Tallinna Ülikool
Informaatika Instituut

Pädevushaldus RESTful veebiteenuste abil

Seminaritöö

Autor: Eigen Lenk
Juhendaja: Mart Laanpere

Tallinn 2010

Sisukord

Sissejuhatus

Käesoleva töö teemaks on pädevushaldus RESTful veebiteenuste abil.

Igale ettevõttele on äärmiselt oluline omada detailset ülevaadet töötajatest, kuid lisaks sellele ka infot selle kohta, kuivõrd heal tasemel ollakse töödataval alal. Teisest küljest on igale töötajale oluline näidata, kuid samas ka tõestada, oma pädevust viisil, mis välistaks enda oskuste üle- ja alahindamise ning mis oleks üheselt aktsepteeritav. Siinkohal tulebki mängu pädevushaldus, mis võimaldab töötajatel tõestada oma komptentsust ning ettevõtetel näha oma töötajate pädevusi. Ettevõttele annab see lisaks veel võimaluse leida täpselt sobiv inimene antud tööülesandele, määrates otsinguparameetrites nõutav pädevuste tase. Samuti on neile oluline tõsta oma töötajate innovaatilisust ning efektiivsust, samas säilitades oma eelist. Läbi aja on selline haldus liikunud üha enam suurkorporatsioonide siseinfolt personaalse arengu, teadmiste jagamise ning e-õppe kasutusse.

Muidugi on pädevused ning sellega seotu vaid üheks osaks suuremast pildist, mis on seotud arengu ja õppega. Üldisel juhul on kompetentside haldus üheks uurimisvaldkonnaks üldisemas teadmiste- ja õpihalduses. Üheks selliseks projektiks on Euroopa komisjoni programmi raames läbi viidav rahvusvaheline uurimisprojekt IntelLEO[1]. Projekti eesmärgiks on arendada teenuseid, mis ühendaksid kaht või enamat ärilist ja haridusvalda kuuluvat kogukonda ning organisatsiooni ühtseks õppekeskkonnaks andes võimaluse edukaks teadmiste jagamiseks.

Töö eesmärgiks on anda ülevaade pädevushaldusest ja kirjeldada selle puhul kasutatavat veebiteenustel põhinevat tarkvara, alustades üldisemalt veebitarkvara arhitektuuri ning sealt edasi veebiteenuste lahti seletamisega.

Autor valis antud teema, kuna puutus valdkonnaga kokku praktikal, mille käigus hakkas teema huvi pakkuma, ning kasutas antud võimalust õpitust seminaritöö kirjutada. Autor leiab, et antud teema on oluline just seetõttu, et pädevused ja nende efektiivne haldamine, kuid see omakorda tähendab ka personali efektiivset haldamist, on üheks nurgakiviks igale edukale ettevõttele.

[1] http://www.intelleo.eu/

Autor on oma töö jaganud kolme suuremasse peatükki, millest esimeses antakse ülevaade tarkavara, täpsemalt veebitarkvara, arhitektuurist ja võimalustest, kuidas arendustööle läheneda. Kirjeldatakse erinevaid mudeleid veebiteenustel põhineva tarvara koostamiseks. Samuti peatub autor lühidalt tarkvaraarhitektuuri ajalool. Teises peatükis tuleb juttu veebiteenustest ning nende jagunemisest, põhiliselt kahest, milleks on SOAP ja RESTful tüüpi teenused. Nende omavahelises võrdluses tuuakse välja mõlemi eelised ja puudused, ning mõningad näited. Viimases peatükis räägib autor lähemalt pädevustest ning pädevushaldusest, sinna alla kuuluvatest mõistetest ja nende omavahelistest soestest ning sellest, millistest teenustest üks pädevushalduse tarkvara koosneb.

1. Muutused veebitarkvara arhitektuuris

Enne kui saab lähemalt rääkida sellest, kuidas on lähenemine tarkvara arhitektuuri läbi aja muutunud, tuleb lahti seletada, mis üldse on tarkvara arhitektuur. Arhitektuur on olemuselt distsipliin, mis käsitleb põhimõtteid, mida kasutatakse millegi kavandamisel, olgu selleks siis maja, laev või tarkvara. Kuigi tarkvara arhitektuuri jaoks ei ole siiani kindlat ühest definitsiooni, mõeldakse selle all üldiselt süsteemi ühe või mitmete osade struktuuri, ja kirjeldatakse sinna alla kuuluvaid tarkvara komponente, nende komponenditega seotud muutujaid ning seoseid nende vahel.[2] Lisaks kavandamisele on arhitektuuril ka teine kasutusvaldkond, milleks on dokumenteerida tehtud otsuseid ja kavandeid, millest on kasu uute arendajate lisandumisel ning erinevate komponentide taaskasutamisel. Eriti saab komponentide taaskasutamisest rääkida veebitarkvaras, mille puhul räägitakse moodulpõhisest arendusest. Moodulpõhisest arendusmeetodist räägin järgnevas peatükis.

