TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Isikunimede analüüs, parsimine ja vormindamine Eesti, Saksamaa ja Suurbritannia nimede näitel

Bakalaureusetöö

Üliõpilane: Liisi Mõtshärg
Üliõpilaskood: 112704IAPB
Juhendaja: dotsent Gunnar Piho

Tallinn
2015

Autorideklaratsioon

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

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Käesoleva bakalaureusetöö „**Isikunimede analüüs, parsimine ja vormindamine Eesti, Saksamaa ja Suurbritannia nimede näitel**“ eesmärgiks on luua meetodid kolmes nimetatud riigis kasutavate isikunimede vormindamiseks ning parsimiseks.

Töös analüüsitakse nimetatud riikides levinumaid nimekujusid ja olemasolevaid nimemudeleid ning valitakse töö teostamiseks sobiv mudel. Seejärel koostatakse vormindusnotatsioon, seda kasutav vormindaja ning parser, mis terve nimestringi osadeks jagab.

Töö tulemusena kirjeldatakse vormindusnotatsioon ning valmib C# teek, mis võimaldab nimetatud riikides enimlevinud kujuga isikunimesid osadeks parsida ning eelnevalt osadeks jaotatud isikunime loodud notatsiooni järgi vormindada.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 67 leheküljel, 7 peatükki, 10 joonist, 16 tabelit ja 4 lisa.

Abstract

The main purpose of this bachelor thesis, „**Analysis, Parsing and Formatting of Personal Names on the Example of Estonia, Germany and the United Kingdom**“, is to create methods for parsing and formatting personal names, using Estonian, German and British names as examples.

Common names of these three countries as well as existing models of personal name are analysed to choose a suitable name model. Next, a formatting notation is constructed, along with a formatter to use it and a parser that breaks full names into separate parts.

The result of this thesis is a description of formatting notation and a C# library that can parse and format personal names found in the three countries mentioned.

The thesis is in Estonian and contains 67 pages of text, 7 chapters, 10 figures, 16 tables and 4 appendices.

Lühendite ja mõistete sõnastik

UML	***Unified Modelling Language*** Noteerimiskeel infosüsteemide mudelite visualiseerimiseks
DTD	***Document Type Definition*** Dokumenditüübi määrang, SGML ja XML dokumentide juurde kooluv fail, mis kirjaldab, kuidas rakendusprogramm dokumendis esinevaid märgendeid tõlgendama peab. [1]
FHIR	***Fast Healthcare Interoperability Resources*** Standardi mustand, mis kirjeldab elektrooniliste terviseandmete vahetamise andmetüübid, elemendid ja liidese. [2]
Parser	Teksti või süntaksit analüüsiv programm
API	***Application Programming Interface*** Kirjeldab mingi ressursi, mille poole teised süsteemis saadava pöörduda. Võimaldab struktureeritud andmevahetust eri süsteemide vahel.
AJAX	***Asynchronous JavaScript and XML*** Võimaldab pärida ressursse, ilma et oleks vaja veebilehte uuesti laadida.
JSON	***JavaScript Object Notation*** Kergekaaluline andmevahetusformaat, mida on lihtne lugeda ja muuta
DTO	***Data Transfer Object*** Andmetevahetusel kasutatav kerge mudel, mille eesmärk on anda andmetele kindel kuju ning mis ei sisalda äriloogikat.
CORS	***Cross-Origin Resource Sharing*** Võimaldab pärida AJAXiga ressursse, mis asuvad teises domeenis (vaikimisi on see keelatud)

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

1.1 Taust ja probleem

Enamiku inimeste jaoks on nimi nende identiteedi tähtis osa, mistõttu on oluline selle korrektne käsitlemine. Infosüsteemides, kus ei saa hoida inimese isiksust, käitumismaneere ega välimust, võib nimi olla ainus tema identiteediga seonduv osa.

Nime on mingil kujul tarvis näidata peaaegu igas süsteemis, olgu siis personaliseeritud tervituses või isikuandmeid sisaldavas baasis suure hulga teiste nimede vahel. Algselt ühel kujul sisestatud nime võib olla tarvis näidata hoopis teisel kujul, näiteks perekonnanime enne eesnime või ainult initsiaale. Võimalikke kasutusjuhte on palju ning need sõltuvad kindlast süsteemist. Mõnel juhul võib olla vaja adresseerida isikut ametlikult, täisnime ja tiitlitega, teisel juhul aga ainult eesnime pidi. Selleks on tarvis meetodit, mis võimaldaks isikunime näidata lihtsalt kirjeldataval kujul – analoogselt kuupäeva vormindamisega.

Vormid, kus inimesel palutakse sisestada oma nimi osade kaupa, on nimeosade eraldamiseks kasulikud, kuid kasutaja jaoks mitte väga mugavad. Samuti ei pruugi eri riikides ja kultuurides olla kasutusel samad nimeosad. Näiteks venelastel on eesnime järel isanimi, ungarlased kirjutavad perekonnanime eesnime ette ja hispaanlastel on üldjuhul mitu perekonnanime. Leidub ka piirkondi, kus isikutel on tavapäraselt ainult üks nimi, näiteks Lõuna-India. [3] Seepärast oleks tarvis ka meetodit, mis suudaks ühes tükis oleva nimestringi ise nimeosadeks jagada.

Võimalus ühte nime eri kujudel näidata on vajalik eelkõige kõigile neile, kes tegelevad isikuandmeid käsitleva või kasutava tarkvara loomisega.

Käesolevas töös lahendatavateks probleemideks on, kuidas kuvada isikunime, kasutades kindlat vormindusnotatsiooni ning kuidas jagada isikunime osadeks, kui see on antud ühe stringina.

Eri kultuurides esineb palju erinevaid nimekujusid ja oleks peaaegu võimatu neid kõiki ühes töös käsitleda. Seepärast otsustati käesoleva töö jaoks välja valida ainult 3 riiki.

Töö algne teemapüstitus tulenes Suurbritannia Leedsi ülikooli proteoomiksite laboratooriumi infosüsteemi Sentry vajadusest näidata nimesid eri kujudel. Seepärast valiti üheks uuritavaks riigiks Suurbritannia. Saksamaa valiti, kuna see on üks maa, kuhu nimetatud infosüsteemi tahetakse laiendada. Eesti valiti, kuna tegu on autori kodumaaga.

1.2 Ülesande püstitus

Selle töö eesmärgiks on luua meetodid, mis võimaldaks Eestis, Suurbritannias ning Saksamaal enimlevinud kujuga isikunimesid osadeks parsida, samuti osadeks jaotatud isikunime kindla formaadi järgi vormindada. Teemapüstitusest lähtuvalt formuleeriti järgnevad põhieesmärgid.

- Leida sobiv nime mudel, mida saaks kasutada nii Eesti, Suurbritannia ja Saksamaa nimede esitamiseks.
    - See eesmärk tingis vajaduse analüüsida ka nende riikide nimekujusid.
- Välja töötada lihtne vormindusnotatsioon, mis võimaldaks isikunime osade kaupa vormindada.
- Koostada algoritm, millega eelnevalt osadeks jaotatud nime vastavalt etteantud formaadile vormindada, s.t mis realiseeriks loodud vormindusnotatsiooni.
- Koostada algoritm, mis oskaks ühe stringina etteantud nime osadeks jagada.

1.3 Metoodika

Isikunimed varieeruvad üsna suurel määral, seetõttu analüüsitakse eesmärgini jõudmiseks kõigepealt nimetatud riikide isikunimesid. Seejärel analüüsitakse kirjandusest leitud olemasolevaid nimemudeleid ning valitakse välja käesoleva töö jaoks sobiv mudel, mida vajadusel ka mugandatakse.

Kui mudel on valitud, koostatakse vormindusnotatsioon, mis võimaldaks isikunimesid vormindada vastavalt etteantud kujule.

Lõpuks realiseeritakse eelnevalt koostatud vormindusnotatsiooni kasutada suutev vormindaja. Samuti realiseeritakse parser, mis suudaks ühe stringina esitatud nime töös valitud mudelile vastavateks osadeks jaotada. Vormindaja ja parser koos valitud isikunime mudeliga vormistatakse C# teegina.

Vormindaja ning parseri töö kontrollimiseks koostatakse *unit test'*id, kasutades NUnit testimisraamistikku. Lisaks luuakse eelnevalt tehtud töö illustreerimiseks ASP.NET MVC ja APS.NET Web API raamistikke kasutades väike veebiteenus, ning seda kasutav veebileht.

Arenduskeskkonnana kasutatakse Visual Studio 2013. Kogu töö on realiseeritud kasutades Microsofti .NET raamistikku versiooninumbriga 4.5.1. Programmeerimiskeeleks valiti C#, kuna just selle keele vastu on autoril sügavam huvi. Veebilehe ja -teenuse omavaheliseks suhtluseks kasutatakse JavaScripti ja AJAX päringuid.

1.4 Ülevaade tööst

Töö koosneb seitsmest peatükist. Esimeses peatükis antakse ülevaade Eestis, Saksamaal ja Suurbritannias levinumatest nimekujudest.

Teises peatükis uuritakse olemasolevaid nimemudeleid ning tuuakse välja nende positiivsed ja negatiivsed küljed. Lõpuks valitakse mudel, mida töö praktilises osas kasutama hakatakse.

Kolmandas peatükis luuakse töö teiseks eesmärgiks olnud vormindusnotatsioon ning antakse ülevaade selle osistest.

Neljandas peatükis räägitakse lähemalt töö kolmandast eesmärgist, notatsiooni realiseerivast algoritmist ning selle tööpõhimõttest.

Viiendas peatükis kirjeldatakse töö neljandat eesmärki, stringi kujul etteantavat nime osadeks parsivat algoritmi.

Kuuendas peatükis käsitletakse vormindaja ja parseri testimist, selleks loodud testjuhte ning testimise tulemusi.

Seitsmendas peatükis antakse lühike ülevaade valminud teeki kasutavast veebiteenusest ning omakorda seda kasutavast veebilehest.

Kogu töö käigus valminud kood on kättesaadav aadressil: https://bitbucket.org/LiisiM/thesis.

2. Isikunimede analüüs

Nimede uurimisega tegeleb onomastika, isikunimedega selle alamharu antroponüümika. Eesti Keele Käsiraamat defineerib isikunime järgnevalt: „Isikunimi ehk antroponüüm on inimest tähistav nimi. Isikunime osad on eesnimi, s.o lapsele tema sünni registreerimisel antav nimi, ja perekonnanimi, s.o vanemailt lapsele kanduv nimi.“ [4] Seega on isikunimi kõige lihtsamas vormis esitatav kujul {eesnimi} {perenimi} (perenimi on töös läbivalt esitatud allajoonituna).

Lisaks eelnevalt nimetatud ees- ja perekonnanimele kuuluvad mõnedel juhtudel nime juurde ka ees- ja järelliited. Need ei moodusta küll osa isikunimest, kuid neid käsitletakse sellega koos käivatena. Liited võivad näidata näiteks seisust, akadeemilist kraadi, auastet, perekonnaseisu. Eesliited on näiteks Hr., Prl, järelliited juunior, seenior, IV. [3]

Selles peatükis käsitletakse lähemalt Eestis, Saksamaal ning Suurbritannias levinumaid isikunimede kujusid. Iga riigi kohta leitakse mustrid, millele antud riigis kasutusel olevad nimed vastavad. Töö skoobist jäetakse välja sisserännanute nimed, olenemata sellest, kui suure protsendi riigi rahvastikust nad moodustavad. Alampeatükkides toodud tabelites kasutatakse nii väljamõeldud kui ka päriselt elavate või elanud inimeste nimesid.

Vastavalt keelereeglitele käsitletakse mustrite leidmisel sidekriipsuga ühendatud liitnimesid ühe nimena. [5]

2.1 Eesti

Eesti Vabariigis reguleerib nimede kasutust Nimeseadus, millele kõik antavad nimed vastama peavad. Nimeseaduse järgi koosneb nimi ühest või mitmest eesnimest ja perekonnanimest. Nime andmisel võib eesnimi koosneda ühest kuni kolmest nimest või kahest sidekriipsuga ühendatud nimest. Perekonnanimi võib olla üks nimi või kaks sidekriipsuga ühendatud nime ja nimed ei tohi sisaldada numbreid või mittesõnalisi tähiseid. [6]

Abiellumisel võivad mõlemad osapooled võtta kasutusele ühe nime, samuti on ühel osapoolel lubatud lisada oma endisele perekonnanimele sidekriipsuga eraldatult partneri perekonnanimi, tingimusel, et kumbki nimi ei koosne eelnevalt mitmest nimest. [6]

Võõrkeelsete nimede puhul kehtivad selle riigi, kus nimi registreeriti, seadused ja reeglid, samuti peab võõrkeelne eesnimi olema ka teises riigis eesnimena kasutusel. Välisriigi kodanikule või kodakondsuseta isikule kohandatakse nimi Eestis esmakordsel isikuandmete deklareerimisel vastavalt ümberkirjutusreeglitele. Kohaldatud nimede puhul võivad nii ees- kui perekonnanimi koosneda ühest või rohkemast nimest. [6] Käesolevas töös keskendutakse ainult Eestis antavatele nimedele, kohandatud nimed jäetakse kõrvale.

Järgnevas tabelis on esitatud Eestis kasutusel olevad nimemustrid, mis on koostatud eelneva Eesti isikunimede kirjelduse põhjal. Esitatud ei ole kõikvõimalikud kombinatsioonid, vaid on üritatud leida iga nimeosa kohta võimalikult palju variatsioone. Mitmest elemendist koosnev nimi on märgitud kujul {nimi}n, kus n tähistab nimede arvu.

Tabel 1. Eesti nimemustrid

Muster	Näide	Üldistatud muster
{eesnimi} {perenimi-perenimi}	Juhan Tamm-Kadakas	{eesnimi}*n* {perenimi}
{eesnimi-eesnimi} {perenimi}	Juhan-Joonas Tamm	
{eesnimi}*n* {perenimi}	Mari Julia Tamm Juhan Jaagup Joonas Tamm	
{eesnimi} {isanimi} {perenimi}	Juri Andrejevitš Andropov	
{eesliide} {eesnimi} {perenimi}	Hr. Juhan Tamm	*{eesliide}* {eesnimi}*n* {perenimi}

Muster {eesnimi} {isanimi} {perenimi} oli väga levinud Nõukogude ajal, kuid tänapäeval seda eestlaste hulgas enam ei kasutata [4]. Käesoleva töö raames käsitletakse isanime kui ühte isiku eesnimedest.

