TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Arvutiteaduse instituut
Informaatika õppekava

Karl Kuusk

Teenuspõhine analüüsiprotsess ja -dokumentatsioon teenusorienteeritud arhitektuuriga süsteemi arendamiseks

Bakalaureusetöö (6 EAP)

Juhendaja: Vambola Leping, MSc

Tartu 2015

Teenuspõhine analüüsiprotsess ja -dokumentatsioon teenusorienteeritud arhitektuuriga süsteemi arendamiseks

Lühikokkuvõte:

Antud bakalaureusetöö eesmärk on anda ülevaade analüüsiprotsessist, mida saab kasutada teenusorienteeritud süsteemide arendamisel. Töö raames vaadeldakse analüüsiprotsessi juhul, kui arendatav süsteem ei ole teenusorienteeritud arhitektuuriga ning kõrvutatakse seda analüüsiprotsessiga, kus süsteem arendatakse teenusorienteeritud arhitektuuri põhimõtteid silmas pidades. Autor on lisanud näited dokumentatsiooni kohta, mida saab kirjeldatud analüüsiprotsessi toetamiseks kasutada.

Võtmesõnad:

SOA, teenusorienteeritud arhitektuur, analüüsiprotsess, dokumentatsioon, REST teenus

Service-oriented analysis process and documentation for developing system with service-oriented architecture

Abstract:

The aim of this Bachelor's thesis is to provide an overview of the analysis process for developing a system with service oriented architecture. The thesis compares two different analysis processes, one when service oriented design principles are applied and another when principles are not followed. Author has added examples of documentation that can be used to support the described analysis process.

Keywords:

SOA, service-oriented architecture, analysis process, documentation, REST service

Sisukord

1. Sissejuhatus

1.1 Taustainfo

Infoühiskonnas laienevad organisatsioonide süsteemid kasvavas tempos. Olemasolevate infosüsteemide funktsionaalsust on vaja pidevalt täiendada, et need täidaksid oma eesmärki. Nõudmised infosüsteemidele kasvavad: süsteem peab asendama võimalikult palju käsitsi tehtavast tööst ning sealjuures olema võimeline operatiivselt teiste süsteemidega infot vahetama.

Täna seisneb probleem seisneb selles, et monoliitse arhitektuuriga süsteemide edasiarendamine muutub peale 3 - 5 aastat väga ajamahukaks ja kulukaks [1]. Parandused ja väikesed muudatused on võimalik edukalt sisse viia, kuid laiapõhjaliste muudatuste sisseviimine, näiteks uue äriprotsessi lisandumine või mitme olemasoleva äriprotsessi ühtlustamine, võib osutuda olemasolevas süsteemis kallimaks kui nende muudatuste realiseerimine uues süsteemis.

Lahenduseks on teenusorienteeritud arhitektuuri (SOA[1]) kasutuselevõtmine. Teenusorienteeritud arhitektuuri põhimõtete järgi on süsteem jagatud teenusteks, mis on kujundatud vastavalt äriprotsessile. Äripõhine lähenemine süsteemile tagab, et muudatuste ilmnemisel äris on võimalik kiirelt ja kuluefektiivselt teha muudatusi ka süsteemis.

Teenusorienteeritud arhitektuur tagab suurema modulaarsuse ning äriloogika kapseldumise kindla teenuse raames. Kapseldatud äriloogika võimaldab teenuste edukat taaskasutamist nii süsteemi siseselt kui ka erinevate väliste süsteemide vahel. Organisatsioonidel, mis kasutavad paralleelselt mitut erinevat süsteemi, võimaldab teenusorienteeritud arhitektuuriga süsteemi kasutamine saavutada pikas perspektiivis olulist kuluefektiivsust. Esialgsete teenuste väljakujundamine on ajamahukam, kuid taaskasutuse tulemusena on võimalik saavutada oluline ajaline ja rahaline võit.

Traditsiooniliselt on süsteemianalüüsi meetodid olnud alati seotud süsteemide arendamiseks kasutatud tehnoloogiatega [2]. Ka SOA kasutuselevõtuga on vaja teha olulisi muudatusi analüüsimeetodites ja –protsessis. Analüüsiprotsess on algusfaasis oluliselt mitmetahulisem kui monoliitse arhitektuuriga süsteemide puhul ning nõuab rohkem äri tundmaõppimist ja laia pildi haaramist. Analüüsiprotsessi käigus tekkiv dokumentatsioon, mis monoliitse

[1] SOA- Service-oriented architecture

arhitektuuriga süsteemide raames on küllaltki kuvapõhine, muutub rohkem teenuspõhiseks ning seeläbi on dokumentatsiooni hulk ka väiksem.

1.2 Ülevaade

Käesoleva bakalaureusetöö raames annab autor esmalt ülevaate analüüsiprotsessist monoliitse arhitektuuriga süsteemi puhul ning toetab seda parema ülevaate saamiseks näidisdokumentatsiooniga.

Järgnevalt tuuakse esile erisused, mistõttu tuleb analüüsiprotsessile monoliitse- ja teenusorienteeritud arhitektuuri puhul erinevalt läheneda. Peatüki raames on kirjeldatud erinevaid teenuse liike, mida süsteemide raames kasutatakse, ning nende omavahelist hierarhiat.

Peatükis „Teenusorienteeritud süsteemide analüüs“ antakse ülevaade erinevatest analüüsistrateegiatest ja -protsessist, mida on võimalik teenusorienteeritud arhitektuuriga süsteemide analüüsimisel kasutada. Analüüsistrateegiatena vaadeldakse kolme enim erialakirjanduses esile toodud lähenemist: kitsamast-laiemaks strateegiat, laiemast-kitsamaks strateegiat ja agiilset (ingl. k *Agile*) strateegiat. Analüüsiprotsessina on lähemalt kirjeldatud autori nägemust analüüsistrateegiast, mis sobib teenusorienteeritud arhitektuuriga süsteemide analüüsimiseks. Analüüsiprotsessi kaks olulisemat alamosa on äriprotsesside kaardistamine ja selle tulemuste põhjal teenuste modelleerimine, sellest lähtuvalt keskendutakse just nende lahkamisele.

Analüüsiprotsessi täiel määral rakendamiseks on vajalik teatav dokumentatsiooni hulk. Töö lõpus on autoripoolne nägemus sellest, mis liiki dokumentatsiooni on vaja teenusorienteeritud süsteemide analüüsimiseks. Kõikide erinevate dokumentatsiooniliikide kohta on loodud fiktiivsed näited, mis lihtsustavad nende praktilist rakendamist.

2. Analüüsiprotsess monoliitse arhitektuuriga süsteemide raames

2.1 Monoliitse arhitektuuriga süsteemid

Käesoleva töö raames mõistetakse monoliitse arhitektuuriga süsteemide all süsteeme, mille arendamisel ei ole täiel määral rakendatud teenuseorienteeritud arhitektuuri põhimõtteid. Sisuliselt tähendab see olukorda, kus süsteemi teatud komponendi arendamisel või täiendamisel avaldab see kogu süsteemile laiemalt mõju. Süsteemi kasvades kasvab ka muudatuste sisseviimise ajakulu. Väike muudatus võib tähendada seda, et süsteemis tuleb see sisse viia mitmesse komponenti ning pärast muudatuste realiseerimist tuleb üle kõikide komponentide läbi viia regressioontestimine ehk testida üle kogu olemasolev funktsionaalsus.

2.2 Monoliitse arhitektuuriga süsteemide analüüs

Monoliitse arhitektuuriga süsteemi analüüsiprotsessi näitlikustamiseks vaadeldakse raamistikus Aranea arendatud süsteemi analüüsiprotsessi. Aranea raamistik on arendatud Eesti tarkvaraettevõtte AS Nortal poolt.

Aranea on avatud koodiga Java põhine Mudel-Vaade-Kontroller (ingl. k *Model-View-Controller*) raamistik, mis pakub tuge veebipõhiste süsteemiste loomiseks, taaskasutades graafilise kasutajaliidese üldist loogikat [3]. Raamistikus programmeerimisel kasutatakse JSP[2] tuge. JSP on tehnoloogia, mille abil pannakse serveri poolel kokku kasutaja arvutisse saadetav kuva [4].

AS Nortal'i näitel on üldistatud tasemel välja toodud, kuidas lähenetakse analüüsiprotsessile Aranea raamistikus süsteeme arendades. Vastavalt üldlevinud praktikale antakse kliendi poolt analüütikule mingil kujul äriline sisend. Sisend arutatakse koos kliendiga läbi ning täpsustatakse ebamäärased kohad. Lahenduse arutelu käigus mõtleb analüütik sellele, kuidas kirjeldatud äriprotsess erinevate kuvade peal kasutajale arusaadavaks teha ning milliste põhiliste andmeolemitega tegu on. Pärast esialgsete kokkulepete saavutamist loob analüütik graafilise prototüübi, mis vaadatakse kliendiga koos üle.

Kliendipoolsete täpsustuste sisseviimise järel alustab analüütik äriprotsessi disainifaasiga selle konkreetse äriprotsessi raames. Analüütiku ülesandeks on koostöös arendajaga luua

[2] JSP- JavaServerPages

andmemudel, et kõik äriprotsessi raames kliendi poolt kirjeldatud andmed saaks korrektselt salvestatud. Üldise andmemudeli loomise järel saab analüütik alustada lahenduse dokumenteerimisega.

Aranea raamistikus süsteemide arendamisel on põhiliseks piiranguks kuvapõhine lähenemine. Kuna kliendi arvutisse saadetav kuva pannakse kokku serveri poolel, ei ole võimalik kasutada kasutajaliidese raames dünaamilist lähenemist. See tähendab, et kui süsteemi kasutaja muudab kasutajaliideses teatud andmehulka, tuleb muudatuste salvestamiseks kogu kuva salvestada ning seejärel uuesti serveri poolt kasutaja arvutisse saata.

