



TAMPEREEN TEKNILLINEN YLIOPISTO

JYRKI LAUNONEN
ALUSTARIIPPUMATON KIELIKÄÄNNÖSTEN
HALLINTATYÖKALU

Diplomityö

Tarkastaja: professori Hannu-Matti Järvinen

Tarkastajan ja aiheen on hyväksynyt

Tieto- ja sähkötekniikan tiedekunnan dekaani

9.8.2017

TIIVISTELMÄ

LAUNONEN, JYRKI: Alustariippumaton Kielikäännösten Hallintatyökalu

Tampereen teknillinen yliopisto

Diplomityö, 45 sivua

Joulukuu 2018

Tietotekniikan diplomi-insinöörin tutkinto-ohjelma

Pääaine: Ohjelmistotuotanto

Tarkastaja: professori Hannu-Matti Järvinen

Avainsanat: mobiilisovellus, kielellinen kääntäminen, Android, iOS, tiedostomuoto-
muunnos

Android ja iOS ovat tällä hetkellä käytetyimmät mobiilikäyttöjärjestelmät. Aloitettaessa mobiilisovellusprojektia on päätettävä, mille järjestelmille sovellus aiotaan toteuttaa, sekä toteutetaanko sovellus natiivisti vai käyttäen sovelluskoodin ristiinkäytön mahdollistavaa sovelluskehystä. Mikäli sovellus toteutetaan usealle järjestelmälle natiivisti, sovelluksen merkkijonot säilötään perinteisesti alustakohtaisissa resurssitiedostoissa moninkertaistaen käännettävän merkkijonotiedon määrän.

Tässä diplomityössä on toteutettu työkalu natiivisti toteutetun monialustaisen mobiilisovellusprojektin merkkijonojen keskitettyä hallintaa varten. Eri alustojen merkkijonoja säilytetään yhteisessä tietokannassa JSON-muotoisessa tekstitiedostossa. Python 3 -kielellä toteutettu kirjasto sisältää toimintoja tietokannan käsittelyyn. Toteutettuja toimintoja ovat muiden muassa merkkijonojen vienti tietokannasta eri kohdealustoille kunkin alustan vaatimassa tiedostomuodossa, tietokannan muuntaminen xlsx-tiedostoksi kääntämistä varten ja käännettyjen merkkijonojen tuonti xlsx-tiedostosta takaisin tietokantaan.

Työkalu vähentää käännettäväksi lähetettävien merkkijonojen lukumäärää. Projekti-
käytössä merkkijonojen määrä on ollut noin kolmasosan pienempi. Toisen alustan
huonosti yhteensopivat kehitystyökalut ovat kuitenkin haitanneet merkkijonojen yh-
teiskäyttöä. Toteutettu työkalu kaipaisikin automaatiikkaa helpottamaan merkkijo-
nojen yhdistämistä.

ABSTRACT

LAUNONEN, JYRKI: A Crossplatform Management Tool For Translation Files

Tampere University of Technology

Master of Science Thesis, 45 pages

December 2018

Master's Degree Programme in Information Technology

Major: Software Engineering

Examiner: Professor Hannu-Matti Järvinen

Keywords: mobile application, translation, Android, iOS, file format transformation

Android and iOS are currently the most used mobile operating systems. When starting a mobile software project it must be decided on which systems the software is going to be implemented and whether the implementation is going to be native or if a cross platform framework allowing code re-use is used.

In this thesis, a tool for centralized management of strings in a natively implemented multiplatform mobile application project has been implemented. Strings for different platforms are kept in a shared database that is a text file in JSON format. A library implemented in Python 3 language contains functionalities to manipulate the database. Some of these functionalities are exporting of the string from database to different target platforms in their required file format, transforming the database to an xlsx-file for used in translation, and importing the translated strings from the xlsx-file back into the database.

The tool reduces the amount of strings sent to translation. In project use there has been a reduction of about a one third in the amount of strings. Badly interoperating development tools of other of the platforms has hindered sharing of the strings. The implemented tool needs automation to ease the unification of the strings.

ALKUSANAT

Tämä diplomityö on tehty Tampereen teknillisen yliopiston Tietotekniikan laitokselle vuosien 2017–2018 aikana. Työssä on toteutettu monialustaisen mobiilisovellusprojektin merkkijonojen hallinnointiin keskitetty tietokanta ja työkalu, joilla eri alustojen merkkijonot saadaan yhtenäistettyä ja kielellisen käännöksen taakkaa pienennettyä. Työ on tehty Vincit Development Oy:lle.

Tietotekniikan laitoksen professori Hannu-Matti Järvinen on toiminut tämän diplomityön tarkastajana. Vincitillä Juha Simola on mentoroinut työn tekemistä palautetta antaen koko työn ajan. Kiitän molempia heidän kiinnostuksestaan aiheeseen, panoksestaan ja työn maaliin ohjaamisesta. Haluan lämpimästi kiittää myös Tiiti ja Timo Kellomäkeä tuesta, neuvoista ja oikoluvusta.

Tampereella, 18. marraskuuta 2018.

Jyrki Launonen

SISÄLLYS

1. Johdanto	1
2. Ohjelmistojen kääntäminen	3
2.1 Kansainvälistäminen	3
2.2 Kansainvälistetty ohjelmistoprosessi	4
2.3 Luonnollisten kielten erityspiirteet	6
2.4 Olemassa olevat merkkijonojen tallennusratkaisut	7
2.4.1 Android	8
2.4.2 iOS	10
2.5 Olemassa olevat käännösten hallintaratkaisut	12
3. Toteutusteknologiat	13
3.1 Python	14
3.2 JSON	16
4. Alustariippumaton hallintajärjestelmä	17
4.1 Ensimmäinen versio	17
4.2 Merkkijonojen keskitetyn hallinnan kokeilu	19
4.3 Tehty järjestelmä	21
4.3.1 Tietorakenne	23
4.3.2 Merkkijonojen vienti	25
4.3.3 Käännöstiedon tuonti	29
4.3.4 Tietokannan rakenne	32
5. Arviointi	36
5.1 Jatkokehitysideat	39

6. Yhteenveto	41
Lähteet	43
Liite A: Mukailtu ABNF	46
Liite B: Toisen version (v2) tallennusmuoto	47
Liite C: Lopullisen version (v3) tallennusmuoto	49

TERMIT JA NIIDEN MÄÄRITELMÄT

ABNF	Augmented Backus–Naur Form. Formaali järjestelmä kielen määrittelyyn ja kuvantamiseen.
Bash	Bourne-again shell. Laajalti Linux- ja macOS-järjestelmissä käytössä oleva komentotulkki.
CAT	Computer Aided Translation. Yleistermi kielellisessä käännöksessä käytettävistä työkaluista.
CLDR	Common Locale Data Repository. Paikallistamiseen käytetyn yleisen tiedon varasto.
CSV	Comma Separated Values. Tekstimuotoinen taulukkomuotoisen tiedon tallennusmuoto.
DOM	Document Object Model. XML-dokumenttien käsittelyyn liittyvä rajapinta.
ICU	International Components for Unicode. Ohjelmiston kansainvälistämiseen ja globalisaatioon erikoistunut CLDR:ää hyväksikäyttävä kirjastojoukko.
JPEG	Joint Photographic Experts Group. Häviöllinen kuvien pakkausmuoto.
JSON	JavaScript Object Notation. Rakenteellinen tekstimuotoinen tiedonvälitysmuoto.
PNG	Portable Network Graphics. Häviötön kuvien pakkausmuoto.
PDF	Portable Document Format. Alustariippumaton dokumenttien tallennusmuoto.
SVG	Scalable Vector Graphics. XML-muotoinen erityisesti vektorikuvien kuvausmuoto.
TM	Translation Memory. Aikaisemmat käännökset muistava käännöstyökalu.
tyhje	Tyhjä merkki kuten välilyönti tai sarkain.
XLSX	Office Open XML Workbook. Eräs taulukkolaskentaohjelmien tiedostomuoto.
XML	Extensible Markup Language. Rakenteellinen tekstimuotoinen tiedonvälitysmuoto.

1. JOHDANTO

Elämme maailmassa, jossa Google ja Apple käyvät mobiililaitteiden valtataistelua Android- ja iOS-käyttöjärjestelmillään. Nämä käyttöjärjestelmät tarjoavat käyttäjilleen suunnilleen samat ominaisuudet ja palvelut, vaikka ne laitteiden ruuduilta katseltuna näyttävätkin vaihtelevan erilaisilta. Teknisesti järjestelmät ovat kuitenkin toisistaan riippumattomia omine ekosysteemeineen, eikä toisessa järjestelmässä toimivaa sovellusta voi sellaisenaan käyttää toisessa. Jotta sovellus voisi toimia molemmissa järjestelmissä, sovellus on joko kirjoitettava kummallekin alustalle täysin omana versionaan tai käytettävä ulkoista ohjelmakoodin ristiinkäytön mahdollistavaa sovelluskehystä.

Sovelluksen käyttäjäkunta voi olla monikielistä jopa silloin, kun sovellusta jaetaan vain yhteen maahan. Hyvänä esimerkkinä on kotimaamme Suomi, jossa viralliset kielet ovat suomi ja ruotsi. Väestössä on myös englannin osajia, ja osa heistä saattaa osata englantia paremmin kuin suomea tai ruotsia. Kaikki sovelluksessa näytettävät tekstit on kehitysvaiheessa käännettävä erikseen jokaiselle halutulle kielelle.

Mikäli sovellusta kehitetään molemmille käyttöjärjestelmille omina versioinaan, on mahdollista, että eri versioiden kehitysnopeus ei täsmää ja tarveajankohdat ominaisuuksissa käytettävillä teksteillä eroavat toisistaan. Versioiden tarvitsemat tekstit ovat todennäköisesti suurilta osin keskenään samoja, ja niinpä tekstien käännettäminen molemmille versioille erikseen kaksinkertaistaa käänntöyön määrän turhaan. Lisäksi kahdesti kääntämisen seurauksena eri versioiden tekstit saattavat poiketa toisistaan. Voisiko tekstit kääntää vain kerran siten, että yhden sovellusversion kehitys ei riippuisi toisen version kehitysvaiheesta?

Tässä diplomityössä kuvataan Vincit Development Oy:n projektikäyttöön toteutettu merkkijonojen keskitetty hallintatyökalu ja sen kaksi iteratiivista esiastetta. Tavoiteltavaa ratkaisussa on riippumattomuus käyttökohteen tiedostomuodosta ja kehitysvaiheesta sekä versionhallinta, muokattavuus ja nopea käyttöönotto. Android- ja iOS-kehityksessä käytettävät resurssitiedostot eivät ole keskenään yhteensopivia, joten käänntötietoa joudutaan prosessissa muuntamaan muodosta toiseen. Ratkaisun perusajatus on sijoittaa tekstit eli merkkijonot käänntöksineen yhteiseen tietokan-

taan, jonka sisältö voidaan muuntaa osittain tai kokonaan haluttuun tiedostomuotoon. Käytännössä tietokanta on JSON-muodossa oleva tekstitiedosto, jonka käsittelyä varten on toteutettu Python-ohjelmointikielellä pienikokoinen kirjasto. Lisäksi kirjastoa käyttäen on toteutettu komentoriviltä ajettavia komentojonotiedostoja suorittamaan yleisimmin käytettyjä muunnostoimenpiteitä.

Tässä diplomityössä toteutetun työkalun avulla merkkijonoista saadaan yhtenäisemmät, ja kielellistä käännoästä vaativien merkkijonojen lukumäärä pienenee. Merkkijonotietokantaan lisätyt merkinnät mahdollistavat merkkijonojen käyttämisen yhteisesti eri alustoilla. Lisäksi työkalu ja sen osat ovat muokattavia ja laajennettavia, koska niiden lähdekoodi on työkalua käyttävän projektin hallinnassa. Tulevaisuuden haaveena on toteuttaa käännoästyötä varten verkkopalvelu siten, että se tarjoaisi kääntäjälle myös visuaalisen kontekstin merkkijonon sijainnista käyttöliittymässä ja mahdollisesta lopullisesta asettumisesta.

Luvussa 2 käydään läpi ohjelmistoprosessin sisältö ja mitä luonnollisten kielten piirteet vaativat ohjelmistolta. Työssä käytetyt teknologiat esitellään luvussa 3 pintapuolisesti. Luku 4 esittelee työssä kehitetystä hallintatyökalusta aikajärjestyksessä kolme eri versiota ja niiden yleiset toimintaperiaatteet ja ominaisuudet. Luvussa 5 arvioidaan työkalun eri versioiden ja kaupallisen toimijan työkalun ominaisuuksia, sekä mahdollisia jatkokehityksen suuntia. Lopuksi luvussa 6 on työn yhteenveto.

2. OHJELMISTOJEN KÄÄNTÄMINEN

Tässä luvussa esitellään ensin ohjelmistojen kansainvälistämiseen liittyvää taustaa, termejä ja huomioon otettavia asioita. Seuraavaksi käsitellään lyhyesti ohjelmistoprosessien keskeinen sisältö. Kolmantena käydään lävitse joitakin luonnollisten kielten piirteitä, joita joudutaan huomioimaan kansainvälisissä ohjelmistoissa. Lopuksi käsitellään olemassa olevia tallennusratkaisuja ja varsinaiset työssä toteutettuun hallintatyökaluun käytetyt teknologiat.

2.1 Kansainvälistäminen

Sovellusten kansainvälisille markkinoille viemiseen liittyvät termit kansainvälistäminen (internationalization) ja paikallistaminen (localization). Toisinaan englanninkielisistä termeistä näkee myös lyhennettyjä versioita *i18n* ja *l10n*, joissa keskellä oleva numero kertoo ensimmäisen ja viimeisen kirjaimen välistä pois jätettyjen kirjainten määrän [1, 2.1 – 2.2]. Molemmille termeille on useita määritelmiä ja lähteestä riippuen niiden sisältö voi vaihdella.

Markkina-alueiden eroavaisuuksien huomioon ottamista tuotteessa tai palvelussa kutsutaan paikallistamiseksi. Huomioon otettavana kohteena voi olla mikä tahansa tuotteen tai palvelun myyntiin tai käyttöön tarvittava osa. Tämä voi vaikuttaa merkittävästi organisaation teknisiin ja liiketoiminnallisiin toimintoihin. Paikallistamiseen sisältyy se, miten myyntiä tehdään; miten tuotteet ja palvelut on suunniteltu, rakennettu ja tuettu; kuinka finanssiraportoinnin järjestelmät on toteutettu; jne. Vaikka kääntämis- ja paikallistamistyössä on päällekkäisyyttä, paikallistaminen koskee yleensä merkittäviä ei-tekstuaalisia tuotteiden tai palveluiden komponentteja täsmällisen kääntämisen lisäksi. [2, s. 11]

Kansainvälistäminen on prosessi, jossa tuotteesta tehdään teknisellä tasolla paikallistettava. Toisin sanoen kansainvälistetty tuote ei vaadi paikallistamisen aikana korjaavaa työstöä tai uudelleensuunnittelua. Sen sijaan se on suunniteltu ja rakennettu mukautumaan helposti eri markkina-alueiden vaatimukseen kehitysvaiheen jälkeen. Kansainvälistämisessä eriytetään tuotteen sidonnaisuudet kulttuurista, kielestä ja

Taulukko 2.1. Joitakin kohteita, jotka kansainvälistämisessä ja paikallistamisessa on syytä muistaa ja ottaa huomioon [2, 3]. Lista ei kata kaikkia mahdollisia kohteita.

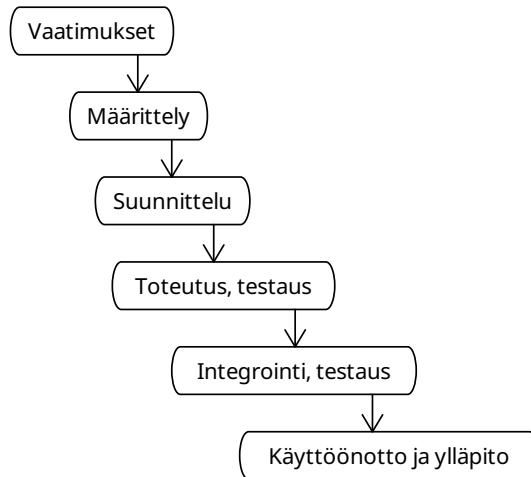
Kohde	Esimerkki
Kirjaimet	FI: A, B, C, ..., Z, Å, Ä, Ö EN: A, B, C, ..., Z
Luku- ja kirjoitussuunta	FI: Vasemmalta oikealle AR: Oikealta vasemmalle
Numerot	FI: 1 000 000,15 EN: 1,000,000.15
Ajanilmaukset	FI: 3. helmikuuta 2018 klo 13.15 EN: February 3, 2018 at 1:15 p.m.
Valuutta	FI: 15,00 € EN: \$15.00
Pikanäppäinten käyttö	FI: <u>T</u> iedosto EN: <u>F</u> ile
Lajittelu ja järjestäminen [4]	FI: ..., X, Y, Z, Å, Ä, Ö EN: A, Å, Ä, B, ..., O, Ö, P, ...
Säädökset	- Ikäraajat (muiden muassa rikosoikeudellinen, täysi-ikäisyys ja lasten tietojenkäsittely). - Arvonlisävero.
Symbolit / kuvakkeet	- (engl.) "Run" kuvattuna juoksevalla ihmisellä, kun tarkoitetaan suorittamista.

markkina-alueesta. Näin tuotteeseen voidaan helposti lisätä tuki uusille markkina-alueille ja kielille. [2, s. 17]

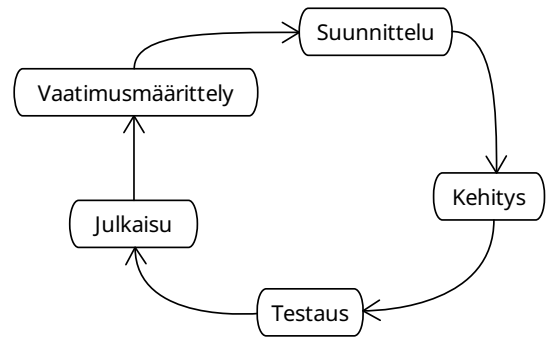
Paikallistamisessa joudutaan ottamaan huomioon kielellisiä, sisällöllisiä, kulttuurillisia ja teknisiä haasteita [2]. Taulukossa 2.1 on esitetty joitakin paikallistamista vaativia kohteita esimerkein. Ideaalitulanteessa paikallistaminen on otettu huomioon jo sovelluksen suunnitteluvaiheessa eikä se aiheuta suuria muutoksia toteutukseen [2].