2.2 Saksamaa

Raamatus „Names of Persons: National Usages for Entry in Catalogues“ tuuakse välja nimedes üldiselt leiduvad osad ning nende erinevad kujud. Selle järgi koosnevad Saksamaa nimed ühest või mitmest eesnimest ning perekonnanimest. Mitmeosaline eesnimi võib, nagu Eestiski, koosneda kahest sidekriipsuga ühendatud nimest või mitmest eraldiseisvast nimest. [7]

Perekonnanimi võib koosneda ühest lihtnimest või liitnimest, mille osad on ühendatud sidekriipsuga või sidesõnaga ’und’ (sks. ’ja’) [7]. Lisaks võib nimi sisaldada ka erinevaid eesliiteid, mis üldiselt tähistavad nime aadlipäritolu. Kuni aastani 1919, mil Saksamaal moodustati Weimari vabariik, oli aadliseisus omandatav ning aadlikele kehtisid eritingimused. Weimari vabariigi põhiseaduse kasutuselevõtmisega 11. augustil 1919 kuulutati kõik kodanikud võrdseiks ning aadlitiitlid võisid edaspidi kasutusel olla ainult osana perekonnanimest [8]. Ülikuseisust tähistavateks eesliideteks on *von, zu, von der, von dem* või *vom, zu der* või *zur, zu dem* või *zum, von und zu*. Algselt tähistasid *von* ja teised eesliited lihtsalt mingist kohast pärit olemist, seetõttu võivad need eesliited kuuluda ka mitte-aadlipäritolu perekonnanime juurde. [9]

Järgnevalt (Tabel 2) on välja toodud Saksamaal levinumad nimemustrid koos näidetega. Tabel on koostatud raamatu „Names of Persons: National Usages for Entry in Catalogues“ alusel. Sidekriipsuga ühendatud nimesid on käsitletud ühe nimena, nagu ka perekonnanimesid koos eesliidetega. Tabelist järeldub, et kõik Saksamaal esinevad nimed saab taandada üldkujule {eesnimi}n {perenimi koos eesliidetega}. Osad perekonnanimed koosnevad mitmest nimest, ka need on antud tabelis võrdsustatud ühe nimega. Erandiks on nimed, kus mitme nimeosa ühendamiseks kasutatakse perekonnanime eesliiteid, mille ees teist nime olla ei pruugi, näiteks Meyer *zu* Selhausen (siinkohal eeldatakse, et ka *zu* Selhausen eraldi võib olla perekonnanimi).

Tabel 2. Saksamaa nimemustrid

Muster	Näide	Üldistatud muster
{eesnimi}*n* {perenimi}	Erich Maria Remarque Maria Schmidt	{eesnimi}*n* {perenimi}
{eesnimi-eesnimi} {perenimi-perenimi}	Maria-Clara Schmidt-Müller	
{eesnimi}*n* {*eesliide/-liited* + perenimi}	Karl Ernst *von* Baer Johann *von der* Hagen Johann *vom* Berg Johann *von zur* Mühlen Johann *von und zu* Urf	
{eesnimi}*n* {perenimi und perenimi}	Johann Strauss und Torney	

Muster	Näide	Üldistatud muster
{eesnimi} {perenimi+kohanimi}	Johann Müller-Meiningen	
	Johann Meyer *zu* Selhausen	{eesnimi}*n* {perenimi}*n*
{eesnimi} {perenimi eesliitega Sankt}	Johann Sankt Goar	{eesnimi}*n* {perenimi}
{eesnimi} {perenimi *genannt* perenimi}	Johann Schmidt *genannt* Müller	
{eesnimi} {tiitel perenime osana + perenimi}	Adolf Friedrich Graf *von* Schack	

2.3 Suurbritannia

Sarnaselt Eestile ja Saksamaale koosneb ka Suurbritannia nimi tavaliselt ühest või mitmest eesnimest ja perekonnanimest. Erinevus seisneb selles, et mitme eesnime korral käsitletakse esimest neist eesnimena ja järgnevaid keskmiste nimedena. [10] Lähtudes peatüki alguses toodud isikunime definitsioonist ei tehta antud töös vahet ees- ja keskmise nime vahel ning käsitletakse neid kõiki eesnimedena.

Järgnevalt (Tabel 3) on toodud Suurbritannias levinumad nimemustrid koos näidetega. Ka see tabel põhineb raamatul „Names of Persons: National Usages for Entry in Catalogues". Tabelist on välja jäetud ainult tiitlist koosnevad nimed, näiteks *Duke of Marlborough.* [7]

Tabel 3. Suurbritannia nimemustrid

Muster	Näide	Üldistatud muster
{eesnimi}*n* {perenimi}	John Smith John Thomas Smith Susan Foreman-Campbell Matthew St. John Susan De La Mare	{eesnimi}*n* {perenimi}
{eesnimi} {perenimi} {perenimi}	Helena Bonham Carter	{eesnimi}*n* {perenimi}*n*
{eesnimi} {perenimi},	Benjamin Disraeli, *Earl of*	{eesnimi}*n* {perenimi},

Muster	Näide	Üldistatud muster
{tiitel}	*Beaconsfield*	*{järelliide}*
{eesnimi} {perenimi}, *{tiitel koos kohanimega}*	Henry Home, *Lord Kames*	
{tiitel} {eesnimi} {perenimi}, *{tiitel}*	*Sir* Thomas Beecham, *bart.*	*{eesliide}* {eesnimi}*n* {perenimi}, *{järelliide}*
{tiitel} {eesnimi} {perenimi}	*Sir* John Smith *Dame* Anne Potter *Lord* Isaac Abbott *Lady* Sarah Cook *Rev.* William Todd *Hon.* Donald Carlyle *Lord Justice* Ian Smith	*{eesliide}* {eesnimi}*n* {perenimi}

Leidub nimekujusid, mille puhul pole kindlalt võimalik öelda, mis osa on eesnimi ja mis perekonnanimi. Parim näide on kolmeosaline nimi, kus pole võimalik öelda, kas keskmine element on ees või perekonnanimi. Sagedamini on tegu eesnimega, kuid mitte alati.

2.4 Kokkuvõte

Kõigi vaadeldud keelte nimed saab taandada üldkujule {eesnimi}*n* {perenimi}, kui lugeda perekonnanimede liited nende nimede juurde kuuluvateks. Leidub ka sellest mustrist erinevaid nimesid, näiteks {eesnimi}*n* {perenimi}*n*. Esimesele mustrile võivad lisanduda ka ees- ja/või järelliited, neid väljendab muster *{eesliide}* {eesnimi} {perenimi} *{järelliide}*.

Kõigi keele puhul on vaja salvestada mitu eesnime ja erandjuhtudel ka mitu perekonnanime. Kuna eesnimesid võib isikul olla palju, oleks vaja ka välja, kus hoida nime, mida inimene ise tavaolukorras kasutada soovib.

Kuigi eelnevates peatükkides toodi ees- ja järelliited eraldi välja ainult Suurbritannia puhul, eeldatakse et neid võib esineda kõigis vaadeldavates keeltes. See tingib vajaduse hoida neid samuti nime juures.

Isikunimedel on ka ühisomadusi, mis ei sõltu keelest ega kultuurist. Nimi võib ajas muutuda, näiteks abiellumisel, lahutuse järel või lihtsalt isiku soovil. Ühel isikul võib korraga olla

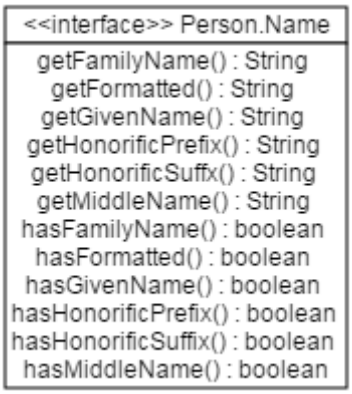
kasutusel ka mitu nime, sel juhul on üks nimedest (arvatavasti) isiku seaduslik nimi ning teised aliased, hüüd- või esinejanimed või mingit muud tüüpi nimed.

3. Nime mudel ja analüüs

Selles peatükis saavutatakse töö esimene põhieesmärk, nimemudeli valik. Selleks analüüsitakse erinevaid kirjandusest leitud nimemudeleid. Allikmaterjalides olid kõik mudelid eri kujul, osad joonistena ja teised teksti või XML kujul. Siin on nad võrdlemise lihtsustamiseks esitatud loogiliste andmemudelitena kasutades UML klassidiagramme. Iga mudeli kohta esitatakse üks diagramm, kus on ära toodud atribuutide nimed ja nende andmetüübid. Diagrammide joonistamiseks kasutati veebipõhist skeemiprogrammi Gliffy [11].

3.1 Olemasolevate mudelite kirjeldused

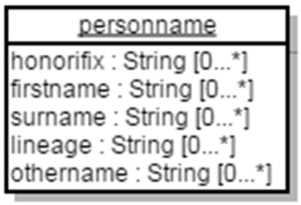
Google'i Androidi raamistikus kasutatakse liideste isiku nime osade kasutamiseks (Joonis 1) [12]. Liidesel on antud nime osade kasutamiseks meetodid *getGivenName()*, *getMiddleName()*, *getFamilyName()*, *getHonorificPrefix()* ja *getHonorificSuffix()*. Kuigi tegu on liidese ja mitte andmemudeliga, saab sellest tuletada kujuteldava andmemudeli, asendades kõik *get* meetodid atribuutidega. Saadud mudelil on atribuudid *givenName, middleName, familyName, honorifixPrefix* ja *honorificSuffix.* Selline mudel on lihtne ja sobib hästi isiku nime kujutamiseks juhtudel, kui on vaja kuvada tema teadaolevat nime antud hetkel ning ei ole tarvis järge pidada nime muutumiste ega kasutusotstarbe kohta. Mudeli puuduseks on ka selle „läänelikkus", „keskmine nimi" on mõiste, mis esineb üldiselt inglise keeleruumis.



Joonis 1. Google Android'i Person.Name liides

Docbook on dokumenditüübi määrang (DTD) struktureeritud dokumentide loomiseks. Nende mudel (Joonis 2) erineb Google omast selle poolest, et *middleName* asemel on kasutusel

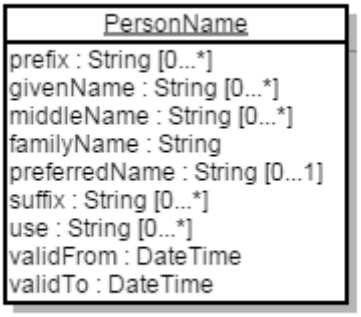
mõiste *other name*, mis võib tähistada ükskõik missugust nime osa, mis pole ei ees- ega perekonnanimi. Täpselt missuguste nimeosade talletamiseks atribuuti kasutatakse, on mudeli kasutaja enda otsustada. Atribuudile määratakse kirjeldav tüüp, näiteks keskmise nime puhul „mi“. Tüübid ei ole kindlalt defineeritud ning nende valik sõltub mudeli kasutajast. [13] Docbook'i mudel on paindlikum, kuna lubab talletada erinevat tüüpi lisanimesid, samal ajal defineerides kindlad väljad enimkasutatud nimeosade jaoks.



Joonis 2. Docbook isikunime mudel

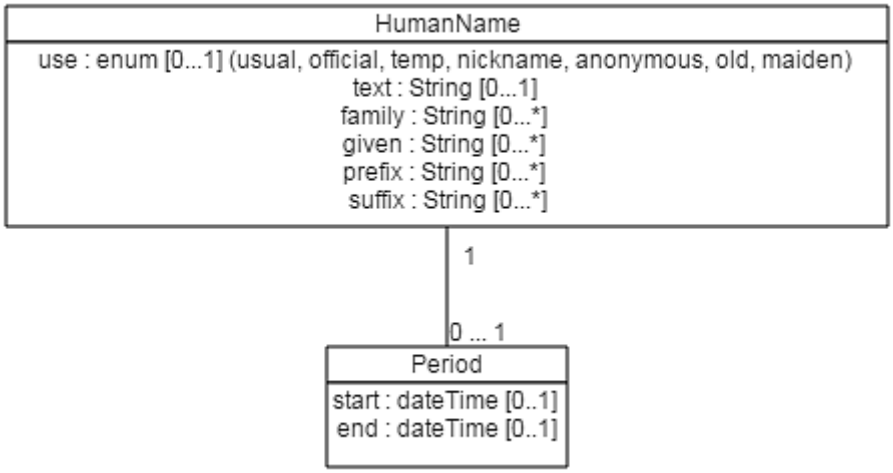
Oma raamatus „Enterprise Patterns and MDA“ kirjeldavad Arlow ja Neustadt täpsemat ja mitmekülgsemat mudelit (Joonis 3). Nende mudel sisaldab lisaks üldlevinud ees-, keskmisele ja perekonnanimele ning ees- ja järelliidetele ka nime kehtivuse algus- ja lõpukuupäeva ning kasutuseesmärki. Samuti kuulub nende nimemudeli juurde atribuut *preferredName*, mis tähistab nime, mida inimene ise igapäevaselt kasutada eelistab. [14] Näiteks mitme eesnimega isik võib eelistada kasutada ükskõik missugust oma eesnimedest, mitte tingimata esimest, või hoopis hüüdnime. Nende mudeli suurimaks puuduseks on see, et nime kehtivuse lõpukuupäev on kohustuslik atribuut. Lõpukuupäeval saab olla kindel väärtus ainult siis, kui on teada, et nimi mingist ajahetkest alates enam ei kehti, näiteks naise neiupõlvenimi peale abiellumist. Isegi inimese surma korral jääb tema nimi kehtima. Seega peaks lõpukuupäev olema mittekohustuslik.

Nende mudeli teiseks puuduseks on kohustuslik perekonnanime väli, mis piirab mudeli kasutamist näiteks juhul, kui teada on ainult eesnimi.