Eelmainitust lähtuvalt on analüüs enamasti realiseeritud kuvapõhiselt. Kuvapõhine dokumentatsioon sisaldab endas viiteid sellele, kuidas andmebaasis andmed salvestatakse ning milline on antud kuva funktsionaalsus.

Analüüsidokumentatsioonina luuakse dokument, mille raames on kirjeldatud nii salvestusreeglid kui ka kuva üldine funktsionaalsus. Lisas „II Monoliitse arhitektuuriga süsteemi analüüsidokument“ on näitena toodud fiktiivne analüüsidokument, mis kirjeldab lihtsa andmete salvestamise vormi funktsionaalsust. Kuva keerukuse kasvamisel suureneb märgatavalt ka dokumentatsiooni maht. Tähelepanuväärne on see, et sisuliselt iga kuva kirjeldamisel tuleb eraldi läbi mõelda salvestusreeglid. Selline lähenemine tähendab iga kuva kirjeldamisel eraldi ajakulu täpsete salvestusreeglite läbimõtlemisele. Lisaks kuulub iga kuvapõhise analüüsidokumendi juurde analüüsi algusfaasis loodud prototüüp (Lisa „III Prototüüp“).

3. Analüüsiprotsessi muutumine

3.1 Analüüsi muutunud paradigma

Paradigma muutus lähtub sellest, et teenuspõhise arhitektuuriga süsteemide puhul on funktsionaalsus väga rangelt teenusesse kapseldatud. Analüüsi tehes ei saa läheneda konkreetse äriprotsessi raames, vaid tuleb tähelepanu pöörata laiemale pildile, et taaskasutatavad teenused oleksid korrektselt kapseldatud ning realiseeritud nii, et neid oleks võimalik süsteemi üleselt üheselt kasutada.

Monoliitse arhitektuuriga süsteemide analüüsimisel on süsteemianalüütiku ülesandeks luua detailne andmemudel ning selle raames keskenduda sellele, kuidas kuvapõhiselt andmete salvestamine käib. Teenusorienteeritud arhitektuuriga süsteemide puhul on analüütikul vaja mõelda abstraktsemalt äriobjektide tasemel. Äriobjektide salvestamine andmebaasi ning nende sealt pärimine käib REST[3] teenuste vahendusel. Seega on analüütiku põhiliseks vastutusalaks teenuste disainimine, andmebaasi salvestamise üksikasjad on teenuse arendaja vastutusalas.

Ärireeglid, mis monoliitse arhitektuuriga süsteemi analüüsi puhul on kirjeldatud kuvapõhiselt, tuleb ära kirjeldada teenuse tasemel. Kuna äriloogika on kapseldatud teenusesse, on seda hiljem võimalik süsteemi üleselt taaskasutada. Sealjuures on äriloogika teenuse sees varjatud kujul, mis lihtsustab oluliselt taaskasutamist, kuna teenust kasutusele võttes ei pea süüvima teenuse äriloogika üksikasjadesse, vaid saab eeldada, et see töötab vastavalt kirjeldatud reeglitele. Sama teenust on võimalik pakkuda ka välistele süsteemidele ilma, et peaks teistele osapooltele teenuse pakkumiseks märgatavat lisafunktsionaalsust ehitama.

3.2 Äriobjektid

Äriobjektid (ingl. k *business entity*) on teenusorienteeritud süsteemide analüüsimisel ja arendamisel võtmekohaks. Analüüsiprotsessi alguses on oluline tuvastada põhilised äris kasutatavad objektid (tavapäraselt näiteks klient, arve, tellimus jne).

Suurte organisatsioonide puhul on vajalik kokku leppida ka kasutatav ontoloogia. Ontoloogia määrab ära, milliseid andmeolemeid organisatsioonis kasutatakse ning kuidas need omavahel seotud on [5]. Juhul kui ontoloogia jääb ühtlustamata on oht, et organisatsiooni sees kasutatakse samade äriobjektide kirjeldamiseks erinevate osakondade

[3] REST- Representational State Transfer

ja inimeste raames erinevaid mõisteid. Sellega seoses võib tekkida olukord, kus sama äriobjekti kohta realiseeritakse mitu üksteist dubleerivat teenust.

3.3 REST teenus

Teenusorienteeritud süsteemide arendamisel kasutatakse REST teenuseid, mis on kasutusel andmete vahendamiseks. REST teenused on saanud populaarseks, kuna need on lihtsam alternatiiv SOAP'il[4] ja WSDL'il[5] põhinevatele teenustele. REST teenused kasutavad andmete liigutamiseks HTTP standardseid käske GET, POST, PUT, DELETE, mille abil on võimalik läbi veebibrauseri andmeid pärida, salvestada ja kustutada [6].

GET teenused tagastavad vastavalt etteantud parameetritele kuvamiskihile kindla andmehulga. PUT ja POST teenuseid kasutatakse andmete salvestamiseks ja uuendamiseks. DELETE teenuse abil on võimalik andmeid kustutada.

REST teenuste olulised eelised on :

- Olekuvabadus – teenused ja nende väljakutsumine on üksteisest sõltumatud. Iga teenus käivitatakse konkreetse väljakutse peale.
- Vahemälus hoidmise võimalus – teenuse kliendil on võimalik konkreetsete teenuste vastused vastavalt vajadusele vahemällu salvestada. Vahemälus andmete hoidmine suurendab oluliselt süsteemi jõudlust.
- Klient-Server eraldatus – teenuse klient, mis teenust kasutab (kuvamiskiht), ja server, mis teenust pakub, on täielikult eraldatud. Seega, kui teenusega liigutatava andmestiku struktuur on eelnevalt kokku lepitud, siis on võimalik kliendi ja serveri poolt eraldiseisvalt arendada. Sellise lähenemise abil on võimalik arendajate tööd paremini planeerida ja ühtlasemalt jaotada.
- Skaleeritavus – korralikult läbimõeldud teenust on võimalik kergesti skaleerida ehk seda on võimalik korraga kasutada paljudel osapooltel.

3.4 Teenuste liigid

Teenused jaotuvad oma sihtotstarbe järgi erinevateks alamgruppideks. Tõeline teenusarhitektuur tähendab olemuselt, et kõrgema taseme teenused saavad taaskasutada alama taseme teenuseid ning seeläbi on vaja kirjutada oluliselt vähem koodi [7].

[4] SOAP- Simple Object Access Protocol
[5] WSDL- Web Service Definition Language

Teenused võib jagada kolme suuremasse alamgruppi [8]:

- äriobjekti põhised teenused (ingl. k *Entity services*)
- tegevustel põhinevad teenused (ingl. k *Task services*)
- üldised teenused (ingl. k *Utility services*)

3.4.1 Äriobjekti põhised teenused

Äriobjekti põhiste teenuste raames luuakse teenused eelnevalt defineeritud äriobjektide pärimiseks ja salvestamiseks. Sellised teenused on teenusmudeli mõistes kõige madalamal tasemel. Kuna samad äriobjektid on enamasti kasutusel mitmes erinevas äriprotsessis, on need teenused väga suure taaskasutamise potentsiaaliga [9]. Laia ulatuse tõttu nõuavad äriobjekti põhised teenused väga suurt investeeringut eelanalüüsi jaoks, mistõttu on selliste teenuste arendamine väga ajamahukas ja kulukas.

3.4.2 Tegevustel põhinevad teenused

Tegevuspõhised teenused

Tegevuspõhiste teenuste olemuseks on realiseerida mingit äriprotsessi või selle osa [10]. Enamasti on sellised teenused küllaltki äriprotsessi spetsiifilised ning taaskasutamise võimalused on piiratud. Kui tegevuspõhiste teenuste realiseerimisel kasutatakse äriobjekti põhiseid teenuseid, annab see märku tasemel teenusarhitektuurist.

Protsessipõhised teenused

Protsessipõhised teenused (ingl. k *Orchestrated task services*) on teenused, mis realiseerivad kõige kõrgemal tasemel protsesside ja tegevuste koordineerimist. Sisuliselt juhivad protsessipõhised teenused tegevus- ja äriobjekti põhiste teenuste kasutamist. Protsessipõhised teenused on näiteks asünkroonsed tegevused ja protsessivoo koordineerimine [11].

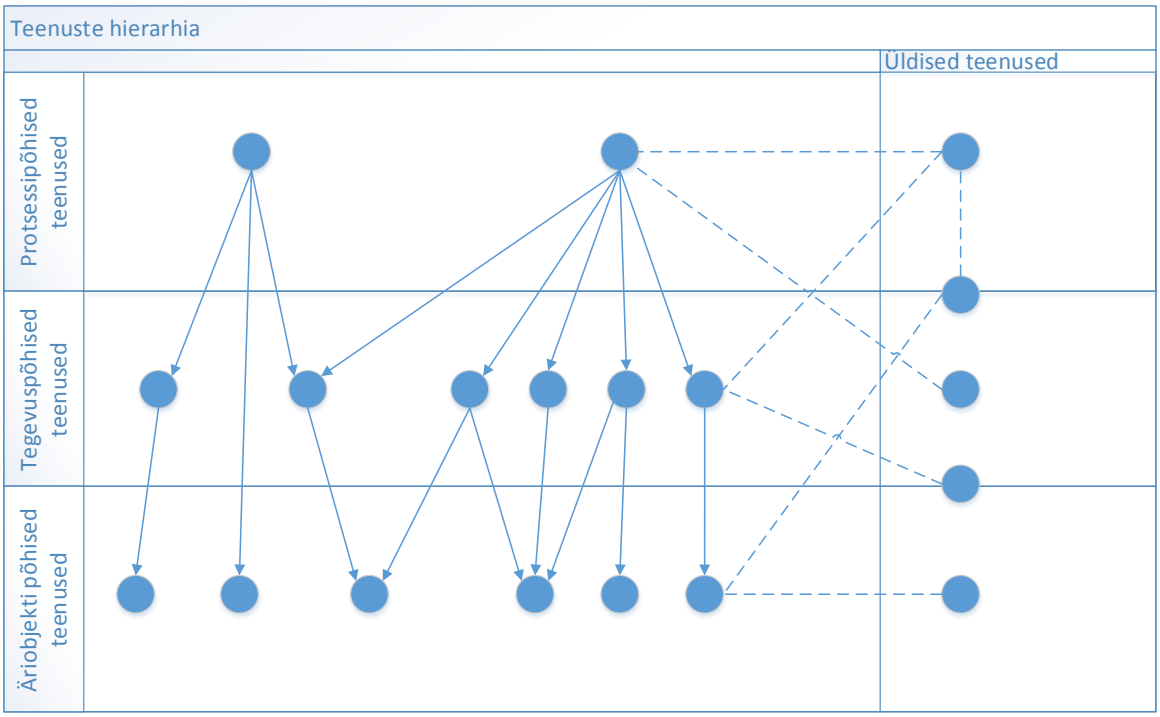
3.4.3 Üldised teenused

Üldised teenused on teenused, mille funktsionaalsus on teha midagi kindlat äriprotsesside üleselt. Alati ei ole vaja siduda teenust kindla protsessi või äriobjektiga, seetõttu on vajadus ka selliste teenuste järgi [12]. Tavaliselt on sellisteks teenusteks näiteks sündmuste logimine, nendest teatamine, täiendavate sisutekstide kuvamine või vigade haldamine.