2.2 Kansainvälistetty ohjelmistoprosessi

Perinteisesti ohjelmistoprojekti on jaettu osiin niin sanotun vesiputousmallin mukaisesti peräkkäisiin vaiheisiin. Kukin vaihe saa ohjeistuksensa edelliseltä vaiheelta, tuottaa niiden perusteella omat tuloksensa ja tarjoaa ne seuraavalle vaiheelle. Laadun varmistamiseksi vaiheiden lopuissa tarkastetaan tulokset vertaamalla niitä



Kuva 2.1. Esimerkki vesiputousmallin vaiheista. Mukailtu lähteestä [5].

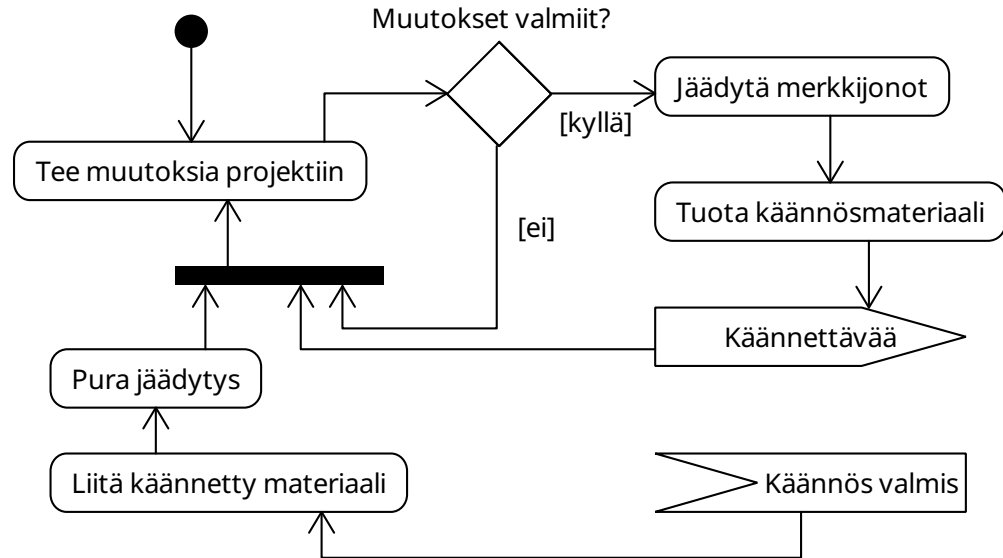


Kuva 2.2. Ketterän kehityksen sykli. Mukailtu lähteestä [6].

ohjeistukseen tarkastuksien tai testauksen avulla. Kuvan 2.1 kaaviossa on esimerkiksi vesiputousmallin mukaisesta vaihejaosta. Vesiputousmallin heikkous on virheistä saatava hidas palaute. Esimerkiksi määrittelyvaiheissa tehty virhe saatetaan huomata vasta integrointivaiheen testauksessa, jolloin sen korjaaminen saattaa vaatia suuria ja kalliita muutoksia jokaiseen välivaiheen tuotokseen. [5, s. 35 – 48]

Vesiputousmallin lisäksi ohjelmistoprojekteissa on kokeiltu useita eri lähestymistapoja jakaa työ vaiheisiin. Osassa malleista on perusratkaisuna käytetty alkuperäistä vesiputousmallia ja lähinnä tuotu malliin lisää joitakin uusia ideoita. 2000-luvun alussa julkistettu idea ketterästä kehityksestä pyrkii löytämään helpompia tapoja ohjelmistoprojektien toteuttamiseksi. Ketterä kehitys yhdistää osia vesiputousmallista, prototyypin tekemisestä ja askelittaisesta kehityksestä. Kehitys tapahtuu esimerkiksi kahden viikon mittaisissa lyhyissä jaksoissa, joiden aikana prosessin vaiheet käydään lävitse. Kuvassa 2.2 on esitetty kaavio ketterän kehityksen vaiheiden toistumisesta. Ketteriä malleja noudatettaessa dokumentaatiota tuotetaan merkittävästi vähemmän kuin perinteisessä vesiputousmallia noudattelevassa projektissa. [6]

Kuvan 2.3 kaaviossa on esitetty erään ohjelmistoprojektin prosessin kääntämiseen keskittyvä osa. Kaaviossa ei oteta kantaa projektin päättymiseen, versiointiin eikä tiedon välitysmuotoihin. Ennen ensimmäistä käännoistyötä prosessiin kuuluu lähinnä ensimmäinen tila, jossa projektia tehdään vapaasti. Sopivassa vaiheessa projektia käännoistyö käynnistetään valinnaisesti jäädyttämällä merkkijono. Jäädytys tarkoittaa, ettei merkkijonoja muuteta käännoistyön aikana ennen kuin jäädytys puretaan käännoistyön päätyttyä. Kääntämisen aikana projektin tekoa voidaan kuitenkin jatkaa rinnakkain. Joissain tapauksissa jäädytystä ei välttämättä tarvita, mutta tehdyt



Kuva 2.3. Esimerkki perinteisestä käännösprosessista ketterän projektin aikana.

muutokset saatetaan joutua kopioimaan tai toistamaan käännösten valmistuttua ja niiden projektiin mukaan ottamisen jälkeen. Käännösten mukaanoton jälkeen mahdollinen jäädytys puretaan.

2.3 Luonnollisten kielten erityispiirteet

Suomen sanojen taivutukset hankaloittavat sovellusten tekstien tekemistä, sillä korvattavia sanoja ei välttämättä ole tarjolla taivutettuina. Tästä voi aiheutua käännöksiin hieman kankeampia rakenteita. Esimerkiksi alun perin englanninkielinen lause "<Bucket> cannot be ordered" saatettaisiin kääntää muotoon "Ei voi tilata: <ämpäri>" tai hieman alkuperäisestä poiketen "Tuotetta <ämpäri> ei voi tilata." Luontevampi muoto olisi "<Ämpäriä> ei voi tilata", jos tuotenimelle on saatavilla taivutusmuodot.

Ajanilmaisussa on kielellistä ja paikallista vaihtelua. Sekä käytetyt välimerkit että päiväyksen osien järjestys saattavat vaihdella. Suomessa yleensä kirjoitetaan päivämäärän pitkä versio muodossa "3. helmikuuta 2018". Yhdysvalloissa päivämäärissä kuukausi kirjoitetaan ennen päivää, "February 3, 2018". Vastaava ero on lyhyessä muodossa, joka Suomessa kirjoitetaan "3.2.2018" ja Yhdysvalloissa "02/03/2018". Kellonajat ilmaistaan Suomessa kirjoitetussa tekstissä käyttäen 24 tunnin merkin-tätapaa pistettä erotinmerkkinä käyttäen, esimerkiksi varttia yli puolen yön kirjoite-taan "0.15". Sen sijaan Yhdysvalloissa käytetään 12-tuntista kelloa puhutun kielen lisäksi myös kirjoitetussa tekstissä. Erotinmerkkinä numeroilla toimii kaksoispiste. Edellinen kellonaika ilmaistaisiin muodossa "12:15 a.m." tai "12:15 AM". [3]

Kieliopillista numeraalia käytetään ilmaisemaan asioiden lukumääriä [7, § 770, § 772]. Numeraaleja ovat esimerkiksi sanat "yksi", "kaksikymmentä" ja "kolmas". Laskettavana asiana voi olla jokin mittayksiköllinen suure tai muu substantiivi kuten "metri" tai "tiedosto" [7, § 770, § 772], esimerkiksi "viisi metriä" tai "yksi tiedosto". Tyypillisesti arvoltaan kymmenen tai sitä suuremmat numeraalit kirjoitetaan numeroita käyttäen [7, § 770]. Numeraali määrittää lauseen muiden määritteiden numerusta niiden sijainnin perusteella [7, § 1305]. Numeraalin ollessa yksikön nominatiivissa toimii se lausekkeen edussanana [7, § 789]. Edussana on yhtäjaksoisessa rakenteessa eli lausekkeessa yleensä pakollinen osa kuten substantiivi substantiivilausekkeessa ja adjektiivi adjektiivilausekkeessa [7, § 439, § 441]. Yksikön nominatiivissa oleva numeraali on tarvittaessa helposti muutettavissa numeroksi. Esimerkiksi "kolme tiedostoa" on ilmaistavissa myös muodossa "3 tiedostoa".

Eri kielten monikkomuodot on luokiteltu kielikohtaisilla matemaattisilla säännöillä kategorioihin zero, one, two, few, many ja other. Joidenkin kategorioiden lukua muistuttava nimi viittaa absoluuttisen arvon sijaan numeroon luvun osana tai muuhun luokitteluperusteeseen. Suomessa ja yli sadassa muussa kielessä käytetään kategorioita one ja other siten, että one-kategoriolla ilmaistaan lukumäärä *tasan 1* ja other-kategoriolla kaikkia muita lukumääriä. Vertailuna samoja kategorioita käyttäen esimerkiksi islannin kielessä kategoriata one käytetään lukumäärän 1 lisäksi muille numeroon yksi päättyvissä lukusanoissa paitsi jos luku päättyy 11, eli esimerkiksi luvuilla 1, 21 ja 341. Muille lukumäärille käytetään kategoriata other. Useampia kategorioita käytetään esimerkiksi latviassa, liettuassa, venäjässä ja arabiasassa. Esimerkiksi kiinassa, japanissa ja koreassa käytetään vain yhtä kategoriata, other. [8]

2.4 Olemassa olevat merkkijonojen tallennusratkaisut

Sovelluksissa olevat merkkijonot voivat olla joko muuttumattomia tai ajonaikaisesti tuotettavia. Esimerkiksi muuttumaton merkkijono voisi olla "Kirjaudu sisään." Muuttuvat merkkijonot kirjoitetaan mallineina käytössä olevan muotoilujärjestelmän vaatimalla tavalla. Järjestelmä tuottaa mallineesta lopullisen näytettävän tekstin ajonaikaisesti. Yksinkertainen muotoilujärjestelmä on mahdollista toteuttaa esimerkiksi alimerkkijonon korvausrutiinilla. Mallinetta "Hei <NIMI>!" ja koodia `malline.replace("<NIMI>", käyttäjän_nimi)` käyttämällä yksi tulos saattaisi olla "Hei Mikko!" Tavallisia yleisiä muotoilujärjestelmiä ovat *printf* ja *MessageFormat*.

Seuraavissa aliluvuissa esitellään tässä työssä toteutetun työkalun kohdeprojektin kannalta oleelliset käyttöjärjestelmät. Molemmat järjestelmät käyttävät merkkijonon muotoiluun pohjimmiltaan `printf`-muotoista muotoilujärjestelmää. Edellisen

esimerkin mukainen tulos saadaan printf-järjestelmässä mallineella "Hei %s!" ja MessageFormat-järjestelmässä mallineella "Hei {}!"

2.4.1 Android

Kalifornialainen Android Inc. aloitti Android-käyttöjärjestelmän kehityksen vuonna 2003 tarkoituksenaan kehittää "älykkäämpiä mobiililaitteita, jotka ovat tietoisia käyttäjänsä sijainnista ja mieltymyksistä" [9]. Google Inc. osti yrityksen vuonna 2005, ja kolme vuotta myöhemmin julkaistiin ensimmäinen Androidin versio [9, 10]. Androidin eri versiot on nimetty aakkosjärjestyksessä herkkujen englanninkielisten nimien mukaan. Esimerkiksi versio 4.4 on nimeltään KitKat ja sitä seuraava 5.0 on Lollipop. Tällä hetkellä viimeisin versio on 9 (Pie).

Ensimmäisten tablettitietokoneiden tullessa markkinoille Android-sovellusten kehitys tablettikäyttöön ei ollut kiinnostavaa käyttäjäkunnan puuttuessa, toisaalta käyttäjäkunta ei kasvanut kovinkaan nopeasti sillä tablettikäyttöön suunnattuja sovelluksia ei ollut markkinoilla [11]. Tästä muna-kana-ongelmasta seuranneesta verkkaaisuudesta huolimatta järjestelmä onnistui laajentumisessaan [11]. Ensimmäinen tabletteja tukeva Android oli vuonna 2011 ilmestynyt versio 3.0 (Honeycomb) [12]. Android laajentui lisää vuonna 2014, kun tukea laajennettiin ensin puettaville laitteille (Wearable devices) kuten rankekelloille Android Wear -alustalla ja myöhemmin televisioihin Android TV -alustalla [13, 14].

Android-sovellusprojekti jakautuu karkeasti kahteen osaan: suoritettavaan ohjelmakoodiin ja resursseihin. Resursseja ovat esimerkiksi ikonit ja kuvat, värimäärittelyt, näkymien asettelut, animaatiot ja merkkijonot. Kaikille resursseille määritellään viittausta varten nimi, esimerkiksi väri `theme_background` tai sovellusvalikossa näkyvä ikoni `launcher`. Resurssien ja niiden kuvaava nimeäminen auttaa sovelluksen ylläpidossa ja mahdollistaa esimerkiksi sovelluksen yleisen taustavärien vaihtamisen vain muuttamalla kyseisen värimäärittelyn arvoa. Yksittäisten resurssien nimien on käytettävä pienellä kirjoitettuja alfanumeerisia merkkejä ja alaviivaa. Resurssikokoelman sisältä löytyvät resurssit voivat lisäksi sisältää isolla kirjoitettuja kirjaimia.

Sovelluksen kuvatiedostot ovat pääasiassa PNG-muodossa (Portable Network Graphics), mutta myös JPEG (Joint Photographic Experts Group) on tuettu muoto. Muut resurssit kirjoitetaan XML-merkintämuodossa. SVG-vektorikuvatiedostojen (Scalable Vector Graphics) käyttö edellyttää niiden muuttamisen Androidin ymmärtämään XML-muotoiseen vektorimuotoon.

```
<resources>
  <string name="authentication.login">Kirjaudu sisään</string>
  <string name="product.notAvailable">%s ei ole saatavilla.</string>
  <string name="days.left.single">Tämä päivä jäljellä.</string>
  <plural name="days.left">
    <item quantity="one">%d päivä jäljellä.</item>
    <item quantity="other">%d päivää jäljellä.</item>
  </plural>
</resources>
```

Listaus 2.1. Merkkijonot Androidissa.

XML-muotoa käyttävät resurssitiedostot voivat sisältää yhden tai useamman resurssin. Yksittäisen resurssin sisältäviä resurssityyppejä ovat näkymien asettelut, animaatiot ja kuvat. Kokoelmatiedostoihin voidaan määritellä merkkijonoja ja yksittäisiä värejä. Esimerkki tällaisesta merkkijonojen resurssikokoelmasta on esitetty listauksessa 2.1.

Android-natiivisovelluksen ohjelmakoodi kirjoitetaan Java-kielellä tai kielellä, joka on käännettävissä Javan tavukoodiksi. Erityisten alustariippumattomien ohjelmointialustojen avulla ohjelmakoodia voidaan kirjoittaa myös C#- ja JavaScript-kielillä. Sen lisäksi, että resursseissa voi olla viittauksia muihin resursseihin, voidaan ohjelmakoodissa viitata resursseihin kehitystyökalujen luoman R-luokan kautta. Esimerkiksi tekstinäkymälle voidaan asettaa näytettävä merkkijono sanomalla `textView.setText(R.string.authentication_login)`.

Sovellusprojektin resurssit voidaan ryhmitellä erilaisten kategorioiden avulla. Käytettävistä muista kuin kuvaresursseista on määriteltävä vähintään oletusversiot, joihin järjestelmä palaa, mikäli tarkempaa versiota ei löydy. Jos sovellukseen määritellään esimerkiksi oletusversioksi englanninkieliset ja lisäksi suomenkieliset tekstit, näytetään sovelluksen tekstit ruotsinkielisellä laitteella englanniksi ja suomenkielisellä laitteella suomeksi. Näin sovellus on tarjolla tarkoitettulle pääkohdeyleisölleen sen omalla kielellä, mutta mahdollistaa muunkielistenkin käyttäen sitä.

Androidin työkalusto mahdollistaa merkkijonojen määrittelyssä yksittäiset merkkijonot, monikkomuodolliset merkkijonot ja merkkijonolistat. Monikkomuodollisissa merkkijonoissa on kuitenkin huomattavaa, että merkkijonossa voi olla vain yksi monikollinen osa. Tämä tarkoittaa, että teksti "Talossa on 1 kissa" on mahdollista ilmaista monikossa "Talossa on 5 kissaa", mutta tekstin "Talossa on 1 kissa ja 1 koira" määrittely on vaikeaa, sillä se joudutaan jakamaan osiin ja kokoamaan käsin sovelluskoodissa lopulliseksi tekstiksi. Lisäksi osiin jakaminen aiheuttaa eri osien yhteenkuuluvuuden ymmärtämisen vaikeutumisen.

