Tallinna Ülikool
Informaatika Instituut

Registriandmete avalikustamisest Ehitisregistri näitel

Bakalaureusetöö

Autor: Hannes Mäehans
Juhendaja: Jaagup Kippar
Välisjuhendaja: Peep Küngas

Autor: .....„……..“ 2012
Juhendaja:.....„……..“ 2012
Välisjuhendaja:.....„……..“ 2012
Instituudi direktor:.....„……..“ 2012

Tallinn 2012

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Sisukord

Sissejuhatus

Käesoleva bakalaureusetöö eesmärgiks on pakkuda välja lahendus lingitud avaandmete komplekti koostamiseks taaskasutades riigi infosüsteemi registrite andmeteenuseid. Eestis puudub senine praktika andmete avalikustamisel kasutades riigi infosüsteemide andmevahetuskihti X-tee ja sellest tulenevalt on tegu pretsedendiga.

Teema aktuaalsust kinnitab valitsusliidu programmis aastateks 2011-2015 punktis E-Riigist I-Riigiks väljatoodud avalike ehk riigi ja kohalike omavalitsuste andmete masinloetavaks muutmine (Isamaa ja Res Publica Liit ning Eesti Reformierakond). Lisaks on käesoleva bakalaureusetöö praktiline osa Majandus- ja Kommunikatsiooniministeeriumi (MKM) hanke „*OpenData in Cloud Piloodi Hange*“ realisatsioon. Autor osaleb hanke võitnud firmas programmeerijana.

Töö esimeses peatükis räägib autor avaandmete mõistest, andmete olemusest ning toob välja kirjeldatud andmete kasutusi. Järgmises peatükis käsitletakse lingitud andmeid, kasutatavaid tehnoloogiaid. Teema arendusena jõutakse lingitud avaandmete ja nende kasutuseni.

Töö kolmandas peatükis kirjeldab autor Ehitisregistri olemust. Lähemalt vaadeldakse tavakasutajale mõeldud ligipääsuvõimalusi, räägitakse X-tee teenustest ning seletatakse valmivat lahendust.

Töö neljandas peatükis võrdleb autor kahte publitseerimistööriista ning käsitleb projektijuhi poolt valitud tööriista paigaldust ning funktsionaalsuse testimist. Kolmandale osapoolele tööriista paigaldamise lihtsustamiseks koostab autor paigaldusjuhendi.

Viiendas peatükis kirjeldab autor andmete avalikustamise võimalusi. Autor kirjeldab hankes pakutud andmete laadimisskeemi ning alternatiivseid lahendusi.

Töö mahukaimas kuuendas peatükis räägib autor X-tee näidisteenuse püstitamisest „*soapUI“* tööriista abil. Autor kirjeldab andmemudeli loomist. Viimase testimiseks peab töö kirjutaja ning projektijuht oluliseks eri liiki ehitiste X-tee päringute vastuseid. Viimaste salvestamiseks loob autor päringuid sooritava skripti, mis salvestab vastused failidena.

Soovitud päringuid pidi sooritama kolmas osapool, mille tõttu lõi autor skripti kasutamiseks juhendi. Näidisvastuste abil koostab autor salvestamisskripti. Ehitisregistri andmete kasutamise mugavdamiseks seotakse andmekogu aadressid Aadressiandmete süsteemi aadressidega. Järgmisena kajastab autor X-tee päringute sisendite ja veatöötlusega tegelevat osa.

Seitsmendas peatükis käsitletakse publitseerimistööriista konfigureerimist ning andmete lingitud kujule viimisest. Töö viimases osas kirjeldab autor valminud lahenduse arhitektuuri 4+1 raamistikust lähtudes komponendi- ja levitusskeemi abil.

Lisa 1 on töö neljandas punktis valminud publitseerimistööriista paigaldusjuhend. Lisa 2 on koostatud näidispäringute skripti paigaldamis- ja kasutusjuhend. Mainitud lisad ning tööga valminud koodinäited asuvad http://www.tlu.ee/~maehans/Bakalaureus/ aadressil ja bakalaureusetööga kaasneval andmekandjal.

1 Avaandmed

Avatud („*Open Definition*“) defineerib printsiibi, mille kohaselt sisu ja andmed peaksid olema kättesaadavad – sonetist statistikani, geenidest geo-andmeteni. Definitsiooni saab kokku võtta lausega: „Sisu või andmed on avatud, kui igaüks saab neid vabalt kasutada, taaskasutada ja jagada – teha seda viidates ning kasutades samu levitamistingimusi" (Open Definiton).

Avaandmed, eriti valitsuse avaandmed, on tohutu ressurss, mis on seni suures osas kasutamata. Paljud inimesed ja organisatsioonid koguvad eri tüüpi andmeid, et täita oma ülesandeid. Valitsuse osa on selles oluline, nii kvantiteedi, tsentraalsuse osas, kui ka seetõttu, et enamik valitsuseandmeid on avalikud ja seetõttu võiks olla kättesaadavad (Dietrich, et al.).

Avatud valitsuse üks põhitugi on vaba juurdepääs teabele ja võimalus vabalt kasutada ning taaskasutada teavet. Ilma andmeteta ei ole võimalik luua koostööd kultuuri ja sidusrühmade vahel. Valitsuse avaandmeid („*Open Government Data“* - *OGD*) peetakse sageli oluliseks aspektiks avatud valitsuses (Bauer & Kaltenböck, 2012).

Andmete ja teabe tõhus haldamine on äärmiselt oluline maailmamajanduses. Andmete ja teadmiste jagamisega on võimalik tagada parem otsustusvõime, suuremate projektide arendus ning edendada säästva energia rahastamist (Bauer & Kaltenböck, 2012).

OGD on ülemaailmne liikumine, mis keskendub valitsuse andmete inimestele avamisele kui ka andmete masinloetavaks muutmisele. Avatakse ainult valitsuse toodetud või tellitud andmed, isikustatud andmeid ei avata (Bauer & Kaltenböck, 2012).

1.1 Avaandmete printsiibid

Andmeid loetakse avatuks, kui on kooskõlas järgnevaga (Bauer & Kaltenböck, 2012):

- **terviklikkus** - Kõik avalikud andmed tehakse kättesaadavaks. Selle alla loetakse kõik andmed, mida ei piira isikuandmete ja muul põhjusel salastatud andmete piirangud;
- **pärineb algallikast** - Andmed on töötluseta kogutud algallikast. Andmeid ei koondata vaid säilitatakse nende detailsus;
- **ajakohasus** - Andmekogum on avaldatud võimalikult kiirelt, et säilitada selle ajakohasus;

- **kättesaadav** - Andmetele on ligipääs võimalikult laial kasutajateringil, võimalikult laia kasutuseesmärgiga;
- **masinloetav** – Andmed on struktureeritud masinloetaval viisil;
- **mitte diskrimineeriv** – Andmed on kättesaadavad kõigile, ilma registreerimiseta.
- **vaba formaat** - Andmed on esitatud vabas formaadis, mis ei ole ühegi ettevõtte ega isiku ainuomand;
- **vaba litsentsiga** - Andmed ei ole kaitstud autoriõiguse, patendi, kaubamärgi ega ärisaladuse regulatsiooniga. Mõistlikud privaatsus- ja turvalisuspiirangud on lubatud.

1.2 Kasutus

Mitmed inimrühmad ja organisatsioonid saaksid kasu andmete kättesaadavusest, sealhulgas valitsus ise. Samal ajal on võimatu ennustada, kus ja kuidas tulevikus väärtus luuakse. Innovatsiooni loomuses on üllatuslike lahenduste teke. Juba praegu on võimalik osutada mitmele valdkonnale, kus avaandmed loovad uut väärtust (Dietrich, et al.):

- demokraatia läbipaistvus;
- osalus;
- uued või täiustatud tooted ja teenused;
- innovatsioon;
- paranenud valitsuse teenused;
- andmeallikate kombineerimisel tekkinud uued teadmised.

Läbipaistvusega seotud projektid nagu Soome „maksu puu“ („*tax tree*“) ja Inglise „kuhu mu raha läheb“ („*where does my money go*“) näitavad kuidas maksumaksja raha kulutatakse. Kanadas avastati avaandmete abil 3.2 miljardi dollari väärtune maksupettus. Taani folketsting.dk jälgib parlamendi tegevust ja õigusloome protsesse, võimaldab näha, mis täpselt valitsuses toimub ja millised liikmed on kaasatud (Dietrich, et al.).

Avaandmed võimaldavad teha paremaid otsuseid elus ning olla inimestel ühiskonnas aktiivsemad. Naine Taanis lõi findtoilet.dk lehe, mis kuvab kõik Taani avalikud tualetid, et põieprobleemidega inimesed saaksid julgemalt väljas käia. Hollandis on teenus www.vervuilingsalarm.nl, mis hoiatab, kui järgmise päeva õhukvaliteedi tase langeb alla kasutaja poolt defineeritud lävendi. New Yorkis saab mugavalt vaadata, kus koeraga jalutada

ning lisaks vaadata inimesi, kes kasutavad samu jalutuskohti. Teenused nagu mapnificent.net ja mapumental.com aitavad otsida piirkondasid, kus elada, võttes arvesse töölejõudmise aega, kinnisvara hindu ning piirkonna ilu. Kõik kirjeldatud näited kasutavad avaandmeid. (Dietrich, et al.)

Majanduslikult on avaandmed samuti väga olulised. Mitmed uuringud on hinnanud avaandmete majandusliku väärtuse ainuüksi Euroopas kümnete miljardite eurodeni aastas. Taani husetsweb.dk aitab leida võimalusi, kuidas parandada kodu energiatõhusust, hõlmates finantsplaneerimist ning pakkudes ehitajate kontakte. Viimane teenus põhineb katastri, riiklike toetuste ning kohaliku äriregistri informatsioonil (Dietrich, et al.).

Google'i tõlge kasutab tohutut hulka Euroopa Liidu dokumente, mis esinevad kõikides Euroopa keeltes, arendamaks tõlkimise algoritme ning seeläbi parandades teenuse kvaliteeti (Kroes, 2010).

Avaandmed on väärtus valitsusele, suurendades valitsemise tõhusust. Hollandi Haridusministeerium on taaskasutamiseks avaldanud kõik haridusega seotud andmed. Sellest alates on vähenenud neile laekuvate küsimuste arv, vähendades töökoormust ja kulusid, samuti on töötajatel lihtsam vastata, kuna on selge, kus vastavaid andmeid võib leida. Avaandmed muudavad valitsuse efektiivsemaks, mis kokkuvõttes vähendab kulutusi. Taani kultuuripärandi osakond avaldab andmeid ja teeb koostööd ajaloo grupiga „*Wikimedia Foundation*", enda ülesannete paremaks täitmiseks. See ei paranda mitte ainult andmete kvaliteeti, vaid teeb ka osakonna väiksemaks (Dietrich, et al.).

Avaandmed ei tähenda ainult valitsuse andmeid vaid hõlmab andmeid ka teistest valdkondadest. Näiteks ettevõtlus, tööstus, kodanikud, mittetulundusühingud, haridus- ja teadusorganisatsioonid. Hetkel tuntuimad on Maailmapank, ÜRO, „*New York Times*", „*The Guardian*" ja „*Open Knowledge Foundation*" (Bauer & Kaltenböck, 2012).

Kuigi on mitmeid juhtumeid, kus avaandmed juba loovad sotsiaalset ning majanduslikku väärtust, ei osata ennustada millised uued võimalused veel tekivad. Uued andmete kombinatsioonid võivad luua uusi teadmisi ja arusaamu, mis võivad luua uusi lahendusi. (Dietrich, et al.)

2 Lingitud andmed

Lingitud andmed („*Linked data*“) loovad seoseid erinevatest allikatest pärit olevate andmete vahel. Andmed võivad pärineda mitmetest andmebaasidest, mis kuuluvad eri organisatsioonidele ja on geograafiliselt erinevates asukohtades või ühe organisatsiooni ebaühtlasest süsteemist. Tehniliselt on lingitud andmed veebis avaldatud andmed, mis on masinloetavad, mille tähendus on selgelt määratletud ja on seotud teiste väliste andmetega ning neid on omakorda võimalik siduda väliste andmekogudega (Bizer, Heath, & Berners-Lee).

Kui tavaveebi primaarseks osaks on hüperlinkide kaudu ühendatud HTML („*HyperText Markup Language*“) dokumendid, siis lingitud andmed tuginevad dokumentidele, mis sisaldavad andmeid RDF („*Resource Description Framework*“) formaadis. Lihtsate seoste loomise asemel kasutavad lingitud andmed RDF-i, et luua defineeritud tüüpidega seoseid suvaliste asjade vahel maailmas (Bizer, Heath, & Berners-Lee).

