi ühendaja märgiti skeemile valet pidi. Valge teemant peab olema elemendi nimega *Room* pool, mitte elemendi nimega *Desk* pool, sest *Room* on selles kontekstis tervik.

.



Joonis 41. Kolmanda skeemi lausete (Joonis 36) järgi joonistatud skeem.

Skeemi, mille üliõpilane joonistas neljandate lausete järgi (Joonis 38) on näha Joonisel 42. Üliõpilane joonistas skeemi peaaegu õigesti. Viga oli sellest, et elementide *Finger* ja *Hand* vaheline ühendaja oleks pidanud olema tüüpi *Composition*, aga üliõpilane joonistas ühendaja tüüpi *Aggregation*.



Joonis 42. Neljanda skeemi lausete (Joonis 38) järgi joonistatud skeem.

Eelnev näitas autorile, et inimesed, kes ei ole varem klassiskeemidega palju kokku puutunud, ei oska alati kokku viia, millist klassiskeemi osa mingi loomuliku keele lause kirjeldab. Samas näidates hiljem üliõpilastele algseid skeeme, millest tarkvara laused genereeris, sai autor tagasisidet, et kui näha skeemi ja sellest genereeritud lauseid kõrvuti, siis ei ole raske aru saada, millist klassiskeemi aspekti mingi lause kirjeldab.

Samuti testis autor tarkvara üle viidavust teistesse süsteemidesse ja kasutajaliidese mugavust, paludes üliõpilastel installeerida tarkvara oma süsteemile arvutisse ning anda tagasisidet kasutajaliidese kasutamise kohta. Üliõpilased proovisid kasutajaliidest reaalajas MS Teamsi kõnes, mis andis autorile kohest tagasisidet selle kasutamise kohta. Kõikides süsteemides tarkvara käivitus ning kõik katses osalejad ütlesid, et kasutajaliidest on lihtne ja intuitiivne kasutada.

8.3 Tarkvara kasutamise näide

Selles jaotises kirjeldatakse kuidas tarkvara kasutada, kui kõik peatükis 7 kirjeldatud sammud on juba läbitud.

Kõigepealt tuleb avada EA projekt. Seejärel tuleb valida kasutajaliideses *Project Browser*-is skeem või pakett, millest soovitakse lauseid genereerida ning teha parem hiirevajutus sellel. EA 12 puhul tuleb avanenud menüüst valida *Scripts->NaturalLanguageGeneration* ja EA 15 puhul tuleb avanenud menüüst valida *Specialize->Scripts->NaturalLanguageGeneration*. Skript genereerib JSON formaadis väljundi, mis pannakse skriptis määratud faili. Kogu protsess on visualieeritud Joonis 10.

Tarkvarakomplektis tuleb käivitada *RunProgram* fail, mis paneb tööle Java tagarakenduse ning kuvab kasutajaliidese avavaate. Kui tagarakenduses on valitud andmete sisselugemiseks sama fail, kuhu skript väljundi kirjutab, siis kuvab kasutajaliides kohe loomuliku keele laused inglise keeles nagu on näha Joonis 12. Tagarakenduses saab

valida faili, millest JSON formaadis andmeid sisse loetakse, vajutades nuppu *Choose input file* ning valida graafiliselt soovitud faili nagu on näha Joonis 11.

Tarkvarakomplektiga tuleb kaasa tekstifail nimega *Estonian.txt*. Kui see fail on samas kaustas, kus on tagarakendus, siis kuvatakse kasutajaliideses üleval tööribal keelevaliku nupp *Estonian* (nupul olev tekst tuletatakse faili nimest). Kui vajutada sellele nupule, siis tõlgitakse kasutajaliideses kuvatud laused eesti keelde nagu on näha Joonis 13.

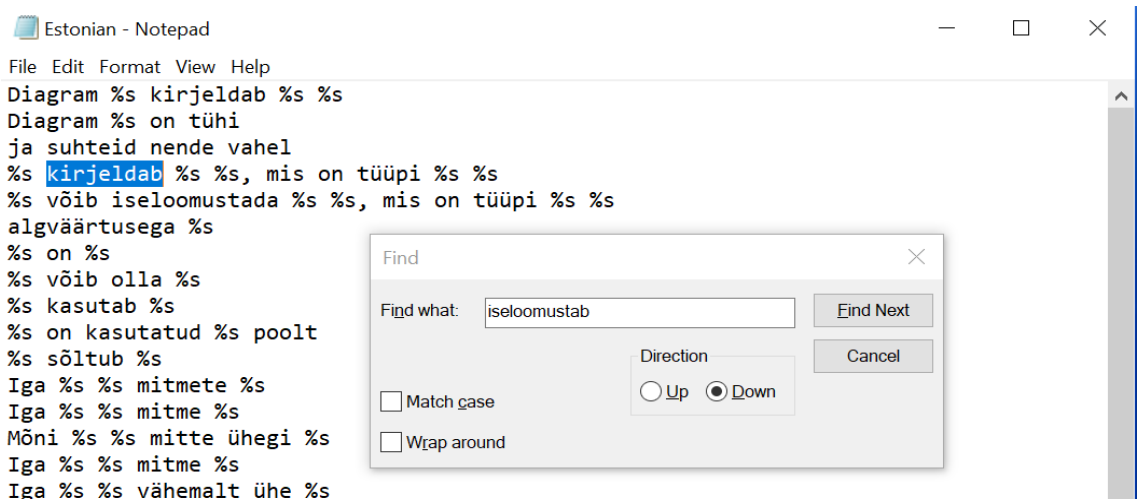
Kui kasutaja soovib lisada tarkvarasse uue keele toe, siis peab ta vajutama tööribal nupule *Add new language*. Seejärel avaneb keele lisamise vaade nagu on näha Joonis 14. Kasutajal tuleb järgida vaate üleval ääres olevat õpetust või jaotises 7.2 olevat kirjeldust. Töö lõpetamiseks tuleb vajutada vaate all ääres asuvat *Save new language* nuppu. Siis kuvatakse hüpikaken, kuhu tuleb sisestada keele nimi. Tagarakendus loob seejärel selle nimega tekstifaili samasse kausta, mis sisaldab tõlgitud lauseid. Kui kasutaja annab keelele nime, mis juba kasutusel (selle kohta on olemas lausete malle sisaldav tekstifail), siis kirjutatakse vana tekstifail uute lausetega üle. Kui nüüd vajutada *Save new language* nupu kõrval asuvat *Back* nuppu, siis kuvab kasutajaliides avavaate, kus on tööribal uue keele valimise võimalus nii nagu on näha Joonis 13.

Iga kord kui kasutaja käivitab EAs paketi/skeemi peal skripti, siis genereeritakse tagarakenduse kasutajaliideses uued laused automaatselt, ilma, et kasutaja peaks tegema lisatoiminguid.

Samuti võib kasutaja lisada *LanguageGenerationScript.xml* faili mitmesse EA projekti ning genereerida vaheldumisi lauseid mitmest EA projektist. Kui need skriptid kirjutavad JSON formaadis infot samasse faili (iga järgmine kirjutamine kirjutab eelmise kirjutamise tulemuse üle), siis selle kasutajad näevad vaheldumisi erinevate EA projektide põhjal genereeritud lauseid.

Lisatud keelte lausete malle saab hiljem muuta/täiendada läbi keelefailide, mis asuvad tagarakendusega samas kaustas. Joonis 13 näha olevad eestikeelsed laused on genereeritud tarkvarakomplektiga kaasa tuleva keelefaili *Estonian.txt* põhjal. Näitena muudab autor keelefaili *Estonian.txt* nii, et Joonis 13 näha olevate eestikeelsete lausete puhul kasutatakse sõna „iseloomustab“ asemel sõna „kirjeldab“.