Kaikki Androidin merkkijonoresurssien merkkijonot voivat sisältää korvauksia eli paikkamerkintöjä, joihin sijoitetaan vaihtuvaa sisältöä. Javan ja Androidin merkkijonojen muotoilujärjestelmä on pääosin yhteensopiva printf-muotoilun kanssa.

2.4.2 iOS

Steve Jobs aloitti iPhoneen suunnittelun 2005 ja joutui tekemään päätöksen siitä, kehitetäänkö iOS aiemman iPodin vai MacOS:n pohjalta. Tätä päätöstä varten hän järjesti sisäisen tiimien välisen kilpailun, jonka voittaja vastaisi iOS:n kehityksestä. Kilpailusta kehkeytyi lopulta kuumatunteinen taistelu tiimien vetäjien väitellessä kyvyistä, voimavaroista, huomiosta ja kunniaista. Lopulta MacOS-tiimi onnistui saamaan järjestelmänsä toimimaan ja Jobs valitsi tämän voittajaksi. [15]

Vuoden 2007 alussa Steve Jobs ilmoitti ensimmäisen iPhoneen julkaistavan kesäkuussa [16]. Laitteen käyttöjärjestelmään viitattiin nimellä OS X, joka saattoi antaa väärän kuvan siitä, että laite olisi pystynyt ajamaan Mac OS X -ohjelmia [17]. Käytännössä käyttöjärjestelmä oli oikeastikin OS X, mutta työpöytä tietokoneiden Mac OS X:n sijaan siitä kutistettu ja mobiililaitteelle optimoitu versio [17]. Nimeä muutettiin aluksi vuonna 2009 muotoon iPhone OS ja lopulta vuonna 2010 nykyiseen muotoonsa iOS [17].

Ohjelmakoodia voidaan kirjoittaa iOS-sovellusprojektiin mm. Objective-C- ja Swift-kielillä. Projektiin voidaan sisällyttää XML- tai tekstimuotoisia resursseja sekä PNG-, JPEG- ja PDF-kuvia. Ohjelmakoodista tehdyt viittaukset resursseihin tapahtuvat merkkijonojen avulla. Tästä seuraa, että käännettäessä sovellusta ajettavaksi paketiiksi puuttuvat tai väärin nimetyt resurssit saadaan selville vain erillisiä lisätyökaluja käyttämällä. Merkkijonojen avaimina voidaan käyttää mitä tahansa merkkijonoa.

Osa sovelluksen käyttäjälle näkyvistä merkkijonoista välitetään listauksen 2.2 esimerkin mukaisessa avain-arvo-pareja sisältävässä *strings*-tiedostossa. Tiedostossa voidaan käyttää C- (`/* Kommentti */`) ja C++-tyylisiä (`// Kommentti`) kommentteja. Koska *strings*-tiedoston muoto on yksinkertainen, monikolliset merkkijonot välitetään erillisen XML-muotoisen *stringsdict*-tiedoston avulla listauksen 2.3

```
"authentication.login"="Kirjaudu sisään";  
"purchase.productNotAvailable"="%@ ei ole saatavilla."  
"days.left.single"="Tämä päivä jäljellä";
```

Listaus 2.2. Merkkijonot iOS:n strings-tiedostossa.

```

<plist version="1.0">
  <dict>
    <key>days.left</key>
    <dict>
      <key>NSStringLocalizedFormatKey</key>
      <string>%1$#@plural@</string>
      <key>plural</key>
      <dict>
        <key>NSStringFormatSpecTypeKey</key>
        <string>NSStringPluralRuleType</string>
        <key>NSStringFormatValueTypeKey</key>
        <string>d</string>
        <key>one</key>
        <string>%d päivä jäljellä.</string>
        <key>other</key>
        <string>%d päivää jäljellä.</string>
      </dict>
    </dict>
  </dict>
</plist>

```

Listaus 2.3. Monikkomuodolliset merkkijonot iOS:n stringsdict-tiedostossa.

esimerkin mukaisesti. Stringsdict on monisanaisempi kuin strings-muoto, mutta mahdollistaa useiden monikko-osien määrittelyn merkkijonoon. Yksittäisen monikon kirjoittaminen stringsdictiin on kuitenkin vaikeampaa kuin Androidin resursseihin, sillä monikkomuodot on aina kirjoitettava kuin niissä olisi useampi monikko-osa. Lisäksi käytetty *plist*-tiedostomuoto on vain yleiskäyttöinen hierarkkisten avain-arvojen tallennukseen tarkoitettu muoto eikä se rakenteessaan tarjoa vastaavaa tukea lukijalle kuin Androidin resurssitiedostot.

Niin sanotuilla *storyboard*-tiedostoilla kehitetään sovelluksen näkymiä ja niiden välisiä navigointipolkuja. Lisäksi yksittäisiä näkymiä tai osa-näkymiä voidaan tallettaa *XIB*-tiedostoihin (XML Interface Builder). Molempiin on mahdollista sijoittaa muuttumattomia merkkijonoja. Merkkijonojen poiminta näistä tiedostoista tuottaa tuloksena strings-tiedoston, jossa merkkijonojen avaimina on satunnaisgeneroitu näkymän avain, kuten `Ppw-ki-5yD`.

Korvausten merkintään käytetään iOS:n merkkijonoissa pääosin printf-yhteensopivaa tyyliä. Merkittävin ero standardiin on erikoistapaus merkkijonojen korvauksissa, joiden korvausmerkintänä käytetään symbolin "%s" sijaan symbolia "%@", esimerkiksi "Hei %@!".

2.5 Olemassa olevat käännosten hallintaratkaisut

Varsinaisen kääntämiseen käytetään usein kääntämiseen ja käännoistyön tukemiseen erikoistuneita työkaluja ja ohjelmistoja. Tällöin puhutaan tietokone-avusteisesta kääntämisestä, englanniksi computer-aided translation tai lyhennettynä CAT. Tällaisia työkaluja ovat termistön hallintajärjestelmä (Terminology Management System), käännosmuisti (Translation Memory) ja konekääntäminen (Machine Translation). Termistön hallinta ja käännosmuisti ovat usein yhdistetty yhdeksi työkaluksi. [2, s. 37 – 40]

Termistön hallintajärjestelmään on talletettu projektissa käytetyt termit ja termien kuvaukset käännoksineen. Joskus termin tiedoissa voi olla myös listattuna vaihtoehtoiset termit ja tietoa käyttötapauksista. Järjestelmä mahdollistaa termien yhtenäisen ja oikean käytön käännoiprojektin varsinaisessa tekstissä. [2, s. 37 – 38]

Käännosmuistityökalu jaottelee tekstit osiin ja pitää muistissaan käännoistyistä osista sekä nykyiset että vanhat versiot. Muistin avulla aiemmin käännoistyjä tekstin osia voidaan hyödyntää myöhemmin alkuperäisten tekstien muuttuessa. Työkalu voi kääntää muuttuneen tekstin automaattisesti tai tarjota kääntäjälle ehdotuksen vanhojen käännoisten perusteella. Käännosmuisti ja termistön hallintajärjestelmä ovat usein yhdistetty samaksi työkaluksi. [2, s. 37 – 39]

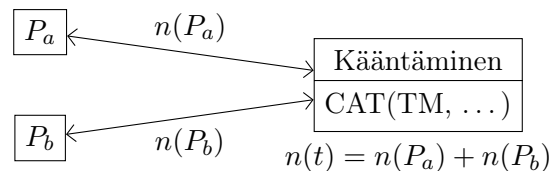
Konekääntämisessä varsinainen kääntäminen tapahtuu automaattisesti eikä edeltäviä käännoistuloksia tarvita käännoistuloksen luomiseksi. Käännoistyökalun algoritmit voivat analysoida käännoistyvän tekstin kieliopillisesti, tehdä tilastollisia päätelmiä tai käyttää näiden mekanismien yhdistelmää tuottamaan käännoisty lopputeksti. Yleensä tuloksen laatu ei yllä kääntäjän tekemän työn tasoon ja kaipaa siksi lähes poikkeuksetta jälkikäsitteilyä. [2, s. 38 – 39]

3. TOTEUTUSTEKNOLOGIAT

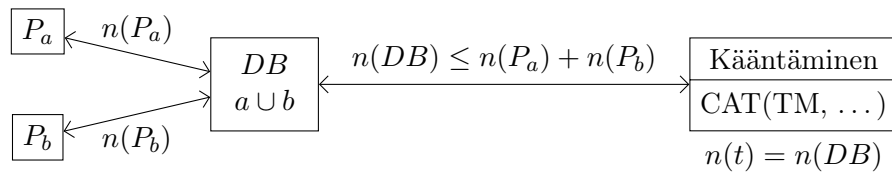
Sovellusprojektien kääntämistyötä varten on olemassa verkkopalveluita, jotka varastoivat sovelluksen merkkijonot omaan tietokantaansa ja tarjoavat käyttöliittymän käännöstyöhön. Tällaisten palveluiden huono puoli on, että yleensä merkkijonoihin ei monialustaisesta projektista voida liittää tietoa merkkijonoa käyttävistä alustoista. Merkkijonojen yhdistäminen palvelun käyttöä varten ja vastaavasti niiden erottelu projektiin tuontia varten voi olla mahdollista, mutta se vaatisi ylimääräistä ja erillistä kirjanpityötä. Palveluiden käyttöön liittyy lähes poikkeuksetta myös sana- ja kielimääriin perustuvia säännöllisiä kustannuksia. Vaihtoehtoisesti sovellusversioiden tekstit voitaisiin kääntää erillisinä käännösprojekteina, jolloin lisäkirjanpitoa tai yhteistä tietokantaa ei tarvittaisi. Tästä aiheutuisi todennäköisesti versioiden välisiä eroja teksteissä ylimääräisten kustannusten lisäksi.

Perinteisesti merkkijonojen kääntäminen tapahtuu projektikohtaisesti. Kuvan 3.1 kaavio kuvaa tilannetta summittaisesti. Alustojen P_a ja P_b merkkijonovarastot sisältävät $n(P_a)$ ja $n(P_b)$ merkkijonoa. Varastot käännetään erikseen esimerkiksi käännöstoimistossa käännöstyökaluja käyttäen. Käännettäväksi lähetetään molempien varastojen kardinaliteettien summa $n(t)$. Varsinaisen käännettävän materiaalin määrä riippuu käännösmuistin olemassaolosta ja toiminnasta.

Työssä toteutetun työkalun ja tietokannan avulla prosessi muuttuu kuvan 3.2 mukaiseksi. Molempien alustojen merkkijonot on talletettu tietokantaan DB , joka sisältää alustojen merkkijonojoukkojen yhdisteen $a \cup b$ eliminoiden merkkijonojen kopiot. Käännettäväksi lähetetään tietokannan sisältämät $n(DB)$ merkkijonoa. Lähetetyn joukon kardinaliteetin odotetaan olevan eliminoinnin takia alustojen merkkijonojen



Kuva 3.1. Kaavio monialustaisen projektin merkkijonojen määristä, kun molemmat alustat P_a ja P_b hoitavat merkkijonojen hallinnoinnin ja kääntämisen toisistaan riippumattomasti.



Kuva 3.2. Kaavio monialustaisen projektin merkkijonoista, kun alustat P_a ja P_b käyttävät yhteistä merkkijonotietokantaa.

leikkauksen verran pienempi joukko kuin alkuperäisten joukkojen kardinaliteettien summa.

Tässä luvussa on kuvattu tässä työssä toteutetussa työkalussa käytetyt ydinteknologiat. Toteutuksen pääasiallisena kielenä käytetty Python-kieli on kuvattu aliluvussa 3.1. Työkalun käsittelemät tietokannat ovat JSON-muodossa. Muoto on yksinkertaisuudessaan käsitelty aliluvussa 3.2.

3.1 Python

Alankomaalainen Guido van Rossum aloitti Python-kielen kehityksen vuoden 1989 loppupuolella ja julkaisi ensimmäisiä sisäisiä versioita kielestä vuonna 1990 Centrum Wiskunde & Informatica -tutkimuslaitoksessa (CWI) Amsterdamissa Hollannissa. Ensimmäinen julkinen versio 0.9.0 julkaistiin vuoden 1991 alussa ja versio 1.0 vuonna 1994. Kielen toinen pääversio (Python 2) julkaistiin vuonna 2000, ja sen viimeisimmät versiot ovat yhä käytössä. Kolmannen pääversion (Python 3) ensimmäinen versio julkaistiin vuonna 2008. [18]

CWI:n kehitteillä ollut Amoeba-käyttöjärjestelmä oli yksi päämotivaatio Pythonin luomiselle: Amoeban järjestelmätyökalujen kehitys oli liian hidasta C-kielellä eikä shell-skripteillä ollut järjestelmän tavanomaisesta poikkeavan luonteen takia mahdollista toteuttaa tarvittavia asioita. Python on ratkaisu molempiin ongelmiin. [19]

Python 2 on jo pitkään ollut poistumassa. Sen tuen päättymisajankohta oli ensin asetettu vuoteen 2015, mutta uuteen 3-versioon siirtymisen hitautta ja vaikeutta lievittämään ajankohtaa siirrettiin eteenpäin vuoteen 2020 [20]. Hitautta selittänee se, ettei Python 2:lle kirjoitettu koodi ole täysin yhteensopivaa Python 3:n kanssa.

Python on moniparadigmainen tulkettava skriptikieli. Siinä on piirteitä sekä olio-ohjelmoinnista että funktionaalisesta ohjelmoinnista. Jotkut Python-tulkit kääntävät uuden ja muuttuneen lähdekoodin tavukoodiksi ennen sen suorittamista [21]. Koneelle tavukoodi on selkeämpää rajatun komentojoukon ja tiukan säännöllisyyden takia, joten kokonaisuuden suorituskyky kasvaa [21]. Varsinainen ohjelman suo-

```
1 import functools
2 import random
3
4 VÄLI = (1, 40)
5 MÄÄRÄ = 6
6 satunnaisluku_fn = functools.partial(random.randint, *VÄLI)
7 joukko = set()
8 while len(joukko) < MÄÄRÄ:
9     joukko.add(satunnaisluku_fn())
10
11 print("Satunnaislukuja:", ", ".join(map(str, sorted(joukko))))
```

Listaus 3.1. *Python-kielinen esimerkki, joka tulostaa konsoliin joukon satunnaislukuja. Eräs tuloste saattaisi olla "Satunnaislukuja: 7, 9, 10, 11, 19, 24".*

ritus tapahtuu Python-virtuaalikoneessa, joka suorittaa tavukoodikomentoja komento kerrallaan [21].

Listauksessa 3.1 on esimerkki Python-kielisestä ohjelmasta, joka luo kuuden numeron joukon eri satunnaislukuja lukuvälillä 1–40 ääripäät mukaanlukien. Riveillä 1–2 tuodaan käytettäväksi joitakin standardikirjaston moduuleita. Rivit 4–5 määrittävät ajoon käytetyt vakiot. Rivi 6 luo uuden parametrattoman funktion `satunnaisluku_fn`, joka kutsuttaessa kutsuu `randint`-funktiota ja antaa sille parametreina `VÄLI`-vakion arvot. Funktion kutsu vastaa funktiokutsua `random.randint(1, 40)`. Rivi 8 toistaa riviä 9 niin kauan, että joukko-muuttujassa on (ainakin) kuusi arvoa. Rivi 9 lisää `joukko`-joukkoon arvon rivillä 6 luodun funktion avulla. Tämä ei välttämättä kasvata joukon kokoa, sillä joukossa jo olevien arvojen uudelleen lisääminen ei vaikuta joukon sisältöön. Lopulta rivi 11 tulostaa joukon sisältämät numerot numerojärjestyksessä pilkuilla toisistaan erotettuina.

Virallisen Python-tulkin asennuspaketissa on mukana standardikirjastoksi kutsuttu joukko kirjastoja kielen käyttöä tukemaan. Tällaisia ovat muiden muassa käyttöjärjestelmän tiedostojenkäsittelypalvelut, lokituspalvelu, merkkijonojen hahmontaminen säännöllisillä lausekkeilla ja joidenkin tiedonvälitysmuotojen kuten JSONin ja XML:n käsittely. Näiden lisäksi on olemassa virallinen hakemisto Python Package Index eli PyPi, johon kehittäjät voivat rekisteröidä ja jakaa kehittämänsä Python-kirjastot ja -sovellukset eli paketit. Tästä hakemistosta löytyy yli 101 000 pakettia vaihteleviin tarkoituksiin [22].