Tim Berners-Lee on loonud reeglistiku kuidas avalikustada andmeid nii, et nad saaksid osaks ülemaailmsest andmeruumist (Bizer, Heath, & Berners-Lee):

- kasutage URI-sid („*Uniform Resource Identifier“*) asjade identifikaatoritena;
- kasutage HTTP URI-sid, et inimesed saaksid neid otsida;
- paigutage URI aadressile informatsioon, kasutage standardeid (RDF, SPARQL – rekursiivne lühend, mis viitab iseendale „*SPARQL Protocol and RDF Query Language“*);
- andmetes linkige teistele URI-dele, et oleks võimalik leida rohkem informatsiooni.

Neid reegleid tuntakse kui „Lingitud andmete põhimõtteid“ („*Linked Data Principles*“), mis on põhiline juhend lingitud andmete avaldamiseks (Bizer, Heath, & Berners-Lee).

2.1 Tehnoloogia

Lingitud andmed toetuvad tehnoloogiatele, mis on veebis fundamentaalsed: URI ja HTTP protokoll. URL-i („*Uniform Resource Locator“*) tuntakse, kui dokumendi või muu olemi aadressi veebis, URI aga pakub üldisemat lahendust maailmas eksisteeriva olemi

identifitseerimiseks. Kui olemid on identifitseeritud URI abil, mis kasutavad „http://“ süsteemi on võimalik neid olemeid vaadata lihtsalt üle HTTP protokolli. HTTP pakub lihtsat ning universaalset meetodit ressursi edastamiseks, mida on võimalik serialiseerida baidivoona (näiteks koera foto) või objekti andmete edastamiseks, mida ei ole võimalik üle võrgu saata (näiteks koer ise). URI ja HTTP lahendusi täiendab andmetevõrgule kriitiline RDF tehnoloogia.

RDF pakub graafipõhist andmemudelit, mis võimaldab luua sobiliku struktuuri ning seosed. RDF mudel kodeerib andmed subjekti, predikaadi ja objekti kolmikutena. Subjekt on kolmiku esimene osa, mis identifitseerib ressursi, mida RDF kolmikuga kirjeldatakse. Teine osa on predikaat, mis määrab subjekti omaduse. Kolmandal kohal on objekt, mida nimetatakse ka omaduse väärtuseks. Objektiks on teise subjekti URI või predikaadi poolt lubatud väärtus (Microsoft Corporation).

Kasutatavate predikaatide kirjeldamiseks defineeritakse ontoloogia. Ülesmärkimiseks kasutatakse veebis edastavat ontoloogia keeles (OWL – „*Web Ontology Language“*). Ontoloogia on sarnane skeemiga, kuna see määrab elementide kogumi, mida saab rakenduses kasutada. Ontoloogia aitab samuti defineerida predikaatide ja objektide domeeni (Microsoft Corporation).

RDF kolmik võib kirjeldada, et kaks isikut A ja B (mõlemad identifitseeritud URI abil) on seotud faktiga, et isik A tunneb isikut B. Sarnaselt võib RDF kolmik siduda isiku C ja teises andmebaasis oleva teadusartikli D väites, et C on D autor. Sel viisil lingitud ressursid võivad pärineda erinevatest andmekogudest, luues andmetevõrgu. Seega võime RDF kolmikuid, mis lingivad erinevaid andmekogumeid, vaadata kui hüperteksti linke, mis seovad kokku dokumentide võrgu.

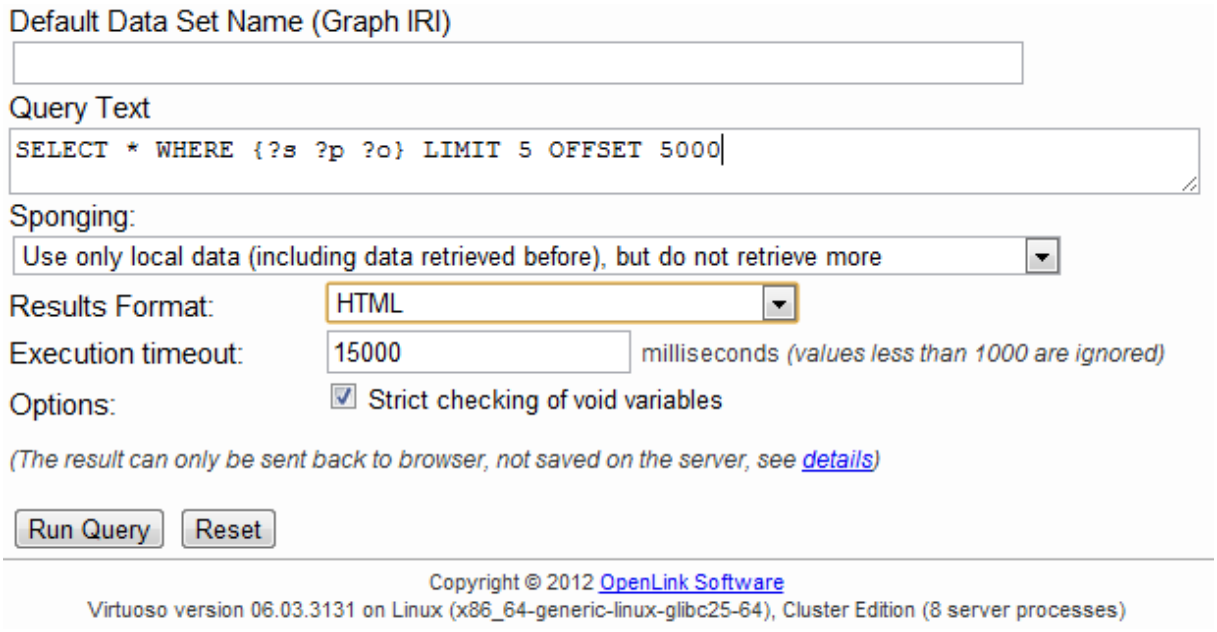
RDF pseudokood (vt Koodinäide 1) kirjeldab Tim Berners-Lee kuuluvust „*MIT Decentralized Information Group*“ organisatsiooni. Subjekti URI-le vastab „dig.csail.mit.edu“ server organisatsiooni kirjeldava RDF-ga. Objekti URI-le vastab w3c.org server Tim Berners-Lee infofailiga (Bizer, Heath, & Berners-Lee). Predikaadi URI-le pöördudes suunatakse ümber http://xmlns.com/foaf/spec/ aadressile, kus on võimalik allalaadida FOAF ontoloogia.

```
Subjekt:   http://dig.csail.mit.edu/data#DIG
Predikaat: http://xmlns.com/foaf/0.1/member
Objekt:    http://www.w3.org/People/Berners-Lee/card#i
```

Koodinäide 1 – RDF kolmik

RDF kõrval on oluline SQL-i sarnane SPARQL protokoll ja päringukeel. Keel võimaldab pärida andmekomplekte, nende ühisosa, eriosa või valikulisi andmeosasid. SPARQL toetab agregaatfunktsioone, alampäringuid, eitust, avaldisi, väärtuste kontrolli ning erinevatele andmekogudele erinevate piirangute defineerimist. Tulem tagastatakse RDF kolmikutena. SPARQL päringukeel võimaldab teha päringuid erinevatele andmeallikatele, olenemata kas andmed on salvestatud algupäraselt RDF kujul või on loodud relatsiooniliste andmebaaside RDF vaated.

Autor tegi SPARQL võimaluste selgitamiseks näidispäringu http://lod.openlinksw.com/sparql aadressil asuvale SPARQL teenusele (vt Ekraanipilt 1). Päringus küsitakse kõik RDF kolmikud, limiteeritakse viiele tulemile ning kasutatakse 5000 kohalist nihet („*offset*“). Vastuses (vt Ekraanipilt 2) on kuvatud subjektidena viis URI, predikaadina on kasutusel kõigil RDF ontoloogia tüüp („*type*“) predikaat ning objektina on kasutusel „*Freebase*“ ontoloogiast mägi („*mountain*“) tüüpi ontoloogia klass. Lihtsamalt seletades on vastuses kirjeldatud, et viis URI-ga määratud subjekti on mäed.



Ekraanipilt 1 – SPARQL teenus

s	p	o
http://freebase.com/guid/9202a8c04000641f8000000000cd8a9e	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://freebase.com/geography/mountain
http://freebase.com/guid/9202a8c04000641f8000000006bb4a82	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://freebase.com/geography/mountain
http://freebase.com/guid/9202a8c04000641f8000000006f79407	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://freebase.com/geography/mountain
http://freebase.com/guid/9202a8c04000641f8000000007341f77	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://freebase.com/geography/mountain
http://freebase.com/guid/9202a8c04000641f8000000006bb1fa9	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://freebase.com/geography/mountain

Ekraanipilt 2 – SPARQL vastus

2.2 Lingitud avaandmed

Maksimaalse kasu saavutamiseks, on oluline andmed luua konteksti, mis loob uusi teadmisi ja võimaldab keerulisi teenuseid ja rakendusi. Lingitud avaandmed (*„Linked Open Data“)* lihtsustab innovatsiooni ja teadmiste loomist ning on oluline vahend teabe haldamisel ja integratsioonil (Bauer & Kaltenböck, 2012). Teed avaandmetest lingitud avaandmeteni on Tim Berners-Lee seletanud viie tärni mudeli abil (vt Tabel 1 – Viie tärni mudel).

★	Informatsioon on veebis saadaval (igas formaadis) avatud litsentsiga
★★	Informatsioon on masinloetav, struktureeritud andmed (nt pildi asemel Exceli fail)
★★★	Kasutatakse vaba formaati (nt Exceli asemel *„Comma Separated Value“* – CSV fail)
★★★★	Kasutatakse W3C loodud standardeid (RDF ja SPARQL)
★★★★★	Konteksti loomiseks on andmed lingitud teiste avaandmetega (Berners-Lee, 2006)

Tabel 1 – Viie tärni mudel

2.3 Lingitud avaandmete kasutus

Lingitud avaandmed on juba laialdaselt kasutusel mitmes valdkonnas, hõlmates järgmist kolme:

1. raamatukogudes:

- keskendutakse andmevahetusele ja ülemaailmse raamatukoguandmestiku loomisele;
- andmeid vahetatakse ning kasutatakse ühiselt raamatukoguväliste institutsioonidega;
- luuakse usaldust semantilise veebi kasvamises;
- hallatakse maailma kultuuripärandi informatsioonigraafi, mis on sellisel viisil usaldusväärne ja püsiv.

2. biomeditsiinis:
   - luuakse põhimõtteid ontoloogia ja sõnavara arendamiseks;
   - parandatakse andmete plahvatuslikku levikut;
   - luuakse koordineeritud ontoloogia, mis on loogiline ja koostalitlusvõimeline;
   - hakatakse toetuma täpsele bioloogilisele esitusele.
3. valitsuses:
   - taaskasutatakse avaliku sektori andmeid;
   - parandatakse siseprotsesse kasutades avaandmete poolt tekitatud uusi andmeid;
   - ühendatakse valitsuse andmed väliste andmetega.

Lingitud avaandmete kasutamine võimaldab pakkuda kvaliteetset teavet ja integreerida andmekogusid, segades oma andmed kolmanda osapoole informatsiooniga. Need rikastatud andmekogud võivad olla lingitud avaandmete portaalis või avaandmete andmeaida süsteemis, mis tagab parema otsuste langetamise, õnnetuste tagajärgede haldamise ja teadmiste juhtimise. Organisatsioonid saavutavad konkurentsieelise järgmiste võimaluste kaudu (Bauer & Kaltenböck, 2012):

- luues mitmeid andmeallikaid kasutavaid mätserdisi („*mashup*“);
- luues rakendusi, mis põhinevad reaalajaandmetele, vähendades replikeerimist.

3 Ehitisregister

Ehitisregister sisaldab kahte liiki andmeid:

- ehitise või selle osa aktuaalseid andmeid;
- dokumentide andmeid.

Ehitis võib olla hoone või rajatis. Ehitise identifitseerimiseks registris omistatakse sellele tarkvara poolt ehitisregistri kood.