Sellised üldist infrastruktuuri pakkuvad teenused on tihipeale äärmiselt kasulikud, kuna neid kasutavad sisuliselt kõik ülejäänud teenused.

3.4.4 Ülevaade teenuste hierarhiast

Joonisel 1 on kujutatud eelmainitud teenused. Joonise pealt nähtub selgelt erinevate teenuste hierarhiline loogika ja taaskasutamise võimalused.



Joonis 1. Teenuste hierarhia.

4. Teenusorienteeritud süsteemide analüüs

4.1 Teenusorienteeritud süsteem

Teenusorienteeritud süsteemina mõistame käesoleva töö raames süsteemi, mis järgib teenusorienteeritud arhitektuurimudelit. Arhitektuurimudeli raames pannakse põhirõhk teenuste kujundamisele ja realiseerimisele süsteemis ning seeläbi peaks kasvama süsteemi kasutava organisatsiooni efektiivsus, agiilsus ja produktiivsus [13]. Arhitektuurimudeli realiseerimisel saab kasutada väga erinevaid tehnoloogiaid ja tooteid.

Sõltumata kasutatavast tehnoloogiast, tuleb teenusorienteeritud süsteemi realiseerimisel järgida üldiseid põhimõtteid [14]:

1. Standardiseeritud teenuse kirjeldus – teenuste ülesehitus ja andmetüübid peavad olema standardsed, et tagada ühtsus ja kasutatavus.
2. Teenused on sõltumatud – teenused peavad olema võimalikult granulaarsed ja seeläbi üksteisest sõltumatud, misläbi ei teki ilma põhjuseta sisulisi seoseid.
3. Teenused on abstraktsed – teenused peavad peitma võimalikult palju sisalduvat äriloogikat, et neid oleks lihtne kasutusele võtta.
4. Teenused on taaskasutatavad – teenused peavad olema realiseeritud nii, et neid oleks võimalik süsteemi üleselt taaskasutada.
5. Teenused on autonoomsed – teenused peavad omama kontrolli kasutatava loogika ja andmete üle, misläbi on tagatud teenuse usaldusväärsus.
6. Teenused on olekuvabad – teenus peab olema realiseeritud nii, et ta töötaks olekuvabalt, seeläbi tekib olukord, kus teenus on alati kättesaadav ning seda on lihtne skaleerida.
7. Teenused on ilmutatud kujul – teenused peavad olema lihtsasti avastatavad, mis tagab, et neid taaskasutatakse.
8. Teenused on lihtsalt liidetavad – teenuseid peab olema võimalik lihtsasti koos kasutusele võtta, seeläbi on keerulise äriloogika puhul võimalik kasutada erinevate teenuste kompositsioone.

4.2 Analüüsistrateegiad

Teenusorienteeritud süsteemi analüüsimisel on kasutusel laiemast-kitsamaks (ingl. k *Top-down analysis*) ja kitsamast-laiemaks (ingl. k *Bottom-up analysis*) strateegia. Kolmanda strateegiana tuuakse välja agiilne meetod (ingl. k *Agile analysis* või *Meet-in-the-middle*),

mis on kombinatsioon eelmainitud strateegiatest. Konkreetse strateegia valik, mida peaks analüüsimisel järgima, sõltub paljuski organisatsiooni eesmärkidest ning sellest, kas püüeldakse pikaajaliste eesmärkide poole või pannakse rõhk lühiajaliste nõuete realiseerimisele [15]. Sealjuures on valikukriteeriumitena olulised veel saadaolev rahaline ressurss ja süsteemi plaanitav kasutusiga. Mida pikem on eeldatav kasutusiga, seda rohkem tuleks kalduda laiemast-kitsamaks strateegia poole.

4.2.1 Kitsamast-laiemaks strateegia

Kitsamast-laiemaks analüüsi raames luuakse teenused vajaduspõhiselt, täitmaks kliendi otseseid ärilisi vajadusi. Esmajoones pööratakse tähelepanu nendele teenustele, mis on kliendi jaoks aegkriitilised. Rakendatava meetodi puhul viidatakse SOA-le, kuid tegelik arhitektuur on lähedal traditsioonilisele monoliitsele arhitektuurile, kuna tähelepanuta jäetakse pikaajalised eesmärgid ja teenuste taaskasutamise potentsiaal. Strateegia raames kogutakse esmalt kokku kliendi nõuded, nende põhjal modelleeritakse teenused, arendatakse need valmis ning tarnitakse (Joonis 2).

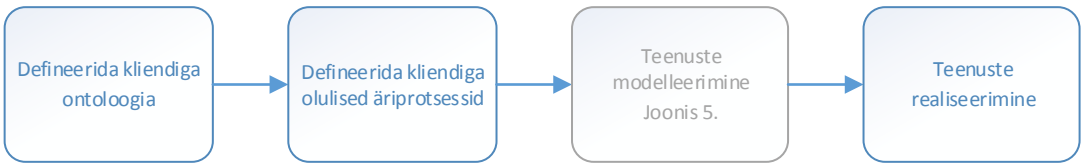
Joonis 2. Kitsamast-laiemaks strateegia.

Kitsamast-laiemaks analüüsi ning arenduse suureks eeliseks on kiire funktsionaalsuse realiseerimine. Kiire realiseerimise käigus ei jõuta kõikidele nüanssidele tähelepanu pöörata, ning seeläbi on oht, et funktsionaalsuse kasvades vajab süsteem refaktoreerimist. See on ühtlasi ka suurim oht, millega strateegia kasutamisel arvestada tuleb.

4.2.2 Laiemast-kitsamaks strateegia

Laiemast-kitsamaks analüüsi puhul analüüsitakse süsteemi koos äriga ning selle põhiselt. Oluliselt suuremat rolli mängib ärianalüüs, kuna süsteemi analüüsimise käigus tuleb enamasti optimeerida ka äriprotsesse. See analüüsistrateegia on märgatavalt strateegilisem ja toetab seeläbi kliendi pikaajalisi eesmärke. Seonduvalt esialgse suure pildi läbitöötamisega nõuab see lähenemine pikemaajalist eelanalüüsi. Strateegia raames tuleb esmalt koostöös kliendiga leida ühine ontoloogia ja kaardistada olemasolevad äriprotsessid. Äriprotsesside kaardistamisel on oluline, et sarnased äriprotsessid saaksid võimaluse piires ühtlustatud. See nõuab aga küllaltki vaba äriprotsessi, mis on omane erasektorile. Avaliku

sektori äriprotsessid on reeglina seadustega fikseeritud ning neid ei ole võimalik nii lihtsalt muuta. Äriprotsessidele eraldi tähelepanu pööramine tagab, et süsteemi raames realiseeritakse vastavad protsessid võimalikult sarnaselt ärile endale [16]. Pärast protsesside kaardistamist tuleb nende põhjal modelleerida, realiseerida ja tarnida vastavad teenused. Joonisel 3 on strateegia visualiseeritud kujul.

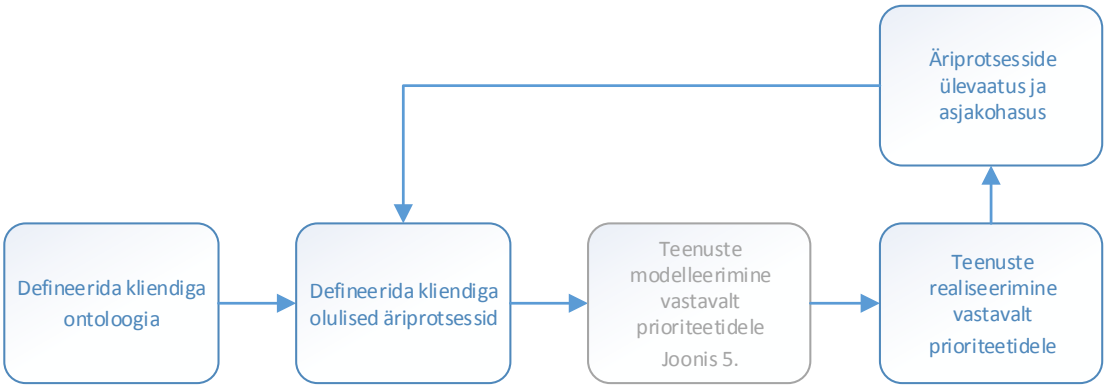


Joonis 3. Laiemast-kitsamaks strateegia.

Laiemast-kitsamaks strateegia rakendamise puhul võib olulise eelisena välja tuua põhjalikkuse ja kuluefektiivsuse pikas perspektiivis. Puhtal kujul laiemast-kitsamaks strateegia rakendamist takistavad organisatsioonide ajalised ja rahalised piirangud. Põhjalik eelanalüüs nõuab aga aega ja suurt eelinvesteeringut. Strateegia suurimaks riskiks on üleplaneerimine ja liiga sügavuti detailidesse laskumine. See võib põhjustada olukorra, kus analüüsi valmides ei ole see enam ajakohane.

