Tallinna Tehnikaülikool
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Valdkonnamudelite arhetüüpidel põhinev loomine Enterprise Architect CASE vahendis

Bakalaureusetöö

Üliõpilane: Caroline Vainomäe
Üliõpilaskood: 120923
Juhendaja: Erki Eessaar

Tallinn
2016

Autorideklaratsioon

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

(*kuupäev*)	(*allkiri*)

Valdkonnamudelite arhetüüpidel põhinev loomine Enterprise Architect CASE vahendis

Annotatsioon

Töö üldiseks eesmärgiks on hulga universaalsete analüüsimustrite Enterprise Architect (EA) CASE vahendis realiseerimine, mis hõlbustaks edaspidi erinevate valdkondade infosüsteemide modelleerimist. Realiseeritavad mustrid peavad olema piisavalt universaalsed, et nende abil oleks võimalik koostada suvalise valdkonna kontseptuaalne andmemudel või valdkonnamudel.

Töö olulisima tulemusena on EA CASE vahendis realiseeritud üheksa analüüsimustrit ning nende integratsiooni muster (http://apex.ttu.ee/archetypes) Töö tulemuseks on ka nende mustrite esitus mustri formaadis, mustrite puuduste analüüs ning konkreetse süsteemi mudelid, kus on demonstreeritud loodud analüüsimustrite rakendamist.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 66 leheküljel, 7 peatükki, 27 joonist, 2 tabelit.

Archetype-based Creation of Domain Models in the Enterprise Architect CASE Tool

Abstract

The general goal of the thesis is to implement a set of universal analysis patterns in Enterprise Architect (EA) CASE tool that would help simplify the process of modeling information systems from various domains. The implemented patterns must be sufficiently universal, so that it would be possible to put together a conceptual data model or domain model of an arbitrary domain with the help of these patterns.

The most important result of this thesis is implementation of nine analysis patterns and their integration pattern in EA CASE tool (http://apex.ttu.ee/archetypes). The other results are representations of these patterns in a pattern format and models of a particular system that have been created by using the implemented analysis patterns.

The thesis is in Estonian and contains 66 pages of text, 7 chapters, 27 figures, 2 tables.

Lühendite ja mõistete sõnastik

Arhetüüp	Archetype Arhetüüpideks nimetatakse abstraktsioone, mis esinevad tihti ning mis on olulised väljaspool üksikuid rakendusi.(Blaha, 2010)
CASE vahend	Computer-Aided Software Engineering tool CASE vahend on tarkvarasüsteem, mis aitab tarkvara arendajat ühe või rohkema tarkvara arendustsükli etapi/faasi/iteratsiooni jooksul (Eessaar, 2012)
Domeenimudel ehk valdkonnamudel	Domain model, Conceptual model Valdkonna mõistete ja nende vaheliste seoste lihtsustatud esitus. Mudel, mis aitab meil mõista ning lihtsustada probleemi (Fowler, 1996)
Integratsiooni muster	Integration pattern Koondav muster, nii öelda mustrite muster, mis võtab kokku komponentmustreid. (Giles, 2012)
Komponentmuster	Component pattern Integratsiooni mustris välja toodud põhimõistele vastav muster.
Kontseptuaalne andmemudel	Conceptual datamodel Analüüsi mudel, kus väljendatakse soovid andmebaasis registreeritavate andmete kohta, kuid ei kavandata andmebaasi ühtegi konkreetset tehnilist platvormi kavandades.

Olem	Entity Infosüsteemiga kirjeldatava süsteemi või valdkonna konkreetne või abstraktne komponent/asi (EKSS, 2015)
UML	Unified Modeling Language Üldotstarbeline visuaalne modelleerimiskeel infosüsteemide ja tarkvara (sealjuures, kuid mitte ainult objektorienteeritud keeltel põhinevate projektide) spetsifitseerimiseks ja visualiseerimiseks. (Vallaste, 2015)

Jooniste nimekiri

Sisukord

1. Sissejuhatus

Süsteemide kontseptuaalsel modelleerimisel leitakse ühe esimese asjana selle süsteemi põhiobjektid ehk põhiolemitüübid, mis vastavad valdkonna põhimõistetele. Kui loodud mudelit piisavalt üldistada, siis võib näha, et suures osas kirjeldab see samu põhimõisteid kui paljude teiste süsteemide mudelid. Selle tõttu võiks modelleerimise efektiivust tõsta see, kui nende mudeli osade nullist väljatöötamise asemel kasutada juba varem loodud mustreid, millest loodava süsteemi kontseptuaalne mudel koostada.

1.1 Taust ja probleem

Töö laiem eesmärk on aidata isikuid, kes soovivad tegeleda konkreetse süsteemi valdkonnamudeli või kontseptuaalse andmemudeli väljatöötamisega, kasutades CASE vahendit Enterprise Architect. Töö tulemusi võib muuhulgas kasutada õppeotstarbelisel eesmärgil.

Töös realiseeritud komponentmustrid aitavad luua valdkonnamudeli või kontseptuaalse andmemudeli (edaspidi kasutan üldnime "kontseptuaalne mudel") ning modelleerimise käigus lihtsustatud integratsiooni mustri abil saab ülevaate modelleeritava süsteemi põhiolemitüüpidest. Mustreid kohandades saab süsteemi modelleerimise protsessi optimeerida – puudub vajadus baaselementide üha uuesti väljatöötamiseks.

1.2 Ülesande püstitus

Töö konkreetseks eesmärgiks on realiseerida Enterprise Architect CASE vahendis üheksa taaskasutatavat komponentmustrit ning nende integratsiooni muster, mis lihtsustaksid edaspidiselt suvalise valdkonna kontseptuaalsete mudelite loomist. CASE vahendi kasutaja peab saama neid mustreid enda mudelite loomisel aluseks võtta. See on võimalik tänu EA pakutavale laiendusmehhanismile, mis võimaldab defineerida mustreid ning neid modelleerimisel aluseks võtta. Samuti tutvutakse töös lühidalt mustri ning modelleerimise

mõistega ning kirjeldatakse mustrite põhjal modelleerimist ning Enterprise Architect vahendiga mustrite loomist protsessi.

Ülesandeks on ka kasutada loodud mustreid vabalt valitud valdkonna näitesüsteemi kontseptuaalse andmemudeli loomisel, et näidata nende mustrite kasutamise võimalikkust ja kasulikkust.

1.3 Metoodika

Komponentmustrid on loodud Michael Blaha 2010. aastal teoses „Patterns of Data Modeling“ (Blaha, 2010) leiduvate arhetüüpide ning Martin Fowler 1996. aasta teoses „Analysis Patterns – Reusable Object Models“ (Fowler, 1996) leiduvate analüüsimustrite põhjal. Integratsiooni mustri loomisel on aluseks võetud Erki Eessaare õppeaine Andmebaasid I ja II õppematerjalid (Eessaar, 2015), kus on dokumenteeritud Giles (2012) esitatud integratsiooni muster.

Mustrite ning mudelite loomine käib Sparx Systems CASE vahendi, Enterprise Architect versioon 12 abil ning need on loodud UML notatsioonist lähtudes.

1.4 Ülevaade tööst

Töö esimeses osas käsitletakse mustri mõistet ning ka täpsemalt analüüsimustreid. Teises osas kirjeldatakse lühidalt modelleerimisprotsessi teooriat ja modelleerimiskeeli. Kolmas osa sisaldab töös realiseeritavate üheksa analüüsimustri ja ühe integratsiooni mustri kirjeldust mustri formaadis. Kuna loetud allikates neid sellises formaadis ei esitatud, saab ka seda lugeda üheks töö tulemuseks. Analüüsimustreid kirjeldatakse nii graafiliselt kui tekstiliselt ning nende kirjeldused on loodud vastavalt mustrite maailmas üldlevinud mallile. Samuti toimub selles osas mustrite analüüsimine ning kirjeldatakse, millised on nende võimalikud probleemikohad ning võimalikud probleemide lahendused.

Viiendas peatükis kirjeldatakse mustrite realiseerimise ning mudelis kasutamise protsessi Enterprise Architect CASE vahendis. Välja on toodud selle töövahendi kasutamise eripärad ning tähelepanekud. Samuti on selles osas Enterprise Architecti lühitutvustus.

Viimases osas demonstreeritakse loodud analüüsimustrite kasutamist e-arvete infosüsteemi maksmise alamsüsteemi näitel. Siin osas kirjeldatakse arhetüüpide valimise protsessi, lähtudes realiseeritud integratsiooni mustrist. Lisaks esitatakse mustrite põhjal loodud esialgne kontseptuaalne andmemudel ning seejärel valdkonna eripärasid arvesse võttes parandatud mudel. Samas peatükis toimub ka loodud mudelite analüüs.

2. Mustrid

Antud peatükk seletab lahti mustri mõiste ning kirjeldab lähemalt infosüsteemide modelleerimisel kasutatavaid analüüsimustreid.

2.1 Mustri mõiste

Mustrid ümbritsevad meid maailmas kogu aeg. „Eesti keele seletava sõnaraamatu“ definitsiooni kohaselt on muster „eeskuju, mall, näidis; millegi läbiv ühine joon, seaduspära vms.“ (2015). Fowler (1996, XV) defineerib mustrit kui ideed, mis on olnud kasulik ühes praktilises kontekstis ning tõenäoliselt on kasulik ka teistes kontekstides. Ehk mustrite põhiline tunnusjoon on selle taaskasutatavus erinevate taustadega süsteemides. Mustrid leiavad kasutust paljudes erinevates valdkondades – näiteks ehituses, tekstiilitööstuses, finantsturgudel ning infosüsteemides. Infosüsteemide valdkonnas eksisteerib omakorda erinevaid tüüpi mustreid – näiteks analüüsimustrid, äriprotsesse kirjeldavad mustrid, andmete modelleerimise mustrid, disainimustrid, arhitektuurimustrid ja SQL-mustrid (Eessaar, 2012)

2.2 Analüüsimuster

Martin Fowler'i (1996, XV) raamatus on analüüsimuster defineeritud järgnevalt – analüüsimustrid on grupid kontsepte ehk mõisteid, mis esindavad ärimodelleerimises tüüpilist/tavapärast struktuuri. Selle definitsiooni ja ka käesoleva töö mõistes kirjeldavad analüüsimustrid probleemi struktuuri. Tegelikult on võimalikud ka käitumuslikud analüüsimustrid, mis kirjeldavad võimalikku äri- või infosüsteemi funktsioone ja sellest tulenevat käitumist (protsesse), kuid nendega tegelemine pole selle töö teema. Analüüsimuster võib olla asjakohane ainult ühes valdkonnas, kuid võib sobida kasutamiseks ka mitmetes valdkondades.