Joonis 3. Arlow ja Neustadt'i isikunime mudel

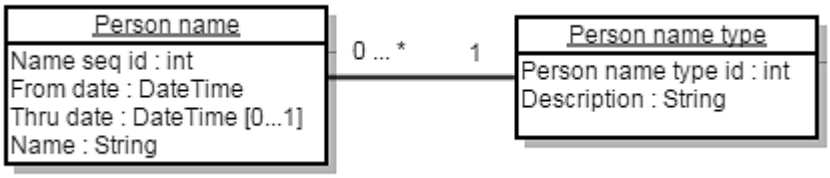
Arlow ja Neustadt mudeliga väga sarnane on ka FHIR (Fast Healthcare Interoperability Resources) mudel [2] (Joonis 4). Erinevalt eelmisest mudelist on FHIR'i omal nime kehtivuse aeg viidud eraldi olemiks *Period*, mille atribuutideks on algus- ja lõpuaeg. Nii olem *Period* kui selle atribuudid on mittekohustuslikud. See võimaldab nime kehtivusperioodi märkimata jätta, kui see spetsiaalselt eraldi määratud ei ole. Lisaks on FHIR mudelis atribuut *text* täisnime hoidmiseks. Puuduvad *middleName* ja *preferredName*, nime hoitakse atribuutides *given* ja *family*. Ükski atribuutidest ei ole ka kohustuslik, mis tähendab seda, et mudelit saab kasutada näiteks ainult ees- või perekonnanime hoidmiseks.



Joonis 4. FHIR isikunime mudel

Kuigi kõige laiemalt levinud nimemudel koosneb eelnevalt defineeritud nimeosadest, ei ole see ainus. Len Silverston kirjeldab oma raamatus „The Data Model Resource Book" nime mudelit, kus nime osade jaoks ei ole antud eraldi atribuute, vaid iga osa on omaette olem ning nimi on nimeosade kogum. Igal nimeosal on kindel tüüp, mis näitab, missuguse osaga tegu

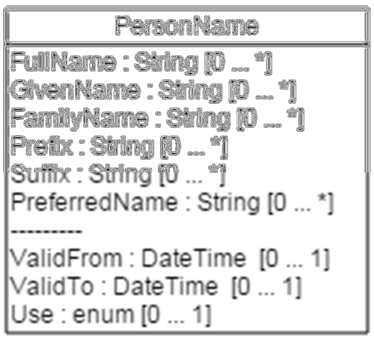
on. Lisaks on igal osal järjekorranumber, mille täpne eesmärk jääb selgusetuks, kuid mis võiks tähistada nii nime elemendi asukohta terves nimes, või ka ainult ühe nimeosa piires. Silverstoni mudel võimaldab nime kujutada väga paindlikult ja kasutada ühte ja sama mudelit väga erinevate nimede korral. Samas soovitab Silverston mudelit kasutada pigem nimede ajaloo talletamiseks, kui üldiseks kasutamiseks. [15] Põhjuseks on arvatavasti see, et nime kasutamiseks tuleks iga pöördumisel käia läbi kogu nimeosade kogum, välja võtta hetkel kehtivad osad ning need õigesse järjekorda seada.



Joonis 5. Silverstoni isikunime mudel

3.2 Mudeli valik

Eelpool kirjeldatud mudelitest kõige täielikumad on FHIR, Silverstoni ja Arlow omad, kuna nendes eristatakse lisaks nime osadele ka nime tüüpi ning need võimaldavad salvestada ajavahemikku, mil nimi on kehtiv. Töö praktilise osa teostamiseks valitud nimemudel (Joonis 6) on kombinatsioon Arlow ja FHIR mudelitest. Silverstoni mudelit ei valitud, kuna see on sobivam nimede ajaloo hoidmiseks, hetkel kehtiva nime haldamiseks ning sellega tegevuste tegemiseks on see liiga keeruline. Mudel valiti kaalutlusega, et seda oleks hiljem lihtne tegelikus rakenduses kasutada, seetõttu sisaldab see atribuute, mis antud töö jaoks on ebavajalikud. Mittekasutatavad atribuudid on joonisel eraldatud joonega.



Joonis 6. Töös kasutatav isikunime mudel

Väli *GivenName* hoiab eesnime ehk isikule (sünnil) antud nime, *FamilyName* hoiab perekonnanime ehk vanematelt päritud nime. Kuna töös võrdsustatakse keskmine nimi eesnimega, ei ole tarvis seda eraldi hoida ja puudub vajadus eraldi välja järele. Juhtude jaoks, kus isikul on mitu eesnime, millest ta üldiselt kasutab ainult ühte-kahte, on mudelis kasutusel Arlow ja Neustadti mudeli atribuut *PreferredName*. Atribuudid *Prefix* ja *Suffix* hoiavad vastavalt ees- ja järelliiteid, atribuudis *FullName* hoitakse tervet sisestatud nime. Järgnevalt (Tabel 4) on toodud Arlow ja Neustadti ning FHIR mudelite atribuudid, kus töös kasutatavasse mudelisse valitud väljad on tähistatud helehalliga. Valitud mudelisse lisati ka atribuudid *ValidFrom* ja *ValidTo*, samuti *Use* nime tüübi määramiseks. *Use* andmetüübiks valiti mudelis *enum*, andmetüüp, mille võimalikud väärtused on nimelised konstandid. [16] Antud töös on *Use* väärtusteks valitud *Official*, *Nickname* ja *Artist*, kuid olenevalt teeki kasutava rakenduse äriloogikast võivad need olla ükskõik missugused.

Tabel 4. Arlow ja Neustadi ning FHIR isikunime mudelite võrdlus

Nime osa	Arlow & Neustadt	FHIR	Valitud mudeli atribuudi nimi
Täisnimi	-	text	FullName
Eesliide	prefix	prefix	Prefix
Eesnimi	givenName	given	GivenName
Keskmine nimi	middleName	märgitakse „given“ alla	
Perenimi	familyName	family	FamilyName
Järelliide	suffix	suffix	Suffix
Hüüdnimi/eelistatud nimi	preferredName	kasutatav väli teadmata, use = „nickname“	PreferredName
Alates	validFrom	period (komposiitandmetüüp, koosneb algus ja lõpu-kuupäevast, mõlema andmetüübiks on *datetime*)	ValidFrom
Kuni	validTo		ValidTo
Kasutus	use	use	Use

4. Vormindusnotatsioon

Teiseks käesoleva töö põhieesmärgiks oli välja töötada notatsioon, millega saaks nime soovitud kujul vormindada. Selleks oli vaja leida erinevad kasutusjuhud, mida toetama peab ning nende põhjal leida nõuded notatsiooni süntaksile. Samuti oli vaja välja valida sobivad tähised, kvantorid ja süntaks ise. Järgnevalt räägitakse kõigist neist lähemalt.

4.1 Kasutusjuhud

Selleks, et notatsioon korralikult toimiks ja sellest kasu oleks, peab see võimaldama vormindada nime enamike üldiste kasutusjuhtude korral. Järgnevalt on toodud mõned üldlevinud nime vormindamise kujud. Kõik kasutusjuhud on illustreeritud väljamõeldud nimedega.

Kasutusjuht: Eesnimi + perekonnanimi
Nimi: Mari von Krahl
Näide: Mari von Krahl

Kasutusjuht: Ainult üks osa mitmest eesnimest + perekonnanimi
Nimi: Mari Julia von Krahl
Näide: Mari von Krahl

Kasutusjuht: Initsiaal + perekonnanimi
Nimi: Mari Julia von Krahl
Näide: M. von Krahl

Kasutusjuht: Kõik initsiaalid + perekonnanimi
Nimi: Mari Julia von Krahl
Näide: M. J. von Krahl

Kasutusjuht: Perekonnanimi, eesnimed
Nimi: Mari Julia von Krahl
Näide: von Krahl, Mari Julia

Kasutusjuht: Igapäevaselt kasutatav nimi + perekonnanimi
Nimi: Eugenia Liisa Kadakas (igapäevaselt kasutab nime Liisa)
Näide: Liisa Kadakas

Nõuded, mida süsteem peab oskama teha:

- Näidata täisnime, s.t mitme eesnime puhul kõiki
- Näidata osa täisnimest, s.t mitme eesnime puhul ühte või mõnda
- Näidata initsiaale
- Näidata kõigi eesnimede asemel eelistatud nime
- Näidata nimeosasid erinevates järjestustes, näiteks {perenimi}, {eesnimi}
- Võimaldada lisada punkte, komasid ja teisi märke nii iga nime osa juurde kui ka neist eraldi (M**.** J**.** versus von Krahl**,** (mitte von**,** Krahl**,**))
- Näidata nime osasid teistest osadest sõltumatult suurte (ja väikeste) tähtedega
- Näidata kõiki nime osasid.

4.2 Tähised

Notatsiooni jaoks sobivate tähiste leidmisel võeti aluseks printsiibid, et tähis peab:

- olema loogiline ja intuitiivne,
- väljendama oma otstarvet.

Kõige lihtsam ja selgem moodus tähiste saamiseks oli võtta valitud nimemudeli atribuutide esitähed (Tabel 5). Eelistatud nime jaoks eraldi tähist ei ole, seda saab näidata eesnime asemel.

Tabel 5. Formaaditähised

Nime osa	GivenName	FamilyName	Prefix	Suffix
Tähis	G	F	P	S

Töö tegemise käigus oli vaja lahendada probleem, kuidas eristada märke, mis korduvad iga nimeosa elemendi juures, näiteks punktid initsiaalide järel, märkidest, mis asuvad terve nimeosa ees või lõpus. Selleks võeti kasutusele lainelised sulud {}. Kõik märgid, mis asuvad sulgude sees, korduvad iga elemendi juures, sulgudest välja jäävad märgid korduvad ainult

ühe korra nimeosade ees vahel või järel, kohas kuhu nad märgitud on. Näiteks, et saada nimest Anna Greete Mets eesnimede initsiaale kujul **„A. G.“**, tuleks kirjutada **„{G.}“** (jutumärgid on osa formaadist).

Kuna inimesel võib ühetüübilisi nimesid olla mitu, näiteks Eestis kuni 3 eesnime, oli tarvis meetodit, millega saaks näidata ainult mõnda nimedest. Selleks kasutatakse tähise järele asetatavat kvantorit (Tabel 6), mis märgitakse kujul *m*:*n* või *n*. *N* määrab, mitut nime näidata, *m* aga seda, mitu nime algusest näitamata jätta. Näiteks kvantor 2:1 korral näidatakse kolmandat nime, kui see olemas on. Kvantori võib ka määramata jätta, sel juhul näidatakse kõiki antud tüübiga nimesid. Erandiks on eesnimi, kus näidatakse kvantor määramata jätmisel eelistatud nime, kui see on defineeritud. Kõiki nimesid näidatakse ka kvantori * kasutamisel, olenemata nimetüübist. Järgnevas tabelis (Tabel 6) esitatakse erinevad kvantorid, näidete illustreerimiseks kasutatakse nime Anna Greete Mets, kus eesnimi on Anna Greete ja on defineeritud eelistatud nimi Greete.

Tabel 6. Formaaditähiste kvantorid

Formaadi-tähis kvantoriga	Kirjeldus	Näide	Näitetulemus
{X}	Kõik nimed või eesnime puhul eelistatud nimi	{G}	G
{X}*	Kõik nimed	{G.}*	A. G.
{X}n	*n* esimest nime	{G.}1	A.
{X}m:n	*n* nime alustades nimest indeksiga *m*	{G.}1:1	G.

Mõnikord võib tekkida vajadus näidata nimesid läbivate suur- või väiketähtedega või initsiaalina (Tabel 7). Läbivaid suurtähti võib tarvis minna näiteks perekonnanime rõhutamiseks. Kuigi tegu on elementaarse stringitöötlusega, on mugavam seda teha nime vormindamise ajal, kuna saab vormindada iga nimeosa eraldi. Terve nimeosa väikeste tähtedega näitamiseks kasutatakse formaaditähist {xx}, läbivate suurtähtede jaoks formaaditähist {XX}. Tähis {Xx} korral nimes tähtede tõstu ei muudeta ja nimesid näidatakse nii nagu nad sisestatud on.

Tabel 7. Formaaditähiste arv ja suur vs väike täht

Formaadi-tähis	Kirjeldus	Näide	Näitetulemus
{x}	Initsiaal, väikese tähega	{f}	m
{X}	Initsiaal, suure tähega	{F}	M
{xx}	Terve nimeosa, väikeste tähtedega	{ff}	mets
{Xx}	Terve nimeosa, esimene täht suur	{Ff}	Mets
{XX}	Terve nimeosa, läbivalt suured tähed	{FF}	METS

5. Vormindaja

Vormindaja ehk *formatter* on programm või meetod, mis muudab sisendi etteantud kujuga väljundiks. Antud töö kolmandaks põhieesmärgiks oli realiseerida algoritm, mis vormindaks isikunime, kasutades eelmises peatükis kirjeldatud vormindusnotatsiooni. Selles peatükis kirjeldatakse valminud vormindajat lähemalt. Vormindaja kood on täies mahus toodud Lisas 1.

Vormindaja koos järgmises peatükis kirjeldatava isikunime parseriga ning isikunime mudeliga on vormistatud C# teegina.

Vormindajal on üks avalik staatiline klass *Formatter*, millel on omakorda üks avalik meetod ning neli privaatset meetodit. Järgnevalt on esitatud vormindaja meetodid koos sisend- ja väljundparameetrite, nähtavuse ning meetodite kirjeldustega.

