TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut

ITI40LT

Mart Laus 123875IAPB

# Efektiivse OAI-PMH standardil töötava metaandmete kogumise kliendi loomine

Bakalaureusetöö

Juhendaja: Marko Kääramees
PhD
Dotsent

Tallinn 2016

# Autorideklaratsioon

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

Autor: Mart Laus

19.05.2016

# Annotatsioon

Töö eesmärk on uurida OAI-PMH standardit ning luua kiire, töökindel ja efektiivne parser ning liides OAI-PMH protokollil andmevahetuse jaoks, tehes seda nii, et korduv andmete küsimine on kuluefektiivne ning komponendid on korduvkasutatavad.

Töös uuritakse, kuidas on kõige efektiivsem kasutada OAI-PMH protokolli poolt pakutavaid päringuid, et vahetada suuri andmekoguseid ning hoida neid uuendatuna. Töös uuritakse ka, kuidas on kõige otstarbekam parsida andmeid, mida üle OAI-PMH protokolli saadetakse.

Töös käsitletakse probleeme, mis tekivad valede päringute kasutamisega suurtel andmekogustel ning valede parsimismeetodite kasutamisega XML vormingus metaandemete töötlemisel. Uurimistöös antakse soovitusi, kuidas protokolli kõige mõistlikumalt kasutada ning kuidas lugeda ning töödelda sünkroniseeritavat infot.

Uurimuse jaoks luuakse Java rakendus kasutades tehnoloogiaid nagu Hibernate, Guice, MySql.

Töö tulemuseks on õigesti disainitud liides andmete sünkroniseerimiseks ja uuendamiseks ning efektiivne parser, mis töötleb andmed võimalikult efektiivselt ning kiirelt kliendi andmebaasi.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 31 peatükki, 10 joonist, 3 tabelit.

# Abstract

## The creation of an effective OAI-PMH protocol metadata harvester

The purpose of this thesis is to analyse the OAI-PMH standard and create an efficient, reliable and fast metadata harvester and parser. Also the resulting client must be cost-effective and the components reusable wherever possible.

The thesis will study and investigate the OAI-PMH standard and decide which is the most effective way to use the protocols various requests, when harvesting and updating big datasets from repositories and data marts. The thesis will also analyse how to efficiently parse the metadata, that is synchronized over the OAI-PMH protocol.

The study will focus on problems that arise, when using inefficient query and parsing logic, when harvesting big amounts of XML formatted metadata. As a result, this thesis will outline the solution for an effective implementation of the OAI-PMH protocol and a way to parse the metadata.

For the purposes of the study a Java application will be created, using technologies including but not limited to Hibernate, Guice and MySql.

The result of this bachelor thesis is a correctly designed application that harvests, parses and updates the metadata over the OAI-PMH protocol to the applications database in an efficient, reliable and swift way.

The thesis is in Estonian and contains 34 pages of text, 31 chapters, 10 figures, 3 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| OAI-PMH | Protocol for Metadata Harvesting, protokoll metaandmete kogumiseks |
| *LOM* | Learning Object Metadata, õpiobjekti andmemudeli standard |
| Dublin Core | Veebiressursside metaandmete kirjeldamise standard |
| EstCore | Eesti digiõppevara metaandmete standard |
| HTTP | Hypertext Transfer Protocol, hüperteksti edastusprotokoll |
| SOAP | Simple Object Access Protocol, lihtne objektipöördusprotokoll |
| DOM | Document Object Model, dokumendi objektimudel |
| XML | Extensible Markup Lanuage, laiendav märgistuskeel |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Informatsiooni tõhus vahetamine tänases globaliseeruvas maailmas vajab efektiivseid ning hoolikalt läbimõeldud lahendusi kasutades protokolle, mis on saanud juba standarditeks. Seega on ülimalt tähtis, et standardit implementeeriv lahendus oleks võimalikult kiire, skaleeruv ning kuluefektiivne.

Antud töös uuritakse õpiobjektide nagu näiteks õpikute ja raamatute metaandmete sünkroniseerimist ning uuendamist repositooriumi ning kliendi vahel üle OAI-PMH protokolli.

Käesoleva töö teema valik tuleneb autori tööalasest kokkupuutest Net Groupis arendatud e-Koolikotiga, mis kasutab õppematerjalide hankimiseks erinevate kirjastajate ning õppematerjalide tootjate repositooriume kasutades sünkroniseerimiseks just OAI-PMH protokolli. Net Groupis arendajana töötades oli autoril võimalus arendada e-Koolikotile OAI-PMH liidest ning sellest tulenevalt puutuda kokku probleemidega, mis tekkisid antud protokolli kasutades.

Käesoleva töö eesmärgiks on uurida OAI-PMH protokolli ning võrrelda standardi kasutusviise ning selles ülekantava info parsimisviise, et selgitada välja kõige kiirem, skaleeruvam ning töökindlam lahendus.

## 1.1 Taust ja probleem

Euroopa haridusmaastikul on käimas hetkel digirevolutsioon, mille tulemuseks peaks valmima ühtne elektrooniline turg digitaalsetele õppematerjalidele. Selle turu üheks alustalaks on protokoll, millega seda infot vahetatakse erinevate riikide, haridusasutuste ning sisupakkujate vahel. Ühtset protokolli on vaja selleks, et iga uue ühenduse loomiseks ei peaks arendama uut liidest, mis raiskaks aega ning raha. Ühtlasi garanteerib ka üldtunnustatud protokoll kvaliteedi ning kiiruse, mida riiklikult tähtsad süsteemid peavad evima. Seega on ülimalt tähtis, et protokolle ning süsteeme kasutatakse arhitektuuriliselt korrektselt ning kuluefektiivselt. Enamasti on ka tarkvara

kvaliteedi mittefunktsionaalsetes nõuetes tingimus, et tarkvara reaktsiooniaeg peab olema piisavalt lühike [1] . Enamasti tagab kiirust ja stabiilsust just läbimõeldud arhitektuur. Selleks, et järgnevad arendajad saaks luua kvaliteetseid ning eelnevale analüüsile tuginevaid rakendusi on vaja teha näidisimplementatsioone ning *proof of concept*'e nagu antud töö praktiline osa.

Tarkvaraarenduse peamine osa tehti aastatel 2015-2016 ettevõtte Net Group jaoks e-Koolikoti projekti raames. Süsteem võeti kasutusele 2016 alguses ning aja jooksul lisandub süsteemi üha rohkem sisupakkujaid üle uuritava protokolli.