„Analüüsi sisuks on vaadata esmalt silma hakkavate nõudmiste taha ja tulla lagedale nõudmiste taga oleva probleemi mentaalse mudeliga.“ (Fowler, 1996, 1). Analüüsimustreid kasutatakse kontseptuaalsete mudelite tegemisel, ehk oluline on mõistete ja seoste

väljaselgitamine ning nende abil probleemi kirjeldamine, mitte süsteemi tehniline optimeeritus ning sisemised tööprotsessid. Fowleri kohaselt on kontseptuaalne mudel „mentaalne mudel, mis aitab meil probleemi mõista ja lihtsustada“ (Fowler, 1996, 2). Kontseptuaalsed mudelid ei käsitle süsteemide tarkvaralist poolt. Selle tõttu ei saa ka oodata analüüsimustrite valimisel süsteemis tehtud päringute kiiruste optimeerimist või kõige tarkvaraliselt efektiivsemaid lahendusi. Analüüsimustrid esitavad lihtsustatud viisil mõisteid ja nende seoseid. Analüüsimustrite kasutamise järel tuleb kasutusele võtta disainimustrid, mis võivad muuhulgas käsitleda süsteemi toimimise efektiivsust ja operatsioonide kiiruse optimeerimist. Disainimustrid kirjeldavad lähemalt süsteemi realisatsiooni struktuuri ja kasutajaliideseid. Analüüsimustrid aitavad koostada esmase suure pildi süsteemist ning alustada selle täpsemat üles ehitamist.

„Analüüsimustrid, vastandina disainimustritele, keskenduvad süsteemi organisatsioonilistele, sotsiaalsetele ja majanduslikele aspektidele, sest need omavad nõuete analüüsis ja süsteemi vastuvõtutestide ning kasutatavuse testide juures keskset rolli“ (Schulz ja Hashler, 2001, 2)

Silverston (2001, 1–2) kohaselt hõlmavad tüüpilisest organisatsiooni andmemudelist 50% osad, mida kasutatakse enamikes organisatsioonides, olenemata organisatsiooni tegutsemisvaldkonnast. Seega tagab juba olemasolevate mustrite või universaalsete mudelite kasutamine mudelite kiire ning efektiivse loomise. Tuleb silmas pidada, et mustrite kasutamisel pole rangeid reegleid ning nende süsteemi toomisel võib ja peab neid kohandama vastavalt infosüsteemi nõuetele. Hea muster on kirjeldatud piisavalt üldiselt, et seda saaks rakendada erinevates valdkondades, kuid igas valdkonnas on eritingimused, mille tõttu peab mustrite alusel loodud esialgseid mudeleid täiustama või muutma (lisama teisele 50%-le vastavad osad).

3. Arhetüüpidel põhinev kontseptuaalne modelleerimine

Käesolev peatükk kirjeldab lühidalt mustripõhist modelleerimist ja arhetüübi mõistet.

3.1 Arhetüübid

EKSS (2015) defineerib arhetüüpi psühholoogia kontekstis kui „inimkonna kollektiivsest alateadvusest lähtuv püsistruktuur, mis teadvuses avaldub universaalse motiivi v. kujundina.“ Modelleerimise kontekstis nimetatakse arhetüüpideks mustreid, mida võib võtta aluseks suvaliste mudelite loomiseks. Antud töös vaadeldakse süsteemide kontseptuaalse struktuuri modelleerimiseks mõeldud arhetüüpe. Need vastavad süsteemi põhiobjektidele ehk põhiolemitüüpidele ja esitavad ühtlasi süsteemi põhimõisteid. Arhetüübid on loodud võimalikult üldise kirjelduse abil selleks, et tagada nende sobivus võimalikult paljude valdkondade süsteemidega. „Arhetüübid on sageli ilmnevad abstraktsioonid, mis väljuvad üksikute rakenduste piiridest“ (Blaha, 2010, 121).

Arhetüüpe on dokumenteerinud diagrammide ja diagrammil olevate elementide vabatekstiliste selgituste abil näiteks Blaha (2010). Michael Blaha kasutas oma arhetüüpide kirjeldamiseks UML ning IDEF1X notatsiooni. Erinevad autorid pakuvad välja erineva hulga arhetüüpe, millele põhinedes süsteeme modelleerida. Toon tabelis 1 võrdluseks välja Piho (2011, 19) ja Giles (2012) pakutavad arhetüübid. Nagu näete, langevad need kokku ainult osaliselt. Antud töös otsustasin valida realiseerimiseks Giles (2012) arhetüüpidele vastavad komponentmustrid, sest Giles pakub ka nende integratsiooni mustri, mille kasutamine teeb modelleerimisega alustamise lihtsamaks.

Tabel 1 Mõned väljapakutud arhetüüpide hulgad

Giles (2012)	**Piho (2011)**
Konto	
Leping	Tellimus
Dokument	

Giles (2012)	Piho (2011)
Sündmus	
Asukoht	
Osapool ja roll	Osapool Osapoole seos
Toode	Toode
Ressurss/vara	Vara/laoseis
Ülesanne	
	Reegel
	Kogus ja raha

3.2 Modelleerimine

Mudel on kirjeldatava süsteemi lihtsustus, mis luuakse ühte või mitut keelt kasutades. Need keeled võivad olla loomulikud keeled, aga ka spetsiaalselt modelleerimiseks loodud visuaalsed või tekstilised keeled. Modelleerimiseks nimetatakse info- ja tarkvarasüsteemide arendamise kontekstis protsessi, kus luuakse süsteemist samm-sammu haaval aina suurema täpsusastmega mudelid ning ideaalis jõutakse niimoodi lõpuks välja töötava tarkvarani. Süsteemi kirjeldamiseks luuakse mitmeid mudeleid, et kirjeldada kõiki süsteemi aspekte (kes, kus, millal, miks, mida). Käesoleva töö tulemus aitab luua süsteemi kontseptuaalse struktuuri mudelit ning töös luuakse näitena kontseptuaalne andmemudel. Kontseptuaalse modelleerimise käigus kirjeldatakse süsteemi probleemi iseloomustavad mõisted ja nende seosed. See on muuhulgas vajalik, et edaspidi oleks võimalikult vähe probleeme seoses mõistetest erinevalt arusaamisega. Kontseptuaalsetele mudelitele saab rakendada teisendusreegleid, mille abil luua esialgne versioon süsteemi tehnilise lahenduse kirjeldusest, mis on mõeldud mingi kindla platvormi jaoks.

Käesolevas töös realiseeritakse EA CASE vahendis abivahend Giles'i (2012) pakutud viisil süsteemi kontseptuaalse struktuuri modelleerimiseks. Selle kohaselt tuleb modelleeritava süsteemi spetsiifikat arvestades kõigepealt lihtsustada integratsiooni mustrit (vt punkt 4.10). Lihtsustatud integratsiooni mustri põhjal leitakse esialgne hulk põhiolemitüüpe ning nende vahelised seosed. Igale põhiolemitüübile vastab omakorda komponentmuster. Just neid

komponentmustreid on võimalik moodustada, kas siis mitmete universaalsete mudelite või mustrite põhjal.

Võttes kasutusele mustrid, on algmudeli loomine efektiivsem ning edaspidistes arendussammudes muudetakse mustrite abil loodud mudelit süsteemispetsiifiliseks. Tulemuseks saadav mudel on piisava kvaliteediga, kuna mustrid on professionaalsete analüütikute poolt kirjeldatud, põhinedes kogemustele, mis on saadud erinevatesse valdkondadesse kuuluvate süsteemide modelleerimisel. Loodud mudeli kvaliteedi tõstmiseks tuleb esialgset mudelit süsteemispetsiifiliselt täiendada. Kuna üldine struktuur on eeskujuks olemas, siis on väiksem võimalus selle tegevusega ebaõnnestuda. Kuna esialgse mudeli loomine toimub tänu mustrite kasutamisele kiiresti ja mustrid aitavad tagada piisavat kvaliteeti, siis peaks olema sellisel viisil modelleerimine vastuvõetav ka paindmetoodikate käigus, mis igal võimalikul viisil üritavad vähendada ressursside üleliigset kulutamist, kuid samas hoida kvaliteeti. Paindmetoodikate kasutamisel võib tunduda süsteemi arhitektuuri kirjeldamine „raiskamisena“, kuid kuna töös pakutava lähenemise korral saadakse kirjeldus vähese kuluga, sobib seda teha ka selle metoodika puhul.

On oluline märkida, et kuigi mustrid on välja töötatud selliselt, et need oleksid võimalikult universaalsed ning võimaldaksid kasutamist mitmetes erinevate valdkondade mudelites, on need rangelt soovituslikud. Modelleerimisel tuleb silmas pidada süsteemi kohta sätestatud reegleid ning neid ümbritsevaid tavasid ning täiustama mustrite abil loodud esialgset mudelit nende järgi. Selle käigus tuleb näiteks kaotada üleliigseid element või lisada uusi elemente, mis täpsustavad algses mudelis kirjeldatud mõisteid.

3.3 Modelleerimiskeeltest

Modelleerimiskeeled võivad olla tekstilised või visuaalsed. Selleks, et modelleerimisel kasutatavad tähised oleksid alati üheselt mõistetavad, on vaja nende loomisel kasutada varem kokku lepitud standardeid. Seega peaks keele kohta olema standard ja kõik seda keelt kasutavad süsteemid peaksid seda standardit järgima. Modelleerimiskeel võib olla universaalne (erinevates valdkondades kasutatav) nagu UML (*Unified Modeling Language*) või mingi kindla valdkonna jaoks mõeldud nagu andmete modelleerimiseks mõeldud IDEF1X (*Integration DEFinition for Information Modeling)*.

Antud töös luuakse mustreid vastavalt UML 2 standardile.

