jamk.fi

Automatisoidut tietomuunnokset

Tekniikan ala

Sami Kauhala

Opinnäytetyö
Huhtikuu 2020
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Kauhala, Sami	Opinnäytetyö, AMK	huhtikuu 2020
	Sivumäärä 41	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Automatisoidut tietomuunnokset
Tekniikan ala

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Rantala Ari, Salmikangas Esa

Toimeksiantaja(t)

Tiivistelmä

Tutkimuksen ongelma syntyi terveydenhuollon projektista, jossa täytyi muuntaa tietoa useasta eri lähdejärjestelmästä yhteiseen jaettuun muotoon. Työssä oli tarkoituksena hankkia muunnettavaa tietoa, siistiä se käsiteltävään muotoon ja muuntaa tieto määriteltyyn malliin. Muunnokset toteutettiin manuaalisesti ja automaattisesti, jotta ajankäyttöä voitiin verrata ja analysoida. Tutkimuksen tavoitteena oli tarjota statistiikkaa tietomuunnoksien kustannustehokkuudesta ja menetelmien tehokkuudesta. Työ rajattiin tekstipohjaisen tiedon kartoittamiseen, parsimiseen ja muuntamiseen. Tuloksien avulla voitiin laskea kustannustehokkuuteen liittyviä laskelmia sekä analysoida automaattisten muunnosten vaikutusta oikean maailman projekteissa.

Tutkimus toteutettiin kahdessa osassa. Ensimmäinen osa käsitti tiedon hankinnan, ohjelmiston tuotannon ja tulosten hankinnan. Toinen osa oli tuloksien analyysi ja laskelmien toteuttaminen. Tutkimuksessa toteutettu ohjelmisto on räätälöity tehtävää varten, mutta ohjelmistossa on myös käytetty hyväksi olemassa olevaa kolmannen osapuolen ohjelmistoa. ANTLR-generaattori hoitaa tutkimuksessa tiedon parsinnan, ja se on liitettynä räätälöityihin tiedon muunnoksiin.

Työn lopputulokset osoittivat, että manuaaliset muunnokset ovat lähes aina eksponentiaalisesti hitaampia kuin automaattiset. Muunnosratkaisun ajankäyttö vaihtelee siten, että manuaaliset muunnokset vaativat vakioisen ajan ja automaattiset muunnokset vaativat projektin alkupäässä enemmän aikaa. Tutkimuksen keskeisin johtopäätös on se, että muunnoksien lukumäärän kasvaessa automaattiset muunnokset ovat aina parempi vaihtoehto kustannustehokkuuden kannalta kuin manuaaliset muunnokset.

Avainsanat (asiasanat)
tietomuunnokset, tiedon kartoitus, parsinta, generaattori

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Kauhala, Sami	Bachelor's thesis	April 2020
		Language of publication: FI
	Number of pages: 41	Permission for web publication: x

Title of publication
Automatic data transformations
Technology field

Degree programme
Information- and communication technology

Supervisor(s)
Rantala Ari, Salmikangas Esa

Assigned by

Abstract

The research problem was born from a healthcare project where we had to transform data from multiple separate source systems to a unified format. The study consists of acquiring data for the transformations, cleaning it to usable formats and performing the transformations to generate predefined output data. The transformations were done manually and automatically to generate transformation times for the study. The times were meant to be compared and analyzed to get statistics about the cost-efficiency of the transformations and transformation methods. The work was limited to using only text-based data. The data was used for mapping, parsing, and transformations. The results were used to calculate the cost-efficiency and analyze the impact of automated transformation in real-world projects.

The research was executed in two parts. The first part includes data gathering, software construction, and result gathering. The second part consists of result analysis and calculation. The software used was custom-made for the study but it also includes some third party software inside. ANTLR-generator handles the data parsing part of the software and it's connected to the custom-made data transformations.

The results of the study stated that manual transformations are almost always exponentially slower than automatic transformations. The usage of time on the transformation solutions varied so that, manual transformations required a constant time while automatic transformation required more time at the beginning of the project. The main takeaway from the results was that, as the number of transformations increase, automatic transformation is always a more cost-efficient choice.

Keywords/tags (subjects)
data transformation, data mapping, parsing, generator

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

1 Johdanto

Tietomuunnoksien automatisoinnilla on tarkoitus ratkaista muunnoksien toteutukseen liittyviä ongelmia. Tekstipohjaisen tiedon kartoittaminen, parsiminen, käsittely ja muuntaminen ovat hitaita ja tarkkuutta vaativia tehtäviä. Tehtävien automatisointi vähentää tai jopa poistaa manuaalisen prosessin virheitä ja tuottaa kustannustehokkaamman lopputuloksen.

Tämän tutkimuksen tarkoituksena oli tuottaa muunnoksien kustannuksiin liittyvää statistiikkaa sekä niistä johdettuja laskelmia ja analyysejä. Laskelmien tarkoitus on olla todisteena tutkimuksessa esitetyn ratkaisun tehokkuudesta ja käytännöllisyydestä. Tutkimuksen perustana on tietomuunnoksien suuri merkitys oikean maailman projekteissa ja niiden toteutukseen liittyvät ongelmat, kuten ajankäyttö ja ylläpito. Pyrkimällä automatisoimaan osan muunnoksien prosessista voidaan säästää eksponentiaalisesti aikaa ja sitä kautta myös rahaa.

Tutkimuksessa aihe on rajattu tekstipohjaisen datan käsittelyyn ja tutkimuksen prosessista on automatisoitu vain muunnososuus. Muut osuudet, kuten tiedon siistiminen ja tulosten analyysi, vaativat manuaalisesti toteutettuja tehtäviä.

2 Datan käsittelyn, kartoituksen ja muunnoksien teoria

2.1 Kartoitukset, muunnokset ja niiden tarve

Datan kartoituksella (data mapping) tarkoitetaan tiedon relaatioiden luomista eri tietomallien välillä (Fatima 2020). Datan kartoitus on erittäin tärkeässä asemassa integraatio-ohjelmistoissa, joissa tuodaan tietoa useasta eri lähteestä. Useimmiten integraatioiden tarkoituksena on koota lähdetieto yhtenäiseen muotoon. Käytännössä datan kartoituksella viitataan yhden tai useamman tietomallin kenttien kartoitusta ja

sovittamista yhtenäiseen lopulliseen malliin. Kartoituksen perusteisiin kuuluu, että on olemassa jokin lähde ja kohdeformaatti. Lähteenä ja kohteena voi toimia tietokanta, datasetti tai terminologia. Terminologia tässä kontekstissa viittaa tapaan klassifioida asioita järjestelmällisesti. Se voisi olla esimerkiksi lääketieteen koodausjärjestelmä, kuten ICD-9. Kartoitukset voivat olla yksi- tai kaksisuuntaisia, mutta joskus tietoa saatetaan yhdistellä useista kentistä yksittäisiin kenttiin, joten kaksisuuntaiset kartoitukset eivät välttämättä ole mahdollisia. (McBride, Gilder, Davis & Fenton 2006.) Tästä esimerkkinä voisi olla mikä tahansa julkinen rajapinta. Julkiset rajapinnat hakevat tietoa useista eri lähteistä, käsittelevät tiedon ja tarjoavat käyttäjille yksinkertaistetun rajapinnan kysellä tietoa.

Kartoittamisen ja muunnosten vaikutuksia oikeassa maailmassa voidaan tarkastella yritysten näkökulmasta. Yrityksien haasteena on hyödyntää eri lähteistä saatua dataa. Yksi tärkeimmistä askelista tässä prosessissa on datan kartoitus. Data täytyy yhtenäistää ja muuntaa muotoon, joka soveltuu liiketoimintaa hyödyntäviin prosesseihin. (Fatima 2020.) Prosessit voivat olla osa kokonaisuutta, joka määrittelee liiketoiminnan onnistuneisuutta. Prosessi voi myös olla osa tuotetta tai palvelua, esimerkkinä aikaisemmin mainittu julkinen rajapinta.

Datan kartoitus on suuressa asemassa terveys- ja sosiaalihuollon sektoreilla, varsinkin nykypäivänä, kun potilastietoja siirretään paperilta sähköiseen muotoon. Yksi tärkeimmistä terveys- ja sosiaalihuollon järjestelmien yhteistoimivuuden ominaisuuksista on datan kartoitus. Järjestelmissä on usein käytetty eri nimityksiä samoille asioille, jolloin voi olla elintärkeää luoda kartoituksia termien välillä. Kartoitusten on oltava säännönmukaisia ja uudelleen rakennettavissa. Ulkopuolisen osapuolen täytyy voida uudelleen luoda kartoitus aikaisempien ohjeiden perusteella. Kartoituksia on myös ylläpidettävä säännöllisesti ja niiden laatu täytyy varmistaa ennen käyttöönottoa. (McBride ym. 2006.)

Datan hyötykäsittelyssä kartoitus on vain yksi osuus. Se voi olla yksi tärkeimmistä ja laatukriittisistä osuuksista, mutta muut osuudet tekevät myös olennaisia asioita. Datan muunnokset ovat yleensä yhdistetty kartoitukseen, koska muuntaminen itsessään tekee käsittelyssä suuren osan työstä. Muuntaminen käytännössä tarkoittaa yhden formaatin muuntamista toiseksi (Tozzi 2018). Kuten aikaisemmin tässä kappaleessa osoitettiin, yrityksen näkökannalta voi olla monia syitä miksi dataa käsitellään ja siivotaan helpommin havainnollistettavaan muotoon. Datan muuntamisessa voi olla kyse yksinkertaisesta prosessista, kuten koodauksen muuntamisesta. Esimerkiksi ASCII-koodattu teksti voidaan muuntaa UTF-8-koodatuksi. Muuntamisessa voi olla kyse myös monimutkaisesta ja moniaskeleisesta prosessista, kuten tietokannan migraatiosta toiselle alustalle.

2.2 Tiedostomuoto

Tiedostomuoto määrittelee tallennetun datan rakenteen ja tyypin. Tyypillinen rakenne tiedostolle voi sisältää otsakkeen (header), metatietoja (metadata), varsinaisen sisällön ja tiedostopäätteen eli EOF-merkin (End Of File). Tiedostomuoto myös määrittelee, onko sisältö selväkielistä tekstiä vai binäärimuodossa. Jotkin tiedostomuodot voivat olla avoimia tai yksityisomistuksellisia. (File Format Definition 2011.) Avoimille muodoille on yleensä hyvä tuki kaikilla alustoilla, mutta yksityisomistuksellisille tiedostomuodoille usein tarvitsee niille tarkoitetun ohjelmiston.

Jokainen merkki tietokoneen muistissa on tallennettu tavuina, joita koodataan erilaisilla tavuesityksillä (character encoding) (Ishida 2015). Nykyään vakiona käytetään usein UTF-8-tavuesitystä, koska se on määritelty tukemaan suurinta osaa maailman kirjoitusjärjestelmistä (Yergeneau 2003). Kirjoitettaessa tekstiä tietokoneella täytyy valita jokin monista koodauksista. Ilman koodausta tietoa ei voisi tulkita oikein, kun sitä yritetään lukea myöhemmin. Välillä voi huomata, että tekstitiedoston merkeissä on selvästi outoja tuntemattomia merkkejä. Tämä on yleensä merkki siitä, että koodaus on ollut erilainen tiedoston luonnin aikana kuin lukemisen aikana.

Tiedostomuotoja on monenlaisia, jopa ääretön määrä, kun lasketaan räätälöidyt formaatit. Työssä rajattiin aloitus- ja lopetusformaatin selväkielisiin tekstipohjaisiin formaatteihin, koska tutkimusongelma sai alkunsa tekstipohjaisista formaateista. Alkuperäisissä dataseteissä oli koodauksena Windows-1252, mutta työssä päätettiin käyttää UTF-8-koodausta tiedostojen datan standardisoimisen takia. Työssä asetettiin myös UTF-8-koodaus kaikille sisääntulo- ja ulostulo tiedostomuodoille.