3.2 JSON

JSON eli JavaScript Object Notation on Douglas Crockfordin vuonna 2002 julkaissut yksinkertainen rajoitettuun sarjallistamiseen perustuva tiedonvälitysmuoto. Se perustuu osaan JavaScript-kielen syntaksia ja on XML:ään verrattuna kevyempi vaihtoehto tiedon välitykseen. Muoto ei itsessään mahdollista rekursiivisten rakenteiden esittämistä. Tällaiset rakenteet voidaan tarvittaessa toteuttaa itse luomalla rakenteeseen vaadittava meta-rakenne ja sitä käsittelevä sovelluskoodi. [23]

Listauksessa 3.2 on esitetty JSON-muodon syntaksi. Määritelmä ei ota kantaa, miten mikäkin JSON-arvo esiintyy muotoa käyttävässä ohjelmointikielessä [24]. Syntaksissa ei oteta myöskään kantaa tyhjeisiin, mutta termien *olio* ja *taulukko* määrittelyjen osien välissä olevat tyhjeet jätetään huomiotta. JSONilla tietoa toisilleen välittävien tahojen on sovittava keskenään siitä, mitä tietoa missäkin JSON-dokumentissa välitetään [24]. Määritelmän mukainen JSON-dokumentti on myös validia JavaScript-koodia, joten JavaScript-ympäristössä dokumentin jäsentäminen muistissa olevaksi tietomalliksi voidaan helpoimmillaan toteuttaa evaluoimalla dokumenttimerkkijono [23]. Tämä on kuitenkin sellaisenaan vaarallista, sillä evaluoinnissa suoritetaan myös mahdollinen ylimääräinen haittakoodi [23]. Niinpä dokumentin muunnos olioksi kannattaa joka tapauksessa tehdä jäsentäjää käyttäen [23].

```

HEX = HEXDIG / %x61-66 ; eli 0-9 / A-F / a-f
eksponentti = ("e" / "E") ["+" / "-"] INTEGER
numero = ["-"] INTEGER ["." INTEGER] [eksponentti]
merkki = UNICODE ; mutta ei %x08, LF, %x0C, CR, %x5C, DQUOTE
pako = "\" ("b" / "n" / "f" / "r" / "\" / DQUOTE)
unicode_pako = "\u" 4HEX

merkkijono = DQUOTE *(merkki / pako / unicode_pako) DQUOTE
arvo = olio / taulukko / numero / merkkijono /
      "true" / "false" / "null"
olio = "{" [merkkijono ":" arvo *("," merkkijono ":" arvo)] "}"
taulukko = "[" [arvo *("," arvo)] "]"

json_dokumentti = arvo

```

Listaus 3.2. *JSON-muodon syntaksi ABNF-kielellä (Augmented Backus-Naur Form) kuvattuna. Mukailtu lähteistä [23, 24].*

4. ALUSTARIIPPUMATON HALLINTAJÄRJESTELMÄ

Tässä luvussa käsitellään merkkijonojen keskitettyyn hallintaan tarkoitettua työkalun kolmea iteratiivista versiota. Aliluvussa 4.1 on kuvattu merkkijonojen keskitetyn hallinnoinnin taustalla ollut ohjelmointialustalta toiselle merkkijonoja kopioiva työkalu. Varsinaista hallinnointiajastusta on mukana vasta aliluvussa 4.2 kuvatussa työkalun toisessa versiossa. Itse lopullinen työkalun versio on kuvattu laajemmin aliluvussa 4.3.

Merkkijonojen keskitetyllä hallinnalla pyritään saavuttamaan sekä säästöjä kääntämisessä että merkkijonojen yhtenäisyyttä ohjelmistoprojektin eri alustojen välillä. Säästöjen taustalla on se, että kääntämisestä saatetaan teettää ulkopuolisella taholla, joka laskuttaa työstä käännettävien sanojen lukumäärän mukaan. Samojen merkkijonojen kääntäminen useaan kertaan kasvattaisi siis kuluja turhaan.

Kaikissa työkalun iteraatioissa on käytetty ytimen toteutukseen Python-kielen pääversiota 3. Kieli on alun perin valittu kattavan standardikirjaston, vähäisten ulkoisten riippuvuuksien, käyttöjärjestelmästä riippumattomuuden ja skriptiluonteen takia. Käyttöä helpottavien komentojen toteutukseen on käytetty lisäksi Bash-skriptejä.

4.1 Ensimmäinen versio

Pääajatus ensimmäisessä versiossa oli saada käyttöön yksinkertainen ratkaisu iOS-merkkijonojen käyttöönottamiseksi Androidilla, sillä kohdeprojekti oli luonteeltaan iOS-sovelluksen muunnos Android-ympäristössä käytettäväksi. Koska Android ei välitä ylimääräisistä attribuuteista merkkijonojen XML-dokumenteissa, kutakin merkkijonoa vastaavan iOS-merkkijonon tunniste voitiin merkitä uuteen attribuuttiin. Tunnisteiden yhdistäminen oli tehtävä, sillä Androidin tunnistenimien rajoitukset ovat tiukemmat kuin iOS:ssä. Idean pohjalta rakennettiin työkalu, joka jäseni iOS-merkkijonotiedostosta merkkijonojen tunnisteet arvoineen ja sijoitti ne luotujen ris-tiiviittausten avulla Android-kohdetiedostoon.

Viittaukset Androidin merkkijonojen lähteistä tarvitsi merkitä vain pääkielen sisältävään tiedostoon taulukon 4.1 esimerkin mukaisesti. Tämän haittapuolena työkalu joutui lukemaan tiedoston jokaisen muun kielen tietojen tuontia varten referenssiksi. Viittausten avulla työkalu pystyi tarkistamaan, että kaikille viitatuille merkkijonoille löytyi lähdemateriaali ja että lähdemateriaalissa ei ollut viittaamattomia, kohdeesta puuttuvia merkkijonoja. Viittauksia oli mahdollista merkitä sekä tavallisiin `<string>`-merkkijonoihin että `<string-array>`-taulukon alkioihin. Työkalu ei varoittanut puutteesta, mikäli elementtiin oli merkitty vihje, ettei elementtiin ole tarkoitus yhdistää tekstiä iOS-lähteestä. Lähdetiedostossa olevat kommentit tulkittiin merkkijonojen kategoriointiin liittyviksi kommentteiksi ja kopioitiin sellaisenaan vastaavaan paikkaan kohdetiedostoon. Kopioinnin yhteydessä iOS:n merkkijonojen korvaukset jäsennettiin, niihin lisättiin järjestysnumerot ja ne muutettiin Androidille sopiviksi. Lisäksi tarkistettiin, että lähdemerkkijonon korvausten määrä ja muoto yhtenivät pääkielen merkkijonon korvausten kanssa.

Itse ytimen lisäksi työkalun suorittamista varten oli pieni komentojonotiedosto, johon konfiguroitiin kielikoodit kaikista kopioitavista kielistä sekä ajoon tarvittavia perustietoja, kuten lähde- ja kohdetiedostojen hakemistopolut. Näiden avulla varsinaisen työn tekevä ydin käynnistettiin sopivilla parametreilla jokaista kieltä varten erikseen. Määritellyistä kielikoodeista rakennettiin myös XML-muotoiset listat kielikoodeista ja maiden lippujen kuvaresurssien nimistä, joiden avulla Android-sovellus pystyi esittämään käyttäjälle listan valittavista kielistä. Uusi tekstimateriaali oli nyt mahdollista tuoda Android-projektiin yksinkertaisesti ajamalla kyseinen komentojono. Merkkijonot päivittyivät automaattisesti ja uuden kielen kokeilu käytössä vaati vain sovelluksen kielivalinnan asettamisen.

Rajoittuneen viittaustavan takia tämä versio ei kyennyt kunnolliseen monikkomuotojen esitykseen eikä yhdistämiseen. Merkkijonotaulukoiden käyttö oli myös kankeaa, sillä jokaiselle taulukon alkiolle erikseen täytyi määritellä viittaus vastaavaan iOS-merkkijonoon. Yksittäiselle merkkijonolle ei ollut mahdollista määrittää kommenttia eikä kontekstia. Joitakin merkkijonoja jouduttiin kuitenkin kääntämään

Taulukko 4.1. Merkkijonojen yhdistäminen työkalun ensimmäisessä versiossa.

Lähde / iOS-version merkkijonot

```
Englanti  "sum" = "Total %@";
Suomi    "sum" = "Yhteensä %@";
```

Android-version merkkijonot ennen työkalun ajoa

```
Englanti  <string name="general.sum" ios="sum">Sum %s</string>
Suomi    <string name="general.sum">Summa %s</string>
```

Android-version merkkijonot työkalun ajon jälkeen

```
Englanti  <string name="general.sum" ios="sum">Total %1$s</string>
Suomi    <string name="general.sum">Yhteensä %1$s</string>
```

erikseen ympäristösidonnaisuuksien takia. Tällaisia ovat esimerkiksi käyttöjärjestelmän käyttöoikeuksien hallintaan liittyvät merkkijonot. Yksisuuntaisen muunnosprosessin takia uusien merkkijonojen tuominen Android-projektiin vaati käytännössä merkkijonon tuonnin ensin iOS-projektiin. Tämä tarkoitti myös sitä, että Android-projektin oli aina oltava iOS-projektia jäljessä. Nopean ratkaisun kehityksen ja tarkemman suunnitteleamattomuuden takia työkalun lähdekoodi ei ollut laajennettava. Koodia työkaluun kertyi 479 riviä ja komentojonoon 102 riviä.

4.2 Merkkijonojen keskitetyn hallinnan kokeilu

Alustamuunnosprojektiin erikoistettu työkalu ei sovellu yleiseen käyttötapaukseen, jossa tarve uusille merkkijonoille voi tulla miltä tahansa kohdealustalta. Siksi päätettiin luoda keskitetty tietokanta, josta merkkijonot voidaan viedä eri alustoille niiden omissa formaateissa. Tietokanta on tekstimuotoinen ja tekstieditorilla muokattava, sillä tämä mahdollistaa nopean aloituksen työkalun kehittämisessä eikä käyttöliittymää tai sen toteutusta tarvitse miettiä. Tekstimuoto mahdollistaa lisäksi tiedoston tallettamisen versionhallintaan ja muutosten seuraamisen versionhallintatyökalun tarjoamin menetelmin.

Tietokannan tallennusmuodoksi valittiin JSON sen kevyen syntaksin ja helpon ohjelmallisen käsittelyn takia. Muotoon suunniteltiin merkkijonojen ryhmittely käyttöpaikan mukaan helpottamaan organisointia. Kullekin merkkijonolle ja merkkijonoryhmälle oli mahdollista määrittää kommentti kuvaamaan kääntäjälle merkkijonojen kontekstia. Merkkijonojen korvausten muodoksi valittiin yleinen printf-muoto (katso esimerkki sivulla 9 listauksessa 2.1). Työkalu muuntaa korvaukset iOS-yhteensopiviksi viettäessä merkkijonoja iOS:n resurssitiedostoihin. Listauksessa 4.1 on esitettyä tietokannan perusmuoto kahdella merkkijonolla ja kahdella kielellä.

Alun käyttöönoton jälkeen tietokantaan lisättiin tieto siitä, mille alustoille mikäkin merkkijono on tarkoitus viedä, sillä eri alustojen ominaisuuksien kehittäminen ei ollut keskenään yhtä nopeaa eikä ominaisuuksia välttämättä toteutettu samassa järjestyksessä. Merkintä mahdollisti samalla merkkijonon avaimen uudelleennimeämisen alustakohtaisesti. Jos merkinnästä puuttui jokin alusta tarkoitti se, ettei merkkijonoa kuulu viedä kyseisen alustan resurssitiedostoon. Yleinen tietokannan muoto on esitetty listauksessa 4.2 mukailla ABNF-notaatiolla, jossa assosiaatiot on merkitty kaksoispisteen avulla, avain kaksoispisteen vasemmalla ja arvo oikealla puolella. ABNF ja tässä työssä käytetty mukautus on kuvattu lyhyesti liitteessä A. Listauksessa 4.2 esitetty muoto on esitetty myös ratapihakaaviona liitteessä B.

```

{
  "group-all": {
    "strings": {
      "general.sum": {
        "comment": "%s = rahasumma",
        "fi_FI": "Yhteensä %s",
        "en_US": "Total %s"
      },
      "authentication.login": {
        "fi_FI": "Kirjaudu sisään",
        "en_US": "Log in"
      }
    }
  }
}

```

Listaus 4.1. *Esimerkki tietokannan ensimmäisestä versiosta.*

Tietokanta on rajoitetusti hierarkkinen, sillä sen määrittelyssä ei ole rekursiota. *Ryhmät* voivat sisältää *kommentin* ja *merkkijonoryhmän*, joka sisältää vapaavalintaisen määrän varsinaisia *merkkijono-olioita*. Vasta merkkijono-olio sisältää itse merkkijonon käännöksineen ja merkinnät alustoista, joihin merkkijono halutaan liittää. Tietokannan juurena toimii olio, jolle voidaan määritellä kommentti, merkkijonoryhmä ja vapaavalintainen määrä varsinaisia ryhmiä. Juuressa olevat ryhmät erottuvat kommentti- ja merkkijonoryhmä-attribuuttien avaimesta vain ryhmän nimessä ole-

```

merkkijono = *UNICODE
locale = 2LCASE "_" 2UCASE
alusta_id = 1*UNICODE ; mutta ei "comment" eikä locale.
merkkijono_id = 1*UNICODE
ryhmä_id = "group-" 1*UNICODE

kommentti = "comment": merkkijono
alusta_alias = alusta_id: merkkijono
käännös = locale: merkkijono

merkkijono_olio =
  merkkijono_id: [kommentti] *alusta_alias *käännös
merkkijono_ryhmä = "strings": *merkkijono_olio

ryhmä = ryhmä_id: [kommentti] [merkkijono_ryhmä]
tietokanta = [kommentti] *ryhmä [merkkijono_ryhmä]

```

Listaus 4.2. *Ensimmäisen tietokantaversioiden määrittely mukailussa ABNF-muodossa.*

van group--etuliitteen avulla. Koska varsinaisessa merkkijono-oliossa kommentti-, alusta- ja kieliattribuutit ovat samassa nimiavaruudessa, on mahdollisista alustojen nimistä rajattu pois joitakin vaihtoehtoja.

Osa jatkokehityksestä perustui työkalun ensimmäisen version lähdekoodiin, mutta työkalun sisäinen tietomalli toteutettiin uudelleen. Merkkijonot ositettiin tavalliseen tekstiin ja korvauksiin. Tämä mahdollisti korvausten tarkemman uudelleentulinnan, jolloin iOS:n tekstikorvausten erikoistapaus (katso aliluku 2.4.2 sivulla 10) jouduttiin käsittelemään vain viettäessä merkkijonoja iOS:n tiedostoihin.

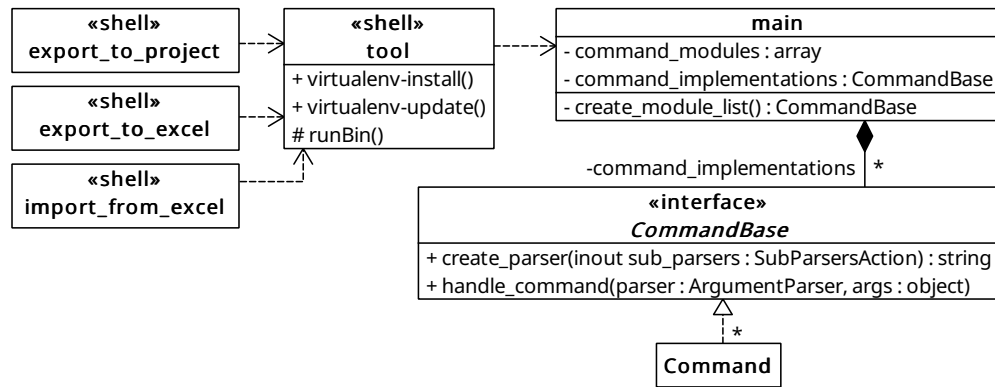
Päivittäiseen käyttöön oli työkalua varten edellisen version tavoin erillinen komentojonotiedosto, johon käyttäjän projektihakemisto ja viettävät kielet tiedostoniminen oli konfiguroitu. Työkalussa oli kuitenkin nyt parempi komentorivin käsittely, mikä mahdollisti työkalun monipuolisemman käytön ja komentoriviohjeen.

Ensimmäiseen versioon verrattuna tällä ratkaisulla kaksialustaisen projektin merkkijonojen tallettaminen ja yhdessä pitäminen oli mahdollista toisistaan riippumattomasti. Toteutus kärsi vieläkin ensimmäisestä ratkaisusta tuotujen osien kómpe-lyydestä, mutta yksinkertaisen tiedostomuodon muunnoksen tekeminen oli nyt verrattain helppoa ja mahdollisti merkkijonojen käsittelyn muunnoksen yhteydessä. Koodia kertyi työkaluun 1 387 riviä ja komentojonoon 24 riviä. Edelliseen versioon verrattuna muutos työkalun rivimäärässä oli +190 % ja -76 %.

4.3 Tehty järjestelmä

Tietokannan rakenne pohjautui toisessa versiossa olleeseen tietokantaan, mutta sitä tarkennettiin ja laajennettiin. Merkkijonoryhmiin lisättiin mahdollisuus sisältää sisäkkäisiä ryhmiä ja niiden määrittelystä tehtiin tarkempi. Tämän seurauksena rakenteeseen voitiin määritellä myös ryhmän kaltainen juuriolio, jossa ryhmän muiden ominaisuuksien lisäksi on myös ympäristön yleisiä konfiguraatietoja ryhmiteltynä *alustoiksi*. Näiden tietojen avulla työkalu voi päätellä, mitä merkkijonoja kuhunkin tulostiedostoon on tarkoitus kirjoittaa.