UML võimaldab graafiliselt kujutada süsteemi funktsioone, protsesse, komponente ja komponentide vahelisi suhteid. Selle keele abil on võimalik luua näiteks kasutusjuhtude-, jada- ning klassidiagramme, milles viimast kasutatakse selles töös mustrite kirjeldamiseks.

Toon tabelis 2 eraldi välja üldistusseoste kitsenduste mõistete selgitused, sest neid kasutatakse töös olevates mudelites. Kitsendused esinevad paaridena, kus esimene kitsendus on kas *Incomplete* või *Complete* ning teine on *Overlapping* või *Disjoint*.

Tabel 2 Üldistusseoste kitsendused

Kitsenduse nimi	Tähendus
Complete (kasutatakse ka nime *Mandatory*)	Iga ülemklassi kuuluv element peab kuuluma ka vähemalt ühte alamklassi.
Incomplete (kasutatakse ka nime *Optional*)	Võib leiduda ülemklassi kuuluvaid elemente, mis ei kuulu ühtegi alamklassi.
Overlapping (kasutatakse ka nime *And*)	Ülemklassi element võib korraga kuuluda mitmesse alamklassi.
Disjoint (kasutatakse ka nime *Or*)	Ülemklassi element võib korraga kuuluda ainult ühte alamklassi.

4. Mustrite realiseerimine

Käesolevas töös realiseeritakse Giles (2012) arhetüüpidele vastavad komponentmustrid ning samuti Giles (2012, 238) pakutud integratsiooni mustrid. Kuna Giles (2012, 238) integratsiooni mustris nimetatud arhetüüpidele (põhimõistetele) vastavatest komponentmustritest on hulgaliselt variatsioone, siis tuleb valida, millised töös realiseerida. Käesoleva töö tulemusena realiseeritavad mustrid (vt Joonis 1 - Joonis 9) põhinevad Blaha (2010) arhetüüpidel ning kui sealt vastet ei leidu, siis Fowler'i (1996) analüüsimustritel. Blaha (2010) arhetüübid valiti töö aluseks põhjusel, et vastav raamat on Tallinna Tehnikaülikooli üliõpilastele virtuaalse raamatukogu kasutamise kaudu kättesaadav ning seega saab õppeprotsessis integreerida raamatu ning realiseeritud mustrite kasutamist.

Rõhutan, et nende mustrite sisu pole minu looming, need on juba eelnevalt kirjeldatud. Samas ei kasuta Fowler (1996) ja Blaha (2010) nende kirjeldamiseks struktuurset formaati, mida mina oma töös kasutan. Seega on käesoleva töö panuseks/lisandväärtuseks nende mustrite üks võimalik struktuursem esitus ning samuti vastavate klassidiagrammide tõlge eesti keelde.

Mustrite kirjeldamiseks võtan eeskuju Meszaros ja Doble mallist, kohandades mustrite kirjeldamise viisi, et muuta see antud projekti eesmärkidega sobivamaks.

Mustri nimi: Peab olema lühike, meeldejääv ja mustrit iseloomustav. Välja on toodud nii eestikeelne kui ingliskeelne nimi.

Allikad: Viited mustri loomisel aluseks võetud materjalidele.

Probleem: Probleem, mida selle mustriga lahendatakse.

Lahendus: Välja pakutud lahendus probleemile, mis hõlmab kontseptuaalset mudelit graafilisel kujul (antud juhul UML diagrammina) ning mudelis esinevate elementide (klasside) tekstilist kirjeldust.

Mustri puudused: Mustri kitsaskohtade ja eripärade vabatekstiline kirjeldus. Detailne puuduste analüüs pole selle töö teema, kuid võiks olla mõne eraldi lõputöö teema.

Seotud mustrid: Juhul, kui leidub, siis viited teistele mustritele, mis on sellega seotud.

Näited: Mustri kasutamise näited. Kuna muster ei kirjelda mitte uut ideed, vaid ennast tõestanud lahendust probleemile, siis võiks olla vähemalt kolm näidet mustri kasutamise kohta. Bakalaureusetöö mahtu arvestades ei hakanud ma otsima näiteid reaalsetest süsteemidest, vaid pakkusin ise välja näiteid, kuidas ja kus seda mustrit kasutada.

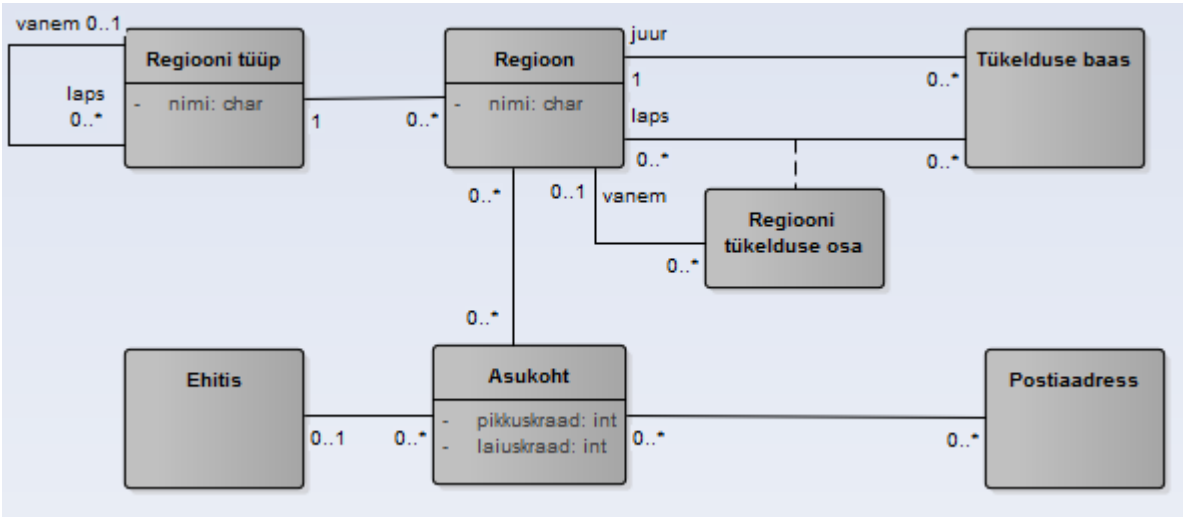
4.1 Muster: Asukoht

Mustri nimi: Asukoht (inglise keeles *Location)*

Allikad: Blaha, 2010, 134 (*Location*)

Probleem: Süsteemis on vaja kirjeldada objektide füüsilist asukohta.

Lahendus :



Joonis 1: Asukoha muster

Asukoha mustris (vt Joonis 1) määrab objekt *Asukoht* ära sellega seotud objekti reaalse asukoha ruumis. Asukoha poole pöördumiseks kasutatakse konkreetset asukoha *Aadressi*. Aadressid on üles ehitatud hierarhiliselt, kuna aadress võib olla erineva täpsusega. Väiksemad territooriumid kuuluvad suuremate alla, näiteks mingi tänav kuulub linnaossa ning see linnaosa kuulub omakorda linna haldusalasse.

Aadressid võivad ka aja jooksul muutuda, kuid kuna käsitleme objekti hetkelist asukohta ruumis, siis aja registreerimine pole antud juhul vajalik.

Asukoht on antud mustri keskne element, millele on antud füüsilised koordinaadid atribuutidena pikkuskraad ja laiuskraad.

Asukohal võib olla *Postiaadress*, kuid ei pruugi. Postiaadressid võivad muutuda ning selle tõttu ei vasta postiaadressile alati füüsilist asukohta. Samuti võib ühes asukohas olla mitu ehitist, millel on erinevad postiaadressid.

Ühes füüsilises asukohas saab asuda maksimaalselt üks *Ehitis*. Ehitise all võib mõista nii hoonet kui rajatist.

Asukoht võib asuda mitmes *Regioonis*. *Regiooni tüübi* all mõistetakse regiooni jaotuse nimetust, nagu näiteks maakond või linn. Regiooni tüübi seos iseendaga kirjeldab tüüpide võimalikku hierarhiad. On teada, et näiteks poliitiliselt jagatakse maa-alad riikideks, mis vastavalt halduskorraldusele võivad jaguneda maakondadeks.

Tükelduse baasi all mõistis Blaha regiooni jaotuse alust nagu haldusjaotus (maakonnad, linnad) ning ajatsoonid. Regioonide hierarhiliseks kirjeldamiseks on kasutanud Blaha kattuva puu *(„overlapping tree“)* malli, mis on lahti seletatud punktis 4.1.1.

Mustri puudused: Ei ole võimalik kirjeldada süsteemis asuva objekti füüsilist asukohta suure täpsusega – näiteks kortermajas korteri numbri tasemel. Objekt on siiski seotud asukohaga ehk täpsete koordinaatidega. Samas selle füüsilise asukohaga on ühes paikneva ehitises kortermaja puhul seotud mitmed postiaadressid. Aadresside modelleerimiseks pakub Blaha (2010, 124) arhetüübi *Aadressid*.

Näited:

- Apteek asub konkreetsete koordinaatidega *Asukohas.* Selles asukohas asub apteegi jaoks rajatud hoone (*Ehitis)* aadressil Tehase 15 *(Postiaadress).* Poliitilise jaotuse *(Tükelduse baas)* alusel kuulub hoone *Regiooni* Harjumaa, mille *Tüüp* on maakond. Sama tükelduse järgi asub hoone Harjumaa alamregioonis Tallinn, mille regiooni tüübiks on linn.

- Veebirakendus, mis vastab päringutele asub serveril, mille füüsiline *Asukoht* on serveripargis, mis asub suures hoones *(Ehitis),* millele on suurusest tulenevalt määratud mitu *Postiaadressi. Postiaadress* oli varasemalt Tehnika 3. Uute halduskorralduste alusel määrati ehitisele uus *Postiaadress* – Tehnika 5.

- Firma kaevandab ressursse suurelt maa-alalt, millele ei vasta ükski *Postiaadress.* Maa-ala mõõt on nii suur, et haldusjaotuse alusel *(Tükelduse baas)* asub see läbi mitme USA osariigi (*Regiooni tüüp*) – New Mexico ning Texase *(Regioon)*. Ajatsoonide alusel jaotuse *(Tükelduse baas)* järgi asub see kaevandus samuti kahes erinevas ajatsoonis *(Regioonis).*

4.1.1 Kattuva puu mall

Kattuva puu malli kasutatakse, kui hierarhiasse kuuluv element võib esineda mitmes puus. Kirjeldades seda malli, lähtudes Asukoha mustrist (vt Joonis 1), käsitletakse *Tükelduse baasi* kui erinevaid puid, kuhu kuuluvad *Regioonid* erinevate jaotuste, nagu ajatsoonide ning poliitilise jaotuse, põhjal. Üks puu juur võib olla mitme tükeldusbaasi aluseks.