4.2.3 Agiilne strateegia

Organisatsioonidele ei ole puhtal kujul laiemast-kitsamaks analüüsiprotsess tihtipeale vastuvõetav [17]. Alternatiivina kasutakse lähenemist, kus laiemast-kitsamaks strateegiast on kasutusel üldine äriprotsesside kaardistamine ning paralleelselt realiseeritakse erinevate äriprotsesside funktsionaalsust vastavalt prioriteetidele, mis olemuselt kuulub kitsamast-laiemaks strateegia juurde. Pärast iga teenuste realiseerimise iteratsiooni vaadeldakse uuesti üle äriprotsessid ning pannakse paika järgnevad prioriteedid (Joonis 4).



Joonis 4. Agiilne strateegia

4.3 Analüüsiprotsess

Teenusorienteeritud arhitektuuriga süsteemide analüüsiprotsess võib sõltuvalt arendatavast süsteemist ja seda arendavast tarkvaraettevõttest olla väga erineva keerukustasemega. Analüüsiprotsessi raames saab kasutada erinevaid lähenemisi ning seeläbi võib ka analüüsi detailsuse tase kordades erineda.

Sõltumata eelnevalt kirjeldatud taustast, on analüüsiprotsessi esimeseks oluliseks sammuks olemasolevate äriprotsesside kaardistamine ning nende optimeerimine. Sellele järgneb kaardistamiste tulemusel tekkinud tulemite põhjal teenuste modelleerimine. Autor on käesolevas bakalaureusetöös tähelepanu pööranud just neile põhilistele protsessidele.

4.3.1 Äriprotsesside kaardistamine

Äriprotsesside kaardistamise esimeseks sammuks on ontoloogia kokkuleppimine, mida on kirjeldatud peatükis „3.2 Äriobjektid“. Ontoloogia kokkuleppimise raames on mõistlik luua äriobjektide kohta ka esialgne dokumentatsioon (kirjeldatud peatükis „4.4.1 Äriobjekti kirjeldus“), mida saab analüüsiprotsessi käigus jooksvalt täiendada. Järgmise etapina on oluline kaardistada olemasolevad äriprotsessid.

Äriprotsesside kaardistamise üheks olulisemaks eesmärgiks on luua dokumentatsioon ja ühtne pilt olemasolevate ärivajaduste kohta. Protsesside kaardistamine annab ülevaate, kes on vastutav kirjeldatud tegevuste eest, kuidas ja millal neid tegevusi peaks esile kutsuma, millised on põhilised äriobjektid ning kuidas andmed protsessi käigus muutuvad [18]. Teise olulise aspektina on mudel alusmaterjaliks, mille alusel saab protsesse automatiseerida ja optimeerida.

Kaardistamise lihtsustamiseks on mõistlik kasutada spetsiaaltarkvara, näiteks Microsoft Visio[6], Enterprise Architect[7] või Bizagi BPM[8]. Vastava tarkvara abil on võimalik äriprotsessid kiirelt ja lihtsalt visuaalselt modelleerida (täpsemalt kirjeldatud peatükis „4.4.2 Protsessiskeem“).

Pärast esialgset kaardistamist tekib visuaalne protsessiskeem, mis on analüütikule ja kliendile üheselt arusaadav. Juhul kui kasutatakse agiilset või laiemast-kitsamaks strateegiat, kus kaardistatakse korraga rohkem kui üks äriprotsess, siis on oluline, et järgmise sammuna võetakse ette äriprotsesside vaheliste kattuvuste leidmine. Kui

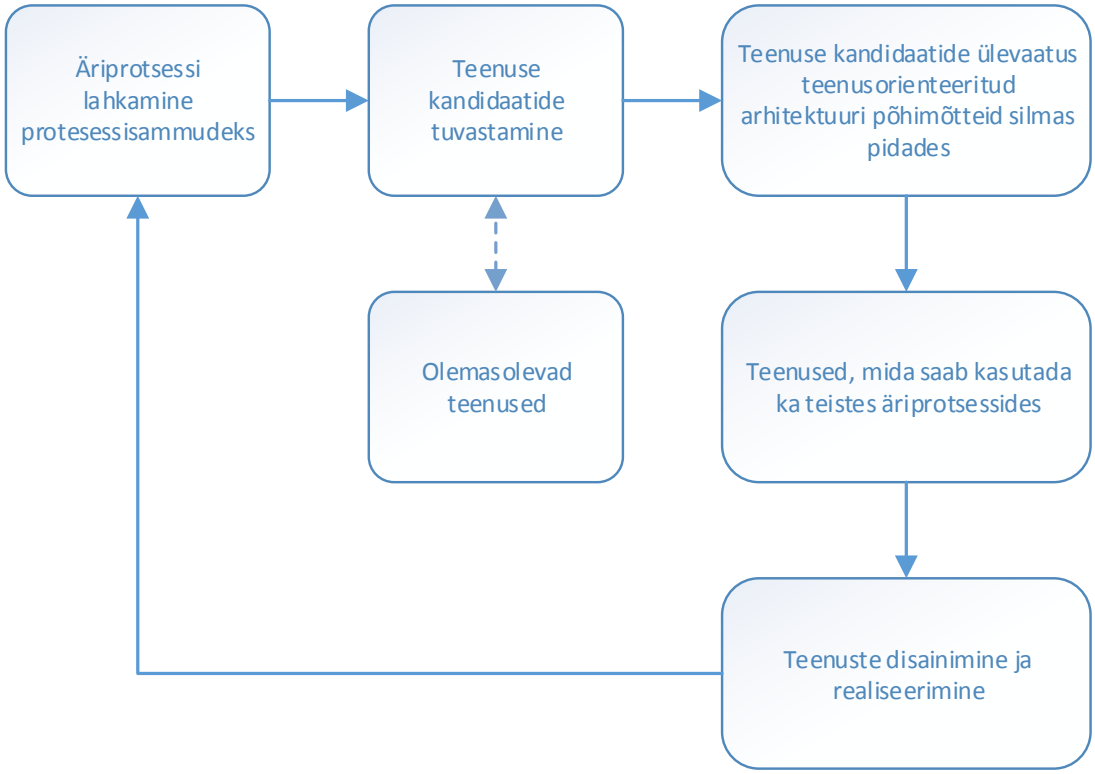
[6] Microsoft Visio- https://products.office.com/en-us/visio/flowchart-software
[7] Enterprise Architect- http://www.sparxsystems.com/products/ea/
[8] Bizagi BPM- http://www.bizagi.com/

äriprotsesside kattuvus on väga suur, tuleks kaaluda nende protsesside ühtlustamist. Teatud juhtudel pole protsesside ühtlustamine võimalik või otstarbekas. Sel juhul tuleb kattuv protsessilõik kirjeldada eraldi äriprotsessina. Sellisel kujul abstraheerimine tagab, et hilisema teenuste modelleerimise käigus on lihtne kujundada tegevuspõhiseid teenuseid, mida kasutatakse mitme erineva protsessi raames.

4.3.2 Teenuste modelleerimine

Teenuste modelleerimine on analüüsiprotsessi kõige olulisem osa, kuna korralikult läbimõeldud teenusarhitektuur tagab, et süsteem järgib teenusorienteeritud arhitektuuri üldiseid põhimõtteid. See omakorda tähendab, et süsteemi ülesehitus toetab ärilisi eesmärke.

Teenuste modelleerimise üldine skeem on esitatud Joonisel 5.



Joonis 5. Teenuste modelleerimine.

Äriprotsessi lahkamine protsessisammudeks

Esimese sammuna tuleb ette võtta eelnevalt kaardistatud äriprotsessid ning lahutada need detailselt protsessisammudeks. Reeglina on äriprotsess üldisem ning üks protsessisamm äriprotsessis võib tähendada mitut sammu teenuse modelleerimise mõistes.

Teenuse kandidaatide tuvastamine

Äriprotsesside mõistlikeks sammudeks jagamise järel tuleb tuvastada võimalikud teenusekandidaadid. Teenusekandidaatide tuvastamisel on abiks, kui mõelda, milliseid andmeid on vaja antud protsessisammu raames modifitseerida või salvestada.

Põhiline teenuste modelleerimise eesmärk on välja selgitada, milliseid teenuseid on vaja järgneva teenuste realiseerimise faasi ajal realiseerida. Seetõttu on oluline meeles pidada, et modelleerimise faasi vältel ei tegeleta veel teenuste realiseerimisega, vaid analüüsitakse abstraktseid teenuse kandidaate, millest kõik ei pruugi saada lõpuks realiseeritud [19].

Teenuse kandidaate tuleb vaadelda võimalikult abstraktsel tasemel, kuna sisulist analüüsi tehes võib juhtuda, et konkreetne teenus ei lähe realiseerimisele ning seeläbi on ka kulutatud analüüsiaeg raisku läinud.

Olemasolevad teenused

Juhul kui tegemist ei ole esimese iteratsiooniga, mille raames teenuseid modelleeritakse, tuleb tähelepanu pöörata ka olemasolevatele teenustele. Juba realiseeritud teenuste jälgmisega on tagatud, et olemasolevaid teenuseid ei dubleeritaks. Olemasolevate teenuste komplektist ülevaate saamiseks on otstarbekas kasutada süsteemi koodi pealt genereeritud teenuste struktuurset kirjeldust, mida on täpsemalt kirjeldatud peatükis „4.4.5 REST API“.