Työkalun komentorivin käsittely jaettiin osiin siten, että jokainen osa voisi olla itsenäinen oma toimintonsa. Näin työkalun käytön ei tarvitse rajoittua vain yhteen tiettyyn käyttötarkoitukseen ja jokaiselle toiminnolle voidaan määritellä omat parametrit ja ohjeensa. Kuvassa 4.1 on esitetty työkalun ajoon liittyvät komentojonot ja luokat. Kaikki työkalun tarjoamat komentototeutukset on periytetty CommandBase-luokasta. Työkalussa on komennot alustakohtaisten tiedostojen käsittelyyn sekä kokonaisvaltaisen merkkijonotiedon vientiin ja tuontiin taulukkolaskentaan käytettyjen



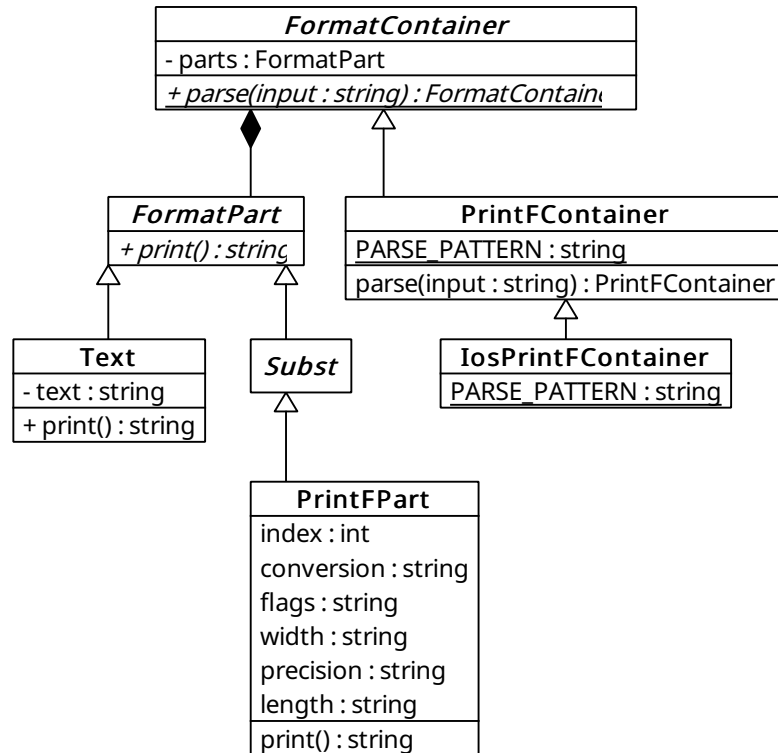
Kuva 4.1. Työkalun komentorivin käsittelyn luokkakaavio komentojonotiedostojen kanssa.

xlsx-tiedostojen avulla. Lisäksi työkalussa on komento, jolla CLDR-tietokannasta (Common Locale Data Repository) löytyvät monikkojen kielikohtaiset luokittelutiedot muunnetaan työkalulle yksinkertaisempaan muotoon.

Työkalun käyttöön on komentojonotiedostoja, joiden on tarkoitus helpottaa työkalun käyttöä. Päivittäistä käännöstiedon vientiä varten on komentojonotiedosto `export_to_project.sh`. Hieman harvemmin tarvittuna kielellistä käännöstä varten käytettyjen taulukkolaskentatiedostojen käsittelyyn on komentojonotiedostot `export_to_excel.sh` ja `import_from_excel.sh`. Lisäksi `tool.sh` on käytettävissä muihin tietokannan käsittelytoimiin. Näiden lisäksi helpotettua käännöstiedon vientiä varten käyttäjä voi luoda itselleen asetustiedoston, johon merkitään haluttujen alustaprojektien hakemistopolku.

Työkalun ydintä lähin osa on `tool.sh`-kutsukomento. Se huolehtii työkalun vaatiman virtuaaliympäristön asentamisesta, päivittämisestä ja käynnistyksestä. Yksinkertaistettuna komentojono ajaa komennon `venv/bin/python -m localizator.main` ja välittää annetut parametrit työkalulle. Komento on käytettävissä myös muista komentojonoista ja tarjoaa työkalun käynnistyspalvelun `runBin`-komentona.

Ulompi komentojono `export_to_project.sh` lukee kehittäjäkohtaiset asetukset `project_conf.sh`-tiedostosta ja luo asetusten ja skriptiin merkittyjen kielten perusteella työkalulle annettavan CSV-tiedoston kaltaisen työjonon. Komento voi tuottaa käännöstiedon yhdelle tai useammalle alustaprojektille yhden ajon aikana asetusten määrittysten mukaisesti. Yksinkertaistettuna komentojono ajaa yhden tai useamman kerran työkalua `tool.sh` avulla: `runBin export-multi tietokanta.json --file työjono.tmp --base projektihakemisto`. Työjonossa on kullakin rivillä yhden vietävän käännöstiedoston vientimäärittäminen, josta



Kuva 4.3. Merkkijonoarvojen luokkakaavio.

tenkin käyttöprojektin kehittäjien harteille, ja voi olla kyseenalaista, onko sellaisen luominen ja ylläpito kannattavaa verrattuna lineaarisemmän rakenteen käyttöön.

Merkkijonoalkio sisältää vähintään alkion nimen. Lisäksi alkiossa voi olla kommentti, alkioon kuuluvat käännökset, merkinnät mihin alustoihin merkkijono kuuluu sekä mahdolliset alustakohtaiset nimet. Alkion sisältämät käännökset ovat yksinkertaisten merkkijonon sijaan eri tyyppisiä `ValueBase`-luokasta periyettyjä arvoja, joiden avulla voidaan esittää sekä yksittäiset `Value`- että monikkomuotoiset `Plural`-merkkijonot. Tämä abstraktio mahdollistaa myös jatkokehityksen mahdolliset merkkijonotaulukot ja monimuotoisemmat monikkomerkkijonot.

Koska ytimellä on mahdollista käsitellä useiden eri sovellusalustojen merkkijonoja, varsinainen merkkijonotieto on edelleen abstrahoitu pois `Value`- ja `Plural`-luokista tarkemmin jäsennettyihin `FormatContainer`-luokkiin. Jäsentämiseen liittyvät luokat ovat kuvattu kuvan 4.3 luokkakaaviossa.

`FormatContainer`-luokan instanssit kuvaavat kokonaisia merkkijonoja yhden tai useamman `FormatPart`-luokan instanssin muodostaman listan avulla. Tavalliset yhden tai useamman merkin mittaiset merkkijonot esiintyvät listassa `Text`-luokan instansseina. Tekstikorvaukset esiintyvät `Subst`-luokasta periyettyinä instansseina, joiden tyyppi riippuu ne sisältävän `FormatContainer`-luokan tyypistä.

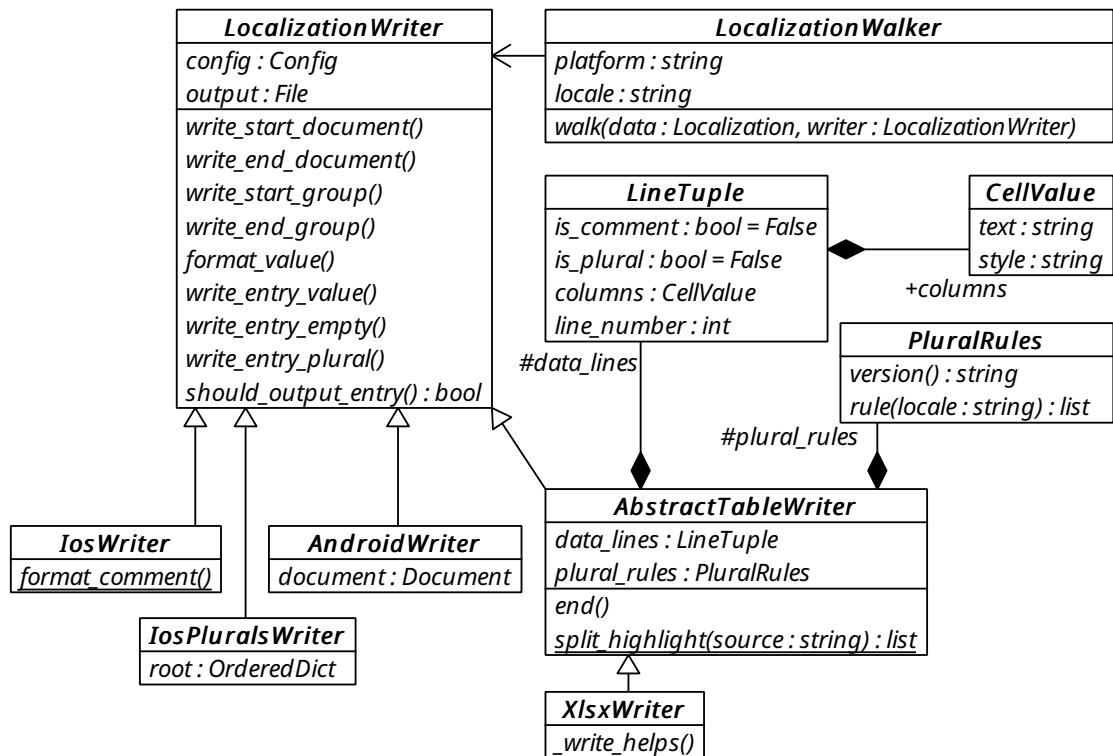
Taulukko 4.2. Merkkijonon esitys `FormatContainer`-luokan instanssina.

Merkkijono:	"Kirjaudu sisään"
	<code>PrintfContainer(parts=[...])</code>
<code>parts[0]</code>	<code>Text("Kirjaudu sisään")</code>
<hr/>	
Merkkijono:	"%s ei ole saatavilla."
Esimerkkisyöte:	("Ämpäri")
Esimerkkitulo:	Ämpäri ei ole saatavilla.
	<code>PrintfContainer(parts=[...])</code>
<code>parts[0]</code>	<code>PrintfPart(index=1, spec="s", src="%s")</code>
<code>parts[1]</code>	<code>Text(" ei ole saatavilla.")</code>
<hr/>	
Merkkijono:	"klo %1\$02d-%2\$02d"
Esimerkkisyöte:	(8, 22)
Esimerkkitulo:	klo 08-22
	<code>PrintfContainer(parts=[...])</code>
<code>parts[0]</code>	<code>Text("klo ")</code>
<code>parts[1]</code>	<code>PrintfPart(index=1, spec="d", width=2, flags="0", src="%1\$02d")</code>
<code>parts[2]</code>	<code>Text("-")</code>
<code>parts[3]</code>	<code>PrintfPart(index=2, spec="d", width=2, flags="0", src="%2\$02d")</code>
<hr/>	
Merkkijono:	"+ %3\$d x %1\$s (%2\$s)"
Esimerkkisyöte:	("Ämpäri", "1 €", 2)
Esimerkkitulo:	+ 2 x Ämpäri (1 €)
	<code>PrintfContainer(parts=[...])</code>
<code>parts[0]</code>	<code>Text("+ ")</code>
<code>parts[1]</code>	<code>PrintfPart(index=3, spec="d", src="%3\$d")</code>
<code>parts[2]</code>	<code>Text(" x ")</code>
<code>parts[3]</code>	<code>PrintfPart(index=1, spec="s", src="%1\$s")</code>
<code>parts[4]</code>	<code>Text(" (")</code>
<code>parts[5]</code>	<code>PrintfPart(index=2, spec="s", src="%2\$s")</code>
<code>parts[6]</code>	<code>Text(" ")</code>

Työkalun sisäisenä merkkijonojen esitysmuotona käytetään `PrintfContainer`-luokkaa. Merkkijonosyöte jäsennetään staattisen `parse`-luokkametodin avulla `PrintfContainer`-luokan instanssiksi. Mukana oleva `IosPrintfContainer` on jäsennykseen käytettyä säännöllistä lauseketta lukuun ottamatta identtinen kantaluokkansa kanssa. `PrintfContainer`-tyyppisten merkkijonojen korvausten tyyppi on `PrintfPart`. Taulukossa 4.2 on esimerkkejä merkkijonojen esityksestä järjestelmässä.

4.3.2 Merkkijonojen vienti

Alustakohtaisia käänöstiedostoja voidaan luoda työkalun avulla viemällä merkkijonotietoa joko yhteen tai useampaan tiedostoon kerrallaan. Vientiprosessin vaiheissa ensimmäisenä luetaan tietokanta muistiin aliluvussa 4.3.1 kuvattuun tietorakentee-



Kuva 4.4. Merkkijonojen vientiin liittyvien luokkien luokkakaavio.

seen. Toisena vaiheena tietokannasta kirjoitetaan haluttu osuus alustalle määritellyssä tiedostomuodossa. Kirjoitusvaiheessa päätellään tietokannan alusta- ja käännösmerkintöjen perusteella, kirjoitetaanko jotakin merkkijonoryhmää ollenkaan tulostiedostoon. Näin kohdealustan näkökulmasta tyhjiä merkkijonoryhmiä tai niiden kommentteja ei tiedostosta tarpeettomasti löydy. Kirjoitettaessa useampaa tiedostoa kerrallaan selvitetään alussa lisäksi, mitkä alusta- ja kieliparit halutaan kirjoittaa mihinkin tiedostoihin. Muistiin luettua tietokantaa käytetään tämän jälkeen kaikille kirjoituskohteille, mikä lyhentää tietokannan lukemiseen kuluva merkittävää aikaa.

Vientiin liittyvien luokkien luokkakaavio on esitetty kuvassa 4.4. Tuloksen muodostaminen tapahtuu hierarkian käsittelevän `LocalizationWalker`-luokan ja varsinaisen tiedostoon kirjoittavien `LocalizationWriter`-luokasta periyettyjen luokkien avulla. `LocalizationWalker` käy tietokantaa lävitse syvyyteen ensin -periaatteella säilyttäen siten tietokannan hierarkkisyyden ja merkkijonojen keskinäisen järjestyksen tulostiedostoon asti. Läpikäynnin aikana olio päättää merkkijonokohtaisesti sen alustamerkintöjen ja kirjoitusolion määritysten perusteella, tarvitseeko merkkijonoa ottaa tulokseen mukaan.

`LocalizationWriter`-kantaluokasta periytyvät luokat toteuttavat konkreettisen osuuden tulostiedoston osien kirjoittamisesta. `LocalizationWalker`-luokasta kutsutaan sen metodeja tietokannan rakenteen mukaan. Rakennetta käsitteleviä me-

Taulukko 4.3. *Esimerkki viivästetystä tiedostoon kirjoittamisesta. Kirjoitettavaan tiedostoon ollaan viemässä android-alustan merkkijonoja, joita esimerkissä on vain yksi. Syvyyteen ensin -läpikäyntimenetelmän perusteella tietokantaa käydään lävitse "rivi kerrallaan". Sekä tietokannan että tiedoston sisältö on yksinkertaistettu. Rivillä 7 tiedostoon ei vielä ole kirjoitettu ryhmän loppua.*

	Rakenne tietokannassa	Pino rivin jälkeen	Tiedosto rivin jälkeen
1	ryhmä-a {	ryhmä-a	-
2	ryhmä-b {	ryhmä-a, ryhmä-b	-
3	merkkijono-b (ios)	ryhmä-a, ryhmä-b	-
4	}	ryhmä-a	-
5	}	-	-
6	ryhmä-c {	ryhmä-c	-
7	merkkijono-c (android,ios)	-	ryhmä-c { merkkijono-c
8	}	-	ryhmä-c { merkkijono-c }

todeja ovat dokumentin alku ja loppu sekä ryhmän alku ja loppu. Loput metodit käsittelevät yksittäistä merkkijonoa eri tasoilla.

Tyhjien ryhmien kirjoittamatta jättämistä varten `LocalizationWalker`-luokassa on ryhmien alkumerkintöjä varten pino, johon lisätään hierarkian mukaisesti sisäkkäisten ryhmien alkumerkinnät. Pinoa käyttämällä ryhmien alkujen kirjoitusta viivästetään siihen asti, että alkumerkinnät on pakko kirjoittaa merkkijonotiedon kirjoittamisen takia. Mikäli hierarkian sisimmässä ryhmässä ei ole ainuttakaan kirjoitettavaa merkkijonoa eikä aliryhmää, palataan hierarkiassa ylöspäin ja pudotetaan pinon päällimmäinen alkumerkintä pois, kunnes joko päädytään ryhmään, jota ei vielä ole kokonaan käsitelty, tai tietokannassa ei ole enempää käsiteltävää. Jos ryhmässä on mukaan otettavia merkkijonoja, pinon sisältö kirjoitetaan tulokseen poistamalla toistuvasti alin alkio ja kutsumalla kirjoitusolion ryhmän aloitusfunktiota alkion tiedoilla, kunnes pino on käyty läpi. Taulukossa 4.3 on esimerkki, kuinka pino toimii yksinkertaisesta tietokannasta viettäessä merkkijonoja tiedostoon.

Merkkijonojen vientiin `LocalizationWriter`-luokassa on omat funktionsa sekä yksittäisille että monikkomuotoisille merkkijonoille. Molempien funktioiden parametrina ovat merkkijonot on etukäteen muunnettu `format_value`-funktiolla tietokannassa olevan `PrintFContainer`-olion kohteena olevan tiedostomuodon mukaiseen muotoon. Esimerkiksi `IOSWriter` tuottaa tässä funktiossa uuden `IOSPrintFContainer`-instanssin. Lisäksi luokka voi ilmaista läpikävijälle, onko kirjoitusolion mielestä merkkijonoa tarpeen kirjoittaa tulostiedostoon. Tätä ominaisuutta käyttämällä on toteutettu iOS:n erilliset tiedostot yksikkö- ja monikkomuotoisille merkkijonoille.

Androidille työkalu tuottaa XML-muotoisia merkkijonotiedostoja `AndroidWriter`-luokan avulla. Luokka tuottaa aiempaa sivulla 9 olevaa listausta 2.1 mukailevan tuloksen Pythonin `xml.dom.minidom`-kirjaston avulla. Merkkijonoja muokataan muodon vaatimilla tavoilla, jotta ne olisivat kohdesovelluksessa siinä tarkoitetussa muodossa kuin ne tietokantaan on kirjoitettu. Tulosta mukauttavia alustalle merkittäviä asetuksia ovat *withQuotes*, *inlineComments*, *indentNestedGroups* ja *indent*. Näistä ensimmäisellä voi määritellä, kirjoitetaanko kaikkien merkkijonojen arvot tulokseen lainausmerkkeihin vai ei. Toisen asetuksen käyttäminen siirtää merkkijonokohtaiset kommentit kirjoitettavaksi XML-elementin attribuuttiin. Mikäli asetus ei ole käytössä, elementin kommentti muodostaa tavallisen XML-kommentin. Ryhmän kommentti voi muodostaa vain XML-kommentin, sillä Androidin merkkijonotiedostossa ei ole erillistä ryhmän käsitettä, johon kommentin voisi liittää. Kaksi viimeistä asetusta hallinnoi tietokannan rakenteen esittämistä tulostiedostossa. Visuaalisena lisänä voidaan asettaa *indentNestedGroups* käyttöön, jolloin aliryhmä sisennetään *indent*-asetuksen osoittaman määrän välilyöntejä edelliseen sisennykseen verrattuna.