Kattuvas puus laste ning vanemate kirjeldamiseks kasutatakse UML notatsioonis kasutuses olevat elementi – sidemeklass („association class“).

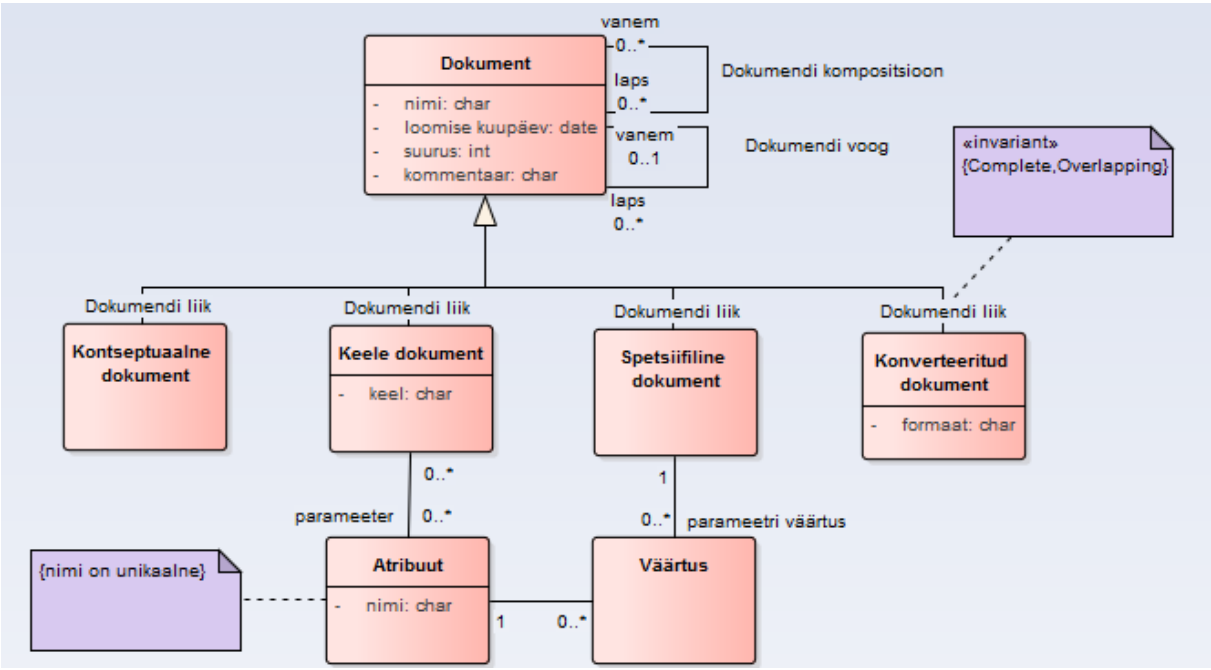
4.2 Muster: Dokument

Mustri nimi: Dokument (inglise keeles *Document)*

Allikas: Blaha, 2010, 128 (Document)

Probleem: Tööprotsessi käigus tekib dokument, mida tuleb säilitada, et huvitatud osapooltel oleks suvalisel ajahetkel saada infot selle olemasolust ja olemusest.

Lahendus :



Joonis 2: Dokumendi muster

Dokumendi mustris (vt Joonis 2) dokumendi identifitseerimiseks kasutatakse sellele määratud nime. Vajalik on jäädvustada ka dokumendi *loomise kuupäev*, et vajadusel veenduda, kas dokument on loodud enne või pärast sündmust, millega seoses see dokument tekkis.

Dokumendid organiseeritakse kahe erineva hierarhilise struktuuri alusel – dokumendi kompositsioon ning dokumendi voog.

Dokument võib koosneda mitmetest alamosadest ning lisadest nagu joonistest, piltidest, tabelitest või alamdokumentidest. Dokument ei ole tingimata jaotatud alamosadeks ning seda kirjeldab 0..* seosetüüp. Samade seosetüüpidega iseendaga seose loomisega kaasnevate probleemide vältimiseks nimetatakse seose otsad erinevalt – „vanemaks“ ja „lapseks“ Alamosade ja lisade omavahelisi seoseid kirjeldab *Dokumendi kompositsiooni* hierarhia.

Dokumente käideldakse ja muudetakse ühelt kujult teisele, alustades *Kontseptuaalse dokumendiga.* Seejärel tõlgitakse see vastavalt vajadusele erinevatesse keeltesse *(Keele dokument*) ning kontseptuaalse dokumendi muutmiseks täpsemaks, et see vastaks reaalsetele parameetritele, määratakse keele dokumendi loomisel sellele kohatäitjad (*Atribuut*). Edasi sisestatakse dokumenti reaalsed andmed (*Väärtus*) kohatäitjate asemele. Viimase etapina viiakse dokument soovitud formaati, mille tulemusena tekib viimase sammuna *Konverteeritud dokument***.**

Selline dokumendi muundamise kulg kuulub *Dokumendi voo* hierarhiasse vabalt valitud sammude järjekorras. Vaatamata sellele, et mustri põhjal on lubatud igasugune muundamise järjekord, avaldas Blaha arvamust, et eelnevalt mustri kasutamisel tuleks eelistada eelnevalt silmas peetud voogu.

Mustri puudused : Dokument on kontseptuaalse, keelelise, spetsiifilise ja konverteeritud dokumendi liikide üldistus ning neil puudub järjestatud seos dokumentide voo hierarhiaga. Sellegi poolest pidas Blaha silmas seda, et dokumentide voog liigub kontseptuaalsest dokumendist konverteeritud dokumendini mudelis vasakult paremale kulgevas järjestuses.

Dokumentidest tekivad erinevad versioonid, kuid mudel seda võimalust ei kajasta. Samuti võib igal dokumendi eksemplaril olla kindel füüsiline asukoht, kuid mudel ei näita seost asukohtadega. Dokumenti menetlevad erinevad osapooled, kuid mudelis puudub seos osapooltega.

Seotud mustrid: Dokument on seotud *Lepingu mustriga* (vt Joonis 4), milles määratletakse lepingu tingimusi ning osapooli, mis kõik on vajalik omakorda dokumendis jäädvustada.

Näited:

- Rahvusvaheline firma teeb töölepingu malli, mida saaks kasutada uute töötajate palkamisel. Mall on praegusel juhul *Kontseptuaalne dokument*. Kuna plaanis on palgata inimesi erinevatest riikidest, tehakse nii eesti kui inglise keelne mall *(*kaks erinevat *Keele dokumenti)*, kus on kohatäitjad nime, palga, ametikoha, kuupäevade ja muu sarnase asemel. Palgatakse uus isik, kellega lepingu sõlmimisel määratakse kohatäitjatele väärtused *(Spetsiifiline dokument)* ja viiakse dokument .pdf kujule *(Konverteeritud dokument)*. Need dokumendid kuuluvad kõik dokumendi voogu.

  Töölepingule lisatakse lisana ka töökoha sisekorra eeskirjad. Antud lisa on dokumendi kompositsiooni hierarhias, kus vanem on tööleping ja laps on sisekorra eeskirjad.

- Mööblieseme väljatöötamisel kuulub *Dokumendi kompositsiooni* joonis, mis on ka *Kontseptuaalne dokument.* Joonisele lisatakse kohatäitjad kohtadele, kuhu tuleb lisada konkreetsed mõõtmed, näiteks nagu „laius“ *(Keele dokument)*. Edasi lisatakse kohatäitjatele välja arvutatud mõõtmed *(Spetsiifiline dokument)* ning tarkvara kasutades joonestatud projekt prinditakse välja paberkujul (*Konverteeritud dokument*).

- Infosüsteemi struktuuri välja töötamisel loodud mudelit saab vaadelda kui *Dokumenti.* Esmalt kasutusele võetud muster on *Kontseptuaalne dokument.* Mustri olemitüüpidele vasteid leides luuakse *Spetsiifiline dokument.* Saadud mudelit võib esitada mitmetes erinevates modelleerimise keeltes *(Konverteeritud dokument).*

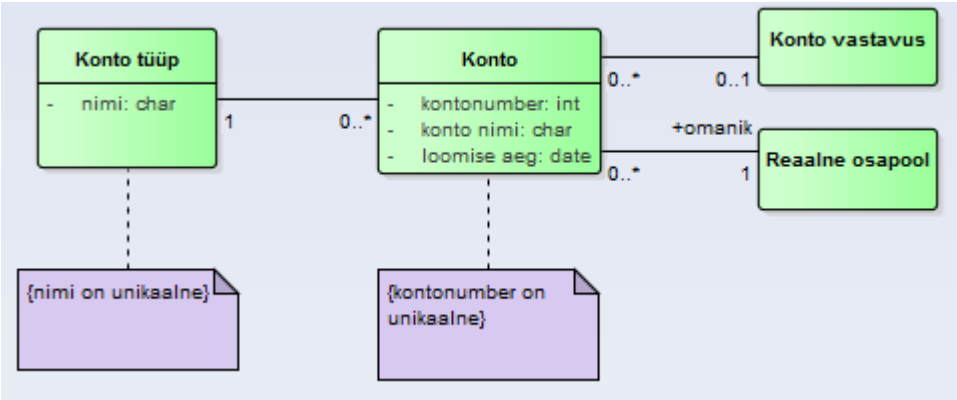
4.3 Muster: Konto

Mustri nimi: Konto (inglise keeles *Account)*

Allikas: Blaha, 2010, 121 (Account)

Probleem: Osapooltel on vajalik registreerida nii oma vara hetkeseisu, kui juurde tulevaid ning lahkuvaid ressursse.

Lahendus :



Joonis 3: Konto muster

Konto on EKSS (2015) definitsiooni põhjal „majanduslikke vahendeid ja nende allikaid ning majanduslikke tehinguid käsitleva informatsiooni rühmitamise ja jooksva arvestuse tabel raamatupidamises“. Konto mustris (vt Joonis 3) vaadeldakse konto mõistet veelgi laiemalt ning raamatupidamine on vaid üks konto mõiste kasutamise valdkondadest.