## 1.2 Ülesande püstitus

Eesmärgid, milleni töö käigus jõutakse:

- Tehakse analüüs OAI-PMH protokolli õigest kasutamisest.
- Võrreldakse erinevaid OAI-PMH protokolli kasutusviise ning andmete parsimise viise.
- Luuakse efektiivne parser ning klient üle OAI-PMH standardi andmete lugemiseks

## 1.3 Metoodika

OAI-PMH protokollil on mitmeid erinevaid meetodeid andmete sünkroniseerimiseks, seetõttu analüüsitakse kõiki võimalikke kasutusviise. Töö läbiviimiseks kasutatakse võrdleva analüüsi meetodit, mille käigus leitakse parim lahendus. Seejärel uuritakse võimalikke riske, mida üks või teine kasutusviis võib kaasa tuua erialakirjanduse abil ning valitakse välja käesoleva töö jaoks sobiv lahendus.

Kui sobiv protokolli implementatsioon on välja valitud, siis analüüsitakse peamiseid viise, kuidas XML vormingus sünkroniseeritavat infot kõige kiiremalt, skaleeruvamalt ning turvalisemalt parsida.

Materjalide sünkronisatsiooni ning uuendamise kestvust mõõdetakse ajaliselt. Üldisel tasemel võrreldakse ka koodi arhitektuuri ja ülesehitust erinevate lähenemiste puhul.

## 1.4 Ülevaade tööst

Esimeses peatükis tehakse sissejuhatus OAI-PMH protokolli. Antakse ülevaade, kus, kuidas ja milleks antud standardit täna kasutatakse.

Teises peatükis tehakse ülevaade formaadist, milles infot üle kantakse, ehk LOM standardist ning LOMil põhinevast EstCorest.

Kolmandas peatükis analüüsitakse OAI-PMH protokolli kasutusviise ning ülekantava info töötlemise viise. Analüüsi põhjal tuuakse välja erinevate lahenduste positiivsed ja negatiivsed küljed. Seejärel valitakse välja optimaalseim viis, mida töö praktilises osas kasutama hakatakse.

Neljandas peatükis analüüsitakse, kuidas luua valitud tehnoloogiatel arhitektuuriliselt kõige loogilisemat ning jätkusuutlikumat lahendust. Pärast seda kirjeldatakse, kuidas loodud arhitektuuril süsteem täpselt töötama hakkab ning millised kasud valitud arhitektuurist tulenevad.

# 2 OAI-PMH protokoll

OAI-PMH protokoll on Open Archives Initiative loodud standard, mis pakub infosüsteemist sõltumatut koostalitusvõimelist standardit metaandmete jagamiseks ning põimamiseks ehk info kogumiseks [2] . See tähendab, et antud raamistikus on kaks osapoolt, esiteks andmete jagajad (*Data Providers*). Teiseks teenusepakkujad (*Service Providers*), kes kasutavad üle OAI-PMH kogutud infot, et pakkuda lisandväärtusega teenust.

Protokolli hakati looma 1999. aastal, kui Paul Ginsparg, Rick Luce, ja Herbert Van de Sompel saatsid välja kutse piiratud hulgale tehnilisetele ekspertidele, et luua masinloetav standard metaandmete jagamiseks digitaalsetele repositooriumitele [3] . Järgnevatel aastatel peeti mitmeid töötubasid ning avastati, et antud lahendusest on huvitatud ka raamatukogud, muuseumid ja kirjastajad. Selleks loodi Open Archives Initiative, mis hakkas tegelema standardi loomise ning edaspidise haldamisega. 2002. aastal lasti välja versioon 2.0, mis polnud tagasiühenduv 1.x arhitektuuriga ning versioon 2.0 on kasutusel tänapäevani ühendades üha rohkem repositooriume ning teenuspakkujaid iga aastaga.

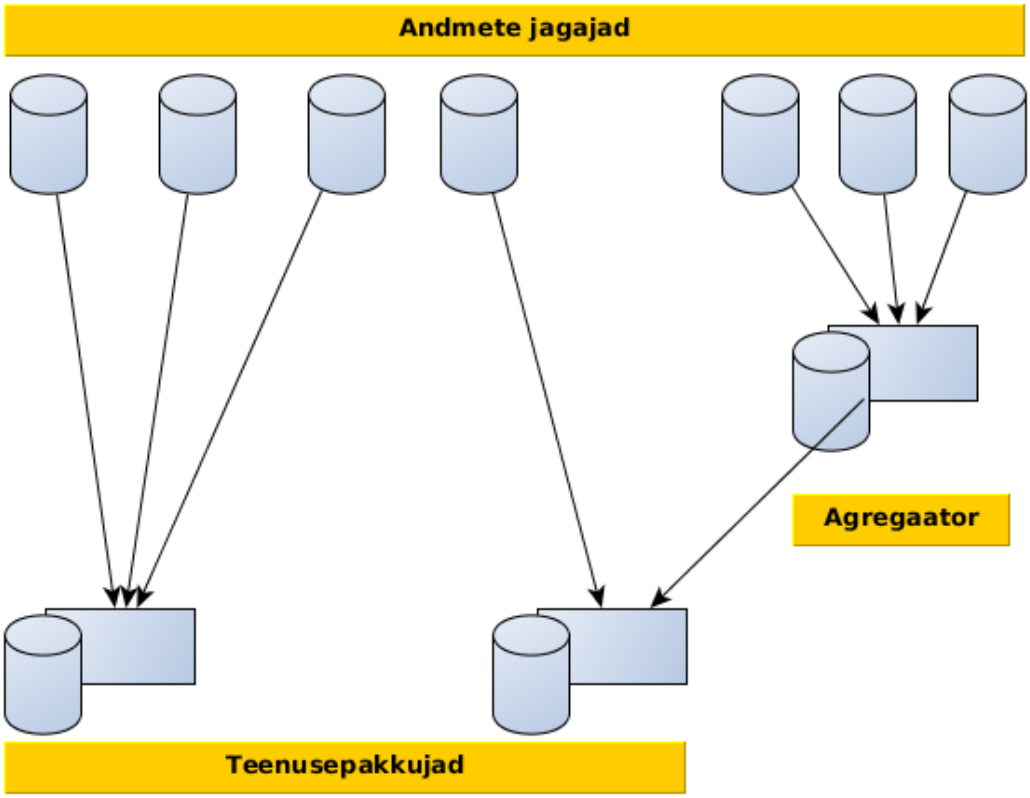
## 2.1 Andmete jagajad

Andmete jagajad ehk repositooriumid teevad üle OAI-PMH protokolli avalikuks metaandmed oma repositooriumis sisalduva kirjanduse kohta. See tähendab, et materjalide täistekste antud protokolliga üle ei kanta, kantakse ainult viide täistekstile. Andmete jagajad võivad olla nii era- kui ka avalikust sektorist. Avaliku sektori osapoolteks võivad olla näiteks raamatukogud või õppeinfosüsteemid. Erasektoris on andmete jagajateks kirjastused või kirjastuste ühendused.

Andmete jagajad võivad töötada ka agregaatoritena ühendades erinevaid infoallikaid üle nii OAI-PMH protokolli kui ka muude mittestandardsete protokollide (Joonis 1).

Agregaatorid jagavad kogutud infot välja üle OAI-PMH protokolli, seeläbi suurendades info hulka, mida on võimalik teenusepakkujatel koguda.