Tabel 8. Vormindaja meetodid

Meetod	Nähtavus	Sisendparameetrid	Tagastav andmetüüp	Kirjeldus
Format	Avalik	Name *name*, String *formatString*	String	Peamine meetod, tagastab vormindatud nime.
parseFormatString	Privaatne	String *formatString*	IEnumerable <String>	Jagab mitmest nimeosast koosneva formaadi väikse-mateks osadeks
formatNamePart	Privaatne	Name *name*, String *formatPart*	String	Vormindab antud formaadijupile vastava nimeosa
getNamePartBy-FormatString	Privaatne	Name *name*, String *identifier*, bool *takeOnlyFullName*	String	Tagastab sobiva nimeosa vastavalt sisendparameetritele

Meetod	Nähtavus	Sisendparameetrid	Tagastav andmetüüp	Kirjeldus
getCaseType	Privaatne	String *identifier*	CaseType (enum)	Tagastab vastavalt antud formaadi-jupile *identifier* soovitava tõstu (Upper | Lower | Normal)

5.1 Kogu nime vormindamine

Nime vormindamisel antakse meetodi *Format* parameetriteks nimi, mida vormindada soovitakse ning soovitavat tulemust kirjeldav tekstiline formaat. Esimese sammuna jagatakse formaadistring osadeks meetodis *parseFormatString*. Selleks kasutatakse regulaaravaldist

(\{.*?\}(?:*|\d*)?(?:(?<!\})\:(?:*|\d*))?)

mis koosneb kolmest osast:

- \{.*?\} – leiab loogelised sulud ja kõik nende sees leiduva esimese } esinemiseni
- (?:*|\d*)? – ühe * või järjestikuse jada numbreid üks või null korda.
- (?:(?<!\})\:(?:*|\d*))?
    - (?<!\})\: – leiab **:**, mis ei järgne otse loogelisele lõppsulule
    - (?:*|\d*) – ning millele järgneb üks * või jada numbritest

?: sulgude alguses tähendab, et tegu on *non-capturing group*'iga ehk sulgudesse püütud väärtust ei saa hiljem edasi kasutada. Seda on tarvis, et leida ainult terved formaadiosad kujul {xx}*m*:*n* ning mitte püüda eraldi gruppidesse ka nende alamosasid. Meetod tagastab leitud formaadijupid.

```
private static IEnumerable<String> parseFormatString(String formatString)
{
    Regex regex = new Regex(@"(\{.*?\}(?:\*|\d*)?(?:(?<!\})\:(?:\*|\d*))?)",
RegexOptions.IgnoreCase);

    return regex.Matches(formatString)
                .Cast<Match>()
                .Select(x => x.Value.Trim())
                .Where(x => !String.IsNullOrEmpty(x));
```

```
}
```

Järgmiseks antakse iga leitud formaadiosa parameetrina meetodisse *formatNamePart*, esimeseks parameetriks antakse vormindatav nimi. *formatNamePart* tagastab vormindatud nime, millega asendatakse algses formaadistringis vastav formaadiosa. Näiteks kui formaadistring on {Gg} {Ff}, siis asendatakse {Gg} vormindatud eesnimega ning {Ff} perekonnanimega.

```
foreach (var formatPart in formatParts)
{
    var namePart = formatNamePart(name, formatPart);
    formatString = formatString.Replace(formatPart, namePart);
}
```

Kui kõik formaadiosad on asendatud, tagastab *Format* vormindatud nime.

5.2 Nimeosa vormindamine

Tavaliselt saab regulaaravaldiste sulgudesse püütud mustrigruppe kasutada neile indeksiga viidates, siin pannakse gruppidele nimed, et neile hiljem lihtsam viidata oleks. Nimeosa vormindamisel leitakse formaadiosast viis nimelist gruppi järgneva regulaaravaldisega.

```
\{(?<Before>[^PGFS]*)(?<Identifier>[PGFS]*)(?<After>[^\}]*)\}
(?:(?<Skip>\d*?(?=\:))\:)?(?<Take>[\d\*]*)
```

- `\{` – algussulg üks kord
- `(?<Before>[^PGFS]*)` – kõik järjestikused tähemärgid, mis ei ole kuulu hulka {P,G,F,S} salvestatakse gruppi nimega *Before*. Leitakse võimalikult vähe hulgale eelnevaid märke.
- `(?<Identifier>[PGFS]*)` – kõik järjestikused tähemärgid, mis kuuluvad samasse hulka, mis eelmises grupis välistati, salvestatakse gruppi nimega *Identifier*
- `(?<After>[^\}]*)` – kõik *Identifier* grupile järgnevad tähemärgid, mis ei ole lõpusulg, salvestatakse gruppi *After*
- `\}` – lõpusulg üks kord
- `(?:(?<Skip>\d*?(?=\:))\:)?` – kõik vahetult sulgudele järgnevad numbrid, mis eelnevad koolonile, salvestatakse gruppi *Skip*. Kui koolonit pole, jääb grupp tühjaks.
- `(?<Take>[\d\*]*)` – kõik ülejäänud numbrid või * salvestatakse gruppi *Take*

Kui formaadijupist on puudu formaaditähis (*Identifier*), siis selle osa vormindamine lõpetatakse ja *formatNamePart* tagastab *null*'i ehk tühiväärtuse. Vastasel juhul leitakse vormindatav nimeosa abimeetodi *getNamePartByFormatString* abil formaaditähise esimese tähe järgi.

```
var namePart = getNamePartByFormatString(name, identifier,
    !String.IsNullOrEmpty(take));
```

Eesnime puhul võib soovitavaks nimeosaks olla nii eesnimi ise kui ka eelistatud nimi. Abimeetod tagastab eelistatud nime ainult sellisel juhul, kui see on olemas, ja kui formaadis ei ole eraldi märgitud, mitut nimeosa elementi näidata soovitakse. Kõigi teiste nimeosade puhul tagastab abimeetod formaaditähisele vastava nimeosa, olenemata sellest, kas sellest tahetakse näidata kõiki elemente või mitte.

Kuna nimeosa hoitakse stringina, kus on koos kõik antud osa elemendid, näiteks 2 või 3 eesnime, lõhutakse see string eraldaja kohalt tükkideks. Ees- ja perekonnanimede puhul on eraldajaks tühik, ees- ja järelliidete puhul on eraldajaks koma, sest liited võivad koosneda mitmest osast, näiteks *Count of London*.

```
IEnumerable<String> names = null;
var namePartSeparator = namePart.IndexOf(',') > 0 ?
    new char[] { ',' } :
    new char[] { ' ' };
names = namePart.Split(namePartSeparator, StringSplitOptions.RemoveEmptyEntries);
```

Peale korrektse nimeosa leidmist on järgmiseks sammuks sellest üleliigsete elementide eemaldamine. Selleks kontrollitakse kõigepealt, kas regulaaravaldise grupi *Skip* väärtus on numbriline. Kui on, parsitakse grupi väärtus täisarvuks ja eemaldatakse nimeelementide massiivi algusest vastav arv elemente:

```
if (!String.IsNullOrWhiteSpace(skip) && skip.ToCharArray().All(c => char.IsDigit(c)))
{
    var amountToSkip = int.Parse(skip);
    names = names.Skip(amountToSkip);
}
```

Seejärel tehakse samasugune protsess läbi grupiga *Take*. Numbrilise väärtuse korral parsitakse see täisarvuks ning võetakse vastav arv elemente nimeelementide massiivi algusest.

```
if (!String.IsNullOrWhiteSpace(take) && take.ToCharArray().All(c => char.IsDigit(c)))
{
    var count = int.Parse(take);
    names = names.Take(count);
}
```

Kui formaaditähis on ühetäheline, s.t tahetakse näidata initsiaali, võetakse järgmisena igast nimest ainult esimene täht. Kuna sidekriipsuga nime initsiaalina näitamise korral peavad mõlemad nimepooled alles jääma [17], lõigatakse sidekriipsu sisaldava nimeelemendi korral antud element kõigepealt sidekriipsu kohalt osadeks, võetakse igast osast esimene täht ning liidetakse saadud esitähed jälle sidekriipsude abil kokku.

```
if (identifier.Length == 1)
{
    names = names.Select(
        n => n.IndexOf('-') <= 0 ?
            n.Substring(0, 1) :
            String.Join("-", n.Split('-').Select(n2 => n2.Substring(0, 1)))
        );
}
```

Järgmise sammuna leitakse formaaditähise järgi soovitav tõst, kasutades abimeetodit *getCaseType* (kasutatakse enumit CaseType võimalike väärtustega Normal, Upper, Lower). Tõstu leidmine toimub nii:

- Ühetähelise formaaditähise puhul, kui tähis on suur täht {X}, tagastab meetod *CaseType.Upper*, väikesetähelise formaaditähise {x} puhul *CaseType.Lower*.
- Kahetähelise tähise korral tagastab meetod vastavalt:
    - {xx} ja {xX} – *CaseType.Lower*
    - {XX} – *CaseType.Upper*
    - {Xx} – *CaseType.Normal*

Kogu nimeosa konverteeritakse vajadusel kas läbivateks suur- või väiketähtedeks.

Viimase sammuna liidetakse vormindatud nimeosa igale elemendile juurde muud märgid, mis looksulgude sees formaaditähise ümber olid. Kõige tavalisem võimalikest märkidest on arvatavasti punkti initsiaalide lõpus. Iga elemendi ette liidetakse regulaaravaldise grupi *Before* väärtus ning lõppu *After* väärtus.

```
names = names.Select(n => String.Format("{0}{1}{2}",
            beforeIdentifier,
            n,
            afterIdentifier)
        );
```

Kõige lõpuks liidetakse kõik massiivi elemendid tühikutega kokku ja tagastatakse vormindatud nimi stringi kujul.

```
return String.Join(" ", names).Trim(' ', ',');
```

Vormindajat saab hõlpsasti laiendada, et see toetaks ka muudes riikides kasutusel olevaid nimeosasid. Selleks tuleks regulaaravaldistes lubada uued vormindustähised ning meetodis *getNamePartByFormatString* defineerida vastavatele tähistele vastavad nimeosad.

6. Parser

Selles peatükis kirjeldatakse töö neljanda põhieesmärgina valminud isikunime parserit. Parser ehk süntaksianalüsaator on tarkvarakomponent, mis otsib andmetest teatud stringe. [18]

Erinevaid nimeparsereid on palju, enamikes populaarsemates programmeerimiskeelte tarvis on loodud vähemalt üks. Antud töös valminud parseri idee ei kuulu täielikult käesoleva töö autorile, vaid on kombinatsioon mitmetest töö käigus uuritud nime parsimise teekidest koos autoripoolselt täienduste ja muudatustega. Ühtegi olemasolevat teeki ei ole üritatud kopeerida. Teadlikult on üle võetud ühe JavaScriptis kirjutatud parseri [19] üldine tööpõhimõte – kõigepealt eemaldatakse ees- ja järelliited, seejärel perekonnanimi ning ülejääv osa võetakse eesnimeks. Samuti on sellest parserist võetud idee kasutada liidete tuvastamiseks tuntud liidete nimekirju.

Antud töös on liited kirjeldatud ressursifailis *Data.resx*. Ressursifail on XML formaadis fail, kus saab hoida nii stringe kui objekte, näiteks pilte, ning neid programselt kasutada. Ressursifaili suureks eeliseks on, et seda saab tekstiredaktoriga avada ja muuta. Teiseks eeliseks on võimalus kasutada lokaliseeritud versioone – muutuja väärtus sõltub keele-seadetest. [20] Kuigi seda võimalust käesolevas töös ei rakendata, saaks seda töö edasiarendustes kasutada ka osade liidete kuvamiseks kasutaja valitud keeles, näiteks eesti keeles näidata nime ees „Hr.“, inglise keeles „Mr.“.

Parseri kood on esitatud täies mahus Lisas 2. Parser koosneb ühest staatilisest klassist *Parser*, millel on üks avalik ja kahest privaatset meetodit, järgnevalt (Tabel 9) on esitatud meetodite nimed, nähtavus, kirjeldused, tagastatavad andmetüübid ning sisend- ja väljundparameetrid. Nimestringi parsimiseks antakse see parameetrina meetodisse *Parse*.

Tabel 9. Parseri meetodid

Meetod	Nähtavus	Sisend-parameetrid	Väljund-parameetrid	Tagastatav andme-tüüp	Kirjeldus
Parse	Avalik	String *nameString*	-	Name	Peamine meetod nime parsimiseks

Meetod	Nähtavus	Sisend-parameetrid	Väljund-parameetrid	Tagastatav andme-tüüp	Kirjeldus
tryRemove Family-Name	Privaatne	String *nameString*	String *foundFamily-Name*, String *nameString-Without-FamilyName*	boolean	Abimeetod perekonnanime eraldamiseks koos kõigi selle osadega. Tagastab leitud nime ja järelejäänud osa
tryRemove Prefixes	Privaatne	String *nameString*	List<String> *foundPrefixes*, String *nameString - Without-Prefixes*	boolean	Abimeetodid nimestringilt ees- ja järelliidete eemaldamiseks, tagastab *List*'i leitud liidetega ja algse nimestringi ilma liideteta
tryRemove Suffixes	Privaatne	String *nameString*	List<String> *foundSuffixes*, String *nameString - Without-Suffixes*	boolean	

6.1 Eesliited

Parsimise esimeseks sammuks on eesliidete eemaldamine. Selleks kasutatakse abimeetodit *tryRemovePrefixes*, mis tagastab *List*'i eemaldatud eesliidetega ja algse nimestringi ilma nendeta. Eesliidete leidmiseks ja eemaldamiseks võetakse ressursifailis *Data.resx* asuv muutuja *Prefixes*, ja lõigatakse see komade kohalt tükkideks. Nii saadakse nimekiri eelnevalt defineeritud eesliidetest. Antud töös on nendeks liideteks: *Miss, Ms, Mr, Mister, Mrs, Dr, Doctor, Prof, Professor, Sir, Dame, Lord Justice, Lord, Lady, Prl, Preili, Pr, Proua, Hr, Härra, Frau, Fraulein, Herr, Hon, The Rt Hon.* [7] Kasutatav nimekiri ei ole kaugeltki täielik,

kuid käesoleva töö eesmärgiks ei olnud käsitletavates riikides esinevate erinevat liiki tiitlite põhjalik analüüs. Vajadusel saab ressursifaili liiteid lisada.

