TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Sander-Sebastian Värv

PDF dokumendi konverteerimine EPUB formaati

Bakalaureusetöö (9 EAP)

Juhendaja: Vesal Vojdani, PhD

Tartu 2017

PDF dokumendi konverteerimine EPUB formaati

Lühikokkuvõte:

Käesolevas töös kirjeldatakse rakendust, mille abil on võimalik kuluefektiivsel viisil konverteerida PDF dokumente EPUB 2.0.1 formaati. Töös on kirjeldatud tehnilisi lahendusi, mille abil on võimalik säilitada dokumendi algne struktuur ja luua sellest müügikõlblik e-raamat. Lisaks kirjeldatakse rakenduse loomisega seotud tehnoloogiaid ning võrreldakse loodud rakendust olemasolevate lahendustega.

Võtmesõnad:

PDF, EPUB, konverteerimine, lõikude tuvastamine

CERCS: P175

Cost effective PDF to EPUB conversion

Abstract:

This thesis describes an application that can convert PDF to EPUB in a cost-effective way. The technical solutions to retain the original structure of the input PDF and create a valid EPUB 2.0.1 file are described. In addition, the thesis gives an overview of the technologies used to create the application and includes the comparison of the application created under this thesis with some preexisting solutions.

Keywords:

PDF, EPUB, conversion, reflowing PDF, paragraph recognition

CERCS: P175

Sisukord

Sissejuhatus

Tehnoloogia areng ja kaasaskantavate nutiseadmete populaarsuse kasv on suurendanud tavapäraste raamatute kõrval ka e-raamatute tarbimist. Lisaks spetsiaalsetele e-lugeritele on võimalik lugeda e-raamatuid ka nutiseadmetest ning arvutitest. Digitaalsete raamatute levinumateks formaatideks on EPUB ja mobi. E-raamatute jaoks loodud formaadid on oluliselt kasutajasõbralikumad kirjastajate ja lugejate seas laialt kasutusel olevast PDF formaadist. PDF formaat on sobilik küll raamatu vorminduse säilitamiseks, kuid selle edasi töötlemine on keeruline.

Käesoleva töö põhiliseks eesmärgiks on luua programm, millega on võimalik kuluefektiivsel viisil konverteerida PDF faile EPUB kujule. Töö on valminud Eesti Digiraamatute Keskus OÜ (EDRK) ja Tartu Ülikooli koostööprojektina. Varem on EDRK-s PDF faile konverteeritud EPUB kujule manuaalselt, kasutades seejuures abivahendeid nagu Calibre ja „pdf2epub", millega on võimalik eraldada tekstist lehealuseid viiteid. Müügikõlbliku EPUB faili koostamine on siiani nõudnud erialaseid teadmisi, näiteks HTML, CSS, JavaScript, XML tehnoloogiatest ja EPUB struktuurist. Parema kuluefektiivsuse saavutamiseks on loodud programm, mille kasutamiseks vajaminevate eelteadmiste hulk on väiksem, mistõttu väheneb seda ülesannet täitva inimese väljaõppele vajaminev aeg. Programm on loodud nelja liikmelise meeskonna poolt, mille juht on autor. Autori otsene panus on välja toodud peatükis 6.4.

Tulemusena loodud rakendus ei eelda kasutajalt varasemaid teadmisi EPUB faili struktuurist ega ka selle loomisel kasutatavatest tehnoloogiatest. Loodud on graafiline kasutajaliides, mille abil on võimalik märgistada sisend PDF failis erinevad elemendid nagu näiteks peatükid, lõigud, pildid, pealkirjad ja lehealused. Selle abil luuakse märgistusele vastav EPUB fail. Kasutaja ülesandeks on vaid erinevad elemendid visuaalselt ära tunda ja korrektselt märgistada.

Töö esimeses peatükis antakse ülevaade tööga seonduvatest failiformaatidest PDF ja EPUB. Järgmisena tutvustatakse hetkel olemasolevaid tehnoloogiaid PDF-ist HTML-i konverteerimiseks ning põhjendatakse valitud suunda rakenduse arendamisel. Kolmandas peatükis tutvustatakse rakendust kasutajast lähtuvalt. Lisaks kirjeldatakse loodud töövoogu. Neljandas peatükis keskendutakse teksti eraldamisele ja lõikude tuvastamisele. Viies peatükk kirjeldab kujunduselementide eraldamist ja säilitamist raamatus. Peatükis tutvustatakse võimalusi piltide eraldamiseks PDF dokumendist ning kirjaviiside säilitamist antud rakenduse kontekstis.

Lisaks kirjeldatakse EPUB raamatu koostamist Epublib teegi abil. Kuues peatükk keskendub detailsemalt implementatsiooni protsessile ning kirjeldab tehnoloogilisi otsuseid ja kasutatud metoodikat, lisaks tutvustatakse töövoogu rakendusest lähtuvalt. Viimasena võrreldakse töö tulemusena loodud rakendust konkureerivate rakendustega, lähtudes seejuures eelkõige lõppkasutaja vajadustest.

1. PDF ja EPUB formaadid

Antud peatükis kirjeldatakse rakenduse töötamise seisukohast olulisi failiformaate. Esimesena kirjeldatakse lähemalt PDF formaati ning seda, kuidas erinevad elemendid selles formaadis kirjeldatud on. Järgnevalt kirjeldatakse EPUB formaati, lähtudes peamiselt EPUB 2.0.1 spetsifikatsioonist. Peatükis põhjendatakse, milleks antud formaat kasulik on ning tuuakse välja antud formaadis kasutusel olevad tehnoloogiad. Lisaks kirjeldatakse kasutusel olevat failistruktuuri.

1.1 PDF formaat

Portable Document Format ehk PDF dokument on laialt kasutusel olev failiformaat, mida kasutatakse tekstifaili algse vorminduse säilitamiseks. Algselt arendas PDF formaadi välja Adobe, kuid alates 2008. aastast on selle spetsifikatsiooni eest vastutav Rahvusvaheline Standardiseerimise Organisatsioon (*International Organisation for Standardization*) standardiseerimisnumbriga ISO 32000-1:2008.

Selleks, et säilitada vormindust sõltumatult vaatamiseks kasutatavast seadmest, teisendatakse kõik sümbolid vektorkujule ning igale sümbolile antakse kaasa absoluutne asukoht lehel x ja y koordinaatidega. Vektorkujul olev tekst kujutab endast käskude jada, mis kirjeldab toiminguid, kuidas on otstarbekas kujutada antud sümbolit. PDF formaadi spetsifikatsioonis [1] ei ole rangelt määratud, kuidas tekst peaks olema kodeeritud. Seetõttu oleneb tekstis olevate tähtede kodeerimise järjekord puhtalt PDF-i kirjutava programmi implementatsioonist. Olenevalt koostatud dokumendist ei pruugi tähtede kodeerimise järjekord sõnades olla järjestikune. Seetõttu ei ole võimalik eraldada struktureeritud teksti PDF dokumendist, kasutades selleks vektoriseeritud teksti kujutamiseks vajaminevate käskude kronoloogilist järjekorda [2, 3].

Mõnel juhul koosneb dokument rasterkujutisest, ehk pikslite maatriksina kujutatavast pildist, mis esitab teksti, kuid ei sisalda endas informatsiooni eraldi tähemärkide kohta. Sellisel juhul on olemas informatsioon ainult piksliväärtuste kohta, millest teksti eraldamine on võimalik vaid optilise märgituvastuse teel.

Sarnane olukord on ka tabelitega. Tabel koosneb vektorkujul olevatest joontest ridade ja veergude eraldamiseks ning nende joonte vahel olevatest sümbolitest. Sisemiselt ei ole PDF-is tabelid eraldi märgendatud ning neid on seetõttu keeruline eristada tavatekstist, mille juurde võivad samuti kuuluda vertikaalsed ja horisontaalsed jooned.

Fondid lisatakse failile enamasti integreerituna, et tagada faili üksühene vormindus. See aga muudab failide mahtu suuremaks, mis võib mõnel juhul muutuda tülikaks. Seetõttu võetakse fontidest enamasti vaid alamhulk sümbolitest, mis on kasutuses antud PDF-is. Võimalik on ka luua PDF faili ilma fonti manustamata, sellisel juhul asendatakse font vaikefondiga, kui faili kuvavas süsteemis antud font puudub [4, 5].

PDF failides ei talletata pilte enamlevinud formaatides nagu PNG, JPG või TIFF vaid kirjeldatakse binaarkujul iga pikslit eraldi kindlas värvusruumis. Selline esitusviis võimaldab näiteks CMYK värvusruumis pilti esitada binaarandmete plokkidena, kus iga piksli esitamiseks on vajalik nelja baidi pikkune plokk [6]. Igale pildile antakse eraldi nimi, näiteks "Im1".

PDF-ist teksti ja teiste elementide eraldamisega seonduvate probleemide lahendamiseks tutvustas Adobe võimalust markeerida PDF-is sisalduvad elemendid ning kategoriseerida need vastavalt sisule. Dokumentide markeerimine on aga valikuline, mitte kohustuslik ning paljud tööriistad isegi ei paku võimalust seda kasutada.

1.2 EPUB formaat

EPUB (*electronic publication)* on e-raamatu standardite kogum. EPUB faili tunneb ära laiendi ".epub" järgi. Selles formaadis faile on võimalik kuvada paljudes erinevates seadmetes, sealhulgas tahvelarvutites, e-lugerites ning nutitelefonides. EPUB formaadi spetsifitseerimise ja standardiseerimise eest vastutab Rahvusvaheline Digitaalse Kirjastuse Foorum (*International Digital Publishing Forum*). Ametlikuks standardiks sai EPUB aastal 2007 versiooniga 2.0. Kolm aastat hiljem, aastal 2010, võeti kasutusele uuenduskuuri läbinud versioon 2.0.1. Praeguseks kõige hilisem versioon on EPUB 3.1, mis väljastati 5. jaanuaril 2017 [7, 8].