Arvutiteaduste puhul on oluline mõiste, millest rääkida, keerukus. Keerukus hõlmab pea kõiki arvutiga seotud valdkondi, sealhulgas tarkvara, mida püütakse formuleerida lihtsamalt mõistevas vormis. Tarkvara arhitektuuri mõiste ei ole kuigi vana, kuid selle fundamentaalseid põhimõtteid on mõningal puhul kasutatud juba 80-ndate keskpaigus. Esmased katsed tarkvara kirjeldada ei olnud iseenesest muud kui lihtsad skeemid kastidest ja neid ühendavatest joontest. Esimene tarkvara arhitektuuri kui konseptsiooni kirjeldus pärineb Edsger Dijkstra sulest, kes pani selle kirja 1968. aastal.[3]

1.1 Ajalugu, monoliitne tarkvara

Tarkvara on alati olnud töömahukas projekt ning selle arendusmeetodid on olnud pidevas arengus. Kuid veel isegi 80-ndate aastate algul ei olnud välja kujunenud formaalseid meetodeid, kuidas peaks arendus olema läbi viidud. Enne kirjutati kood ja alles seejärel mõeldi, kuidas seda struktueerida, mis viis projekti suurenenud ajakulu ning lõhki läinud eelarveteni. Programmid olid tihti täis vigu, ning võisid isegi viia surmaga lõppenud õnnetusteni. Ühe näitena võib tuua niinimetatud Therac 25 intsidendi, mille puhul radiatsiooni teraapia masina spetsiifiline süsteem ei lõpetatud tööd, kui patsiendile väljastati juba surmavat

[2] http://www.ibm.com/developerworks/rational/library/feb06/eeles/
[3] http://homepages.cwi.nl/~apt/ps/dijkstra.pdf

doosi radiatsiooni.[4] Hakati otsima lahendusi nendele probleemidele ja tähelepanu hakati pöörama arenduseks vajalikele töövahendite ning meetodite välja töötamisele. Hakati tähelepanu pöörama professionaalsusele, näiteks võeti kasutusele litsentseerimine.

Ühest kihist koosnev tarkvara, milles ühendatakse nii kasutajaliidest kui ka andmete töötlust, nimetatakse monoliitseks tarkvaraks.[5] See tähendab seda, et programm on täielikult ühes tükis ning sellest midagi eraldada pole võimalik, kuna kõik aspektid on välja töötatud just sellest programmist lähtuvalt. Tänapäeval leiab selline arendusviis üha vähem kasutust. Miks? Üheks põhjuseks kindlasti see, et arendajad tahavad olla võimalikult paindlikud kliendi soovidega arvestamisel ka projekti kestel, mitte ainult algstaadiumis. Monoliitse tarkvara arenduse puhul enamasti kasutatud metoodika oli kosk mudel, mille puhul iga etapp oli enne projekti algust täpipealt paika pandud, mis välistas muudatused projekti kestel ning lõpuks võis arenduse periood venida ning rahaline kulu suureneda.[6] Põhiliseks põhjuseks oligi see, et sisse oli vaja viia muutusi, mida ette ei nähtud ning mis antud mudelisse ei sobinud.

Olgugi, et tänapäeva tarkvara arenduses kasutatakse teisi meetodeid, ei tähenda see seda, et programmeerijaid peaks tunduvalt rohkem olema. Oma 1975. aastal ilmunud raamatus „The Mythical Man-Month" väidab Frederick Brooks, et arendajate lisandumine pigem aeglustab kui kiirendab projekti, peamiselt seetõttu, et iga uue inimese puhul lisandub vältimatult suhtlust, see tähendab, et igale uuele inimesele tuleb tööd tutvustada, ja juba hästi töötavasse meeskonda sulandada, mis viib selleni, et meeskonna sisene kommunikatsioonikeerukus kasvab. Kuid suuremate tööde puhul on suurem arendajate arv möödapääsmatu, kuna tarkvara peab alati olema täiuslikum ning kiiremini valmima. Selle valutumaks saavutamiseks on mitmeid võimalusi, millest ma järgnevalt räägingi.

1.2 Moodulpõhine veebitarkvara

Üheks selliseks võimaluseks on moodulpõhine tarkvara. Mis on moodul? Moodul on mingi suurema süsteemi üks alamsüsteem, mida on võimalik käsitleda täielikult iseseisvalt. Sellest lähtuvalt on moodulpõhine arendus, mida kasutatakse nii tarkvara, antud juhul veebitarkvara, kui ka näiteks autode ehitusel, üles ehitatud nii, et see koosneb mitmest iseseisvast osast ning

[4] http://sunnyday.mit.edu/papers/therac.pdf
[5] http://msdn.microsoft.com/en-us/library/aa480455.aspx
[6] http://www.linux.com/archive/feed/31530

iga osa kallal töötavad erinevad inimesed. See annab võimaluse projekti kiiremini läbi viia, kuna ükski komponent ei sõltu teisest, see tähendab, et keegi ei pea ootama kellegi teise järel, et oma osa valmis teha. Lõppfaasis ühendatakse moodulid ühtseks tervikuks. Moodulpõhise arendusega tarkvara võib nimetada ka hajutatud arhitektuuriga tarkvaraks, kuna osad on nõrgalt või mitte üldse seotud.