Kuten aikaisemmassa kappaleessa mainittiin, tutkielmassa käytetty data rajattiin tekstipohjaisiin tiedostomuotoihin. Käytetyt tiedot voisi klassifioida kahdella yleisellä tavalla. Data voi olla ensisijaista tai toissijaista. Ensisijainen data on itseluotua tai generoitua. Toissijainen data on muilta saatua dataa. (Data Types & File Formats n.d.) Datasetit, jotka haettiin avoindata.fi-sivulta, ovat toissijaista tietoa, eli tiedon on tuottanut jokin muu taho. Päinvastoin itse tuottamani datan tulokset ovat ensisijaisia, joita käytetään tämän tutkimuksen toteutuksessa. Tieto voi olla kvalitatiivista, jolla viitataan tekstiin, kuviin, videoihin, ääneen ja havaintoihin, tai se voi myös olla kvantitatiivista eli numeerista dataa (Data Types & File Formats n.d.). Valitussa datassa on kvalitatiivista havaintodataa liittyen koulutuksien kävijöihin ja arvioihin. Toinen datasetti on kyselydataa, joka voitaisiin määritellä havaintoihin perustuvaksi dataksi eli kvalitatiiviseksi.

2.3 Tiedonkäsittely ja parsiminen

Parsiminen on lauserakenteen analyysin prosessi, jota pääosin käytetään luonnollisien kielten käsittelyyn. Parsimista voisi ytimekkäästi kuvailla syötteen (input) pilkkomiseksi yksittäisiin lauseenjäseniin, jotta voidaan havainnoida jäsenten kieliopillista tyyppiä sekä niiden syntaktisia relaatioita. Syöte tässä tapauksessa voisi olla kirjoitettua tai puhuttua sisältöä. (Rangra & Madhusudan 2015.) Lauseenjäsenten relaatiot tarkoittavat parsimisessa käytetyn kieliopin mukaisia sääntöjä, niiden vaikutusta lauseenjäseniin ja virkkeiden muodostukseen. Esimerkiksi suomen kielen parsimisessa voitaisiin aloittaa merkkitasolla, josta päästään sanatasolle. Sanoja voitaisiin lajitella

verbeiksi, subjekteiksi ja niin edelleen. Sen jälkeen voitaisiin lajitella lauseenjäsenten yhdistelmiä virkkeiksi, kappaleiksi ja luvuiksi. Lopulta koko tekstiä voitaisiin kuvailla yksittäisten kokonaisuuksien ja päätteiden (terminal) relaatioiden avulla.

Jokainen kieli sisältää oman puheosuutensa (part-of-speech), jonka mukaan päätteitä voidaan luokitella. Joitakin sanoja voidaan kuitenkin tulkita monella eri tavalla. Tästä esimerkkinä on englannin kielen sana ”book”. Sanan merkitys ja relaatio vieressä oleviin sanoihin on se, mitä parsimisella pyritään tulkitsemaan. Tulkintojen avulla voidaan luoda puurakenne, jossa relaatiot ovat kuvattuna. (Ranga ym. 2015.)

Luonnollisen kielen parsimiseen on olemassa monia keinoja. Kaksi yleisintä lähestymistapaa ovat ylhäältä-alas ja alhaalta-ylös parsinta. Tässä tutkimuksessa keskityttiin ylhäältä-alaspäin parsintaan, koska ANTLR-generaattori käyttää strategiaa, joka perustuu tähän. Ylhäältä-alas parsinnassa aloitetaan parsinta juurisolmusta, josta liikutaan alas vasemmalta oikealle viimeiseen lehtisolmuun saakka. Tekniikan etu on siinä, että se ei koskaan tutki puita, jonka lopputuloksena ei ole edellisen solmun tyyppi. Tästä syntyykin tekniikan suurin heikkous, eli tekniikka voi luoda puita lukematta sisältöä, minkä seurauksena joudutaan ottamaan askelia taaksepäin ja laskemaan uusia potentiaalisia tuloksia. (Ranga ym. 2015.)

Tutkimuksessa käytettiin ANTLR-kirjastoa, joka hyödyntää ALL(*) (all star) -parsintastrategiaa. Strategia eroaa muista ylhäältä-alaspäin strategioista pyrkimällä eroon niissä havaituista heikkouksista. ALL(*)-strategian toiminta perustuu deterministisen äärellisen automaatin (DFA, Deterministic Finite Automaton) periaatteisiin. Tämä tarkoittaa, että annettua sisältöä pyritään vertaamaan määriteltyihin sääntöihin, ja kun sopiva tila (state) saavutetaan, valitaan yksi sääntöjen mahdollisuuksista. Jos ennustamisen aikana havainnoidaan, että seuraava merkki on tietyn tyyppinen, määritellään kaikki muut merkkiä käyttämättömät vaihtoehdot ylimääräisiksi. Tästä poikkeuksena ovat rekursiiviset säännöt, jotka ovat usein kontekstivapaita sääntöjä. (Pratt & Fisher 2011.) Kontekstivapaat säännöt viittaavat tapaan kuvata kielen rakennetta si-

sällöstä huolimatta. Luonnollisissa kielissä se viittaa esimerkiksi lauseen rakenteeseen eli syntaksiin. Rekursiivisten sääntöjen käsittelyssä voidaan usein kääntyä vetäytymiseen (backtracking), joka matkii PEG (Parsing Expression Grammar) -tyyppisten parserien toimintaa.

2.4 Parseri-generaattorit

Parseri-generaattori periaatteessa toimii siten, että ensin kirjoitetaan kielioppi, joka määrittelee parsimisen säännöt ja merkistön. Sen jälkeen voidaan automaattisesti generoida lähdekoodia, jolla pystytään parsimaan jono merkkejä. Generoitu parseri yrittää sovittaa sisään tulevaa tietoa kielioppiin ja ajaa tulokset kuuntelijan (listener) läpi. Näihin perusperiaatteisiin perustuu myös ANTLR-generaattori, jonka parsintastrategian toimintaa käytiin läpi edellisessä luvussa.

Parseri-generaattoreilla on omanlaisensa kielioppisyntaksinsa. Tutkimuksessa aiotaan pysyä rajauksen sisällä ja käsitellä vain ANTLR-generaattorin kieliopin syntaksia sekä toimintaa. ANTLR-kieliopin syntaksi voidaan jakaa yksinkertaisiin säännönmukaisiin palasiin. Ensin kirjoitetaan nimi, jonka jälkeen tulee kaksoispiste ja säännön määritys. Yksittäinen sääntö lopetetaan puolipisteellä. Sääntöjen määritykset voivat olla päättäviä (terminal) tai ei-päättäviä (nonterminal). Esimerkki päättävästä määrityksestä voisi olla EOF eli tiedoston päätemerkki. Kaikkien päättävien merkkien nimi on kirjoitettu isoilla kirjaimilla. Päinvastoin ei-päättävät kirjoitetaan pienillä kirjaimilla, ja ne voivat sisältää päättäviä määrityksiä, ei-päättäviä määrityksiä sekä viittauksia itseensä. Tämä yksinkertainen kielioppisyntaksi mahdollistaa rekursiiviset säännöt ja erilaisten hierarkioiden toteuttamisen.

ANTLR-generaattori luo kieliopista kuuntelijan, jonka avulla voidaan ajon aikana tutkia ja käsitellä päättävien ja ei-päättävien sääntöjen löytämiä arvoja. Arvoja voidaan käsitellä miten tahansa, jotta saadaan luotua haluttu lopputulos. Usein kirjastoa käytetään kuitenkin ohjelmointikielten tai rakenteellisen datan käsittelyyn, jossa esiintyy

rekursiivisia rakenteita. Näistä voidaan muodostaa abstrakti syntaksipuu (AST, abstract syntax tree), jonka avulla voidaan helposti havainnollistaa parsittua sisältöä.

2.5 Muunnoksien automatisointi

Tässä tutkimuksessa automatisointi rajoitettiin siten, että itse muunnoksien prosessi on automatisoitu, mutta muut askeleet joudutaan tekemään manuaalisesti. Manuaalisiin askeliin kuuluu lähdedatan generointi aloitusformaattiin ja lopputulosten ajastusten laskutoimitukset. Automatisoitu prosessi sisältää aloitusformaatin muuntamisen ulostuloformaattiin. Koko prosessia voisi kuvailla osittain automatisoiduksi. Ohjelmiston suunnitteluvaiheessa pyrittiin ottamaan huomioon vaihtelevat määrät aloitusdataa ja mahdollistamaan datan muuntamisen välittämättä, muunnetaanko 10 alkiota vai 1000.

3 Tutkimusmenetelmät, metodit ja aineisto

3.1 Perusta

Tutkimusongelmaa kuvastaa lause *Tiedon käsittely, rakenteen kartoitus ja muuntaminen on hidasta*. Tämä voi koskea tiedon ylläpitämistä eli normalisointia ja siistimistä. Se voi myös koskea tiedonkäsittelyä abstrakteina osina tai osien muuntamista eri formaatteihin. Tutkimuksessa on tarkoitus vastata kysymykseen, *kuinka voimme nopeuttaa tietomuunnosten toteuttamista?* ja mitkä tekijät vaikuttavat muunnosten sekä käsittelyn hitauteen. Tutkimus pyrki myös selvittämään muunnosmenetelmien laatua ajalliselta ja kustannusnäkökannalta. Pohjadata tutkimukseen saatiin avoindata.fi-sivustolta. Tutkielman toteutukseen valittiin dataa, joka sisälsi yhden tutkimuksen ja yhden tilastollisen datasetin.

3.2 Tutkimusongelman syntyperä

Tutkimusongelma sai alkunsa terveys- ja sosiaalihuollon projektista, jossa tehtiin useita käännöksiä lähdejärjestelmistä saaduille tiedoille. Käännöksiä tehdessä huomattiin hyvin nopeasti, että manuaalisesti kääntäessä tietoa työstä tulisi todella vaivalloista ja kallista. Tutkimusongelman ratkaisu syntyi ajatellessa tietomuunnosten kustannuksia, muun muassa liittyen ajankäyttöön ja projektin aikatauluun, sekä miten niitä voisi optimoida.

Tämän tutkimuksen prosessi ei voi olla vain yksipuoleinen, joten tarvitaan enemmän ratkaisuja, joista lähestyä ongelmaa. Ensimmäisessä ratkaisussa lähestytään ongelmaa manuaalisella painotuksella. Toinen ratkaisu on päinvastoin eli ensin automaattisesti. Tutkimusongelman muuttaminen tähän muotoon viittaa tutkimuksen sivutavoitteisiin, joiden tarkoitus on selvittää, miten aikaisempi materiaalin läpikäynti vaikuttaa lopullisiin tuloksiin seuraavilla muunnoskerroilla.

Tutkimusongelma sisältää siis kaksi tutkittavaa kokonaisuutta. Tärkeämpi kokonaisuus on itse muunnoksiin käytetty aika ja ajankäytön painotus muunnoksia tehtäessä. Toinen kokonaisuus on tutkia materiaalin kompleksisuuden ja aikaisemman perehtymisen vaikutusta muunnoksien toteuttamisessa.

3.3 Tutkimuksen kuvaus

3.3.1 Yleistä

Tutkimuksessa pyrittiin selvittämään tietomuunnosten kustannustehokuutta tutkien, miten ajankäyttö, automatisointi ja tiedon kompleksisuus vaikuttavat asiaan. Varsinainen tutkimus koostui kahdesta osioista sekä niihin valmistelevista prosesseista.