3.1 Ligipääs tavakasutajale

Ehitisregistri kodulehel on autentimata kasutajal võimalik otsida ehitisi kolme otsingu alusel: aadressi otsing, katastritunnuse otsing ning ehitise otsing. Aadressiotsing jaguneb kaheks: klassifikaatoripõhiseks ning täisteksti otsinguks. Esimesel juhul tuleb kasutajal vastavalt aadressile valida maakond, omavalitsus, asustusüksus, väikekoht, liikluspind, nimi ja aadressinumber. Sõltuvalt aadressist ei ole kõik väljad alati täidetavad. Lisaks on võimalik klassifikaatoripõhises otsingus valida, kas soovitakse valida kehtivate aadresside või kõikide aadresside hulgast. Täisteksti otsingus tuleb määrata aadressi aktuaalsus valides kehtiva, ajaloos või kõikide aadresside hulgast. Seejärel tuleb sisestada aadress tekstina. Katastritunnuse otsingus tuleb sisestada katastritunnus või selle osa. Ehitise otsingus tuleb sisestata ehitisregistri kood või ehitise nimetus. Lisaks tuleb valida ehitise tüüp: hoone, rajatis. Kõikide tekstipõhiste otsingute korral tuleb valida otsingusõna asukoht. Valikuteks on „alguses“, „sisaldub“ ja „täpne otsing“. Vastavalt otsingutüübile kuvatakse leitud aadressil või katastritunnusel paiknevate ehitiste nimistu. Näidisotsingus (vt Ekraanipilt 3) kasutas autor aadressina „Narva mnt 25“. Valitud aadressil kuvatakse nimistu 12 ehitise kohta. Ehitisregistri koodile klikkides suunab veebileht valitud ehitise detailvaatele (vt Ekraanipilt 4). Autor valis ehitise ehitisregistrikoodiga 101024649, mille nimetus on „Peahoone Narva mnt 25“. Samale vaatele jõuab täpset ehitiseotsingut kasutades. Detailvaates kuvatakse järgnevad andmed: ehitise esmase kasutuselevõtu aasta, kavandatava kasutamise lõpetamise aeg, nimetus, staatus, energiamärgis, aadress, kasutamise otstarve ning üldised olulised tehnilised näitajad. Edasi on võimalik navigeerida materjalid, tehnosüsteemid, muud andmed, geomeetria, kinnistu, ehitise osad ning dokumendid vaatele.

Ekraanipilt 3 – Narva mnt 25 otsing

Ekraanipilt 4 – Ehitise detailvaade

Ehitisregistri andmed on tavakasutajale kättesaadavad, kuid ei ole masintöödeldavad ning ei pakuta salvestamisvõimalust.

3.2 Ligipääs üle X-tee

Riigi infosüsteemide andmevahetuskihi X-tee kaudu pakub Ehitisregister 17 teenust (Riigi Infosüsteemi haldussüsteem):

- andmeprobleemi_teatis - teatan andmeprobleemist (alampäring);
- dokumendid - isikuga seotud dokumendid Ehitisregistris;
- ehitised - ehitised katastriüksustel (alampäring);
- EN_EhitiseAndmed - E-Notar ehitise andmete päring;
- EN_EhitiseOsaAndmed - E-Notar ehitise osa andmete päring;
- EN_EhitiseOtsing - E-Notar ehitise otsing;
- EN_LisaBroneering - E-Notar lisa broneering;
- EN_OrgYldAndmed - E-Notar KOV-de kontaktandmed;
- EN_OtsiAadressiAdr - E-Notar aadressi otsing aadressi järgi (informatiivne);
- EN_OtsiAadressiAdrTxt - E-Notar aadressi otsing aadressi järgi teksti alusel;
- EN_OtsiAadressiIsk - E-Notar aadressi otsing isiku järgi;
- EN_OtsiAadressiKy - E-Notar aadressi otsing katastritunnuse järgi;
- EN_OtsiDokumenti - E-Notar dokumendi otsing;
- katastriyksusega_seotud_objektid - katastriüksusel asuvate objektide viimase muudatuse aeg;
- katastriyksuse_teatis - teatan katastriüksuse (alampäring);
- vallasvara - registreeritud vallasvara andmeid Ehitisregistris;
- vallasvara_piirangud - vallasvara piirangud (alampäring).

X-Tee kaudu andmete kasutamiseks tuleb sõlmida leping andmekogu haldajaga, kasutada turvaserverit ning luua SOAP („*Simple Object Access Protocol*“ - lihtne objektipöödusprotokoll) päringuid. Tavakasutajal enamasti sellised võimalused puuduvad.

3.3 Avaandmete lahendus

Avaandmete lahendusena asuvad Ehitisregistrist pärinevad avaandmed pilves. Andmed on struktureeritud ning allalaetavad. Andmed on masinloetavad ning agregeeritavad. Lisaks SPARQL päringute abil on kasutajal võimalik pärida vaid soovitud andmeid. Viimane mugavdab mätserdiste loomist.

MKM-i hanke raames luuakse näidisrakendus, mis kasutab Ehitisregistri avaandmete SPARQL päringuid. Teise andmekoguna tuleb kasutusele Kreedix OÜ poolt hallatava Inforegister.ee andmekogu teenused. Näidisrakenduse põhifunktsiooniks on kuvada firmade poolt okupeeritud ruumide pinna keskmist ruutmeetrite arvu valitud piirkonna kohta. Navigeerimine toimub Google Maps abil maakonnatasemelt kuni ehitiseosa tasemeni. Ehitisregistri andmekogu andmestikust saab kasuliku pinna spetsifikatsiooni ning Inforegister.ee pakub ettevõtete aadresside andmestikku. Andmekogude vahel luuakse seos kasutades Aadressiandmete süsteemi (ADS) standardseid aadresse. Näidisrakenduse arendus jääb käesoleva bakalaureusetöö skoobist välja.

4 Andmete avalikustamine

Andmete avalikustamiseni tuli läbida mitu sammu, mida projekti skoobis on jaotatud kolmeks põhietapiks:

- andmete publitseerimistööriista valimine ja funktsionaalsuse testimine;
- andmete laadimine publitseerimistööriista;
- andmete vastendamine („*mapping*“) RDF vaadeteks.

Lingitud andmete publitseerimistööriistad jagunevad kaheks: tööriistad, mis serveerivad salvestatud RDF kolmikute kogumikke ja tööriistad, mis pakuvad relatsioonilistele andmeallikatele lingitud andmete vaateid („*views*“). Kõik linkimise tööriistad toetavad URI ja RDF kirjeldusi, kusjuures osa pakub lisaks SPARQL päringute võimalust ning RDF tõmmiseid („*dumps*“) (Bizer, Heath, & Berners-Lee).

Ehitisregistri avaandmete publitseerimisel soovis projektijuht andmed salvestada relatsioonilise andmemudelina ning luua sellele RDF vaated. Lahenduses kasutatakse kõrgeimat lingituse taset ning võetakse kasutusele SPARQL päringu võimalused. Lisatingimuseks on tööriista ühilduvus PHP programmeerimiskeelega. Käesolevas peatükis võrdleb autor kahte tööriista: *D2R Server* ning *Virtuoso OpenSource Edition*.

4.1 D2RQ - D2R Server

D2RQ platvorm loob relatsioonilisest andmebaasist RDF graafid. Päringud tõlgitakse vastenduse alusel SQL keelde. Lahendus pakub RDF-põhist juurdepääsu ilma andmeid replikeerimata. D2RQ lingitud andmete võimalused (Bizer & Cyganiak, The D2RQ Platform – Accessing Relational Databases as Virtual RDF Graphs):

- lingitud andmete juurdepääs relatsioonilisele andmebaasile;
- RDF tõmmised;
- SPARQL päringud relatsioonilisele andmebaasile;
- ontoloogiate tugi;
- Apache Jena API (Java lingitud andmete raamistik) juurdepääs.

Toetatud andmebaasid:

- Oracle;

- MySQL;
- PostgreSQL;
- Microsoft SQL Server;
- HSQLDB;
- Interbase/Firebird;
- ODBC ja JDBC andmeallikad.

D2RQ avatud lähtekoodiga tarkvara on publitseeritud Apache litsentsiga. Lähtekood on kättesaadav „*GitHub*“ repositooriumis. Süsteemi nõudeks on Java 1.5 või uuem (Bizer & Cyganiak, Getting Started with D2RQ | The D2RQ Platform).

4.2 OpenLink - Virtuoso OpenSource Edition

Virtuoso on suure jõudlusega objekt-relatsiooniline SQL andmebaas, mis hõlmab endas transaktsioonide võimalust, SQL kompilaatorit, protseduuride keelt JAVA ning .NET toega, varukoopiate võimalust, SQL-99 tuge ja palju muud. Andmebaasimootor toetab RDF vaateid, mis tagab kiiruse. Andmebaas toetab ODBC, JDBC, ADO, .NET ning OLE/DB draivereid Windows, Linux ja MAC OS X operatsioonisüsteemidele. Virtuoso OpenSource lingitud andmete võimalused (OpenLink Software):

- lingitud andmete juurdepääs relatsioonilisele andmebaasile;
- RDF kolmikute üleslaadimine ja salvestamine;
- SPARQL päringud relatsioonilisele andmebaasile;
- SPARUL ehk SPARQL/Update päringud;
- ontoloogiate tugi.

Välised andmebaasid on toetatud ainult tasulisel versioonil:

- Firebird;
- DB2;
- Informix;
- Ingres;
- MySQL;
- Microsoft SQL Server;
- Oracle;
- PostgreSQL;

- Progress;
- Sybase;

Virtuoso avatud lähtekoodiga versioon on allalaetav OpenLink Software kodulehelt lähtekoodina või binaarkujul Linux, Unix ja Windowsi distributsioonidele (OpenLink Software).

4.3 Võrdlus ja valik

D2R ja Virtuoso pakuvad mõlemad mugavat võimalust andmete RDF vastenduseks olemasolevatele andmebaasidele. Virtuoso eeliseks D2RQ kõrval on sisene andmebaasimootor, mis toetab RDF vaateid ilma SQL päringuteks tõlkimata. Virtuoso sisemisele andmebaasimootorile on draiverid Windowsi, Linuxi ja MAC OS X operatsioonisüsteemidele. Mõlemad toetavad SPARQL päringuid. Virtuoso toetab uudse tehnoloogiana SPARUL päringuid, mille süntaks on tuletatud SPARQL keelest ning võimaldab graafe uuendada, luua ning kustutada. D2R lahendust kasutab DailyMed, pakkudes informatsiooni müüdavate ravimite kohta. DBtune.org kasutab „*MusicBrainz*“ muusika andmestiku linkimiseks samuti D2R platvormi, sisaldades üle 36 miljoni RDF kolmiku (MusicBrainz). Virtuosot kasutab üks suurimaid avaandmete lahendusi „*LOD Cloud Cache*“, mis puhverdab lingitud avaandmete ühises pilves olevaid andmeid. Teenus hõlmab üle 15.4 miljardi RDF kolmiku (W3 Consortium, 2011). Samuti kasutab Virtuoso lahendust DBpedia, mis eraldab Wikipedia.org lehelt struktureeritud sisu. Viimane omab kolmikuid üle 3.6 miljoni asja kohta (DBpedia.org).

	D2R Server	Virtuoso OpenSource
Andmebaasimootor	-	X
Andmebaasi draiver	-	Windows, Linux, MAC OS X
Lingitud andmete juurdepääs relatsioonilisele andmebaasile	X	X
RDF tõmmised	X	-
RDF üleslaadimine ja salvestamine	-	X
SPARQL päringud	X	X

SPARUL ehk SPARQL/Update päringud	-	X
Ontoloogiate tugi	X	X
Välisandmebaaside tugi	X	X (tasulisel versioonil)
Tööstuslikus kasutuses	DailyMed; DBtune.org;	LOD Cloud Cache; DBpedia

Tabel 2 - Publitseerimistööriistade võrdlus

Projektijuhi poolt osutus valituks Virtuoso OpenSource. Otsust mõjutasid järgmised asjaolud: tööriist on valideeritud suuri andmemahte kasutava DBpedia poolt, tööriist toetab ontoloogiaid, sisene andmebaasimootor on koos eri platvormide draiverite toega, RDF ja SPARQL tehnoloogia abil toetab viie tärniga tähistatud lingitud avaandmete taset.

4.4 Virtuoso OpenSource paigaldamine ja funktsionaalsuse testimine

Järgmisena paigaldas autor valitud publitseerimistööriista. Virtuoso hostiks sai valitud Debian 6 „*Squeeze*“ operatsioonisüsteem. Esmapaigaldus toimus Debiani paketihalduri „*apt*“ abil. Paigaldus käivitati ühe käsuga, paigalduse käigus tuli määrata Virtuoso administraatorite paroolid.

Süsteemi funktsionaalsust testides oli esimeseks eesmärgiks katsetada RDF vaadete vastenduse loomisvõimalust. Autor lõi kaks lihtsat võõrvõtmega seotud tabelit. Automaatne vastendamine toimub läbi kolme nupuvajutuse. Predikaatide nimed luuakse tabeli väljanimede järgi. Erinevate tabelite kirjed luuakse erinevate objektidena. Võõrvõtmete abil on viimased aga predikaadi abil siiski seotud. Keerulisemate vastenduste jaoks on võimalik redigeerida vastenduste definitsioone.