Käännöstiedostojen tuottaminen iOS-alustalle tapahtuu `IosWriter`- ja `IosPluralsWriter`-luokkien avulla. Ensimmäinen näistä tuottaa sivulla 10 olevan listauksen 2.2 mukaisen tiedoston yksikkömuotoisista merkkijonoista. Tietokannan merkkijonot kirjoitetaan tiedostoon suoraviivaisesti yksi merkkijono riviä kohden. Merkkijonolle määritelty kommentti kirjoitetaan merkkijonotiedon perään komentti-merkinnällä erotettuna. Ryhmät erottuvat tiedostossa tyhjällä rivillä ja mahdollinen ryhmän kommentti sijoitetaan ennen ryhmän merkkijonoja. `IosPluralWriter` vastaa monikkojen vaatiman *plist*-muotoisen tiedoston tuottamisesta. Esimerkki tiedoston sisällöstä on nähtävillä sivulla 11 listauksessa 2.3. Luokka muodostaa monikkomerkkijonoista ensin sisäisen tulostiedostoa vastaavan avain-arvo-tietorakenteen, joka lopuksi kirjoitetaan tiedostoksi Pythonin mukana olevan **plistlib**-kirjaston avulla.

Kielikäännöstä varten työkalulla voidaan tuottaa xlsx-muotoinen tiedosto taulukkolaskentasovelluksella muokattavaksi. Xlsx-muunnoksia varten tietokantaan on luotava erillinen näennäisalusta, johon sisällytetään kaikki tietokannasta käännettäväksi tarkoitettavien merkkijonojen alustat. Tietokannasta muodostetaan ensin välitulos `AbstractTableWriter`-luokan avulla, jossa riveinä ovat kommentit ja yksittäiset merkkijonot ja sarakkeina merkkijonot yksilöivän tietosarakkeen jälkeen valitut kielet. Monikkomuotoiset merkkijonot laajennetaan kuudeksi riviksi, yksi rivi kutakin mahdollista monikkumuotoa varten. `AbstractTableWriter`-luokasta on periytetty konkreettinen taulukkomuotoon kirjoitettava luokka `XlsxWriter`, jonka tehtävänä on tuottaa lopullinen xlsx-muotoinen tiedosto **xlsxwriter**-kirjaston avulla.

la välitulokseen kerätystä tiedosta. Xlsx-tiedostoon lisätään myös muodoltaan yksinkertainen vakio-ohjeteksti opastamaan taulukon käyttöä käännöstyössä. Kullekin ohjeen käännökselle luodaan tiedostoon oma taulukko.

Välituloksen `AbstractTableWriter`-luokasta voitaisiin periyttää myös esimerkiksi CSV-tiedostoon kirjoittava versio, mutta muodon rajoittuneisuuden takia sellaista ei ole toteutettu. Xlsx-muotoiseen tiedostoon on lisätty muokkausta helpottavia taulukkolaskennan ominaisuuksia. Kääntämättömät merkkijonot ovat korostettu solun sisältöön perustuvalla automaattisella taustavärillä. Taulukossa muokattavia soluja ovat vain ne, joissa on käännettävää tai käännettyä merkkijonotietoa. Erityisesti monikkomuotoisten merkkijonojen kullekin kielelle epäoleelliset solut ovat lukittu muokkaukselta ja merkitty harmaalla taustavärillä. Solujen lukitseminen estää taulukossa olevien tietokannan kommenttien ja merkkijonojen avaimien vahinkomuutokset. Lisäksi merkkijonojen kommenttikentissä olevat erityiset asteriskeilla merkityt alueet korostuvat punaisella. Tarpeettomat monikot selvitetään CLDR-tietokannasta käsitellyn kielikohtaisen tiedon perusteella. Eri kieliversioiden vertailu ja hyödyntäminen käännöstyössä on mahdollista, sillä merkkijonon kaikki mukaan otetut versiot ovat taulukossa rinnakkain.

Javan `properties`-tiedostojen muodostukseen käytetään **jproperties3**-kirjastoa ja `PropertiesWriter`-luokkaa. Tiedostomuoto on yksinkertainen avain–arvo-pareja sisältävä tekstitiedosto. Muodossa ei ole mahdollista esitellä merkistököodausta vaan se vaatii, että tiedosto on koodattu ISO-8859-1-merkistöllä, tutummin läntisellä merkistöllä. Tämän rajoitteen takia merkistöön kuulumattomat merkit koodataan pakomerkinä `\uNNNN` jossa `NNNN` sisältää neljä heksadesimaalista numeroa, jotka muodostavat osoituksen Unicode-merkistöstandardin koodipisteeseen. Koodausta varten Pythonin **codecs**-kirjastoon rekisteröidään merkistöön sopimattomien merkkien virheenkäsittelijä, joka muodostaa merkin pakomerkinä UTF-16-koodauksen mukaisesti. Osa virheenkäsittelijästä on esitetty listauksessa 4.3. Näin esimerkiksi euro-merkki `U+20AC` koodautuisi merkkijonoksi `\u20ac` ja syntymäpäiväkakkumuoto `U+1F382` merkkijonoksi `\ud83c\udf82`. Tämän lisäksi Javan `MessageFormat`-luokan toiminnallisuuden takia merkit `'{'` ja `'}'` sisältävistä merkkijonoista kahdennetaan heittomerkit. `Properties`-tiedostojen vientiin ei ole toteutettu monikkomerkkijonojen tukea.

4.3.3 Käännöstiedon tuonti

Työkalussa on toteutettu lähes kaikille tuetuille vientimuodoille myös luokat vastaavien muotojen lukemista varten. Lukijoiden kantaluokka `Reader` määrittelee vain

```

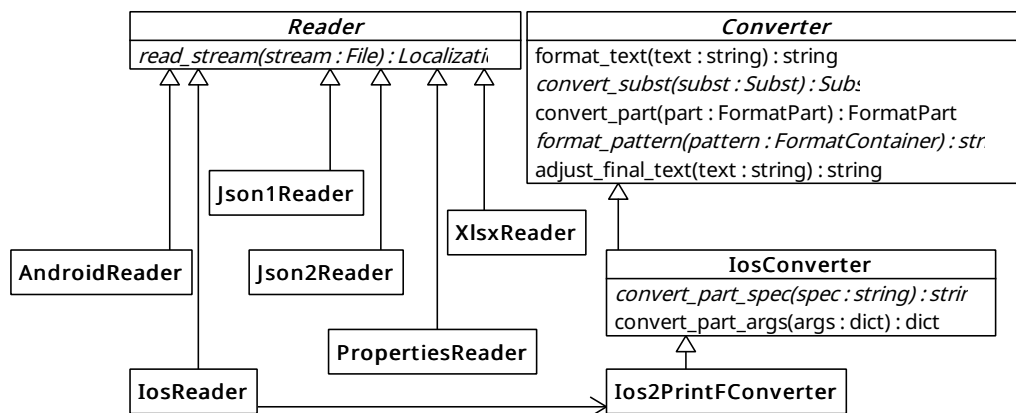
char_value = ord(non_encodable_character)
if char_value < 0x10000:
    return "\\u%04x" % char_value
if char_value <= 0x10ffff:
    U = char_value - 0x10000
    W1 = 0xd800 | (U >> 10)
    W2 = 0xdc00 | (U & (2**10 - 1))
    return "\\u%04x\\u%04x" % (W1, W2)

```

Listaus 4.3. Merkin koodaaminen UTF-16-muotoon pakomerkinällä RFC-2781 kohdan 2.1 mukaisesti [25].

yhden vaaditun metodin, jonka on tarkoitus tuottaa uusi `Localization`-olio luki-
malla annettua tiedostoa. Tiedostojen lukemiseen käytettyjen luokkien luokkaka-
vio on esitetty kuvan 4.5 kaaviossa. Tuotetun tietokannan merkkijonoissa olevien
korvauksien tulee olla *printf*-muodossa mahdollisten muotomuunnosten toimivuu-
den takaamiseksi. Työkalussa olevat toiminnot käsittelevät vain `Reader`-toteutusten
avulla tuotettuja `Localization`-luokan olioita.

Androidin XML-tiedostojen luku on toteutettu `xml.dom.minidom`-kirjaston avul-
la. Kirjasto toteuttaa DOM-rajapinnan (Document Object Model) minimaalisesti.
Rajapinnan toimintaperiaatteeseen kuuluu, että tiedosto luetaan kokonaisuudessaan
ja siitä tuotetaan käsiteltävä dokumenttiolio. Dokumentissa olevia elementtejä ja
attribuutteja voidaan käsitellä vapaa-valintaisessa järjestyksessä. Androidin merk-
kijonotiedostossa olevat elementit ovat hierarkialtaan joko lapsettomia tai sisältävät
yhden jälkeläistason. Päätasolla lukija päättää elementin nimen perusteella käytet-
tävän jatkojäsennyksen tavan: yksittäinen merkkijono tai monikkomuotoinen merk-
kijono. Tulostietokantaan luetut merkkijonot on sijoitettu hierarkian päätasolle.



Kuva 4.5. Merkkijonojen eri tiedostomuodoista lukemiseen ja `Localization`-
tietokannaksi muodostamiseen liittyvien luokkien luokkakaavio.

```

1  ^ \s* "
2  (?P<key> [^"] +? )
3  " \s*
4  = \s* "
5  (?P<val> .*? )
6  " \s* ;
7  \s*
8  (?: // \s* (?P<comment> .* ) )?

```

Listaus 4.4. *Strings-tiedostojen jäsentämiseen käytetty säännöllinen lauseke väljennetyssä muodossa.*

Listauksessa 4.4 esitettyä säännöllistä lauseketta on käytetty jäsentämään iOS:n strings-tiedostoja niiden vähäisen hierarkian takia. Lauseke tuottaa rivikohtaisesti avaimen, merkkijonon arvon ja kommentin. Lisäksi algoritmi yrittää päätellä C-tyylisten kommenttien alku- ja loppumerkkien perusteella, onko luettava materiaali kommentoitu pois vai ei. Koska iOS käyttää merkkijonojen korvauksissa merkintää %@ perinteisen %s-merkinnän sijaan, luettujen merkkijonojen korvaukset muunnetaan `Ios2PrintfConverter`-luokan avulla oikeaan muotoon.

Jäsennyksen säännöllinen lauseke on toteutettu käyttämällä Pythonin mukana olevaa säännöllisten lausekkeiden kirjastoa. Lausekkeeseen sijoitetut useat `\s*`-osat poistavat jäsennettävästä syötteestä mahdolliset merkityksettömät tyhjeet. Rivien 1–3 sisältö poimii syöteriviltä merkkijonolle lainausmerkkien väliin asetetun avaimen tulokseen nimelle `key`. Rivit 4–6 poimivat tulosmerkkijonolle arvon niin ikään lainausmerkkien välistä ja asettavat arvon tulokseen nimelle `val`. Lopulta rivi 8 poimii syöterivin lopusta valinnaisen `//`-merkkien jälkeen olevan kommentin tulokseen nimelle `comment`.

Javan `properties`-tiedostojen lukeminen on toteutettu `PropertiesReader`-luokassa käyttäen **jproperties3**-kirjastoa. Kirjasto tuottaa syötetiedostosta listan `Property`- ja `Comment`-olioita. `Property`-oliot muunnetaan suoraviivaisesti järjestelmän käyttämiksi `Localization`-luokkaan liitettäviksi `Entry`- ja `Value`-olioiksi. Mahdolliset `Comment`-oliot tulkitaan kuuluvaksi listassa seuraavaksi olevaan `Property`-olioon.

Edellisen aliluvun 4.3.2 mukaiset `xlsx`-muotoiset tiedostot luetaan **openpyxl**-kirjastolla. Lukemiseen ja kirjoittamiseen on käytetty eri kirjastoja niiden erilaisen vahvuusalueiden takia. **openpyxl** kykenee käsittelemään `xlsx`-tiedostoja yleisellä tasolla, muttei juuri tue tekstin tyylytystä. **xlsxwriter** on nimensä mukaisesti vain kirjoittamiseen tarkoitettu kirjasto, mutta tukee tekstin tyylimäärittämiä paremmin.

```

1  ^"
2  (.*)
3  "
4  (? :
5      (\w+) )?§

```

Listaus 4.5. *Xlsx-tilukun ensimmäisessä sarakkeessa olevien avainten jäsenyykseen käytetty säännöllinen lauseke väljennetyssä muodossa.*

Taulukko 4.4. *Esimerkki xlsx-tiedoston sisällöstä. Monikon "days.left" muodot zero, two, few ja many on lukittu, sillä suomen ja englannin kielessä ei näitä muotoja käytetä.*

	fi_FI	en_GB
# Yleiset merkkijonot "product.notAvailable"	%s ei ole saatavilla.	%s is not available.
# Jäljellä olevat päivät "days.left.single"	Tämä päivä jäljellä.	This day left.
"days.left":zero		
"days.left":one	%d päivä jäljellä.	%d day left.
"days.left":two		
"days.left":few		
"days.left":many		
"days.left":other	%d päivää jäljellä.	%d days left.

Taulukon luku alustetaan selvittämällä taulukossa mukana olevat kielet ensimmäiseltä riviltä. Tämän jälkeen taulukko käydään rivi kerrallaan lävitse siten, että ensimmäisen sarakkeen perusteella päätellään rivillä olevan tiedon tyyppi ja luodaan sen perusteella Entry-olioon kielikohtaiset arvot. Ensimmäinen sarake jäsennetään listauksen 4.5 mukaisella säännöllisellä lausekkeella. Taulukon rivit, joiden ensimmäinen solu ei ole lausekkeen mukainen, jätetään huomiotta. Listauksen rivit 1–3 poimivat merkkijonon avaimen lainausmerkkien välistä. Rivit 4–5 määrittelevät valinnaisen lisäkkeen avaimelle siten, että avain ja lisäke ovat erotettu kaksoispisteellä ja lisäkkeen on oltava vähintään yhden merkin mittainen. Lisäke voi määritellä esimerkiksi monikkomuodon kategorian nimen. Taulukko 4.4 sisältää esimerkkinä joitakin sivulla 9 listauksessa 2.1 esitettyjä merkkijonoja siten, kuin ne näkyisivät xlsx-tiedostossa.

4.3.4 Tietokannan rakenne

Järjestelmän tietovarastona toimii aliluvussa 4.2 sivulla 19 esitetyn työkalun toisen version tavoin JSON-muotoinen tiedosto. Toisessa versiossa olleet implisiittiset ryhmärakenteet ja alustojen määritykset muutettiin eksplisiittisiksi sekä selvyiden

```

merkkijono = *UNICODE
locale = 2LCASE "_" 2UCASE
alusta_id = 1*UNICODE ; mutta ei "comment" eikä locale.
merkkijono_id = 1*UNICODE
ryhmä_id = 1*UNICODE
argumentti = 1*UNICODE

kommentti = "comment": merkkijono
alusta_alias = alusta_id: merkkijono
luokka_sana = "zero" / "one" / "two" / "few" / "many" / "other"
monikko = luokka_sana: merkkijono
käännös = locale: merkkijono / 1*monikko

merkkijono_olio = merkkijono_id: [kommentti] *alusta_alias *käännös
merkkijono_ryhmä = "strings": *merkkijono_olio

ryhmä_olio = ryhmä_id: [kommentti] [merkkijono_ryhmä] *ryhmä
ryhmä = "groups": *ryhmä_olio

alustan_valinta_muoto = "format": "android" / "ios" / "iosPlural" /
                             "properties" / "xlsx"
alustan_valinta_sisällytys = "includePlatforms": *alusta_id
alustan_valinta_muu = argumentti: ANY
                       ; argumentit ja arvot riippuvat muodosta

alustan_valinnat = alustan_valinta_muoto / alustan_valinta_sisällytys /
                  alustan_valinta_muu
alusta_määrittely = alusta_id: *alustan_valinnat
alustojen_määrittely = "platformConfig": *alusta_määrittely

tietokanta = alustojen_määrittely [kommentti] [merkkijono_ryhmä] *ryhmä
             ; tietokanta on myös ryhmä

```

Listaus 4.6. *Lopullisen tietokannan määrittely mukailussa ABNF-muodossa.*

että paremman laajennettavuuden vuoksi. Edellisessä versiossa ei esimerkiksi ollut mahdollista luoda ryhmää avaimella `comment` eikä `group-titles`, sillä ne sekoituivat tietokannan omiin rakennemäärittelyksiin. Listauksessa 4.6 esitetyn tietokannan rakenteen kuvauksen mukaisesti `ryhmä_olio` ja `merkkijono_olio` voivat nyt sisältää mitä tahansa nimiä avaimina ilman sekoittumisen vaaraa. Rakenne noudattelee aliluvussa 4.3.1 kuvattua varsinaista sovelluksen tietomallia.

Tietokannan luku `Localization`-olioksi tapahtuu muuntamalla tietokannassa olevat oliot vastaaviksi tietorakenteen olioiksi. Juurioliosta löytyvä `alusta_määrittely` kopioidaan sellaisenaan tietorakenteeseen, mikä jättää varsinaisten arvojen oikeellisuuden tulkinnan arvon lukijan vastuulle. Luvusta vastaava `Json2Reader`-luokka on jaoteltu funktioihin tietorakenteen mukaan: `juuri`, `ryhmä`, `ryhmän ryhmät`, `ryhmän alkio`, `alkio` ja `arvo`. Ryhmien luku tapahtuu aliryhmien osalta rekursiivisesti. Luvun aikana työkalu varmistaa, että jokaisen merkkijono-olion `merkkijono_id`

```

import json.encoder
import re
dct_patch = {
    "\u00ad": "\\u00ad", # SHY
    "\u00a0": "\\u00a0", # NBSP
}
dct_modified = json.encoder.ESCAPE_DCT.copy()
dct_modified.update(dct_patch)
json.encoder.ESCAPE = re.compile(
    json.encoder.ESCAPE.pattern.replace(
        "]", "".join(dct_patch.values()) + "]"
)
json.encoder.ESCAPE_DCT = dct_modified
json.encoder.encode_basestring = \
    json.encoder.py_encode_basestring
json.dump(oliopuu, ...)