Seni kuni nimestring algab mõne defineeritud liitega, eemaldatakse see nimestringi algusest ning lisatakse leitud eesliidete *List*'i.

```
while (prefixes.Any(p => nameString.StartsWith(p)))
{
    foreach (var prefix in prefixes)
    {
        if (nameString.IndexOf(prefix) == 0)
        {
            var takeLength = (nameString.IndexOf('.') == prefix.Length ?
                prefix.Length + 1 :
                prefix.Length);
            foundPrefixes.Add(nameString.Substring(0, takeLength).Trim(_charsToTrim));
            nameString = nameString.Substring(takeLength).Trim(_charsToTrim);
        }
    }
}
```

Kui võimalikult palju eelnevalt teadaolevaid eesliiteid on kätte saadud, kontrollitakse ega nimestringi alguses ei ole punktiga lõppevaid tundmatuid elemente. Juhul kui mõni leitakse, lisatakse ka see teiste leitud eesliidete juurde ning eemaldatakse nimestringist.

```
var nameParts = input.Split(' ');
var names = new Stack<String>(nameParts.Reverse());

while (names.Peek().IndexOf('.') > 0)
{
    foundPrefixes.Add(names.Pop());
}
```

Lõpuks seatakse väljundparameetri väärtuseks nimestring, kust on eemaldatud kõik leitud eesliited.

```
nameStringWithoutPrefixes = String.Join(" ", names);
```

6.2 Järelliited

Järgmisena eemaldatakse nimest järelliited. Selleks vaadatakse kõigepealt, kas nimes on mõni koma. Kui on, võetakse nimestringi lõpp alates esimesest komast, jagatakse komade kohalt osadeks ning lisatakse kõik nii saadud elemendid leitud järelliidete *List*'i.

```
foundSuffixes = new List<String>();
var suffixStartIndex = nameString.IndexOf(',');
if (suffixStartIndex >= 0)
{
    foundSuffixes = nameString.Substring(suffixStartIndex)
                        .Split(',')
                        .Where(s => !String.IsNullOrWhiteSpace(s))
                        .Select(s => s.Trim())
                        .ToList();

    nameString = nameString.Substring(0, suffixStartIndex).Trim(_charsToTrim);
}
```

Arvestatakse ka võimalusega, et järelliide ei ole komaga eraldatud. Selleks kasutatakse sarnaselt eesliidetega ressursifaili *Data.resx*, kus muutujas *Suffixes* on eelnevalt defineeritud järelliited (*Esq, PhD, MSc, BSc, I , II, III, IV, V, VI, VII, VIII, IX, X, XI, XII, XIII, XIV, XV, Sr, Jr, Senior, Junior, Seenior, Juunior*). Need lõigatakse komade kohalt osadeks ning saadud nimekirjaga hakatakse võrdlema nimestringi lõppu. Seni kuni lõpp kattub mõne nimekirjas oleva liitega, eemaldatakse liide nimestringist ning lisatakse leitud järelliidete ette. Lisatakse just ette, et säilitada liidete õiget järjekorda.

```
var definedSuffixes = PersonNames.Data.Suffixes.Split(',');
while (definedSuffixes.Any(s => nameString.EndsWith(s)))
{
    foreach (var suffix in definedSuffixes)
    {
        if (nameString.EndsWith(suffix))
        {
            var suffixIndex = nameString.LastIndexOf(suffix);
            foundSuffixes.Insert(0, nameString
                                        .Substring(suffixIndex).Trim(_charsToTrim));

            nameString = nameString.Substring(0, suffixIndex).Trim(_charsToTrim);
        }
    }
}
```

Kuna kõigepealt eemaldatakse nime lõpust komadega eraldatud järelliited ja alles seejärel kontrollitakse vastavust teadaolevate liidete nimekirjale, ei suuda parser kätte saada järelliiteid, mis asuvad komaga eraldatud liite järel, aga ise komaga eraldatud ei ole, näiteks John Smith, PhD MSc. Selle näite puhul leiab parser ainult ühe liite: PhD MSc.

6.3 Perekonnanimi

Kui ees- ja järelliited on eemaldatud, eeldatakse et järele on jäänud ainult nimi ise. Kuna perekonnanime puhul on mitu juhtu, mida tuleb kontrollida, toimub ka selle eraldamine eraldi

abimeetodis *tryRemoveFamilyName*. Kogu nimi (k.a eesnimi) jagatakse tühikute kohalt osadeks ning pannakse pinusse (*Stack*).

```
var nameParts = nameString.Split(' ');
var names = new Stack<String>(nameParts);
```

Pinu toimib LIFO ehk *Last In, First Out* põhimõttel. Seejärel hakatakse pinust ükshaaval elemente eemaldama.

Kõige lihtsamal juhul on perenimi üheosaline, st ei sisalda tühikuid, seega võetakse esialgu vaikimisi perekonnanimeks viimane pinusse sisestatud element.

```
var familyNameQueue = new Stack<String>();
familyNameQueue.Push(names.Pop());
```

Seejärel kontrollitakse esiteks ega järgmine element pinus ei ole perekonnanimes esineda võiv liitsõna (*und, and, genannt*). Kui selline sõna leidub, lisatakse leitud perekonnanimele see liitsõna ja pinus sellele eelnev sõna.

```
var familyNameJoins = PersonNames.Data.FamilyNameJoins.Split(',');
if (names.Count > 0 &&
    familyNameJoins.Any(j => j.Equals(names.Peek(),
        StringComparison.InvariantCultureIgnoreCase)))
{
    familyNameQueue.Push(names.Pop());
    familyNameQueue.Push(names.Pop());
}
```

Järgmisena kontrollitakse nn „aadliliiteid“ (*von, der, zum* jne), nende esinemisel lisatakse nad samuti järjest leitud perekonnanimele, kuni järgmine pinus olev sõna ei ole defineeritud liide.

```
var nobilityPrefixes = PersonNames.Data.NobilityParticles.Split(',');
while (names.Count > 0 &&
    nobilityPrefixes.Any(n => n.Equals(names.Peek(),
        StringComparison.InvariantCultureIgnoreCase)))
{
    familyNameQueue.Push(names.Pop());
}
```

Viimasena kontrollitakse, ega järgmine sõna (st aadliliitele järgnev) ei ole nimekirjas *DefinedWordsInFamilyName*, mis nagu nimigi ütleb, sisaldab teadaolevaid sõnu, mis võivad perekonnanimes esineda.

```
var wordsInFamilyName = PersonNames.Data.DefinedWordsInFamilyName.Split(',');
if (names.Count > 0 &&
    wordsInFamilyName.Any(t => t.Equals(names.Peek(),
        StringComparison.InvariantCultureIgnoreCase)))
{
```

```
    familyNameQueue.Push(names.Pop());
}
```

Lõpuks liidetakse kõik leitud perekonnanime elemendid kokku ning seatakse väljundparameetrite väärtuseks leitud nimi ning sisendiks antud nimestring ilma perenimeta.

```
familyName = String.Join(" ", familyNameQueue);
nameStringWithoutFamilyName = String.Join(" ", names.Reverse());
```

6.4 Eesnimi

Parseri peameetodis *Parse* anti algne nimestring kõigepealt sisendparameetrina eesliidete eraldamise meetodisse. Seejärel seati algse nimestringi väärtuseks meetodi väljund-parameeter, milleks oli see sama string ilma eesliideteta.

```
tryRemovePrefixes(nameString, out foundPrefixes, out nameString);
```

Sama tehnikat korrati ka järelliidete ja perekonnanime eemaldamisel. Peale kõigi teist nimeosade eemaldamist peaks olema alles jäänud ainult eesnimi.

```
name.GivenName = nameString;
```

Peale parseri valmimist tuli välja, et C# jaoks oli üks nime parsimise teek (CSharp-Name-Parser) [21] juba olemas. Selle implementatsioon erineb käesolevas töös tehtust päris palju. Järgmises peatükis testitakse valminud parserit ning vormindajat. Samuti võrreldakse valminud parseri ja CSharp-Name-Parser'i testide tulemusi.

7. Testimine

Parseri ja vormindaja testimiseks koostati unit testid, mis kataksid võimalikult palju mõlema komponendi funktsionaalsusest. Testid koostati parseri ja vormindaja jaoks eraldi, võttes esimese jaoks aluseks nimekujude analüüsis käigus leitud erinevad nimemustrid ning teise jaoks vormindaja kasutusjuhud ning vormindusnotatsiooni kirjeldused. Testid oma lõplikul kujul koostati peale parseri ja vormindaja valmimist ning neid kasutati komponentide esmaste versioonide testimiseks. Järgmistes alampeatükkides antakse ülevaade testjuhtudest, testidest endist ning testimise tulemustest.

7.1 Testjuhud

„Testjuht on testsisendist, läbiviimistingimustest ning oodatavatest tulemustest koosnev komplekt, mis on loodud kindla sihitusega, näiteks kindla käskude jada läbimiseks või teatud nõudele vastavuse kontrollimiseks.„ [22] Siin peatükis toodus testjuhtudel kindlad läbiviimis-tingimused puuduvad, toodud on ainult sisend, oodatav väljund ning mida testitakse.

Parseri testjuhtude leidmisel võeti aluseks eelnevalt välja toodud nimede mustrid, millest iga mustri kohta leiti vähemalt üks nimi. Juhul kui muster kordus, näiteks igas vaadeldud keeles eksisteerib nimekuju {eesnimi} {perenimi}, võeti kasutati eelistatult Eesti nime. Nimede puhul, kus oluliseks osaks on perekonnanime testimine, jäetakse eesnimi eraldi välja kirjutamata, sest perekonnanime eemaldamisel jääb niigi alles ainult eesnimi. Järgnevas tabelis (Tabel 10) on esitatud mõned testjuhud, kõik testjuhud asuvad Lisas 4.

Tabel 10. Parseri testjuhud

Muster	Nimi	Testitav osa	Oodatav tulemus
{eesnimi}*n* {perenimi}	Anna Mets	Eesnimi	Anna
		Perenimi	Mets
{eesnimi-eesnimi} {perenimi-perenimi}	Jana-Liisa Mutt-Harakas	Eesnimi	Jana-Liisa
		Perenimi	Mutt-Harakas
{eesnimi}*n* {*eesliide/-liited* + perenimi}	Johann *von der* Hagen	Eesnimi	Johann
		Perenimi	*von der* Hagen

{eesnimi}*n* {perenimi und perenimi}	Johann Holm und Schwarzwald	Perenimi	Holm und Schwarzwald
{eesnimi} {perenimi eesliitega Sankt|St.}	Johann Sankt Goar	Perenimi	Sankt Goar
{eesnimi} {perenimi}, *{tiitel}*	Benjamin Disraeli, *Earl of Beaconsfield*	Eesnimi	Benjamin
		Perenimi	Disraeli
		Järelliide (defineerimata, komaga eraldatud)	Earl of Beaconsfield
{tiitel} {eesnimi} {perenimi} *{tiitel}*	*Sir* Thomas Birch *MSc*	Eesliide	Sir
		Eesnimi	Thomas
		Perenimi	Birch
		Järelliide (defineeritud, komaga eraldamata)	MSc

Vormindaja testjuhud koostati selliselt, et oleksid kaetud kõik võimalikud formaadiosa kujud. Testid on jagatud nimeosade kaupa.

Eesnime testjuhtudes (Tabel 11) sisaldub lisaks nime enda testimisele ka kvantorite testimine.

Tabel 11. Vormindaja testjuhud eesnimede ja kvantorite kohta

Nimi	Formaat	Mida testitakse	Oodatud tulemus
Anna Mets	{G.}	Initsiaali ühe nime korral	A.
Jaan-Markus Murakas	{G.}	Initsiaali liitnime korral	J-M.
Jaan Markus Murakas	{G.}	Initsiaali mitme nime korra	J. M.
	{Gg}1	Kvantorit	Jaan
	{Gg}1:1	Kvantorit + elemendi vahelejätmist	Markus
- eelistatud nimi Markus	{Gg}	Eelistatud nime näitamist kvantori puudumisel	Markus

Nimi	Formaat	Mida testitakse	Oodatud tulemus
	{Gg}*	Kvantorit * eelistatud nime olemasolul	Jaan Markus
Anna Greete Julia Mets	{Gg}1:*	Kvantorit * + elemendi vahelejätmist rohkem kui 2 nimeelemendi korral	Greete Julia
	{Gg}1:1	Kvantorit + elemendi vahelejätmist, kui tahetakse mingit keskmist elementi	Greete

Perekonnanimede testjuhtudes (Tabel 12) kõiki kvantoreid läbi ei testida, sest kvantorite töö ei sõltu sellest, mis nimeosaga tegemist on ning eesnimede juures neid juba testiti. Kvantoreid kasutatakse perekonnanime testides ainult niipalju, kui on vaja perenime korrektse vormindamise kontrollimiseks.

Tabel 12. Vormindaja testjuhud perekonnanimede kohta

Nimi	Formaat	Mida testitakse	Oodatud tulemus
Jaan Markus Murakas	{F.}	Initsiaali	M.
Johann von der Hagen	{Ff}	Tiitliga perenime	von der Hagen
	{Ff}1	Tiitliga perenimest osa näitamist – ei tohiks näidata ainult esimest tiitlit	von der Hagen

Ees- ja järelliidete testides (Tabel 13) kontrollitakse ainult, kas on võimalik näidata kõiki liiteid korraga või vastupidi ainult osasid neist. Ei kontrollita ainult esimese tähe näitamist, sest see ei ole reaalne kasutusjuht, lisaks on nagunii eelnevatest testidest teada, kas initsiaali näitamine toimib või mitte.

Tabel 13. Vormindaja testjuhud ees- ja järelliidete kohta

Nimi	Formaat	Mida testitakse	Oodatud tulemus
Sir Thomas Birch	{Pp}	Eesliidet	Sir

Nimi	Formaat	Mida testitakse	Oodatud tulemus
Colonel Sir Thomas Birch	{Pp}1	Mitme eesliite puhul ainult ühe näitamist	Colonel
The Rt Hon. Sir Thomas Birch	{Pp}	Mitmeosalise elemendiga eesliidet	The Rt Hon. Sir
	{Pp}1	Mitmeosalise elemendiga eesliitest ühe näitamist	The Rt Hon.
Sir Thomas Birch Jr, PhD, Esq	{Ss,}	Mitme järelliite puhul kõigi näitamist	Jr, PhD, Esq
	{Ss}1	Mitme järelliite puhul esimese näitamist	Jr

Suur- ja väiketähtede testides (Tabel 14) vaadeldakse, kas nimeosa läbivalt suurte või väikeste tähtedega näitamine toimib. Siinkohal ei tehta vahet, missugust nimeosa vormindatakse, kuna see toimub ühtemoodi iga nimeosa puhul.

Tabel 14. Vormindaja testjuhud suur- ja väiketähtede kohta

Nimi	Formaat	Mida testitakse	Oodatud tulemus
Anna Mets	{gg}	Nimeosa läbivalt väikeste tähtedega kuvamist	anna
	{FF}	Nimeosa läbivalt suurte tähtedega kuvamist	METS
	{Ff}	Nimeosa kuvamist sisestatud kujul, s.t muutmata	Mets

Lisaks nimeosade eraldi testimisel koostati testjuhud ka peatüki alguses toodud kasutusjuhtude kohta (Tabel 15).