Edasisel funktsionaalsuse testimisel osutus probleemiks ODBC (*Open Database Connectivity* – avatud andmebaasipöördus) ühenduse loomine. Virtuoso dokumentatsioon ei aidanud probleemi lahendada ning autor oli sunnitud otsima alternatiivseid õpetusi. Abi oli http://fumi.me/2010/07/07/virtuosoontowiki/ veebilehest, mis kirjeldab Virtuoso OpenSource ning OntoWiki - vabavaralise semantilise viki paigaldust. OntoWiki on realiseeritud PHP programmeerimiskeeles ning kasutab Virtuoso andmebaasiga suhtlemiseks ODBC ühendust. Veebilehel kirjeldatud paigaldus osutus esialgsest kordades keerulisemaks. Autor alustas uut paigaldust värskest Debian 6 *„Squeeze“* installatsioonist. Paigaldamiseks tuli Virtuoso

paketid alla laadida, laetud paketid avada, lisada sõltuvused, muuta lähtekoodi formaati ning seejärel paketeerida tehtud muudatused. Seejärel oli võimalik paigaldada kohandatud Virtuoso OpenSource eksemplar. Paigalduse käigus tuli mõistagi sisestada administraatorite paroolid nagu lihtpaigalduseski.

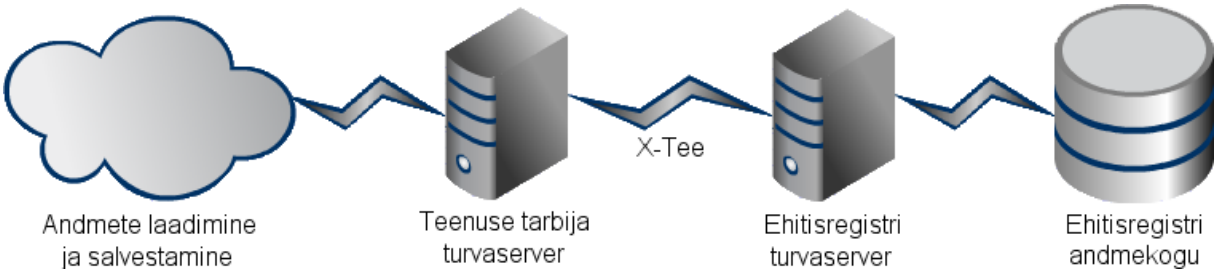
Järgmisena juhendab veebileht ODBC konfigureerimispakettide paigaldust. Peale viimase paigaldust tuleb seadistada andmeallika nimi (DSN - „*Data Source Name*“). PHP keeles ODBC ühenduse katsetamiseks tuleb paigaldada veel PHP-ODBC pakett. Seejärel on süsteem valmis PHP ühenduse loomiseks.

Paigaldamise hõlbustamiseks koostas autor Virtuoso OpenSource ODBC paigaldusjuhendi (vt Lisa 1, http://www.tlu.ee/~maehans/Bakalaureus/Virtuoso-OpenSource-ODBC-Paigaldusjuhend.pdf).

5 Andmete laadimise võimalused

Paljud riigi infosüsteemi kuuluvad registrid, kaasaarvatud Ehitisregister, pakuvad andmeteenuseid andmete vahetamiseks üle X-tee ning nende taaskasutamine võimaldaks keerulisemate registrite puhul luua kuluefektiivsemalt avaandmete publitseerimise lahendusi.

Tegemist on standardse X-tee skeemiga, kus päringud tehakse teenuse tarbija turvaserverile, mis suhtleb üle X-tee Ehitisregistri turvaserveriga, lubatud päringu korral edastab Ehitisregistri turvaserver päringu andmekogule. Vastus saadetakse mööda tuldud teed andmekogult Ehitisregistri turvaserverile, üle X-tee teenuse tarbija turvaserverile ning sealt omakorda päringu tegijale. (vt Joonis 1).



Joonis 1 – Andmed üle X-Tee

Avaandmete publitseerimiseks sobis Ehitisregistri andmeteenustest lahenduse loomise hetkel kõige paremini „*EN_EhitiseAndmed*“, kuna see väljastab enamiku avaandmeteks sobivatest andmetest ehitise kohta. Teenuse sisendiks on ehitise identifikaator. Päringute koostamiseks annab andmekogu haldaja registris olevate ehitiste identifikaatorid. Päringu vastus sisaldab isikustatud andmeid, mis ei kuulu avalikustamisele:

- omandi andmed;
- omaniku piirangud;
- vallasomandid;
- arestide ja keeldude andmed
- pandid.

RIA esindajatega kohtumisel toodi välja X-tee ja EN_EhitiseAndmed teenuse kaudu andmete laadimise negatiivsed aspektid:

- laadimisel tekib turvarisk (isikustatud andmed koonduvad);
- anonümiseerimine on andmete omaniku kontrolli alt väljas;

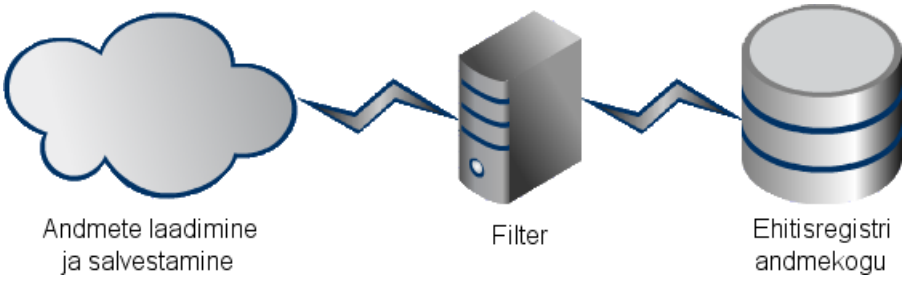
- X-tee kasutamine lisab käideldavusele üldkulu („*overhead*“);
- X-tee statistika moonutamine.

Alternatiivsete lahendustena, mis eemaldavad isikustatud andmed, pakuti välja andmekogule filtri loomist ning turvaserveri kodeerimisteenuse kasutamist.

5.1 Andmekogu filtreerimine

Esimene alternatiiv on päringu vastuse filtreerimine andmekogus. Filtreerides ei ole vaja kasutada turvalist andmekanalit, andmekogule lisatakse avalik teenus. Päring tehakse filtris asuvale teenusele, filter teeb päringu andmekogule ning eemaldab isikustatud väljad (vt Joonis 2). Filtreerimisega saavutataks järgmised positiivsed aspektid:

- isikustatud andmed eemaldatakse andmekogu juures;
- erasektorile ühine ligipääs.



Joonis 2 – Andmed läbi filtri

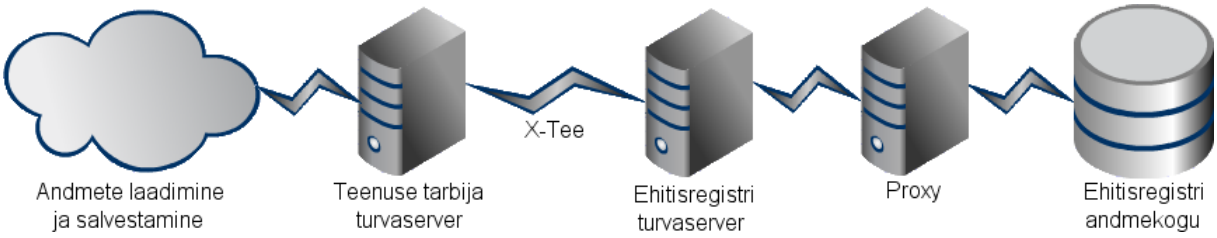
5.2 X-Tee kodeerimisteenuse kasutamine

Alates versioonist 5.0 pakuvad X-tee turvaserverid pseudonümiseerimise ("kodeerimise") teenust, et võimaldada anonüümsete agregaatanalüüside läbiviimist. Teenus kodeerib delikaatsed andmed päringuvastustes nii, et isikut pole võimalik identifitseerida, kuid samas on võimalik erinevatest andmekogudest saadud andmete põhjal koostada vajalik agregaat- ehk koondandmekogu (Laur, 2010).

Andmekogule loodaks proxy teenus, mis pärib andmeid esialgselt teenuselt. Proxy ülesandeks on lisada soovitud andmeväljadele kodeerimise vajadust tähistav märgis. Päringu korral tehakse pilvest päring uuele teenusele. Teenuse tarbija turvaserver edastab päringu Ehitisregistri turvaserverile, mis küsib andmeid proxy käest. Proxy teeb päringu andmekogule, andmekogu vastab proxy serverile, mis lisab kodeerimisteenuse märgise.

Ehitisregistri turvaserver pseudonümiseerib („kodeerib“) märgitud väljad ning edastab andmed teenuse tarbija turvaserverile. Sealt liigub vastus pilve (vt Joonis 3). Kodeerimisega saavutataks järgmised positiivsed aspektid:

- isikustatud andmed eemaldatakse andmekogu juures;
- teenus on kättesaadav teistele X-tee osalistele;
- mitme andmekogu pseudonümiseeritud andmeid saab konfidentsiaalsust kaotamata kokku panna.



Joonis 3 – Andmed läbi proxy ja üle X-tee

6 X-tee päringud, andmemudel ja salvestamine

6.1 Näidisteenus

Andmete laadimisskripti loomiseks lõi autor näidisteenuse. Viimase realiseerimiseks on mugav kasutada veebiteenuseid analüüsivat soapUI rakendust, mis võimaldab luua projekti veebiteenuse kirjeldamise keele failist (WSDL - „*Web Services Description Language*“). Rakendus loob WSDL-is kirjeldatud teenuste nimistu, genereerib näidispäringud ning soovi korral loob näidisteenuse.

WSDL-i allalaadimiseks suundus autor Riigi infosüsteemi haldussüsteemi (RIHA) veebilehel Ehitisregistri teenuste vaatele, salvestas EN_EhitiseAndmed teenuse kirjelduse ning lõi soapUI projekti. Peale näidisteenuse edukat käivitust tutvus autor PHP „*SoapClient()*“ klassiga, mis tegeleb veebiteenustega. Esimese õnnestunud päringu salvestas autor XML („*Extensible Markup Language*“) failina.

6.2 Andmemudel

Näidisteenuse vastuse ning sisekasutuseks mõeldud dokumendi „Ehitisregistri X-tee päringute kirjeldus“ alusel hakkas autor koostama andmete salvestamiseks andmemudelit. Mugavama RDF vastenduse eesmärgil sai andmemudelis andmeväljade nimedena kasutatud ontoloogia standardeid järgivaid väärtusi. Tabelite ja väljade nimed on selge tähendusega, tabelinimedes kasutatakse „*UpperCamelCase“* ning väljade nimedes „*lowerCamelCase*“ tekstivormingut. Esialgne andmemudel sisaldas 16 tabelit. Lõplikus andmemudelis (vt Joonis 4, http://www.tlu.ee/~maehans/Bakalaureus/Andmemudel.SQL) on lisaks kolm ADS tabelit, üks seoste ning veatöötluse tabel.

Joonis 4 - Ehitisregistri avaandmete andmemudel

6.3 Näidisvastused

Andmemudeli testimiseks ning salvestamisskripti loomiseks oli autor ning projektijuht huvitatud eri liiki ehitiste X-tee vastustest. Autori ülesandeks oli koostada etteantud ehitise identifikaatorite alusel laadimisskript, mis salvestaks päringute vastused XML failidena. Kohalikus süsteemis lahenduse testimiseks konfigureeris autor soapUI näidisteenuse võimalikult sarnaseks päris X-tee lahendusega. Konfiguratsioon erines ainult turvaserveri aadressi poolest.

Kuna laadimisega pidi tegelema kolmas osapool, siis tegevuse mugavdamiseks pakib laadimisskript salvestatud XML failid arhiiviks kokku ning kustutab XML failid. Arhiivi loomiseks kasutas autor PHP „*ZipArchive()*“ klassi. Lahenduse pakkis autor kokku ning laadis http://www.soatrader.com/ veebiserverisse. Lahenduse mugavamaks kasutamiseks koostas autor kolmandale osapoolele kasutusjuhendi, mis on mõeldud Debian 5 või 6 operatsioonisüsteemile. Juhendi käigus käsitletakse järgmiste vajalike pakettide „*unzip*“, „*apache2*“, „*php5*“ ning „*libapache2-mod-php5*“ paigaldust. Seejärel tuleb „*wget*“ abil allalaadida http://www.soatrader.com/ serverist arhiiv. Viimane lahti pakkida sobivasse kausta. Järgmisena tuleb muuta konfiguratsioonifaili ja muuta kausta kirjutamisõigusi. Seejärel on süsteem andmete laadimiseks valmis. Järgmise sammuna tuleb veebilehitsejas avada http://localhost/ehr.php aadress. Peale päringute loomist kuvatakse viide loodud arhiivile. Kirjutamisõiguste või SOAP päringute vigade korral kuvatakse vastavad veateated.