Tänaseks on ametlikult registreeritud 2990 repositooriumi [4] , kuid kuna registreerimine pole kohustuslik, siis see number ei väljenda tegelikku andmepakkujate hulka. Näiteks Eesti repositooriumitest on registreeritud vaid Tartu Ülikooli teenused, kuid registreerimata on näiteks antud töös kasutatav Waramu repositoorium [5] , mis serveerib portaali Koolielu materjale.




Joonis 1. OAI-PMH protokolli osapoolte diagramm

## 2.2 Teenusepakkujad

Teenusepakkujad on infosüsteemid, mis koondavad üle OAI-PMH protokolli materjalide metaandmed ja seeläbi pakuvad antud infosüsteemis kasutajatele lisaväärtust, näiteks otsimine või erinevate materjalide kombineerimine.

Eestis on teenusepakkujaks näiteks e-Koolikott, mis koondab andmed erinevates repositooriumites ehk digitaalse õppevara kogudes olemasoleva õppevara kohta [6] . Antud töö eesmärk ongi leida parim OAI-PMH liidese lahendus just teenusepakkujatele.

## 2.3 Saadetav metainfo

Metainfo, mida üle OAI-PMH protokolli saadetakse on XML vormingus. Kõikidele OAI-PMH päringute vastuse juurelemendi nimi on OAI-PMH, kus on kolm atribuuti, mis kirjeldavad vastuse nimeruumi ning skeemi vastuse valideerimiseks (Joonis 2). Päringu vastuse esimesed kaks elementi on alati kuupäev ning päringu info, kolmanda elemendi nimi on sooritatud päringu nimi. Seega kui tehakse GetRecord päring, siis on kolmanda elemendi nimi GetRecord, kui tehakse ListIdentifiers päring, siis kolmas element on ListIdentifiers.

*Record* objekt on ühe konkreetse täisteksti metaandmete kogum ehk näiteks ühe raamatu või õpiku *record* objekt sisaldaks kogu metaainfot antud täisteksti kohta. *Record* objekt on nii GetRecord kui ListRecords päringute vastustes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
         http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
      <responseDate>2002-05-01T19:20:30Z</responseDate>
      <request verb="GetRecord" identifier="oai:arXiv.org:hep-th/9901001"
          metadataPrefix="oai_dc">http://an.oa.org/OAI-script</request>
      <GetRecord>
            <record>
                  ...
            </record>
      </GetRecord>
</OAI-PMH>
```

Joonis 2. OAI-PMH GetRecord päringu näidisvastus [8]

## 2.4 Saadetava metainfo formaat

OAI-PMH protokoll võimaldab repositooriumitel pakkuda oma infot mitmetes erinevates vormingutes. Selle saavutamiseks on andmete küsimisel võimalik täpsustada metainfo formaati. See tähendab seda, et OAI-PMH ise ei defineeri täpset metaandemete formaati, mis käib *record* objekti sisse, kuid kõik kasutatavad vormingud peaksid evima endas minimaalselt Dublin Cores [12] sisalduvat infot.

Dublin Core on minimaalne vajalik infopakett, et kirjeldada andmeobjekti – näiteks nagu pealkiri, autor ning viide reaalsele materjalile. Antud uurimistöös kasutatakse palju Eesti vajaduste jaoks välja töötatud EstCore vormingus XMLi. EstCore standard on palju detailsem ning kirjeldavam kui Dublin Core. EstCore baseerub IEEE LOM standardil [13] , millest on paljudel riikidel on modifitseeritud versioon, mis sobib just antud riigi vajadustega. Näiteks on oma versioon IEEE LOMist nii Norras, Rootsis kui ka Taiwanis.

# 3 Lahenduse meetodite analüüs

Antud peatükis analüüsitakse erinevaid OAI-PMH protokolli päringuid ning levinumaid XMLi töötlemise viise. Protokolli päringutest vaadeldakse ainult päringuid, mida saab kasutada andmete täielikuks sünkronisatsiooniks ning pidevaks uuendamiseks. Protokollis on veel päringuid, mis annavad informatsiooni repositooriumi enda kohta, kuid need ei ole seotud metaandmete üle kandmisega repositooriumist teenusepakkuja süsteemi.

## 3.1 Nõuded

Andmete sünkroniseerimise liidesele saab esitada nii funktsionaalseid kui mittefunktsionaalseid nõudeid. Antud liidese kontekstis esitas töö autor järgnevad nõuded:

- Liides peab olema piisavalt kiire, et sünkroniseerida ning uuendada andmeid Eesti repositooriumitest mõistliku ajaga
- Liidese disainist tulenevalt ei tohi tekkida olukordi, kus suure dokumendi lugemisel saab muutmälu täis
- Liidese arhitektuur peab olema taaskasutatav nii uuendamisel kui esmakordsel sünkroniseerimisel.

## 3.2 OAI-PMH päringute analüüs

### 3.2.1 ListRecords

OAI-PMH standardi dokumentatsioon kirjeldab ListRecords päringut kui meetodit, mida peaks kasutatama andmete ülekandmiseks (ingl. *harvest*) repositooriumist teenusepakkuja süsteemi [7] .

ListRecords XML objekt sisaldab üldjuhul mitut erinevat *record* objekti ühes vastuses *ListRecords* päringule. *Record* objektide arv päringu vastuses on iga repositooriumi enda defineerida, kuid *record* objekti ennast ei poolitata. Selleks, et saata suurt hulka *record* objekte, võimaldab ListRecords kontrollida saadetavate metaandmete voolu *resumptionTokeniga* (Joonis 3).

*ResumptionToken* on põhimõtteliselt kursor, mis ütleb, kust andmete ülekandmist jätkata. Kui repositooriumis on näiteks tuhat kirjet ja ühe vastuse suuruseks on määratud sada kirjet, siis pärast esimese saja saatmist ütleb kaasa antav *resumptionToken*, *et* järgmise *ListRecords* päringuga tuleb andmete saatmist jätkata alates saja esimesest *record* objektist.

```
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/>
      <responseDate>2016-04-17T12:47:32Z</responseDate>
      <request verb="ListRecords" metadataPrefix="oai_lom">
            ...
      </request>
      <ListRecords>
            <record>...</record>
            <record>...</record>
            <resumptionToken>7b5c0ec148316b9</resumptionToken>
      </ListRecords>
</OAI-PMH>
```

Joonis 3. OAI-PMH ListRecords päringu näidisvastus

### 3.2.2 ListIdentifiers

ListIdentifiers on ListRecords päringu vähendatud versioon, kust on välja võetud täielik *record* objekt ning tagastatakse ainult *record* objektis sisalduv *header* objekt [9] . Header objekti sees on objekti nagu näiteks raamatu unikaalne identifikaator antud repositooriumis ning tema viimase uuendamise kuupäev (Joonis 4). See tähendab, et ListIdentifiers päring tagastab kordi vähem informatsiooni kui ListRecords. Antud omadus on tähtis info uuendamise kontekstis, kuna kui on vaja kontrollida ainult seda, et kas objekt on uuenenud, siis ListIdentifiers päringuga ei laeta alla kogu objekti nagu

näiteks teeb seda ListRecords. ListIdentifiers päringul on võimalus kontrollida info voolu *resumptionToken* objektiga nagu ListRecords päringu puhul.