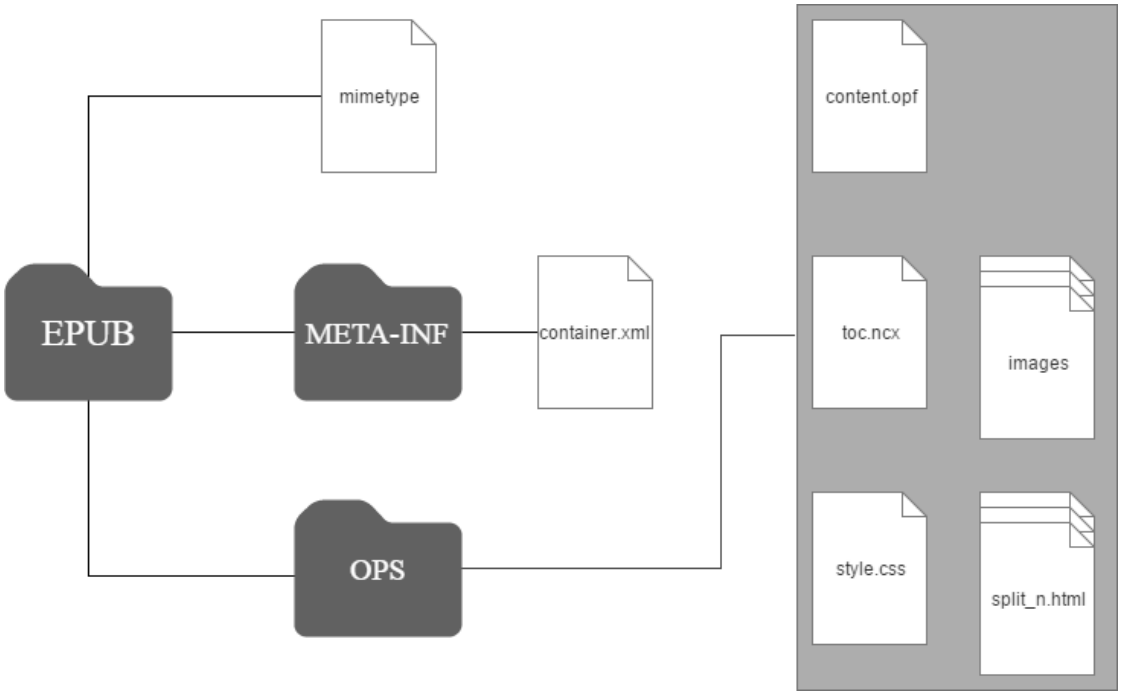
EPUB formaadi üheks põhiliseks eeliseks on teksti kuvamine kasutajale modifitseeritaval viisil, ehk formaat ei määra kindlaks teksti küljendust. Kasutajal on võimalik valida teksti fonti, suurust ja paigutust. Seadistusvõimalused parandavad kasutajakogemust, võimaldades kohandada küljendust vastavalt lugemisseadme ekraani suurusele, eraldusvõimele ja kasutaja soovile [9, 10].

Alates EPUB 3.0 formaadist [11] on võimalik luua ka fikseeritud küljendusreeglitega e-raamatuid [12]. Seda võimalust kasutatakse eelkõige raamatute puhul, kus küljendust ja sisu

on keeruline eraldada või juhul, kui osa informatsiooni jääb edastamata küljenduse muutmisel. Sellisteks raamatuteks on näiteks kokaraamatud ja pildiraamatud.

EPUB formaadis raamat sarnaneb oma ülesehituselt suuresti veebilehele. Kogu raamatu sisu on kirjutatud XHTML failis ning kujundusreeglid on kirjeldatud selle juurde kuuluvas CSS failis. Kõik EPUB faili kuuluvad alamfailid ja -kataloogid on kokku pakitud ühte konteinerisse, mille laiend on ".epub".

Selles formaadis raamatud võivad sisaldada vektorgraafikat SVG formaadis ja pilte, mille failiformaat pole üheselt kindlaks määratud. EPUB faili kuuluvad lisaks veel OPS (*Open Publication Structure*), OPF (*Open Packaging Format*), OCF (*Open Container Format*) ja NCX failid. Näidis EPUB 2.0.1 struktuurist on kujutatud joonisel 1.



Joonis 1. Autori loodud joonis EPUB struktuurist

OPF fail, mis üldiselt kannab nime content.opf, sisaldab endas digitaalse raamatu metaandmeid, mis baseerub *Open Packaging Format* spetsifikatsioonil. Selles failis on registreeritud kõik OPS kausta kuuluvad failid ning lisaks kirjeldatud raamatu metaandmed, mille alla kuulub näiteks informatsioon autori kohta, raamatu kirjeldus ja temaatika.

NCX fail, mis sisaldab raamatu sisukorda, teeb kasutajal raamatus navigeerimise mugavamaks. Iga XHTML formaadis peatükk on selles failis registreeritud koos peatüki pealkirjaga. Üldiselt on kokku lepitud, et NCX fail kannab nime toc.ncx.

Raamatu sisu asub peatükkideks eraldatuna HTML failides. Iga peatüki kohta on eraldi fail, mis kannab enamasti nime „split_n“ või „chapter_n“, kus n tähistab peatüki numbrit. Olenevalt EPUB standardi versioonist peavad peatüki failid olema vastavad XHTML 1.1 või HTML 5 standardile.

2. Kirjanduse ülevaade

Antud peatükis kirjeldatakse ilmunud teadusartiklite põhjal olemasolevaid lahendusi ja tehnoloogiaid PDF-i teisendamiseks EPUB ja (X)HTML kujule. Valitud on kolm kõige paremaid tulemusi andnud ning kõige otsesemalt antud töö temaatikaga seotud artiklit. Lisaks kirjeldatakse vabavaralist teeki pdf2htmlEX, mida on kasutatud ka antud töö raames loodud rakenduses.

Varem on PDF-ist EPUB-i konverteerimine olnud suuresti käsitöö, kuid mõnel juhul on rakendatud ka programmaatilist lähenemist. Simone Marinai jt. [13] on kirjeldanud, kuidas konverteerida täisautomaatselt PDF-ist EPUB raamatut, taastades seejuures puuduvad lingid lehealuste märkuste ja raamatus olevate viidete vahel. Antud meetod põhineb tõenäosuslikul mudelil, mis kasutab lõikude tuvastamiseks infot rea taande ning vahede kohta. Lehealused märkused tuvastatakse mudeliga, mis põhineb peatüki sees leiduvate ja lõpus olevate viidete arvulisel suhtel. Tekstiviidete eraldamisel on saavutatud täpsus 99.41%. Samas on mainitud artiklis, et loodud lahendus ei suuda hakkama saada matemaatiliste valemite, tabelite ja kirjanduse loeteludega. Lahenduses on kasutatud PDF-i töötlemiseks IDR Solutions'i teeki jPedal.

Teiseks olemasolevaks lahenduseks struktureeritud teksti eraldamiseks PDF-ist on Alexandru Constantin jt. [14] kirjeldatud lahendus PDFX. Seda on võrreldud ka käesoleva töö raames loodud rakenduses kastutatavate teekidega Poppler ja pdf2htmlEX. Lahendus kasutab kaheastmelist klassifitseerimismeetodit, kus esimesena liigitatakse lehe osad kolmeks komponendiks: tekstiks, piltideks ja taustaks. Seejärel analüüsitakse iga klassifitseeritud elementi, luues selle esinemise kohta esinemiskordade põhjal tõenäosusliku jaotuse. Klassifitseerimisprotsessi teise sammuna proovitakse ühendada semantiliselt sarnased osad loogilisteks üksusteks. Esmalt eraldatakse dokumendi tekstiline osa ning seejärel täpsemad alajaotused. Rakendus on kasutamiseks avalikult kättesaadav [15].

Kolmas lahendus PDF dokumendi töötlemiseks pärineb Jing Fang jt artiklist [16]. Selle meetodi puhul tehakse üldistusi suunal väiksemast ühikust suuremani. Esmalt tuvastatakse tähed, millest moodustatakse read, millest omakorda saadakse teksti plokid. Teksti plokkidest moodustatakse lõigud ning kõige lõpuks leitakse lõikudele loogiline lugemise järjekord. Antud lahenduses ei kasutata levinud algoritmi „X-Y cut" [17], vaid lõikude järjestuse

tõenäosus arvutatakse kahealuselise graafi kaudu, kus iga tipp vastab ühele lõigule ning tippudevaheline serv tähistab tõenäosust, et esimese tipuga tähistatud lõigule järgneb teine. Pilte sisaldavas dokumendis on lahenduse täpsuseks hinnatud 81.4%.

Kirjeldatud lahendused suudavad pakkuda täisautomaatset protsessi küllaltki suure täpsusega, kuid töötavad kasutaja jaoks musta kastina. Lisaks on mainitud kõikide lahenduste juures, et pilte ja lehealuseid märkuseid sisaldavate raamatute puhul ei pruugi rakendus sama head tulemust saavutada. PDFX kohta käivas artiklis on välja toodud, et kolmanda taseme pealkirjade tuvastamine on vaid 42,93% täpsusega. Juhul kui mõni olulisem element, näiteks pealkiri, pilt või tekstilõik jääb märkimata, tuleb kasutajal sellegipoolest muudatused ise sisse viia vastavat HTML faili modifitseerides. Seetõttu on otsustatud, et käesoleva bakalaureusetöö raames valminud rakendus peab olema paindlikum ja andma kasutajale võimaluse kujundada raamat oma käe järgi ilma, et kasutaja peaks omama teadmisi HTML märgendusest.

3. Rakenduse tutvustus

Rakenduse loomise eesmärgiks on võimaldada kasutajal konverteerida PDF formaadis faile EPUB kujule, ilma et tal peaks olema eelteadmisi HTML süntaksi ja EPUB struktuuri kohta. Seega taheti luua kasutajasõbraliku graafilise kasutajaliidesega rakendus, millega oleks mugav märgendada elemente.