Teenuse kandidaatide ülevaatus

Teenuse kandidaatide väljaselgitamisele järgneb nende valideerimine, toetudes teenusarhitektuuri üldisele põhimõtetele, millele kõik teenused vastama peavad. Eraldi tasub tähelepanu pöörata taaskasutatavuse, autonoomsuse ja abstraktsuse põhimõtetele. Teenuse taaskasutatavuse maksimeerimiseks tasub vajadusel kaasata liiaga andmeid, mis tagab, et sama teenust on võimalik süsteemiüleselt kasutada. Eriti oluline on seda silmas pidada äriobjekti põhiste teenuste puhul. Autonoomsuse all tuleb silmas pidada, et teenus käsitleks mingit kindlat äriobjekti või äriprotsessi ning ei oleks kattuvusi teiste teenuste kandidaatidega. Samuti tasub jälgida, et teenus oleks kujundatud abstraktselt ehk teenus peidaks võimalikult palju sisalduvast äriloogikast. See lähenemine tagab, et teenuse kasutuselevõtmine mõne teise äriprotsessi raames on võimalikult lihtne.

Teenused, mis ei täida kirjeldatud põhimõtteid, tuleb uuesti üle vaadata ning vajadusel teenuse skoopi laiendada või kitsendada. Juhul kui skoobi muutmine ei ole võimalik ning

teenuse valmimine ei ole ajakriitiline, siis tuleks teenus jätta kandidaadi tasemele ning tulla selle juurde tagasi järgmise modelleerimisiteratsiooni raames.

Teenuste taaskasutatavus teiste äriprotsesside raames

Teenuse kandidaate tasuks järgnevalt vaadata koostöös teiste äriprotsessidega. Võib nähtuda, et mõne teise äriprotsessi raames on vaja analoogset teenust. Enamasti on sellisteks äriobjekti põhised- ja üldised teenused. Selliste teenuste ilmnemisel tasub need jätta kandidaatide tasemele ning tulla nende juurde tagasi selle äriprotsessi raames, kus need samuti kasutusel on. See tagab, et teenus on ühtne ning katab mõlema äriprotsessi vajadused täies ulatuses.

Kui teenuse realiseerimine on ajakriitiline ehk ilma seda realiseerimata ei ole võimalik edasi minna, siis tasub põgusalt üle vaadata teine äriprotsess, mis seda teenust kasutab, ja selle detailsemad vajadused ning need antud teenuse skoobi sisse planeerida. Juhul kui teise äriprotsessi vajadused on ebaselged, siis on otstarbekas luua teenus liiaste andmetega.

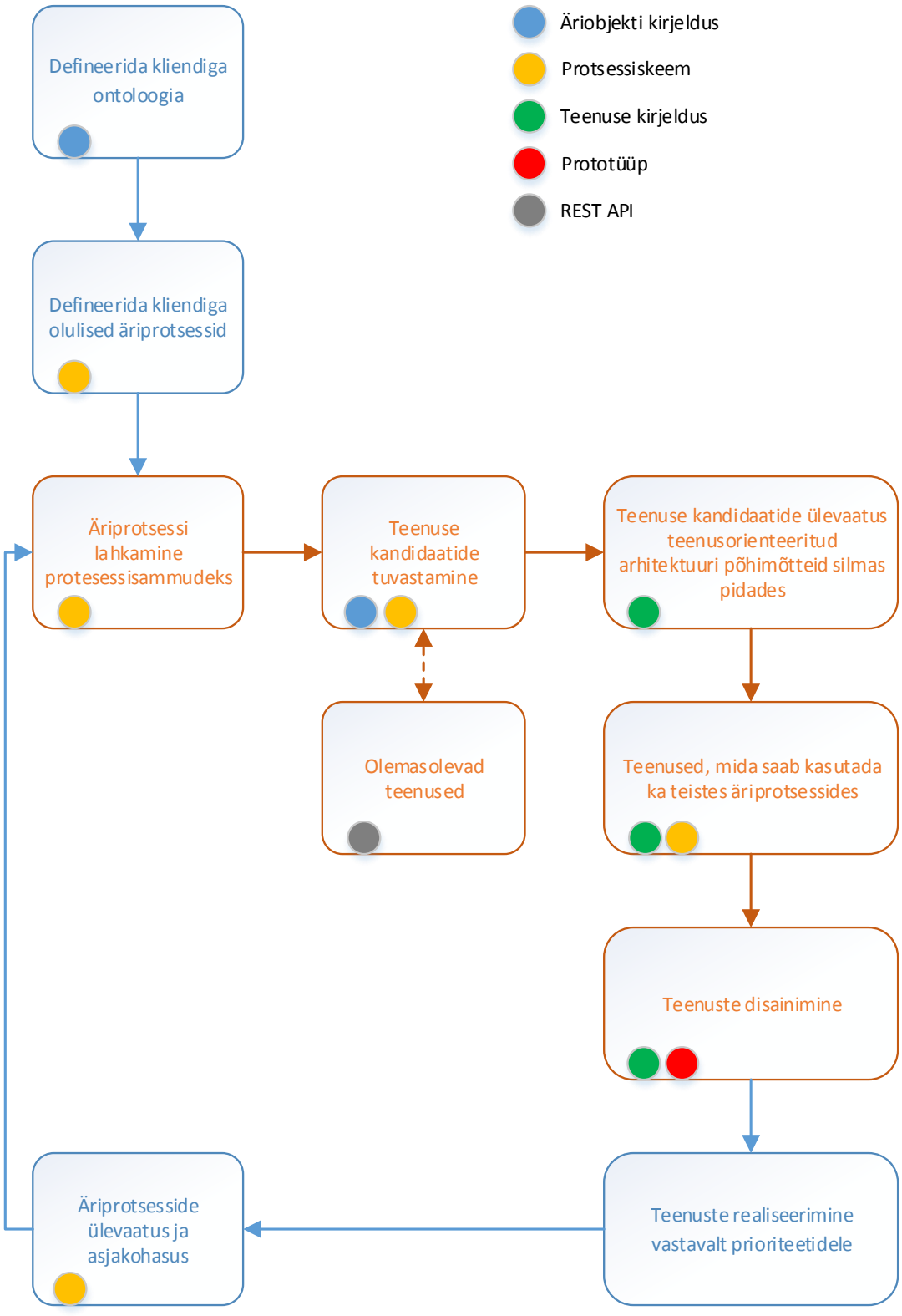
Teenuste disainimine ja modelleerimine

Teenuste modelleerimise viimaseks sammuks on välja valitud teenuste disainimine ja nende realiseerimisse suunamine. Disainifaasi raames tuleb detailsemalt tähelepanu pöörata teenusega liigutatavatele andmetele ja kuvadele, mis neid teenuseid kasutama hakkavad. Tagamaks teenuse kohaldumise kuvaga, tasub disainifaasis joonistada süsteemi kuvadele esialgsed prototüübid.

Prototüübi joonistamisel nähtub, kui palju andmeid on vaja konkreetsel kuval välja näidata ning kuidas need kuval paigutuvad. Sellest tulenevalt saab täpsustada teenuste disaini. Näiteks nimekirjade realiseerimisel on teenusorienteeritud arhitektuuriga süsteemidel olulised piirangud. Kui teenusorienteeritud arhitektuuriga süsteemist päritakse nimekirja suur andmehulk, siis vaikimisi saadetakse kogu see andmehulk korraga kasutaja veebibrauserisse, mis võib põhjustada selle kokkujooksmise. Samas kui selle aspektiga teenuse disainimisel arvestada, siis saab vastava REST teenuse ehitada nii, et andmeid tagastatakse kasutaja arvutisse fikseeritud alamhulga kaupa ehk pagineeritakse vastus (ingl. k *Pagination*).

4.4 Dokumentatsioon teenusorienteeritud süsteemi arendamiseks

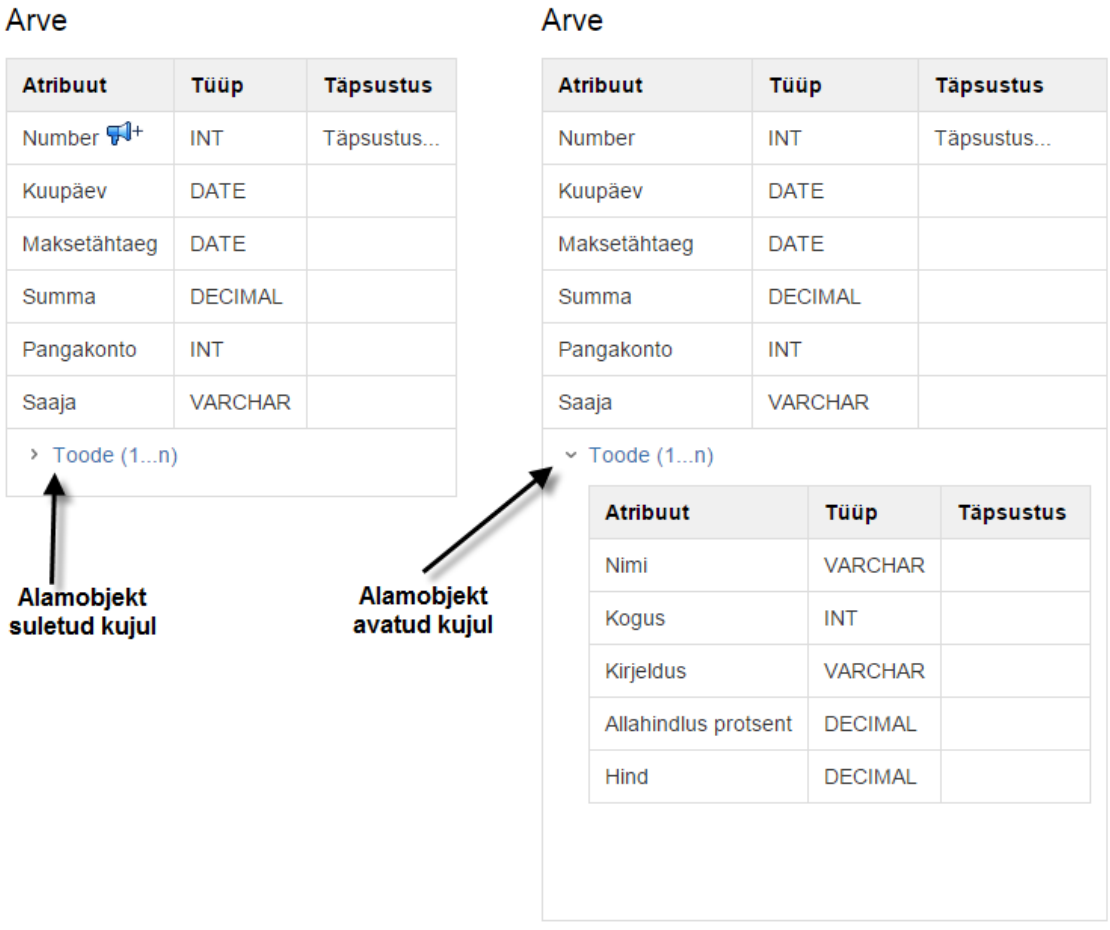
Antud peatükis on välja toodud autori arvates minimaalne dokumentatsiooni komplekt, mille abil on võimalik teenusorienteeritud süsteemi analüüsi läbi viia. Iga dokumentatsiooni elemendi kohta on toodud näited. Alljärgnevalt on esitatud täielik analüüsi protsessiskeem, mis kirjeldab seda, millises analüüsietapis mingit kindlat dokumentatsioonielementi enamasti kasutatakse (Joonis 6). Jooniselt nähtub, et dokumentatsioonielemendid ei kuulu alati selgelt mingi kindla analüüsiprotsessi sammu juurde. Selline pilt peegeldab reaalset pilti tarkvaraarendusest, kus analüüsiprotsess ei ole sirgjooneline ja fikseeritud ning dokumentatsioon on pidevalt ajas muutuv.