Moodulpõhisel tarkvara arendusel on mitu tahku. Ühest küljest on teoreetiline pool, mis ütleb seda, et programmi osi ehk komponente peaks saama üksteisest eraldada ning iseseisvalt arendada ja käsitleda.[7] Samuti peaks mooduleid saama kasutada teistes programmides, mis vähendaks nii arenduseks kuluvat aega kui raha. Kuid praktilise poole pealt ei ole see alati nii lihtne, ning arendajatel on vahel mõistlikum endal vajalikud tükid kirjutada, kui kasutada kellegi teise poolt valminud osi. Kuid praktilisusel on omakorda kaks tahku. Üks tahk on traditsioonilisel kujul tarkvara, mille puhul moodulpõhine arendus ei ole alati kõige kasulikum, kuna raske on leida ühiseid osi kõigi niivõrd erinevate programmide vahel. Teiselt poolt on veebitarkvara, mille puhul kasutatavad veebiteenused sobivad täpselt antud arendusmudelisse. Kindlasti on selle sobivuse üheks põhjuseks veebistandard ning protokollid (näiteks HTTP), mis on kõigi programmide puhul ühiseks jooneks.

1.3 SOA

Veebi puhul oleks mõistlik kasutada teenustele orienteeritud arhitektuuri (service-oriented architecture), mis põhineb sellel, et klientrakendus saadab serverile formaalselt defineeritud teate, näiteks XML Schema SOAPi puhul, milles on kirjas nii välja kutsutav teenus kui teenuse tööks vajalikud parameetrid. Veebiteenus võib olla vastavalt vajadusele sünkroonne või asünkroonne. Sünkroonse ning asünkroonse teadete edastuse vahe on selles, et esimesel juhul peale teate saatmist programm seisatub ning ootab vastuse ära[8], kuid teisel juhul programm jätkab tööd ning vastusega tegeletakse siis, kui see saabub.

SOA rakenduste, siinkohal pean silmas neid, mis on serveripoolsed, puhul on tegemist moodulpõhise mudeliga, kui nende vahel on nõrgad seosed (*loose coupling*), mis tähendab seda, et iga teenus on kursis teiste kättesaadavate teenustega, kuid mis ei tähenda, et neid välja

[7] http://www.techdirt.com/articles/20060405/027227.shtml
[8] http://msdn.microsoft.com/en-us/library/ms713563(VS.85).aspx

kutsutakse. Samuti tagab SOA üldjuhul viisi, kuidas anda klientrakendusele teada kättesaadavatest teenustest.[9]

Järgnevalt mõningad põhimõtted, mis defineerivad SOA teenuste arendust ja kasutust. Esiteks, teenuse osad, nagu eespool öeldud, peaksid olema modulaarsed ning moodulite vahel peaks olema võimalus suhelda (*interoperability*). Teiseks peavad teenused vastama nii üldlevinud kui valdkonna-spetsiifilistele nõuetele. Kolmandaks peavad teenused olema identifitseeritavad ning nende tegevust peab saama jälgida.

SOA arhitektuuri kasutavad mitmed tehnoloogiad, sealhulgas SOAP/RPC, REST, CORBA ja teised. SOAP ja REST veebiteenustest räägin lähemalt antud töö järgmises peatükis. SOA skeemi jaoks vaata Lisa 1.

1.4 Kihiline arhitektuur

Kihilise arhitektuuri puhul räägitakse arhitektuurist, milles kasutajaliides, progammiloogika ning andmed on üksteisest eraldatud. Suhtlus käib kihilt-kihile, alati eelmisele või järgmisele kihile, mitte kunagi üle ühe. Enamasti kasutakse seda klient-server rakenduste puhul, mille alla kuuluvad ka kõik veebilahendused -ja rakendused. Üldlevinud on kolmekihilisus.[10]

- **Esitluskiht** - See on kiht, mida näeb lõppkasutaja. See võib olla kas graafiline või tekstiline kasutajaliides, või veebileht. Kommunikeerumine käib rakenduskihiga ning kasutajale väljastakse operatsioonide tulemusi. Veebitarkvara puhul on see tavaliselt veebiserver, kus asuvad väljastatavad leheküljed.

- **Rakenduskiht** - Rakenduskiht ehk loogikakiht on kiht, mille ülesandeks on ette antud operatsioonide täitmine. Suhtlus käib nii esitluskihiga, mis näitab mida ning milliste parameetritega on vaja teha, ning andmekihiga, kus on olemas ülesannete täitmiseks vajalikud andmed.

[9] http://msdn.microsoft.com/en-us/library/aa480021.aspx
[10] http://www.linuxjournal.com/article/3508

Veebitarkvara puhul kuulvad sellesse kihti näiteks Java EE, PHP, ASP.NET ja teised platvormid.

- **Andmekiht –** Veebirakenduste puhul moodustavad andmekihi enamasti andmebaasid, kuid tavaprogrammides võivad olla ka failid. Suhtlus käib rakenduskihiga, mille jaoks väljastatakse nõutavaid andmeid või muudetakse olemasolevaid vastavalt käsklusele.