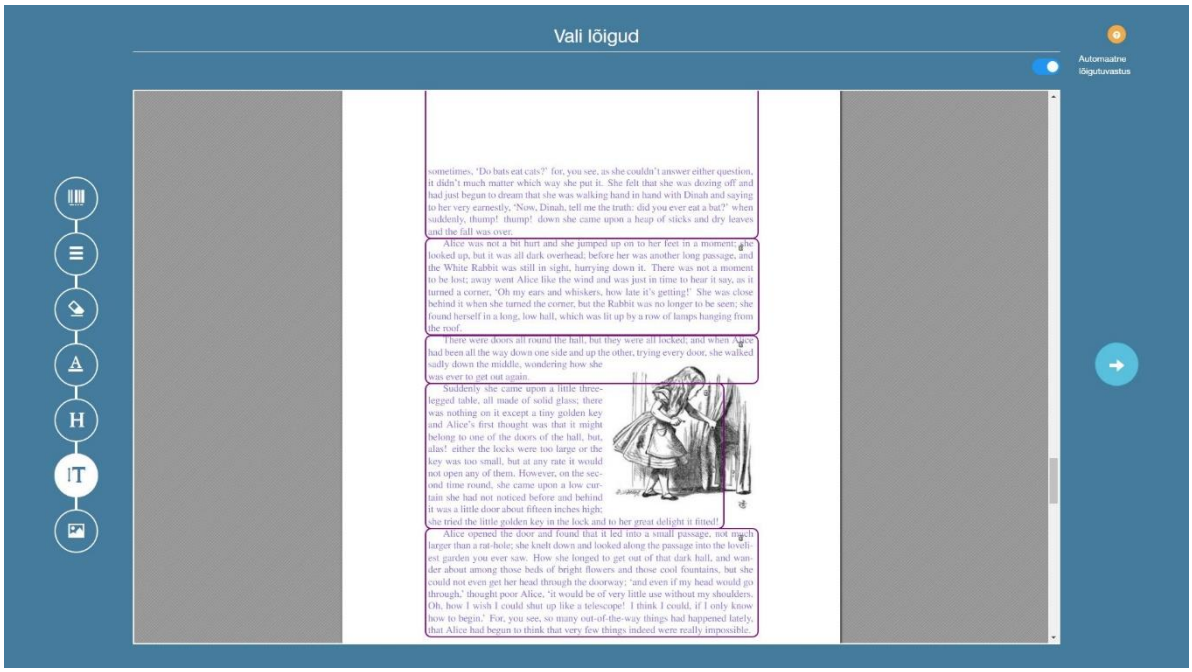
Loodud rakenduses on kasutaja eesmärgiks jaotada raamat peatükkideks ning ära tunda raamatusse kuuluvad elemendid nagu näiteks lõigud, pildid ja pealkirjad. Seejuures tuleb iga eraldiseisev element märgendada eraldi kastiga. Ülesande mugavamaks muutmiseks on loodud töövoog, mis töötab tsükliliselt peatüki kaupa, kuni kõik peatükid on märgendatud.

Kasutaja esimeseks ülesandeks on märgendada tiitellehe pöördel olevad trükise andmed, mida kutsutakse impressumiks, seda tehakse vaid korra. Seejärel tuleb määrata koht käsiloleva ja järgmise peatüki vahel, mille peale dokument jaotatakse kaheks osaks: märgitud kohast üles poole jääb töödeldav peatükk ja allapoole jääb ülejäänud raamat. Järgmise sammuna tuleb ära kustutada e-raamatusse mitte kuuluvad elemendid, milleks on näiteks leheküljenumbrid. Seejärel on tarvis märgendada lehealused märkused, mis hiljem lisatakse peatüki lõppu ja lingitakse tekstisisese viitega. Järgmisena on vaja märgendada lõigud, mille käigus on võimalik lasta programmil lõigud märgendada ning hiljem üle vaadata ja vajadusel korrigeerida programmi automaatset märgendust. Kui lõigud on märgendatud, jäävad alles vaid pildid ja tabelid, mis eraldatakse mõlemad rasterkujul. Tsüklilist töövoogu korratakse kuni kasutaja ei märgista enam käsiloleva ja järgmise peatüki vahelist kohta, ehk jõutud on viimase peatükini.

Tsükli lõppedes liigutakse vormi juurde, kus kasutajal on võimalik lisada raamatule metaandmed, kohustuslikeks väljadeks on siinkohal pealkiri, keel, ISBN ja UID identifikaatorid. Ilma nende väljadeta ei oleks loodud e-raamat valiidne.

Valiidseks loetakse EPUB faili, mille peatükid ei sisalda süntaktilisi vigu ja on valiidsed XHTML 1.1 standardi kohaselt. Lisaks peavad kõik manustatud failid nagu näiteks pildid olema registreeritud „content.opf“ failis. Kõik XHTML failid peavad olema registreeritud „toc.ncx“ failis, mis loob ka sisukorra. Seejuures on oluline ka failide järjekord, *mimetype*

fail peab olema EPUB konteineris alati esimesel kohal.



Joonis 2. Kuvatõmmis lõikude valimise tööriistast

Joonisel 2 on rakenduse kuvatõmmis, millel on kujutatud lõikude märgendamist, lillat värvi kastides on rakenduse märgendatud lõigud. Igat kasti on võimalik kasti servast kinni võttes venitada suuremaks ning väiksemaks ning kustutada, vajutades prügikasti märki üleval paremas nurgas. Rakenduse vasakus ääres on sammulugeja, mis näitab, millise sammu juures parajasti ollakse. Üleval paremal nurgas olev oranž küsimärgiga nupp kuvab vastavalt tööriistale abiteksti ning nool paremale rakendab hetkel kasutuses olevat tööriista ja liigub edasi järgmise sammu juurde.

Tööriistad on vastavas järjestuses lõikude märgendamise hõlbustamiseks. Juhul kui lehekülje numbreid ning lehealuseid märkuseid ei eemaldataks enne lõikude märkimist, oleks keerulisem märgendada lõike, mis jätkuvad järgmisel leheküljel, ilma seejuures ebavajalikke elemente märgendamata. Lisaks on kergem ära tunda lõike, kui puuduvad segavad elemendid, milleks on ka pealkirjad.

4. Teksti eraldamine ja lõikude tuvastus

Peatükk keskendub tehnilistele lahendustele teksti eraldamiseks PDF dokumendist. Teises alapeatükis tutvustatakse vabavaralist teeki pdf2htmlEX ning selle kasutamist töö raames loodud rakenduses. Lisaks kirjeldatakse joonealuste märkuste eraldamist ja linkimist raamatuga. Viimasena käsitletakse lõikude tuvastamist antud rakenduse kontekstis.

4.1 Teksti eraldamine PDF dokumendist

PDF dokumendid, millega programmi testiti, ei olnud markeeritud, ehk puudusid märgendid elementide tüübi kohta, mistõttu oli vaja leida viis teksti eraldamiseks koos elementide äratundmisega. Algse lahendusena prooviti teksti eraldada PDF dokumendist, kasutades selleks Oracle poolt arendatavat teeki PDFBox. Sellega õnnestus küll eraldada tekst PDF-ist, kuid tulemus ei olnud piisavalt kvaliteetne ja töökindel, et kirjastus seda väljaantavate raamatute puhul kasutada saaks. Teksti eraldamise käigus läks kaduma suur hulk raamatu jaoks vajalikku informatsiooni, mida hiljem taastada ei saanud. Näiteks kaotas dokument teksti eraldamise käigus oma struktuuri ning paljudel juhtudel oli mitu lõiku üksteise külge kleebitud. Lisaks kadus ära eraldi vormindus pealkirjadele ning lehealustele, mille tõttu hiljem neid määrata oleks võimalik vaid asukoha põhjal. PDFBox-i abil loodi musta kasti sarnane programm, kus ühest otsast läheb sisse sisend ning peale nupuvajutust tuleb teiselt poolt välja väljund, milleks on algse PDF dokumendi toortekst. Sellisel viisil saadud väljund ei pruugi aga kasutajat rahuldada ning seetõttu sooviti muuta programmi interaktiivsemaks, andes kasutajale protsessi üle parem kontroll. Interaktiivsuse lisamiseks loodi süsteem, kus kasutaja ülesandeks on visuaalselt ära tunda ning märgistada erinevad teksti osad nagu näiteks pealkirjad, lõigud ja lehealused. Tulemuseks oli vaja saada iga peatüki kohta eraldi hüpertekstdokument, milles teksti elemendid on korrektselt märgendatud vastavalt XHTML 1.1 standardile.

Kõige otstarbekamaks viisiks osutus konverteerida PDF esmalt HTML kujule, mis võimaldab rakendada sellele erinevaid JavaScript'is loodud tööriistu. Vastavale kujule konverteerimiseks kasutati pdf2htmlEX teeki, mille abil loodi sisendfailile vastav HTML fail (täpsem kirjeldus eelmises peatükis). Väljundina saadud HTML failis oli iga teksti rida eraldi *div* elemendis ning selles sisalduvad elemendid segaselt struktureeritud. Rida eraldavas *div* elemendis olid sõnad, tähed ja tühikud ebakorrapäraselt jaotatud *div* konteineritesse, mille vahel olid *span* elemendid algse vorminduse säilitamiseks.

Töövoost lähtuvalt osutus parimaks teksti eraldamise variandiks kasutada JavaScript'is kirjutatud tööriistu. Selle abil märgendab kasutaja osa dokumendist kastiga, mille peale otsitakse dokumendi ridasid tähistavate *div* elementide seast välja need, mille koordinaadid klapivad kasutaja joonistatud kastiga. Selekteeritud elemendid värvitakse vastavalt kasutatavale tööriistale, et kasutajale anda märku, milline osa dokumendist on reaalselt eraldatav. Juhul kui kasutaja joonistatud kastis tekst värvi ei muuda, ei kata kast ühtegi rea elementi täielikult ja seetõttu ei saa sellest eraldada ka teksti. Sellisel juhul tuleb kasutajal valitud ala suurendada, et mahutada sellesse vähemalt üks rea element.

Rea elemendi leidmisel eraldatakse sellest tekst, kasutades *textContent* atribuuti, mis tagastab DOM elemendi ning selle laste tekstilise sisu. Eraldatud tekstile lisatakse vajalikud hüpertekstmärgistused ning sorteeritakse asukoha järgi, arvestades selekteeritud elemendi absoluutset kõrgust dokumendis. Hilisem sorteerimine on vajalik, et säilitada teksti algne järjestus lehel, juhul kui kasutaja ei vali elemente järjestatult. Sorteeritud list elementidega saadetakse serverile, kus see lõpuks raamatu sisule liidetakse.

4.2 Pdf2htmlEX kasutamine

Kõige optimaalsemaks lahenduseks PDF-ist EPUB-i konverteerimisel osutus esmalt konverteerida PDF dokument HTML kujule. Selle ülesande sooritamiseks otsustati kasutada vabavaralist teeki pdf2htmlEX.