Laiemalt on konto midagi, mille abil on võimalik jäädvustada ning hallata ressursse. *Konto tüüp* kirjeldab, millise kontoga on tegemist – kas e-maili konto, pangakonto, kulukonto või muud sarnast.

Kontodel on olemas omanik, kellele vastab element *Reaalne osapool*. Reaalne osapool on isik, organisatsioon või programm, millega pole seotud rolli.

Konto vastavus on konto puhul vajalik, kui on vaja liigutada ressursse mitme konto vahel ning jäädvustatud on seos mitme konto vahel.

Mustri puudused: Mustris ei ole kontodes kirjeldatud konto hetkeseisu. Kontode puhul on hetkeseis oluline parameeter, kuna konto objekti põhieesmärk on anda ülevaade osapoole käsutuses olevatest vahenditest. Hetkel seda mustrit kasutades puudub konto omanikul ülevaade kontol toimuvast. Samamoodi on oluline hoida kontode kohta logi sisse tulevatest ja välja liikuvatest transaktsioonidest. Selle kohta pakub Blaha (2010, 141) välja arhetüübi *Transaktsioon*.

Seotud mustrid: Kontoga on seotud *Osapoole muster* (vt Joonis 5), sest kontol on omanik.

Näited:

- Ravimifirmal AS Ravim *(Reaalne osapool)* on kahes erinevas pangas pangakontod *(Konto tüüp)*. Et firma jaoks oleks võimalik liigutada summasid kahe panga vahel (erinevate kontode vahel) märgib *Konto vastavus* element seose kahe panga vahel.

- Mööblidetailide firmal *(Reaalne osapool)* asuvad tooted erinevates ladudes *(Konto tüüp)*. Firmal on üle maailma mitmeid omavahel seotud ladusid, mida seostab *Konto vastavuse* element.

- Isikul Johnil *(Reaalne osapool)* on e-maili konto *(Konto tüüp)*. Sellel kontol saab isik jälgida ning hallata oma kirjavahetusi. E-maili konto võib olla seotud mitme teise teenusepakkuja e-maili kontoga.

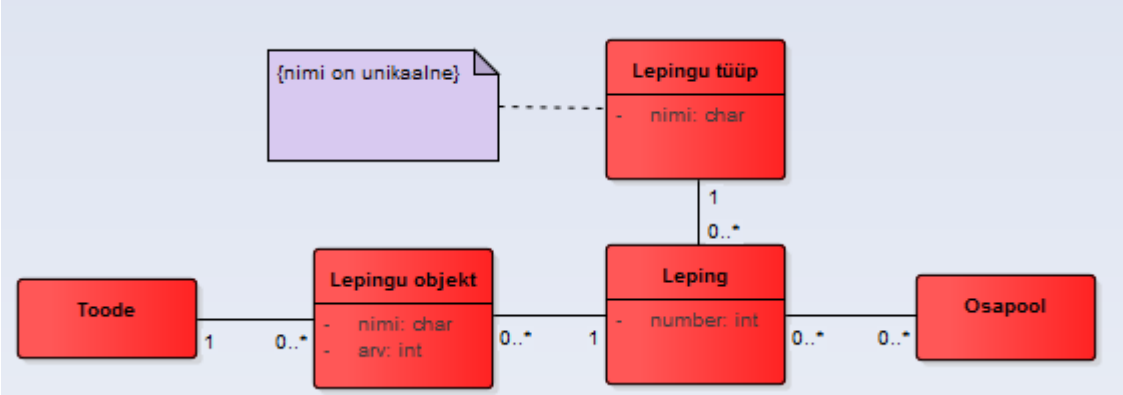
4.4 Muster: Leping

Mustri nimi: Leping (inglise keeles *Contract)*

Allikas: Blaha, 2010, 125 (Contract)

Probleem: Süsteemis on vaja kirjeldada osapoolte vahel sõlmitud kokkulepet, mille täitmise tulemusena, kui kõik järgivad lepingus toodud tingimusi, siis saavad nad ka lepinguga määratud hüvesid.

Lahendus :



Joonis 4: Lepingu muster

Lepingu mustris (vt Joonis 4) *Leping* sõlmitakse ühes või mitme *Osapoole* vahel. Blaha peab mustris oluliseks seda, et lepinguga peab olema seotud reaalne osapool (näiteks haigla AS Haigla), mitte roll nagu „teenuse pakkuja“. Leping rakendub toodetele, mida nimetatakse *Lepingu objektideks*. Lepingu objekte peab osapool esitama teatud koguse (*arv*). *Lepingu tüübi* objekt aitab lepinguid liigitada. Lepingu tüüpide näited on: tööleping, ostutellimus, arve ja reklaamleping.

Mustri puudused: Osapoole rolli kirjeldamine konkreetse lepingu kontekstis peaks käima kaudselt, läbi *Osapoole mustri*, kus saab kirjeldada rolle, mis on ka osapooled. Leping tuleks siduda osapooltega, mis on rollid (vt *Osapoole muster Joonis 5*). See komplitseerib mudelit ja selle alusel loodud tarkvara.

Samuti on siinses mustris lepingu objektil määratud seos ainult tootega. Reaalsuses võib lepingu objekt olla ka osutatav teenus. Samuti pole kirjeldatud võimalust, et lepingu abil vahetavad osapooled finantsressursse. Näiteks juhul, kui sõlmitakse tööleping, pakub üks osapool teenust (oma aega ning oskuseid) ning teine osapool pakub vastu kokku lepitud rahasumma.

Mustris ei ole kirjeldatud ka lepingu objekti koguse määramiseks kasutatavat ühikut. Ühik on vajalik olemitüüp, kuna erinevatel objektidel võib see olla erinev ning seda on võimalik ka konverteerida teisteks samaväärseteks kogusteks. Lepingu objekti atribuut *arv* ei kirjelda objektide hulka piisavalt täpselt ning võib öelda, et tegu on Blaha poolt liigse lihtsustusega.

Lepingul on suure tõenäosusega olemas tingimused, millest mõlemad osapooled peavad kinni pidama. Tingimusteks võivad olla tähtajad või muud osapoolte jaoks toote üle andmise juures olulised kriteeriumid. Lepingu tingimusi võetakse arvesse toote loomise protsessis ülesannete täitmisel. Muster ei näe ette võimalust nende tingimuste fikseerimiseks.

Seotud mustrid: Lepinguga on seotud *Osapoole muster* (vt Joonis 5), mis kirjeldab täpsemalt lepingu sõlmijaid**.** Lepinguga on seotud ka *Toote muster* (vt Joonis 8), kus kirjeldatakse toote olemust ja omadusi.

Näited:

- Kohvi tootja ja kauplus on *Lepingu Osapooled*. Lepingus sätestatakse, et kohvi tootja peab pakkuma kauplusele *Toodet* „Kohviuba“ koguses 100 pakki. 100 pakki „Kohviuba“ on *Lepingu objekt.* Kuna tegu on toote vahetamisega rahasumma vastu, siis on tegu ostulepinguga, mis on ka *Lepingu tüübiks.*

- Kaks firmat *(Osapoolt)* sõlmivad koostöölepingu *(Lepingu tüüp)*, milles võib olla sätestatud konkreetne toode (*Lepingu objekt),* mida hakatakse koos välja töötama, kuid konkreetne objekt võib ka puududa.

- Eraisik *(Osapool)* on kohustatud käsunduslepingu *(Lepingu tüüp)* alusel valmistama partii lusikaid *(Lepingu Objekt)*. Üksik lusikas on *Toode*.

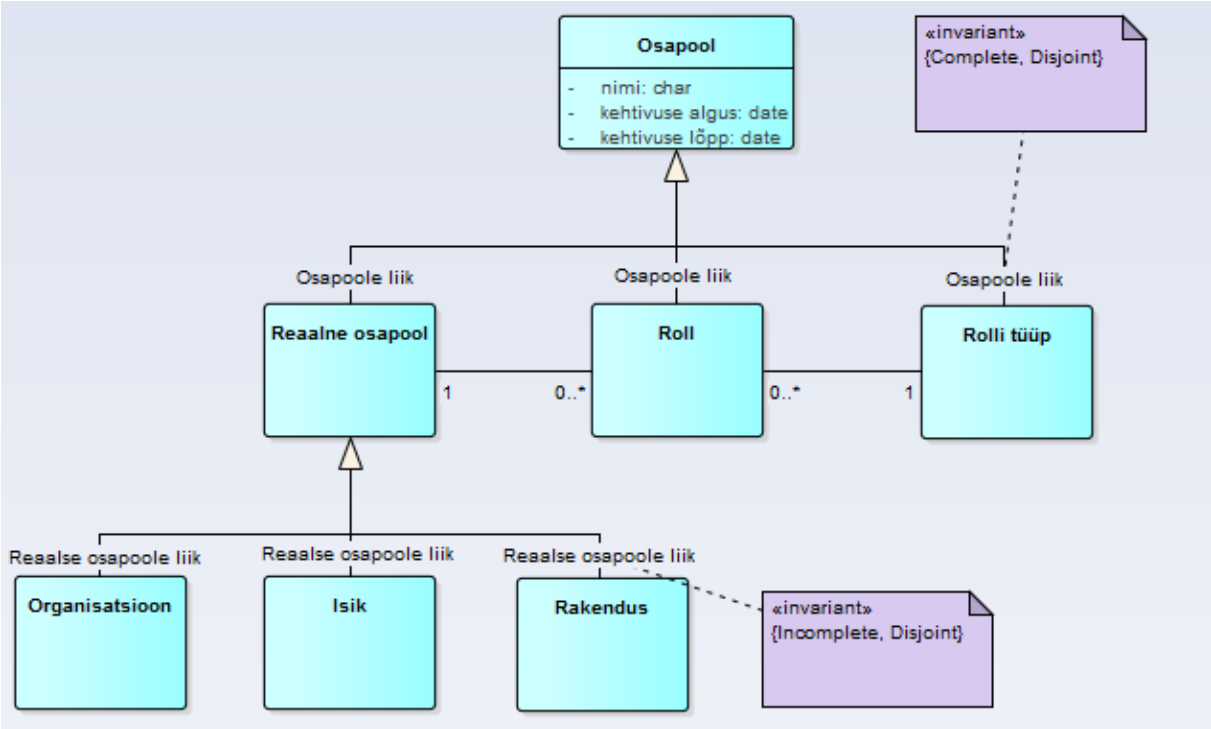
4.5 Muster: Osapool

Mustri nimi: Osapool (inglise keeles *Party)*

Allikas: Blaha, 2010, 122 (Actor)

Probleem: Kuidas käsitleda süsteemis tegevusi läbi viivaid isikuid või asutusi nii, et piisavalt üldistatult oleks käsitletud kõiki osapoolte jaotusi?