Tabel 15. Vormindaja testjuhud kasutusjuhtude baasil

Nimi	Formaat	Mida testitakse	Oodatud tulemus
Mari von Krahl	{Gg} {Ff}	Eesnimi + perenimi	Mari von Krahl

Nimi	Formaat	Mida testitakse	Oodatud tulemus
Mari Julia von Krahl	{Gg}1 {Ff}	Üks eesnimi + perenimi	Mari von Krahl
	{G.}1 {Ff}	Üks initsiaal + perenimi	M. von Krahl
	{G.} {Ff}	Kõik initsiaalid + perenimi	M. J. von Krahl
	{Ff}, {Gg}	Perekonnanimi, eesnimed	von Krahl, Mari Julia
Eugenia Liisa Kadakas (igapäevaselt kasutatav nimi Liisa)	{Gg} {Ff}	Igapäevaselt kasutatav nimi + perekonnanimi	Liisa Kadakas

7.2 Testid

Parseri kohta tehti 26 testi, vormindaja kohta 25.

Testide kirjutamiseks kasutati NUnit'it, unit testimise raamistikku .NET keelte jaoks. [23] Visual Studiol on ka endal unit testide raamistik olemas, kuid selle võimalused ei olnud antud projekti jaoks piisavad. Kuna testjuhud on väga sarnased ning erinevused on ainult sisendites ja oodatud tulemustes, tehti nii vormindaja kui parseri jaoks üks testmeetod ning sellele vastav andmeallikas. NUnit hoolitseb selle eest, et meetod käivitataks iga ette antud parameetrite objekti kohta. Visual Studio testraamistiku puuduseks oligi, et see ei võimalda kasutada ühte testi erinevate sisendandmete puhul.

Testide kood on täies mahus toodud Lisas 3.

Testmeetodi andmeallikas loeb testandmed tekstifailist, kus väljad on eraldatud semikoolonitega ning iga rea kohta, mis ei ole tühi ega kommentaar (tähistatud #'ga) võtab reast kindlatel positsioonidel olevad elemendid. Seejärel annab ta iga rea kohta testmeetodisse *TestCaseData* objekti, ootab vastuse ära ning annab järgmise. *TestCaseData* objekti argumentideks on testmeetodi parameetritesse antavad väärtused. Samuti defineeritakse igal objektil oodatav tulemus, testi nimi ning kategooria. NUnit kontrollib ise, kas testmeetodi

tagastava väärtus vastab *TestCaseData* oodatavale väärtusele. Kui ei, siis on test selle testandmete rea kohta läbi kukkunud.

```
public static IEnumerable NamesForParser
{
    get
    {
        var file = System.IO.File.ReadAllLines("parser_test_data.txt");
        foreach (var line in file)
        {
            var parts = line.Split(';');
            if (String.IsNullOrWhiteSpace(line) || parts.Length != 6 ||
                line.StartsWith("#"))
            {
                continue;
            }
            var category = (parts[5].Length > 0 ? parts[5] : "GeneralParser");

            yield return new TestCaseData(parts[0])
                .Returns(new String[] { parts[1], parts[2], parts[3], parts[4] })
                .SetCategory(category)
                .SetName(String.Format("Parse - {0} - {1}", category, parts[0]));
        }
    }
}
```

Testi kategooriat kasutati antud töös testide grupeerimiseks. Visual Studios saab teste kategooria järgi grupeeritult vaadata *Test Explorer* aknas, valides grupeerimistunnuseks *Traits*.

Vormindaja testandmed asuvad failis *format_test_data.txt*, mis peab olema testide projekti juurkaustas. Testandmete väljad on järgmised: *Formaadistring;Oodatav tulemus; Eesnimi;Perenimi;Eesliide;Järelliide;Eelistatud nimi; Testi kategooria;Testi nimi*

Testi parameetriteks on formaadistring ning nime osadest moodustatud *Name* objekt. Testi oodatavaks tulemuseks on teise elemendi väärtus, kategooriaks testi kategooria. Testi nimi on kujul „Format – {kategooria} – {testi nimi}.

```
[TestFixture]
public class FormatterTests
{
    [Test, TestCaseSource(typeof(TestDataFactory), "NamesForFormatter")]
    public String TestFormat(PersonName name, String format)
    {
        return Formatter.Format(name, format);
    }
}
```

Parseri testandmed asuvad analoogselt failis *parser_test_data.txt* ning väljad on: *Parsitav täisnimi;Oodatav eesnimi;Oodatav perekonnanimi;Oodatavad eesliited;Oodatavad järelliited;Testi kategooria*.

Parseri testi parameetriks on parsitav täisnimi, testi enda nimi on kujul „Parse – {kategooria} – {täisnimi}“. Test tagastab neljaosalise massiivi nimeosadega, mida NUnit võrdleb testandmete failist saadud nimeosade massiiviga. Kui massiivi elemendi kattuvad, loetakse test õnnestunuks.

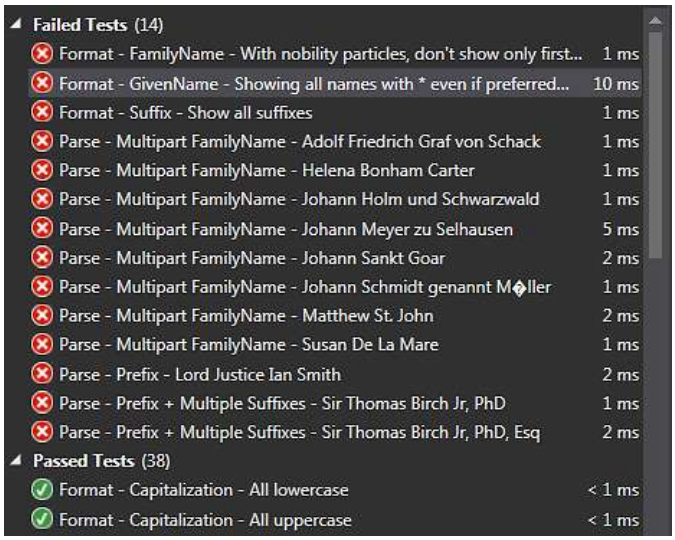
Testis ei võrrelda parsimisel saadud nimeobjekti otse testandmetest ehitatud nimeobjektiga, sest isegi kui kõik atribuudid on võrdsed, ei ole tegu sama objektiga. Teiseks võimalikuks lahenduseks oleks olnud üle kirjutada *PersonName* klassi Equals meetod, kuid see oleks tähendanud, et objektid oleks loetud võrdseteks ka kahe erineva kuid samanimelise isiku korral. Ainult nime parsimisel pole see oluline, kuid *PersonName* klassi tegelikus süsteemis kasutamisel oleks see võinud probleeme tekitada.

7.3 Tulemused

Esimesel testimisel ebaõnnestus 3 vormindaja ning 11 parseri testi. (Joonis 7)

Selgus, et parser ei suuda korrektselt parsida nime, kus perekonnanimes on kaks või rohkem elementi, mida ei saa käsitleda aadlitiitli või mingi kindla enne teada oleva elemendina (näiteks St. perekonnanimes St. John). Et parser selliste nimedega hakkama ei saa, oli ka juba enne teada ning antud töös seda probleemi ei lahendata.

Samuti ei parsitud alguses korrektselt sidesõnadega ühendatud või tiitleid sisaldavaid mitmeosalisi perekonnanimesid. Veel oli probleeme Briti eesliitega *Lord Justice*, mille puhul parser leidis kõigepealt defineeritud eesliite *Lord*, eemaldas selle ning allesjäänud *Justice* ei tuvastatud enam eesliitena. Lahenduseks oli *Lord Justice*'i *Lord*'ist ettepoole tõstmine. Probleeme oli ka järelliidetega, kus osad olid eelnevalt defineeritud ja osad mitte – seal salvestati liited vales järjekorras.

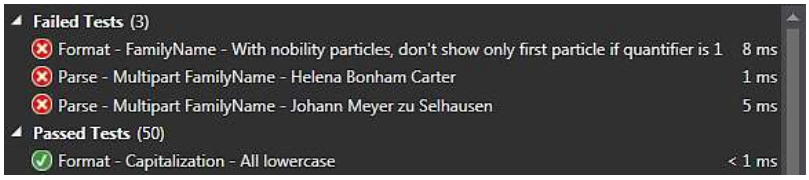


Joonis 7. Testimise algsed tulemused

Lisaks tõi testimine välja ka mõned vormindaja puudujäägid, näiteks näidati perekonnanimest *von der* Hagen esimese elemendi küsimisel ainult *von*'i, oleks pidanud näidatama tervet nimeosa, sest liited *von* ja *der* ei ole eraldiseisvad nimed, vaid kuuluvad Hagen'i juurde, moodustades sellega ühtse terviku. Lahenduseks oleks kvantori kasutamisel kontrollida, kas perekonnanime alguses on mõni muutujates *NobilityParticles* ja *DefinedWordsInFamilyName* olevatest sõnadest ning need näidatavate elementide arvu hulka arvata. Seda lahendust käesolevas töös enam ei realiseeritud.

Samuti tõid vormindaja testid välja ühe loogikavea eelistatud nime näitamises, kus eelistatud nime näidati ka siis, kui kasutati kvantorit ***** (mis peaks igal juhul kuvama kõik nime elemendid) ja ühe vea järelliidete vormindamises. Need vead parandati.

Joonis 8 illustreerib testimise viimaseid tulemusi, kus ainsad ebaõnnestuvad kaks parseri testi, mis puudutavad mitmeelemendilisi perekonnanimesid ja üks vormindaja test, mis kontrollib liidetega perekonnanimede korrektset vormindamist kvantorite kasutamisel.



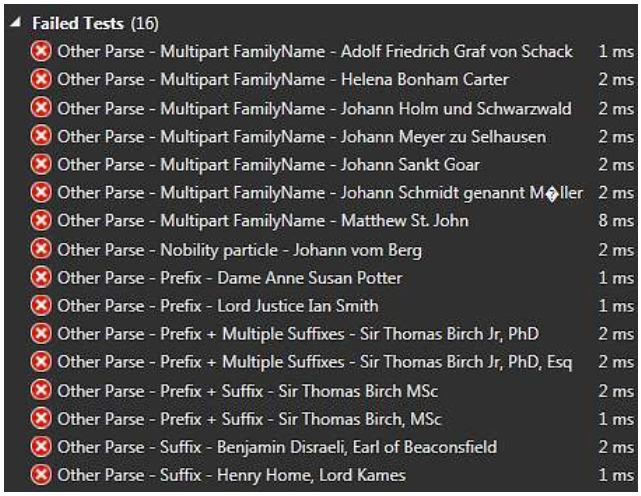
Joonis 8. Testimise viimased tulemused

Viimase sammuna võrreldi töös valminud parserit juba olemasoleva teegiga CSharp-Name-Parser [21]. Selleks lisati teegi projekt tööle ning tehti olemasoleva parseri testmeetodi kõrvale analoogne, mis kasutab sama sisendit, kuid lisab testide nimele ette sõna *Other*. Kuna

parseri testis võrreldakse leitud väljade väärtusi, mitte objekte endid, tuli testmeetodi tagastatavas massiivis ainult ära muuta mudeli atribuudid.

Esimese katsega õnnestus kõigist parseri 26 testist ainult 6, neist 5 testisid kõige lihtsamaid, ees- ja perekonnanimest koosnevaid nimesid.

Lähemal uurimisel selgus, et parser konverteerib kõigi nime elementide esitähed suur-tähtedeks. Peale testi võrdlustingimuste muutmist nii, et tähtede tõstu ei arvestataks, õnnestus 10 testi.



Joonis 9. Teise parseri testide tulemused

Teine parser ei suuda korrektselt parsida mitmest elemendist koosneva perekonnanimega isikunimesid. Samuti on probleeme ees- ja järelliiteid sisaldavate nimede parsimisega. Ebaõnnestus ka üks test, mis puudutas Saksa liitega perekonnanime, arvatavasti sellepärast, et teegis ei ole liide *vom* defineeritud.

Testide tulemustest võib järeldada, et antud töö tulemusena realiseeritud parser on efektiivsem, kui eelnevalt olemas olnud parser. Loodud vormindaja puhul pole eelnevat, millega võrrelda, kuid testide tulemused näitavad, et vähemalt töös kirjeldatud testjuhtudel on vormindaja tulemused korrektsed.

8. Visualiseerimine

Töö viimase osana valmis väike veebileht teegi töö demonstreerimiseks. Järgnevalt on lühidalt kirjeldatud veebilehte ja selle kasutatavat veebiteenust.

8.1 API

Veebiliidese ja teegi vahelise suhtluse võimaldamiseks realiseeriti ASP.NET Web API [24] raamistikku kasutades väike veebiteenus. ASP.NET Web API raamistik võimaldab kiirelt ja lihtsalt ehitada REST printsiipe järgiva teenuse. [25]

Veebiteenusel on üks URL, */api/Names/Format*, mis töötab kahte moodi. Üks võimalus on seda kasutada juba eraldatud nimeosade kindla formaadi järgi vormindamiseks, teine võimalus on anda kogu nimi sisse ühe parameetrina ja lasta süsteemil endal see osadeks parsida. Võimalikud parameetrid on esitatud tabelis 16. Teenus tagastab JSON kujul DTO, kus on nimi täiskujul, osadena, soovitud väljundformaat ning vormindatud nimi. DTO ehk *Data Transfer Object* on lihtne mudel, mis ei sisalda äriloogikat ning mille ainus eesmärk on andmete struktureeritult koos hoidmine nende eri protsesside vahel liikumisel.