Arendajatele pakub kihiline arhitektuur mitmeid eeliseid, millest põhiliseimaks võib lugeda seda, et vajadusel saab ühte kihti muuta, see tähendab, koodi ümber kirjutada või välja vahetada, teisi kihte mõjutamata. Teise põhilise eelisena võib välja tuua selle, et andmekihi ja rakenduskihi eraldamisega on võimalik tarkvarale langevat koormust jagada, see tähendab kui andmebaas asub eraldi serveris, väheneb koormus serverile kus asub rakenduskiht, näiteks PHP serveri näol.

1.5 SCA

SCA ehk *Service Component Architecture* puhul on tegemist suhteliselt uue spetsifikatsiooniga, mis kirjeldab programmide ja süsteemide ehituse modelleerimist kasutades selleks SOA-d. SCA täiendab komponentidel ehk moodulitel põhinevat arhitektuuri, eelnevaid lähenemisi teenuste rakendamisele ning baseerub avatud standarditel. Põhimõtteliselt on SCA eesmärk jagada veebiteenuse väljatöötamine kahte ossa, millest esimene on komponentide arendus ning teine eri komponentide ühendamine ühtseteks osadeks ehk komposiitideks. Viimane on osa SCA põhidefinitsioonist. Pole vahet, kas komponendid on kirjutatud Javas või C++ keeles - kui need on programmeeritud SCA mudelite järgi, saab neid omavahel ühendada.
Komposiit, nagu öeldud, koosneb komponentidest. Komponendid võivad töötada ühe protsessina ühes arvutis või mitmetes protsessides mitmes arvutis korraga, samuti võivad komponendid kasutada ühte või mitut erinevat tehnoloogiat. Komposiidi defineerimiseks on konfiguratsiooni fail. SCA komponendidl on 3 põhiosa: teenus (*service*), omadus (*property*) ning viide (*reference*).[11]

Teenuse puhul on tegemist mingi hulga operatsioonidega, mida on võimalik kliendil käivitada. Teenuse kirjeldamiseks võib kasutada nii Java liideseid kui WSDLi (vaata 2. peatükki). Lisaks sellele, et komponendil on oma hulk teenuseid, võib vaja minna ka teise komponendi teenuseid, milliseid saab sel juhul määrata viiteid kasutades. Viited teistele komponentidele aitavad kaasa ka sellele, et arendajatel on lihtsam näha seoseid ja sellest tekkivat struktuuri. Samuti tähendab see seda, arendaja ei pea kirjutama eraldi koodi, mis otsiks üles vajalikud välised komponendid – SCA teeb seda automaatselt käitusaegselt. Lisaks teenustele ning viidetele on komponenditel ka omadused, mida saab komponendi siseselt kasutada. Ntäiteks võib komponent endas sisaldada omadust, kus on kirjas, mis riigis asub teenuse server ja seda kasutades vastavalt vajadusele teenuse vastust kohandada.[12]

[11] http://www.ibm.com/developerworks/library/specification/ws-sca/
[12] http://www.davidchappell.com/articles/Introducing_SCA.pdf

2. Veebiteenused

Veebiteenuste puhul on üldjuhul tegemist tarkvara raamistikuga ehk APIga (*application programming interface*), mis asub veebis ning on kättesaadav üle võrgu, näiteks läbi Interneti. Käivitatavad programmi osad asuvad välises serveris ja kliendi osa on vaid teenuseid välja kutsuda ilma lisaosi paigaldamata.[13]

Laias laastus jagunevad veebiteenused kaheks. Enim levinud teenused on need, mis kasutavad kommunikatsiooni puhul SOAP standardile vastavaid sõnumeid ja vastuseid. Veebiteenuse jaoks kasutavad klient ja server suhtluseks HTTP protokolli ning defineeritud hulka sõnumeid koos kindla struktuuriga vastustega. Neid vastuseid kujutatakse üldiselt kas XML või JSON vormingus. Teist liiki teenused on RESTful veebiteenused, mis kasutavad standardseid HTTP käske GET, POST, harvemal juhul DELETE, ning mille väljakutsutav funktsioon on määratud URI-ga.[14]
Järgnevalt räägingi lähemalt neist kahest veebiteenuse tüübist ja toon välja nende eripärad, eelised ja puudused ning mõningad näited.

2.1 SOAP

SOAP ehk *Simple Object Access Protocol*, on protokoll ja spetsifikatsioon struktuurse ning kindlaks määratud tüüpi informatsiooni vahetamiseks hajutatud keskkonnas, mis baseerub XML vormingul ning sobib veebiteenuste juures kasutamiseks. Tegemist ei ole süsteemiga, mis tegelikkuses sõnumeid saadaks ja vastu võtaks, vaid sõnumi ning selle transpordi mudeliga seotud loogikaga. Ühenduse loomise teenuse ja kasutaja vahel ning transpordiga tegeleb HTTP protokoll. SOAP ei pane paika programmi semantikat ega mudeleid programmeerimise jaoks, vaid defineerib lihtsa mehhanismi moodulpõhise programmi ning moodulites olevate andmete jaoks. SOAP teade koosneb ümbrikust (envelope), milles omakorda sisalduvad teate päis (header) ja põhiosa (body). Päise ja põhiosa sisu ei määra SOAP, vaid see on rakenduse-spetsiifiline, samas, paneb SOAP paika kuidas neid käidelda. Päise olemasolu teates on valikuline ning see võimaldab edastada informatisooni, millega ei pea otseselt tegelema rakendus. Põhiosa seevastu on teates kohustuslik.