```
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/">
      <responseDate>2016-04-17T13:46:23Z</responseDate>
      <request verb="ListIdentifiers" metadataPrefix="oai_lom">
            ...
      </request>
      <ListIdentifiers>
            <header>
                  <identifier>8dc992d0ca8180207c</identifier>
                  <datestamp>1999-12-31T22:00:00Z</datestamp>
            </header>
            <header>
                  <identifier>c3f67f9a3aa7aab89d</identifier>
                  <datestamp>1999-12-31T22:00:00Z</datestamp>
            </header>
            <resumptionToken>1efa17adaf657cb7c168</resumptionToken>
      </ListIdentifiers>
</OAI-PMH>
```

Joonis 4. OAI-PMH ListIdentifiers päringu näidisvastus

### 3.2.3 GetRecord

GetRecord päringut kasutatakse üksiku *record* objekti saamiseks tema unikaalse identifikaatori järgi (Joonis 5). Seega selleks, et teha GetRecord päringut peab info pärija teadma oodatava objekti identifikaatorit [10] . Identifikaatorid võivad olla olemas juba teenusepakkuja andmebaasis varasematest päringutest, kuid identifikaatorite saamiseks võib ka kasutada ListIdentifiers päringut.

```
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/ ">
      <responseDate>2016-04-17T14:11:51Z</responseDate>
      <request verb="GetRecord" identifier="a0d8ee"
metadataPrefix="oai_lom">
            ...
      </request>
      <GetRecord>
            <record>
                  <header>
                        <identifier>a0d8ee</identifier>
                        <datestamp>1999-12-31T22:00:00Z</datestamp>
                  </header>
                  <metadata>
                        ...
                  </metadata>
            </record>
      </GetRecord>
</OAI-PMH>
```

Joonis 5. OAI-PMH ListIdentifiers päringu näidisvastus

### 3.2.4 Päringute kiiruse võrdlus

Antud peatükis viiakse läbi mõõtmised kõigi kolme päringu kohta. Leitakse ListRecords päringu keskmine kiirus viiekümne õpiobjekti kohta. Arvutatakse ka välja ListIdentifiers ja GetRecord päringute ajaline kulu viiekümne materjali kohta. Päringute kiiruse mõõtmiseks kasutati Eesti avalikku Waramu repositooriumi. Mõõdeti ainult serveri vastuse aega antud päringule, kõikvõimalike muude failide laadimisaega, mida repositooriumi veebileht võib jagada ei mõõdetud.

Tabel 1. Päringute kiiruse võrdlus

| **Meetod / Hulk** | **1. Katse (s)** | **2. Katse (s)** | **3. Katse (s)** | **4. Katse (s)** | **5. Katse (s)** | **Keskmine (s)** |
|---|---|---|---|---|---|---|
| ListRecords / 50 | 2.78 | 2.62 | 2.55 | 2.79 | 2.70 | 2.6880 |
| ListIdentifiers / 50 | 0.479 | 0.463 | 0.477 | 0.466 | 0.449 | 0.4668 |
| GetRecord / 1 | 0.047 | 0.044 | 0.043 | 0.041 | 0.044 | 0.0438 |

GetRecord päringu keskmine päringuaeg on 0.0438 sekundit ning tehes seda 50 korda on viiekümne materjali pärimise koguaeg 2.19 sekundit. Sellele juurde liites ListIdentifiers päringu keskmise vastuseaja 0.4668, saame kahe meetodi ajaliseks kogukuluks 2.6568 sekundit. See teeb kahe lahenduse ehk kombinatsioon ListIdentifiers ja GetRecords päringutest ning GetRecords päringu ajaliseks vaheks 0.0312 sekundit, kui iga materjali küsimine ning töötlemine toimub jadamisi.

Seega võib järeldada, et ListRecords päring ning kombinatsioon GetRecord ja ListIdentifiers päringutest on praktiliselt sama kiired jadapäringute puhul. Väikene erinevus ListRecords kahjuks võib olla nii mõõtmisviga kui ka antud teenusepakkuja implementatsiooni omapära.

## 3.3 Parsimismeetodite analüüs

Antud peatükis vaadeldakse  hetkel vabavaraliselt kasutatavaid populaarsemaid XMLi töötlemise viise. XPathi ning DOMi parsimise lahendusi on võimalik programmaatiliselt implementeerida paljudel erinevatel viisidel, kuid antud peatükis võrreldakse nende sooritusvõimet antud rakenduses töödeldava dokumendi struktuuri kontekstis.

### 3.3.1 DOMi parsimine

DOM ehk Document Object Model on viis, kuidas kirjeldada struktureeritud infot, mis antud uurimistöö konteksits on XML dokument. DOM defineerib iga dokumendi elemendi infoobjekti,tema atribuudid ning meetodid, kuidas neist infot kätte saada [15] . DOMi parsimiseks nimetatakse antud meetodite kasutamisel info pärimist.

DOMi parsimisel on kolm moodust, kuidas elementide väärtusi kätte saada. Kõige tavalisem ning enimkasutatavaim viis on meetodi *getElementsByTagName()* kasutamine, mis käib läbi kogu dokumendi ning otsib soovitud elemendi või elemendid tema märgendi nime järgi. Teine võimalus on ise manuaalselt kõik dokumendi elemendid läbi käia ning otsida seeläbi soovitud infot. Kolmas viis on minna elemendist elementi kasutades nende vahelisi seoseid.

### 3.3.2 XPathiga parsimine

XPath ehk XML Path Language on keel, mis võimaldab leida XML dokumendist päringu järgi soovitud elementi [14] . XML dokument põhineb puu struktuuril, kus on puu võra ning lehed. XPath seega pakubki võimalust kirjeldada teekonda puu juureelemendist soovitud leheni.

See tähendab seda, et elemendi või elementide leidmiseks kirjeldatakse ära teekond, kuidas asukohani täpselt jõuda. See väldib olukorda, kus peaks iga päringu sooritamisel käima läbi kogu dokumendi nagu näiteks DOMi parsimisel *getElementsByTagName* meetodit kasutades.