Lahendus :



Joonis 5: Osapoole muster

Osapool on keegi või miski, mis võtab osa süsteemis toimuvatest protsessidest. Osapoole mustris (vt Joonis 5) *Osapool* on *Reaalse osapoole***,** *Rolli* ja *Rolli tüübi* üldistus.

Reaalne osapool on konkreetne *Isik*, *Organisatsioon* või *Rakendus*. Piisavalt arukas rakendus, põhimõtteliselt agent, võib olla andmete vahetusel oluline osaline. Reaalseks osapooleks võib olla näiteks isik John Smith või organisatsioon AS Haigla.

Rolli tüübiks nimetatakse positsiooni, millega võivad olla seotud füüsilisest osapoolest eraldi õigused, kohustused või muud omadused. Rolli tüüpideks võivad olla raamatupidaja, arve saatja või maksja.

Rolli objektil on seosed nii Reaalse osapoolega kui Rolli tüübiga, kuna roll on ühend neist kahest. Rolliks on näiteks raamatupidaja (*Rolli tüüp*) John Smith (*Reaalne osapool*) või toote valmistaja (*Rolli tüüp*) AS Mööblitehas (*Reaalne osapool*).

Mustri puudused: Antud muster on väga üldine ning selle abil ei ole võimalik kirjeldada erinevaid rolle eraldi olemitüüpidena.

Näiteks luues selle mustri järgi mudelisse olemitüübi *Transportija*, mis on *Organisatsiooni* alamklass, on *Roll* ja *Rolli tüüp* liiased ning relevantne on ainult *Reaalne osapool.*

Seotud mustrid: Osapoolel võib olla asukoht, millele vastab *Asukoha muster* (vt Joonis 1).

Näited:

- Süsteemis on vajalik kirjeldada firma töötajate õiguseid. Selleks kirjeldatakse *Osapool* firma juhataja *(Rolli tüüp), Osapool* Ivo Tellis *(Reaalne osapool, Isik)* ja *Osapoole* juhatajaks olemine *(Roll).* Viimasel osapoolel on õigused alates hetkest, kui ta tööle asub *(kehtivuse algus)* kuni töölt lahkumiseni *(kehtivuse lõpp).*

- Päringu tegemiseks peab isik pöörduma *Rakenduse* poole, mis on *Osapool.* Rakenduse päringu vastu võtjaks olemist kirjeldab *Roll* ning *Rolli tüübiks* on päringu vastu võtja.

- Isik Madis Karu *(Osapool)* võib olla ühe kaubandusettevõtte jaoks nii *Rolli tüübiga* klienditeenindaja kui ka *Rolli tüübiga* klient. Seega, kui ta ostab mõnda toodet, on ta kirjeldatud *Rollis* klient Madis Karu ning kui ta müüb kliendile, siis on ta *Rollis* klienditeenindaja Madis Karu.

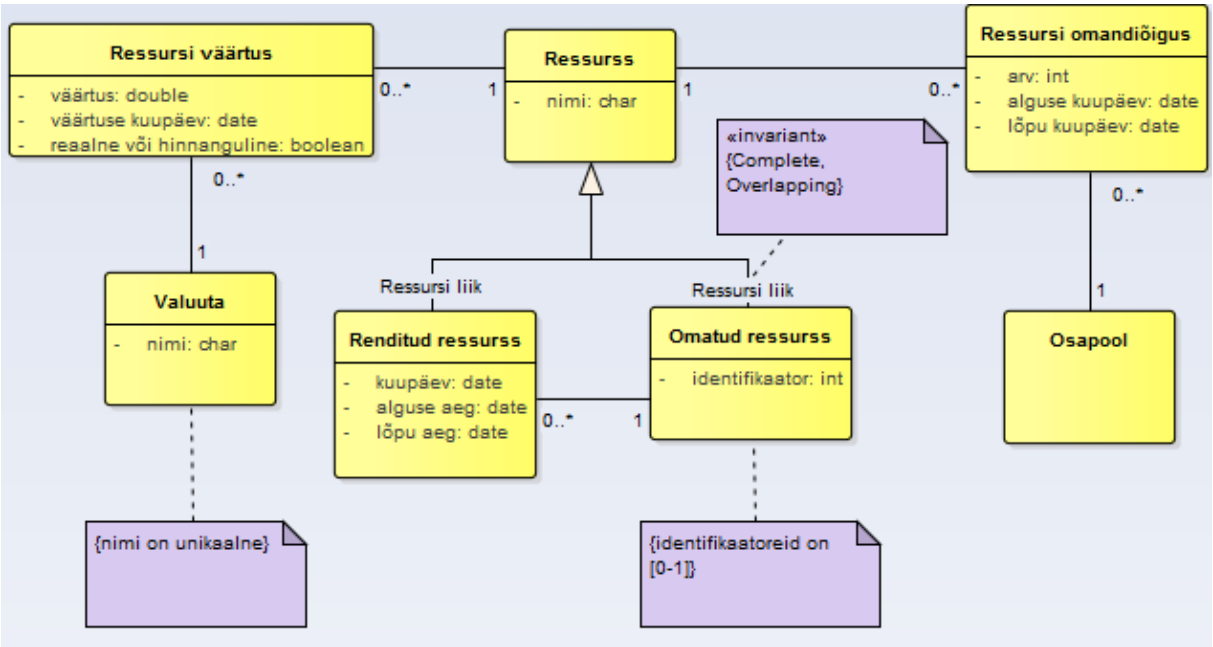
4.6 Muster: Ressurss

Mustri nimi: Ressurss (inglise keeles *Resource)*

Allikas: Blaha, 2010, 124 (Asset)

Probleem: Kuidas käsitleda osapoolele kuuluvat kapitali, mida saaks rakendada ülesannete sooritamiseks?

Lahendus :



Joonis 6: Ressursi muster

Ressursi mustris (vt Joonis 6) *Ressurss* on süsteemi *Osapoole* omand, millele on määratud *Ressursi väärtus.*

Ressurss on seotud osapoolega läbi *Ressursi omandiõiguse* objekti. Omandiõigus kirjeldab omatud ressursi kogust *(arv)* ning ressursi ja osapoole seose kestmist *(alguse kuupäev, lõpu kuupäev)*.

Ressursiga on seotud ka väärtus, et selle omanik saaks ressursi kasutamise kulusid hinnata. Ressursi väärtust ei ole alati võimalik täpselt välja arvutada ning sel juhul määratakse see

hinnanguliselt – väärtuse andmist kirjeldab Boolean väärtus *reaalne või hinnanguline.* Väärtused võivad olla ka erinevates *Valuutades***.**

Ressurss võib olla osapoole poolt kasutuses vaid teatud ajavahemikus, mille korral on tegu *Renditud ressursiga*. Kõige tüüpilisem olukord on see, kui ressurss kuulub otseselt osapoolele, see on *Omatud ressurss*.

Mustri puudused: Ressursi omandiõigus peab kirjeldama omatud kogust, kuid sellel puudub ressursi ühiku kirjeldus.

Ressursi mustris ei ole kirjeldatud võimalus, et ressursid võivad olla erinevat liiki. Ressurss võib kujutada nii kasutatavat territooriumi, töövahendeid, tooraineid kui ka töötajate poolt tehtavat tööd.

Seotud mustrid: Ressursil on olemas omandiõiguse läbi omanik, mida täpsustatakse *Osapoole mustriga* (vt Joonis 5) Ressursil võib olla konkreetne asukoht, mida kirjeldatakse konkreetsemalt *Asukoha mustri* (vt Joonis 1) all.

Näited:

- Ravimifirmas *(Osapool)* tahetakse luua juurde ravimit X *(Omatud ressurss)*. Selleks tuleb ravimi loomiseks määrata asutuse töötajaid, kelle kasutatav aeg on *Renditud ressurss,* ning tuleb kasutada ravimi komponente/tooraineid, mis on firma *Omatud ressursid*. Toorainete väärtuseks *(Ressursi väärtus)* on määratud 8000 eurot.

- Tarkvarafirma *(Osapool)* ei oma kontorit *(Ressurss).* Kontori pind renditakse *(Renditud ressurss)* ning vastavalt rendilepingule on määratud ka kontori väärtus *(Ressursi väärtus).*

- Töötaja *(Osapool) Ressursiks* on aeg, mida ta rendib välja firmale, kus ta töötab. Töötaja jaoks on tema aeg *Omatud ressurss,* kuid firma jaoks on see *Renditud ressurss.* Töötaja ajal on hinnanguline väärtus, mis määratakse töölepingus.

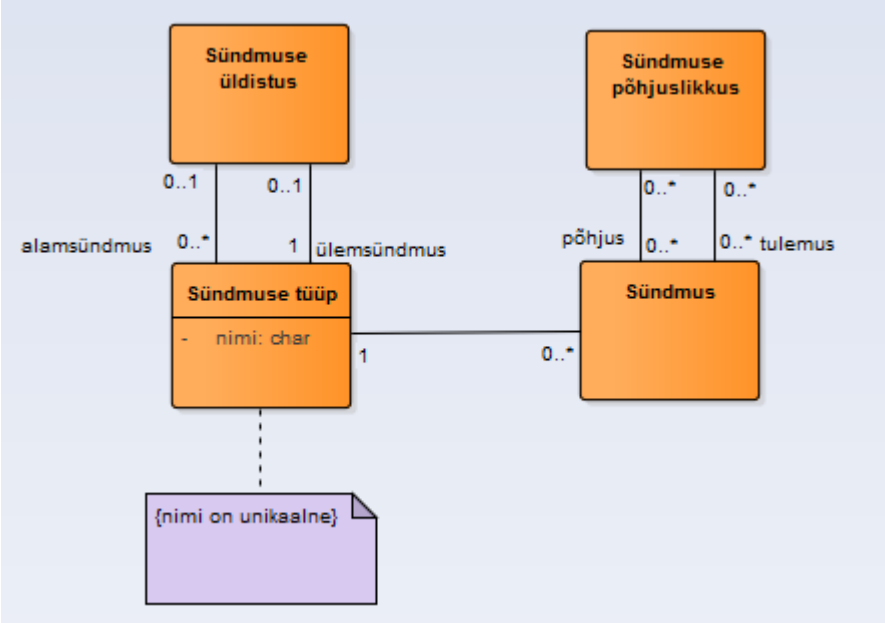
4.7 Muster: Sündmus

Mustri nimi: Sündmus (inglise keeles *Event)*

Allikas: Blaha, 2010, 130 (Event)

Probleem: Süsteemis on vaja eraldi objektidena kirjeldada sündmused, mis kutsuvad esile teisi protsesse ning mille tulemusel tekivad teised süsteemi objektid.