Ensimmäinen osio tutkimuksesta on jaettu kahteen kokonaisuuteen. Ensimmäisessä ratkaisussa käännetään tieto ensin manuaalisesti, josta saadaan käännöksille pohjalinja. Seuraavaksi käännetään tieto automaattisesti käyttäen ongelmalle räätälöityä ohjelmistoa. Ohjelmisto siivoaa lähdetiedon, parsii sen sekä käsittelee tiedosta halutun ulostuloformaatin muotoista. Manuaalisten ja automaattisten muunnosten aikoja verrataan toisiinsa ja tuloksista pyritään analysoimaan tutkimuksen vastauksia.

Siivoamisessa varmistetaan, että tieto on yhtenäistä, eli sen täytyy noudattaa määritettyä formaattia. Siivottu tieto parsitaan käyttäen parsesi-generaattorin luomaa pohjaa. Parsittu tieto voidaan käsitellä haluttuun ulostuloformaattiin. Käännöksissä otetaan huomioon käännökseen käytetty aika ja niistä saatu statistiikka. Aikojen statistiikasta voidaan esimerkiksi laskea, kuinka kauan kestäisi tehdä n määrä käännöksiä. Toinen osio tutkimuksesta on tulosten analyysi sekä kustannusarvioita n. määristä käännöksiä. Olkoot kustannusarviot oikeasta maailmasta johdettuja esimerkkejä.

3.3.2 Tutkimuksen tarkoitus

Tutkimuksen tarkoitus oli tarjota statistiikkaan perustuvia todisteita siitä, miten tiedon muunnoksien automatisointi voi säästää sitä hyödyntäville rahaa ja aikaa. Tutkimuksesta saatu tärkein tieto on muunnoksiin käytetty aika sekä miten ajan käyttö on painotettu. Aikojen perusteella voidaan tehdä laskelmia, johtopäätöksiä ja esimerkkejä, joita saattaisi tulla eteen oikean maailman projekteissa. Tutkimuksessa pyrittiin myös arvioimaan automatisoinnin kustannuksia. Laskelmat perustuvat automatisointiin käytetyistä henkilötyöpäivistä ja kuvitteellisen työntekijän kuukausipalkasta.

3.3.3 Odotetut tulokset

Tietokoneet ovat yleisesti nopeampia ja tehokkaampia kuin ihmiset tietyissä tehtävissä. Tekstipohjaisen tiedon parsiminen on yksi näistä tehtävistä. Voitaisiin arvioida lopputuloksien olevan painottunut automatisoinnin tehokkuuden puolesta. Tuloksien

pitäisi osoittaa, että jopa kohtuullisen pieni määrä muunnoksia maksaa itsensä takaisin nopeasti. Voidaan myös olettaa, että suurin osa automatisoituihin muunnoksiin käytetystä ajasta tapahtuu projektin alkupäässä, josta ajankäyttö nopeasti putoaa pieneksi ja pysyy pienenä. Tähän verrattuna manuaalisesti tehtyjen muunnoksien olettaisin tarvitsevan lähes konstanttisen ajan. Sen lisäksi manuaalisissa muunnoksissa voi helposti esiintyä virheitä johtuen inhimillisistä syistä.

Ajankäytön erot edellä mainittujen ratkaisuiden välillä voidaan olettaa olevan eksponentiaalisia riippuen muunnosten määrästä sekä alkuperäisen tiedon kompleksisuudesta. Viimeisenä huomiona voidaan olettaa, että tietyt rajatapaukset tai poikkeukset voivat antaa vastakkaisen tuloksen. Tästä esimerkkinä vaikkapa videoiden oikolukeminen, johon lisätään hahmojen ilmeiden ja eleiden kuvauksia. Poikkeuksiin voi kuulua tietoa, joka on hyvin hankalaa muuntaa abstrakteiksi konteksteiksi, jota tietokone pystyy ymmärtämään.

3.4 Menetelmät ja metodit

Tutkimusmenetelmiin sisältyi aineiston etsiminen, käsittely, tulosten raportointi ja analysointi. Aineisto tutkimuksessa on avoindata.fi-sivustolta saadut datasetit ja niitä käsittelemällä saadut tulokset. Tulosten raportointi käsittää muunnoksien aikojen laskutoimitukset ja niiden analysointi käsittää laskelmia kustannustehokkuudesta, ajankäytöstä ja vertailuista aikojen välillä.

3.4.1 Menetelmien analyysi

Datan muuntamisen menetelmien valinnoilla on vaikutusta lopputulokseen, varsinkin projektin aikataulun suhteen. Mitä enemmän räätälöityjä vaihtoehtoja halutaan toteuttaa, sitä enemmän aikaa kuluu projektin alkuvaiheessa. Myös jatkokehitys ja ylläpitokustannukset kasvavat yksityisomistuksellisen ohjelmiston kautta.

Vaihtoehtoisesti voidaan hyödyntää olemassa olevaa ohjelmistoa kuten ANTLR-generaattoria datan parsimiseen. Datan kartoittamiselle on olemassa monia ohjelmistoja ja menetelmiä. Näistä esimerkkinä Clio-projekti, jolla voidaan automatisoida datan kartoitukselle soveltuvia kyselyjä (Fagin, Haas, Hernández, Miller, Popa & Velegrakis 2009). Valmiit ohjelmistot tarjoavat käytettävän alustan, josta lähteä liikkeelle, mutta vaihtokauppana on aina ohjelmiston soveltuvuus, käyttöönoton esteet ja ohjelmiston rajoitteet. Ohjelmiston käyttöönotto voi vaatia perehtymistä ja koulutuksia, eikä se välttämättä tarjoa kaikkea vaadittua toiminnallisuutta. Tämän tutkimuksen puitteissa päätettiin soveltaa olemassa olevaa ohjelmistoa ja sovittaa sitä räätälöityyn muuntaja ratkaisuun. Valinnat johtuivat henkilökohtaisesta mielenkiinnosta ja projektin aikataulun tiukkuudesta.

3.4.2 Valittu menetelmä

Harkittaessa ratkaisua ongelmiin yritettiin ottaa huomioon tärkeimpien tekijöiden ja taustatietojen vaatimuksia sekä rajoituksia. Tiedettiin, että muunnettava tieto oli tekstipohjaisessa formaatissa ja valmiiksi koneluettavissa. Tämän takia piti valita tapa parsia tieto käsiteltävään muotoon. Lopulta valittiin ANTLR4-generaattori, jolla saatiin tehokkaasti generoitua tiedon parsimiselle tarvittavat ohjelmistonosat. Vaihtoehtoisena parsintamenetelmänä olisi ollut rakentaa oma räätälöity parseri. Oli kuitenkin käytännöllistä hyödyntää mahdollisimman paljon olemassa olevia ohjelmistoja ja työkaluja, jotta työ tulisi valmiiksi tehokkaammin ja aikataulussa pysyen. Oman parserin ylläpito ja kehitys vaatii huomattavia resursseja, varsinkin käsitellessä monimutkaisempaa tietoa.

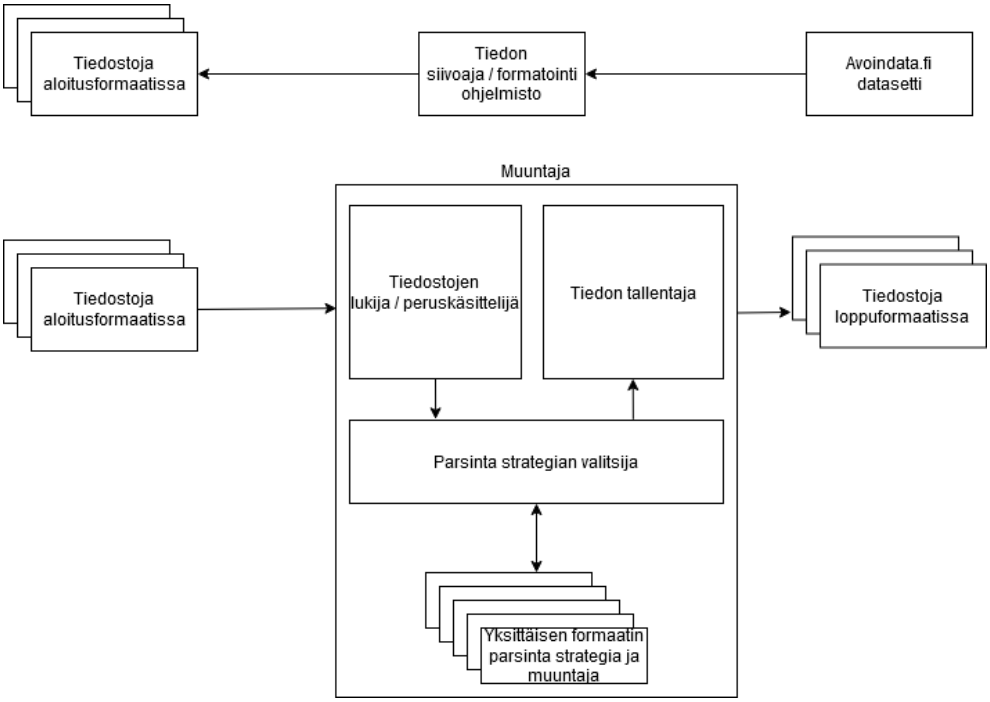
ANTLR4 on tehty Java-ohjelmointikielellä, mutta sille on olemassa sitoumuksia (bindings) myös muille kielille. Java-ohjelmointikieli oli ennestään tuttu, joten räätälöity ohjelmisto toteutettiin Javalla. Parsimisen jälkeen tieto on helposti käsiteltävissä ja siitä voidaan muodostaa ulostuloformaatin rakenteen muotoista tieto. Muunnoksien suorituksen aikana lisättiin tarvittavat statistiikat jokaiseen tulokseen, josta jälkikäteen voitiin koota erilaisia taulukoita ja laskelmia.

3.4.3 Ohjelmiston kuvaus

Tutkimuksessa haluttiin mallintaa reaalimaailman tilannetta, jossa alkuperäinen tieto on kompleksisessa formaatissa. Tiedon tulisi olla muodossa, joka on muunnoksen tarpeessa, että sitä voidaan käsitellä tehokkaasti. Haluttiin myös käyttää tietoa, joka ei ollut generoitu satunnaisesti, vaan kerätty ihmisiltä erilaisten kyselyiden tai tutkimusten avulla. Tutkimuksessa päädyttiin hyödyntämään dataa sivulta avoindata.fi. Alkuperäinen tieto piti siistiä, eli tyhjät ja puuttuvat arvot piti korvata kelvollisilla arvoilla, sekä ylimääräiset kentät piti jättää pois. Tämän tehtävän sain tehtyä pienellä avustusohjelmistolla, jonka tarkoitus oli toteuttaa tiedon siistiminen ja aloitusformaateista johdetuiden mallien täyttäminen tiedolla. Lisäksi varmistin, että kaikki käytössä olevat tiedot olivat oikein koodattu UTF-8:n mukaan ja linjapäätteet olivat UNIX-tyylisiä. Tämä teki parserin kieliopin tuottamisesta suoraviivaista ja järjestelmällistä.

Seuraavaksi toteutettiin pääohjelmisto, joka lukee aloitusformaatissa olevat tiedot sisään, parsii tiedon sekä muuntaa sen haluttuun loppuformaattiin. Tiedon parsiminen ja muuntaminen tapahtuu tehdasmallia (Factory pattern) hyödyntäen. Tehdasmallin käyttö teki muunnosten lisäämisestä ohjelmistoon helppoa ja tehokasta. Halutessa voitaisiin lisätä enemmän muunnoksia ohjelmistoon, eikä edes tarvitsisi muuttaa muuta ohjelmistoa millään tavalla. Käytännön prosessin toimintaa kuvastaa kuvio 1.