```
<record>
	<header>
		<identifier>a0d8ee</identifier>
		<datestamp>1999-12-31T22:00:00Z</datestamp>
	</header>
	<metadata>
		<title>Pealkiri</title>
	</metadata>
</record>
```

Joonis 6. Näidis XML dokument

Seega selleks, et näites (Joonis 6) olevast XML-ist *title* välja väärtus oleks kirjeldada tema teekond juurelemendist temani, ehk kui juurelement on *record*, siis oleks teekond /record/metadata/title ning kogu ülejäänud XMLi läbi ei vaadata. See tähendab, et *header* elemendi ja ka kogu allpool olev sisu jääb töötlemata, kuna seda pole vaja ning programm on selle võrra kiirem.

### 3.3.3 Parsimise analüüs

Antud peatükis võrreldakse kahe enimlevinud XML dokumendi töötlemise mooduse kiirust. DOMi parsimismeetodil kasutatakse *getElementsByTagName() (Joonis 8)* meetodit, kuna see on enimkasutatav ning kasutajasõbralikuim. Xpathiga info töötlemisel kasutatakse täpselt kirjeldatud teekondi soovitud elementideni (Joonis 7), vastavalt Xpathi standardile.

Testserverina kasutati avalikku Waramu teenust ning viidi läbi kaks katset - 5000 ning 9000 materjaliga, 9000 oli mõõtmiste läbiviimise hetkel repositooriumi ümardatud kogumaht. Igast dokumendist loeti välja ning salvestati seitse põhilist infoobjekti antud mõõtmiste otstarbeks. Metainfost väljavõetud infoobjektid olid õppematerjali pealkiri, autorid, kirjeldus, identifikaator, keel, märksõnad ning link õppematerjalile. Selle jaoks programmeeriti kaks eraldi komponenti, mis erinesid vaid viisi poolest, kuidas XML dokumendist soovitud elemendid välja loeti. Mõlema parsimismeetodiga viidi läbi viis katset ning arvutati välja keskmised ajad.

```
private List<LanguageString> getTitles(Document doc) {
  List<LanguageString> titles;
  Node node = getNode(doc, "//*[local-name()='estcore']/*[local-
name()='general']/*[local-name()='title']");
  titles = getLanguageStrings(node, languageService);

  return titles;
}
```

Joonis 7. Näidis Xpathiga parsimisest

```
private List<LanguageString> getTitles(Document doc) {
  List<LanguageString> titles;
  NodeList nodeList = doc.getElementsByTagName("title");
  titles = getLanguageStringsFromList(nodeList,
languageService);