[13] http://www.w3.org/TR/ws-arch/
[14] http://www.w3.org/TR/ws-arch/

2.1.1 WSDL

WSDL ehk Web service description language on XML-vormingus keel, millega defineeritakse veebiteenuseid. Kõige lihtsamal kujul on WSDL failis ära määratud serveri poolsed lõpp-punktid (endpoints) ehk pordid, mille poole kindla sõnumiga pöördudes käivitatakse serveris nõutav funktsioon, mis omakorda tagastab vastuse. Võimalike kasutatavate funktsioonide hulga määrab pordi tüüp (port type), ehk hulk operatsioone, mida on võimalik käivitada. Sõnumid on kirjeldus andmetest ja andmetüüpidest, mida on vaja funktsiooni täitmiseks ning vastuse tagastamiseks. Sõnumid, operatsioonid ja lõpp-punktid on abstraktsel kujul, ehk neil ei ole otsest seost serveripoolse rakendusega, see tähendab ei ole oluline, mis keeles veebiteenus kirjutatud on. See võimaldab teenust taaskasutada. Teenuse moodustavad teatud hulk sisuliselt seotud lõpp-punkte.[15]

2.2 RESTful

Representational State Transfer on selgelt erineva lähenemisega arhitektuuritüüp võrreldes eelmises peatükis räägitud SOAP-iga. REST-i puhul saab igast teenusest rääkida kui eri rakenduse seisundist, millele vastab kindel URI. Iga URI puhul saab kasutada nii GET kui POST käske, millest esimest kasutatakse siis, kui tahetakse midagi serveri poolt kätte saada, teist aga siis, kui on vaja serverile midagi saata, mis omakorda neid siis edasi töötleb. Seisund ei kajastu füüsiliselt serveris, vaid see on olemas sisus, mida saadab klient rakendusele ning see omakorda kliendile tagasi. Näiteks kui kasutaja on esilehel, on ta rakenduse algseisundis, ning vajutades mõnele lingile, annab see käsu uude seisundisse minna ning see kliendile tagastada.

Kui SOAPi puhul tagastati klientrakendusele mingis kindlas vormingus vastus, mida siis klient omakorda vajalikule kujule töötleb, siis RESTful tüüpi veebirakenduste puhul tagastatakse juba vormindatud vastus tavaliselt kas HTML või XHTML kujul, see tähendab kliendile tagastatakse rakenduse uus seisund või kui lihtsamalt öelda, lehekülg. Samas, vastus ei pea olema koos kogu lehekülje vorminguga vaid võib olla ka üks osa leheküljest, näiteks tabel, mida siis klientrakenduses lisatakse, kuhu vaja. Kuid RESTful teenus võib ka tagastada JSON kujul vastuse, mida on siis võimalik töödelda samamoodi nagu SOAPi puhul XMLi.

[15] http://www.w3.org/TR/wsdl

2.3 Eelised ja puudused

SOAPi eeliseks võib lugeda seda, et veebiteenuse poolt tagastatav vastus on abstraktsel kujul ja ei sisalda visuaalset vormingut. See tagab selle, et ühte teenust saab kasutada mitmes kohas ning seda seejärel klientrakenduses vastavalt vajadusele näidata. Teine eelis on see, et kuna SOAP jookseb enamasti HTTP protokollil, ei teki probleeme tulemüüride ega paketikaotustega. Samas SOAPi puuduseks on see, et kui kasutada ühe veebiteenuse vastust rakenduses mitmes erinevas kontekstis, tuleb vastav vorminduse kood dupleerida või muuta vastavalt vajadusele. Teise SOAPi puudusena võib välja tuua selle, et tegemist on väga paljusõnalise formaadiga ning suuremate sõnumite puhul võib see aeglaseks jääda. [16]

RESTful seevastu võib tagastada juba vormindatud vastuse ning klientrakendusel ei jää üle muud, kui seda näidata. RESTfuli eelis on samas ka tema puudus, ehk juhul kui veebiteenuse poolt saadud vastus on juba vormindatud, on selle teise visuaalsesse konteksti viimine keeruline.

[16] http://www.ibm.com/developerworks/library/ws-pyth9/

2.4 Näiteid

Järgnevalt üks näide RESTful tüüpi veebiteenuse jaoks. Kui pöörduda veebiserveris aadressi *users/tonutamm* poole GET käsuga, tagastatakse kasutaja *tonutamm* andmed, näiteks kasutajakonto või profiili kujul. Samas, pöördudes GET käsuga aadressi *users* poole, tagastatakse nimekiri kõigist kasutajatest. Pöördudes *users/tonutamm* poole POST käsuga, võib see näiteks tähendada kasutaja andmete muutmist, mille väärtused on siis olemas POSTi muutujates.