Kuvio 1. Muunnoksen prosessi



3.4.4 Muunnosprosessin yleiskuvaus

Muunnosprosessi on ohjelmistossa yksittäinen kokonaisuus, joka yhdistyy ohjelmistoon yleisen rajapinnan (interface) kautta. Rajapinnan tehtävä on hankkia muunnoksen lopputulos ilman että se ottaa mitään kantaan sen sisältöön. Lopputulokset kootaan yhteen (aggregate) ja tallennetaan yhteen ulostulotiedostoon. Ratkaisujen muunnosten tarkemman kuvauksen avataan tarkemmin luvussa 4.

4 Toteutetut muunnokset

4.1.1 Ensimmäinen ratkaisu: Manuaalinen painotus

Ensimmäisen ratkaisun painotuksena ovat manuaaliset muunnokset. Muunnosten ajankäytölle haluttiin luoda pohjalinjan, johon jälkikäteen vertaan automaattisesti toteutettujen muunnoksien tuloksia. Pohjalinjan manuaalisille muunnoksille luo viisi käännöstä, joista otetaan keskiarvo ja ääriarvojen erot keskiarvoon nähden. Näiden arvojen perusteella voidaan arvioida kuinka kauan kestäisi, että manuaalisiin muunnoksiin käytetty aika ylittää automaattiset muunnokset.

Automaattisiin muunnoksiin käytetty aika jakautuu kahtia, siten että on perusaika ja muunnoksille käytetty aika. Perusaika koostuu ohjelmiston luurangon toteuttamisesta. Luuranko sisältää sisään tulevien tiedostojen lukemisen, valmistelun ja valmiiden muunnosten tallentamisen. Molemmat ratkaisut käyttävät samaa perusaikaa koska ohjelmiston luuranko on jaettu. Automaattisiin muunnoksiin käytetty aika koostuu perusajasta, sekä muunnosten toteutus ajasta. Muunnoksien toteutus kattaa ANTLR-generaattorin kieliopin tekemisen ja parsitun tiedon muuntamisen lopulliseen ulostulo muotoon.

4.1.2 Tietomuodot

Ensimmäiselle muunnokselle käytettiin datasettiä verkkosivulta avoindata.fi. Datasetissä oli tietoa Helsingin kaupungin henkilöstön koulutuksista vuodesta 2014 alkaen. Datasettiä päivitetään edelleen tänä päivänä. CSV-formaatissa oleva datasetti ladattiin 5. päivä helmikuuta 2020, joten käytetty data on siihen päivään saakka ajan tasalla. Lähdedatassa on kuvattu koulutustapahtumien nimet, ajankohdat, palautteiden arvostelut sekä osallistujatietoja. Datasetti valittiin, koska se sisälsi tekstipohjaista- ja numeraalista tietoa.

Aloitusformaatiksi luotiin lähes yksi-yhteen sopiva rakenne verkkosivulta saaduista otsikoista. Rakenteessa muutettiin kenttien järjestystä sekä sijaintia ja lisättiin tarvittavat välimerkit aloitusformaattiin. Aloitusformaatin oli tarkoitus muistuttaa nopeasti tehtyä muistiinpanoa, jonka esimerkiksi sihteeri olisi voinut luoda. Formaatti on kuitenkin luotu siten, että se ei ole tehokkaasti suunnitellussa rakenteessa. Datasetti saatiin muunnettua aloitusformaattiin käyttämällä kappaleessa 3.4.3 mainittua tiedon siivoamiselle ja valmistelemiselle tarkoitettua apuohjelmistoa.

Aloitusformaatin tallennusformaatti on .txt ja sen sisältöä ja rakennetta voidaan nähdä kuviossa 2. Tiedostot ovat koodattu UTF-8 merkistöä käyttäen ja linjapäätteinä on UNIX-tyyliset päätteet. Koodaukseksi valittiin UTF-8 merkistö, koska se tukee kaikki vaadittuja merkkejä. Ohjelmisto kehitettiin pääosin Linux-pohjaisella käyttöjärjestelmällä, joten UNIX-tyyliset linjapäätteet olivat luonnollinen valinta. Aloitusformaatin syntaksi koostuu kentistä. Kenttä alkaa nimellä, jonka jälkeen tulee kaksoispiste. Kaksoispisteiden jälkeen tulee datasetistä saadut arvot, joita on kuvattu formaatilla $tunniste. Tunnisteet vaihdettiin lähdedatasta saaduilla arvoilla, jonka jälkeen tiedosto voitiin tallentaa käytettäväksi. Aloitusformaatille tehty malli on kuviossa 3.

Kuvio 2. Ensimmäisen aloitusformaatin malli

```
# Koulutustapahtuma #            PVM: $Alku - $Loppu
Nimi: $TapahtumanNimi
ID: $TapahtumaID
Kurssiryhmä: $Kurssiryhma

=======

Palautteiden keskiarvo: $Palauteka
Opetuspäivien lukumäärä: $Opetuspaivienlkm
Koulutuspäivät yhteensä: $Koulutuspaivat

=======

Koulutukseen osallistuneet:

 - Miehet: $OsallM
 - Naiset: $OsallN
 - Muut: $OsallO
 - Yhteensä: $Osallistunut
```

Kuvio 3. Ensimmäisen aloitusformaatin esimerkki

```
# Koulutustapahtuma #            PVM: 04.11.2014 - 04.11.2014
Nimi: Sote: Lync - perusteet ip / Henke, Taltu, Tievi
ID: 21479
Kurssiryhmä: Tietotekniikkakoulutus

=======

Palautteiden keskiarvo: 4,22
Opetuspäivien lukumäärä: 0,5
Koulutuspäivät yhteensä: 5,5

=======

Koulutukseen osallistuneet:

 - Miehet: 0
 - Naiset: 11
 - Muut: 0
 - Yhteensä: 0
```

Ulostuloformaatiksi valittiin JSON-formaatti, koska se on erinomaisesti tuettu ja tehokas tapa esittää ihmis- ja koneluettavaa tietoa. Tässäkin tapauksessa valittiin koodaukseksi UTF-8-merkistö ja linjapäätteiksi UNIX-tyyliset päätteet. Ulostuloformaatti ei muistuttanut sisääntuloformaattia vaan pyrki standardisoimaan tavan esittää henkilökoulutusten tietoja. Sisään tulevasta tiedosta otettiin ainoastaan arvot, tiedoston nimi, muunnoksen päivämäärä ja siihen kulunut aika. Aloitusformaatin tiedot ovat koottu yhteen lopputuloksia sisältävään tiedostoon, lopullisen tiedostokoon pienentämiseksi. Ulostuloformaatin sisältöä kuvaa neljäs kuvio.

Kuvio 4. Ensimmäinen ulostuloformaatti


```
{
    "file": "test-4510bbac-bcfc-4122-b168-dfd094900717",
    "transform_date": "17.2.2020",
    "transform_time": "00:02:01.14",
    "data": {
        "begin_date": "01.01.2014",
        "end_date": "30.04.2014",
        "course_name": "Sote: Vuoden 2013 koulutusten hännät, Perhe- ja lähisuhdeväkivalta",
        "course_id": "19918",
        "course_group": "Sote-täydennyskoulutus",
        "feedback_mean": 0,
        "teaching_days_total": 0,
        "training_days_total": 0,
        "participants": {
            "men": 0,
            "women": 0,
            "others": 0,
            "total": 0
        }
    }
},
```


4.1.3 Prosessi ja ratkaisu

Muunnosprosessi ohjelmistossa toteutettiin siten, että itse muunnoksien toteutus voi olla tehty miten tahansa. Ohjelmisto ei ota kantaa siihen mitä muunnos tekee prosessin aikana, mutta lopputuloksen on täytettävä muunnokselle tarkoitetun rajapinnan tarpeet. Muunnoksen lopputuloksen täytyy palauttaa sisältönsä Javan merkkijono (String) tyyppisenä, sekä tiedostopolun, joka osoittaa tallennettavaan tiedostoon. Tiedoston nimet muodostettiin satunnaisesti, ettei päällekkäisyyksiä tapahtuisi.

Ensimmäisen ratkaisun muunnoksen prosessi koostui useasta osuudesta. Ensimmäisenä täytyi jakaa aloitusformaatti semanttisiin kokonaisuuksiin. Kokonaisuuksien täytyi olla tarpeeksi abstrakteja, jotta lekseri ja parseri pystyivät käsittelemään ne. Koska rakennetta kuvaava tieto ei ollut kontekstivapaata, päätettiin heti alussa luoda ANTLR-kieliopin, joissa oli kontekstia sisältäviä sääntöjä. Sääntöjen luonnin jälkeen voitiin automaattisesti generoida kieliopin toteuttava lekseri ja parseri, joita käytettiin muunnoksen kolmannessa osuudessa. Toisen osuuden tehtävä oli jakaa sisään

tuleva tieto aikaisemmin mainittuihin semanttisiin kokonaisuuksiin. Tiedon parsinta päätettiin tehdä osissa, koska kaiken sisällön käsitteleminen samaan aikaan tuotti välillä virheitä kieliopin määritteiden tunnistamisen kanssa. Muunnoksen kolmas osuus käsitteli tiedon kokonaisuudet yksi kerrallaan ja rakensi niistä ulostuloformaatin sisällön muistuttavan arvoluokan osioita. Osiot yhdistettiin lopulliseksi tulokseksi muunnoksien viimeisessä osuudessa.

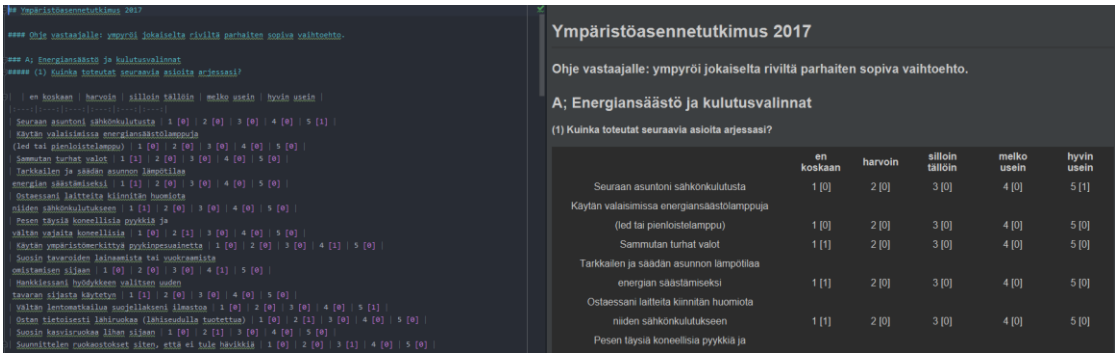
4.1.4 Toinen ratkaisu: Automatiikan painotus

Toisessa ratkaisussa oli tarkoitus muuntaa monimutkaisempaa tietoa, jotta manuaalinen muunnos olisi työläämpi tehdä. Aikaisemmissa olettamissa arvioitiin, että manuaalisen muunnoksen ollessa työläämpi, tulisi myös muunnos tapojen tehokkuudessa suurempia eroja. Toinen ratkaisu rakennettiin ensimmäisessä ratkaisussa toteutetun ohjelmiston päälle, hyödyntäen olemassa olevia metodeja sekä rajapintoja.

4.1.5 Tietomuodot

Toisen ratkaisun lähtöformaatin malli syntyi kääntämällä alkuperäinen .pdf formaatissa oleva kyselylomake .md (markdown) formaattiin. Markdown formaatti valittiin, koska se on suosittu tekstipohjainen formaatti ohjelmistoyhteisöissä. Markdown formaatti on myös riittävän runsassanainen (verbose), jotta sen sisällöstä sai helposti luotua rakenteellisesti monimutkaisempaa. Aloitusformaatin datasetti valittiin avoindata.fi sivustolta. Datasetti sisältää Helsingin ja Vantaan ympäristöasennekyselyn vastauksia vuodelta 2017. Kysely pyrki selvittämään ihmisten ymmärrystä, huolia ja asenteita liittyen ympäristöön ja sen suojeluun. Aloitusformaatin rakennetta kuvastaa viides kuvio.