  return titles;
}
```

Joonis 8. Näidis DOM parsimisest

Tabel 2. Parsimismeetodite kiiruse võrdlus 5000 materjaliga

| **Meetod** | **1. Katse (s)** | **2. Katse (s)** | **3. Katse (s)** | **4. Katse (s)** | **5. Katse (s)** | **Keskmine (s)** |
|---|---|---|---|---|---|---|
| XPath | 170.791 | 167.232 | 171.502 | 175.267 | 177.192 | 172.3968 |
| DOM | 190.178 | 197.395 | 192.208 | 196.681 | 199.924 | 195.2772 |

Antud katse põhjal on XPathiga andmete töötlemine ümardatult üle 12 protsendi kiirem kui DOM meetodiga. Vahe tuleneb sellest, et XPath on efektiivsem dokumendi töötlemismeetod olukordades, kus töödeldav metaandmete kogum on suur ning teda tuleb korduvalt läbi otsida, et välja võtta soovitud infoobjektid.

Tabel 3. Parsimismeetodite kiiruse võrdlus 9000 materjaliga

| **Meetod** | **1. Katse (s)** | **2. Katse (s)** | **3. Katse (s)** | **4. Katse (s)** | **5. Katse (s)** | **Keskmine (s)** |
|---|---|---|---|---|---|---|
| XPath | 480.537 | 464.883 | 485.772 | 468.272 | 475.331 | 474.959 |
| DOM | 678.211 | 688.103 | 655.662 | 632.881 | 679.617 | 666.8948 |

Mõõtmistest võib järeldada, et XPathiga parsimine oli 9000 materjali ning seitsme infoobjektiga ligi 30 protsenti kiirem kui DOM parsimine.

Esimese ja teise katse põhjal võib väita, et mida rohkem on sünkroniseeritavat infot, seda rohkem kasu annab XPathiga info töötlemine. Kui esimeses katses oli XPath ainult 12 protsenti kiirem, siis teises katses, kus oli 4000 materjali rohkem, oli XPath ligi 30 protsenti kiirem kui DOM parsimine.

## 3.4 Lahenduse valik

### 3.4.1 Liidese meetodite valik

Tulenevalt OAI-PMH standardist on olemas kaks võimalikku viisi, kuidas ehitada liidese töötamise loogikat – kasutada ListRecords päringut või ListIdentifiers ja GetRecord päringut koos.

Esimene ja OAI-PMH enda dokumentsiooni poolt soovitatud lahendus on kasutada ListRecords päringut. Antud lahenduse eelisteks on:

- Lahendus kasutab ainult üht päringut kogu sünkronisatsiooni ning uuendamise läbiviimseks.
- Antud implementatsiooni soovitab standardi looja.

- Tulenevalt ühest päringust on liidese loomine võrdlemisi odav nii arhitektuuriliselt kui ka ajaliselt.

See toob kaasa kaks suuremat probleemi:

- ListRecords ei lase päringu tegijal kontrollida vastuse suurust, kuna *record* objektide arvu vastuses määrab repositoorium. Kuna kontrollimatu suurusega vastus loetaks vahemällu ning vahemälu piiratud suuruse tõttu võib süsteem anda vea (Javas OutOfMemoryError) ning päringu vastus jääb seeläbi vastu võtmata. Sama probleem esineb väga suurte SOAP päringute puhul [11] , mida kasutab ka Eesti X-tee. See tooks kaasa vajaduse luua eraldi HTTP päringu vastuse tükkidena lugemise loogika. Kuid kuna mälu otsa saamise viga tuleks välja ainult juhtudel, kus ainult mõni ListRecords päringu vastus on liiga suur, siis võib see arenduse käigus jääda märkamata. Selle tõttu võib veaparandus jääda implementeerimata või lahendus implementeeritakse hiljem, mis on kulukam.

- Teine probleem on objektide värskendamisega, kus uuendamiseks peaks iga kord töötlema läbi kõik repositooriumi materjalid. See võib tähendada olukorda, kus näiteks igaöisel sünkroniseerimisel töödeldakse läbi kõik dokumendid, isegi kui ühtegi neist uuendatud ei ole. Selle probleemi vältimiseks peaks looma mingi eraldi teenuse, mis kontrolliks uuendatud materjalide olemasolu, mis tähendaks lisakeerukust süsteemile.

Teine võimalus on kasutada ListIdentifiers päringut, et itereerida üle kõigi antud repositooriumis sisalduvate materjalide identifikaatorite. Iga unikaalse identifikaatori kohta on võimalik teha repositooriumile eraldi päring GetRecord ning lugeda vastuses ainult üht *record* objekti. Antud lahenduse eelisteks on:

- Välditakse olukorda, kus vastus võib olla liiga suur, kuna igale päringule vastatake ainult ühe *record* objektiga.

- Andmete uuendamiseks saab kasutada antud liidest ilma modifikatsioonideta, kuna töödeldakse *record* objekti, mida pole veel teenusepakkuja andmebaasis või mille uuendamiskuupäev on sama, mis andmebaasis oleval.

Antud lahenduse puuduseks on:

- Kuna tuleb realiseerida kaks HTTP päringut, siis võib liides olla arhitektuuriliselt keerulisem kui esimesel lahendusel.

Kahte lahendust võrreldes võib järeldada, et OAI-PMH standardi enda poolt toetatav lahendus võib tunduda küll esmapilgul lihtne, kuid toob kaasa mõningaid väga suuri riske ning halba arhitektuuri. Seega võib öelda, et liides, mis kasutab ListRecords ja GetRecord päringute kombinatsiooni, on vähem veaohtlik ning toetab sama lahendusega nii esmakordset sünkronisatsiooni kui edaspidist uuendamist võrdlemisi lihtsalt.

### 3.4.2 Liidese parseri valik

Läbiviidud mõõtmiste tulemustest võib järeldada, et õige parsimisloogika valik ei ole mitte ainult ajaline võit vaid reaalse probleemi vältimine suurte infomahtude juures. Nimelt saab antud analüüsi põhjal väita, et mida suuremaks lähevad aja jooksul repositooriumid, seda rohkem kasu annab Xpathiga parsimine. Antud katse näitab ka kui tähtis on arvestada parsimismeetodi valikul suurte andmekogustega liidese disainifaasis, sest väikeste andmemahtude puhul on vahe marginaalne, kuid suurte andmemahtude puhul võib vale meetodi valik osutuda süsteemile halvavaks, kui info töötlemise ajaline kasv on eksponentsiaalne. Tähtis on ka ära märkida, et XPath välistab ka võimaluse salvestada valesid elemente. Näiteks kui otsitakse dokumendist elementi, mida kasutatakse mitmes kohas, siis Xpathiga loetakse vaid õiges kontekstis olevad elemendid, kuid DOMiga loetakse kõik antud elemendi kordused terves dokumendis. Seega valitakse antud uurimistöö rakenduse jaoks XPathi kasutav XML dokumenti töötlev lahendus.