Pdf2htmlEX on vabavaraline teek litsentsiga GPLv3, mille abil on võimalik teisendada PDF fail HTML kujule säilitades seejuures võimalikult originaalilähedast kujundust [18]. Teek on kirjutatud C++ keeles ja seetõttu on seda kasutatud Java projektis välise teegina. Pdf2htmlEX võtab sisendiks eeltöötlemata PDF faili ja loob väljundiks HTML lehe koos CSS vormis kujundusega, mis olenevalt konfiguratsiooni parameetritest võib olla nii eraldi failina kui ka HTML failis manustatud. Sarnaselt saab ka valida, kas pildid, font ja JavaScript failid on manustatud kujul või lisatakse väljundile eraldi failidena.

Selleks, et väljundiks olevas HTML failis oleks algse PDF dokumendi kujundus maksimaalselt säilinud, koostab teek iga lehe ja sellel leiduva rea kohta eraldi HTML elemendi *div*. Rida ümbritsevatele *div* elementidele antakse spetsiifilised kujundusreeglid, määrates igale elemendile *class* atribuudi, mis kasutab varem kirjeldatud kujundusreegleid CSS failist. Rida ümbritsevate *div* elementidele lisatakse ka *span* elemente, mille eesmärgiks on kas tekitada joonduse säilitamiseks piisavalt tühikuid või edasi anda mõnda kirjatüüpi.

Eelnevalt mainitud struktuuri tulemusena valmib HTML leht, mis näeb algsele PDF dokumendile sarnane välja, kuid on sisemiselt keerulise struktuuriga HTML dokument [19]. Seetõttu ei saa selle teegi väljundit otse kasutada e-raamatu loomiseks, kuid see annab hea alus-põhja, millest saab erinevate tööriistade abil vajaliku välja sõeluda ja puhastada üleliigsest mürast.

Käsk, millega antud teeki kasutatakse, on järgmine:

```
pdf2htmlEX --fit-width 800 --hdpi 200 --vdpi 200 --font-size-multiplier 10 --
heps 0.5 --veps 0.5 --correct-text-visibility 1 --dest-dir väljundkaust pdf_fail
```

Käsus väljendab pdf2htmlEX funktsiooni nime ja sellele järgnevad järgmised argumendid:

- --fit-width, mis määrab väljundi lehe maksimaalse laiuse pikslites, mis on vajalik erandjuhtudeks, kui sisestatud PDF dokument on ebastandardse suurusega
- --hdpi määrab lahutusvõime horisontaalsel tasandil. Seda mõõdetakse ühikus dpi (eesti keeles täppi tolli kohta).
- --vdpi määrab vertikaalse lahutusvõime tasandil, mis on määratud samuti dpi ühikus
- --font-size-multiplier on vaja korrektseks teksti eraldamiseks, sest paljud veebilehitsejad on määranud ära minimaalse fondi suuruse, mida kuvatakse. Sellest alla poole jäävat sisu ei kuvata
- --heps kirjeldab horisontaalsel tasandil maksimaalset lubatud nihet pikslites. Pdf2htmlEX optimeerib teksti paigutust antud nihke piires
- --veps kirjeldab vertikaalsel tasandil maksimaalset lubatud nihet
- --correct-text-visibility hoiab ära mõningatel juhtudel teksti kattumise ja parandab loetavust
- --dest-dir on kaust, kuhu kirjutatakse väljundfail

Pdf2htmlEX teegi pakutava konverteri seadistusvõimalused on tunduvalt laiemad kui töös kasutatud, kuid ülejäänud seadistustel piisas vaikeväärtustest.

Tulemusena tekkis määratud väljundkausta PDF failiga samanimeline HTML fail, mis on vorminduselt väga sarnane algsega. Väljundfaili juurde kuuluvad JavaScript ja CSS failid on manustatud kujul, et vältida vigu faili hilisema serveerimise käigus.

4.3 Joonealuste märkuste eraldamine ja linkimine

Joonealuste märkuste eraldamine on tavalise teksti eraldamisest mõnevõrra erinev, sest lisaks joonealuse teksti eraldamisele tuleb see siduda peatüki sisus oleva viitega. Joonealuste märkuste eraldamiseks tuleb kasutajal esmalt kursoriga valida tekstist lingitav element, mis viitab joonealusele märkusele. Selleks võib olla näiteks tärn või number. Seejärel tuleb vajutada noole sümbolit selekteeritud teksti juures ning tõmmata kast ümber lehealusele märkusele, millele teksti element viitab. Loodava seose kinnitamiseks tuleb vajutada luku sümbolit. Tulemusena tekitatakse loodavas e-raamatus link kahe elemendi vahel navigeerimiseks.

Kursoriga teksti selekteerimiseks kasutatakse JavaScript meetodit *Window.getSelection()*, mis tagastab selekteeritud teksti vahemiku *range* objektina, millelt saab sisu pärida *toString()* meetodiga. Saadud sisu tõstetakse esile seda siniseks värvides ning sellest ülaindeksit luues. Elemendi juurde lisatakse nupud valiku eemaldamiseks ning järgmise elemendi valimiseks.

Lehealuse märkuse eraldamisel kastiga ning selle kinnitamiseks luku märgile vajutamisel luuakse link, millele lisatakse loomise järjekorrale vastav id, et vastavad elemendid omavahel siduda. Lehealusele märkusele lisatakse ka tagasiviitava noolega nupp, millele vajutades liigutakse tagasi vastava tekstiviite juurde.

4.4 Lõikude tuvastamine

Lõikude tuvastamiseks kasutatakse antud rakenduses JavaScript'i, kus vaadeldava dokumendi osast eraldatakse kogu tekst ridade kaupa. Kõige efektiivsemaks viisiks osutus analüüsida ridu regulaaravaldistega, et kindlaks teha, kas tegemist võib olla potentsiaalse lõiku alustava või sulgeva reaga. Read grupeeritakse vastavalt tulemusele ning märgendatakse kastidega.

Rakenduse loomisel katsetati ka algoritmi, mis on inspireeritud Simone Marinai jt [13] loodud lahendusest, kus tekstiread jaotatakse lõikudesse vastavalt nende sisule ja paiknevusele lehel. Kui rida algab suure tähe ja taandreaga suurendatakse tõenäosust, et tegemist võib olla lõiku alustava reaga. Lisaks võrreldakse distantsi järgneva ning eelneva reaga ning vaadatakse, kas rida lõppeb lauselõpu märgiga. Tõenäosust, et tegemist on lõiku lõpetava reaga

hinnatakse eelnevalt mainitud parameetritele lisaks ka rea pikkuse järgi. Rea pikkus arvutatakse tähemärkide arvu pealt, kategoriseerides tähemärgid laiuse järgi kolme klassi. Lisaks liidetakse rea pikkusele konstant vastavalt sellele, kas rida sisaldab taanet säilitavaid *span* elemente. Kogutud andmete põhjal arvutatakse iga rea puhul tõenäosus, et tegemist on lõiku alustava või lõpetava reaga. Potentsiaalsed lõigud kategoriseeritakse vastavalt leitud tõenäosustele.

Rea asukohta ja sisu arvestava lähenemise kitsaskohtadeks on juhud, kus lõiku poolitab pilt, lehekülje lõpp või muu element. Sellistel juhtudel eksis algoritm enamus kordadel ning seetõttu ei ole lõpplahenduses kasutusel. Käesoleva töö raames loodud rakenduses lihtsustab lõikude eraldamist asjaolu, et lehekülje numbrid ning lehealused märkused eemaldatakse enne lõikude eraldamist dokumendist. Seetõttu suudab ka regulaaravaldisega tuvastus pakkuda piisavat täpsust, olles seejuures tunduvalt kiirem eelmainitud algoritmist.

5. Kujunduselementide säilitamine ja raamatu koostamine

Antud peatükis on töö raames loodud rakendusest lähtuvalt kirjeldatud kujunduselementide nagu näiteks piltide ja kirjaviiside säilitamist. Lisaks on kirjeldatud EPUB formaadis raamatu koostamist vabavaralise Java teegi Epublib abil.

5.1 Piltide ja tabelite eraldamine PDF dokumendist

Pilte hoitakse PDF dokumendis binaarandmetena, kirjeldades iga pikslit eraldi oma värvusruumis. Seetõttu on otse PDF-ist piltide eraldamine küllaltki keerukas. PDF-ist on võimalik eraldada pilte, kasutades selleks spetsiaalseid PDF töötlemiseks loodud teeke nagu näiteks Poppler, PDFBox või IText.

Algselt loodi rakenduses süsteem, mille abil eraldati pildid sisendfailist, kasutades Java teeki iText. Piltide eraldamine dokumentidest sellise lähenemise abil küll õnnestus, kuid lisaks soovitud piltidele eraldati ka rasterkujul olevad kujunduselemendid nagu lehekülje päises olevad jooned ja lõike eraldavad vahejooned. Lisaks tekkis probleem eraldatud piltide sobitamisega algsetesse asukohtadesse ja nende juurde kuuluvate allkirjade säilitamisega ning korrektse kujutamisega.

Mainitud probleemide tõttu otsustati võtta alternatiivne suund, andes kasutajale võimalus määrata manuaalselt pildiks eraldatav osa leheküljest. Eesmärgi saavutamiseks otsustati luua piltide eraldamiseks ja loomiseks sarnane süsteem ülejäänud sisu eraldamisele, kus kasutaja ülesandeks on hiirega selekteerida lähtedokumendist sobiv osa.

Piltide eraldamiseks HTML dokumendist kasutati vabavaralist JavaScript teeki Html2Canvas [20]. Antud teek on MIT litsentsiga, mistõttu võib seda kasutada ka ärilistel eesmärkidel, lisades tarkvarale koopia antud autorikaitse õigustest. Html2Canvas teek on loodud hüpertekst-dokumendist rasterkujutise loomiseks. Kujutis luuakse, kasutades HTML5 standardis tutvustatud elementi *canvas*. Antud teek on mõeldud tervest hüpertekst-dokumendist pildi tegemiseks, kuid loodud rakenduses oli vaja luua kujutis ainult kasutaja märgendatud alal. Soovitud pildi saamiseks leitakse esmalt kasutaja märgendatud kasti koordinaadid ning seejärel luuakse koopia märgendatud lehest ning selle vahetult kohal ja all olevatest lehtedest. Kopeeritud lehtedele rakendatakse Html2Canvas teeki, mis loob *canvas* elemendi, millel on kujutatud kolm kopeeritud lehte rasteriseeritud kujul. Seejärel leitakse *canvas* elemendilt kasutaja joonistatud algse kasti koordinaadid ja lõigatakse kujutis vastavasse suurusesse.