Kuvio 5. Toinen aloitusformaatti. Tekstipohjainen esitys vasemmalla, tyylitetty esitys oikealla



Ympäristöasennetutkimus 2017

Ohje vastaajalle: ympyröi jokaiselta riviltä parhaiten sopiva vaihtoehto.

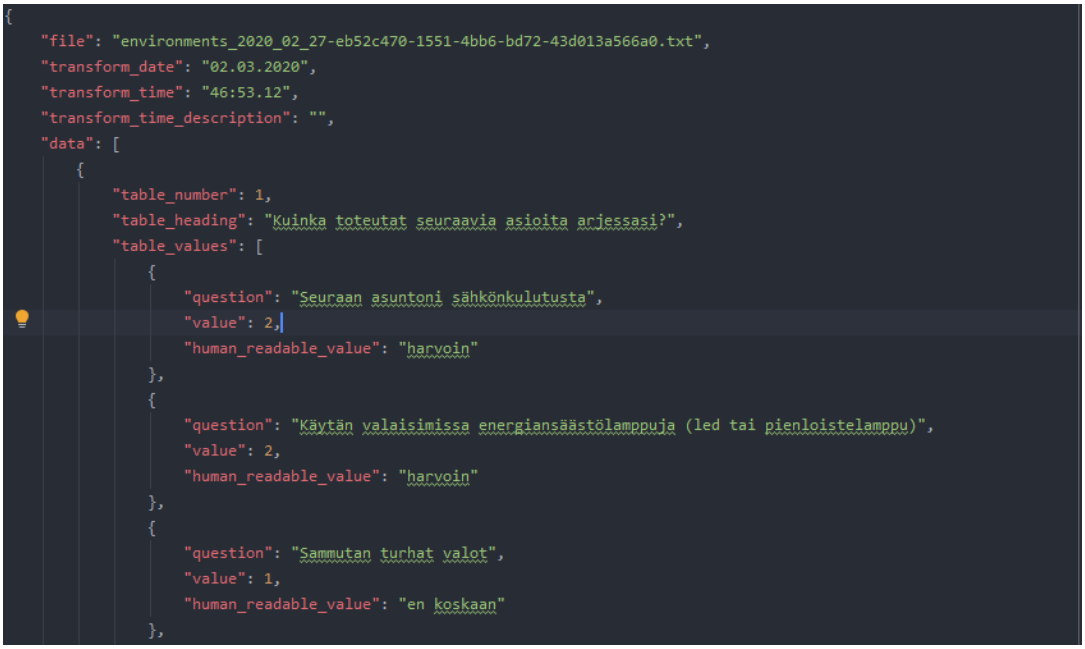
A; Energiansäästö ja kulutusvalinnat

(1) Kuinka toteutat seuraavia asioita arjessasi?

	en koskaan	harvoin	silloin tällöin	melko usein	hyvin usein
Seuraan asuntoni sähkönkulutusta	1 [0]	2 [0]	3 [0]	4 [0]	5 [1]
Käytän valaisimissa energiansäästölamppuja (led tai pienloistelamppu)	1 [0]	2 [0]	3 [0]	4 [0]	5 [0]
Sammutan turhat valot	1 [1]	2 [0]	3 [0]	4 [0]	5 [0]
Tarkkailen ja säädän asunnon lämpötilaa energian säästämiseksi	1 [1]	2 [0]	3 [0]	4 [0]	5 [0]
Ostaessani laitteita kiinnitän huomiota niiden sähkönkulutukseen	1 [1]	2 [0]	3 [0]	4 [0]	5 [0]
Pesen täysiä koneellisia pyykkiä ja vältän vajaita koneellisia	1 [0]	2 [1]	3 [0]	4 [0]	5 [0]
Käytän ympäristömerkittyä pyykinpesuainetta	1 [0]	2 [0]	3 [0]	4 [1]	5 [0]
Suosin tavaroiden lainaamista tai vuokraamista omistamisen sijaan	1 [0]	2 [0]	3 [0]	4 [1]	5 [0]
Hankkiessani hyödykkeen valitsen uuden tavaran sijasta käytetyn	1 [1]	2 [0]	3 [0]	4 [0]	5 [0]
Valitsen lentomatkailua suojellakseni ilmastoa	1 [0]	2 [0]	3 [0]	4 [0]	5 [1]
Ostan tietoisesti lähiruokaa (lähiseudulla tuotettua)	1 [0]	2 [1]	3 [0]	4 [0]	5 [0]
Suosin kasvisruokaa lihan sijaan	1 [0]	2 [1]	3 [0]	4 [0]	5 [0]
Suunnittelen ruokaostokset siten, että ei tule hävikkiä	1 [0]	2 [0]	3 [1]	4 [0]	5 [0]

Ulostuloformaatti suunniteltiin mahdollisimman abstraktiksi. Formaatti pyrittiin luomaan jättämällä alkuperäinen rakenne huomioimatta ja tunnistamaan datasta vain olennaiset osuudet. Näihin kuului muun muassa osioiden otsikot, kysymykset ja kyselyihin vastanneiden valinnat. Valintoja kuvasi numeraalinen arvo suoraan lähdedatasta ja selväkielinen arvo, joka sopisi paremmin ihmiselle luettavaksi. Poikkeuksena sääntöön oli kyselyihin vastanneiden iän ilmoitus. Ulostuloformaatti käännettiin ihmisluettavan iän numerosta ja määreestä. Esimerkki iän ihmisluettavasta arvosta on *65 vuotta*. Ulostuloformaatin rakenteen voi nähdä kuviossa 6.

Kuvio 6. Toinen ulostuloformaatti



```
{
  "file": "environments_2020_02_27-eb52c470-1551-4bb6-bd72-43d013a566a0.txt",
  "transform_date": "02.03.2020",
  "transform_time": "46:53.12",
  "transform_time_description": "",
  "data": [
    {
      "table_number": 1,
      "table_heading": "Kuinka toteutat seuraavia asioita arjessasi?",
      "table_values": [
        {
          "question": "Seuraan asuntoni sähkönkulutusta",
          "value": 2,
          "human_readable_value": "harvoin"
        },
        {
          "question": "Käytän valaisimissa energiansäästölamppuja (led tai pienloistelamppu)",
          "value": 2,
          "human_readable_value": "harvoin"
        },
        {
          "question": "Sammutan turhat valot",
          "value": 1,
          "human_readable_value": "en koskaan"
        },
```


Aloitus- ja ulostuloformaatti molemmat noudattivat aikaisempaa linjaa koodauksen ja linjapäätteiden suhteen. Ulostuloformaatin alkuosa (kuvio 6) on vain pieni osa lopputulosta. Rakenteesta voidaan kuitenkin nähdä, että jokainen sisään tuleva tiedosto jaettiin yksittäisiin tauluihin, jotka jaettiin yksittäisiin kysymyksiin ja vastauksiin.

4.1.6 Ratkaisu ja prosessi

Toinen ratkaisu oli toteutettu samoja periaatteita käyttäen kuin ensimmäinen. Ensin luettiin sisään aloitusdata. Data jaettiin yksittäisiin tuloksiin eli alkioihin. Yksi alkio oli tässä tapauksessa yhden kyselyyn vastanneen valinnat. Alkio jaettiin kahteen osaan, joista ensimmäisenä oli taulukko osuus ja toisena ”vapaat” kysymykset. Vapaat kysymykset olivat rakenteeltaan erilaisia, mutta taulukoiden mukaan myös monivalintakysymyksiä. Sisään tuleva data päätettiin jakaa kahtia käsittelyssä koska ANTLR-kieliopin toteuttaminen oli yksinkertaisempaa osakokonaisuuksille. Jakamisen jälkeen

alkion osuudet parsittiin ja muunnettiin ulostuloformaatin osiin. Ulostuloformaatin osat yhdistettiin yhdeksi kokonaisuudeksi viimeisessä askeleessa. Tämä yhdistetty kokonaisuus tallennettiin tuloksille tarkoitettuun tiedostoon. Ratkaisu otti huomioon yksittäisiin käännöksiin käytetyn ajan, jotta voitiin luoda tilastoja ja analyysejä tutkimusta varten.

5 Tulokset

5.1 Ensimmäisen ratkaisun tulokset

Ensimmäisen ratkaisun manuaalisille muunnoksille saadut ajat noudattivat aikaisempia olettamia. Tein viisi manuaalista muunnosta, joihin kului keskimäärin 2 minuuttia, 12 sekuntia ja 872,8 millisekuntia per muunnos. Suurimmat erot johtuivat siitä, miten käsittelin tallennettavan rakenteen. Nopeimman muunnoksen sain tehtyä, kun kopioin valmiiksi tehdyn rakenteen tyhjillä kentillä ja täytin tiedot yksitellen. Nopein muunnos viiden käännöksen joukosta oli 1 minuuttia, 51 sekuntia ja 17 millisekuntia. Aika oli 21 sekuntia ja 855,8 millisekuntia nopeampi kuin keskimääräinen aika. Hitain muunnos joukosta on 2 minuuttia, 45 sekuntia ja 70 millisekuntia. Hitaimmassa muunnoksessa kesti enemmän aikaa kuin keskimäärin, koska tein sen ensimmäisenä ja jouduin luomaan myös lopullisen rakenteen muunnoksen aikana. Hitain muunnos on 32 sekuntia ja 197,2 millisekuntia hitaampi kuin keskimääräinen muunnos. Kuviosta X voidaan todeta, että muunnoksien ajat kohtaavat noin 380 muunnoksen kohdalla. Sininen viiva, joka viittaa automaattisiin muunnoksiin kasvaa todella vähän y-akselilla riippumatta muunnoksien lukumäärästä. Muunnoksien lukumäärän kasvaessa aikaero myös kasvaa. Taulukoissa 1 ja 2 on esitetty ensimmäisen ratkaisun tuloksia.