Lahendus :



Joonis 7: Sündmuse muster

Sündmuse mustris (vt Joonis 7) kirjeldatav *Sündmus* on konkreetsel ajahetkel aset leidev *nähtus ehk olukord*, mis võib omakorda põhjustada teisi sündmusi. *Sündmuse tüübid* on jagatud hierarhiasse, kuna ühel tüübil võivad ülem- ning alamsündmused. Näiteks alamsündmusel kohviubade purustamine on olemas ülemsündmus kohvi valmistamine. Kohvi valmistamisel võib olla omakorda ülemsündmus, näiteks kliendi tellimuse täitmine.

Sündmuse põhjuslikkuse objekt kirjeldab olukorda, kus üks sündmus võib olla põhjustatud mitmest teisest sündmusest ning see sündmus võib omakorda põhjustada teiste sündmuste toimumise.

Mustri puudused: Sündmuse kirjeldus on väga üldine ning selle tõttu on raske mudelis selle mustri põhjal kirjeldada konkreetseid sündmuse toimumise põhjuseid ning tagajärgi. Element *Sündmuse põhjuslikkus* on liiga üldine. On võimalik küll *Sündmuse* element määrata konkreetsete sündmuste üldistuseks, kuid muid tüüpi seoseid sündmuste vahel see muster ei toeta.

Sündmuste üldistuste hierarhia ei toeta mitmest pärimist.

Seotud mustrid: Sündmuse asukohta kirjeldab täpsemalt *Asukoha muster* (vt Joonis 1). Sündmusega seotud isikuid kirjeldab lähemalt *Osapoole muster* (vt Joonis 5).

Näited:

- Isik tellib baristalt kohvi, mis on *Sündmus,* mille tüübiks on tellimine. *Sündmuse Põhjuslikkuse* läbi põhjustab see *Sündmuse,* mille tüüp on kohvi valmistamine. Kohvi valmistamise alamsündmused on näiteks kohviubade jahvatamine, kohvi keetmine, piima lisamine ja nii edasi.

- Isikule saadetakse pildifail oma puhkusest. Selle isiku jaoks on puhkuse pildi kätte saamine *Sündmus*, mille tüübiks on pildi faili kätte saamine. Läbi *Sündmuse põhjuslikkuse* on see seotud sündmusega, kus isik vaatab kätte saadud pilti.

- Kui üliõpilane deklareerib oma õppekava *(Sündmus),* on õppeinfosüsteemi jaoks tegu õppekava deklareerimise *Sündmuse tüübiga. S*elle tulemusena õppeinfosüsteem küsib üliõpilaselt selle õppekava kinnitamist *(Sündmus).*

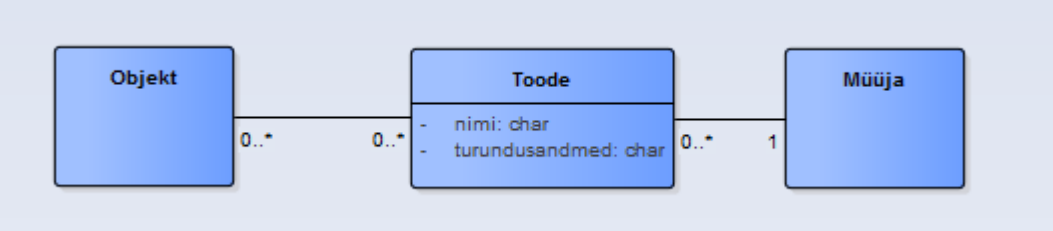
4.8 Muster: Toode

Mustri nimi: Toode (inglise keeles *Product)*

Allikas: Blaha, 2010, 139 (Product)

Probleem: Kirjeldada süsteemis sihipäraste tegevuste tulemusel tekkivat objekti, mida müüja saab soovi korral müüa.

Lahendus :



Joonis 8: Toote muster

Toote mustris (vt Joonis 8) *Toode* on kogum *Objektidest*, mida *Müüja* soovib tooteturul müüa. Üks toode võib koosneda mitmest objektist ning üks objekt võib asuda ka mitmes erinevas tootes. Toote turundusandmed on info toote kohta turul müümiseks. Need andmed hõlmavad näiteks toote hinda ja sihtgruppe.

Mustri puudused: Toote spetsifitseerimisel pole võimalik kirjeldada toote omadusi, mis esitavad osapooltele olulist informatsiooni toote parameetrite kohta. Kui need salvestada turundusandmetes ühe andmeväärtusena (näiteks tekstina), siis on nende lugemine ja muutmine tülikas ja veaohtlik.

Tootel puudub väärtuse kirjeldus. Turundusandmed kirjeldavad küll müümiseks vajalikku informatsiooni, kuid peaks olema võimalik määrata selle rahalist väärtust ja seda eraldi atribuudi väärtusena salvestada. Tootel võib neid väärtuseid olla isegi mitu – omahind, soovituslik müügihind, minimaalne müügihind. Kui vaadata ka kõiki neid toote omadustena, siis on tegemist eelmise lõigu erijuhtumiga.

Seotud mustrid: Toote omaniku täpsustab *Osapoole muster* (vt Joonis 5). Toote asukoha määramiseks kasutatakse *Asukoha mustrit* (vt Joonis 1). Toote valmistamiseks kasutatakse erinevaid ressursse, mida kirjeldab *Ressursside muster* (vt Joonis 6).

Näited:

- Arendusfirma *(Müüja)* loob programmi *(Toote),* mis koosneb mitmetest alammoodulitest ja programmifailidest *(Objektid).* Programmi turuhind ja muud parameetrid on määratud *turundusandmetes*.

- Kauplus ostab sisse *Toote* pliiats selle pliiatsi tootjalt (*Müüja*). Pliiats ei koosne objektidest, vaid ongi üksik toode. Pärast seda, kui kauplus on pliiatsi endale ostnud ja soovib seda edasi müüja, vaadeldakse süsteemis kauplust kui *Müüjat.*

- Raamatupidamisfirma *(Müüja)* aitab väikefirmadel koostada aastaaruandeid, kus iga koostatud aruanne on eraldi *Toode.*

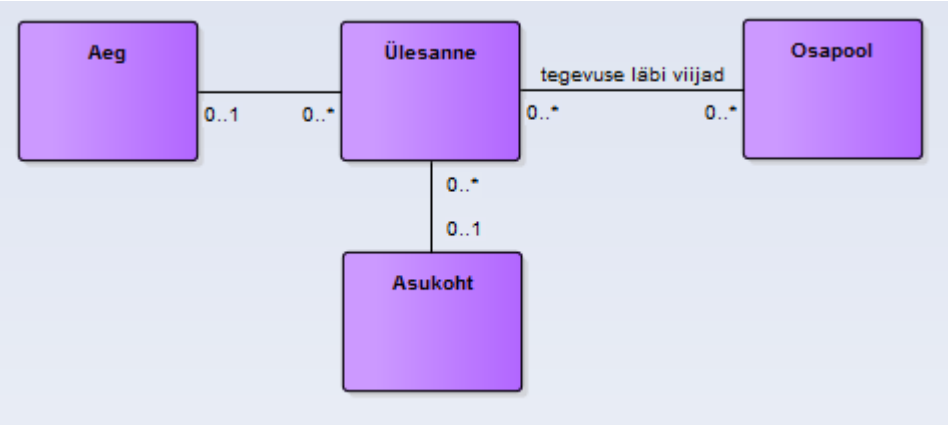
4.9 Muster: Ülesanne

Mustri nimi: Ülesanne (inglise keeles *Task)*

Allikas: Fowler, 1996, 158 (Action)

Probleem: Vajalik on kirjeldada sündmust, mida osapool peab või pidi läbi viima.

Lahendus :



Joonis 9: Ülesande muster

Ülesande mustris (vt Joonis 9) kirjeldatav *Ülesanne* on tegevus, mida *Osapooled* soovivad ühel ajahetkel läbi viia. Ülesannet ei saa samastada sündmusega, kuna sündmuse all mõistetakse igasugust juhtumit, mis toimub suvalisel ajahetkel. Ülesanne on samas pigem plaan, mida osapooled kavatsevad või kavatsesid teha konkreetsel ajahetkel. Ülesandega seotud ajahetke kirjeldatakse elemendiga *Aeg*. Ülesandega võib olla seotud kindel *Asukoht*.

Mustri puudused: Ülesanne ei ole seotud tingimustega, mis sätestavad selle läbiviimise korralduse ega oodatavate tulemustega. Ülesanne pole seotud lepinguga, mille täitmiseks see võib olla tekkinud. Pole võimalik teada saada, kas ülesanne on täidetud või mitte. Ülesandel võib olla mitu erinevat ajahetke – näiteks väljamõtlemise aeg, registreerimise aeg, oodatav täitmise aeg, kõige hilisem lubatud täitmise aeg, tegelik täitmise aeg. Samas saab ülesande antu mudeli alusel seostada vaid ühe ajahetkega. Pole võimalik kirjeldada seda, et ülesanne koosneb alamülesannetest. Ülesandega saab olla seotud maksimaalselt üks asukoht.

Osapoole rolli kirjeldamine konkreetse ülesande kontekstis peaks käima kaudselt, läbi *Osapoole mustri*, kus saab kirjeldada rolle, mis on ka osapooled. Leping tuleks siduda osapooltega, mis on rollid (vt *Osapoole muster* Joonis 5). See komplitseerib mudelit ja selle alusel loodud tarkvara.