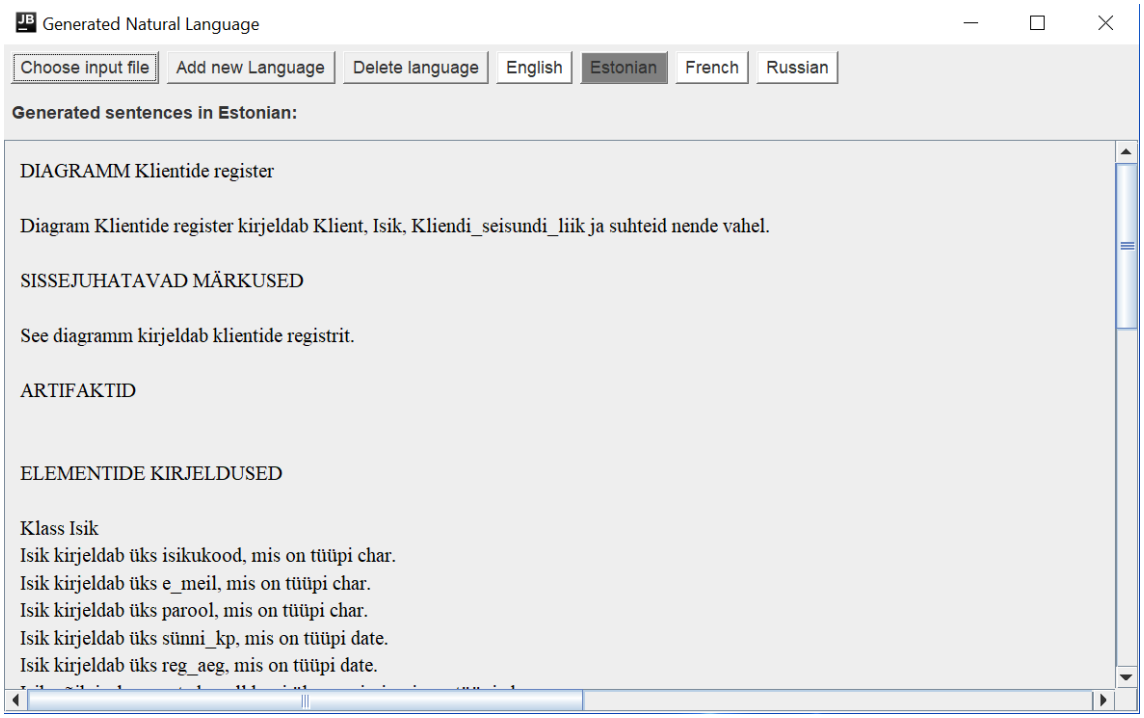
Autor avas tekstifaili *Estonian.txt* tekstiredaktoris, näiteks *Notepad*-is. Kõige kiirem viis vajaliku koha leidmiseks on vajutada CTRL ja F klahvikombinatsiooni ning sisestada sõna, mida soovitakse muuta. Tuleb vajutada nuppu *Find next,* kuni leitakse soovitud lause. Seejärel tuleb muuta soovitud sõna (Joonis 43). Juhul kui lause malli struktuur ei ole sobiv, siis võib terve lause malli ümber kirjutada. Oluline on nende muudatuste käigus mitte eemaldada failist ühtegi *%s* märgijada, sest see on kohahoidja, mille asemele sisestatakse mudelist loetud andmed.



Joonis 43. Näide keelefaili muutmisest.

Autor salvestas tekstifaili ning käivitas uuesti tagarakenduse. Ta valis tööribalt keelevalikuks *Estonian*. Elementide kirjelduse alajaotuses on nüüd lausetes sõna „iseloomustab“ asemel sõna „kirjeldab“ nagu on näha Joonis 44.

Kokkuvõttes, kui täpsustuvad soovid, kuidas peaks ühte või teist klassidiagrammi aspekti loomulikus keeles väljendama, siis saab vähemalt osasid parandusi teha ilma tarkvara ennast muutmata.



Joonis 44. Kasutajaliideses kuvatud laused peale *Estonian.txt* faili sisu muutmist.

Kui leitakse, et lisatud keelt pole enam vaja, siis võib selle ära kustutada. Selleks on kasutajaliideses tööribal *Delete language* nupp, mis viib keelte kustutamise vaatesse (Joonis 17).

Keelte kustutamise vaates on vasakul tekstifaili nimi ning paremal punane nupp kirjaga *Remove*. Vajutades nupul *Remove* kustutatakse vastav tekstifail tagarakenduse kaustast ning see keelevalik kaob avavaate tööribalt.

Tarkvara on testitud ja töötab nii EA 12 kui ka töö kirjutamise hetkel (2021. aasta sügis) värskeimas EA 15 versioonis. Ainukesed väikesed erinevused on tarkvara installeerimise ning EAs skripti käivitamise puhul, mis on välja toodud peatükis 7.

Autor testis tarkvara ka osaliselt venekeelsete lausete mallidega keelefailiga ja veendus, et ka mitte-ladina tähtede kasutamise korral rakendus töötab.

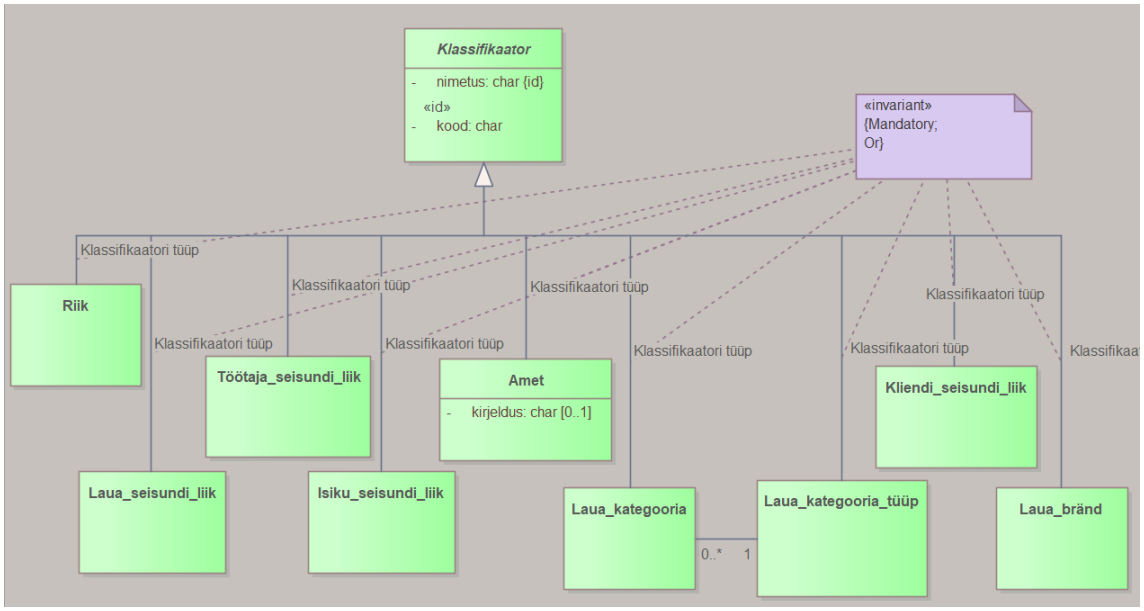
8.4 Lausete genereerimise näide

Järgnevalt esitatakse tarkvara genereeritud laused autori poolt „Andmebaasid I“ õppeaine projektis [33] loodud olemi-suhte diagrammidest e skeemidest. Tegemist on eestikeelsete lausetega. Need skeemid on tükeldatud registripõhiselt. Iga skeemi järel esitatakse selle

põhjal genereeritud lausendid. Kasutatud EA projektifail on leitav ka GitHub-i salvest: https://github.com/alvaltna/NaturalLanguageGenerator failist nimega *diagrammid.eap.*

8.4.1 Klassifikaatorite register

Joonisel 45 esitatakse klassifikaatorite registri olemi-suhte diagramm.



Joonis 45. Autori „Andmebaasid I“ õppeaine projekti klassifikaatorite registri analüüs.

Joonisel 46 – 50 esitatakse tarkvara genereeritud laused Joonisel 45 oleva skeemi kohta.

SKEEM Klassifikaatorite register

Skeem Klassifikaatorite register kirjeldab Klassifikaator, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik, Amet, Laua_kategooria, Laua_bränd, Laua_kategooria_tüüp, Kliendi_seisundi_liik ja suhteid nende vahel.

ELEMENTIDE KIRJELDUSED

Klass Amet
Amet võib iseloomustada null kuni üks kirjeldus, mis on tüüpi char.

Klass Isiku_seisundi_liik

Klass Klassifikaator
Klassifikaator iseloomustab üks kood, mis on tüüpi char.
Klassifikaator iseloomustab üks nimetus, mis on tüüpi char.