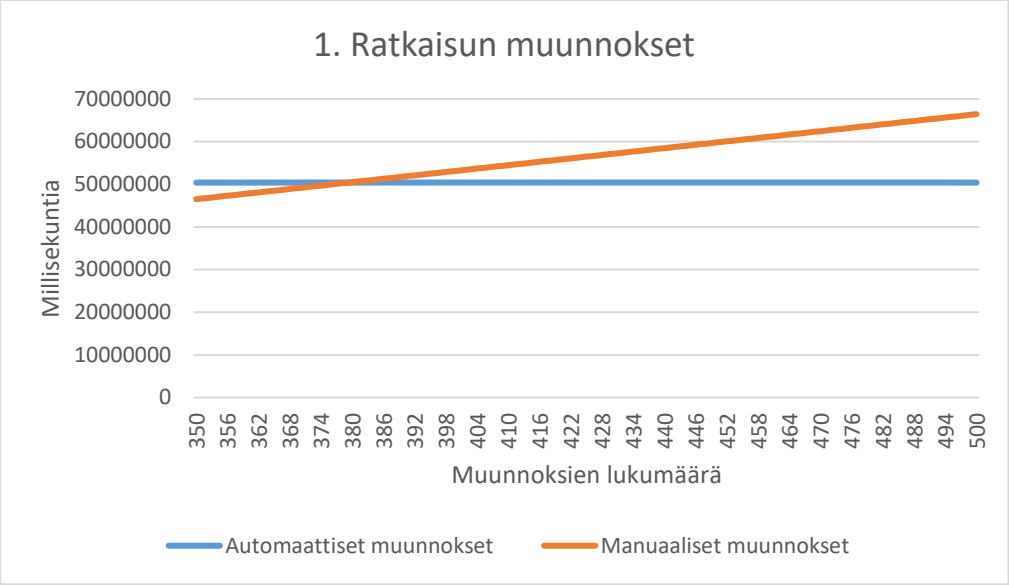
Taulukko 1. Manuaaliset muunnokset

Määrite	**Aika**	**Selite**
Hitain aika	2:45.70	minuuttia:sekuntia.millisekuntia
Nopein aika	1:51.17	minuuttia:sekuntia.millisekuntia
Keskimääräinen aika	2:12.872,8	minuuttia:sekuntia.millisekuntia

Taulukko 2. Automaattiset muunnokset

Määrite	**Aika**	**Selite**
Hitain aika	873	millisekuntia
Nopein aika	< 0	millisekuntia
Keskimääräinen aika	4,835	millisekuntia

Kuvio 7. Ensimmäisen ratkaisun muunnokset



Automaattisten muunnosten tulokset noudattivat aikaisempia olettamia samalla tavalla kuin manuaaliset muunnokset. Muunnokset olivat keskimäärin nopeita, mutta tuhannen muunnoksen joukosta löytyi suuria vaihteluita. Osaan automaattisista muunnoksista kului nolla (0) millisekuntia eli niin vähän, ettei ohjelmisto ei edes huomioinut aikaa tallennuksessa. Vastakohtana olivat muunnokset, joihin kului yksinään lähes sekunti. Tuhanteen automaattiseen muunnokseen kului yhteensä 4 sekuntia ja 831 millisekuntia. Keskimääräinen aika yhdelle muunnokselle oli 4,836 millisekuntia. Nopeimmat muutokset olivat nolla millisekuntia, joten pienimmän ja keskiarvon välillä oleva aika oli itse keskiarvo. Hitain muunnos kesti näiden muunnoksien aikana 873 millisekuntia, joka oli 868,164 millisekuntia keskimääräistä hitaampi.

5.2 Toisen ratkaisun tulokset

Toisen ratkaisun manuaalinen muunnos oli ensimmäisestä poikkeava. Poikkeavuus syntyi muunnokselle käytetyn formaatin kompleksisuudesta ja koosta. Formaatti sisältää kymmeniä yksittäisiä datapisteitä (data point). Datapisteiden muuntaminen manuaalisesti on hidasta ja virhealtista toimintaa. Muunnoksia tehdessäni totesin oman kykyni keskittyä olevan tärkein asia onnistumiselle. Virheitä saattoi tulla, kun katsoi väärää riviä tai kolumnia. Sen lisäksi kirjoitusvirheet ja JSON-rakenne virheet olivat yleisiä. Toisen muunnoksen hitaus johtui myös datasetin vastausten koosta. Minulla kului ensimmäisen muunnoksen tekemiseen 46 minuuttia, 53 sekuntia ja 12 millisekuntia. Tämä aika ei ole yhtä tarkka kuin koneellisten muunnosten, johtuen minun omasta hitaudestani aloittaa ja pysäyttää ajanottoa. Päätin olla tekemättä viittä muunnosta manuaalisesti, koska muunnoksiin käytetty aika oli niin suuri, että suurinkaan vaihtelu ei vaikuttaisi lopputuloksiin merkittävästi. Käytin yhden muunnoksen aikaa referenssinä siitä, kuinka kauan kestäisi keskimäärin toteuttaa yksi manuaalinen muunnos. Kuviosta 8 voidaan todeta, että manuaalisia muunnoksia voisi toteuttaa noin 21 ennen kuin automaattiset muunnokset olisi toteutettu. Automaattisten muunnosten toteutuksen jälkeen aikaerot kasvavat eksponentiaalisesti jokaisen muunnoksen kanssa. Taulukoissa 3 ja 4 on esitetty toisen ratkaisun tuloksia.

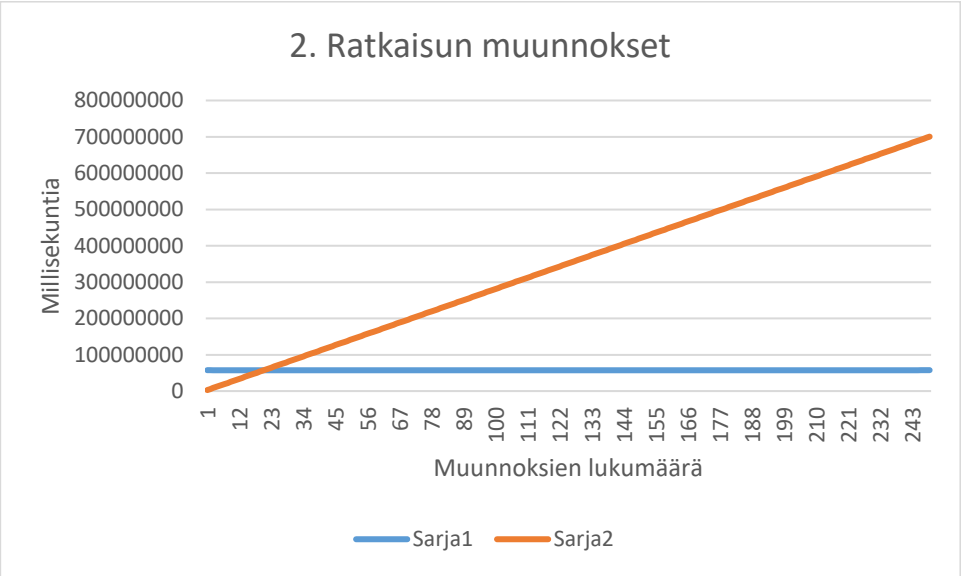
Taulukko 3. Manuaaliset muunnokset

Määrite	**Aika**	**Selite**
Hitain aika	46:53.12	minuuttia:sekuntia.millisekuntia
Nopein aika	46:53:12	minuuttia:sekuntia.millisekuntia
Keskimääräinen aika	46:53:12	minuuttia:sekuntia.millisekuntia

Taulukko 4. Automaattiset muunnokset

Määrite	**Aika**	**Selite**
Hitain aika	308	millisekuntia
Nopein aika	39	millisekuntia
Keskimääräinen aika	76,987	millisekuntia

Kuvio 8. Toisen ratkaisun muunnokset



Automaattinen muunnos antoi tuloksia samassa linjassa, kuin ensimmäisen ratkaisun muunnokset. Toisen ratkaisun automaattisiin muunnoksiin yhdeltä testiajolta kului 19 sekuntia ja 170 millisekuntia. Testiajoon kuului 249 vastauksen tulokset. Päätin käyttää 249 tuhannen sijasta, koska tulokset olivat suuria verrattuna ensimmäisen ratkaisun tuloksiin. Esimerkiksi tässä tapauksessa 249 vastauksen lopputulos oli noin kuusi kertaa suurempi tiedostokooltaan kuin ensimmäisen ratkaisun lopputulos. Keskimäärin yhteen muunnokseen kului 76,988 millisekuntia, joista nopein muunnos oli 39 millisekuntia ja hitain 308 millisekuntia. Pienin ero muunnosten ja keskimääräisen

muunnoksen välillä oli 37,988 millisekuntia. Toisaalta suurin ero oli 231,012 millisekuntia.

5.3 Tulosten analyysi

Manuaalisten tulosten tarkkuus on inhimillisiä virheitä lukuun ottamatta tarpeeksi tarkka ainakin referenssiä varten. Manuaalisten muunnosten ajanotto tapahtui Windows 10 käyttöliittymän Hälytykset ja kello sovelluksen ajastin ominaisuutta käyttäen. Laitoin ajastuksen päälle, tein muunnoksen manuaalisesti ja pysäytin ajastuksen. Ajan tarkkuuteen vaikuttaa ainoastaan inhimilliset seikat, kuten kyky pysäyttää ajastin heti muunnoksen valmistuessa.

Muunnoksia tehdessäni huomasin seuraavia tekijöitä. Suurin vaikuttaja oli kirjoitusnopeus, johon myös liittyy keskittymiskyky. Lopullisen formaatin rakentaminen spontaanisti muunnoksia tehdessä oli hidasta ja virhealtista. Huomasin, että rakenteen kopiointi säästi huomattavan määrän aikaa ja vähensi virheitä. Tästä on todisteena ensimmäisen ratkaisun ensimmäinen ja toinen muunnos. Tein tarkoituksella ensimmäiseen muunnokseen rakenteen spontaanisti vasta muunnosta tehtäessä, kun taas toiselle muunnokselle kopioin valmiin rakenteen. Valmis rakenne oli paljon nopeampi täyttää. Muunnosten aikaerot ovat merkittäviä. Ensimmäiselle muunnokselle, jolle toteutin rakenteen muunnoksien aikana käytin 2 minuuttia, 45 sekuntia ja 70 millisekuntia. Toiselle muunnokselle käytin 2 minuuttia, 1sekuntia ja 14 millisekuntia. Toinen muutos oli 44 sekuntia ja 56 millisekuntia nopeampi kuin ensimmäinen. Voidaan siis päätellä, että manuaalisten muunnosten nopeus riippuu kirjoittajan nopeudesta, sekä valmisteluista ennen muunnoksen aloittamista. Oletan kuitenkin toteutettujen muunnosten perusteella, että muunnosten lukumäärästä riippumatta olisi keskimääräinen aika silti noin 2 minuuttia.

Toisessa ratkaisussa manuaalisia muunnoksia tehtäessä tuli vastaan ajankäytön tehokkuus. Tein tutkimusta varten yhden muunnoksen manuaalisesti käyttäen samoja

menetelmiä kuin ensimmäisen ratkaisun muunnoksille. Lopputuloksen luomiseksi kuluin kuitenkin niin paljon aikaa, että päätin olla tekemättä enempää muunnoksia manuaalisesti. Ylimääräisten muunnosten tekeminen olisi voinut antaa luotettavamman ajan lopullisessa vertailussa, mutta ajan vaihteluilla ei olisi ollut merkittävää vaikutusta tuloksiin. Vaikka manuaalinen muunnos olisi 10 minuuttia nopeammin toteutettu kuin saamani aika, olisi aika niin hidas, että yhden muunnoksen aikana voisi ajaa noin 28744 muunnosta automaattisesti. Laskelma ottaa huomioon manuaalisen muunnoksen pohja-ajan eli 46 minuuttia 53 sekuntia ja 12 millisekuntia ja vähentää pohja-ajasta pois 10 minuuttia, sekä jakaa jäljellä olevan ajan automaattisten muunnosten keskimääräisellä ajalla. Toisena todisteena automatiikan nopeudesta voisi olla laskelma automaattisen muunnoksen hitaimman ajan ja manuaalisen muunnoksen nopeimman ajan vertailu. Ottaen aikaisemmin lasketun ajan eli pohja-ajan, josta vähennetään 10 minuuttia ja automatiikan hitaimman ajan eli 308 millisekuntia, saadaan muunnoksien lukumääräksi 7185. Se tarkoittaa, että jopa huonoimmassa tapauksessa automatiikka on merkittävästi nopeampi kuin manuaalinen muunnos.

Näistä tuloksista voidaan päätellä, että toiselle ratkaisulle tehdyn manuaalisen muunnoksen merkitys on vain referenssitasolla. Toisen ratkaisun manuaalisen muunnoksen tulosaika on viite oikeaan tilanteeseen, jossa käytettäisiin samanlaisia aloitusformaatteja. Viite kuitenkin riittää antamaan suuntaa manuaalisten muunnosten hitaudesta. Omalta osaltani voin sanoa, että tein muunnoksen niin tehokkaasti, kuin pystyin. Käytin myös hyväkseni aikaisemmin todettuja hyviä käytänteitä eli kopioin olemassa olevia rakenteita, sekä tekstiä siellä missä mahdollista.