```

***Listaus 4.7.** Tyhjemerkkien pakomerkintä JSON-tiedoston kirjoituksessa Python 3.5 -ympäristössä. Listauksesta on jätetty pois kirjaston alkuperäisten arvojen talletus ja palautus.*

on uniikki. Olion käännoksistä löytyvän arvon tyyppin perusteella päätellään, onko kyseessä tavallinen merkkijono vai monikko. Mikäli kyseessä on monikko, oliosta luetaan kaikki monikkoryhmien merkkijonot riippumatta kielelle sallituista arvoista. Käännosten merkkijonot jäsennetään käyttämällä `PrintFContainer`-luokan jäsennysfunktia.

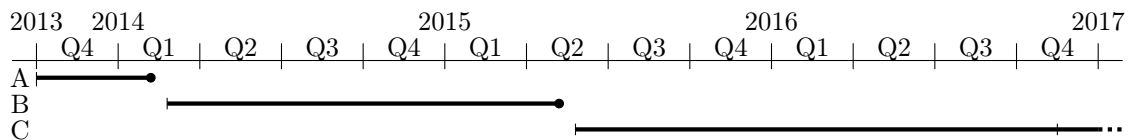
Tietorakenteen tallentaminen takaisin JSON-muotoiseksi tietokannaksi tapahtuu `Json2Writer`-luokan avulla luomalla `Localization`-olion tiedoista ensin Pythonin perustietotyypeistä muodostettu puu. Pääosin tämä tapahtuu käymällä ryhmärakenne syvyyteen ensin -menetelmällä läpi ja tekemällä edellisessä kappaleessa kuvattujen muunnosten käänteisoperaatiot. Avain-arvo-parien tallettamiseen käytetään kuitenkin `OrderedDict`-olioita olioiden avainten järjestyksen säilyttämiseksi. Ilman tätä järjestys saattaisi muuttua ajokertojen välillä aiheuttaen versionhallintaan tarpeettomia muutoksia ja mahdollisia vaikeasti selvitettäviä ristiriitoja.

Oliopuun tiedostoksi tallettamiseen käytetään Pythonin standardikirjaston `json`-kirjastoa. Sen toimintaa mukautetaan ajonaikaisesti tietokannan käsin muokkaamista helpottamaan siten, että tietyt tyhjemerkkit kirjoitetaan tiedostoon niiden varsinaisen merkkiesitystavan sijaan pakomerkinnällä. Listauksessa 4.7 esitetyllä tavalla kirjaston kirjoituksessa merkkien pakomerkintään käytettäviin muuttujiin lisätään SHY eli Soft Hyphen ja NBSP eli Non-Breaking Space -merkkien Unicode-arvot. Lisäksi merkkijonojen koodaamiseen käytetty funktio pakotetaan käyttämään Pyt-

honilla kirjoitettua versiota. Tämä ohittaa mahdollisen natiivikoodilla kirjoitetun optimoidun version, jonka toimintaa ei edellä kuvatuilla toimilla voida muuttaa.

5. ARVIOINTI

Käännösten hallintajärjestelmästä on tässä työssä perehdytty kolmeen iteratiiviseen versioon. Lisäksi vertailussa on mukana eräs kaupallinen työkalu, jonka käyttöä on yhdessä projektissa seurattu välillisesti. Tehtyjä työkaluja on käytetty Vincitillä projekteissa A ja C sekä kaupallista työkalua projektissa B. Vanhin näistä on 2014/Q1 päättynyt projekti A. Projekti B on päättynyt 2015/Q2 ja projekti C on yhä nytkin käynnissä. Työkalun käyttöönottoon on muissakin projekteissa esitetty kiinnostusta. Kuvassa 5.1 on havainnollistettu projektit aikajanelle. Taulukossa 5.1 on jäsennetty versiokohtaisesti, mitä ominaisuuksia kukin ratkaisu mahdollistaa.



Kuva 5.1. Projektien A, B ja C ajanjaksot.

Ensimmäisen version tarpeet olivat varsin pienet, mutta tarkat. Versio rakennettiin tarpeita tarkasti silmällä pitäen, eikä se käytännössä sovellu muuhun kuin alkupe-
räisen projektin käyttöön. Järjestelmää käyttämällä projektiin tuotiin kuitenkin 11 uutta kieltä lähdekielen lisäksi.

Toisen version kehityksessä oli visio järjestelmän laajemmasta käytöstä ja parem-
masta laajennettavuudesta. Sen toteutus kuitenkin tapahtui tiukassa aikataulussa, eikä toteutuksen teknistä suunnittelua ehditty miettiä kunnolla. Järjestelmä ehti olla projektikäytössä yli vuoden, ja siihen tuotettiin suomen- ja englanninkielisiä merkkijonoja. Sinä aikana tietokantaa ei käännetty ulkopuolisilla tahoilla. Toisen version tallennusmuoto oli muunnettavissa systemaattisesti käytettäväksi työkalun kolmannen version kanssa. Tietokantaan ehdittiin tätä versiota käyttäessä määrittelemään merkkijonoja 560 kappaletta Androidille ja 400 kappaletta iOS:lle. Merkkijonoja oli kokonaisuudessaan vain 750 kappaletta merkkijonon yhdistämisen vuoksi. Kahdella kielellä nämä muodostivat 4 444-rivisen JSON-tiedoston.

Varsinainen toteutettu järjestelmä eli kolmas versio on tätä kirjoitettaessa ollut käytössä projektissa C 2016/Q4 alkaen. Sinä aikana projektin tietokantaan on luotu

Taulukko 5.1. Ominaisuusvertailu järjestelmän eri versioiden ja erään kaupallisen tuotteen kesken. Merkkien selitykset: + Kyllä / Ominaisuus on tuettu; - Ei / Ominaisuus ei ole tuettu; * Katso huomiot; ? Ei tiedossa, katso huomiot.

Ominaisuus	v1	v2	v3	Kaupallinen	Huomiot
Työkalua käyttäneet projektit	A	C	C	B	
Muunnos Android → iOS	-	*	*	-	1
Muunnos iOS → Android	+	*	*	-	1
Monikot	-	-	+	*	2
Sanojen suvut	-	-	-	?	3
ICU MessageFormat	*	*	*	*	4
Merkkijonotaulukot	+	-	-	+	
Julkinen tallennusmuoto	+	+	+	-	
Erillinen tietokanta	-	+	+	+	
Versiointiystävällinen	+	+	+	-	
Vienti muihin muotoihin	-	-	+	+	
Tuonti muista muodoista	-	-	+	+	
Käännösmuisti	-	-	-	+	
Sanasto	-	-	-	+	
Hajautus eri kääntäjille	+	?	+	+	5
Tilastot käännösten valmiudesta	-	-	-	+	
Keskeneräisten havainnollistus	-	-	*	+	6
Edullinen hinta käyttäjälle	+	+	+	-	
Vaikutusmahdollisuudet	+	+	+	-	
Riippumattomuus käyttöjärjestelmästä	-	+	*	-	7

1. Muunnokset ovat tehtävissä versioissa 2 ja 3 vain yhteisen tietokannan avulla. Kaupallisessa versiossa toimenpide on tehtävissä käsityönä käännösmuistia hyödyntäen.
2. Kaupallista versiota varten kaikille kielille on määriteltävä kaikki monikkokategoriat. Tämä voi aiheuttaa varoituksia projektin lähdekoodin kääntäjältä.
3. Mikään järjestelmän versio ei tue ominaisuutta suoraan. ICU MessageFormat-muodon käyttö merkkijonoissa mahdollistaa sanojen sukujen käytön. Kaupallisen vaihtoehdon tuesta ei ole näyttöä.
4. MessageFormat-muodon käyttöä ei varsinaisesti ole estetty missään ratkaisussa. Käyttömahdollisuus riippuu kohdejärjestelmän kehityskehiksestä ja ohjelmoiduista ominaisuuksista.
5. Toisen version kehitys ei ehtinyt käännöskäyttöön asti.
6. Kolmannen version havainnollistus on käytettävissä vain taulukkolaskentatiedostoksi talletettuna.
7. Kolmas versio on ytimeltään riippumaton. Kommentojonotiedostot toimivat lähinnä POSIX-yhteensopivissa järjestelmissä kuten Linux ja macOS. Kommentojonot ovat lähinnä käyttöä helpottavia, eikä niitä ole pakko käyttää.

uusia ja muokattu vanhoja merkkijonoja. Versiota on koeteltu myös varsinaisessa käännösprosessissa. Edelliseen versioon verrattuna kolmas versio on helpommin laajennettava, vaikka se kaipaakin kipeästi jatkokehittämistä ja mahdollisesti suunnitelmallista uudelleentoteuttamista. Koodia työkaluun kertyi 4 545 riviä ja komentojoihin 231 riviä. Verrattuna toisen version rivimääriin, on lopullisessa toteutuksessa rivejä +228 % ja +863 %.

Tietokannassa olevat merkkijonomääritykset sisältävät kommentin ja käännösten lisäksi merkinnät mille alustoille merkkijono halutaan ottaa mukaan. Merkintä mahdollistaa myös merkkijonon avaimen vaihtamisen alustakohtaisesti. Merkinnän avulla merkkijonojen olemassaolo tai alustassa mukanaolo ei riipu minkään alustan kehitysvaiheesta.

Projektin C tietokanta sisälsi alunperin merkkijonot suomeksi ja englanniksi, ja niitä on kolmannen version käytön aikana käännetty viisi uutta kieliversiota, mikä nostaa kokonaiskielimäärän seitsemään. Tietokannassa on noin 1 480 merkkijonoa käännöksineen, mikä tarkoittaa noin 16 400 -rivistä JSON-tekstitiedostoa. Suomenkieliset versiot merkkijonoista sisältävät noin 5 030 sanaa. Alustamäärityksiä on kertynyt 43 kappaletta, ja niistä 37 kohdentuu iOS:lle. Loput kuusi alustaa sisältävät kaksi määritystä Androidille, kaksi taustajärjestelmälle, yhden ennakkokäännöksille ja yhden varsinaista käännettävää taulukkolaskentatiedostoa varten. Tietokannasta on määritelty Androidille käyttöön 700, iOS:lle 910 ja taustajärjestelmälle 120 merkkijonoa, yhteensä 1 730 merkkijonoa. Android- ja iOS-merkintöjen joukkojen yhdisteen suuruus on 1 350 kappaletta. Androidin ja iOS:n merkkijonoista noin 32 % on käytetty yhteisesti molemmille alustoille. Teorettinen maksimi yhteiskäytölle on 100 %.

Tekstimuotoinen tietokanta soveltuu hyvin versionhallintajärjestelmille, sillä useat valmiit järjestelmät toimivat parhaiten tekstitiedostojen versioinnissa. Tietokanta voidaan muuntaa työkalun avulla sekä sovelluskehityksessä tarvittaviksi tiedostoiksi että käännöstyössä käytettäväksi taulukoksi. Käännöstyön aikana tietokantaan tapahtuneet muutokset eivät häiritse käännettyyn tekstiin tuontia tietokantaan. Tuonnista aiheutuneet muutokset ovat nähtävissä ja yksitellen hyväksyttävissä tai hylättävissä versionhallintajärjestelmän tarjoamilla työkaluilla. Toteutuskielenä käytetty Python mahdollistaa työkalun ja kirjaston käytön parhaiten OS X- ja GNU/Linux-käyttöjärjestelmissä.

Käännösjärjestelmän tietokantaa voidaan muokata yksinkertaisimmillaan tekstieditorilla. Erillistä räätälöityä työkalua tietokannan manipulointiin ei toistaiseksi ole toteutettu. Näin ollen muokkaustoimenpiteiden käyttömukavuus riippuu täysin käy-

tetystä editorista. Esimerkiksi tietokannan ryhmähierarkiaa voi olla vaikea seurata ihmisen luettavaksi tarkoitettu muotoilusta huolimatta, jos editori ei tarjoa hierarkian korostusta, hierarkkista hakua tai rakennepolkua kohdistimen sijaintiin. Nämä luultavasti kasvattavat kynnystä lisätä tietokantaan uusia merkkijonoja.

Työkalun lopullinen versio ei kykene yhdistämään iOS:n eri lähteistä tulevaa käännösmateriaalia olemassa olevaan materiaaliin. Kaikki yhden lähteen merkkijonot päätyvät tietokannassa omaan ryhmäänsä käyttäen omaa alustamäärittystä. Operaatioissa lisätään tietokantaan avainkohtaisesti uudet ja kirjoitetaan yli vanhat merkkijonot. Valinnaisesti lähteestä poistetut merkkijonot poistetaan myös tietokannasta. Yksinkertaisen toimintatavan vuoksi tietokanta saattaa sisältää toistuvia merkkijonoja esimerkiksi eri näkymien "Jatka"-painikkeista. Painikkeen eri versiot näkyvät käännettävässä materiaalissa erillisinä, joten riippuu varsinaiseen kielelliseen kääntämiseen käytetyn työkalun käännösmuistin toiminnasta ja olemassaolosta, joudutaanko versiot kääntämään kerran vai monesti. Vastaavaa toisteisuutta voi aiheutua myös, jos tuotava näkymä on toteutettu jo toisella alustalla. Jos toistumista ei huomata, saatetaan käännökset joutua tekemään toisellekin alustalle, mikä mitätöi työkalun käytön hyödyn. Tätä varten työkalun pitäisi pystyä yhdistämään uuden näkymän merkkijonot tietokannassa jo oleviin merkkijonoihin.

Toinen versio työkalusta ideoitin ja otettiin käyttöön vasta muutaman kuukauden päästä projektin C aloittamisesta, jolloin projektista oli olemassa molemmilla alustoilla olemassa jo jonkinlainen prototyyppi. Käyttöönnotossa merkkijonojen avaimien muodosta sovittiin vain pääpiirteisistä muutosäännöistä. Tämän seurauksena avaimissa on huomattavissa epäyhtenäisyyttä pisteytyksen ja isojen kirjainten käytössä. Projektin tietokannan koko nykytilanteessa lienee syy, miksi kannan muokkaaminen voi tuntua vastenmieliseltä.

5.1 Jatkokehitysideat

Kriittisin kehityskohde työkalulle on tietokannan manipuloinnin helpottaminen. Pie nitöisin apu lienee toteuttaa jollekin yleiselle eri käyttöjärjestelmille saatavilla olevalle tekstieditorille laajennus tietokannan navigoinnin ja hierarkian visualisoinnin avuksi. Laajennus voisi samalla tarjota hakutoiminnallisuuden, joka kohdistuu merkkijonon avaimeen tai valittuun kieleen.

Kokonaisvaltaisempi ratkaisuvaihtoehto työkalun käytettävyyteen voisi olla toteuttaa työkalun idean pohjalta verkkopalvelu, jossa merkkijonot ja niiden hierarkia alustamäärittysineen olisi visualisoitu sopivalla tavalla. Tämä mahdollistaisi myös

käännöstyön siirtämisen työhön soveltumattomista taulukkolaskentatiedostoista työtä helpottavaan ympäristöön. Samalla kääntämistä varten ei tarvitsisi luoda osittaista tai kokonaista kopiota tietokannasta eikä kuljettaa tätä kopiota eri ihmisten kautta kääntäjille ja takaisin. Erikoisominaisuutena verkkopalvelu voisi tarjota lähes reaaliajassa päivittyvän esikatselukuvan merkkijonon muotoutumisesta ja sijainnista sovelluksessa. Verkkopalvelu tarkoittaisi todennäköisesti tässä työssä esitetyn tietokannan korvaamista relaatiokannalla. Palvelu vaatisi luultavasti myös oikean CAT-työkalun ominaisuuksia kuten käännösmuistin.

Tällä hetkellä yhdellä merkkijonolla voi olla kutakin alustaa kohden vain yksi nimi. IOS storyboardien materiaalin yhdistämisessä muuhun tietokannan sisältöön on mahdollinen ongelma, että yhdessä storyboardissa saattaa olla useampi samaa tarkoittava merkkijono eri avaimilla. Tällöin olisi tarpeellista pystyä määrittelemään kannassa olevalle merkkijonolle useampi nimi yhden alustan sisällä.

Ryhmälle voisi myös määritellä alustat, joille ryhmän merkkijonot oletuksena otetaan mukaan. Ryhmässä oleville merkkijonoille voisi lisäksi määritellä poikkeuksena lisättävät ja poistettavat alustat ja käytettävät avaimet. Tämä poistaisi esimerkiksi tarpeen toistaa yhden storyboardista tuodun ryhmän sisällä storyboardiin liittyvää alustamerkintää jokaiselle merkkijonolle erikseen.

Tuotaessa uusia merkkijonoja tietokantaan esimerkiksi storyboardista kannattaisi työkalun ehkä tarkistaa löytyykö vastaavia merkkijonoja jo tietokannasta. Tällöin tietokantaan ei tule käännettäväksi toistuvia samaa tarkoittavia merkkijonoja. Yhdistämisestä saattaa kuitenkin koitua tarpeetonta monimutkaisuutta lähdemateriaalin muutosten vaatimien toimenpiteiden päättelystä.

Työkalun arkkitehtuuri kaipaisi kunnollista uudelleensuunnittelua. Toteutuksen ytimen rakennetta on pääasiallisesti ohjannut tietokannan rakenne tuottaen jonkinlaisen malli-ohjain-hybridin. Nämä arkkitehtuuriosat voisi olla hyvä eriyttää siten, että malleja manipuloidaan ohjaimilla eikä malleilla olisi juurikaan omaa toimintalogiikkaa.