Joonis 6. Analüüsi protsessiskeem.

4.4.1 Äriobjekti kirjeldus

Äriobjekti kirjeldus on äärmiselt oluline sisend teenuse realiseerimiseks. Esimene ülevaade äriobjektidest ja nende põhilistest atribuutidest tuleb tekitada juba ontoloogia kokkuleppimise käigus. Analüüsiprotsessi käigus saab äriobjektide nimekirja ja täpseid andmeid täpsustada. Äriobjekti täielik kirjeldus peab olema valmis hiljemalt teenuse disainifaasiks. Äriobjekti ja tema atribuutide kirjeldamiseks sobivad kõik tarkvarad, mille abil on võimalik andmeid struktuurselt tabeli kujul esitada. Äriobjekt on enamasti mitmetasemeline ehk sellel on alamobjektid. Mitmetasemeliste äriobjektide kirjeldamiseks on otstarbekas kasutada tarkvara, mis võimaldab alamobjektide avamist ja sulgemist (Joonis 7). Näitena on toodud fiktiivne äriobjekt „Arve“, mis on kirjeldatud kasutades tarkvara Confluence[9].

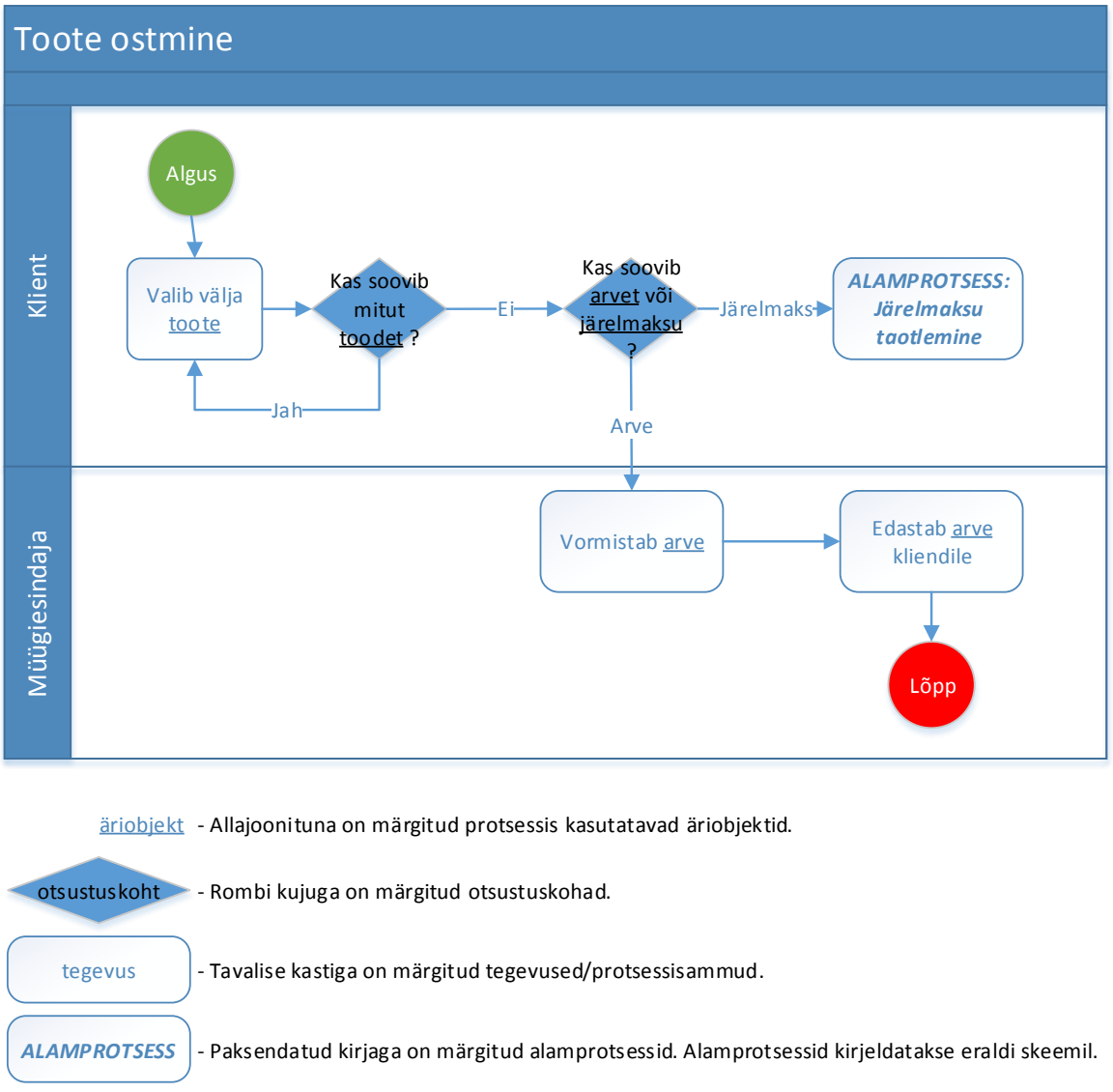


Joonis 7. Äriobjekti kirjeldus.

[9] Confluence- https://www.atlassian.com/software/confluence

4.4.2 Protsessiskeem

Protsessiskeem on ülevaatlik skeem äriprotsessist, mille raames on kirjeldatud protsessis osalevad tegutsejad (ingl. k *Actors*), kasutusel olevad äriobjektid, tehtavad tegevused ning otsustuskohad. Juhul kui protsessid on keerulised, on mõistlik kasutada mitmetasandilisi protsessiskeeme, kus kõrgema taseme skeemid annavad edasi üldist protsessi ning iga protsessisamm võib endast kujutada alamprotsessi. Näitena toodud protsessiskeem on loodud fiktiivse protsessi põhjal, mis kajastab toote ostmisprotsessi (Joonis 8). Protsessiskeemi joonistamiseks on kasutatud tarkvara Microsoft Visio.



Joonis 8. Protsessiskeem.

4.4.3 Teenuse kirjeldus

Teenuse kirjeldus annab detailse ülevaate teenuse funktsionaalsusest. Kirjelduses on märgitud äriloogika, mis teenuse sees peitub ning lisaks on esile toodud erisused ning aspektid, millele teenust realiseerides tähelepanu pöörata tuleb. Teenuse kirjelduse jaoks pakub töö autor välja üldise malli (Näide 1), mille raames on kirjas oluline sisendinfo arendamise eelduseks. Malli raames on kirjeldatud fiktiivset teenust, mille abil saab pärida arve andmed. Teenuse kirjelduse mall võib sõltuvalt kasutatavast tehnoloogiast ja arendatava süsteemi erisustest olla erinev autori poolt pakutust.

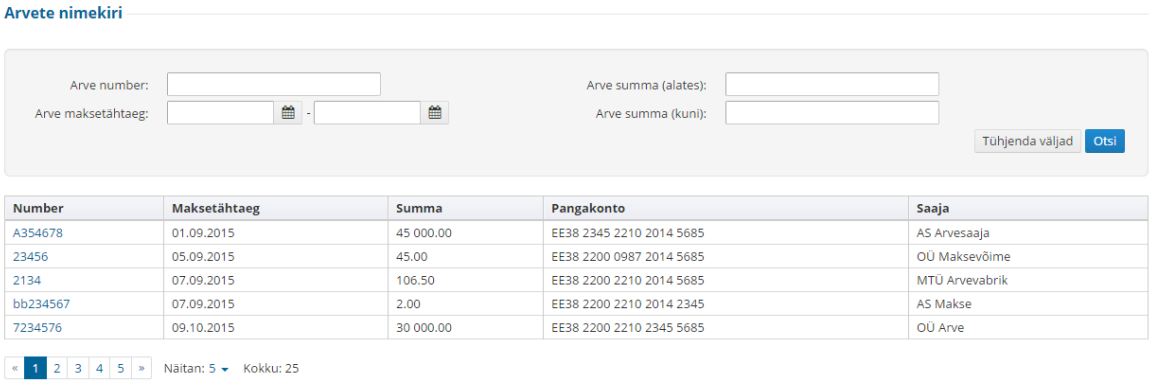
Teenuse nimi	Päri arved
Teenuse kirjeldus	Teenuse abil saab pärida konkreetse komplekti arvetest, mis vastavad etteantud parameetritele. Tagastatakse nimekiri äriobjektidest ARVE.
Meetod	GET
Sisendparameetrid (kohustuslikud)	*
Sisendparameetrid (valikulised)	Arve number, Arve maksetähtaeg, Arve summa (alates), Arve summa (kuni)
Põhivoog	Vastavalt etteantud parameetritele tagastatakse nimekiri äriobjektidest ARVE.
Alternatiivvoog	1) Etteantud parameetritega arvet ei leidu, viga „Arvet ei leidu“. 2) Etteantud parameetritega leidub üle 1000 objekti, viga „Leidub üle 1000 arve, palun kitsenda otsingut“.
Märkmed	Kui vastusena leidub üle 100 objekti, siis tuleks teenuse vastus pagineerida. Vaata prototüüpi Link prototüübile, vastus tuleks pagineerida vastaval kujul.