On võimalik kasutada ka URI parameetreid, ehk URI lõppu lisatavaid muutujaid. Näiteks *topics/majandus?page=2* tähendaks otsingu vastest teise leheküle näitamist või *cars/mercedes?year=1980*, mis tagastaks kõik 1980. aasta Mercedese mudelid.

Töö autor toob ka ühe lihtsa näite SOAPi jaoks, kus ei ole välja toodud rakenduse koodi vaid ainult saadetavad teated. Tegemist on veebiteenusega, mis tagastab raamatu nime järgi otsides autori nime. Klient-rakendus saadab serverile sellise teate:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetBookAuthor xmlns:m="http://www.books.org/authors">
      <m:BookName>To Kill a Mockingbird</m:BookName>
    </m:GetBookAuthor>
  </soap:Body>
</soap:Envelope>
```

Serveris asuv teenus tagastab sellise vastuse:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <m:GetBookAuthorResponse xmlns:m="http://www.books.org/authors">
           <Author>Harper Lee</Author>
       </m:GetBookAuthorResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3. Pädevused

Nüüd, kui on kirjeldatud seda, kuidas läheneda veebiteenuste arendamisele, mis on üheks komponendiks pädevushalduse tarkvara juures, tuleb rääkida sellest, mis üleüldse on pädevus. Võib tekkida küsimus, mis vahe on pädevusel ja kompetentsil, inglise keeles vastavalt „*competence*" ning „*competency*". Ingliskeelsetel autoritel on kahe mõiste vahel teatav segadus, kuid üldjoontes on piir tõmmatud – „*competence*" on potensiaal midagi teha, kuid „*competency*" all mõeldakse tegelikku saavutusvõimet. ÕSi andmetel tähendavad nad mõlemad eesti keeles ühte ja sama, seega, järgnevas peatükis kasutab autor mõlemat mõistet segamini.

Pädevusi võib defineerida kui oskuste, teadmiste ja võimete kogumit.[17] Inimestel on kõiksugu pädevusi, kus mõnes ollakse võimekamad kui teises. Mõningaid oskusi, nagu näiteks suhtlemine, kasutatakse igapäevases elus, kuid on ka oskusi, mida kasutatakse tööalaselt. Neid võib nimetada pädevusteks ning nendeks võivad olla nii oskus kasutada arvutit ning leida vajalikke materjale Internetist, kui ka näiteks olla pädev akadeemiliselt, et oma valdkonda siis teistelegi õpetada. Olgu oskus milline tahes, ühel hetkel tekib vajadus oma oskust tõestada. See vajadus võib tekkida tööle kandideerides kui ka juba tööl olles. Lihtne lähenemisviis oleks teha inimesele test, valikvastustega või mitte, ning kontrollida tema teadmisi antud valdkonnast. See meetod on küll äärmiselt lihtne, kuid pole kindlasti absoluutse tõena võetava tulemusega. Mis siis saab, kui inimene teeb testi uuesti või kasutab spikrit, ning saab kaks korda parema tulemuse? Kas võib siis kindlasti järeldada, et ta on kaks korda pädevam kui eelnevalt? Siit tulenebki vajadus pädevusi süsteemsemalt hinnata. Üheks võimaluseks on inimesel lasta koostada portfoolio oma tehtud töödest antud alal. Eriti mugav oleks seda teha veebis, kuhu saaks ka üles laadida materjale, mis on antud pädevusega seotud, ning neid seejärel omavahel siduda. Materjaliks võib olla nii tunnistused kui ka mõned muud dokumendid. See annab võimaluse hinnata inimese oskusi läbi tehtud tööde, mitte põhineda ainult testidel.

Pädevusnõudeid peetakse tänapäeval oluliseks kolmes valdkonnas: inimressursi juhtimisel (ehk personalitöös), kutsehariduse/täiendõppe õppekavade koostamisel ja töötaja tööalase toimetuleku hindamisel (Lachance, 1999; Lucia & Lepsinger, 1999).

[17] http://publications.andreas.schmidt.name/schmidt_kunzmann_OntoContent06.pdf

Pädevushaldusest personalitöös rääkisin ma sissejuhatuses, samas, lisada võib veel seda, et pädevusepõhine valik lähtub eeldusest, et tööga hakkama saamine ja tööga rahulolu on kõige suurem siis, kui töötaja pädevused vastavad tema tööülesannetele. Kutsehariduses ja täiendõppes kasutatakse pädevusi kui indikaatoreid, et välja töötada relevantseid õppematerjale, õppekavasid ja õpitegevusi. Tulevaste tööandjatega koostöös sõnastatud pädevusnõuded annavad võimaluse tuvastada kattumised ja puuduvad teemad õppekavades. Töötajate toimetuleku hindamise puhul pakuvad pädevusnõuded objektiivsemat laadi raamistiku parimate töötajate väljaselgitamiseks (näiteks tulemuspalga määramisel või koolitusvajaduse hindamisel).

3.1 Pädevuste ontoloogia