6.4 Salvestamine

Autor alustas andmete salvestamisskripti loomisega pärast soapUI näidisteenuse püstitamist. Esimeseks mureks oli string tüüpi XML vastuse parsimine PHP „*stdClass*“ tüüpi objektiks. Kolleegidelt nõu küsides, sain soovituse vältida PHP standardseid XML parsereid. Viimased jäävad hätta X-tee nimeruumide tõlgendamisega. Autor on eelnevalt kasutanud NuSOAP tööriista, mis hõlbustab SOAP päringute tegemist. X-tee päringu puhul ei ole seda mõistlik kasutada kuna tööriist genereerib automaatselt SOAP ümbriku („*SOAP-Envelope*“), mis erineb X-tee spetsifikatsioonist. Appi tuleb tööriist „*nusoap_parser()*“ XML parsimise klassiga. Peale klassi eksemplari loomist on võimalik küsida „*array*“ tüüpi „*soapresponse*“ muutujat. Katsetamisel toimis parsimine ilusti. Autori isiklikel eelistustel muudetakse massiiv „*stdClass*“ tüüpi objektiks. Hilisemalt andmeid salvestades tekkisid anomaaliad. Objektiks teisendamisel kadusid mõned andmeväljad. Peale pikemat veaotsingut jõudis autor tagasi

XML-i parsimiseni. NuSOAP parsimisel tagastatakse XML-is array tüüpi elementide atribuut „*SOAP-ENC*“ PHP massiivi elemendina. Lahendusena sai „*stdClass“* objektiks teisendamisel eemaldatud kõik vastava nimega massiivi elemendid.

Järgmise sammuna andmete salvestamise mugavdamiseks lõi autor „*DatabaseHelperODBC*“ klassi. Klassi loomist inspireeris Joomla! platvormi andmebaasiklassid. Loodud klass sisaldab järgmisi funktsioone:

- setQuery($query) – SQL päringu määramine;
- execute() – päringu käivitamine;
- loadObjectList() – tagastab päringu vastuse objektide massiivina;
- loadObject() – tagastab ühe objekti, kui vastus ei sisalda täpselt üht vastust tagastatakse veateade;
- quote($text) – sisestatud tekst ümbritsetakse ühekordsete jutumärkidega;
- nameQuote($text) – sisestatud tekst jaotatakse punkti kohalt eriosadeks, osad käivad läbi quote() funktsiooni ning siis liidetakse taas punkti abil. Eesmärgiks andmebaasi ja tabelinime jutumärkidega ümbritsemiseks;
- insertid() – tagastab viimati sisestatud primaarvõtme väärtuse;
- getColumns($table) - tagastab tabeli väljade massiivi;
- isColumnNumeric($table, $column) – sisene funktsioon kontrollimaks, kas väli on numbrilist tüüpi;
- getTables() – tagastab andmebaasitabelite massiivi;
- insertOrUpdateObject($table, &$object, $key = null) – funktsiooni sisendiks on tabelinimi, viit baasi laetavale objektile ning valikuliselt sisestatav primaatvõtme väljanimi. Funktsioon koostab Virtuosole omase *„INSERT REPLACING“* päringu, mis olemasoleva primaarvõtme korral uuendab kirjet. Päring pannakse kokku sisestatud objekti alusel. Objekti muutujanimed võetakse päringus tabeli andmeväljade nimeks, muutujate väärtused vastavate andmeväljadele sisestatavateks väärtusteks.
- insertObject($table, &$object, $key = null) – funktsioon teeb „*INSERT*“ päringu kasutades eelmise funktsiooniga samu põhimõtteid. Korduva primaarvõtme korral tagastatakse veateade;
- updateObject($table, &$object, $key, $doNotUpdate = array()) – funktsioon loob „*UPDATE*“ päringu, kasutades sisenditena tabelinime, sisestatava objekti viita,

primaarvõtme väljanime ning lisaks on võimalik massiivina sisestada väljadenimed, mida ei soovita uuendada. Primaarvõtme puudumisel tagastatakse veateade;

- printHTMLTable() – kuvab määratud päringu tulemi HTML tabelina.

Andmebaasiklass tekitas valmiduse andmete salvestuseks. Autor täiendas soapUI näidisvastust eelmainitud sisekasutuseks mõeldud dokumendi abil. Dokumendis on andmeväljadele määratud näidisandmed. Seejärel oli võimalik katsetada andmebaasi salvestust. Peale andmeid sisaldava vastuse objektiks parsimist tuli objekt viia kooskõlla andmemudeliga. Selleks automatiseeritud võimalus puudus, kuna andmemudel sai loodud vastavalt RDF vaadete mugavaks loomiseks. Autor lõi laadimisskripti osana „*EHRResponseSaver"* klassi, mis tegeleb vastendusega. Peale vastendusest toimub „*DatabaseHelperODBC*" klassi „*insertOrUpdateObject*" funktsiooni abil sisestus andmebaasi. Andmetega täidetud näidispäringu salvestus oli edukas.

Peale punktis **Error! Reference source not found.** käsitletud eri liiki ehitiste vastuste laekumist uuris autor soapUI võimalusi lähemalt. Näidisteenuse vastust saab juhtida skriptimiskeelte „*Groovy Script Library*" ning „*JavaScript*" abil. Autor otsustas esimese kasuks ning lõi skripti, mis leiab päringust ehitise identifikaatori ning selle alusel vastab päringule failisüsteemis oleva XML faili sisuga, mille nimes sisaldub ehitise identifikaator.

Reaalsete vastuste salvestamisel ilmnesid tõrked. Veaotsimisel jõudis autor arusaamiseni, millele oli eelnevalt viidanud ka soapUI näidisvastus, mis sisaldades elementide juures kommentaare sisuga „*optional*" ehk kõik väljad ei ole kohustuslikud. Massiivi tüüpi andmete salvestamisel ei tekkinud probleemi kuna kasutatakse „*foreach*" lahendust. Kui massiiv puudub või on tühi siis vastavaid andmebaasi salvestusi ei käivitata. Tavatüüpi sisestusel, aga ei kontrollitud, kas vastuses otsitav element eksisteerib. Vealahendusena kasutas autor valiklauset („*if-clause*") ning „*sizeof()*" funktsiooni.

6.5 Aadressiandmete süsteemiga sidumine

Ehitisregistri andmetes olevad aadressiandmed ei ole ADS kujul. „*Aadresstekstina"* väli on viidud ADS kujule kuid vastuses pakutavad aadressitasemete klassifikaatorid ei ühti. Edasiste andmete mugavama kasutamise ning hanke raames loodavas näidisrakenduses kasutatava

teise andmekoguga parema ühildumise eesmärgil avalikustatakse Ehitisregistri lahenduse juures kolm ADS-i tabelit (vt Joonis 4):

- ADS.Tasandid – aadressi komponendid jagunevad kaheksaks tasandiks (maakond, omavalitsus, asustusüksus, väikekoht, liikluspind, nimetus, aadressinumber, korterinumber);
- ADS.AadressiKomponendid – sisaldab eritasemete infot;
- ADS.Aadress – sisaldab viiteid erinevate tasemete aadressikomponentidele, lähiaadressi, täisaadressi, koordinaate ning olekut.

6.5.1 Aadressiotsing

Aadressi otsimisel alustas autor komponentide otsingust. Ehitisregistri „*aadressitase1*“ ning „*aadressitase2*“ väärtused üldjuhul kattuvad „*AadressiKomponendid“* tabelis olevate kirjete primaarvõtmega. Samas leidub lahkhelisid „*aadressitase2*“ puhul. X-tee vastuses oleva Tallinna linna „*aadressitase2“* väärtuseks on 784, aga ADS.AadressiKomponendid tabelis Tallinna linna primaarvõtme väärtus on 3075189. Lisaks primaarvõtmele on tabelis „*kood*“ väli, millelt võib samuti otsida X-tee vastusest saadud aadressitaset. Kolmandat aadressi komponenti otsib lahendus juhul, kui vastusena saadud ehitise aadressis on rohkem, kui kolm aadressitaset. Viimasena otsitakse „*aadresstekstina“* välja viimast osa. Mainitud väli jaotatakse komakohtade alusel eri osadeks ning tekstist eemaldatakse kõik numbrilised sümbolid. Seejärel tehakse otsing ADS.AadressiKomponendid tabelile, kus otsitakse „*nimetus*“ või „*nimetusliigiga*“ väljaga ühtivat osateksti. Viimasele järgneb päring ADS-i Aadress tabelile. Päringus kasutatakse leitud komponentide identifikaatoreid ning otsitakse aadressi viimast osa seekord koos numbriliste sümbolitega. Otsingul aadressidele oleku piirangut ei rakendatud. Selle tulemusena sai autor 54-st näidisvastuses olevast aadressist kõigile vaste.

Ehitisregistri aadressi „*aadresstekstina*“ väli on viidud ADS süsteemi kujule. Selle ajendil lähenes autor aadressiotsingule ka alternatiivse ideega. ADS Aadress tabeli „*taisaadress*“ väljalt „*aadresstekstina*“ väärtust otsides leitakse 52-le aadressile vasted. Valitud meetod osutus kiiremaks ning seetõttu tehakse viimasena kirjeldatud otsing esimesena. Kui otsingule ei leita vastet sooritatakse otsing kasutades esimesena kirjeldatud lahendust.

6.5.2 Ehitiseosa aadressiotsing

Aadressiotsinguga leitakse ehitisele ADS aadress. Näidisrakenduse loomiseks soovitakse leida ehitistes olevate ruumide aadress. Ehitisregistri vastuses ehitiseosadel aadressiga otsene seos puudub. Ehitiseosal on väli „*tahis*“, mille väärtuseks ehitiseosa ruumi tähis. Autor jätkas ehitiseosa aadressi otsingut kasutades ehitisele leitud aadresse. Kui ehitiseosal eksisteerib „*tahis*“ ning ehitisele on leitud ADS aadress otsitakse ehitiseosale aadressi. Otsingus kasutatakse leitud aadressi komponendiidentifikaatoreid ning lahiaadressi väärtusele lisatakse sufiksina ehitiseosa tähis ning sooritatakse otsing. 86-le otsingule leiti 80 juhul vasted. Vasteta jäi neljal juhul eramajade päring, kus ehitisel on üks ehitiseosa, millel on tähis.

6.6 Ehitise identifikaatorid ja veatöötlus

Ehitisregistri andmekogus on üle 900 000 erineva ehitise. Salvestamisskripti sisendiks on ehitise identifikaator, mille alusel päring koostatakse. Ehitise identifikaatorite sisselaadimise, ning salvestamise ebaõnnestumise haldamiseks tuleb luua komponent. Esialgse lahendusena nägi autor ehitise identifikaatorite sisselaadimiseks failist lugemise võimalust. Ebaõnnestunud päringute korral salvestataks tekstifaili ehitise identifikaatorid. Vääramatu jõu tõttu võib andmete laadimine sootuks katkeda ning seetõttu tuleks samuti hoida viimati töödeldud ehitise identifikaatorit. Viimase hoidmine tekitaks kolmanda faili. Selline lahendus tekitas ebakindluse ning autor hakkas otsima alternatiivset lahendust.

Alternatiivse lahendusena luuakse Virtuoso andmebaasi „*_TEMP_EHITIS*“ tabel, mis sisaldab järgmisi väljasid:

- ehitiseIdentifikaator – ehitisregistri haldaja poolt antud kõik ehitise identifikaatorid;
- staatus – 0-töötlemata, vaikeväärtus, 1-edukalt salvestatud, -1-salvestamisel tekkis tõrge;
- veatekst – sisaldab väärtust ainult -1 staatuse korral;
- sekundid – ehitise identifikaatori töötlemisele kulunud aeg;
- muudetud – kirje muutmise ajahetk.

Ehitise identifikaatorite andmebaasist pärimiseks ning veatöötluse haldamiseks koostas autor „*EhitiseIDHandler“* klassi, mille ülesandeks on küsida kirjeldatud tabelist X arv töötlemata ehitiseIdentifikaatoreid. Tegevust sooritatakse „*while*“ tsüklis, kontrollides vastusena saadud

kirjete arvu. Kui tingimus on tõene jätkatakse ehitiste töötlemisega. Luuakse tabelit uuendav ajutine objekt, millele koheselt määratakse ehitiseIdentifikaator, staatus -1 ning muudetud väärtuseks SQL funktsioon NOW(). Jätkatakse X-tee päringuga, mille tühja vastuse korral määratakse objektile veateade. Kui päring on edukas, edastatakse andmed „*EHRResponseSaver“* klassile. Salvestamise õnnestumisel määratakse staatuseks 1, vastand olukorras määratakse veateade. Objektile määratakse ehitise töötlemisele kulunud aeg ning „*DatabaseHelperODBC*“ klassi abil uuendatakse andmebaasi kirje. Veatöötluse lõpetamisega oli salvestamisskript valminud ning oli võimalik jätkata järgmise etapiga.
(vt http://www.tlu.ee/~maehans/Bakalaureus/Salvestamine.RAR).

7 Andmete vastendamine RDF vaadeteks

Lihtsamate vastenduste loomine on Virtuoso OpenSource haldusliideses tehtud väga mugavaks. Tuleb suunduda „*RDF*“ menüüst „*RDF Views*“ vaatesse. Valida tabelid ning vajutada „*Generate & Publish*“ nuppu. Sellise lühiprotseduuri abil on tabelid vastendatud ning kättesaadavad SPARQL päringute kaudu.