Näide 1. Teenuse kirjelduse mall.

4.4.4 Prototüüp

Prototüüp on süsteemi arendamisel oma visuaalse iseloomu tõttu arendajale äärmiselt oluline sisend. Prototüübi pealt nähtub, milliseid nüansse tuleb teenuse arendamisel silmas pidada ning milline hakkab välja nägema süsteemi kuva, mille raames seda teenust kasutama hakatakse. Kui prototüüp on loodud nii, et seda saab HTML koodiks teisendada, siis on see aluseks kasutajaliidese arendamisel.

Prototüüpimiseks on saadaval väga erinevaid tarkvaratooteid, millest osad põhinevad HTML[10] koodil ning mille abil on võimalik luua kõrge funktsionaalsega prototüüpe. Samas on saadavad ka tooteid, mille abil on võimalik luua lihtsaid visandeid (ingl. k *Mockup* või *Wireframe*). Joonisel 9 on näitena loodud AS Nortal'i prototüübi mootorit[11] kasutades prototüüp fiktiivsest süsteemist, mille põhjal arendatakse arvete nimekiri.



Joonis 9. Prototüüp.

4.4.5 REST API

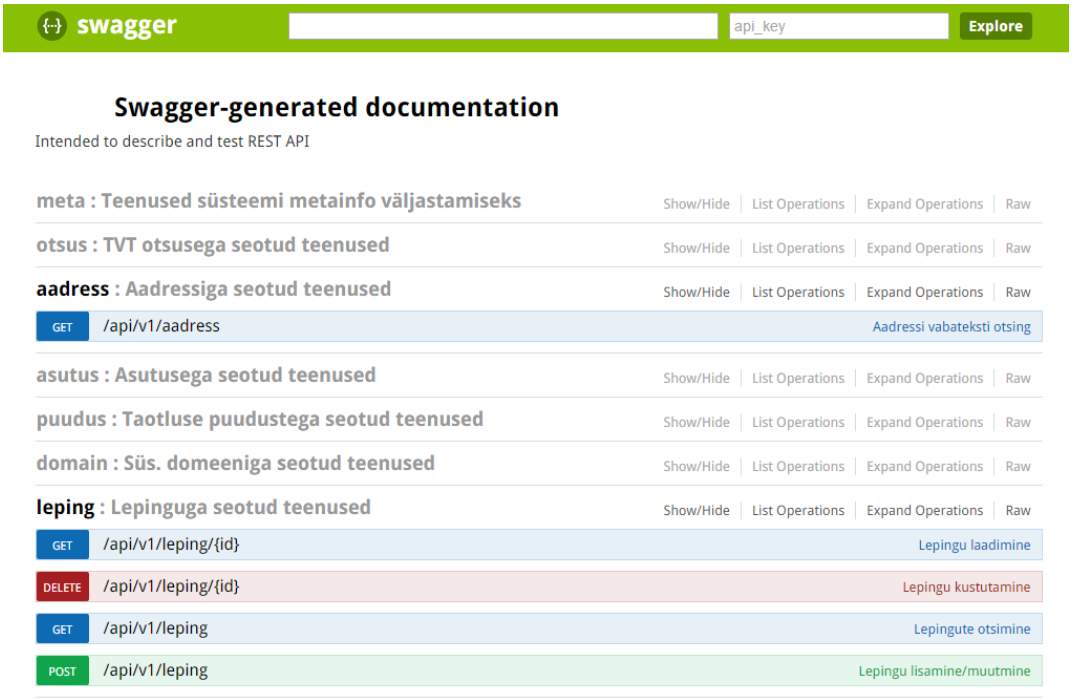
REST API[12] all mõistetakse struktuurset kirjeldust kõikidest teenustest, mis on vastava süsteemi raames loodud. REST API on nii inim- kui ka masinloetav, mis teeb sellest äärmiselt kasuliku töövahendi [20].

REST API abil on võimalik lihtsalt saada ülevaade kasutusel olevatest äriobjektidest ja realiseeritud teenustest. Sealjuures saab lugeja ülevaate objektide ja teenuste omavahelisest struktuurist. Struktuuri alusel on võimalik lihtsalt eristada erinevaid teenuste liike (täpsemalt kirjeldatud peatükis „3.4 Teenuste liigid").

REST API kirjeldamiseks on kasutusel erinevad keeled, mille abil on võimalik kogu API automaatselt koodi pealt genereerida. API sisaldab endas ka andmestruktuuride kirjeldust, mida teenuse raames salvestatakse. Käesoleva bakalaureusetöö raames on vaadeldud raamistikku Swagger[13]. Swaggeril on lihtne ja arusaadav kasutajaliides, mis teeb selle kasutamise väga mugavaks. Swaggeri avalehel on kirjeldatud kogu loodud teenuste hulk ja

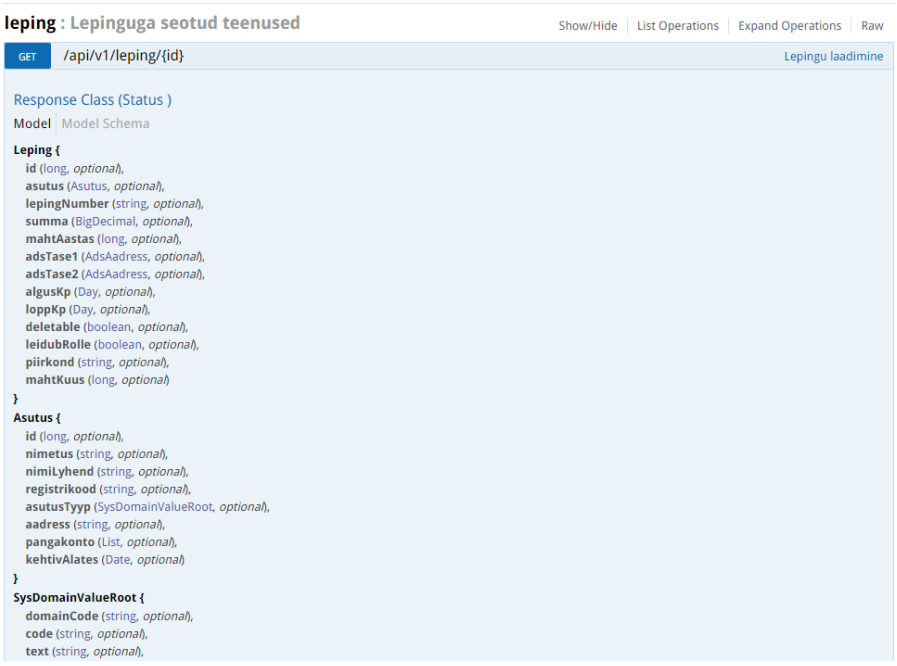
[10] HTML- HyperText Markup Language
[11] AS Nortal'i prototüübi mootor- Ettevõtte poolt arendatud tarkvara lihtsate HTML prototüüpide loomiseks
[12] REST API- Representational State Transfer Application Programming Interface
[13] Swagger- http://swagger.io/

hierarhia, mis teeb sellest analüüsifaasis äärmiselt tarviliku abivahendi, mille abil on lihtne saada ülevaade juba realiseeritud teenustest (Joonis 10).



Joonis 10. Teenuste nimekiri.

Teenuse sisuga tutvumiseks on võimalik avada teenuse detailne kirjeldus, mis on abiks teenuse raames kasutatavast andmestruktuurist ülevaate saamiseks ja teenuse testimisel. Joonisel 11 on toodud näide teenusest, mis on avatud detailse sisuga tutvumiseks.



Joonis 11. Teenuse kirjeldus avatud kujul.

5. Kokkuvõte

Antud bakalaureusetöö raames anti ülevaade monoliitse- ja teenusorienteeritud arhitektuuriga süsteemide analüüsiprotsessi erinevustest. Erinevuste väljatoomiseks kirjeldati põgusalt analüüsiprotsessi monoliitse arhitektuuriga süsteemide puhul ning seejärel toodi välja põhjused, miks on vaja analüüsiprotsessi sõltuvalt arhitektuurist muuta.

Töö raames toodi esile ka erinevad analüüsistrateegiad, mida teenusorienteeritud arhitektuuriga süsteemide puhul rakendada saab. Põhilise osa tööst moodustas autori poolt välja pakutud analüüsiprotsess, mida saab kasutada teenusorienteeritud süsteemide analüüsimisel.

Analüüsiprotsessi kirjeldamise käigus oli viidatud dokumentatsiooninäidistele, mis toetavad autori poolt pakutud protsessi kasutusele võtmist. Töö praktilise iseloomu tõttu loodab autor, et kirjeldatud protsess ja dokumentatsiooninäidised leiavad täies ulatuses rakendamist mõne teenusorienteeritud süsteemi analüüsimisel.

Töö autor töötab igapäevaselt teenusorienteeritud arhitektuuriga süsteemi analüütikuna, mis oli ka antud töö kirjutamise ajendiks. Igapäevaelus on teenusorienteeritud süsteemi analüüsimise algusprotsess küllaltki vaevaline ning seetõttu ei olnud autor veendunud, et kirjeldatud arhitektuurimudel on eraettevõtluses kasutamiseks piisavalt kuluefektiivne ja jätkusuutlik.