Ontoloogia üleüldiselt on formaalne kirjeldus teadmistest konseptuaalsel kujul, see tähendab mingi valdkonna jaoks oluliste mõistete ning nendevaheliste seoste kogum. Seostel võivad olla nimed. Kui rääkida pädevuse ontoloogiast, siis näiteks pädevuse ja töötaja vaheline seos võib kanda nime „omab kompetentsi”. Kindlasti ei kuulu pädevuste ontoloogiasse ainult pädevused iseseisvalt, vaid ka teemad, millesse pädevused kuuluvad. Samuti sisaldab ontoloogia töötajaga seotuid mõisteid ning seoseid, nagu näiteks tõestus ehk materjal, mis seob omavahel töötaja ja kompetentsi ning tõestab pädevust mingis valdkonnas. Tõestusele lisandub tase ning aja mõiste, ehk millal on õpitu omandatud. Ontoloogia siseselt on töötaja jaoks tegemist tõestustega, samas kui kompetentsi poolelt vaadelduna võib seda nimetada ka kompetentsi tasemeks.

Järgnevalt tabel pädevuste ontoloogia jaoks, mis pärineb IntelLEO projekti prototüübist ning mis peaks andma hea ülevaate sellest, kuidas on omavahel seotud pädevused ning tõestused.. Tabelis on ära toodud omadused, andme- või klassi tüüp, olenvalt omaduse tüübist, ning kirjeldus.

Class	Property	Data type	Description
Competence	domainTopicRef	Skos::Concept	Reference to the concept of the domain ontology that best describes the competency.
	skillRef	Skos::Concept	Reference to the skill that form the basis of the competency.
	Dc::Description	Xsd::String	A human readable description of the competency.
	requires	Competence	Dependency relationship between two competencies - in order to attain one, another one has to be already acquired.
	prerequisiteFor	Competence	Inverse of the requires property.
CompetenceRecord	forCompetence	Competence	The property establishes a relation with the competency that the given record is about.
	recordedLevel	CompetenceLevel	The recorded competency level; can be: Beginner, Intermediate, and Advanced
	recordedDate	Xsd::dateTime	The date when the competency record was created.
	competenceSource	foaf:Document or la:Activity	The resource or activity which was used for acquiring the competency.
	validatedUsing	Validation	The validation method that was applied to verify someone's competency.

Tabel 1 Pädevuste ontoloogia

3.2 Pädevushalduse tarkvara

Pädevushalduse tarkvara eesmärgiks on võimaldada pädevuste haldamist nende lisamise, kirjeldamise, vajadusel muutmise ning omavaheliste seoste tekitamise teel. Omavahelised seosed tähendavad seda, et üks või enam pädevust on teisele pädevusele eelduseks. Sellise halduse põhjaks on kompetentside kataloog. Samuti peaks tarkvara võimaldama kasutajal määrata, millised pädevused on tal olemas ning lisada tõestusmaterjali portfoolio või tunnistuste näol.

Sellist tarkvara võib leida põhiliselt kahel kujul. Üks, kus tarkvara arendus käib ühes suures tükis ning kasutajaskond on organisatsioonisisene. Organisatsioonisisene on ka pädevusinfo. Teine võimalus on ühtne teenus läbi ühe või mitme serveri, kuhu saavad liituda mitmed firmad, ning andmed on omavahel jagatavad. See võimaldab töötajal oma pädevusi ja nendega seotud tõestusmaterjale alles hoida ka pärast töökoha vahetust jms. Sellise tarkvara arenduse jaoks sobib, ning on praktikas kõige rohkem kasutaud, hajutatud arhitektuuriga veebirakendus. Veeb on kõigile üheselt kättesaadav ja võimaldab mugavat rakendusega suhtlust. Kompetentside ja kasutajaga seotud veebiteenused on eraldiseisvad ning eraldi kasutatavad. Arendajaid võib olla mitmeid, kes töötavad kõik eri osade kallal, kuid ühe standardi järgi, mis võimaldab osi omavahel ühildada.

Üks sellise arendusmudeliga projekt kannab nime TENCompetence[18], ja see on Euroopa Liidu poolt rahastatud 4-aastane projekt, mille eesmärgiks luua vastav infrastruktuur elukestva õppe ning pädevuste arenguks. Tulevikus saavad sellega liituda nii Euroopa üksikisikud kui ka organisatsioonid. Projekti aluseks on avatud lähtekoodiga ja standarditel põhinev tehnoloogia. Infrastruktuuri võimalused saavad olema järgmised: teadmiste jagamine, õpisuundade valik, hindamine, portfoolio ja teadmiste- ning õppekeskkondade toetamine. [19]
Projekti väljatöötamises ja arenduses osalevad mitmed ettevõtted ning ülikoolid üle Euroopa.

[18] http://www.tencompetence.org
[19] http://www.tencompetence.com

3.3 Pädevushalduse veebiteenused

Järgnevad on põhilised veebiteenused, mis on ühe lihtsama pädevushalduse tarkvara juures olulised. Teenused on jaotatud teemade järgi. Toon välja, mida teenus teeb, millised on sisendparameetrid ning milline on väljund.