Eelmainitud operatsioonide tulemusena saadi *canvas* element, mis sisaldab kasutaja valitud osa hüpertekst-dokumendist, mis on saadud algsest PDF dokumendist, kasutades pdf2htmlEX teeki.

Kasutatava pildi eraldamiseks *canvas* elemendist kasutatakse järgmist JavaScript meetodit:

```
canvas.toDataUri(type)
```

Meetod loob *canvas* elemendist Base64 kodeeringus kujutise, millele saab eraldi täpsustusena anda DOMString kujul tüübi, näiteks "image/png". Loodud pilt on punktitihedusega 96 dpi, mis on piisav e-lugeri või tahvelarvuti ekraanilt vaatamiseks.

Vanematel e-lugeritel ei tohi kasutajakogemuse parandamiseks ühegi peatüki HTML faili suurus ületada 300 KB, mistõttu ei saa pilte Base64 kujul lisada, sest ka väikese pildi lisamine manustatuna võib tõsta faili suurust üle lubatud normi. Probleemi vältimiseks tuleb pildid salvestada PNG kujul ning lisada raamatusse manuaalselt. Selle jaoks on rakenduses loodud Java funktsioon, mis salvestab Base64 kodeeringus pildi PNG formaadis ning tagastab salvestatud pildi nime. Pildi dekodeerimise ja PNG formaati kirjutamisel salvestatakse pilt ajutiselt failisüsteemi ning tagastatakse pildi nimi, mis lisatakse peatüki dokumenti. Eraldatud PNG kujul pilt lisatakse raamatule hiljem, kasutades Epublib teeki. Peale raamatu loomist kustutatakse pilt failisüsteemist.

5.2 Fontide eraldamine ja kirjaviiside säilitamine

Enamikul juhtudel on fondid PDF failides manustatud kujul, et säilitada algne vormistus sõltumatult faili kuvavast süsteemist. Pdf2htmlEX eraldab fondid dokumendist, kasutades selleks teeki Poppler, ning kasutab neid väljundiks loodavale hüpertekst-dokumendile võimalikult originaalilähedase kujunduse andmiseks. Vaikesättena on eraldatavad fondid väljund HTML failis WOFF formaadis manustatud kujul. Nii fondi formaati kui ka selle manustamist saab muuta vastavate pdf2htmlEX teegi käsurea argumentidega. Juhul kui sisend-PDF ei sisalda informatsiooni kasutatud fontide kohta, proovib pdf2htmlEX teek luua võimalikult sarnase alternatiivi, kombineerides tuntumaid fonte või asendades vaikefondiga.

Pdf2htmlEX väljund HTML dokumendis on kasutatud fontide esitamiseks CSS reeglit "@font-face". Antud reeglile saab rakendada atribuute, millega on võimalik määrata fondi nime, allikat ning kirjastiili ehk, kas font peaks olema kursiivis või poolpaksus kirjas. Selleks, et rakendada antud reeglit tekstile tuleb määrata teksti sisaldava DOM elemendi "font-family" atribuut vastavalt varem mainitud "@font-face" reeglis olevale nimetusele.

Pdf2htmlEX teegis on tekstile fondi rakendamiseks iga "@font-face" reegli kohta loodud CSS reegel, mis kasutab "font-family" atribuuti. Iga selline reegel kannab sama nime, mis antud "font-family" atribuut. Times New Roman fondiga ja suurusega 12, teksti esitamiseks kasutab antud teek järgmist koodi:

```
@font-face {
   font-family: ff1;
   src: url(data:application/font-woff) format("woff");
}

.ff1 {
   font-family: ff1;
   line-height: 0.934082;
   font-style: normal;
   font-weight: normal;
   visibility: visible;
}

<div class=ff1>Selle elemendi sisu on fondiga ff1</div>
```

Fontide järjestikusel eraldamisel kaotab pdf2htmlEX ära nende nimed ning asendab need tähisega "ff", millele järgneb eraldamise järjekorranumber, alates ühest. Näiteks, kui dokumendis esineb font Times New Roman esimesena ning Verdana teisena, määratakse fontide nimedeks vastavalt *ff1* ja *ff2*. Fontide nummerdamine käib seejuures kuueteistkümnendsüsteemis, mis on teegi pdf2htmlEX puhul valdav. Sarnaselt fontidele on kuuteistkümnendsüsteemis nummerdatud ka lehekülje identifikaatorid ning ridade numbrid.

Kursiivi ja poolpaksu kirja esitamiseks hüpertekst-dokumendis on mitmeid võimalusi, neist kõige levinumad on CSS reeglite abil *font-style* ja *font-weight* atribuutide määramine, või *em* ning *strong* siltide kasutamine HTML-is. Pdf2htmlEX teek annab teksti stiile edasi ebastandardsel viisil kasutades selleks ainult erinevaid fonte. Ühe ja sama fondi erinevad variatsioonid, näiteks *bold* ja *italic*, eraldatakse kahe erineva fondina ning rakendatakse tekstile sarnaselt varem toodud näitele.

Rida eraldavale *div* elemendile määratakse fonti kirjeldav klass alati selle rea esimese sümboli järgi. Kui antud reas leidub ka teise teksti fondi või stiiliga elemente, siis luuakse rida eraldava *div* elemendi sisse lisaks *span* element, millel on määratud järgmist fonti kirjeldav klass. Kui reas on rohkem erinevaid elemente, siis korratakse sama algoritmi, ehk lisatakse eelneva elemendi sisse täpsustav *span* element, millel on font klassina määratud. Näiteks, kui PDF dokumendis on tekst "**Bold** tavaline *Italic*" ehk dokument koosneb vaid ühest reast

ning sisaldab kolme erinevat teksti stiili, siis luuakse antud teksti kirjeldamiseks järgmine HTML struktuur:

```
<div class="t m0 x0 h1 y0 ff1 fs0 fc0 sc0 ls0 ws0">
   Bold
   <span class=ff2>
      ta
      <span class="_ _0"></span>
      valine
      <span class="ff3">
         I
         <span class="_ _0"></span>
         tali
         <span class="_ _0"></span>
         c
      </span>
   </span>
</div>
```

Ülaltoodud näites on poolpaksus kirjas märgistatud fontide kujutamiseks olulised klassid *ff1*, *ff2* , *ff3* ning eraldatud teksti osad. Teksti vahel asuvad *span* elemendid, klassiga "_ _[number]", mis on vajalikud teksti joonduse säilitamiseks. Antud näitest on näha, et iga uue teksti stiili puhul eraldatakse see eraldi fondina ning rakendatakse vastavalt ülaltoodud algoritmile, kus iga järgnev element on eelneva sisse paigutatud ning sisemisele elemendile määratud klass fondi kirjeldamiseks kirjutab üle väliste klasside väärtused.

```
<div class="t m0 x0 h1 y0 ff1 fs0 fc0 sc0 ls0 ws0">
   tavaline
   <span class=ff2>
      bold
      <span class=ff1>
         tavaline
      </span>
   </span>
</div>
```

Ülaltoodud koodi näites on pdf2htmlEX väljund reast, millel on kolm sõna „tavaline **bold** tavaline". Loetavuse huvides on näitest eemaldatud varem mainitud joonduse säilitamiseks vajalikud *span* elemendid. Vaheldumisi olevate elementide puhul rakendatakse sama loogikat, mis varemgi ehk kui rida on kujul "tavaline **bold** tavaline", sisaldab rida eraldav *div* element klassi *ff1*, mis tähistab tavalist kirjatüüpi. Seejärel on rida ümbritseva elemendi sees *span* element, millel on klass *ff2* poolpaksu kirja tähistamiseks. Rea lõpus olevat tavalist

kirjatüüpi tähistatakse omakorda poolpaksu kirja tähistamiseks mõeldud *span* elemendi sees paikneva tavalise kirjatüübi tähistamiseks loodud *span* elemendi klassiga *ff1*.

Kõige lihtsam variant kirjastiilide säilitamiseks oleks eemaldada joondust säilitavad *span* elemendid ning lisada eraldatud fondid koos vastava CSS-iga väljundiks olevasse e-raamatusse. Selline variant ei osutunud aga võimalikuks, sest fondid võivad olla mõningal juhul autoriõigustega kaitstud, samuti peaks olema fondi valik lõppkasutaja teha.

Tarvis oli leida alternatiivne lahendus, millega oleks võimalik teksti tüübid märgistada HTML siltidega *em* ja *strong*. Lahendusena otsustati kasutada Poppler teeki kuuluvat komponenti *pdffonts*. See on Linuxile mõeldud käsurea rakendus, millega on võimalik PDF dokumendist eraldada selles kasutatud fontide nimed ning informatsioon nende manustamise kohta [21]. Sama teeki kasutab fontide eraldamiseks Poppler'i kaudu ka pdf2htmlEX.

Pdffonts rakendus leiab PDF dokumendis kasutatud fontide nimed nende kasutamise järjekorras. Seetõttu on võimalik leida üksühene seos *pdffonts* väljundi ja selliste väljund HTML-i fontide vahel, millel on nimi ära kaotatud.

Rakenduses on loodud Java klass FontExtractionUtil, mille meetod *extractFonts(Path)* eraldab PDF dokumendist fontide nimed, kasutades *pdffonts* rakendust. Samuti tagastab antud meetod MultiMap tüüpi objekti, mille võtmetele "bold", "italic" ja "boldItalic" vastavad PDF dokumendist leitud fondid, mille nimes leiduvad parajasti vastavad fraasid. Tulemuseks saadakse MultiMap, mis on järgmisel kujul:

```
{"bold":["ff1", "ff3", "ff4"], "italic": ["ff2", "ff5"], "boldItalic": ["ff6"]}
```

Saadud MultiMap tüüpi objekt saadetakse JavaScripti, kus rekursiivne funktsioon lisab teksti eraldamise käigus poolpaksule kirjale HTML sildi *strong* ning kursiivile *em*.

5.3 Raamatu koostamine teegiga Epublib

Epublib on vabavaraline Java teek, mis võimaldab EPUB formaadis e-raamatuid lugeda ja kirjutada [22]. Epublib'il on LGPLv3 litsents, mis tähendab, et seda tohib kasutada ka ärilistel eesmärkidel, kuid dokumentatsioonis tuleb antud tarkvarale viidata. Teegil on kaasas ka graafiline kasutajaliides, kuid antud töös on kasutatud seda vaid Java teegina ilma graafilise kasutajaliideseta. Epublib võimaldab loodavale raamatule lisada metaandmeid ja sisu, ilma et peaks sisukorda manuaalselt koostama ja kaasapandavaid faile registreerima.