Klass Kliendi_seisundi_liik

Klass Laua_bränd

Klass Laua_kategooria

Klass Laua_kategooria_tüüp

Klass Laua_seisundi_liik

Joonis 46. Klassifikaatorite registrist genereeritud laused.

Klass Riik

Klass Töötaja_seisundi_liik

ÜHENDAJAD

Iga Klassifikaator peab omama vähemalt ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.
Iga Klassifikaator võib samal ajal omada ainult ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.

Klass Amet
Amet on Klassifikaator.
Piirang: Mandatory; Or.
Iga Amet võib olla seotud mitme Töötaja.
Mõni Amet ei ole seotud mitte ühegi Töötaja.

Klass Isiku_seisundi_liik
Isiku_seisundi_liik on Klassifikaator.
Piirang: Mandatory; Or.
Iga Isiku_seisundi_liik võib olla seotud mitme Isik.

Joonis 47. Klassifikaatorite registrist genereeritud laused.

Mõni Isiku_seisundi_liik ei ole seotud mitte ühegi Isik.

Klass Klassifikaator
Klassifikaator võib olla Riik.
Piirang: Mandatory; Or.
Klassifikaator võib olla Laua_seisundi_liik.
Piirang: Mandatory; Or.
Klassifikaator võib olla Töötaja_seisundi_liik.
Piirang: Mandatory; Or.
Klassifikaator võib olla Isiku_seisundi_liik.
Piirang: Mandatory; Or.
Klassifikaator võib olla Amet.
Piirang: Mandatory; Or.
Klassifikaator võib olla Laua_kategooria.
Piirang: Mandatory; Or.
Klassifikaator võib olla Laua_kategooria_tüüp.
Piirang: Mandatory; Or.
Klassifikaator võib olla Kliendi_seisundi_liik.
Piirang: Mandatory; Or.
Klassifikaator võib olla Laua_bränd.
Piirang: Mandatory; Or.

Klass Kliendi_seisundi_liik
Iga Kliendi_seisundi_liik võib olla seotud mitme Klient.
Mõni Kliendi_seisundi_liik ei ole seotud mitte ühegi Klient.
Kliendi_seisundi_liik on Klassifikaator.
Piirang: Mandatory; Or.

Klass Laua_bränd
Laua_bränd on Klassifikaator.
Piirang: Mandatory; Or.
Iga Laua_bränd võib olla seotud mitme Laud.
Mõni Laua_bränd ei ole seotud mitte ühegi Laud.

Joonis 48. Klassifikaatorite registrist genereeritud laused.

Klass Laua_kategooria
Laua_kategooria on Klassifikaator.
Piirang: Mandatory; Or.
Iga Laua_kategooria võib olla seotud mitme Laud.
Mõni Laua_kategooria ei ole seotud mitte ühegi Laud.
Iga Laua_kategooria peab olema seotud täpselt üks Laua_kategooria_tüüp.

Klass Laua_kategooria_tüüp
Iga Laua_kategooria_tüüp võib olla seotud mitme Laua_kategooria.
Mõni Laua_kategooria_tüüp ei ole seotud mitte ühegi Laua_kategooria.
Laua_kategooria_tüüp on Klassifikaator.
Piirang: Mandatory; Or.

Klass Laua_seisundi_liik
Laua_seisundi_liik on Klassifikaator.
Piirang: Mandatory; Or.
Iga Laua_seisundi_liik võib olla seotud mitme Laud.
Mõni Laua_seisundi_liik ei ole seotud mitte ühegi Laud.

Klass Riik
Riik on Klassifikaator.
Piirang: Mandatory; Or.
Iga Riik kui isikukoodi_riik võib olla seotud mitme Isik.
Mõni Riik kui isikukoodi_riik ei ole seotud mitte ühegi Isik.

Klass Töötaja_seisundi_liik

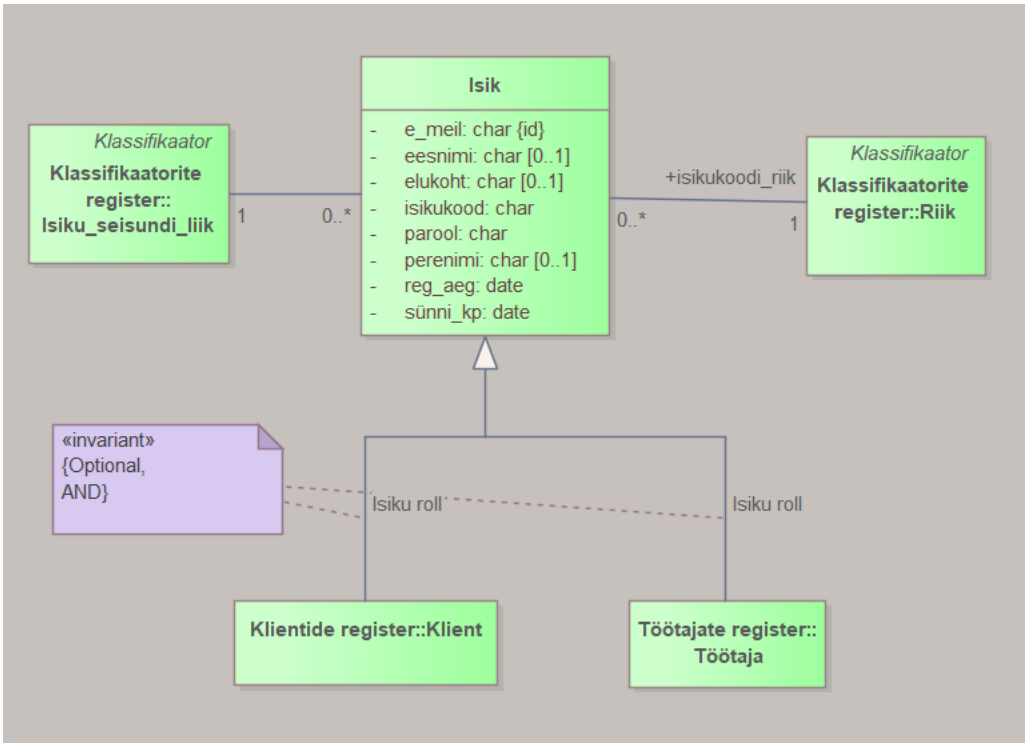
Joonis 49. Klassifikaatorite registrist genereeritud laused.

Töötaja_seisundi_liik on Klassifikaator.
Piirang: Mandatory; Or.
Iga Töötaja_seisundi_liik võib olla seotud mitme Töötaja.
Mõni Töötaja_seisundi_liik ei ole seotud mitte ühegi Töötaja.

Joonis 50. Klassifikaatorite registrist genereeritud laused.

8.4.2 Isikute register

Joonisel 51 esitatakse isikute registri olemi-suhte diagramm.



Joonis 51. Autori „Andmebaasid I“ õppeaine projekti isikute registri analüüs.

Joonisel 52 - 54 esitatakse tarkvara genereeritud laused Joonisel 51 oleva skeemi kohta.

SKEEM Isikute register

Skeem Isikute register kirjeldab Isik, Riik, Isiku_seisundi_liik, Klient, Töötaja ja suhteid nende vahel.

ELEMENTIDE KIRJELDUSED

Klass Isik
Isik iseloomustab üks isikukood, mis on tüüpi char.
Isik iseloomustab üks e_meil, mis on tüüpi char.
Isik iseloomustab üks parool, mis on tüüpi char.
Isik võib iseloomustada null kuni üks eesnimi, mis on tüüpi char.
Isik võib iseloomustada null kuni üks perenimi, mis on tüüpi char.
Isik iseloomustab üks sünni_kp, mis on tüüpi date.
Isik võib iseloomustada null kuni üks elukoht, mis on tüüpi char.
Isik iseloomustab üks reg_aeg, mis on tüüpi date.

Klass Isiku_seisundi_liik

Klass Klient
Klient iseloomustab üks on_nõus_tülitamisega, mis on tüüpi boolean algväärtusega false.

Klass Riik

Klass Töötaja

Joonis 52. Isikute registrist genereeritud laused.

ÜHENDAJAD

Iga Klassifikaator peab omama vähemalt ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.
Iga Klassifikaator võib samal ajal omada ainult ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.