Virtuoso kasutab eelnevalt räägitud URI-de asemel IRI-si („*Internationalized Resource Identifier“*). IRI on URI üldistus, peamiseks erinevuseks on see, et URI kasutab ASCII märgistikku, IRI aga Unicode märgistikku. Viimane võimaldab kasutada hiina, jaapani, korea ning kirillitsa märke.

7.1 Virtuoso seadistus

Lihtsate vastenduste katsetuste tulemusena mõistis autor, et Virtuoso tuleb ümber konfigureerida. Vastendatud RDF vaadetes kajastus mitmel kohal IRI alguses „*localhost:8890*“. Autor oli huvitatud, et IRI-s kasutatakse serveri nime, tulevikus veebiaadressi, kus Virtuoso eksemplar paikneb.

Publitseerimistööriista on võimalik konfigureerida kasutades haldustööriista või muutes konfiguratsioonifaili. Autor valis sätete muutmiseks teise variandi. Virtuoso konfiguratsioon asub /etc/virtuoso-opensource-6.1/ kaustas virtuoso.ini failis. Vaikeväärtusena töötab Virtuoso HTTP server 8890 pordil. HTTP serveri tüüpilisele 80 pordile suunamiseks tuleb HTTPServer sektsioonist ServerPort väärtus asendada. Peale konfiguratsiooni salvestamist ning Virtuoso taaskäivitamist vastas Virtuoso 80 pordil.

Autor katsetas uuesti vastenduste loomist, mille tulemusena IRI-d sisaldasid endiselt „*localhost:8890*“ aadressi. Konfiguratsiooni edasi uurides leidis autor „URIQA“ sektsiooni, mille muutuja „*DefualtHost*“ väärtuseks oli „*localhost:8890*“. Autor muutis väärtust ning taaskäivitas jällegi Virtuoso. Peale viimast konfiguratsiooni muudatust toimis vastendus nii nagu autor oli soovinud. IRI-de aadressides oli soovitud serverinimi.

Konfiguratsiooni edasi uurides muutis autor ka SPARQL sektsioonis olevaid muutujaid. SPARQL päringu vaates on võimalik määrata vaikepäringut ning -graafi, millele otsing

tehakse. Viimast on võimalik konfiguratsioonis muuta „*DefaultGraph*“ muutuja alt. Vaikeväärtusena on muutuja väärtuseks „htttp://localhost:8890/dataspace“ ning seadistus on väljakommenteeritud. Kuna käesolevas projektis avatakse ehitisregistri andmed, kaasproduktiks on ADS andmestik, siis graafi väärtuseks märgiti EHR-i andmestik. Lisaks määras autor vaikepäringule („*DefaultQuery*“) uue väärtuse (vt Koodinäide 2). Päringu tulemusena tagastatakse kõik subjektid, predikaadid ning objektid EHRi andmestikust. Lisana on päringus defineeritud 10 vastuseline piirang. Nii on esmakatsetused kiiremad ning nõuab vähem serveri poolset ressurssi.

```
SELECT * WHERE {?subject ?predicate ?object} LIMIT 10
```

Koodinäide 2 – SPARQL vaikepäring

7.2 ADS-i ja EHR-i vastendamine

Automaatselt vastendades luuakse kahe andmekogu olemid ühte ontoloogiasse. Andmekogusid eraldi vastendades ei teki olemite vahelisi seoseid. Autor otsis vastendust, kus eri andmekogu olemid luuakse erinevatesse ontoloogiatesse, kuid säiliksid seosed. Sobiliku lahenduse leidmiseks hakkas autor katsetama eriviisilisi vastendusi. Virtuoso haldab vastendust Virtuoso poolt defineeritud meta-skeemikeele (MSL „*Meta Schema Language*“) abil, mis on SPARQL keele laiendus. Selle abil luuakse tabelitest IRI klassid ning veergudest IRI-d (OpenLink Software).

Autor lõi vastendusfailid kasutades haldustööriista „*RDF Views*“ vaates olevat „*Generate via Wizard*“ meetodit. Eelnevalt tuli märkida kõik soovitud tabelid ning soovitud ontoloogia nimi. Autor valis ontoloogia nimeks EHR, tabelitena EHR ning ADS tabelid, et loodaks ka andmemudelite vahelised seosed. Järgmisest vaatest „*Prepare to Execute*“ vajutades jõutakse vaatesse, kus kuvatakse vastenduse ning ontoloogia definitsioon. Kuna autor valis ontoloogia nimeks EHR tuli ADSi tabelite definitsioone ümber kirjutada ADSi nimele. Esialgu muutis autor lihtsamad nimeruumid ning hiljem võõrvõtmete seosed. Autor tegi koopa tehtud muudatustest ning vajutas „*Apply Changes*“. Seejärel leht uuendas ning võis vajutada „*Execute*“. Peale veebiserveri mõningast mõtlemist kuvati leht, kus näidati teadet vastenduse õnnestumise kohta. Lisaks kuvas leht viited mõnele EHR IRI-le ja ADS ontoloogiale. EHRi andmestik ja ontoloogia olid kättesaadavad, samuti ADSi andmestik, kuid viimase ontoloogia ei olnud kättesaadav.

Autor jätkas vastenduse loomist katse-eksituse meetodil. Muudetud definitsioonides oli ADS definitsioonid EHRi omades eespool. Autor muutis järjekorda ja lõi mitmeid eri kombinatsioone, kuid siiski oli tulem sama. Ühe katsetuse tulemusena proovis autor esialgselt määratud ontoloogia nimena ADS-i. Selle tulemusena ei kuvatud EHR-i ontoloogiat. Autor tegi järelduse, et vastendused tuleb eraldi luua. Eraldi vastenduste loomisel tekkisid mõlema andmekogu ontoloogiad ning andmed olid samuti kättesaadavad. Kõige olulisemana eksisteerisid ka andmemudelite vahelised seosed.

Virtuoso automaatse vastenduse abil loob võõrvõtme väljale kaks ontoloogia predikaati. Esimese väärtus on IRI teisele objektile. Teise väärtus on tabeli väljal olev väärtus. Esimest tüüpi predikaatide nimed luuakse automaatselt ning viimaseid tuli kohandada. Autor jättis teist tüüpi tabeliväljade vastenduse välja. Peale viimaseid muudatusi oli vastendus valmis kasutamiseks (vt http://www.tlu.ee/~maehans/Bakalaureus/Vastendus.RAR).

7.3 Jõudlus

Virtuoso jõudluse testimiseks tegi autor kuute tüüpi eripäringuid kasutades kolme eri meetodit. Esimeseks meetodiks on Virtuoso host masinas käsureakliendiga „*isql-vt*“ SQL päringute tegemine. Teiseks on sama kliendi kasutamine, kuid seekord sooritatakse SPARQL päringuid. Kolmanda meetodina kasutas autor Virtuoso SPARQL veebiteenust. Viimane ei kuva kulunud aega, mille tõttu autor sooritas päringuid kasutades PHP programmeerimiskeelt ning cURL funktsiooni andmete pärimiseks. Autor sooritas kõiki päringuid 10 korda ning arvutas kulunud aja keskmise. Tabelis on kirjeldatud päringute väljundid ning vastavat meetodit kasutades kulunud aeg millisekundites (vt Tabel 3). Autor rõhutab, et Virtuoso on paigaldatud virtuaalmasinasse ning päringu sooritamiseks kuluv aeg võib erineda reaalsest süsteemist.

Päring	SQL	SPARQL	PHP SPARQL
EHR.Ehitis tabeli ühe kirje kõik väljad	4,1	1,9	5,4
EHR.Ehitis tabeli ühe kirje „*kasulikpind*“ välja päring	0,3	0,3	4,1
EHR.Ehitis tabeli kõikide kirjete „*kasulikupind*“ välja summa	0,5	0,7	3,5
EHR.Ehitis seotud EHR.EhitiseOsa tabeliga, tulemis	1,1	4,2	8,8

primaarvõtme			
ADS.AadressiKomponendid tabeli ühe kirje kõik väljad	0,4	0,9	5,9
ADS.AadressiKomponendid tabeli kirjete loendamine	100,6	4632,5	4784,2

Tabel 3 - Päringute võrdlused

Autor koostas parima tulemi saavutamiseks päringud EHR-i tabelitele, kuhu andmestikku ei ole veel laetud ning ADS.AadressiKomponendid tabelile, kus on ligikaudu 1,2 miljonit kirjet. Kasutades „*isql-vt*“ tööriista SQL ja SPARQL-i päringute suhteline erinevus ei ole märkimisväärne väljaarvatud ADS.AadressiKomponendid tabelil agregaatfunktsiooni „*count()*“ kasutades. Viimasel korral on kestuse erinevus ligikaudu 46 kordne. Kui agregaatfunktsioon välja arvata kinnitab tulemus käesoleva töö 4.2 punktis kirjeldatud kiireid RDF vaateid.

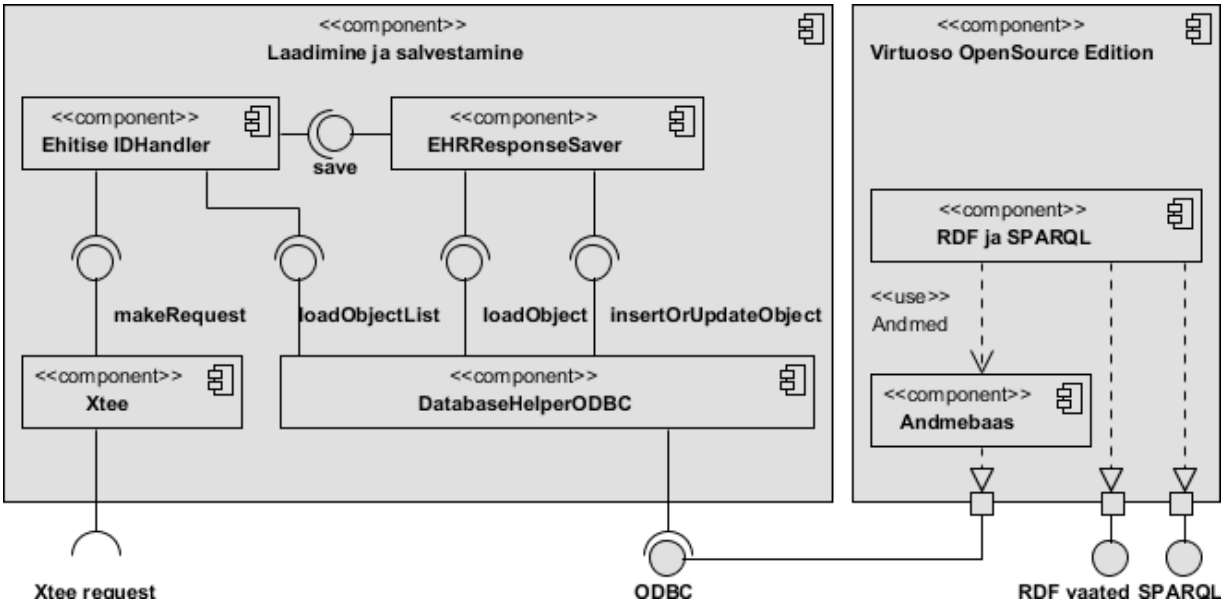
Virtuoso SPARQL veebiteenuse ja PHP kaudu tehtud päringud „*isql-vt*“ SPARQL päringutest kahe kuni kümne kordse erinevusega. Siin tuleb arvesse võtta mitmeid asjaolusid: PHP päring, veebiteenuse üldkulu ning võrgu latentsus.

8 Lahenduse struktuur

Andmete laadimise struktuuri kirjeldamiseks on autor kasutanud 4+1 arhitektuurist pärinevat arendusvaadet ning füüsilist vaadet. Arendusvaatest on välja toodud komponendiskeem. (vt Joonis 5). Skeemil on autori poolt loodud andmete laadimise ja salvestamiskomponendi ning Virtuoso alamkomponendid:

- EhitiseIDHandler – Komponent tegeleb ehitiste identifikaatorite ja veatöötlusega;
- Xtee – Komponent tegeleb X-tee päringutega;
- EHRResponseSaver – Komponent tegeleb päringu vastuse andmebaasi objektidele vastendamisega ning ADS aadresside leidmisega;
- DatabaseHelperODBC – Tegeleb andmebaasi ühenduse ja päringutega;
- Andmebaas – Hoiab andmeid;
- RDF ja SPARQL – Tegeleb andmebaasi vastendamisega.