Tööprotsessi käigus sai autor korduvalt kinnitust sellele, et teenusorienteeritud arhitektuuri täiel määral rakendamiseks on vajalik suur eelinvesteering eelanalüüsi näol. See lõi kindla teadmuse, et valitud arhitektuurimudel on tulevikuks õige valik ning sellega kaasnevad tulud ilmnevad alles projekti hilisemas elutsüklis.

Samuti nähtus, et teenusorienteeritud arhitektuuriga süsteemide puhul on analüütiku roll äärmiselt oluline, kuna arhitektuur peab ideaalis peegeldama äri ennast. Eelkirjeldatust tulenevalt sai autor töö kirjutamise käigus palju uusi teadmisi ning asub neid igapäevatöös kasutusele võtma.

6. Viited

[1] K. Holley ja A. Arsanjani, „100 SOA Questions Asked and Answered,“ Pearson Education. Inc, 2010, p. 18.

[2] K. Decreus ja G. Poels, „Putting Business into Business Process Models,“ %1 *Annual IEEE International Computer Software and Applications Conference*, 2008.

[3] K. Roebuck, „Spring Framework: High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors,“ Emereo Publishing, 2012, p. 58.

[4] [Võrgumaterjal]. Available: http://searchsoa.techtarget.com/definition/Java-Server-Page. [Kasutatud 21 06 2015].

[5] T. Erl, „Service-Oriented Architecture (SOA): Concepts, Technology, and Design,“ Prentice Hall, 2005, p. 364.

[6] [Võrgumaterjal]. Available: http://en.wikipedia.org/wiki/Representational_state_transfer. [Kasutatud 21 06 2015].

[7] MuleSoft, INC, [Võrgumaterjal]. Available: https://www.mulesoft.com/resources/esb/service-orchestration-and-soa. [Kasutatud 07 07 2015].

[8] T. Erl, „SOA: Principles of Service Design,“ Prentice hall, 2009, p. 43.

[9] T. Erl, „Service-Oriented Architecture (SOA): Concepts, Technology, and Design,“ Prentice Hall, 2005, p. 393.

[10] T. Erl, „SOA: Principles of Service Design,“ Prentice Hall, 2009, pp. 44-45.

[11] M. Rouse. [Võrgumaterjal]. Available: http://searchsoa.techtarget.com/definition/service-orchestration. [Kasutatud 07 07 2015].

[12] T. Erl, „SOA: Principles of Service Design,“ Prentice Hall, 2009, p. 46.

[13] T. Erl, „SOA: Principles of Service Design,“ Prentice Hall, 2009, p. 38.

[14] T. Erl, „SOA: Principles of Service Design,“ Prentice Hall, 2009, pp. 71 - 74.

[15] T. Erl, „Service-Oriented Architecture (SOA): Concepts, Technology, and Design,“ Prentice Hall, 2005, p. 362.

[16] T. Erl, „SOA: Principles of Service Design,“ Prentice Hall, 2009, p. 52.

[17] T. Erl, „Service-Oriented Architecture (SOA): Concepts, Technology, and Design,“ Prentice Hall, 2005, p. 370.

[18] L. Xie, L. Xu ja P. d. Vrieze, „Lightweight Business Process Modelling,“ %1 *International Conference on E-Business and E-Government*, 2010.

[19] T. Erl, „Service-Oriented Architecture (SOA): Concepts, Technology, and Design,“ Prentice Hall, 2005, p. 398.

[20] [Võrgumaterjal]. Available: http://en.wikipedia.org/wiki/Overview_of_RESTful_API_Description_Languages. [Kasutatud 21 06 2015].

Lisad

I. Terminid

Süsteem	Käesoleva töö raames veebirakendus.
Äri(protsess)	Käesoleva töö raames äriprotsess ja kõik sellega kaasnev, et organisatsioon saaks täita oma eesmärke.
Ontoloogia	Ühise sõnavara ja mõistete kasutamise kokkulepe.
Dokumenteerimine	Arendajatele vajaliku dokumentatsiooni loomine analüüsi raames.
Refaktoreerimine	Süsteemi arhitektuuri parendamine / koodi puhastamine.
Abstraheerimine	Teenuse või teenuse arenduse keerukuse viimine mingile kindlale üldistavale tasemele.
Teenuse skoop	Teenuse arenduse maht koos selle juurde kuuluva keerukusega.
Iteratsioon	Üks kordus mitmeetapilisest protsessist, mida korduvalt läbi viiakse.
Liiased andmed	Käesoleva töö raames andmed, mida ei pruugita kasutada, kuid on siiski vajalikud.

II. Monoliitse arhitektuuriga süsteemi analüüsidokument

Eesmärk

Antud dokumendi kirjeldab uue arve lisamist süsteemi ning selle salvestamist andmebaasi.

Visuaalne väljund

Vaata arve lisamise prototüüpi *„viide prototüübile“*

Privileegid

- Kuva nägemiseks peab kasutajal olema privileeg *arve.vaatamine*
- Kuvalt muudetaval kujul nägemiseks peab kasutajal olema privileeg *arve.sisestamine*

Sisendparameetrid

Sisendid puuduvad

Atribuudid

1. Arve liik
    a. Valikmenüü loendi ARVE_LIIK väärtustest
    b. Kohustuslik
2. Arve number
    a. Tekstisisestusväli
    b. Kohustuslik
3. Arve esitaja nimi
    a. Tekstisisestusväli
    b. Kohustuslik
4. Arve esitaja registrikood
    a. Numbrisisestusväli
    b. Mittekohustuslik
5. Pangakonto number
    a. Tekstisisestusväli
        i. Väljale rakendub pangakonto numbri korrektsuse kontroll *„viide kontrollile“*
    b. Kohustuslik
6. Arve kuupäev
    a. Kuupäevasisestusväli
    b. Kohustuslik
7. Arve maksetähtaeg
    a. Kuupäevasisestusväli
        i. Juhul kui kuupäev on minevikus, kuvada kasutajale hoiatusteade „Sisestatud arve maksetähtaeg on minevikus !“
    b. Kohustuslik
8. Arve summa-
    a. Numbrisisestusväli
        i. Juhul kui väljalt „Arve liik“ valiti loendi väärtus ’TAGASIOSTU_ARVE’, võib väljale sisestada negatiivse arvu.
        ii. Muul juhul tuleb negatiivse arvu sisestamise korral kuvada veateade „Arve summa ei saa olla negatiivne !“
    b. Kohustuslik

Tegevused

<u>Nupp „Salvesta“</u>

1. Arve salvestatakse vastavalt kirjeldatud salvestusreeglitele.
2. Kuva suletakse ja kasutaja suunatakse tagasi nimekirja.

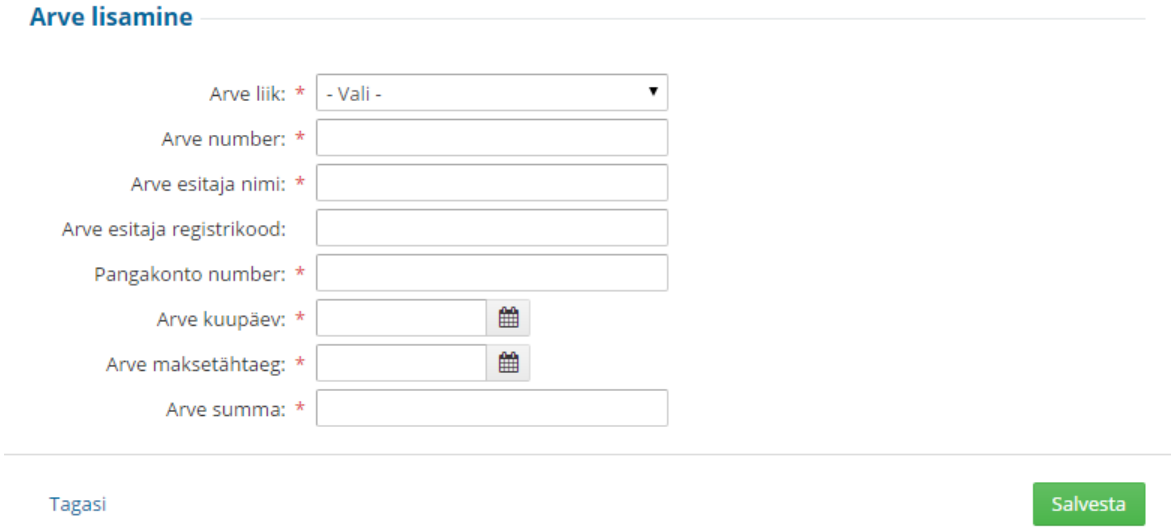
<u>Link „Tagasi“</u>

1. Andmeid ei salvestata, kasutaja suunatakse tagasi nimekirja.

Salvestusreeglid

Andmebaasivälja nimi	Tüüp	Täpsustus / reeglid
ARVE.LIIK_KOOD	VARCHAR	Kasutaja sisestatud väärtus
ARVE.NUMBER	VARCHAR	Kasutaja sisestatud väärtus
ARVE.ESITAJA_NIMI	VARCHAR	Kasutaja sisestatud väärtus
ARVE.ESITAJA_REG_KOOD	VARCHAR	• Kasutaja sisestatud väärtus • Juhul kui tühi, salvestatakse 'TEADMATA'
ARVE.PANGAKONTO	VARCHAR	Kasutaja sisestatud väärtus
ARVE.KUUPAEV	DATE	Kasutaja sisestatud väärtus
ARVE.MAKSETAHTAEG	DATE	Kasutaja sisestatud väärtus
ARVE.SUMMA	NUMBER	Kasutaja sisestatud väärtus
ARVE.SISESTAMISE_AEG	DATE	Salvestamise hetkeaeg
ARVE.STAATUS_KOOD	VARCHAR	Alati salvestatakse 'SISESTATUD'

III. Prototüüp



IV. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Karl Kuusk** (sünnikuupäev: 25.06.1992)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose **Teenuspõhine analüüsiprotsess ja -dokumentatsioon teenusorienteeritud arhitektuuriga süsteemi arendamiseks**, mille juhendaja on Vambola Leping,

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

Tartus, **13.08.2015**