Iga Isik ei pea omama Isiku roll tüüpi järgnevatest: Töötaja, Klient.
Iga Isik võib samal ajal omada rohkem kui ühte Isiku roll tüüpi järgnevatest: Töötaja, Klient.

Klass Isik
Iga Isik peab olema seotud täpselt üks Riik kui isikukoodi_riik.
Iga Isik peab olema seotud täpselt üks Isiku_seisundi_liik.
Isik võib olla Töötaja.
Piirang: Optional, AND.
Isik võib olla Klient.
Piirang: Optional, AND.

Klass Isiku_seisundi_liik
Isiku_seisundi_liik on Klassifikaator.
Iga Isiku_seisundi_liik võib olla seotud mitme Isik.
Mõni Isiku_seisundi_liik ei ole seotud mitte ühegi Isik.

Klass Klient
Klient on Isik.
Piirang: Optional, AND.
Iga Klient peab olema seotud täpselt üks Kliendi_seisundi_liik.

Joonis 53. Isikute registrist genereeritud laused.

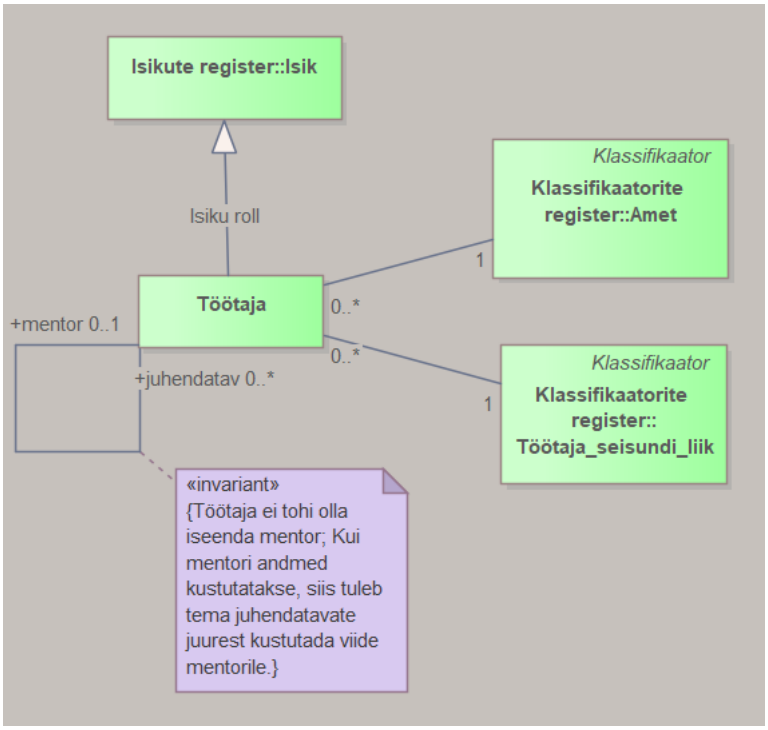
Klass Riik
Riik on Klassifikaator.
Iga Riik kui isikukoodi_riik võib olla seotud mitme Isik.
Mõni Riik kui isikukoodi_riik ei ole seotud mitte ühegi Isik.

Klass Töötaja
Töötaja on Isik.
Piirang: Optional, AND.
Iga Töötaja peab olema seotud täpselt üks Amet.
Iga Töötaja peab olema seotud täpselt üks Töötaja_seisundi_liik.
Iga Töötaja võib olla seotud mitme Laud.
Mõni Töötaja ei ole seotud mitte ühegi Laud.
Iga Töötaja kui juhendatav võib olla seotud koige rohkem ühe Töötaja kui mentor.
Mõni Töötaja kui juhendatav ei ole seotud Töötaja kui mentor.

Joonis 54. Isikute registrist genereeritud laused.

8.4.3 Töötajate register

Joonisel 55 esitatakse töötajate registri olemi-suhte diagramm.



Joonis 55. Autori „Andmebaasid I“ õppeaine projekti töötajate registri analüüs.

Joonisel 56 - 58 esitatakse tarkvara genereeritud laused Joonisel 55 oleva skeemi kohta.

SKEEM Töötajate register

Skeem Töötajate register kirjeldab Töötaja, Isik, Töötaja_seisundi_liik, Amet ja suhteid nende vahel.

ELEMENTIDE KIRJELDUSED

Klass Amet
Amet võib iseloomustada null kuni üks kirjeldus, mis on tüüpi char.

Klass Isik
Isik iseloomustab üks isikukood, mis on tüüpi char.
Isik iseloomustab üks e_meil, mis on tüüpi char.
Isik iseloomustab üks parool, mis on tüüpi char.
Isik võib iseloomustada null kuni üks eesnimi, mis on tüüpi char.
Isik võib iseloomustada null kuni üks perenimi, mis on tüüpi char.
Isik iseloomustab üks sünni_kp, mis on tüüpi date.
Isik võib iseloomustada null kuni üks elukoht, mis on tüüpi char.
Isik iseloomustab üks reg_aeg, mis on tüüpi date.

Klass Töötaja

Klass Töötaja_seisundi_liik

Joonis 56. Töötajate registrist genereeritud laused.

ÜHENDAJAD

Iga Klassifikaator peab omama vähemalt ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.

Iga Klassifikaator võib samal ajal omada ainult ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.

Iga Isik ei pea omama Isiku roll tüüpi järgnevatest: Töötaja, Klient.
Iga Isik võib samal ajal omada rohkem kui ühte Isiku roll tüüpi järgnevatest: Töötaja, Klient.

Klass Amet
Amet on Klassifikaator.
Iga Amet võib olla seotud mitme Töötaja.
Mõni Amet ei ole seotud mitte ühegi Töötaja.

Klass Isik
Iga Isik peab olema seotud täpselt üks Riik kui isikukoodi_riik.
Iga Isik peab olema seotud täpselt üks Isiku_seisundi_liik.
Isik võib olla Töötaja.
Isik võib olla Klient.

Klass Töötaja
Töötaja on Isik.
Iga Töötaja peab olema seotud täpselt üks Amet.
Iga Töötaja peab olema seotud täpselt üks Töötaja_seisundi_liik.
Iga Töötaja võib olla seotud mitme Laud.
Mõni Töötaja ei ole seotud mitte ühegi Laud.
Iga Töötaja kui juhendatav võib olla seotud koige rohkem ühe Töötaja kui mentor.
Mõni Töötaja kui juhendatav ei ole seotud Töötaja kui mentor.

Joonis 57. Töötajate registrist genereeritud laused.

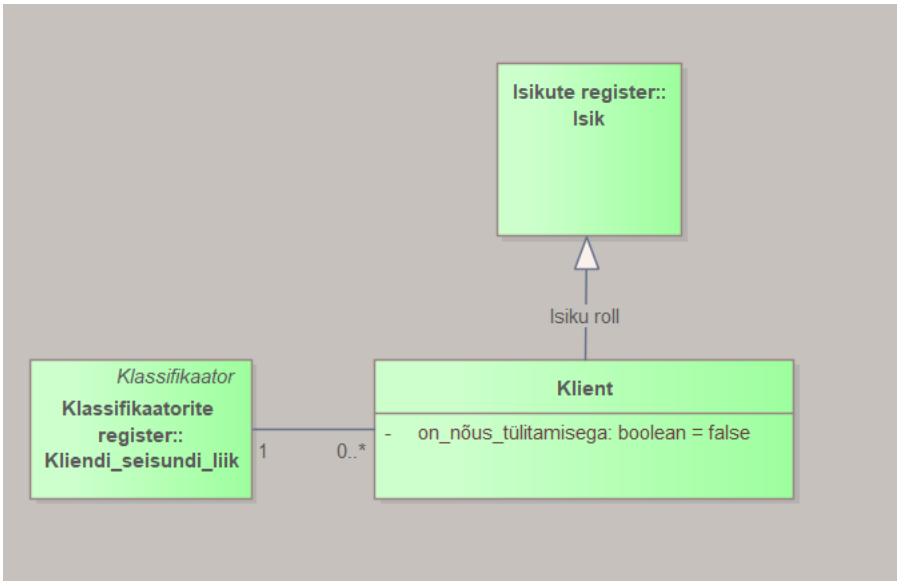
Piirang: Töötaja ei tohi olla iseenda mentor; Kui mentori andmed kustutatakse, siis tuleb tema juhendatavate juurest kustutada viide mentorile..