Tabel 16. Veebiteenuse URL'i parameetrid

Parameeter	Kohustuslik	Parameeter	Kohustuslik
format	Jah	format	Jah
givenName	Ei	fullName	Jah
familyName	Ei	preferredName	Ei
prefix	Ei		
suffix	Ei		
preferredName	Ei		

Kuna teistest domeenidest AJAXiga andmete küsimine on üldiselt keelatud, ka samas domeenis aga erinevatel portidel asuvate ressursside vahel, oli tarvis veebiteenuses lubada CORS (Cross-Origin Resource Sharing). Selleks tuli lisada projektile teek Microsoft.AspNet.Cors ja selle sõltuvused ning teenuse kontrolleri klassis lisada klassile

endale ja tema meetoditele *route* attribuudid. Viimase sammuna tuli *WebApiConfig* klassi lisada järgmised read:

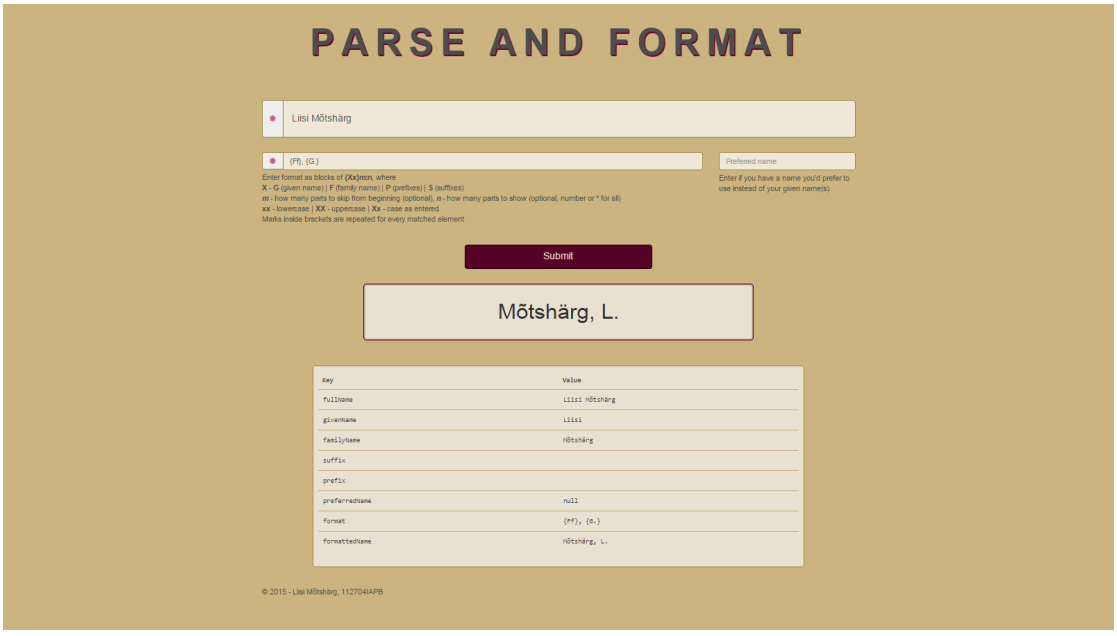
```
var cors = new EnableCorsAttribute(
              ConfigurationManager.AppSettings["corsAllowedDomains"], "*", "*");
config.EnableCors(cors);
```

Need lubavad CORS päringud muutujas *corsAllowedDomains* defineeritud domeenidelt.

8.2 Veebileht

Veebilehe *back-end*'i tegemiseks kasutati ASP.NET MVC [26] raamistikku, kuna Visual Studio genereerib projekti kondikava koos vajalike sõltuvuste ja teekidega ise kiirelt valmis. Lehe välimuse jaoks kasutati Bootstrapi, mis on üks levinumaid *frontend* raamistikke, ning võimaldab kiirelt valmis teha ilusa välimusega saidi. [27]

Lehel on vorm, kus on võimalik sisestada nimi ning formaat, kuidas nime näha soovitakse. Vajutades nupule „*Submit*", tehakse AJAX päring eelmises alampeatükis kirjeldatud veebiteenuse aadressile *api/Names/Format*. Kasutatakse varianti, kus kogu nimi saadetakse ühe parameetrina ning parser peab selle ise osadeks parsima ja saadud jupid vormindajale sisendiks andma. Vormiväljad antakse kaasa urli parameetritena. Päringu õnnestumisel näidatakse vormi all vastuseks saadud vormindatud nime. Samuti näidatakse vastuseks saadud JSON'it lehel tabelina, kus esimeses tulbas on JSON objekti võti ja teises väärtus.



Joonis 10. Lehe ekraanitõmmis

9. Kokkuvõte

Käesoleva töö eesmärgiks oli luua meetodid, mis võimaldaks Eestis, Saksamaal ja Suurbritannias enimlevinud kujuga isikunimesid etteantud formaadi järgi vormindada, samuti ühes tükis olevaid isikunimesid osadeks parsida.

Eesmärgi saavutamiseks leiti nimemudel, mida saab kasutada nii Eesti, Saksamaa kui ka Suurbritannia isikunimede haldamiseks. Mudel leiti kolme riigi nimede ning olemasolevate nimemudelite analüüsi teel. Selgus, et enamikku nimesid saab esitada kujul *{eesliide}n* {eesnimi}*n* {perenimi} *{järelliide}n*.

Töö esimese tulemusena valmis vormindusnotatsioon ning selle kirjeldus. Notatsioon võimaldab vormindada suvalist nime, mida saab kujutada tööks valitud nimemudelit kasutades.

Töö teiseks tulemuseks on teegina vormistatud C# funktsioonid, mis võimaldavad näidata isikunime soovitud kujul vastavalt etteantuad formaadistringile ning ühe stringina oleva nime osadeks parsida.

Testimine näitas, et nii vormindaja kui parser teevad seda, mida nad olid algselt tegema plaanitud. Vormindajat saab kasutada peaaegu iga nime jaoks, mis on eelnevalt osadeks jagatud (ei saa kasutada eesliidetega perenimest esimese nime näitamiseks, selle tulemuseks on esimene eesliide). Parserit saab kasutada nimede jaoks, mille perekonnanimi on kas üheosaline või sisaldab lisaks ühele perekonnanimele kindlalt defineeritud liiteid.

Nimetatud puudused on seotud nimekujudega, mida väga sageli ei esine. Seega võib öelda, et töö eesmärk sai täidetud.

Edasiarendusvõimalusi on mitmeid, tähtsamad neist on järgmised:

- Kasutada nimede parsimisel nime kuju tähistamiseks sama notatsiooni kui vormindamisel. See võimaldaks täpselt määrata, missugused on erinevad nime osad ning parsimise täpsus paraneks tunduvalt. Notatsiooni kasutamisel laheneks ka probleem mitmeosaliste perekonnanimedega.

- Nime parsimisel kuvada tulemus kasutajale ning anda talle võimalus tulemust korrigeerida. See aitaks eriti juhtudel, kui isiku nimi on ebatavalise kujuga ning parseri tulemus on vale.

- Laiendada toetatud ees- ja järelliidete hulka.

- Tuntud ees- ja järelliidete tõlkimine nimede lokaliseerimiseks. Käesolevas töös loodud teeki edasi arendades saaks selleks kasutada ressursifaile. Nii saaks näiteks saaks Eesti arsti „doktor Jaan Tamm“ nime inglise keeles kirjutada „*doctor* Jaan Tamm“.

Summary

The purpose of this thesis was to create a method for parsing most names found in Estonia, Germany and the UK and formatting them according to given format string.

To achieve this purpose, the first action was to choose a name model that could be used to manage most personal names of these three countries. The model was chosen by analysing different name patterns of those countries, as well as existing name models. As a result, it was found that most names could be described with the pattern *{prefix}n* {given name}*n* {family name} *{suffix}n*.

The first result was created formatting notation and it's description. The notation allows to format any name that can be described using chosen name model.

The second result was a C# library, that exposes methods for formatting previously separated names according to given format string and parsing a single namestring into multiple parts.

Testing proved that both formatter and parser are capable of doing what they were originally intended to do. Formatter can be used for formatting almost any name that has been separated into parts beforehand. Parser can be used for names where family name consists of one name or includes pre-defined prefixes. Therefore, the purpose of this thesis was fulfilled.

Kasutatud kirjandus

1. Vallaste.ee e-Teatmik: IT ja sidetehnika seletav sõnaraamat. DTD (Document Type Definition) [WWW] http://vallaste.ee/sona.asp?Type=UserId&otsing=129 (21.05.2015)

2. FHIR [WWW] http://www.hl7.org/FHIR/datatypes.html#humanname (03.05.2015)

3. Ishida R. Personal names around the world. [WWW] http://www.w3.org/International/questions/qa-personal-names (02.05.2015)

4. Eesti Keele Käsiraamat [WWW] http://www.eki.ee/books/ekk09/index.php?p=6&p1=5&id=548 (03.05.2015)

5. Oxford Dictionaries. Hyphen [WWW] http://www.oxforddictionaries.com/words/hyphen (18.05.2015)

6. Nimeseadus. – *Riigi Teataja* I, 06.03.2015, 32 [WWW] https://www.riigiteataja.ee/akt/106032015032 (27.04.2015)

7. Names of Persons: National Usages for Entry in Catalogues / International Federation of Library Associations and Institutions. 4th ed. Mörlenbach : K. G. Saur Verlag GmbH & Co., 1996

8. Die Verfassung des Deutschen Reichs, Artikkel 109, III [WWW] http://www.gesetze-im-internet.de/wrv/index.html#BJNR013830919BJNE000200314 (08.05.2015)

9. Adelsforschung, Institut Deutsche. Adelszeichen und Adel: Kennzeichnet das 'von' in jedem Fall eine Adelsfamilie? [WWW] http://home.foni.net/~adelsforschung/faq1.htm (19.05.2015)

10. Behind the Name. English Names [WWW] http://www.behindthename.com/glossary/view/english_names (25.04.2015)

11. Gliffy [WWW] https://www.gliffy.com (07.05.2015)

12. Person.Name | Android Developers [WWW] https://developer.android.com/reference/com/google/android/gms/plus/model/people/Person.Name.html (08.05.2015)

13. DocBook: The Definitive Guide [WWW] http://www.docbook.org/tdg/en/html/personname.html (08.05.2015)

14. Arlow, J., Neustadt, I. Enterprise patterns and MDA: Building Better Software with Archetype Patterns and UML. 1st ed. Boston : Addison-Wesley, 2004

15. Silverston, L. The Data Model Resource Book. Revised Edition, Volume 1. New York : John Wiley & Sons, Inc., 2001

16. enum (C# Reference) [WWW] https://msdn.microsoft.com/en-us/library/sbbt4032.aspx (19.05.2015)

17. Aadressiandmete käsiraamat [WWW] http://ads.maaamet.ee/?id=1125 (26.05.2015)

18. Vallaste.ee e-Teatmik: IT ja sidetehnika seletav sõnaraamat. Parser http://vallaste.ee/index.htm?Type=UserId&otsing=253 (19.05.2015)

19. GitHub. Humanparser [WWW] https://github.com/chovy/humanparser (06.05.2015)

20. Resources in .Resx File Format [WWW] https://msdn.microsoft.com/en-us/library/ekyft91f%28v=VS.90%29.aspx (18.05.2015)

21. GitHub. CSharp-Name-Parser [WWW] https://github.com/ianlee74/CSharp-Name-Parser (19.05.2015)

22. IEEE Standard for Software and System Test Documentation : IEEE Std 829-2008. New York, Institute of Electrical and Electronics Engineers, Inc.

23. NUnit [WWW] http://www.nunit.org (16.05.2015)

24. ASP.NET Web API [WWW] http://www.asp.net/web-api (05.05.2015)

25. MSDN. ASP.NET Web API [WWW] https://msdn.microsoft.com/en-us/library/hh833994(v=vs.108).aspx (20.05.2015)

26. ASP.NET MVC [WWW] http://www.asp.net/mvc (05.05.2015)

27. Bootstrap [WWW] http://getbootstrap.com (05.05.2015)

Lisa 1. Vormindaja lähtekood

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

namespace PersonNames
{
  public static class Formatter
  {
    public static String Format(PersonName name, String formatString)
    {
      // cut format string to parts
      var formatParts = parseFormatString(formatString);

      // loop through parts and format them separately
      // finally replace format part/placeholder in format string
      foreach (var formatPart in formatParts)
      {
        var namePart = formatNamePart(name, formatPart);
        formatString = formatString.Replace(formatPart, namePart);
      }
      return formatString;
    }

    private static IEnumerable<String> parseFormatString(String formatString)
    {
      Regex regex = new Regex(@"(\{.*?\}(?:\*|\d*)?(?:(?<!\})\:(?:\*|\d*))?)",
RegexOptions.IgnoreCase);

      return regex.Matches(formatString)
                  .Cast<Match>()
                  .Select(x => x.Value.Trim())
                  .Where(x => !String.IsNullOrEmpty(x));
    }

    private static String formatNamePart(PersonName name, string formatPart)
    {
      var regex = new
Regex(@"\{(?<Before>[^PGFS]*)(?<Identifier>[PGFS]*)(?<After>[^\}]*)\}(?:(?<Skip>[0-
9]*?(?=\:))\:)?(?<Take>[0-9\*]*)", RegexOptions.IgnoreCase);

      var match = regex.Match(formatPart);

      var identifier = match.Groups["Identifier"].Value;
      var beforeIdentifier = match.Groups["Before"].Value;
      var afterIdentifier = match.Groups["After"].Value;
      var take = match.Groups["Take"].Value;
      var skip = match.Groups["Skip"].Value;

      var caseType = CaseType.Normal;

      // if identifier was missing, we can't really format anything, return null
      if (String.IsNullOrEmpty(identifier))
      {
        return null;
      }
```

```
      // get part of name to format
      var namePart = getNamePartByFormatString(name, identifier,
!String.IsNullOrEmpty(take));

      // split by comma, if namepart contains commas, else by space
      IEnumerable<String> names = null;
      var namePartSeparator = namePart.IndexOf(',') > 0 ?
          new char[] { ',' } :
          new char[] { ' ' };
      names = namePart.Split(namePartSeparator,
StringSplitOptions.RemoveEmptyEntries);


      if (!String.IsNullOrWhiteSpace(skip) &&
          skip.ToCharArray().All(c => char.IsDigit(c)))
      {
        // number of names to show
        var amountToSkip = int.Parse(skip);
        names = names.Skip(amountToSkip);
      }

      if (!String.IsNullOrWhiteSpace(take) &&
          take.ToCharArray().All(c => char.IsDigit(c)))
      {
        // number of names to show
        var count = int.Parse(take);
        names = names.Take(count);
      }