Automaattisten muunnosten toteutukseen kului yhteensä noin 24 tuntia, josta 6 tuntia on molemmille yhteistä aikaa. Ensimmäisen ratkaisun muunnosten toteuttamiseen kului noin 14 tuntia eli 6 tuntia perusosuuden toteuttamiseen ja 8 tuntia ANTLR-generaattorin kieliopin ja muunnososuuden toteuttamiseen. Toisen ratkaisun toteuttamiseen kului noin 16 tuntia, jonka ajankäyttö jaettiin samalla tavalla.

Muunnosten kustannustehokkuus saadaan, kun lisätään ratkaisun automaattisten muunnoksien aikaan toteutuksen aika ja jaetaan tulos manuaalisten muunnosten ajalla. Laskelmissa täytyy myös ottaa huomioon, että manuaalisia muunnoksia voidaan toteuttaa koko kehitys ja ajoprosessin ajan. Manuaalisten muunnosten keskiaika oli 2 minuuttia, 12 sekuntia ja 872,8 millisekuntia per muunnos. Automaattisten muunnosten toteutusaika oli 14 tuntia. Laskettaessa muunnoksiin käytettyjen aikojen eroja saadaan muunnosten määräksi 416. Tarkoittaen, että siinä ajassa, kun tehdään 416 manuaalista muunnosta, voidaan vaihtoehtoisesti toteuttaa automaattinen muuntaja ja 416 automaattista muunnosta.

Tehtäessä 416 manuaalista muunnosta, kuluisi tehtävään noin 1,75 henkilötyöpäivää aikaa. Olettaen, että henkilön palkka on 2500 € kuussa ja henkilötyöpäivän pituus 8 tuntia, maksaisi muunnosten tekeminen noin 208 euroa. Esimerkki voi antaa kuvan, että erot eivät ole kovin merkitseviä, mutta erot kasvavat merkittävästi, kun tehdään 1000 muunnosta. Manuaalisesti käännettäessä 1000 alkiota tietoa kuluisi tehtävään noin 33,6 tuntia aikaa. Tämä on olettaen, että ihminen ei tekisi mitään virheitä ja kykenisi muuntamaan tietoa järjestelmällisesti, kuin tietokone. Samaan lukumäärään automaattisia muunnoksia kuluisi noin 14 tuntia, 4 sekuntia ja 836 millisekuntia. Toisin sanoen, automaattiset muunnokset tarvitsevat tuotantovaiheen jälkeen vain lyhyen ajan tehtävän suorittamiseen. Muunnoksiin kulutettu aika ja niiden erot kasvavat eksponentiaalisesti alkioiden lukumäärän kasvaessa. Laskematta monetaarisia kuluja voidaan nähdä, että manuaalisten muunnosten toteuttaminen ei ole yhtä käytännöllistä kuin automaattisten muunnosten toteutus.

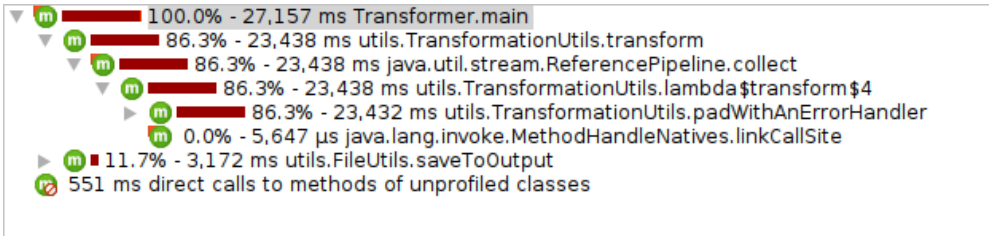
Aikaisempi esimerkki on hyvin yksinkertaisella datalla. Voidaanko sen perusteella olettaa, että kaikki automaattiset muunnokset ovat kustannustehokkaampia? Tämän tarkasteluun voidaan hyödyntää toisen ratkaisun tuloksia. Toisessa ratkaisussa data on paljon kompleksisempaa, joten sen muuntaminen on myös työläämpää. Sen sijasta, että käytetään ohjelmiston toteutukseen kulunutta pohja-aikaa, kasvatetaan aika 100 kertaiseksi. Uutena pohja-aikana olisi siis 600 tuntia, johon lisätään muunnokseen käytetty aika, eli 10 tuntia. Laskettaessa manuaalisten- ja automaattisten

muunnosten aikaerot, saadaan tulokseksi noin 780. Laskutoimituksesta saatu jäljellä oleva aika ei olisi tarpeeksi, että voitaisiin tehdä enempää muunnoksia. Tulos kertoo, että automaattisiin muunnoksiin käytetty aika on erittäin aloituspainotteista. Vaikka aloitusaika vaihdettiin 100 kertaiseksi, olisi automaattiset muunnokset kustannustehokkaampia toteuttaa. Tuhannen muunnoksen kohdalla manuaalisiin muunnoksiin menisi noin 171,4 tuntia enemmän aikaa, kuin automaattisiin, vaikka automaattisten muunnosten pohja-aika olisi 600 tuntia. Käyttämässäni datasetissä on 1561 riviä vastauksia. Kaikkien vastausten manuaalisiin muunnoksiin menisi noin 610 tuntia enemmän aikaa kuin automaattisiin, jos pohja-aika olisi 600 tuntia. Minun ohjelmani tapauksessa perusaika oli 6 tuntia, jonka perusteella manuaaliset muunnokset olisivat noin 1214 tuntia hitaampia toteuttaa.

Automaattisten muunnosten ajan tarkkuus on todella tarkka koska aika on laskettu järjestelmän millisekunneissa. Muunnoksien aikojen vaihtelut ovat paljon suurempia suhteessa manuaalisten muunnosten aikoihin. Ajoin muuntajan ohjelman profiloijan läpi ja tuloksista saadaan ilmi, että ainakin Java implementaation roskien poisto (garbage collection) vaikuttaa tietyin väliajoin suorituskykyyn. Profiloinnin tilastoista (kuvio 9) voidaan myös todeta, että ”transform” metodi käytti 86,3 % ohjelman ajon kokonaisajasta. Seuraavaksi isoin ajankäyttö on lopullisen tiedoston tallentamiselle, johon kului 11,7 % ajasta. Vielä tarkemmin suorituskyvystä kertoo ongelmapaikat (hot spots), joista voidaan todeta, että ANTLR-generaattorin luoman ohjelmiston parsinta käyttää suurimman osan prosessorin ajasta. Kuviosta 10 nähdään ohjelmistossa esiintyneet ongelmapaikat. Muistinkäytön statistiikasta voidaan todeta, että eniten muistia vaatii tekstisisältö eli merkkijonot. Merkkijonot käyttivät muunnosten ajonaikana noin 150 megatavua muistia (kuvio 11). Profilointi antaa yksinkertaista statistiikkaa ohjelmiston toiminnasta, sekä sen potentiaalisista puutteista. Muuntajaohjelmistoni suurin puute on moniajon puuttuminen. Profiloinnista nähdään (kuvio 12), että ohjelmisto käyttää vain yhtä säiettä (thread) ajonaikana. Lisäämällä moniajoa ohjelmistoon voitaisiin nopeuttaa hitaampia osuuksia huomattavasti. Tätä kautta voitaisiin saada muunnoksien ajaminen vielä tehokkaammaksi. Omassa ohjelmistossani aika ei

vielä tule ongelmaksi, mutta jos lähdetiedossa olisi miljoonia tai miljardeja alkioita voitaisiin nopeuttaa ajoa vähintään moniajamalla tiedostojen parsintaa.

Kuvio 9. Ohjelmiston metodi kutsuntapuu



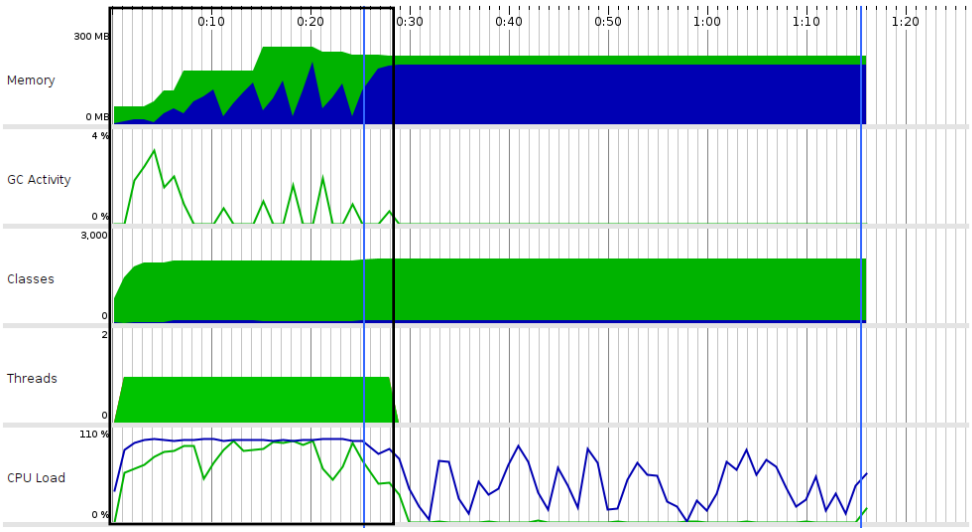
Kuvio 10. Ohjelmiston ongelmapaikat

Hot Spot	Self Time
org.antlr.v4.runtime.atn.ParserATNSimulator.adaptivePredict	9,326 ms (34 %)
org.antlr.v4.runtime.Parser.match	4,785 ms (17 %)
org.antlr.v4.runtime.Parser.consume	2,449 ms (9 %)
java.lang.String.format(java.lang.String, java.lang.Object[ ])	1,937 ms (7 %)
org.antlr.v4.runtime.tree.ParseTreeWalker.walk	1,435 ms (5 %)
java.lang.String.format(java.util.Locale, java.lang.String, java.la...	949 ms (3 %)
java.nio.file.Files.readAllLines	634 ms (2 %)
org.antlr.v4.runtime.RuleContext.getText	621 ms (2 %)
Transformer.main	546 ms (2 %)
org.antlr.v4.runtime.DefaultErrorStrategy.sync	502 ms (1 %)
org.antlr.v4.runtime.CharStreams.fromString	376 ms (1 %)
java.util.stream.ReferencePipeline.collect	285 ms (1 %)
java.lang.String.getBytes	228 ms (0 %)
java.time.LocalDate.now	209 ms (0 %)
org.antlr.v4.runtime.ParserRuleContext.getRuleContext	197 ms (0 %)
java.lang.ClassLoader.loadClass	177 ms (0 %)
java.lang.String.split	134 ms (0 %)
org.antlr.v4.runtime.BufferedTokenStream.LA	121 ms (0 %)
org.antlr.v4.runtime.Parser.enterRule	108 ms (0 %)
org.antlr.v4.runtime.tree.TerminalNodeImpl.getText	104 ms (0 %)
java.lang.String.join	94,129 µs (0 %)
org.antlr.v4.runtime.Parser.exitRule	89,523 µs (0 %)
org.antlr.v4.runtime.ParserRuleContext.getRuleContexts	85,075 µs (0 %)
java.lang.invoke.MethodHandleNatives.linkMethodHandleConst...	78,931 µs (0 %)
utils.TrainingsParser.field_value	66,927 µs (0 %)
java.util.Collection.stream	52,197 µs (0 %)
utils.EnvironmentsParser.table_value_choice	51,381 µs (0 %)
java.time.format.DateTimeFormatter.ofPattern	51,258 µs (0 %)
java.lang.invoke.MethodHandleNatives.linkCallSite	46,399 µs (0 %)
org.antlr.v4.runtime.ParserRuleContext.getToken	44,541 µs (0 %)
model.trainings.TrainingsResult.toString	43,791 µs (0 %)
java.time.format.DateTimeFormatter.<clinit>	43,387 µs (0 %)
utils.EnvironmentsParser.table_text	42,044 µs (0 %)
org.antlr.v4.runtime.Parser.<init>	41,684 µs (0 %)