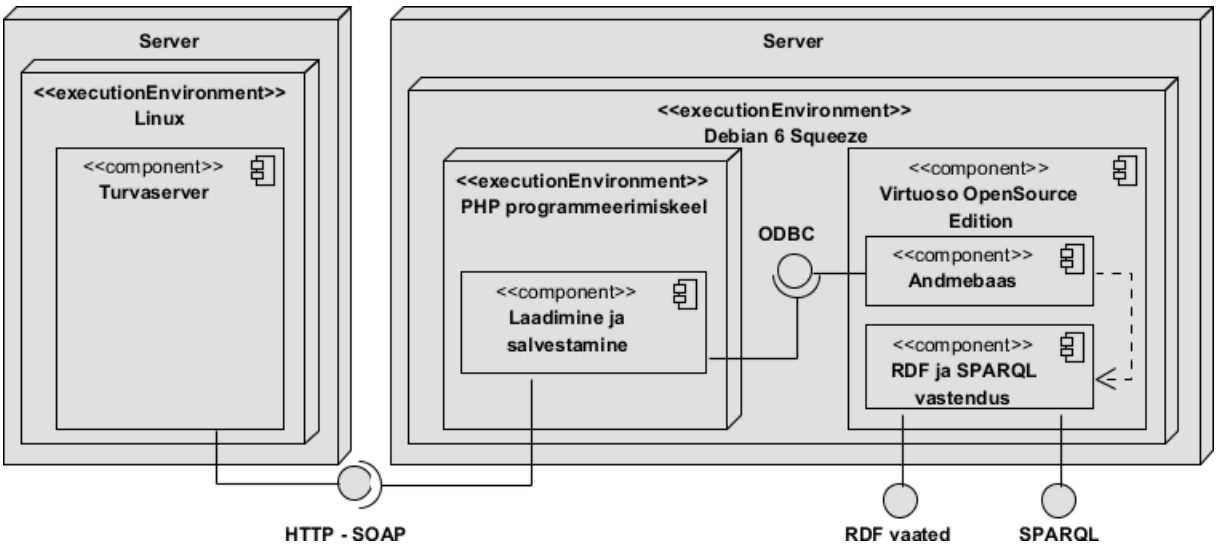
Teine oluline osa on Virtuoso OpenSource Edition-i andmebaasi alamkomponent. DatabaseHelperODBC suhtleb ODBC kaudu mainitud komponendiga.



Joonis 5 – Komponendiskeem

Füüsilisest vaatest on kajastatud levitusskeem (vt Joonis 6). Käesoleval skeemil on kajastatud autori poolt loodud lahendus, kus serverisse on paigaldatud operatsioonisüsteem Debian 6 „*Squeeze*“, kuhu on omakorda paigaldatud Virtuoso OpenSource Edition, PHP ning ODBC

paketid. PHP keeles realiseeritud laadimise ja salvestamisekomponent teeb üle HTTP ning SOAP protokolli päringud Linuxi serverile, kuhu on paigaldatud X-tee turvaserver. Saadud vastused salvestab komponent ODBC andmebaasiliidest kasutades Virtuoso andmebaasi. Viimane pakub lisaks RDF vaateid ning SPARQL päringuid. Kirjeldatud arhitektuuriga lahendus toimib ning on testitud näidisandmetega.



Joonis 6 – Levitusskeem

Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli välja selgitada parim lahendus andmete avalikustamiseks. Teoreetilises osas kirjeldas autor avaandmete mõistet, olemust. Lisaks tõi autor välja kasu saavutamiseks potentsiaalsed kohad ning juba toimivad näited. Töö jätkus lingitud andmete ja tehnoloogia kirjeldamisega. Seejärel jõuti lingitud avaandmeteni, mis kujul avaldatakse ka Ehitisregistri andmed.

Projekti skoobis jaotati praktiline lahendus kolme põhietappi: publitseerimistööriista valimine, andmete laadimine ja andmete viimine lingitud avaandmete kujule. Esimese asjana tuli valida andmete avaldamiseks mõeldud publitseerimistööriist. Autor võrdles kahte tööriista ning sisese andmebaasimootori ning madalalt tasemelt toetatud lingitud andmete tehnoloogiate tõttu osutus valituks Virtuoso OpenSource Edition.

Andmete laadimiseks võrreldi kolme lahendust. Esmalt standardne X-tee skeem, kus päringud tehakse turvaserverite kaudu. Esimese alternatiivina käsitleti andmekogu juures teenuse filtreerimist, millega oleks loodud andmekogule uus avalik teenus. Teise alternatiivina oleks kasutatud turvaserveri pseudonümiseerimise teenust, mis kodeeriks delikaatsed andmed päringuvastustes nii, et isikut ei oleks võimalik identifitseerida. Soovitud väljadele oleks lisatud proxy teenuse abil pseudonümiseerimist vajav märgis.

Andmete laadimisskripti loomiseks seadis autor näidisteenuse, mis imiteeris X-tee teenust. Selle abil osutus võimalikuks testida X-tee päringuid. Viimase õnnestumisel lõi autor andmemudeli, mida käesolevas bakalaureusetöös illustreeris ERD skeem. Andmemudeli testimiseks ning andmebaasi salvestamisskripti loomiseks kasutas autor eri liiki ehitiste X-tee vastuseid. X-tee vastused edastas projektijuhile andmekogu esindaja. Vastused erinesid näidisteenuse omast ning vastuste abil sai salvestamisskript parandatud. Aadressiandmete süsteemiga sidumise eesmärgil hakkas salvestamisskript tegelema ka Ehitisregistri poolt saadetud aadressile ADS-i aadressi vaste otsimisega. Ehitiste identifikaatorite ning veatöötlusega lahendamiseks kasutas autor andmebaasi tabelit, milles hoitakse kõiki ehitisregistri ehitiste identifikaatoreid ning vastavalt päringu õnnestumisele salvestatakse märge andmebaasi.

Sellele järgnes andmete lingitud kujule viimine. Publitseerimistööriistas hoitakse andmeid relatsioonilises andmebaasis, mistõttu lingitud avaandmete kujule viimiseks tegi autor vastenduse andmebaasi tabelitele. Enne viimase alustamist tuli aga konfigureerida publitseerimistööriist. Andmebaasi vastenduse loomisel tekkis mõningaid komplikatsioone. Peale korduvate katsete ning muudatuste tagajärjel jõudis autor lahenduseni. Peale edukat vastendust tegi autor jõudlusteste. Testid mõõtsid SQL, SPARQL ja SPARQL veebiteenuse ja PHP kaudu tehtud päringutele kuluvat aega. Peale agregaatfunktsiooni SQL ja SPARQL päringutele kulus ligikaudu võrdväärne ajahulk. PHP kaudu loodud päringutele kulus rohkem aega, mis võib sõltuda PHP päringutest, SPARQL veebiteenuse üldkulust ja võrgu latentsusest.

Viimases osas kirjeldas autor valminud struktuuri, mida illustreeris komponendi- ja levitusskeem. Komponendiskeemil on autori loodud komponendid ja omavahelised suhtluskanalid. Levitusskeemil on kajastatud lahenduse poolsed serverid ning serverite ja komponentide vahelised suhtluskanalid. Käesoleva töö lõpuks valmis lahendus, mis on testitud näidisandmetega, kuid bakalaureusetöö esitamistähtajaks ei ole loodud rakendus lõplikult paigaldatud.

Summary

Publishing Open Government Data. The Case of Estonian Register of Buildings

The aim of the present thesis is to find out which is the best solution for the register to publish open government data. So far Estonia has no experience in publishing open government data using the Data Exchange Layer X-Road and therefore it sets a precedent.

The fact that it has been stated in the Government of the Republic Action Programme 2011-2015 in the chapter "From E-State to I-State" that it is essential to change the data of the government and the local governments into machine-readable public data only confirms the necessity of the topic. In addition, the practical part of the thesis is the realization of the pilot project titled „OpenData in Cloud" which is one of the public procurements of the Ministry of Economic Affairs and Communications. The author of the present paper works as a programmer in the organization implementing the above-mentioned project. The data of the Estonian Register of Buildings was chosen for publishing by the author in cooperation with the project manager.

In the scope of the project the practical solution was divided into three basic stages: choosing the publishing tool, loading the data and putting it into the Linked OpenData form. The first task was to choose the publishing tool. The author was contemplating between two different tools, but due to the internal database engine and the low-level format technology in Linked Data the Virtuoso OpenSource Edition was chosen for research.

In order to download the data three different solutions were compared. Firstly the standard X-Road model was observed, where the searches are done through the Security servers. As the first alternative the filtering of the service was considered, which would have created a new public service for the database. The second alternative would have been to use the pseudonymization service of the Security server, which would encode the sensitive information in a way that the person could not be identified. Using the proxy service the tag could be added to the areas requiring pseudonymization. The creation of the alternative solutions would have required additional resources outside the public procurement and

therefore, the representatives of the Ministry of Economic Affairs and Communications and the Estonian Register of Buildings decided to choose the already existing X-Road solution.

For the creation of the data loading script the author produced a Mock Service, which is an imitation of the X-Road service. It gave the possibility to test the X-Road requests. After that being a success the author generated a model, which was introduced in the present thesis with the ERD model. In order to test the model and to create the loading script the author used the X-Road answers of different types of buildings. The X-Road answers were presented to the project manager by the representative of the database. The answers differed from the ones on the Mock Service model and the answers helped correct the saving script. As the Address Data System was attached to it, therefore, the saving script also started to find matches to the Address Data System. To find solutions for the building identificators and error management the author used the database table which contains all of the identificators of the buildings and according to the success of the search the information was saved into the database.

Taking the data into the Linked OpenData form was the following step. The publishing tool keeps the data in relational database, therefore, to take the data into the form of the Linked Open Data the author used mapping with the database tables. Before starting the last one, the publishing tool needed configuration. Also some complications occurred during the mapping process. After several attempts and changes the author managed to find solution. After author had successfully created mappings, performance tests were created. Tests measured time spent for making SQL, SPARQL and SPARQL through PHP queries. Except aggregate function SQL and SPARQL queries take nearly a same amount of time. SPARQL through PHP is taking more time which could depend on PHP query, SPARQL Endpoint overhead and network latency.

In the final part of the thesis the author describes the designed structure, which is illustrated by the component diagram and the deployment diagram. The component diagram shows the author´s components and communication channels. The deployment diagram shows the servers and the communication channels between servers and components. As a result of the research a solution was created, which was tested with Mock Service.

Kasutatud kirjandus

Bauer, F., & Kaltenböck, M. (2012). *Linked Open Data: The Essentials.* Vienna, Austria: edition mono/monochrom.

Berners-Lee, T. (27. 07 2006. a.). *Linked Data - Design Issues.* Kasutamise kuupäev: 26. 03 2012. a., allikas World Wide Web Consortium (W3C): http://www.w3.org/DesignIssues/LinkedData

Bizer, C. (09. 11 2009. a.). *wiki.dbpedia.org : Architecture*. Kasutamise kuupäev: 26. 03 2012. a., allikas wiki.dbpedia.org: http://wiki.dbpedia.org/Architecture

Bizer, C., & Cyganiak, R. (kuupäev puudub). *Getting Started with D2RQ | The D2RQ Platform*. Kasutamise kuupäev: 26. 03 2012. a., allikas The D2RQ Platform – Accessing Relational Databases as Virtual RDF Graphs: http://d2rq.org/getting-started

Bizer, C., & Cyganiak, R. (kuupäev puudub). *The D2RQ Platform – Accessing Relational Databases as Virtual RDF Graphs*. Kasutamise kuupäev: 26. 03 2012. a., allikas The D2RQ Platform – Accessing Relational Databases as Virtual RDF Graphs: http://d2rq.org/

Bizer, C., & Cyganiak, R. (kuupäev puudub). *The D2RQ Platform – Accessing Relational Databases as Virtual RDF Graphs.* Kasutamise kuupäev: 26. 03 2012. a., allikas The D2RQ Platform – Accessing Relational Databases as Virtual RDF Graphs: http://d2rq.org/

Bizer, C., Heath, T., & Berners-Lee, T. (kuupäev puudub). *Linked Data - The Story So Far.* Kasutamise kuupäev: 25. 03 2012. a., allikas Tom Heath: http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf

DBpedia.org. (kuupäev puudub). *wiki.dbpedia.org: About*. Kasutamise kuupäev: 23. 04 2012. a., allikas wiki.dbpedia.org: http://wiki.dbpedia.org/About

Dietrich, D., Gray, J., McNamara, T., Poikola, A., Pollock, R., Tait, J., et al. (kuupäev puudub). *Why Open Data?* Kasutamise kuupäev: 14. 03 2012. a., allikas Open Data Handbook: http://opendatahandbook.org/en/why-open-data/index.html

Ehitisregister. (kuupäev puudub). *Kasutajajuhend.* Kasutamise kuupäev: 06. 04 2012. a., allikas Ehitisregister: www.ehr.ee/static/ehrjuhend.doc

Isamaa ja Res Publica Liit ning Eesti Reformierakond. (kuupäev puudub). *Erakonna Isamaa ja Res Publica Liit ning Eesti Reformierakonna valitsusliidu programm.* Kasutamise

kuupäev: 29. 04 2012. a., allikas Valitsus - Valitsus.ee: http://valitsus.ee/UserFiles/valitsus/et/uudised/taustamaterjalid/Valitsusliit%20I.pdf