      // show only initials if formatstring letter part length is 1
      if (identifier.Length == 1)
      {
        names = names.Select(
            n => n.IndexOf('-') <= 0 ?
                n.Substring(0, 1) :
                String.Join("-",
                n.Split('-').Select(n2 => n2.Substring(0, 1)))
            );
      }

      caseType = getCaseType(identifier);
      switch (caseType)
      {
        case CaseType.Lower:
          names = names.Select(n => n.ToLowerInvariant());
          break;
        case CaseType.Upper:
          names = names.Select(n => n.ToUpperInvariant());
          break;
        case CaseType.Normal:
        default:
          break;
      }

      names = names.Select(n => String.Format("{0}{1}{2}", beforeIdentifier, n,
afterIdentifier));

      return String.Join(" ", names).Trim(' ', ',');
    }
```

```
    private static String getNamePartByFormatString(PersonName name, String
identifier, bool takeOnlyFullName)
    {
      var letter = identifier.Substring(0, 1).ToUpper();
      var namePart = "";

      switch (letter)
      {
        case "G":
          // if preferred name exists and we don't want all name elements
          if (!String.IsNullOrWhiteSpace(name.PreferredName) && !takeOnlyFullName)
          {
            namePart = name.PreferredName;
          }
          else
          {
            namePart = name.GivenName;
          }
          break;
        case "F":
          namePart = name.FamilyName;
          break;
        case "P":
          namePart = name.Prefix;
          break;
        case "S":
          namePart = name.Suffix;
          break;
      }

      return namePart;
    }

    private enum CaseType { Upper, Lower, Normal }

    private static CaseType getCaseType(String identifier)
    {
      var chars = identifier.ToCharArray();
      var caseType = CaseType.Normal;

      if (Char.IsLower(chars[0]))
      {
        caseType = CaseType.Lower;
      }
      else if (Char.IsUpper(chars[0]))
      {
        if (chars.Length == 1 || Char.IsUpper(chars[1]))
        {
          caseType = CaseType.Upper;
        }
      }

      return caseType;
    }
  }
}
```

Lisa 2. Parseri lähtekood

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace PersonNames
{
  public static class Parser
  {
    // Chars to always remove from beginning and end of found names
    private static char[] _charsToTrim = { ' ', ',' };

    public static PersonName Parse(String nameString)
    {
      // remove excess whitespace
      nameString = nameString.Trim();

      var name = new PersonName { FullName = nameString };

      List<String> foundPrefixes = null;
      tryRemovePrefixes(nameString, out foundPrefixes, out nameString);
      name.Prefix = String.Join(",", foundPrefixes);

      // remove suffixes
      List<String> foundSuffixes = null;
      tryRemoveSuffixes(nameString, out foundSuffixes, out nameString);
      name.Suffix = String.Join(",", foundSuffixes);

      // remove familyname
      String foundFamilyName = "";
      tryRemoveFamilyName(nameString, out foundFamilyName, out nameString);
      name.FamilyName = foundFamilyName;

      // given name remains
      name.GivenName = nameString;

      return name;
    }

    private static bool tryRemovePrefixes(String nameString, out List<String>
foundPrefixes, out String nameStringWithoutPrefixes)
    {
      foundPrefixes = new List<String>();
      var prefixes = PersonNames.Data.Prefixes.Split(',');

      // remove all prefixes that match defined ones and put them in list
      while (prefixes.Any(p => nameString.StartsWith(p)))
      {
        foreach (var prefix in prefixes)
        {
          if (nameString.IndexOf(prefix) == 0)
          {
            // if there is . right after prefix, take this too
            var takeLength = (nameString.IndexOf('.') == prefix.Length ?
                prefix.Length + 1 :
                prefix.Length);
```

```
            foundPrefixes.Add(nameString.Substring(0, takeLength)
                .Trim(_charsToTrim));

            nameString = nameString
                .Substring(takeLength)
                .Trim(_charsToTrim);
          }
        }
      }

      var nameParts = nameString.Split(' ');
      var names = new Stack<String>(nameParts.Reverse());

      // remove all prefixes that end with .
      while (names.Peek().IndexOf('.') > 0)
      {
        foundPrefixes.Add(names.Pop());
      }

      nameStringWithoutPrefixes = String.Join(" ", names);
      return true;
    }

    private static bool tryRemoveSuffixes(String nameString, out List<String>
foundSuffixes, out String nameStringWithoutSuffixes)
    {
      foundSuffixes = new List<String>();

      var suffixStartIndex = nameString.IndexOf(',');
      if (suffixStartIndex >= 0)
      {
        foundSuffixes = nameString.Substring(suffixStartIndex)
                                  .Split(',')
                                  .Where(s => !String.IsNullOrWhiteSpace(s))
                                  .Select(s => s.Trim())
                                  .ToList();

        nameString = nameString
            .Substring(0, suffixStartIndex)
            .Trim(_charsToTrim);
      }

      var definedSuffixes = PersonNames.Data.Suffixes.Split(',');
      while (definedSuffixes.Any(s => nameString.EndsWith(s)))
      {
        foreach (var suffix in definedSuffixes)
        {
          if (nameString.EndsWith(suffix))
          {
            var suffixIndex = nameString.LastIndexOf(suffix);
            foundSuffixes.Insert(0, nameString
                .Substring(suffixIndex)
                .Trim(_charsToTrim));

            nameString = nameString.Substring(0, suffixIndex)
                .Trim(_charsToTrim);
          }
        }
      }

      nameStringWithoutSuffixes = nameString;
```

```
      return true;
    }

    private static bool tryRemoveFamilyName(String nameString, out String familyName,
out String nameStringWithoutFamilyName)
    {
      var nameParts = nameString.Split(' ');
      var names = new Stack<String>(nameParts);

      var familyNameQueue = new Stack<String>();

      // take last as familyname
      familyNameQueue.Push(names.Pop());

      // if next is join word, add this and element before it to family name
      var familyNameJoins = PersonNames.Data.FamilyNameJoins.Split(',');
      if (names.Count > 0 &&
          familyNameJoins.Any(j => j.Equals(names.Peek(),
              StringComparison.InvariantCultureIgnoreCase)))
      {
        familyNameQueue.Push(names.Pop());
        familyNameQueue.Push(names.Pop());
      }

      // check for nobility particles before familyname, if any exists, take them too
      var nobilityPrefixes = PersonNames.Data.NobilityParticles.Split(',');
      while (names.Count > 0 &&
          nobilityPrefixes.Any(n => n.Equals(names.Peek(),
              StringComparison.InvariantCultureIgnoreCase)))
      {
        familyNameQueue.Push(names.Pop());
      }

      // also check for nobility title in name
      var wordsInFamilyName = PersonNames.Data.DefinedWordsInFamilyName.Split(',');
      if (names.Count > 0 &&
          wordsInFamilyName.Any(t => t.Equals(names.Peek(),
              StringComparison.InvariantCultureIgnoreCase)))
      {
        familyNameQueue.Push(names.Pop());
      }

      familyName = String.Join(" ", familyNameQueue);
      // everything that remains are given names (reverse, because they are in
reversed order in stack)
      nameStringWithoutFamilyName = String.Join(" ", names.Reverse());

      return true;
    }
  }
}
```

Lisa 3. Testide lähtekood

```
using System;
using NUnit.Framework;

namespace PersonNames.Tests
{
  [TestFixture]
  public class FormatterTests
  {
    [Test, TestCaseSource(typeof(TestDataFactory), "NamesForFormatter")]
    public String TestFormat(PersonName name, String format)
    {
      return Formatter.Format(name, format);
    }
  }
}
```

```
using System;
using NUnit.Framework;

namespace PersonNames.Tests
{
  [TestFixture]
  public class ParserTests
  {
    private CSharpNameParser.NameParser otherParser = new
CSharpNameParser.NameParser();

    [Test, TestCaseSource(typeof(TestDataFactory), "NamesForParser")]
    public String[] TestParse(String nameString)
    {
      var name = Parser.Parse(nameString);
      return new String[] { name.GivenName, name.FamilyName, name.Prefix, name.Suffix
};
    }

    [Test, TestCaseSource(typeof(TestDataFactory), "NamesForOtherCSharpParser")]
    public String[] TestParseOtherCharpParser(String nameString)
    {
      var name = otherParser.Parse(nameString);
      return new String[] {
                name.FirstName.ToUpperInvariant(),
                name.LastName.ToUpperInvariant(),
                name.Salutation.ToUpperInvariant(),
                name.Suffix.ToUpperInvariant()
              };
    }
  }
}
```

```
using System;
using System.Collections;
using NUnit.Framework;

namespace PersonNames.Tests
{
  public class TestDataFactory
  {
    //format;expected;given;family;prefix;suffix;preferred;category;testname
    public static IEnumerable NamesForFormatter
    {
      get
      {
        var file = System.IO.File.ReadAllLines("format_test_data.txt");
        foreach (var line in file)
        {
          var parts = line.Split(';');
          if (String.IsNullOrWhiteSpace(line) || parts.Length != 9 ||
line.StartsWith("#"))
          {
            continue;
          }
          var category = (parts[7].Length > 0 ? parts[7] : "GeneralFormat");

          yield return new TestCaseData(new PersonName
          {
            GivenName = parts[2],
            FamilyName = parts[3],
            Prefix = parts[4],
            Suffix = parts[5],
            PreferredName = parts[6]
          }, parts[0])
              .Returns(parts[1])
              .SetCategory(category)
              .SetDescription(parts[8])
              .SetName(String.Format("Format - {0} - {1}", category, parts[8]));
        }
      }
    }

    // fullname;given;family;prefix;suffix;category
    public static IEnumerable NamesForParser
    {
      get
      {
        var file = System.IO.File.ReadAllLines("parser_test_data.txt");
        foreach (var line in file)
        {
          var parts = line.Split(';');
          if (String.IsNullOrWhiteSpace(line) || parts.Length != 6 ||
line.StartsWith("#"))
          {
            continue;
          }
          var category = (parts[5].Length > 0 ? parts[5] : "GeneralParser");

          yield return new TestCaseData(parts[0])
              .Returns(new String[] { parts[1], parts[2], parts[3], parts[4] })
              .SetCategory(category)
              .SetName(String.Format("Parse - {0} - {1}", category, parts[0]));
        }
      }
```

```
    }

    // fullname;given;family;prefix;suffix;category
    public static IEnumerable NamesForOtherCSharpParser
    {
      get
      {
        var file = System.IO.File.ReadAllLines("parser_test_data.txt");
        foreach (var line in file)
        {
          var parts = line.Split(';');
          if (String.IsNullOrWhiteSpace(line) || parts.Length != 6 ||
line.StartsWith("#"))
          {
            continue;
          }
          var category = (parts[5].Length > 0 ? parts[5] : "GeneralParser");

          yield return new TestCaseData(parts[0])
              .Returns(new String[] {
                           parts[1].ToUpperInvariant(),
                           parts[2].ToUpperInvariant(),
                           parts[3].ToUpperInvariant(),
                           parts[4].ToUpperInvariant()
                       })
              .SetCategory(category)
              .SetName(String.Format("Other Parse - {0} - {1}", category, parts[0]));
        }
      }
    }
  }
}
```

Lisa 4. Parseri testjuhud

Muster	Nimi	Testitav osa	Oodatav tulemus
{eesnimi}*n* {perenimi} või {eesnimi} {isanimi} {perenimi}	Anna Mets	Eesnimi	Anna
		Perenimi	Mets
	Jaan Markus Murakas	Eesnimi (2 tükki)	Jaan Markus
		Perenimi	Murakas
{eesnimi-eesnimi} {perenimi}	Jana-Liisa Mutt	Eesnimi	Jana-Liisa
		Perenimi	Mutt
{eesnimi} {perenimi-perenimi}	Ave Murakas-Mutt	Eesnimi	Ave
		Perenimi	Murakas-Mutt
{eesnimi-eesnimi} {perenimi-perenimi}	Jana-Liisa Mutt-Harakas	Eesnimi	Jana-Liisa
		Perenimi	Mutt-Harakas
{eesnimi}*n* {*eesliide/-liited* + perenimi}	Karl Ernst *von* Baer	Eesnimi	Karl Ernst
		Perenimi	*von* Baer
	Johann *von der* Hagen	Eesnimi	Johann
		Perenimi (koos liidetega)	*von der* Hagen
	Johann *vom* Berg		*vom* Berg
	Johann *von zur* Mühlen		*von zur* Mühlen

Muster	Nimi	Testitav osa	Oodatav tulemus
	Johann *von und zu* Urf		*von und zu* Urf
{eesnimi}*n* {perenimi+kohanimi}	Johann Meyer *zu* Selhausen	Perenimi	Meyer zu Selhausen
{eesnimi}*n* {perenimi und perenimi}	Johann Holm und Schwarzwald	Perenimi	Holm und Schwarzwald
{eesnimi}*n* {perenimi eesliitega Sankt|St.}	Johann Sankt Goar	Perenimi	Sankt Goar
	Matthew St. John	Perenimi	St. John
{eesnimi}*n* {perenimi *genannt* perenimi}	Johann Schmidt *genannt* Müller	Perenimi	Schmidt *genannt* Müller
{eesnimi}*n* {tiitel perenime osana + perenimi}	Adolf Friedrich Graf *von* Schack	Eesnimi	Adolf Friedrich
		Perenimi	Graf *von* Schack
{eesnimi}*n* {perenimi mitme otseselt nimesse kuuluva eesliitega}	Susan De La Mare	Perenimi	De La Mare
{eesnimi}*n* {perenimi} {perenimi}	Helena Bonham Carter	Perenimi	Bonham Carter
{eesnimi}*n* {perenimi}, *{tiitel}*	Benjamin Disraeli, *Earl of Beaconsfield*	Eesnimi	Benjamin
		Perenimi	Disraeli
		Järelliide (defineerimata, komaga eraldatud)	Earl of Beaconsfield

Muster	Nimi	Testitav osa	Oodatav tulemus
	Henry Home, *Lord Kames*	Eesnimi	Henry
		Perenimi	Home
		Järelliide (tiitel koos kohanimega)	Lord Kames
{tiitel} {eesnimi}*n* {perenimi}	*Dame* Anne Susan Potter	Eesliide	Dame
		Eesnimi	Anne Susan
		Perenimi	Potter
	Lord Justice Ian Smith	Eesliide	Lord Justice
		Eesnimi	Ian
		Perenimi	Smith
{tiitel} {eesnimi}*n* {perenimi}, *{tiitel}*	*Sir* Thomas Birch, *MSc*	Eesliide	Sir
		Eesnimi	Thomas
		Perenimi	Birch
		Järelliide	MSc
{tiitel} {eesnimi}*n* {perenimi} *{põlvkond}, {tiitel}*	*Sir* Thomas Birch *Jr*, *PhD*	Eesliide	Sir
		Eesnimi	Thomas
		Perenimi	Birch
		Järelliide (mitu tükki)	Jr, PhD

Helehalliga on tähistatud nimekujud, mida parser töödelda ei suuda.