## 3.5 Lahenduse valiku kokkuvõte

Läbiviidud analüüsi põhjal luuakse rakendus, mis põhineb kahel eelnevalt uuritud ning tõestatud lahendusvariandil. Rakendus hakkab seega kasutama OAI-PMH protokolli kahe päringu hübriidlahendust, milleks on GetRecord ning ListIdentifiers päringud. Liidese parsimismeetodiks valitakse XPath, kuna ta on märgatavalt kiirem ning ka usaldusväärsem. Antud lahendus tagab

# 4 Rakendus

## 4.1 Arhitektuuri analüüs

Antud rakenduse üldine arhitektuur põhineb objektorienteeritud programmeerimise paradigmal. Liidese disainimisel kasutatakse programmeerimise mustrit Iteraator.

Objektorienteeritud arhitektuur osutus valituks, kuna liides tegeleb metaandmete ja repositooriumide objektide töötlemisega, mida võib olla erinevat tüüpi, kuid millel on ühiseid infovälju [16] . Loodud parser peab näiteks toetama info sünkronisatsiooni nii EstCore kui ka LOM formaadis, mis saavad jagad üht ühist parseri objekti, kuid alamobjektis lisada standardi spetsiifilisi välju ning loogikat. Antud arhitektuur on ka selle tõttu väga sobiv, et süsteem sünkroniseerib päris maailmas olemasolevaid objekte, mis on üks objektorienteerituse põhiideid. Tarkvaraarenduse seisukohalt on objektorienteeritud arhitektuur antud rakenduses kasulik ka selletõttu, et antud stiil julgustab kasutama põhimõtteid nagu komponentide taaskasutamine ning ühtekuulumine. Komponentide taaskasutamine ning pärimine väldib koodi kordamist ning võimaldab koodi paremini lugeda. Ühtekuulumise printsiibi all viiakse selle rakenduse kontekstis ühte eesmärki täitvad väljad ning meetodid ühte objekti, mis teeb objektide struktuuri kergemine jälgitavaks ning hallatavaks.

Tarkvara disaini mustrid on üldised taaskasutatavad lahendused tihtiesinevatele probleemidele tarkvaraarenduse protsessis [18] . Seega on mõistlik arhitektuuri loomisel uurida olemasolevaid mustreid, et lahendada probleem tööstuse standardit järgi ning kasutada ära teadmisi, mis on juba antud valdkonnas olemas. Antud rakenduse osa, mis küsib ükshaaval repositooriumist materjale on klassikaline probleem, kus on vaja järjestikku itereerida üle objektide kollektsiooni, mille siseehitus ei pruugi olla teada või mille teadmine lisaks keerukust. Selleks on loodud Iteraator muster, mis võimaldab kliendil käia läbi kollektsiooni infot nii, et klient ei tea kuidas kollektsiooni hoidev konteiner antud kollektsiooni sisemiselt salvestab [17] .

Antud rakenduse analüüsi osas valitud kahe meetodi – ListIdentifiers ning GetRecord loogika toetab Iteraatori mustri kasutamist kahekordselt. Nimelt saab jagada loogika kaheks iteraatoriks vastavalt kahele OAI-PMH teenusele, mida süsteem kasutab. Esiteks tähendab see seda, et saab ükshaaval itereerida üle iga unikaalse identifikaatori, mida repositoorium pakub. Teenus, mis antud iteraatorit kasutab, ei pea seega kunagi teadma sellest, et iteraator sisemiselt teeb päringuid repositooriumile resumptionToken loogika alusel. Antud iteraatorit kasutava teenuse jaoks on antud iteraator lihtsalt nimekiri identifikaatoritest, mida saab mugavalt kasutada

Teine Iteraator mustri kasutus on võimalik materjalide ükshaaval pärimisel repositooriumist, kui kasutatakse GetRecord päringut [10] . Materjalide iteraator saab sisemiselt kasutada eelnevalt kirjeldatud identifikaatorite iteraatorit ja seeläbi abstraheerida materjalidest hooliva materjalide iteraatori jaoks kogu identifikaatorite loogika. Seega näiteks repositooriume haldav teenus, kes soovib saada ainult materjalide objekte ja salvestada neid andmebaasi, küsib vaid järgmist materjali. Seeläbi ei tea materjalide iteraatorit kasutav teenus kunagi, et sisemiselt kasutab materjalide iteraator identifikaatorite iteraatorit, keerulist XMLi töötlemisloogikat ning kaht erinevat päringut.

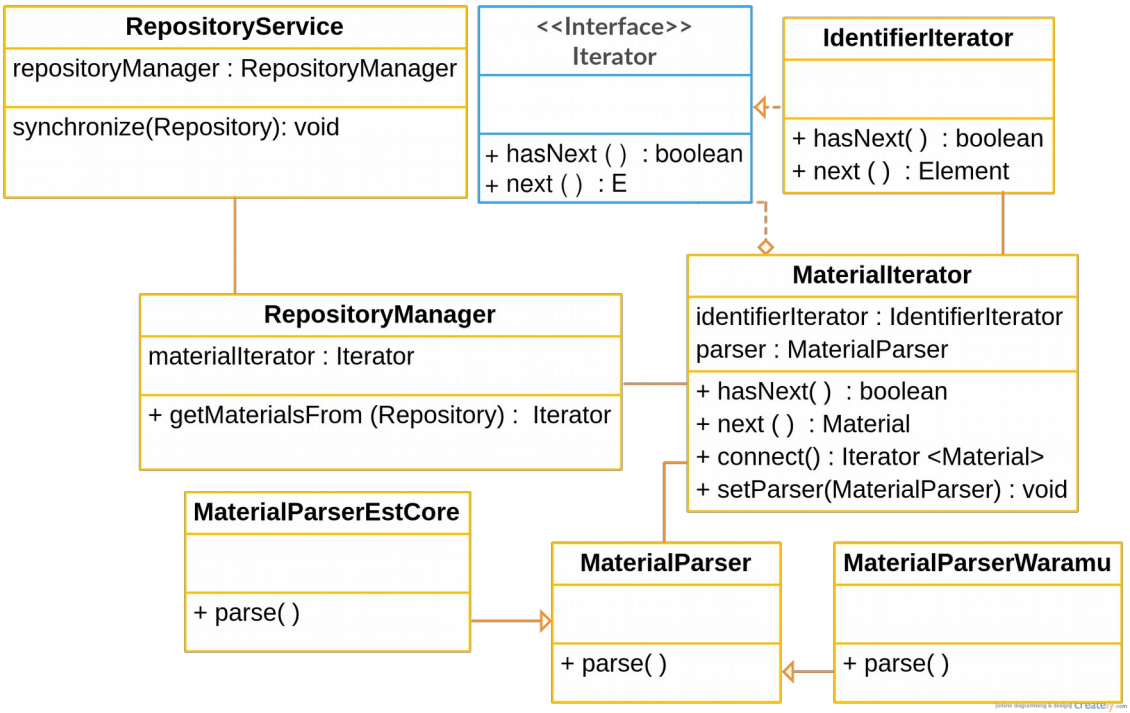
Rakendus ei tegele mitte ainult andmete esmakordse sünkronisatsiooni vaid ka värskendamisega. Metainfot värskendatakse uuendamise kuupäeva järgi, mis on olemas ka ListIdentifiers [7] päringus. See tähendab seda, et kui antud liides uuendab juba olemasolevat infot, siis tal on võimalik võrrelda kuupäeva, mis on andmebaasis selle kuupäevaga mis on ListIdentifiers päringus. Kui materjali pole uuendatud võrreldes selle versiooniga, mis on juba olemas, siis ei tehta antud materjali kohta GetRecord [10] päringut. Sellega hoitakse kokku nii süsteemi ressurssi kui ka aega, mis kulub antud liidese käitamisele.

Antud arhitektuur on seega taaskasutatav, loogiliselt eraldatud ning tänu oma sisemisele arhitektuurile vähem veaohtlik kui võimalikud alternatiivid. Loodud disain on nii aja kui kuluefektiivne ning kasutab disainimustreid, mis on juba ennast tõestanud sarnastes ülesannetes.

## 4.2 Liides realisatsioon

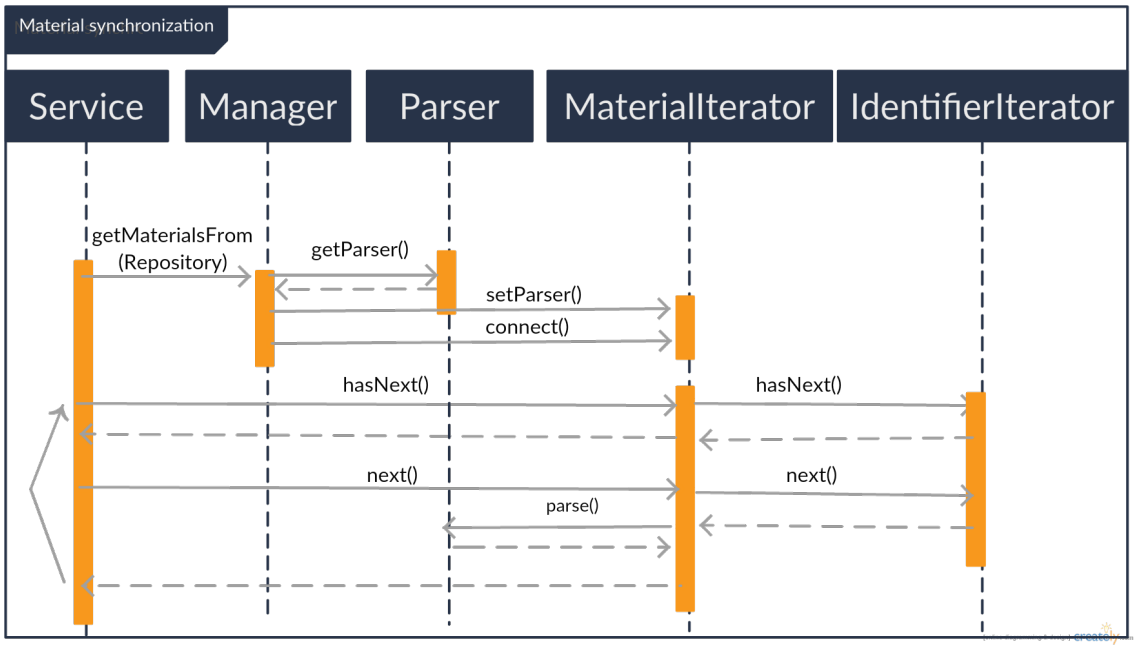
Antud peatükis antakse täielik ülevaade protsessist, mis sünkroniseerib andmeid üle OAI-PMH liidese ning töötleb XML vormingus metainfot.

Loodud moodul kasutab Java platvormi, sõltuvuste sisestamiseks kasutatakse Google poolt loodud tehnoloogiat Guice, mis teeb koodi muutmise ning taaskasutamise lihtsamaks ning loogilisemaks [19] . Pärast informatsiooni sünkronisatsiooni ning töötlemist salvestatakse info MySQL andmebaasi Hibernate objekt-relatsioonilise andmeteisendaja abil. Antud tehnoloogiate kasutust illustreerivad Joonis 9 ning Joonis 10. Kokku on loodud moodulis üle kahekümne klassi.




Joonis 9. Liidese lihtsustatud klassidiagramm

Sisenemispunkt loodud komponenti on RepositoryService (Joonis 9). Rakendus, mis soovib antud komponenti kasutada, kutsub välja synchronize meetodi ning annab kaasa RepositoryService'ile informatsiooni repositooriumi kohta, mille metainfot soovitakse sünkroniseerida. Kui repositooriume on mitu, siis kutsutakse antud meetod välja iga repositooriumi kohta. Antud lahendus on seega universaalne igale repositooriumile.