Raamatu koostamiseks luuakse esmalt uus objekt *Book*. Raamatule lisatakse peatükke meetodiga *addSection(Resource)*, mis võtab argumendiks klassi Resource isendi. Igale peatükile saab määrata ka pealkirja, mis hiljem kuvatakse, andes see Resource konstruktorile sõne kujul ette. Loodud rakenduses määratakse sisukorras iga peatüki nimeks selle peatüki esimene pealkiri.

Loodavale raamatule on võimalik lisada ka metaandmeid, kasutades selleks *Metadata* objekti, millele on võimalik lisada standardseid raamatusse kuuluvaid andmeid nagu pealkiri ja autor, kui ka oma loodud välju, näiteks identifikaator. Antud rakenduses lisatakse igale raamatule standardsed metaandmed, milleks on pealkiri, autor, kirjastaja, keel, teema ja kirjeldus, ning lisaks ka kaks identifikaatorit - ISBN ja UID. ISBN on rahvusvaheline raamatu standardinumber ja UID on unikaalne identifikaator raamatu üheseks määratlemiseks.

6. Implementatsiooni protsess

Antud peatükis kirjeldatakse töövoogu rakendusest lähtuvalt, tuues välja suhtluse serveriga ning taustal toimuvad protsessid. Teises alapeatükis kirjeldatakse, kuidas on lahendatud erinevate tööriistade vahel navigeerimine ja tööriistade vahetamine. Lisaks kirjeldatakse rakenduse loomisel tekkinud peamisi tehnilisi ja põhimõttelisi probleeme ning nende üldiseid lahendusi. Viimases alapeatükis antakse ülevaade autori otsesest panusest rakenduse loomisel.

6.1 Töövoog rakendusest lähtuvalt

Rakenduse töövoogu on visualiseeritud BPMN-na lisas 1. PDF dokumendi üles laadimisel esmalt valideeritakse dokument ja sõltuvalt tulemusest pöördutakse järgmise sammu juurde või antakse kasutajale teada veast ning palutakse laadida üles valiidne PDF. Seejärel konverteeritakse üles laaditud PDF HTML-iks kasutades pdf2htmlEx teeki ning eraldatakse sellest fondid kasutades Poppler pdffonts teeki. Niipea, kui konverteerimine on lõppenud saadetakse loodud HTML kliendi arvutisse ning luuakse serveris tühi EPUB raamat.

Kliendi veebilehitsejas kuvatakse konverteeritud HTML ning lisatakse sellele tööriistad, millega seda töötlema saab hakata. Seejärel päritakse serverilt info fontide kohta, mis eraldati eelmises sammus. Info fontide kohta saadetakse JSON formaadis.

Esimese ülesandena tuleb kliendil ära tunda raamatust impressum, mis asub enamasti tiitellehe pöördel. Juhul kui raamatus impressum puudub, saab selle sammu ka vahele jätta. Impressumi märgendamisel eraldatakse vastavad HTML elemendid ning saadetakse serverisse, kus luuakse vastav impressumi peatükk ning lisatakse see raamatusse. Valiidsuse tagamiseks puhastatakse peatüki HTML kasutades jTidy teeki. Juhul, kui jooksvale peatükile järgneb veel mõni peatükk, on kasutaja ülesandeks dokument poolitada. Poolituskoha valimisel värvitakse kõik sellest joonest üleval pool olevad elemendid siniseks, et eristada jooksvasse peatükki jäävad elemendid. Valiku kinnitamisel lõigatakse HTML märgitud koha pealt kaheks. Üles poole jääv sisu kuvatakse kasutajale ning rakendatakse sellele järjestikuselt tööriistu. Allapoole jääv sisu salvestatakse mällu, et seda saaks hiljem töödelda.

Tööriistad, ehk JavaScript failides olevad funktsioonid erinevat tüüpi sisu eraldamiseks, on rakendatud dokumendile järjestikuselt. Vastav järjestus on tingitud vajadusest eemaldada

lehekülgede lõpus olevad elemendid nagu lehekülje numbrid ja märkused enne lõikude eraldamist, kus mõningatel juhtudel jätkub lõik järgmisel leheküljel ja vastasel juhul märgendataks ka ebavajalikud elemendid.

Viimaseks tööriistaks on piltide eraldamine, kus luuakse kasutaja märgendatud kastiga lehest ning sellele vahetult eelnevast ja järgnevast lehest koopia ning eraldatakse pilt kasutades html2canvas teeki. Saadud kujutis lõigatakse välja vastavalt kasutaja märgendatud ala koordinaatidele. Pilt teisendatakse base64 kujule ning saadetakse seejärel serverisse. Serveris eraldatakse pilt PNG faili, lisatakse pooleliolevasse raamatusse ning tagastatakse faili viide. Tagastatud viide lisatakse vastavasse HTML elementi.

Peale piltide eraldamise tööriista sorteeritakse kõik eraldatud elemendid nende positsiooni järgi algses raamatus, et säilitada algne lugemisjärjekord sõltumatult kastide tõmbamise järjekorrast. Seejärel saadetakse sorteeritud elemendid serverisse. Serveris korratakse taas peatüki loomise loogikat, kus kõik elemendid lisatakse peatüki HTML faili, puhastatakse ning lisatakse raamatusse. Juhul kui kasutaja märkis ära peatüki poolitamise asukoha, pöördutakse tagasi peatüki poolitamise funktsionaalsuse juurde ning kasutajal on võimalus märkida poolitamiskoht järgmise peatüki vahel. Kui on jõutud viimase peatükini, siis enam poolitamiskohta ei märgita ning peale pildi eraldamise tööriista ja eraldatud HTML elementide serverisse saatmist väljutakse töövoo tsüklist ning liigutakse järgmise sammu juurde.

Viimane kasutajapoolne tegevus on metaandmete lisamine. Kohustuslikud väljad on pealkiri, keel, ISBN ja UID identifikaatorid. Lisaks on võimalik lisada kaanepilt ja tiitellehe pilt. Piltide lisamisel luuakse mõlema jaoks eraldi peatükid ning lisatakse raamatusse. Peale metaandmete lisamist raamat valideeritakse. Kasutajal on võimalik näha valideerimistulemust ning laadida raamat alla.

6.2 Tööriistade vahetamine

Iga tööriist on erineva ülesandega ja seetõttu mõnevõrra erineva ehitusega. Kõiki tööriistu ühendab funktsionaalsus tõmmata kaste ja eraldada nende seest sisu. Edasine sisu töötlemine sõltub aga nõutud väljundist. Selleks, et kaste oleks võimalik hiirega tõmmata, märgitakse HTML-iks teisendatud sisendfailis iga hiire vajutus. Vajutuse jälgimine jQuery *ready()* funktsiooni abil käivitatakse kohe, kui leht on laaditud. Iga tööriist on eraldi Ja-

vaScript failis ning kutsutakse dokumendi laadimise peale välja. Seega tööriistade vahetamiseks muudetakse HTML elemendis iFrame vastava tööriista skripti viidet ja laetakse leht uuesti, rakendades selle sisule eelnevat tööriista.

6.3 Rakenduse loomisel tekkinud tehnilised probleemid

Rakenduse loomise käigus tekkis palju tehnilisi takistusi, kuid selles alapeatükis käsitletakse vaid kõige läbivamaid ja keerulisemaid. Rakenduses on kasutatud palju ebastandardseid lahendusi, et saavutada soovitud tulemus.

Esimese suurema probleemina leiti, et mõnel juhul kaotab pdf2htmlEX väljund oma vorminduse ning osa tekstist hakkab kattuma, mistõttu muutub dokument loetamatuks. Lähemal uurimisel selgus, et kui loodud HTML faili parsida Java XML parseriga jSoup, muudetakse automaatselt dokumendi taanet, et muuta dokument seeläbi loetavamaks. Taande muutmisel lisati aga automaatselt tühikuid ning dokument kaotas oma algse vorminduse ning muutus mõnel juhul seeläbi loetamatuks. Lahenduseks oli dokumenti käidelda tavalise tekstifailina, mis vältis taande muutmist ning sellega kaasnenud probleeme.

Teise suurema probleemina ei õnnestunud piltide eraldamine JavaScript teegiga Html2Canvas, kui sisendfaili pikkus oli üle kolmekümne lehekülje. Sellisel juhul eraldati tühi pilt, mis koosnes ainult valgetest pikslitest. Selgus, et probleemi põhjustas veebilehitsejate piirang *canvas* elemendi suurusele. Nimelt saab *canvas* element olla Google Chrome ja Mozilla Firefox veebilehitsejates maksimaalselt 32 767 pikslit lai ja kõrge. Internet Exploreri puhul on selleks piiranguks 8192 pikslit. Html2Canvas võtab argumendiks HTML elemendi, milles soovitakse rasteriseeritud kujutist luua. Algses versioonis anti argumendiks kogu HTML fail, väiksemate üksuste puhul pildi tegemine mõnel juhul ebaõnnestus ning tulemuseks oli kas osa kujutisest või vale vormindusega kujutis. Hiljem lõigati rasteriseeritud kujutisest välja kasutaja poolt märgendatud ala. Kui dokumendi kõrgus oli suurem kui veebilehitseja piirang lubas, tagastati hoopis maksimaalse suurusega valge leht, millest hiljem lõigati välja valge kujutis, mis ei sisaldanud vajalikku informatsiooni. Lahenduseks klooniti pildi tegemise hetkeks dokumendist vaid vaadeldav leht ning vahetult selle all ning kohal asuvad lehed ning uuele loodud objektile rakendati Html2Canvas teeki. Sellisel juhul jääb lehekülgede arv alati maksimaalse suuruse piiresse ning varem mainitud probleemi ei teki. Lisaks on selline lahendus efektiivsem, sest kujutis luuakse vaid vajalikust osast dokumendist, mitte kogu dokumendist.