Kuvio 11. Ohjelmiston muistinkäyttö

Name	Instance Count	Size
char[ ]	465,551	148 MB
java.lang.String	365,130	8,763 kB
java.util.Formatter$FixedString	64,527	1,548 kB
java.util.Formatter$Flags	48,086	769 kB
java.util.Formatter$FormatSpecifier	48,076	1,923 kB
java.util.Formatter$Flags[ ]	48,067	1,673 kB
java.lang.Object[ ]	40,939	1,918 kB
int[ ]	35,680	10,135 kB
java.lang.StringBuilder	34,060	817 kB
model.environments.EnvironmentsGenericData	25,896	621 kB
java.util.ArrayList	22,512	540 kB
java.util.Formatter	16,451	526 kB
java.util.Formatter$FormatString[ ]	16,451	779 kB
java.util.regex.Matcher	16,448	1,052 kB
java.util.concurrent.ConcurrentHashMap$Node	2,814	90,048 bytes
model.environments.EnvironmentsTableData	2,739	65,736 bytes
java.util.HashMap$Node	2,584	82,688 bytes
java.util.ArrayList$Itr	2,435	77,920 bytes
java.lang.Class	2,350	265 kB
org.antlr.v4.runtime.atn.ATNConfig	2,302	73,664 bytes
byte[ ]	2,216	16,753 kB
java.lang.Object	2,154	34,464 bytes
model.environments.EnvironmentsNonTableData	1,494	35,856 bytes
model.environments.EnvironmentsNonTableDataEntry	1,465	35,160 bytes
model.trainings.TrainingsData	999	47,952 bytes
model.trainings.TrainingsParticipants	999	31,968 bytes
model.trainings.TrainingsResult	999	31,968 bytes
org.antlr.v4.runtime.atn.EpsilonTransition	658	15,792 bytes
org.antlr.v4.runtime.misc.Interval	598	14,352 bytes
java.lang.Class[ ]	588	15,000 bytes
java.util.Hashtable$Entry	527	16,864 bytes
Total:	1,292,529	197 MB

Kuvio 12. Ohjelmiston säikeet. Musta ruutu oli ajonaikainen resurssien käyttö



6 Pohdinta

Kuten aiemmassa luvussa todettiin, automaattiset muunnokset voivat olla eksponentiaalisesti nopeampia kuin manuaaliset muunnokset. Tässä tutkimuksessa käytettiin yksinkertaista dataa, jonka oli sovitettu tutkimusongelmaan. Oikean maailman tilanteessa data ei aina välttämättä ole yksinkertaisessa muodossa, eikä automaatiota välttämättä edes harkita muunnoksien tekemisessä. Tutkimuksen perusteella voidaan kuitenkin todeta, että tilanteissa, joissa dataa voidaan parsia tai käsitellä, olisi syytä harkita automatiikan toteuttamista. Vaikka data olisi hyvin monimutkaista ja ajankäyttö painottuisi projektin alkuun, voidaan nähdä suuria käytännöllisiä hyötyjä muunnosten ajamisessa myöhemmin. Projektihenkilöt voivat myös toteuttaa muita tehtäviä, kun muunnoksien ensimmäinen versio on tehty. Ainoastaan ylläpito ja parannustehtävät ovat tarpeellisia ensimmäisen toteutuksen jälkeen. Suurin hyöty automaattisista muunnoksista on henkilöresurssien ja ajan säästö muunnoksia ajaessa.

Manuaalisesti käännetyissä tiedostoissa muutosten tekeminen on hidasta ja virhealtista. Muutokset täytyy tehdä jokaiselle tiedostolle erikseen. Manuaalisissa muunnoksissa suurin ongelma piilee jatkuvien muutoksien toteuttamisessa. Tarvittaessa nopeasti toteutettavia muutoksia malleihin tai sisältöön on yksinkertaisempaa käyttää muuntajan kaltaista ohjelmistoa. Muuntajaohjelmistossa olisi mahdollista muuttaa lopullinen tietorakenne täysin alle tunnissa. Muunnoksia suunnitellessa täytyy kuitenkin ottaa huomioon datan sopivuus ja laajuus ratkaisujen toteutuksiin nähden. Muunnettavan datan ollessa hyvin yksinkertaista voidaan käyttää myös yksinkertaisempia menetelmiä, esimerkiksi komentolinja skriptiä.

Prosessi tiedonkäsittelyyn on usein samanlainen kuin tässä tutkimuksessa. Ensin tieto luetaan muistiin kokonaan tai osissa. Sen jälkeen tieto normalisoidaan, jotta sitä voidaan helpommin käsitellä. Normalisointi voi olla niinkin yksinkertaista kuin koodauksen muuttaminen. Se voi myös sisältää yksittäisten datapisteiden käsittelyä, for-

matointia tai vaihtamista, jotta saavutetaan standardi formaatti lähdetiedoille. Normalisoinnin jälkeen tulee käsittelyvaihe, joka tämän tutkimuksen tapauksessa sisälsi tiedon parsimisen ja muunnoksen. Käytin hyödykseni ANTLR-generaattoria, joka jälkikäteen voidaan todeta olevan tehokas, mutta hieman liian jyhkeä ratkaisu. ANTLR-generaattori loistaa rekursiivisten rakenteiden parsimisessa. Rekursiivisia rakenteita esiintyy useimmiten ohjelmointikielissä. Käyttämissäni malleissa ei esiintynyt merkittäviä rekursiivisia rakenteita, joten en voinut saada kaikkea hyötyä irti ANTLR-generaattorin ominaisuuksista.

Vaihtoehtoisesti olisin voinut luoda oman räätälöidyn parserin, jolla tiedon käsittely olisi saattanut tapahtunut paljon tehokkaammin. Halusin kuitenkin tutustua ANTLR-generaattoriin oman henkilökohtaisen mielenkiinnon takia. Räätälöity parseri mahdollistaisi profiloinnin käytön vaikutuksellisemmin, kun voisi myös muokata parseria sisältäpäin ohjelmiston parantamiseksi. Parseri voitaisiin myös sovittaa yksittäisille ongelmille, mutta räätälöidyn ratkaisun vaihtokauppana on ajankäyttö. Projekteissa, joissa täytyy tehdä suurempia määriä muunnoksia, olisi kuitenkin taloudellista tehdä räätälöity ratkaisu, jotta voitaisiin välttää lukkiutumista muiden tahojen määrittelemiin rajoihin.

Räätälöidyn parserin lisäksi voitaisiin profiloinnin avulla selvittää niin sanotut ongelmapaikat (hot spots). Ongelmapaikat kertovat kuvailevasti ohjelmiston heikoimmat kohdat. Ohjelmiston toimintaa voitaisiin muuttaa tehokkaammaksi uudelleen toteuttamalla ongelmia aiheuttavat metodit ja logiikkarakenteet. Parannuskeinoja olisivat muun muassa moniajo, muuttumattomien (immutable) olioiden käyttö ja välimuistin käyttö paikoissa, joissa prosessointi tuottaa samoja tuloksia usein. Sen lisäksi voitaisiin optimoida muistinkäyttöä rajoittamalla muuttujien näkyvyysaluetta. Omassa ohjelmassani suurin osa muistista kuluu merkkijonoihin ja niiden käsittelyyn. Merkkijonojen optimointi on vaikeaa koska niissä on usein uniikkia dataa, johon ei voida välttämättä soveltaa aikaisempia parannuskeinoja. Muualla ohjelmistossa voitaisiin kuitenkin ottaa käyttöön moniajoa, jonka seurauksena ohjelmiston hitaimmat osuudet olisivat moninkertaisesti nopeampia. Parhain paikka minun ohjelmistossani käyttää

moniajoa olisi tekstitiedon parsinta. Kuten aikaisemmin nähtiin, parsinta kulutti yli 80 prosenttia koko ohjelmiston ajon ajasta. Moniajolla voitaisiin lyhentää aikaa merkittävästi.

Tutkimuksessa automatisointi rajattiin hyvin pieneen osa-alueeseen. Ohjelmistolle olisi mahdollista tehdä vielä enemmän automaatiota edeltäviä työkaluja. Esimerkiksi tiedon siivoamiselle voisi tehdä dynaamisemman ratkaisun ja antaa loppukäyttäjän valita mitkä kentät jätetään pois, mitkä arvot vaihdetaan selkokielisemmiksi tai miten korruptoituneet alkiot käsitellään. Tämä yhdistettynä yksinkertaiseen käyttöliittymään ja automatisoituun muunnosprosessiin mahdollistaisi dataformaattien muunnoksen vain muutamalla hiiren painalluksella. Tutkimus keskittyi enemmän itse muunnoksien automatisointiin, joten muunnokset vaativat hieman enemmän valmistelua.

Tietomuunnos on aina relevantti ja ajankohtainen aihe ohjelmistoalalla. Suuri osa ohjelmistosta toimii jonkinlaisen tiedon ympärillä ja usein tietoa täytyy muuntaa eri muotoihin. Automatisointi on nykyaikana yhä enemmän relevantti aihe, kun datan määrä kasvaa mahdottoman suureksi. Tutkimuksen automatisointi on hieman alkeellisella tasolla, ainakin verrattuna erilaisiin tekoälyratkaisuihin. Suurin ongelma rakenteellisesti ja semanttisesti muuttuvan tiedon muunnoksissa on juuri vaihtuvuus datassa. Tämän takia valitsin muunnoksille keinoja, jotka ovat käytännöllisempiä toteuttaa. Prosessissa on oltava erittäin tehokas tiedon siistiminen, jotta virheitä aiheuttavia syötteitä ei käsitellä. Vaihtoehtoisesti muuntajan virheenkäsittelyn on kyettävä sietämään virheitä ja tallentamaan vajavaista sisältöä.

Lähteet

Data types & file formats. N.d. Verkkoartikkeli. Viitattu 5.4.2020. https://data.library.virginia.edu/data-management/plan/format-types/.

Fagin, R., Haa, L.M., Hernández, M., Miller, R.J., Popa, L. & Velegrakis, Y. 2009. Conceptual Modeling: Foundations and Applications. Viitattu 16.4.2020. Clio: Schema Mapping Creation and Data Exchange. Berliini: Springer.

Fatima, N. 2020. Understanding data mapping and its techniques. Verkkoartikkeli. Viitattu 30.3.2020. https://www.astera.com/type/blog/understanding-data-mapping-and-its-techniques/.

File format definition. 2011. Verkkosivu. Viitattu 1.4.2020. https://techterms.com/definition/file_format.

Ishida, R. 2015. Character encodings for beginners. Verkkoartikkeli. Viitattu 2.4.2020. https://www.w3.org/International/questions/qa-what-is-encoding.

McBride, S. Gilder, R. David, R & Fenton, S. H. 2006. ”Data mapping” Journal of AHIMA 77, no.2. Verkkojulkaisu. Viitattu 31.3.2020. http://library.ahima.org/doc?oid=65895.

Parr, T. & Fisher, K. S. 2011. LL (*): The foundation of the ANTLR parser generator. Viitattu 6.4.2020. https://www.antlr.org/papers/LL-star-PLDI11.pdf.

Rangra, R. & Madhusudan. 2015. Basic parsing techniques in natural language processing. Viitattu 5.4.2020. http://www.warse.org/IJACST/static/pdf/file/ijacst02432015.pdf.

Tozzi, C. 2018. Data transformation in practice: 3 real-world data transformation examples. Verkkoartikkeli. Viitattu 31.3.2020. https://blog.syncsort.com/2018/11/big-data/data-transformation-3-examples/.

Yergeau, F. 2003. UTF-8, a transformation format of ISO 10646. Viitattu 4.4.2020. https://tools.ietf.org/html/rfc3629.