6. YHTEENVETO

Tässä työssä on käyty lävitse mobiilisovellukseen liittyvien merkkijonojen hallintaan tarkoitettuja ratkaisuja ohjelmistoprojektissa, jossa samaa sovellusta kehitetään useammalle kohdealustalle yhtä aikaa. Tarkoituksena ratkaisuisissa on pitää eri alustoilla olevat merkkijonot yhtenäisinä ja pyrkiä minimoimaan varsinaiseen kääntämistyöhön tarvittava työ. Ratkaisuja on vertailtu keskenään ja erään kaupallisesti saatavilla olevan käännösten hallintajärjestelmän kanssa.

Ensimmäinen työkalun versio on palvellut tarkoitustaan hyvin poistamalla toiseen alustan merkkijonojen käännöstarpeen kokonaan. Toinen versio on lähinnä prototyyppi kolmannelle versiolle. Kolmas työkalun versio muuttaa ohjelmistoprojektin merkkijonoihin ja kääntämiseen liittyvää prosessia merkittävästi. Alustakohtaisten merkkijonokantojen sijaan käytetään yhteistä merkkijonojen tietokantaa, johon on käännösten lisäksi merkitty kohdealustat, joihin merkkijono on tarkoitus viedä.

Kolmas versio lisää kehittäjän vaivannäön tarvetta, sillä työkalussa ei ole automaattista merkkijonojen yhdistintä eikä tietokannan muokkaamiseen tarkoitettua erikoistettua editoria, vaan yhtenäistys on täysin kehittäjän vastuulla. Työkalu siirtää käännöshallinnan tai kääntäjän tehtäviä kehittäjälle. Järjestelmän arvioinnin yhteenvedo on esitetty taulukossa 6.1.

Taulukko 6.1. *Yhteenvedo tavoitteista ja niiden toteutumisesta.*

Ominaisuus	Arvio
Riippumattomuus kohdealustassa käytettävästä tiedostomuodosta	Tuki eri tiedostomuodoille on erillinen osa työkalua.
Riippumattomuus kohdealustan kehitysvaiheesta	Merkkijono merkitään käyttöön alustakohtaisesti.
Versiointi	Tekstitiedosto on versioitavissa.
Muokattavuus	Tietokanta on muokattavissa tekstieditorilla.
Merkkijonojen käyttö yhteisesti / yhtenäistys	35 % mobiilialustojen merkkijonoista on yhteisiä työkalun ja tietokannan avulla.

Verrattaessa kolmatta versiota kaupalliseen käännöstenhallintatyökaluun ja kokonaisvaltaiseen käännösorganisaatioon, ei voida aukottomasti sanoa, kumpaa ratkaisua kannattaa käyttää. Verrattavissa olevia projekteja ei tähän mennessä ole riittävästi vertailukelpoisilla pohja-asetelmilla. Puutteeseen vaikuttaa muiden muassa tehdyn työkalun tuoreus, keskeneräisyys ja siitä seuraava julkaisemattomuus, sekä kaupallisten käännösprojektien vähäisyys projekteissa, joissa kirjoittaja on ollut mukana.

Työkalu vaikuttaa idean jatkokehityksen kannalta potentiaaliselta, mutta kaipaa tarkempaa arkkitehtuurista uudelleensuunnittelua. Lisäksi tutkittavaksi jää, olisivatko muut vastaavat ratkaisut käyttö- tai kehityskelpoisia vastaavaan käyttöön.

LÄHTEET

- [1] Esselink, B. *Practical Guide to Localization*. John Benjamins Publishing Company, 2000. 498 s.
- [2] Lommel, A. *The Globalization Industry Primer*, 2007. 60 s. [Viitattu: 1.10.2018] Saatavilla: http://www.academia.edu/3179195/The_globalization_industry_primer.
- [3] Locale Data Summary, 2018. [Viitattu 16.8.2018] Saatavilla: <http://www.unicode.org/cldr/charts/34/summary/fi.html> ja <http://www.unicode.org/cldr/charts/34/summary/en.html>.
- [4] Collation charts, 2018. [Viitattu 18.9.2018] Saatavilla: <http://www.unicode.org/cldr/charts/33/collation/fi.html> ja http://www.unicode.org/cldr/charts/33/collation/en_US_POSIX.html.
- [5] Haikala, I., Märijärvi, J. *Ohjelmistotuotanto*. Valikko. 11. Helsinki 2006, Talentum. 440 s.
- [6] Stoica, M., Ghilic-Micu, B., Mircea, M., Uscatu, C. Analyzing Agile Development - from Waterfall Style to Scrumban. *Informatica Economica*, 20(4):5–14, 2016.
- [7] Hakulinen, A., Vilkuna, M., Korhonen, R., Koivisto, V., Heinonen, T. R., Alho, I. *Iso suomen kielioppi*, 2004. [Viitattu 9.8.2018] Saatavilla: <http://scripta.kotus.fi/visk>.
- [8] Language Plural Rules, 2018. [Viitattu 27.8.2018] Saatavilla: http://www.unicode.org/cldr/charts/33/supplemental/language_plural_rules.html.
- [9] Elgin, B. Google Buys Android for Its Mobile Arsenal. 2005. [Viitattu 26.1.2017] Saatavilla: http://www.webcitation.org/5wk7sIvVb?url=http%3A%2F%2Fwww.businessweek.com%2Ftechnology%2Fcontent%2Faug2005%2Ftc20050817_0949_tc024.htm.
- [10] Morrill, D. Announcing the Android 1.0 SDK, release 1. 2008. [Viitattu 6.1.2017] Saatavilla: <https://android-developers.blogspot.in/2008/09/announcing-android-10-sdk-release-1.html>.
- [11] Gruber, J. The iPad's Dominance of the Tablet Market. 2011. [Viitattu 2.2.2017] Saatavilla: http://daringfireball.net/2011/07/ipad_dominance.

- [12] Ducrohet, X. Android 3.0 Platform Preview and Updated SDK Tools. 2011. [Viitattu 9.2.2017] Saatavilla: <https://android-developers.googleblog.com/2011/01/android-30-platform-preview-and-updated.html>.
- [13] Pichai, S. Sharing what's up our sleeve: Android coming to wearables. 2014. [Viitattu 9.2.2017] Saatavilla: <https://android.googleblog.com/2014/03/sharing-whats-up-our-sleeve-android.html>.
- [14] Pichai, S. Coming to a screen near you. 2014. [Viitattu 9.2.2017] Saatavilla: <https://googleblog.blogspot.fi/2014/06/google-io-2014-keynote.html>.
- [15] Satariano, A., Burrows, P., Stone, B. Scott Forstall, the Sorcerer's Apprentice at Apple. 2011. [Viitattu 2.3.2017] Saatavilla: <https://www.bloomberg.com/news/articles/2011-10-12/scott-forstall-the-sorcerers-apprentice-at-apple>.
- [16] Honan, M. Apple unveils iPhone. 2007. [Viitattu 6.3.2017] Saatavilla: <http://www.macworld.com/article/1054769/smartphones/iphone.html>.
- [17] iPhone Q&A – What operating system does the iPhone use? 2007, 2010. [Viitattu 6.3.2017] Saatavilla: <http://www.everymac.com/systems/apple/iphone/iphone-faq/iphone-runs-os-x-not-macos-x-cannot-run-macos-x-applications-skype-or-ipod-games.html>.
- [18] Van Rossum, G. A Brief Timeline of Python. 2009. [Viitattu 14.3.2017] Saatavilla: <http://python-history.blogspot.fi/2009/01/brief-timeline-of-python.html>.
- [19] Van Rossum, G. Personal History - part 1, CWI. 2009. [Viitattu 20.3.2017] Saatavilla: <http://python-history.blogspot.fi/2009/01/personal-history-part-1-cwi.html>.
- [20] Peterson, B. Python 2.7 Release Schedule. 2008. [Viitattu 14.3.2017] Saatavilla: <https://www.python.org/dev/peps/pep-0373/>.
- [21] Glossary – Python 3.6.1 documentation. [Viitattu 30.3.2017] Saatavilla: <https://docs.python.org/3/glossary.html#term-bytecode>.
- [22] PyPI - the Python Package Index. [Viitattu 30.3.2017] Saatavilla: <https://pypi.python.org/pypi>.
- [23] Crockford, D. JSON: The Fat-Free Alternative to XML, 2006. [Viitattu: 1.11.2018] Saatavilla: <http://www.json.org/fatfree.html>.
- [24] Ecma International. The JSON Data Interchange Syntax, 2007. [Viitattu 1.11.2018] Saatavilla: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>.

- [25] Hoffman, P., Yergeau, F. UTF-16, an encoding of ISO 10646, 2000. [Viitattu 19.4.2018] Saatavilla: <https://tools.ietf.org/html/rfc2781>.
- [26] Crocker, D., Overell, P. Augmented BNF for Syntax Specifications: ABNF, 2008. [Viitattu 18.1.2018] Saatavilla: <https://tools.ietf.org/html/std68>.

LIITE A: MUKAILTU ABNF

Augmented Backus–Naur Form eli ABNF on tarkoitettu tiedonvälitysprotokollien formaaliin määrittelyyn. Tässä työssä käytetty mukailu lisää muotoon mahdollisuuden avain–arvo-parien yksinkertaisempaan määrittelyyn.

Lyhyesti muoto on määritelty seuraavasti [26]:

- Terminaalit kirjoitetaan isoilla kirjaimilla, kuten UNICODE.
- Säännöt sisältävät nimen ja määrittelyn: sääntö = määrittely. Säännöllä voidaan määrittellä sekä terminaali että yhdistelmä sääntöjä.
- Ryhmä määritellään sulkeilla, (ensimmäinen toinen).
- Vaihtoehtoisuus merkitään kauttaviivalla: ensimmäinen / toinen.
- Toistuvuuden määräraajat merkitään <min>*<max>elementti jossa oletus arvolle min on 0 ja arvolle max ääretön. Näin *elementti määrittelee min-kä tahansa määrän elementti-elementtiä, 1*elementti vaatii vähintään yhden ja *1elementti sallii korkeintaan yhden elementti-elementin.
- Eksakti toistuvuus voidaan merkitä elementtiä edeltävällä numerolla, kuten 8UNICODE joka vastaa siis määrittelyä 8*8UNICODE.
- Vapaaehtoiset elementtijonot merkitään hakasulkuihin, [vapaaehtoinen].
- Kommentit merkitään puolipisteellä ; ja ne jatkuvat rivin loppuun asti.
- Literaalit kirjoitetaan lainausmerkkien väliin.

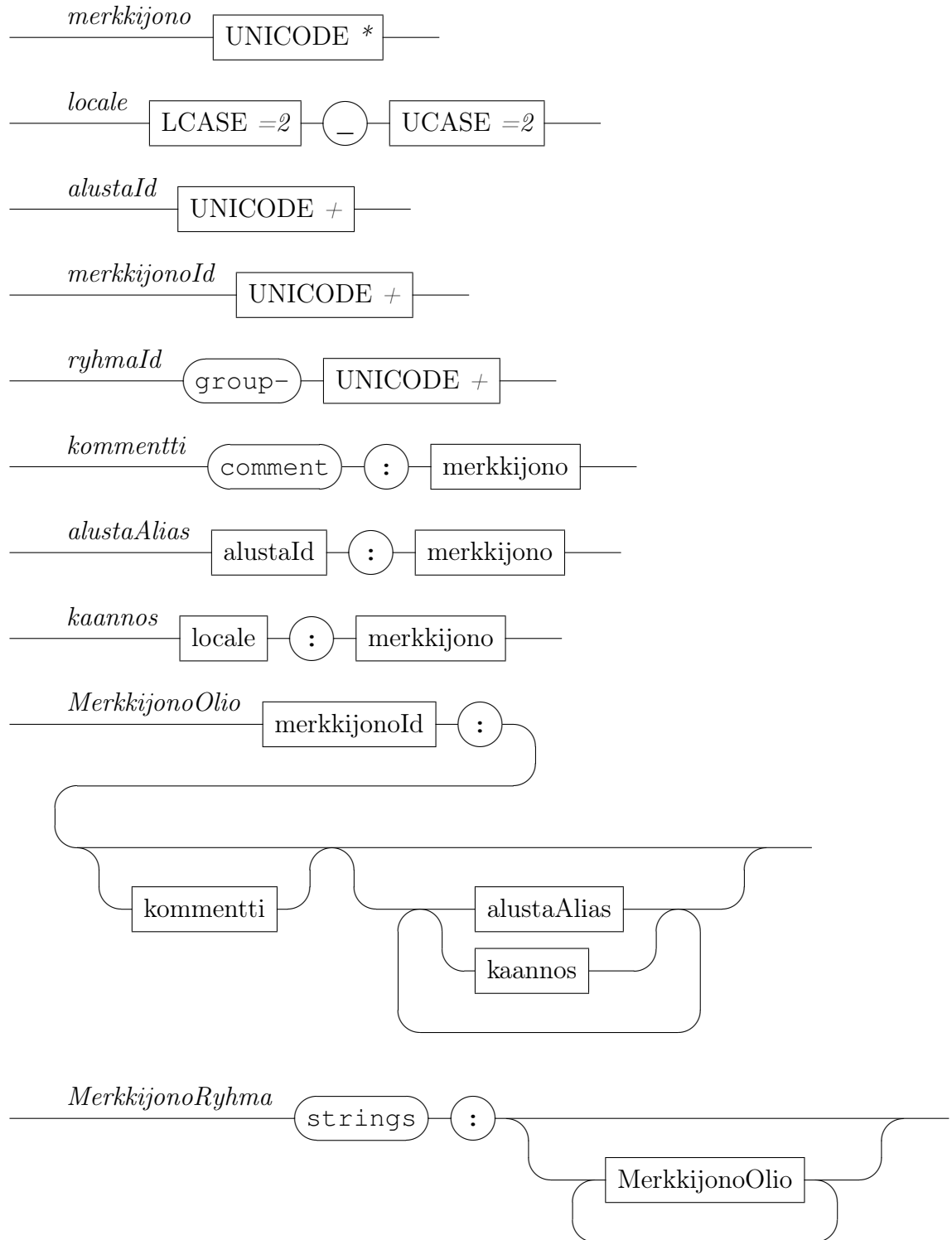
Tässä työssä käytetty mukailu lisää sääntömäärittelyyn olion ominaisuuksien määrittelyn kaksoispistenotaatiolla mahdollistaen kokonaisten olioiden määrittämisen:

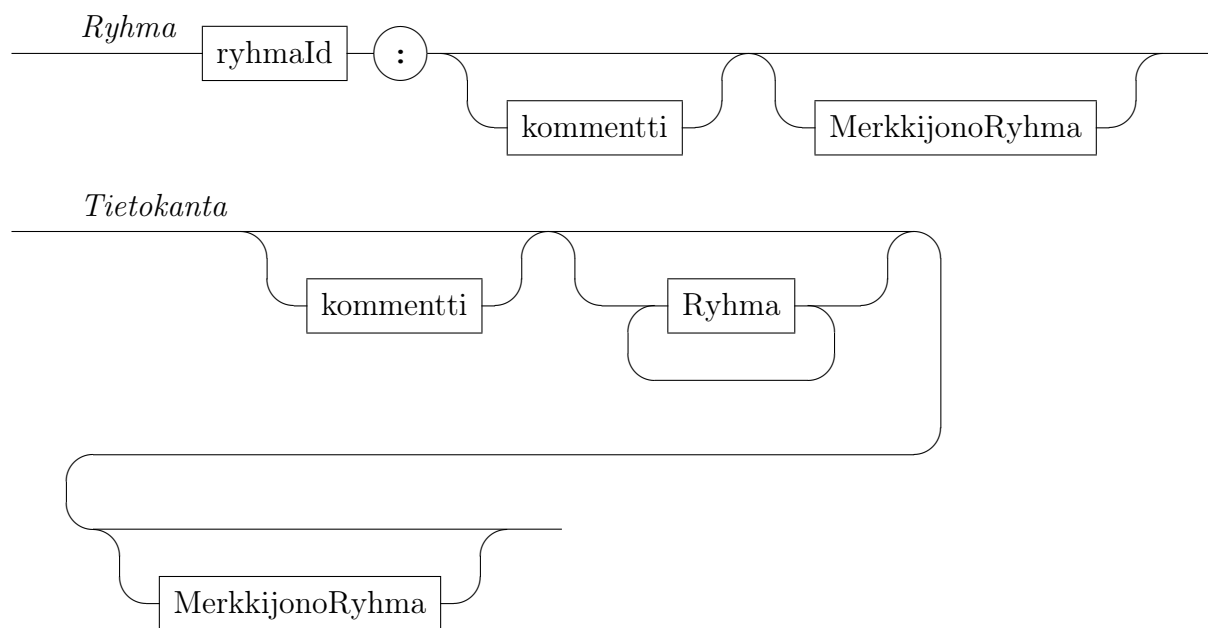
```
arvo = terminaalisääntö / olio-sääntö
avain-arvo-sääntö = avain: arvo
olio-sääntö = *avain-arvo-sääntö
```

Lisäksi määritellään löysästi joitakin oletusterminaleja:

```
UNICODE ; Mikä tahansa unicode-merkki
LCASE ; Pieni kirjain
UCASE ; Iso kirjain
BOOLEAN ; Boolean-olion arvo
INTEGER ; Kokonaisluku
ANY ; Mikä tahansa edellisistä TAI olio-sääntö
```

LIITE B: TOISEN VERSION (V2) TALLENNUSMUOTO





LIITE C: LOPULLISEN VERSION (V3) TALLENNUSMUOTO