Klass Töötaja_seisundi_liik
Töötaja_seisundi_liik on Klassifikaator.
Iga Töötaja_seisundi_liik võib olla seotud mitme Töötaja.
Mõni Töötaja_seisundi_liik ei ole seotud mitte ühegi Töötaja.

Joonis 58. Töötajate registrist genereeritud laused.

8.4.4 Klientide register

Joonisel 59 esitatakse klientide registri olemi-suhte diagramm.



Joonis 59. Autori „Andmebaasid I“ õppeaine projekti klientide registri analüüs.

Joonisel 60 - 61 esitatakse tarkvara genereeritud laused Joonisel 59 oleva skeemi kohta.

SKEEM Klientide register

Skeem Klientide register kirjeldab Isik, Klient, Kliendi_seisundi_liik ja suhteid nende vahel.

ELEMENTIDE KIRJELDUSED

Klass Isik
Isik iseloomustab üks isikukood, mis on tüüpi char.
Isik iseloomustab üks e_meil, mis on tüüpi char.
Isik iseloomustab üks parool, mis on tüüpi char.
Isik võib iseloomustada null kuni üks eesnimi, mis on tüüpi char.
Isik võib iseloomustada null kuni üks perenimi, mis on tüüpi char.
Isik iseloomustab üks sünni_kp, mis on tüüpi date.
Isik võib iseloomustada null kuni üks elukoht, mis on tüüpi char.
Isik iseloomustab üks reg_aeg, mis on tüüpi date.

Klass Kliendi_seisundi_liik

Klass Klient
Klient iseloomustab üks on_nõus_tülitamisega, mis on tüüpi boolean algväärtusega false.

Joonis 60. Klientide registrist genereeritud laused.

ÜHENDAJAD

Iga Klassifikaator peab omama vähemalt ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.

Iga Klassifikaator võib samal ajal omada ainult ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.

Iga Isik ei pea omama Isiku roll tüüpi järgnevatest: Töötaja, Klient.
Iga Isik võib samal ajal omada rohkem kui ühte Isiku roll tüüpi järgnevatest: Töötaja, Klient.

Klass Isik
Iga Isik peab olema seotud täpselt üks Riik kui isikukoodi_riik.
Iga Isik peab olema seotud täpselt üks Isiku_seisundi_liik.
Isik võib olla Töötaja.
Isik võib olla Klient.

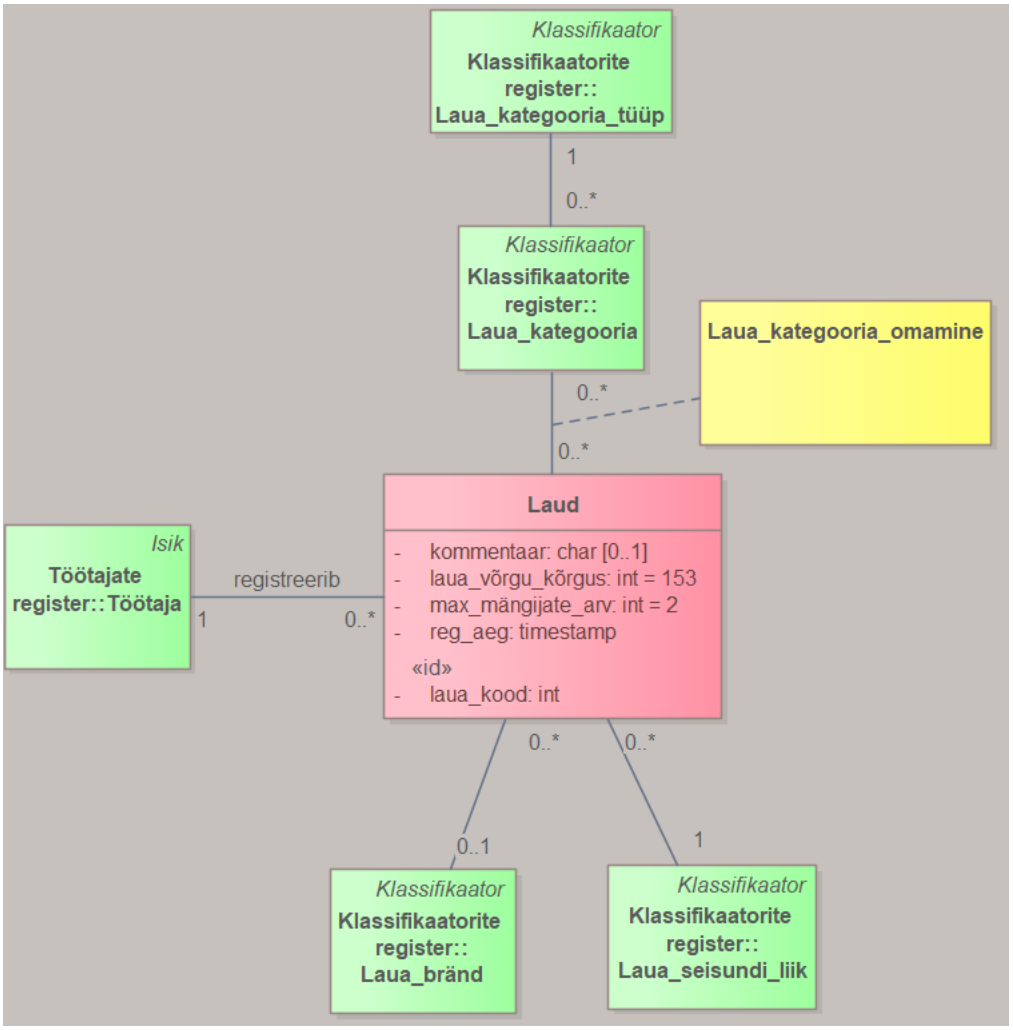
Klass Kliendi_seisundi_liik
Iga Kliendi_seisundi_liik võib olla seotud mitme Klient.
Mõni Kliendi_seisundi_liik ei ole seotud mitte ühegi Klient.
Kliendi_seisundi_liik on Klassifikaator.

Klass Klient
Klient on Isik.
Iga Klient peab olema seotud täpselt üks Kliendi_seisundi_liik.

Joonis 61. Klientide registrist genereeritud laused.

8.4.5 Laudade register

Joonisel 62 esitatakse laudade registri olemi-suhte diagramm.



Joonis 62. Autori „Andmebaasid I“ õppeaine projekti laudade registri analüüs.

Joonisel 63 – 65 esitatakse tarkvara genereeritud laused Joonisel 62 oleva skeemi kohta.

SKEEM Laudade register

Skeem Laudade register kirjeldab Töötaja, Laud, Laua_seisundi_liik, Laua_kategooria, Laua_kategooria_omamine, Laua_bränd, Laua_kategooria_tüüp ja suhteid nende vahel.

ELEMENTIDE KIRJELDUSED

Klass Laua_bränd

Klass Laua_kategooria

Klass Laua_kategooria_omamine

Klass Laua_kategooria_tüüp

Klass Laua_seisundi_liik

Klass Laud
Laud iseloomustab üks laua_kood, mis on tüüpi int.
Laud võib iseloomustada null kuni üks kommentaar, mis on tüüpi char.
Laud iseloomustab üks reg_aeg, mis on tüüpi timestamp.
Laud iseloomustab üks laua_võrgu_kõrgus, mis on tüüpi int algväärtusega 153.
Laud iseloomustab üks max_mängijate_arv, mis on tüüpi int algväärtusega 2.

Klass Töötaja

Joonis 63. Laudade registrist genereeritud laused.

ÜHENDAJAD

Iga Klassifikaator peab omama vähemalt ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.
Iga Klassifikaator võib samal ajal omada ainult ühte Klassifikaatori tüüp tüüpi järgnevatest: Amet, Laua_kategooria, Laua_kategooria_tüüp, Kliendi_seisundi_liik, Laua_bränd, Riik, Laua_seisundi_liik, Töötaja_seisundi_liik, Isiku_seisundi_liik.

Iga Isik ei pea omama Isiku roll tüüpi järgnevatest: Töötaja, Klient.
Iga Isik võib samal ajal omada rohkem kui ühte Isiku roll tüüpi järgnevatest: Töötaja, Klient.