Kolmandaks keerulisemaks probleemiks osutus väljundina loodava EPUB faili valiidsuse tagamine. Juhtudel, kui kasutaja eksib ning märgib kasti, millest tegelikku sisu ei eraldata, luuakse tühi HTML element. Koos teiste elementidega satuks selline element ka raamatusse ning põhjustaks seeläbi valideerimisel vea. Lisaks vastavad JavaScript'is loodavad elemendid HTML spetsifikatsioonile, kuid EPUB peatüki failid peavad vastama XHTML 1.1 standardile. Nimetatud standardid erinevad sellepoolest, et XHTML standard nõuab kõikide elementide korrektset lõpetamist lõpu sildiga, kuid HTML seda ei nõua. Lisaks sellele on vajalik XHTML 1.1 puhul osad elemendid, nagu näiteks pildid, pakkida konteiner-elementide sisse. Kõikidele eelmainitud valiidsuse probleemidele pakub lahendust Java teek jTidy, millega on võimalik HTML faili puhastada ja korrastada ning muuta seda valiidseks ka XHTML standardi järgi. jTidy teek võimaldab määrata väljundfaili *Doctype* parameetrit meetodiga *setDocType(String)*, mis võtab argumendiks dokumendi tüüpi kirjeldava *fpi* formaadis sõne. jTidy teegi aktiivne arendus on aga lõppenud ning seetõttu ei ole antud meetodit võimalik kasutada. Programm määrab ise, et kõige tõenäolisemalt võib tegemist olla XHTML 1.0 Transitional standardiga ning teeb vastavalt sellele sisendfailis korrektuurid. Lahenduseks on töödelda antud teegiga vaid dokumendi keha ning hiljem töödeldud keha sisestada varem loodud XHTML standardile vastavasse malli. Käsk, millega jTidy teegiga on võimalik muuta HTML dokumenti valiidseks XHTML dokumendiks on järgmine:

```
Tidy tidy = new Tidy();
tidy.setInputEncoding("UTF-8");
tidy.setOutputEncoding("UTF-8");
tidy.setXHTML(true);
tidy.setWraplen(0);
tidy.setPrintBodyOnly(true);
//tidy.setDocType("\"-//W3C//DTD XHTML 1.1//EN\"");
tidy.parseDOM(new ByteArrayInputStream(htmlBody.getBytes()), outputStream);
```

Ridadel kaks ja kolm määratakse ära sisend- ja väljundfaili kodeering, et säilitada täpitähtede korrektne kuvamine. Neljandal real olev käsk määrab väljundfaili tüübi, milleks on XHTML. Järgmisel real olev käsk *tidy.setWraplen(0)* on vajalik, et koodirida loetavuse huvides ei murtaks. Vastasel juhul tekiksid väljundis kohtadesse, kust rida murti, tühikud. Koodis on välja toodud ka *Doctype* parameetri muutmiseks vajalik käsk, mis ei tööta jTidy teegis oleva ja klassi Lexer.java meetodis *fixDocType()* asuva vea tõttu.

6.4 Autori otsene panus rakenduse loomisesse

Rakendus on loodud neljaliikmelise meeskonna poolt. Meeskonda kuulusid Sander-Sebastian Värv, Meelis Tapo, Karl-Mattias Tepp ja Silver Vapper. Autor oli meeskonna juht, vastutades töö koordineerimise eest ning tagades suhtluse EDRK-ga. Süsteemi administreerimise eest vastutas Karl-Mattias Tepp. JavaScript tööriistadega teksti eraldamisega tegeles Meelis Tapo. Autori loodud on rakenduse kasutajaliides koos selle disainiga, JavaScript tööriistade tsüklilise vahetamise funktsionaalsus, rakenduse suhtlus kliendi masina ja serveri vahel, EPUB faili komplekteerimine ning pakendamine, piltide eraldamine eraldi failidesse, fontide ning kirjaviiside säilitamine, lehealuste elementide eraldamine, peatüki failide valiidseks muutmine, sisukorra loomine ning raamatule lisamine ja lõikude tuvastamine, mis on loodud koostöös Meelis Tapoga. Lisaks osales autor funktsionaalsuse loomises, millega on võimalik hinnata konverteerimise ajamahtu lähtuvalt sisend-PDF failist. Paraku eelmainitud funktsionaalsust rakenduse lõppversioonis ei ole ebaproportsionaalselt suure käitusaja tõttu. Ajakulu hindamist oleks võimalik kasutada eeltöötlus faasis, integreerides selle süsteemi, mis haldab kogu konverteerimise protsessi.

7. Konkureerivad rakendused

Selles peatükis võrreldakse loodud rakendust reaalselt kasutuses olevate konkureerivate rakendustega. Võrdlemisel hinnatakse eelkõige konverteerimiseks kuluvat aega, kasutusmugavust ning lõpptulemust.

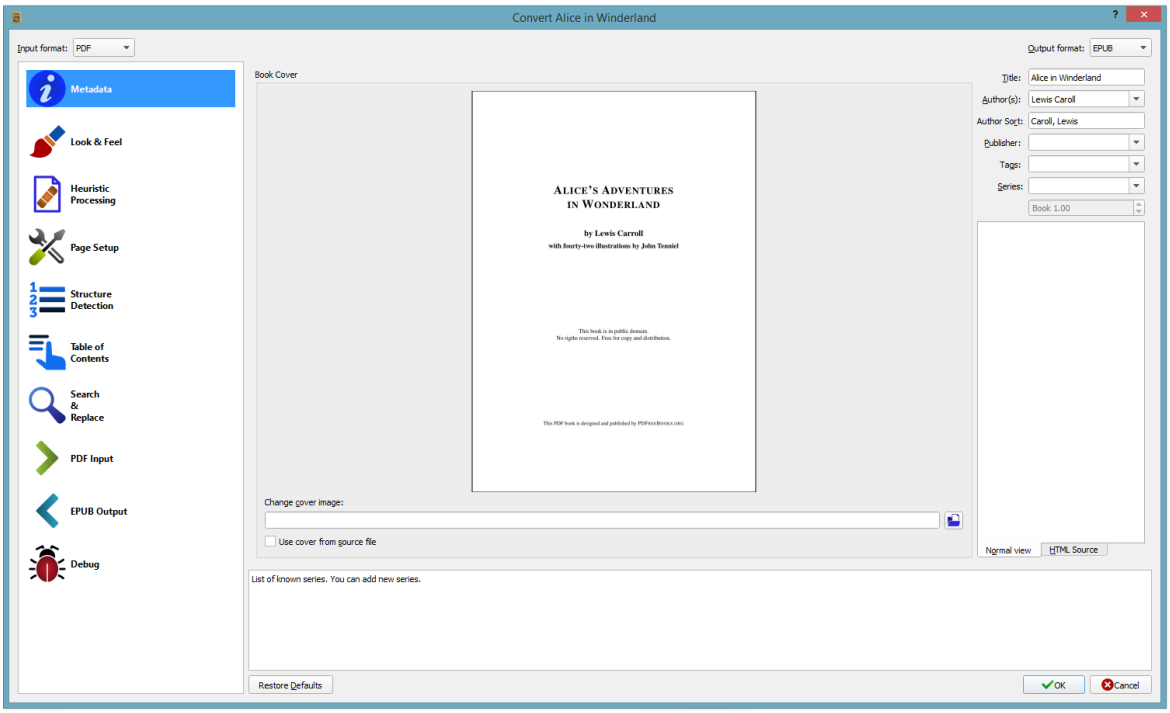
7.1 EDRK kasutatud rakendus „pdftoepub“

Eesti Digiraamatute Keskuses on kasutatud siiani tööriista „pdftoepub“, mis on loodud spetsiaalselt sellega töötava isiku eelistusi silmas pidades. Tööpõhimõtteks on analüüsida tähemärgi suurust ning asukohta, tehes selle pealt üldistusi. Rakendus kuvab kasutajale osa sisendfailist koos igale reale vastavate fondi suuruste ning muude parameetritega. Sellele vastavalt peab kasutaja määrama, kas tegemist on pealkirja või tava tekstiga ning märkima vastava suuruse tabelis vastavasse lahtrisse. Rakendus suudab ära tunda ka lehealuseid viiteid, kuid nende linkimine tuleb siiski teha kasutajal käsitsi, muutes selleks peatüki HTML koodi. Elementide tüübi määramine käib punkti suuruse järgi. Kasutajal tuleb õigesse lahtisse sisestada punkti suurus, mille peale eraldatakse vastavad elemendid. Väljundfaili puhul on võimalik seadistada vormistuslikke parameetreid nagu näiteks lehe äärise suurust.

Asjatundliku kasutaja käes võib rakendust pidada kindlasti küllaltki efektiivseks, kuid sellega müügikõlbliku raamatu loomine vajab siiski parajal hulgal teadmisi nii rakenduse enda, kui ka EPUB standardis kasutusel olevate tehnoloogiate nagu näiteks HTML ja CSS kohta. Vajaminevate eelteadmiste hulk on antud töö käigus loodud rakenduse puhul väiksem ning sellega seoses on suurem ka kasutusmugavus. Lõpptulemusi rakenduste vahel võrrelda ei saanud, sest autorile kättesaadavad failid on järeltoimetatud ning ei anna seetõttu objektiivset tulemust.

7.2 Calibre

Calibre on vabavaraline tarkvara, millega on võimalik täisautomaatselt luua PDF dokumendist EPUB raamatuid. Kasutusmugavus on rakendusel väga hea, võimalik on määrata raamatu kohta lisaparameetreid, et parandada tuvastuse täpsust ning lisaks on võimalik sisse lülitada ka heuristilise analüüsi süsteem, mis parandab peatükkide ning teiste elementide äratundmist sisenddokumendist. Kasutusmugavus on Calibre’l käesoleva töö käigus loodud rakendusest parem ning vajaminev eelteadmiste hulk väiksem. Lõpptulemus seevastu on märgatavalt kehvem.



Joonis 3. Kuvatõmmis Calibre'ga PDF konverteerimisest

Joonisel 3 on kuvatõmmis rakenduse Calibre osast, millega on võimalik konverteerida PDF dokumente erinevatesse e-raamatu formaatidesse. Vasakul ribal on seadistusvõimalused, millega saab määrata sisendraamatu parameetreid ning kujundada väljundina loodavat e-raamatut. Lisaks on võimalik valida parameeter „Heuristic Proccessing", mis rakendab sisend raamatule heuristilisi algoritme, et parandada loodava e-raamatu kvaliteeti. Rakenduse katsetamisel ilmnes väljundfailides tõsiseid puudujääke, näiteks oli kõikidel juhtudel raamatusse leheküljenumbrid sisse jäänud ning mitte ühelgi korral ei loodud korrektset sisukorda. Lisaks ilmnes probleeme piltide paigutuse ning suurusega. Ühel juhul esines väljundraamatus ka valiidsuse probleeme. Kokku katsetati rakendust kümne raamatuga.