Kroes. (29. 07 2010. a.). *Parlamentary questions*. Kasutamise kuupäev: 11. 04 2012. a., allikas European Parliament: http://www.europarl.europa.eu/sides/getAllAnswers.do?reference=E-2010-3436&language=EN

Laur, M. (27. 20 2010. a.). *Turvaserveri kasutusjuhend.* Kasutamise kuupäev: 10. 04 2012. a., allikas http://ee.x-rd.net/docs/est/turvaserveri_kasutusjuhend.pdf

Microsoft Corporation. (kuupäev puudub). *What is the Resource Description Framework?* Kasutamise kuupäev: 23. 04 2012. a., allikas MSDN – Explore Windows, Web, Cloud, and Windows Phone Software Development: http://msdn.microsoft.com/en-us/library/aa303722.aspx

MusicBrainz. (kuupäev puudub). *Start Page | Musicbrainz D2R Server*. Kasutamise kuupäev: 23. 04 2012. a., allikas Musicbrainz D2R Server: http://dbtune.org/musicbrainz/

Open Definiton. (kuupäev puudub). *Defining the Open in Open Data, Open Content and Open Services*. Kasutamise kuupäev: 15. 03 2012. a., allikas Open Definition: http://opendefinition.org/

OpenLink Software. (kuupäev puudub). *Mapping Relation Data to RDF with*. Kasutamise kuupäev: 25. 04 2012. a., allikas http://virtuoso.openlinksw.com/: http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html

OpenLink Software. (kuupäev puudub). *Virtuoso Open-Source Wiki : Main.VirtLinkRemoteTables*. Kasutamise kuupäev: 26. 03 2012. a., allikas Virtuoso Open-Source Wiki: http://www.openlinksw.com/dataspace/dav/wiki/Main/VOSSQL2RDF

OpenLink Software. (kuupäev puudub). *Virtuoso Open-Source Wiki : Main.VOSIntro*. Kasutamise kuupäev: 25. 03 2012. a., allikas Virtuoso Open-Source Wiki: http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSIntro

OpenLink Software. (kuupäev puudub). *Virtuoso Open-Source Wiki : Virtuoso SPARQL*. Kasutamise kuupäev: 26. 03 2012. a., allikas Virtuoso Open-Source Wiki: http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSPARQL

Riigi Infosüsteemi haldussüsteem. (kuupäev puudub). *Infosüsteem: Riiklik ehitisregister*. Kasutamise kuupäev: 06. 04 2012. a., allikas RIHA - Riigi infosüsteemi haldussüsteem: https://riha.eesti.ee/riha/main#1333717240808tybgwGKFoKHfT1e

W3 Consortium. (19. 08 2011. a.). *LargeTripleStores - W3C Wiki*. Kasutamise kuupäev: 26. 03 2012. a., allikas W3C Wiki: http://www.w3.org/wiki/LargeTripleStores

LISAD

Lisa 1 Virtuoso OpenSource ODBC paigaldusjuhend

Käesolev juhend on loodud täiendamaks ametlikku juhendit, mille lahkhelide tõttu ei õnnestunud luua Virtuoso OpenSource andmebaasiga ODBC (Open Database Connectivity) ühendust kasutades PHP programmeerimiskeelt. Juhendis paigaldatakse Virtuoso OpenSource Edition, ODBC, Apache ja PHP Debian 6 operatsioonisüsteemile.

Vajalike sõltuvuste paigaldamine

Paigalda järgnevateks etappideks vajalikud sõltuvused kasutades apt-get paketihaldurit. Paigaldamise käigus küsib Debiani installatsiooni meediat.

```
apt-get install dpkg-dev devscripts libreadline5-dev quilt -y
```

Koodinäide 3 - Sõltuvuste paigaldamine

Virtuoso paigaldamine

Lae alla vajalikud paketid wget tööriista abil:

```
wget http://ftp.debian.org/debian/pool/main/v/virtuoso-
opensource/virtuoso-opensource_6.1.2+dfsg1-1.dsc
wget http://ftp.debian.org/debian/pool/main/v/virtuoso-
opensource/virtuoso-opensource_6.1.2+dfsg1.orig.tar.gz
wget http://ftp.debian.org/debian/pool/main/v/virtuoso-
opensource/virtuoso-opensource_6.1.2+dfsg1-1.diff.gz
```

Koodinäide 4 Pakettide allalaadimine

Paki lahti laetud paketid dpkg-source abil:

```
dpkg-source -x virtuoso-opensource_6.1.2+dfsg1-1.dsc
```

Koodinäide 5 - Lahti pakkimine

Liigu kausta:

```
cd virtuoso-opensource-6.1.2+dfsg1
```

Koodinäide 6 - Kausta vahetamine

Ehita sõltuvused aptitude abil:

```
aptitude build-dep virtuoso-opensource -y
```

Koodinäide 7 - Sõltuvuste ehitamine

Määra lähtekoodi formaadiks 3.0 „*Quilt*“ kasutades echo käsku:

```
echo "3.0 (quilt)"> debian/source/format
```

Koodinäide 8 - Lähtekoodi formaadi muutmine

Pakenda debuild abil:

```
debuild -us -uc
```

Koodinäide 9 - Pakendamine

Paigalda Virtuoso paketihalduri dpkg abil. Paigalduse käigus tuleb süsteemsetele kasutajatele dba ja dav määrata parool:

```
dpkg -i ../*.deb
```

Koodinäide 10 - Virtuoso paigaldamine

Käivita virtuoso-opensource teenus:

```
/etc/init.d/virtuoso-opensource-6.1 start
```

Koodinäide 11 - Virtuoso teenuse käivitamine

Apache ja PHP

Paigalda apache2 ja PHP5 paketihalduri apt-get abil:

```
apt-get install apache2 libapache2-mod-php5 php5-cli php5-common
php5-cgi -y
```

Koodinäide 12 - Apache ja PHP paigaldamine

ODBC ja PHP-ODBC

Paigalda ODBC paketid:

```
apt-get install odbcinst odbcinst1debian1 php5-odbc -y
```

Koodinäide 13 - ODBC pakettide paigaldamine

Määra failis „/etc/odbcinst.ini“ Virtuoso-ODBC draiveri asukoht:

```
[virtuoso-odbc]
Driver  = /usr/lib/odbc/virtodbc.so
```

Koodinäide 14 - Virtuoso-ODBC draiveri määramine

Määra failis „/etc/odbc.ini“ ODBC andmeallikad:

```
[ODBC Data Sources]
VDS  = Virtuoso

[VDS]
Driver  = virtuoso-odbc
Description  = Virtuoso Open-Source Edition
ServerName  = localhost:1111

[VOS]
Driver  = /usr/lib/odbc/virtodbc.so
Description  = Virtuoso OpenSource Edition
Address = localhost:1111
```

Koodinäide 15 - Andmeallikate määramine

Taaskäivita Apache2 teenus:

```
/etc/init.d/apache2 restart
```

Koodinäide 16 - Apache2 taaskäivitamine

Virtuoso ja ODBC draiveri paigalduse testimine

Virtuoso OpenSource veebiserver töötab vaikeväärtusena 8890 pordil. Eduka paigalduse korral vastab http://localhost:8890/conductor/ aadressil haldustöörist „*Conductor*“ (vt Ekraanipilt 5**Error! Reference source not found.**). Sisse saab logida kasutajanime: „dba“ ning paigalduse käigus määratud parooli abil.

Ekraanipilt 5 - Conductor vaade

Aadressil http://localhost:8890/sparql/ vastab SPARQL Endpoint (vt Ekraanipilt 6).

Ekraanipilt 6 - SPARQL Endpoint

PHP-ODBC ühenduse katsetamiseks loome „/var/www/odbc-test.php“ faili järgneva sisu:

```
<?php
$conn   = odbc_connect('VOS', 'dba', 'dba');
$query  = 'SELECT DISTINCT ?g WHERE {GRAPH ?g {?s ?p ?o.}}';
$result = odbc_exec($conn, 'CALL DB.DBA.SPARQL_EVAL(\'' .
$query . '\', NULL, 0)');
?>
<ul>
<?php while (odbc_fetch_row($result)): ?>
    <li><?php echo odbc_result($result, 1) ?></li>
<?php endwhile; ?>
</ul>
```

Koodinäide 17 – ODBC päring PHP keeles

NB! odbc_connect(’VOS’, ’dba’, ’dba’) tuleb märkida paigaldamise ajal määratud parool. Eduka ühenduse korral vastatakse aadressil http://localhost/odbc-test.php järgmiste ridadega:

- http://www.openlinksw.com/schemas/virtrdf#
- http://localhost:8890/DAV
- http://www.w3.org/2002/07/owl#

Koodinäide 18 – Kuvatav vastus

Lisa 2 Näidisandmete laadimine

Käesolev juhend on mõeldud Apache, PHP ja näidisandmete laadimisskripti paigaldamiseks Debian 5.x või 6.x operatsioonisüsteemile.

Päring

Päring tehakse ehr.EN_EhitiseAndmed.v1 teenusele järgmiste ehitiste ja rajatiste kohta:

Ehitise Id	EHR kood	Nimetus	Staatus	Maakond	Linn/vald
1328844	102023504	aiamaja	K	Ida-Viru maakor	Jõhvi vald
2472456	103016815	ÄRIHOONE	K	Pärnu maakond	Pärnu linn
2492006	103048354	MAJANDUSHOONE	K	Pärnu maakond	Pärnu linn
2636596	220192107	VEDELGAASIMAHUT	K	Pärnu maakond	Pärnu linn
2684746	101000215	elamu	K	Harju maakond	Tallinna linn
2699634	101013142	Elu- ja ärihoone	K	Harju maakond	Tallinna linn
2704674	101016398	elamu-kauplus	K	Harju maakond	Tallinna linn
2704982	101016435	elamu	K	Harju maakond	Tallinna linn
2797654	220231475	välisgaasitorustik	K	Harju maakond	Viimsi vald
2802130	120233498	Moe- ja vabaajakeskuse	K	Tartu maakond	Tartu linn
2977258	220430897	kasvuhoone	K	Ida-Viru maakor	Jõhvi vald
3349506	120537397	ÄRIKESKUS	K	Harju maakond	Tallinna linn
3355236	220545560	Jõgeva-Põltsamaa gaasit	K	Jõgeva maakond	Jõgeva linn
3355236	220545560	Jõgeva-Põltsamaa gaasit	K	Jõgeva maakond	Jõgeva vald
3512372	220591485	maagaasi jaotustorustik K	K	Harju maakond	Tallinna linn
3541676	120603902	eramu	E	Harju maakond	Tallinna linn
3638122	120644987	Bensiinijaam	M	Jõgeva maakond	Jõgeva linn
3638860	220645262	Piirdeaed	E	Harju maakond	Vasalemma vald

Tabel 4- Päritavad ehitised

Sõltuvuste paigaldamine

Paigalda arhiivitööriist unzip:

```
apt-get install unzip -y
```

Koodinäide 19 – Unzip paigaldamine

Apache ja PHP paigaldamine

Paigalda Apache2 ja PHP5:

```
apt-get install apache2 php5 libapache2-mod-php5 -y
```

Koodinäide 20 - Apache2 ja PHP paigaldamine

Taaskäivita apache2:

```
/etc/init.d/apache2 restart
```

Koodinäide 21 - Apache taaskäivitamine

Laadimisskript

Tõmba wget abil alla ZIP arhiiv /var/www kausta:

```
wget -O /var/www/ehr.zip http://www.soatrader.com/emta_test/ehr/ehr.zip
```

Koodinäide 22 - Laadimisskripti alla laadimine

Seejärel paki lahti alla laetud arhiiv /var/www kausta:

```
unzip /var/www/ehr.zip -d /var/www/
```

Koodinäide 23 - Arhiivi lahtipakkimine

Konfigureerimine

Ava konfiguratsiooni muutmiseks järgnev fail /var/www/config.php:

```
nano /var/www/config.php
```

Koodinäide 24 - Konfiguratsiooni faili avamine

Ilmub järgmine vaade

Ekraanipilt 7 - Konfiguratsiooni muutmine

Määra „*turvaserver*“, „*asutus*“ ja „*isikukood*“ muutujatele sobivad väärtused. Salvestamiseks ning väljumiseks vajuta CTRL ja X, seejärel Y ning Enter.

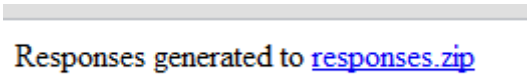
Määra kaustale õigused:

```
chmod 0757 /var/www/
```

Koodinäide 25 - Kausta õiguste määramine

Andmete laadimine

Navigeeri aadressile http://localhost/ehr.php, andmete laadimise õnnestumisel kuvatakse järgmine leht



Ekraanipilt 8 - Laadimise õnnestumine

Responses.zip arhiivis paiknevad päringute vastused XML failidena.