Klass Laua_bränd
Laua_bränd on Klassifikaator.
Iga Laua_bränd võib olla seotud mitme Laud.
Mõni Laua_bränd ei ole seotud mitte ühegi Laud.

Klass Laua_kategooria
Laua_kategooria on Klassifikaator.
Iga Laua_kategooria võib olla seotud mitme Laud.
Mõni Laua_kategooria ei ole seotud mitte ühegi Laud.
Iga Laua_kategooria peab olema seotud täpselt üks Laua_kategooria_tüüp.

Klass Laua_kategooria_omamine

Klass Laua_kategooria_tüüp
Iga Laua_kategooria_tüüp võib olla seotud mitme Laua_kategooria.

Joonis 64. Laudade registrist genereeritud laused.

Mõni Laua_kategooria_tüüp ei ole seotud mitte ühegi Laua_kategooria.
Laua_kategooria_tüüp on Klassifikaator.

Klass Laua_seisundi_liik
Laua_seisundi_liik on Klassifikaator.
Iga Laua_seisundi_liik võib olla seotud mitme Laud.
Mõni Laua_seisundi_liik ei ole seotud mitte ühegi Laud.

Klass Laud
Iga Laud peab olema seotud täpselt üks Töötaja.
Iga Laud peab olema seotud täpselt üks Laua_seisundi_liik.
Iga Laud võib olla seotud mitme Laua_kategooria.
Mõni Laud ei ole seotud mitte ühegi Laua_kategooria.
Iga Laud võib olla seotud koige rohkem ühe Laua_bränd.
Mõni Laud ei ole seotud Laua_bränd.

Klass Töötaja
Töötaja on Isik.
Iga Töötaja peab olema seotud täpselt üks Amet.
Iga Töötaja peab olema seotud täpselt üks Töötaja_seisundi_liik.
Iga Töötaja võib olla seotud mitme Laud.
Mõni Töötaja ei ole seotud mitte ühegi Laud.
Iga Töötaja kui juhendatav võib olla seotud koige rohkem ühe Töötaja kui mentor.
Mõni Töötaja kui juhendatav ei ole seotud Töötaja kui mentor.

Joonis 65. Laudade registrist genereeritud laused.

9 Arendusvaade

Selles peatükis kirjutab autor olemasoleva tarkvara puudustest ning sellest, millist funktsionaalsust võiks tulevikus veel juurde lisada.

9.1 Olemasoleva lahenduse puudused

Üldine puudus, mis tulenes töö alguses tehtud valikutest on see, et lahendus töötab ühe konkreetse (sealhulgas tasulise) CASE vahendiga.

Ülekantavuse testimisega seoses on puuduseks, et tarkvara tööd ei testitud macOS ja Linux operatsioonisüsteemidel, kuigi EA vahendit saab põhimõtteliselt ka nendel käivitada [33].

Tarkvaral on puudusi, mida autor ei jõudnud arenduse käigus kõrvaldada või vastavat funktsionaalsust realiseerida. Üheks neist on see, et kui lisada uus keel, siis ei pruugi genereeritud loomuliku keele laused olla selles keeles täiesti grammatiliselt korrektsed. Näiteks kui tõlkida kasutajaliideses uue keele lisamise mallis kõik laused eesti keelde ja kasutada tekkinud faili lausete genereerimisel, siis lausetes puuduvad eesti keele käändelõpud EAst skeemidelt/pakettidest lausetesse tulevatelt sõnadelt.

Järgnev on näide lausetest, mida praegune tarkvara versioon genereerib.

- Isik iseloomustab üks isikukood, mis on tüüpi char.
- Iga Isik peab olema seotud täpselt üks Isiku_seisundi_liik.

Tegelikult võiksid laused olla sellised.

- Isik**ut** iseloomustab üks isikukood, mis on tüüpi char.
- Iga Isik peab olema seotud täpselt ü**he** Isiku_seisundi_liigi**ga**.

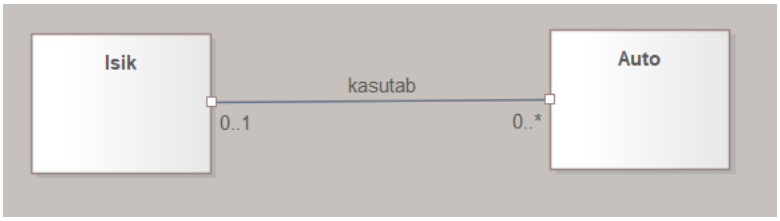
Sellele probleemile ei ole kerget lahendust, sest isegi kui lisada tarkvarasse eesti keele käändelõppude tugi, on kõikide keelte kõikide iseärasustega arvestamine väga keeruline

ja mahukas ülesanne. Võib-olla oleks kõige mõistlikum lisada täiendav tugi kõige laiemalt kasutatavatele keeltele ning ülejäänud keelte puhul jätta genereerimise protsess selliseks nagu see on hetkel.

Teise puudusena võiks välja tuua tarkvara vähese veatöötluse. Iga kord kui kasutaja sisestab EA skeemile infot, mida tarkvara ei oska kasutada, ei genereeri tarkvara ühtegi lauset. Näiteks kui kasutaja lisab sidemele võimsustiku, mida tarkvara ei oska kasutada (näiteks võimsustiku *0..1* asemel võimsustik *null..üks*), siis tarkvara ei näita kasutajaliideses veateadet, vaid lihtsalt ei genereeri ühtegi lauset. Kui kasutajaliideses kuvataks selle asemel genereeritud lausete puudumise põhjust selgitav veateade, siis saaks kasutaja kiiremini teada, kus on viga ning mida tuleks teistsuguse tulemuse saamiseks EA skeemil muuta.

Kolmandaks puuduseks on see, et infot JSON kujul väljundi faili kirjutav skript on aeglane juhul kui tuleb kirjutada infot paljudest pakettidest. Tagarakendus kontrollib iga kahe sekundi järel, kas skripti väljundfaili sisu on muutunud. Probleemi tekkimisele osutab see, et peale paljudest pakettidest väljundi genereerimist trükib tagarakendus neli kuni viis korda järjest konsoolile info, et väljundfaili sisu on muutunud. See tähendab, et skript ikka veel kirjutab väljundfaili uut infot. See teeb suuremahuliste EA projektide puhul loomuliku keele lausete genereerimise ja kuvamise aeglaseks.

Neljandaks puuduseks on see, et generaator ei kasuta loodud lausetes ühendajate nimesid. Põhjus on selles, et nende nimede kasutamine võib viia ebakorrektsete lausete genereerimiseni, mis selgitavad modelleeritavat reaalsust valesti. Näiteks Joonisel 66 oleva klassidiagrammi koostaja tahtis väljendada, et käesoleval ajahetkel kasutab iga autot 0..1 isikut.



Joonis 66. Ühendaja koos nimega.

Lausete koostamisel pole samas infot, kas isik kasutab autot või auto kasutab isikut. Sellises mudelis pole määratud, mis suunas tuleb sõna „kasutab“ lugeda. Võimalikuks

tulevaseks lahenduseks oleks määrata sellisele seosele suund (*Direction*) ja arvestada sellega lausete genereerimisel.

Viiendaks puuduseks on see, et peale keele lausete malle sisaldava tekstifaili loomist ei saa seda malli muuta rakenduse kaudu, vaid selleks tuleb tekstifail avada tekstiredaktoris. Otse tekstifailiga töötades ja seal muudatusi tehes võib muutujale jääda ebaselgeks, milliste mudelielementide kohta käivaid lauseid mingi muudatus täpsemalt mõjutab. Selline ebaselgus võib tekitada vigu tõlkimisel.

Kuuendaks puuduseks on see, et tarkvara kasutajaliides on ainult inglise keeles ja tarkvarasse pole sisseehitatud võimalust kasutajaliidese lihtsaks tõlkimiseks teistesse keeltesse.

9.2 Tarkvara edasine arendus

Lisaks eelmises jaotises toodud puuduste kõrvaldamisele on tarkvara veel edasi arendada.

UML 2.5 keel kirjeldab 14 skeemi e skeemi tüüpi [35] EA on väga laialdaste võimalustega tööriist, mis toetab lisaks UMLile ka teisi keeli (nt SysML ja Archimate). Seetõttu on veel palju mudelielemente, mille kohta autori tarkvara praegusel hetkel ei oska ühtegi loomuliku keele lauset genereerida.