Kompetentsid

- Teenus, mille välja kutsumisel kuvatakse kõik antud pädevushalduse valdkonda kuuluvad kompetentsid. Päringu esitamisel parameetreid pole vaja kasutada ning teenus tagastab nimekirja kompetentsidest koos nende juurde kuuluvate omadustega. Samuti on ära märgitud eelduseks olev kompetents, juhul, kui see eksisteerib.
- Teenus, mille välja kutsumisel kuvatakse kõik omadused mingi ühe kompetentsi kohta. Päringu parameetriks on otsitava kompetentsi unikaalne identifikaator, kas siis tekstilisel või numbrilisel kujul.

Töötaja/Kasutaja

- Teenus, mille välja kutsumisel kuvatakse kõik antud kasutaja olemasolevad kompetentsid ning nende tasemed.
- Teenus, mille välja kutsumisel kuvatakse antud kasutaja lisatud tõestusmaterjali või portfooliot ning näidatakse, millise kompetentsi juurde need käivad. Teenuse sisendiks on kasutaja unikaalne identifikaator.
- Teenus, mis võimaldab kasutajal tõestusmaterjali fotode ning dokumentide näol serverisse lisada.

Tarkvara juurutamise juures tuleks kindlasti rõhku pöörata sellele, et kõik organisatsiooni töötajad mõistaksid, mis on juurutatava projekti eesmärk ja ulatus, ning milline on firma suunitlus antud projekti puhul. See on üks põhilisi, kuid kõige tihedamini alahinnatud, osi pädevushalduse tarkvara arenduse juures. Tuleks esitada formaalne ülevaade pädevushaldusest, tarkvarast ning selle võimalustest, soovitavalt gruppideks jaotatult töötaja osakonna või tööülesannete järgi, olenevalt muidugi töötajaskonna suurusest. Väikese arvu töötajate puhul ei ole selline jaotamine vajalik. Pädevuste tutvustamise juures tuleks rõhku pöörata aspektidele, millel positiivsed tulemused nii töötaja kui organisatsiooni jaoks.[20]

[20] http://www.trainingreference.co.uk/learning_management_systems/competence_management.htm

Kokkuvõte

Antud seminaritöö eesmärgiks oli anda ülevaade pädevushaldustarkvarast ja kirjeldada selle jaoks sobivat ning enim kasutatavat veebitarkvara arhitektuuri. Töö peaks andma lugejale ülevaate tarkvara arhitektuurist ja selle ajaloost; sellest, mis on ja kuidas töötavad veebiteenused ja kuidas nende arendusele läheneda, kui ka sellest, mis on pädevused ning millest koosneb üks pädevushalduse tarkvara.

Käesolevat seminaritööd kirjutades tutvusin lähemalt ja sain palju uut teada tarkvara arhitektuuri ajaloost ja veebitarkvara arenduse metoodikatest. Samuti sain ülevaate sellest, kuidas töötavad veebiteenused ning millised tehnoloogiad on arenduseks olemas. Tööd kirjutama asudes polnud ma päris kindel, et ma mõistsin, mis on pädevused ning milline peaks olema pädevushaldus tarkvara ning mida see täpselt teeb. Materjale uurides jõudsin selgusele, et eduka ettevõtte personali puhul on kompetentsid ja kompetentsihaldus üheks oluliseks osaks.

Kavatsen antud tööd edasi kirjutada bakalaureusetöö raames, lisades töösse pädevushalduse arenduse praktilise poole ning praktika käigus kogetu.

Kasutatud kirjandus

1. Frederick, B. (1975); The Mythical Man-Month

2. Open SOA Collaboration. (URL)
   http://www.osoa.org/display/Main/Home (10.09.2010)

3. Service Component Architecture. (URL)
   http://www.ibm.com/developerworks/library/specification/ws-sca (16.09.2010)

4. Web Services Architecture. (URL)
   http://www.w3.org/TR/ws-arch/ (27.09.2010)

5. Chappell, D. (2007); Introducing SCA. (URL)
   http://www.davidchappell.com/articles/Introducing_SCA.pdf (16.09.2010)

6. RESTful Web Services. (URL)
   http://www.oracle.com/technetwork/articles/javase/index-137171.html (20.08.2010)

7. Cowan, J. (2005); RESTful Web Services. (URL)
   http://home.ccil.org/~cowan/restws.pdf (10.09.2010)

8. TENCompetence. (URL)
   http://www.tencompetence.com (15.10.2009)

9. Layered Application Guidelines. (URL)
   http://msdn.microsoft.com/en-us/library/ee658109.aspx (15.10.2010)

10. Kunzmann, C., Schmidt, A. (2006); Towards a Human Resource Development Ontology for Combining Competence Management and Technology-Enhanced Workplace Learning. (URL)
    http://publications.andreas.schmidt.name/schmidt_kunzmann_OntoContent06.pdf (10.09.2010)

11. Childers, J. (2003); Monolithic software decimates IT budgets (URL)
    http://www.linux.com/archive/feed/31530 (16.09.2010)