Antud rakenduse loodud EPUB raamatute kvaliteet on tavakasutajale piisav, kuid sellest ei piisa loodud EPUB formaadis raamatu müümiseks.

7.3 Online konverterid

PDF formaadist EPUB formaati konverteerimiseks ei ole tingimata tarvis eraldiseisvat programmi, vaid piisab ka veebis vabalt kättesaadavatest rakendustest. Rakendustega nagu näiteks ToEpub on võimalik konverteerida mõne nupuvajutusega raamat PDF formaadist EPUB formaati [23]. Saadavalolevaid veebi rakendusi on mitmeid, kuid võrdlusesse valiti ToEpub just oma kasutusmugavuse ning väljundi kvaliteedi tõttu. Rakendust on äärmiselt

mugav kasutada, konverteerimiseks tuleb vaid oma PDF dokument lohistada vastavasse kasti, misjärel see üles laetakse ning mõne sekundiga e-raamatuks konverteeritakse. Lisaks on võimalik üles laadida ka mitu raamatut ja luua nendest järjekord, mille alusel neid järjest konverteeritakse.

Kümne testitud PDF dokumendi korral saadi kõikidel juhtudel valiidne väljund-EPUB fail, kuid kvaliteet oli kõikidel juhtudel madal. Tekstist oli kadunud algne struktuur ning puudusid eristataval kujul pealkirjad. Väljundfailist raamatu lugemine oli raskendatud struktuuri puudujääkide ning puuduvate elementide tõttu.

Kokkuvõte

Müügikõlbliku e-raamatu loomine PDF dokumendist on siiani olnud suures osas käsitöö ning eeldanud teadmisi sellega seonduvatest tehnoloogiatest. PDF dokumendi loomisel kaotatakse suur osa strukturaalsest informatsioonist, mis tuleb EPUB formaadis raamatu loomiseks taastada.

Algse struktuuri taastamiseks on antud bakalaureuse töö raames loodud töövoog, mille käigus konverteeritakse PDF formaadis raamat esmalt HTML kujule. EPUB formaadis hoitakse raamatu sisu peatükkidena XHTML kujul, mistõttu on esmane HTML-iks konverteerimine hea alguspunkt edasisele töötlusele. Kasutajast lähtuva tulemuse saavutamiseks töödeldakse loodud HTML dokumenti erinevate teksti elementide eraldamiseks mõeldud tööriistadega, millega on kasutajal võimalik määrata dokumendi ülesehitust ning kujundust.

Antud töö tulemusena on loodud graafilise kasutajaliidesega rakendus, millega on võimalik konverteerida kuluefektiivsemalt PDF dokumente EPUB 2.0.1 formaati. Rakenduse arendamisel on eelkõige lähtutud vajadusest maksimaalselt säilitada algset kujundust ning struktuuri. Loodud rakendus ei eelda kasutajalt varasemaid teadmisi EPUB-ist ega sellega seonduvatest tehnoloogiatest. Kasutaja ülesandeks on vaid visuaalselt ära tunda raamatusse kuuluvad elemendid ning need graafilise kasutajaliidese abil vastavalt tüübile ära märkida. Seeläbi taastatakse dokumendi algne struktuur. Eelmainitud sammude tulemusena luuake müügikõlblik EPUB 2.0.1 formaadis e-raamat.

8. Viited

[1] Adobe ametlik PDF dokumentatsioon. (http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf) (25.01.2017)

[2] IDR Solutions ametlik blogi PDF kohta, teksti kodeerimine. (https://blog.idrsolutions.com/2017/01/the-2017-guide-to-pdf-files-text-in-pdfs/) (25.01.2017)

[3] IDR Solutions ametlik blogi PDF kohta, teksti kodeerimine. (https://blog.idrsolutions.com/2017/01/the-2017-guide-to-pdf-files-text-in-pdfs-part-2/) (25.01.2017)

[4] IDR Solutions ametlik blogi PDF kohta, fontide kodeerimine. (https://blog.idrsolutions.com/2009/05/introductory-pdf-font-tutorial/) (25.01.2017)

[5] IDR Solutions ametlik blogi PDF kohta, fontide kodeerimine. (https://blog.idrsolutions.com/2009/05/pdf-font-technologies/) (25.01.2017)

[6] IDR Solutions ametlik blogi PDF kohta, piltide kodeerimine. (https://blog.idrsolutions.com/2010/09/understanding-the-pdf-file-format-images/) (25.01.2017)

[7] IDPF EPUB standard. (http://idpf.org/epub) (25.01.2017)

[8] IDPF EPUB 3, punkt 2.6. (http://www.idpf.org/epub/301/spec/epub-overview.html) (25.01.2017)

[9] IDPF EPUB 2.0.1 ametlik spetsifikatsioon, punkt 1.0. (http://www.idpf.org/epub/20/spec/OPS_2.0.1_draft.htm) (25.01.2017)

[10] IDPF EPUB 2.0.1 (http://idpf.org/epub/201) (25.01.2017)

[11] IDPF EPUB 3.0.1 spetsifikatsioon. (http://www.idpf.org/epub/301/spec/epub-publications.html#sec-package-metadata-fxl) (25.01.2017)

[12] IDPF Fixed-Layout EPUB 3 spetsifikatsioon. (http://www.idpf.org/epub/fxl/) (25.01.2017)

[13] S. Marinai, E. Marino and G. Soda. Conversion of PDF Books in ePub Format. *2011 International Conference on Document Analysis and Recognition*. Beijing, 2011, lk 478-482.

[14] Constantin A., Pettifer S., Voronkov A. PDFX: Fully-automated PDF-to-XML Conversion of Scientific Literature. *Proceedings of the 2013 ACM symposium on Document engineering.* Florence, Italy, 2013, lk 177-180.

[15] PDFX. (http://pdfx.cs.man.ac.uk/) (25.04.2017)

[16] Fang J., Tang Z., Gao L. Reflowing-driven paragraph recognition for electronic books in PDF. *Document Recognition and Retrieval XVIII.* 2011.

[17] J. L. Meunier. Optimized XY-cut for determining a page reading order. *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. 2005, nr 1, lk 347-351.

[18] pdf2htmlEX github. (https://github.com/coolwanglu/pdf2htmlEX) (30.01.2017)

[19] Wang L., Liu W. Online publishing via pdf2htmlEX. 2013. (http://coolwanglu.github.io/pdf2htmlEX/doc/tb108wang.pdf) (25.04.2017)

[20] html2canvas. (https://html2canvas.hertzen.com/) (06.04.2017)

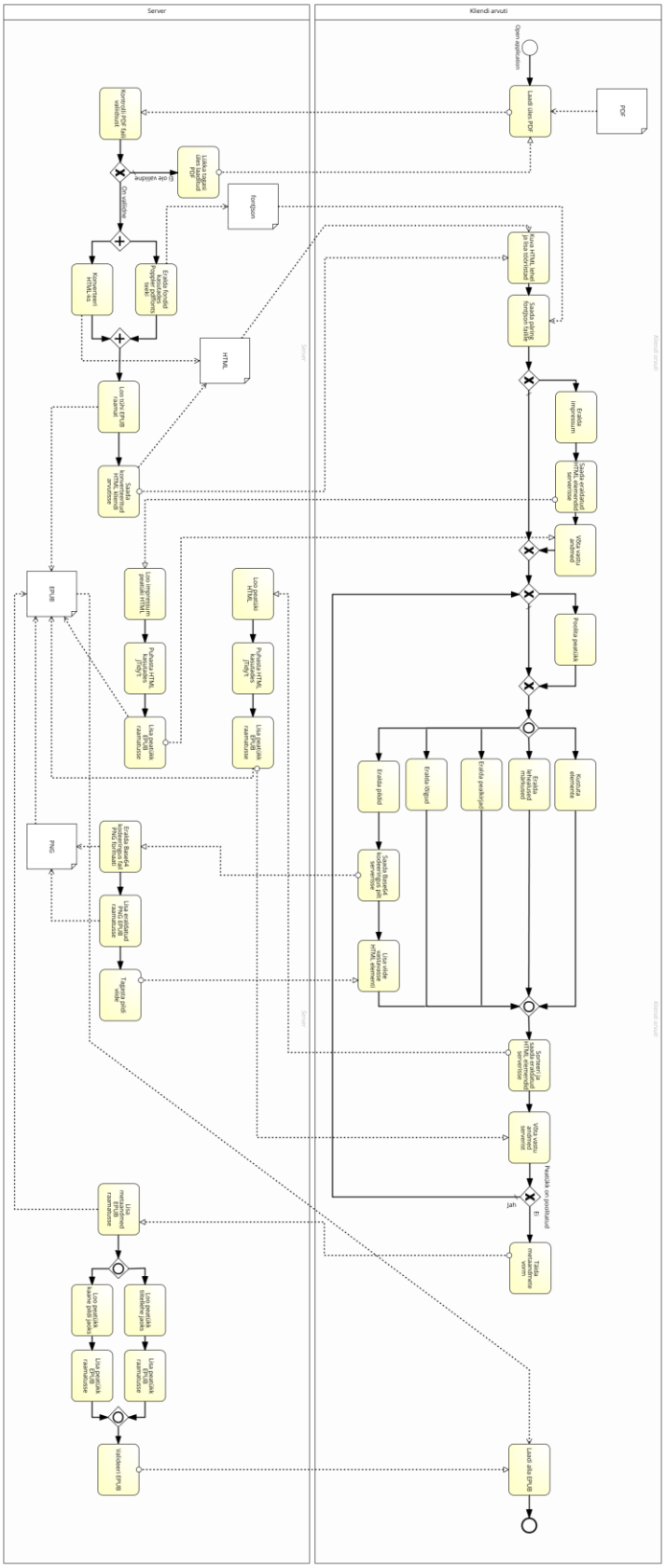
[21] Poppler pdffonts. (https://github.com/davidben/poppler/blob/master/utils/pdffonts.cc) (10.04.2017)

[22] Epublib. (http://www.siegmann.nl/epublib) (06.04.2017)

[23] Rakendus ToEpub (http://toepub.com/) (30.04.2017)

Lisad

I. BPMN mudel rakendusest



II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Sander-Sebastian Värv**,

(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   PDF dokumendi konverteerimine EPUB formaati,
   (*lõputöö pealkiri*)

   mille juhendaja on Vesal Vojdani,

   (*juhendaja nimi*)

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **11.05.2017**