Tulevikus võiks tarkvara suuta lauseid genereerida, mitte ainult UML klassiskeemide kohta, aga kõikide UML skeemi tüüpide kohta, mida EA toetab. Selle realiseerimine hõlmab endas skripti täiendamist nii, et skript päriks EAlt infot rohkemate mudelielementide kohta. Tagarakenduses tuleks põhiliselt täiendada *Phrases* klassi rohkemate sõnade ja fraasidega, mis moodustaksid loomuliku keele laused uutele EA elementidele ja ühendajatele, milledesse sisestatakse skriptist tulnud andmed. Samuti tuleks täiendada lähtekoodis *sentenceGenerator* paketis *SenteneceBuilder* klassi nii, et see suudaks luua korrektsed loomuliku keele laused, tuginedes EAst tulnud uuele infole ning *Phrases* objekti klassiväljade sõnadele, fraasidele ja lausetele.

Kindlasti on EAs toetatud keeltes elemente, mille verbaliseerimiseks tuleb tarkvara lähtekoodi lisada uusi klasse või koguni uusi pakke. Kuid vähemalt on loodud tarkvara näol olemas aluspõhi, mida on võimalik laiendada.

Lisaks sellele võiks täiendada keele lisamise funktsionaalsust, et loodud loomuliku keele laused oleksid grammatiliselt korrektsemad. Üheks viisiks on eelmises jaotises esitatud idee lisada täiendav tugi kõige laialdasemalt kasutusel olevatele keeltele. Teine viis on keele lisamisel võimaldada lisada infot selliste grammatiliste konstruktsioonide (nagu näiteks käändelõpud) kohta, mis on kasutusel paljudes keeltes.

Nagu on nimetatud peatükis 4 võiks üheks tuleviku uurimissuunaks olla ka üldlahenduse loomine, mis töötab erinevate CASE vahenditega.

10 Kokkuvõte

Bakalaureusetöö eesmärgiks oli luua Enterprise Architecti (EA) CASE vahendile (modelleerimisvahendile) täiendus, mis genereeriks UML klassiskeemidest loomuliku keele (inimkeele) laused. Mudelite ja lausete poolt esitatav info peab olema võimalikult samasugune, st muutub mudeli esitusviis, kuid selle juures peab infokadu olema võimalikult väike. Täiendus peab suutma genereerida lauseid inglise keeles ja eesti keeles ning kasutajatel peab olema võimalus lisada uusi keeli. Täiendus ei tõlgi mudelielemente, vaid kasutab olemasolevaid mudelielementide nimesid lausete koostamiseks.

Töö tulemusena valmis tarkvara, mis on võimeline UML klassiskeemidest genereerima loomuliku keele lauseid nii inglise keeles kui ka eesti keeles. Kasutaja saab lisada uusi keeli ning tarkvara on võimeline igas lisatud keeles genereerima loomuliku keele lauseid. Uute keelte lisamine toimub läbi malli, mille abil luuakse keelefail. Töö alamtulemuseks on klassidiagrammi kirjeldavate lausete eestikeelne versioon.

Töö tulemust kontrolliti ühiktestidega, testides tarkvara suuremahulisemate klassiskeemide peal ning lastes kolmel Tallinna Tehnikaülikooli üliõpilasel joonistada klassiskeeme tarkvara genereeritud lausete põhjal. Seejärel pidid osalised vastama küsimustele. Kõikide nende kontrollimiste tulemusena julgeb autor väita, et töös püstitatud eesmärk saavutati ning tarkvara vastab töö sissejuhatuses ja peatükis 3 välja toodud nõuetele. Tarkvara töötamist on testitud EA 12-s ja EA 15-s.

Tegemist on avatud lähtekoodiga vaba tarkvaraga, millel on MIT litsents. Lähtekood koos ühiktestidega ning samuti kompileeritud tarkvara ning installeerimise juhised on leitavad GitHub-i salvest: https://github.com/alvaltna/NaturalLanguageGenerator.

Tarkvara saab täiendada ja paremaks muuta. Järgmisteks sammudeks võiks olla tarkvara funktsionaalsuse laiendamine, et saaks genereerida loomuliku keele lauseid ka teiste UML skeemi tüüpide põhjal ning keelelise toe täiendamine nii, et lisatud keeltesse tõlgitud laused oleksid grammatiliselt korrektsemad.

Kasutatud kirjandus

[1] Vikipeedia, vaba entsüklopeedia, *Abstarct Window Toolkit*, 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/Abstract_Window_Toolkit Kasutatud: 26.12.2021

[2] Vikipeedia, vaba entsüklopeedia, *Gradle*, 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/Gradle Kasutatud: 26.12.2021

[3] Vikipeedia, vaba entsüklopeedia, *HTML* 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/HTML Kasutatud 26.12.2021

[4] Vikipeedia, vaba entsüklopeedia, *JavaScript*, 2021, [Online]. Loetud aadressil: https://et.wikipedia.org/wiki/JavaScript Kasutatud 26.12.2021

[5] Vikipeedia, vaba entsüklopeedia, *Java Development Kit*, 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/Java_Development_Kit Kasutatud 30.12.2021

[6] Vikipeedia, vaba entsüklopeedia, *JSON*, 2021, [Online]. Loetud aadressil: https://et.wikipedia.org/wiki/JSON Kasutatud: 26.12.2021

[7] Annaabi, *Loomulik keel*, 2022, [Online]. Loetud aadressil: https://annaabi.ee/loomulik-keel@ Kasutatud 04.01.2022

[8] OMG, *MDA-The Architecture of Choice for the Changing World*, 2021, [Online]. Loetud aadressil: https://www.omg.org/mda/#:~:text=Model%20Driven%20Architecture%C2%AE%20(MDA,logic%20from%20underlying%20platform%20technology Kasutatud: 26.12.2021

[9] Vikipeedia, vaba entsüklopeedia, MIT *License*, 2022, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/MIT_License Kasutatud: 03.01.2022

[10] Vikipeedia, vaba entsüklopeedia, *Object Management Group*, 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/Object_Management_Group Kasutatud: 26.12.2021

[11] Vikipeedia, vaba entsüklopeedia, *Swing(Java)*, 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/Swing_(Java) Kasutatud: 26.12.2021

[12] Vikipeedia, vaba entsüklopeedia, *Visual Basic*, 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/Visual_Basic Kasutatud: 26.12.2021

[13] Agile Modeling, *Agile Modeling (AM) Home Page*, 2022, [Online]. Loetud aadressil: http://www.agilemodeling.com/ Kasutatud: 03.01.2022

[14] Sparx Systems, *Third Party Extensions*, 2021, [Online]. Loetud aadressil: https://sparxsystems.com/products/3rdparty.html Kasutatud: 26.12.2021

[15] Verhovtsov, M., "Generation of Sentences Based on UML Class Diagrams" [Bakalaureusetöö], Infotehnoloogia teaduskond, Taltech, Tallinn, Eesti, 2011, [Online]. Loetud aadressil: https://maurus.ttu.ee/download.php?aine=346&document=36300&tyyp=do Kasutatud: 26.12.2021

[16] Cacoo, *Intro to UML 2.5 diagram types and templates*, 2021, [Online]. Loetud aadressil: https://cacoo.com/blog/intro-uml-diagram-types-templates/ Kasutatud 03.01.2022

[17] Managedagile, *Is UML Still Relevant Today? How Is it Used in an Agile Environment?*, 2021, [Online] Loetud aadressil: https://managedagile.com/is-uml-still-relevant-today/ Kasutatud: 26.12.2021

[18] Vikipeedia, vaba entsüklopeedia, *Class diagrams*, 2021, [Online]. Loetud aadressil: https://en.wikipedia.org/wiki/Class_diagram Kasutatud: 26.12.2021

[19] Reggio, G., Leotta, M., Ricca, F., Clerissi, D., "What are the used UML diagrams? A Preliminary Survey", *CEUR Workshop Proceedings*, vol 1078, [Online]. Loetud aadressil: http://ceur-ws.org/Vol-1078/paper1.pdf Kasutatud 03.01.2022

[20] Vikipeedia, vaba entsüklopeedia, *Enterprise Architect (software)*, 2021, [Online] Loetud aadressil: https://en.wikipedia.org/wiki/Enterprise_Architect_(software) Kasutatud 26.12.2021