Joonis 10. Liidese lihtsustatud jadadiagramm

Pärast sünkronisatsiooni alustamist küsib RepositoryService teenus RepositoryManager'i käest *getMaterialsFrom* meetodiga (Joonis 10) eelnevalt analüüsitud iteraatori. RepositoryService kasutab antud iteraatorit, et käia läbi kõik antud repositooriumi materjalid ning salvestada need andmebaasi. RepositoryManager'i kasutamisel seab antud teenus üles ühenduse repositooriumiga läbi MaterialIterator'i käsu connect. Vastavalt repositooriumis kastutatavale XMLi ülesehitusele, seatakse RepositoryManager'is vastav parser setParser meetodiga, sest metainfo võib olla näiteks nii EstCore kui ka LOM formaadis. See tähendab, et võimalikult palju komponente on lahtiühendatud protokollispetsiifilisest loogikast.

Seega kui MaterialIterator saab oma Iterator (Joonis 9) arhitektuurist tuleneva käsu next, siis ta pöördub IdentifierIterator'i poole samuti käsuga next ning saab materjali unikaalse identifikaatori. Selle põhjal teeb ta OAI-PMH GetRecord [10] päringu, saab materjali XML notatsioonis metainfo. Pärast seda töödeldaks metainfo Material objekti eelnevalt seatud parseriga, mis pärib üldisest MaterialParser (Joonis 9) objektist. Antud arhitektuur tagab selle, et kirjutatakse võimalikult vähe repositooriumi spetsiifilist koodi.

Loodud liidese ning parseri arhitektuur pole seega protokollist olenev nii RepositoryManager'i kui ka RepositoryService'i osas, mis tähendab seda, et näiteks uute OAI-PMH protokollile konkureerivate protokollide tekkimisel või uute metaandemete formaatide lisandumisel saab süsteemi ümber ehitada väikese vaevaga ning taaskasutada võimalikult palju olemasolevat koodi.

# 5 Kokkuvõte

Antud töö eesmärgiks oli analüüsida OAI-PMH protokolli ning üle liidese kantava metainfo parsimise võimalusi. Analüüsi tulemusena pidi valmima efektiivne OAI-PMH protokolli liides ning ülekantava metainfo töötleja.

Eesmärgi saavutamiseks uuriti OAI-PMH protokolli, erinevaid parsimisvõimalusi ning viise kuidas luua võimalikult modulaarne arhitektuur antud komponendi valmistamiseks. Analüüsi tulemusena leiti, et on olemas turvalisem ning skaleeruvam viis andmeid sünkroniseerida, kui on hetkel protokolli tavapraktika.

Töö tulemustest võib ka järeldada, et XML vormingus info parsimine, antud rakenduse vajaduste jaoks on kõige efektiivsem kasutades XPath tehnoloogiat. Kõige suurema ajavõidu andis antud tehnoloogia just väga suurte andmemahtude puhul.

Töö käigus valmis arhitektuuriliselt läbimõeldud, korduvkasutatav ning potentsiaalseid riske maandav OAI-PMH liidese ja parseri komponent, mida on võimalik integreerida igasse Java platvormil töötavasse rakendusse hoolimata ülekantava info suurusest või formaadist.

Töö võimalikeks edasiarendusteks on antud loogika laiendamine teistele protokollidele ning andmeformaatidele.

Töö käigus saavutati eesmärk luua kiire ning efektiivne moodul info sünkroniseerimiseks üle OAI-PMH protokolli. Õige tehnoloogia ning arhitektuuri kasutamine muutis süsteemi kiiremaks, turvalisemaks ning tulevikukindlamaks.

# Kasutatud kirjandus


[1] Prof. Kuldar Taveter. Tarkvarasüsteemi nõuete inseneeria, 22 [Online] http://maurus.ttu.ee/sts/wp-content/uploads/2014/09/IDK0071-Loeng-2-Tarkvaras%C3%BCsteemi-n%C3%B5uete-inseneeria.pdf (10.03.16)

[2] Carl Lagoze, Herbert Van de Sompel. The Open Archives Initiative Protocol for Metadata Harvesting [Online] https://www.openarchives.org/OAI/openarchivesprotocol.html#ProtocolFeatures (10.03.16)

[3] Leona Carpenter. OAI for Beginners, the Open Archives Forum online tutorial [Online] https://www.oaforum.org/tutorial/english/page2.htm (16.04.2016)

[4] Carl Lagoze, Herbert Van de Sompel. Registered Data Providers [Online] https://www.openarchives.org/Register/BrowseSites (16.04.2016)

[5] Martin Sillaots. Waramu [Online] http://koolielu.ee/pages/view/5220/waramu (16.04.2016)

[6] Digitaalse õppevara keskus [Online] https://ekoolikott.ee/about (16.04.2016)

[7] Carl Lagoze, Herbert Van de Sompel. ListRecords [Online] https://www.openarchives.org/OAI/openarchivesprotocol.html#ListRecords (17.04.2016)

[8] Carl Lagoze, Herbert Van de Sompel. XML Response Format [Online] https://www.openarchives.org/OAI/openarchivesprotocol.html#XMLResponse (17.04.2016)

[9] Carl Lagoze, Herbert Van de Sompel. XML Response Format [Online] https://www.openarchives.org/OAI/openarchivesprotocol.html#ListIdentifiers (17.04.2016)

[10] Carl Lagoze, Herbert Van de Sompel. GetRecord [Online] https://www.openarchives.org/OAI/openarchivesprotocol.html#GetRecord (17.04.2016)

[11] SOAP - Very large XML response - OutOfMemoryError [Online] http://stackoverflow.com/questions/6926620/soap-very-large-xml-response-outofmemoryerror (20.04.2016)

[12] Dublin Core Metadata Element Set, Version 1.1 [Online] http://dublincore.org/documents/dces/ (20.04.2016)

[13] Learning Resource Meta-data Specification [Online] http://www.imsglobal.org/metadata/index.html (09.05.2016)

[14] Jonathan Robie, Michael Dyck, Josh Spiegel. XML Path Language (XPath) 3.1 [Online] https://www.w3.org/TR/xpath-31/ (12.05.2016)

[15] What is the DOM? [Online] http://www.w3schools.com/xml/dom_intro.asp (12.05.2016)

[16] Object-Oriented Architectural Style [Online] https://msdn.microsoft.com/en-us/library/ee658117.aspx#ObjectOrientedStyle (15.05.2016)

[17] Kuchana P., Software Architecture Design Patterns in Java, Journal of Communication, Taylor & Francis Group, 2004, pp. 147-158.

[18] Definition: pattern (design pattern) [Online] http://searchsoftwarequality.techtarget.com/definition/pattern (15.05.2016)

[19] Guice [Online] https://github.com/google/guice (18.05.2016)