[21] Sparx Systems, *Unified Modeling Language (UML)*, 2021, [Online]. Loetud aadressil: https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/whatisuml.html Kasutatud: 26.12.2021

[22] Nekrassova, M., "Andmebaasi arendamisega seotud mudeliteisenduste täiustamine Enterprise Architect CASE vahendis", [Bakalaureusetöö], Infotehnoloogia teaduskond, Taltech, Tallinn, Eesti, 2015, [Online]. Loetud aadressil: https://digi.lib.ttu.ee/i/?3454 Kasutatud 04.01.2022

[23] *OPCAT Version 3.0*, Getting started guide, OPCAT software, United States, 2007, [Online]. Loetud aadressil *http*://web.mit.edu/deweck/Public/opcat/OPCAT_Manual_3.0.pdf Kasutatud 03.01.2022

[24] Ghezai, A., Bruinsma, R., "Students with the Use of Automated Feedback for Software Design in E-learning Environments", [Bakalaureusetöö], Arvutiteaduse ja insenerinduse teaduskond, Gothenburgi ülikool, Chalmers tehnikaülikool, Gothenburg, Rootsi, 2017, [Online].
Loetud aadressil: https://gupea.ub.gu.se/bitstream/2077/52662/1/gupea_2077_52662_1.pdf Kasutatud: 26.12.2021

[25] Abdalazeim, A., Meziane, F., "A review of the generation of requirements specification in natural language using objects UML models and domain ontology", *Procedia Computer Science*, vol. 189, pp 328-334, 2021, [Online].
Loetud aadressil https://www.sciencedirect.com/science/article/pii/S1877050921012266 Kasutatud: 26.12.2021

[26] *Design science and action research*, 2016, [Online]. Loetud aadressil: https://www.youtube.com/watch?v=yYa5g50MHaY Kasutatud 03.01.2022

[27] Hevner, A.R., March, S.T., Park, J., Ram, S.: "Design science in information systems research", *MIS Quart*, vol. 28, pp. 75-105, 2004, [Online].
Loetud aadressil: https://core.ac.uk/download/pdf/81714032.pdf Kasutatud: 26.12.2021

[28] Mountain Goat Software, *User Stories*, 2022, , [Online]. Loetud aadressil: https://www.mountaingoatsoftware.com/agile/user-stories

[29] Stats, *Standardipõhine tarkvaratehnika sõnastik*, 2022, [Online]. Loetud aadressil: https://stats.cyber.ee/ Kasutatud 03.01.2022

[30] Vallaste, *e-teatmik*, 2022, [Online]. Loetud aadressil: http://www.vallaste.ee/ Kasutatud 03.01.2022

[31] Toose, A.,"Mudelivahetuse pendeltestimine MOF-i toetavates CASE vahendites kasutades UML 2.x ja XMI 2.1", [Magistritöö], Infotehnoloogia teaduskond, Taltech, Tallinn, Eesti, 2013 Kasutatud 04.01.2022

[32] Sparx Systems, *Repository class*, 2021, [Online]. Loetud aadressil: https://sparxsystems.com/enterprise_architect_user_guide/14.0/automation/repository3.html Kasutatud: 26.12.2021

[33] Maurus, *Andmebaasid I (ITI0206)*, 2022, [Online]. Loetud aadressil: https://maurus.ttu.ee/aine_index.php?aine=380 Kasutatud 04.01.2022

[34] Sparx Systems, *Installing Enterprise Architect Under Linux or macOS*, 2022, [Online]. Loetud aadressil: https://sparxsystems.com/enterprise_architect_user_guide/14.0/product_information/install_ea_wine.html Kasutatud 04.01.2022

[35] Uml-diagrams, *UML 2.5 Diagrams Overview*, 2020, [Online]. Loetud aadressil: https://www.uml-diagrams.org/uml-25-diagrams.html Kasutatud 03.01.2022

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Alvar Valtna

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Enterprise Architect CASE vahendi UML klassiskeemidest inimkeelsete lausete genereerimise tarkvara“, mille juhendaja on Erki Eessaar
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.12.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Verhovtsov, M., “Generation of Sentences Based on UML Class Diagrams” töös kirjeldatud laused

Diagram name describes *Class A*, *Class B, ...* and relationships between them.

Joonis 67 Töös väljatoodud skeemi kirjelduse lause.

Class is characterized by *M* to *N attributes 1*, …”.

The fact that the attribute is optional or its cardinality is “many” is better expressed using modal verbs and quantifiers like “at least” / “at most”. The following templates describe the case, where attribute 1 has multiplicity [1..*], attribute 2 – [N..*], attribute 3 – [0..1], attribute 4 – [0..N] and attribute 5 – [*].

Class is characterized by **at least one** *attribute 1* and **at least** *N attributes 2.*
Class **can** be characterized by *attribute 3*, **at most** *N attributes 4* and **many** *attributes 5*.

Joonis 68. Töös väljatoodud elemendi kirjelduse lause.

Superclass can be *subclass 1*, *subclass 2*, … or *superclass N*.

Joonis 69. Töös väljatoodud ülemklassi ühendajate kirjelduse lause.

Subclass is *a superclass 1, superclass 2* and *superclass N*.

Joonis 70. Töös väljatoodud alamklassi ühendajate kirjelduse lause

Class A depends on *class B*.

Joonis 71. Töös väljatoodud sõltuvuse ühendajate kirjelduse lause.

Driver uses car controls.

Joonis 72. Töös väljatoodud kasutatavuse ühendajate kirjelduse lause.

Steering wheel is car controls. Gas pedal is car controls.

Joonis 73. Töös väljatoodud realisastsiooni ühendajate kirjelduse lause.

*	Each *Class A* can be associated with many *Classes B*.
0..*	Each *Class A* can be associated with many *Classes B*. Some *Class A* is associated with no *Classes B*.
1..*	Each *Class A* must be associated with at least one *Class B*. Each *Class A* can be associated with many *Classes B*.
1	Each *Class A* must be associated with exactly one *Class B*.
0..1	Each *Class A* can be associated with at most one *Class B*. Some Class A is associated with no *Classes B*.
0..n	Each *Class A* can be associated with at most n *Class B*. Some *Class A* is associated with no *Classes B*.
1..n	Each *Class A* can be associated with at most *n Classes B*. Each *Class A* must be associated with at least one *Class B*.
n..m	Each *Class A* can be associated with at most *m Classes B*. Each *Class A* must be associated with at least *n Classes B*.
n	Each *Class A* must be associated with exactly *n Classes B*.

Joonis 74. Töös väljatoodud assotsiatsiooni ühendajate kirjelduse laused, kõikide võimsuste puhul.

A country has a city. A city belongs to a country.
Each country can have many cities. Each country must have at least one city.

A country has a capital city. A capital city belongs to a country.
Each country must have exactly one capital city.

Joonis 75. Töös väljatoodud liitmise ühendajate kirjelduse lause näited.

Class A as *Role A* is associated with *Class B* as *Role B*.

Joonis 76. Töös väljatoodud rolle sisaldavate ühendajate kirjeldamise lause.

Employee as subordinate is subordinate of employee as superior.

Joonis 77. Töös väljatoodud refleksiivse assotsiatsiooni ühendaja kirjeldamise lause koos rollidega.

Lisa 3 – Verhovtsov, M., “Generation of Sentences Based on UML Class Diagrams” töös kirjeldatud skeemi kohta koostatavate lausete esituse ülesehitus

1. Diagram name and enumeration of classes.
2. Diagram comments from the upper part of the diagram.
3. Each class, from more general to specific ones:
   - 3.1. Generalization and interface realization relationships.
   - 3.2. Class attributes.
   - 3.3. Class operations.
   - 3.4. Comments that only relate to this class.
4. Each usage dependency relationship.
5. Each association:
   - 5.1. Direct simple association sentence.
   - 5.2. (Optional) Reverse simple association sentence.
   - 5.3. If there are multiplicities, direct and reverse sentences that include multiplicities.
   - 5.4. Comments that only relate to this association.
6. Comments that related to multiple elements.
7. Diagram comments from the bottom part of the diagram.

Joonis 78. Töös väljatoodud üldine lausete jaotus ning kuvamise järjestus.