Seotud mustrid: Ülesande teostajat kirjeldatakse *Osapoole mustri* (vt Joonis 5) abil. Ülesannet viiakse läbi asukohas, mida täpsemalt kirjeldatakse *Asukoha mustris* (vt Joonis 1).

Näited:

- Ehitusfirma *(Osapool)* annab eraisik Johnile *(Osapool)* ülesande, kus on sätestatud, et ehitusfirma peab Johni elutoa seina värvima järgmisel teisipäeval *(Aeg).*

- *Osapoole* raamatupidaja *Ülesanne* oli maksta välja töötajate novembri kuu palgad ning selleks määratud *Aeg* on päev enne palgapäeva, 5. november. Ülesannet sooritatakse raamatupidaja töökohas, firma kontoris *(Asukoht).* Fakti, kas palkade maksmise ülesanne sai täidetud antud muster ei kirjelda.

- Bakalaureusetöö kirjutamine on *Ülesanne*, mille sooritamise konkreetne *Asukoht* ning *Aeg* pole määratud. Ülesandega on seotud *Osapool* üliõpilane. Küll aga peab olema määratud töö kirjutamise tähtaeg, mida antud muster ei kirjelda.

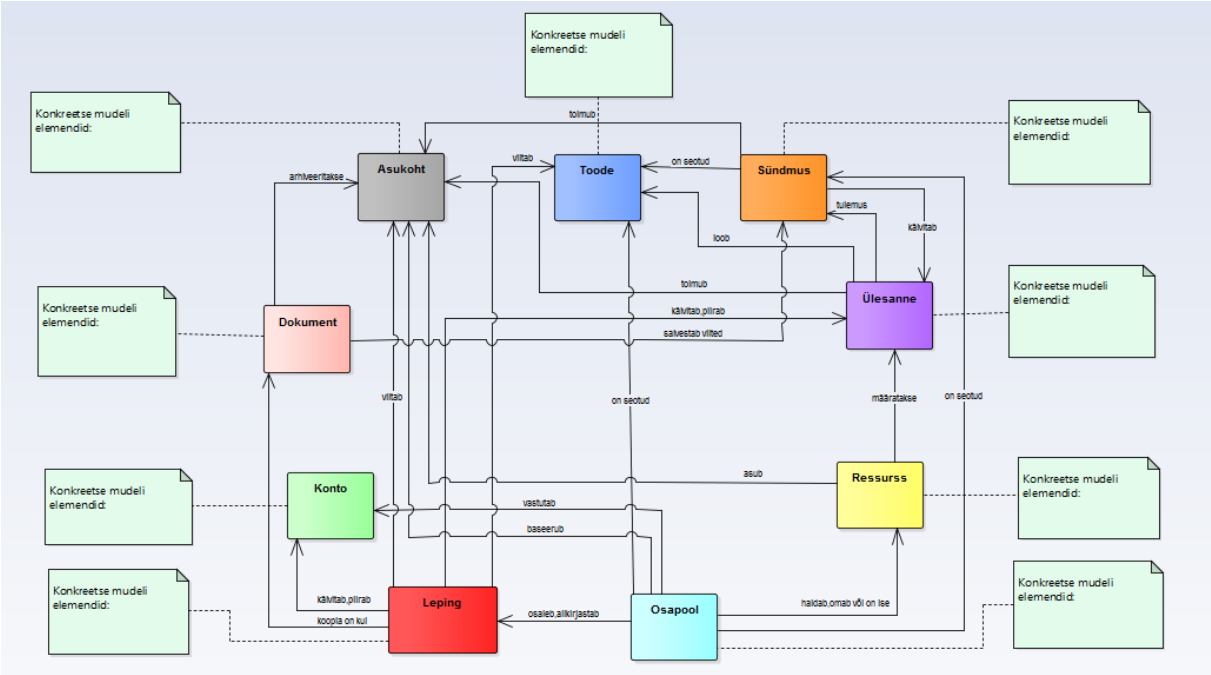
4.10 Integratsiooni muster

Mustri nimi: Integratsiooni muster

Allikad: (Giles, 2012)

Probleem: Süsteemide algsel modelleerimisel on vaja saada ülevaade probleemvaldkonna põhikomponentidest ning nende vahelistest seostest. Vaja on realiseeritud mustrite värvidele legendi.

Lahendus:



Joonis 10: Integratsiooni muster

Integratsiooni muster (vt Joonis 10) on mustrite muster, mis esitab komponentmustrite seostamise raamistiku. See näitab, mis on süsteemi modelleerimisel kasutatavad arhetüübid (põhiolemitüübid), et lihtsustada edasist mustrite valikut süsteemi jaoks. Iga arhetüübiga on seotud kommentaari element, kus mustri kasutaja märgib realiseeritavas mudelis arhetüüpidele vastavad põhiolemtüübid.

Kõikides süsteemides ei pruugi olla vajalik kasutada kõiki põhiolemitüüpe ning integratsiooni mustri kasutamisel tuleb silmas pidada, et see on vaid soovituslik süsteemi struktuur. Reaalne süsteem võib olla palju vähemate elementidega või sellel on vaja kasutada lisaks valdkonna-spetsiifilisi mustreid.

Kuna see integratsiooni muster on võetud aluseks realiseeritud mustrite valikul, on diagrammidel kasutatud komponentmustrite juures samasugust värvigammat nagu integratsiooni mustris.

Mustri probleemid: Nagu punktis *Lahendus* mainitud, ei ole integratsiooni muster täielik ning seda tuleb kohendada vastavalt vajadusele. Mudelisse võib lisanduda võib ka teisi mustreid, mida integratsiooni mustris pole kirjeldatud.

Seosed teiste mustritega: Antud muster kirjeldab seoseid punktides 4.1kuni 4.9 realiseeritud mustrite vahel. Samuti on integratsiooni mustri elementide värvid legendiks alammustrite värvidele.

5. Mustrite realiseerimine ning kasutamine EA CASE vahendis

Käesolevas peatükis on mustrite loomise ning modelleerimisel mustri kasutamise juhend Enterprise Architect töövahendiga. Mustrite realiseerimise ja kasutamise protsessi kirjeldamiseks võtan näiteks punktis 4.5 kirjeldatud Osapoole mustri (Joonis 5).

Enterprise Architect on modelleerimise ning disaini CASE (*Computer-Aided Software Engineering Tools*) vahend, mis on loodud Sparx Systems poolt. See vahend võimaldab muuhulgas luua UML 2 spetsifikatsioonil põhinevaid mudeleid.

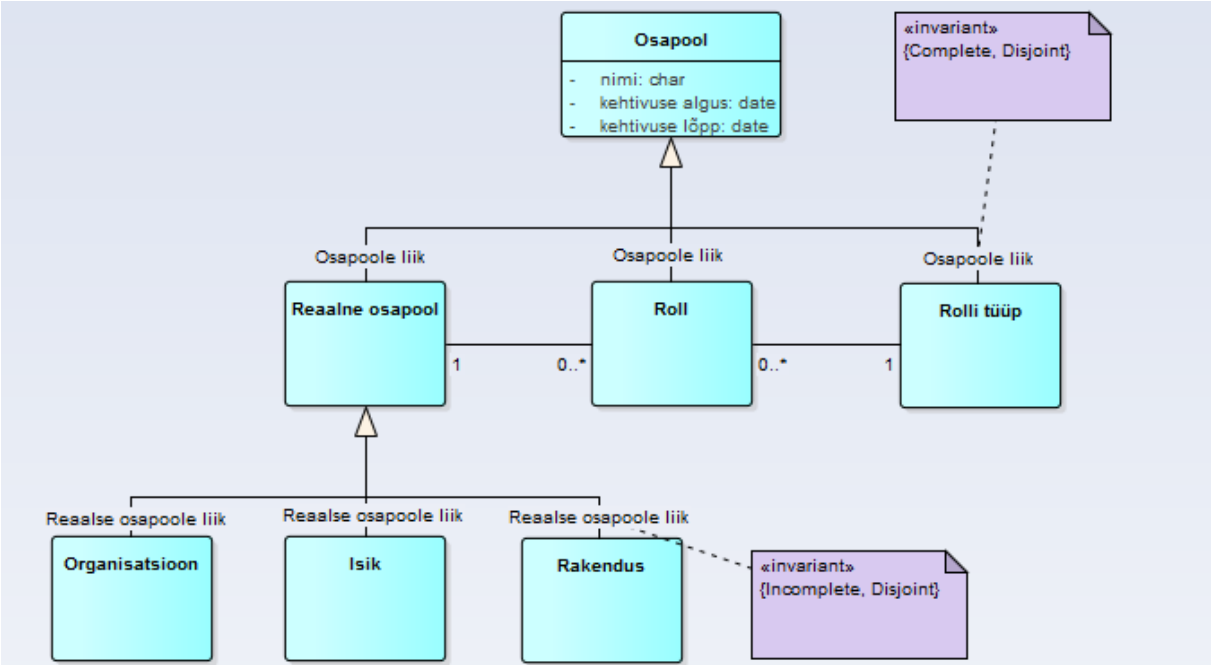
Käesolevas töös on kasutatud Enterprise Architect versiooni 12, mis anti Sparx Systems poolt välja 2015. aastal. Enterprise Architect sisaldab sisseehitatud võimalust taaskasutada ühte ja sama mudelit erinevates projektides. Selle töö korral on nendeks mudeliteks peatükis 4 kirjeldatud mustrid.

CASE vahendi jaoks loodud mustrid on kättesaadavad avalikult veebilehelt: http://apex.ttu.ee/archetypes.

5.1 Mustri loomise protsess

Mustri loomiseks tuleb esimese sammuna luua Enterprise Architecti uus projekt ning selles mustrit kujutav diagramm. Kuna käesolevas projektis realiseeritakse kontseptuaalse struktuuri modelleerimiseks mõeldud mustreid, siis tuleb luua klassidiagramm. Kui soov on luua mitu mustrit, siis ei pea looma mitut erinevat projekti. Piisab ühest projektist, milles mustriteks minevad diagrammid ning nendel diagrammidel olevad elemendid on erinevatel pakettidel.

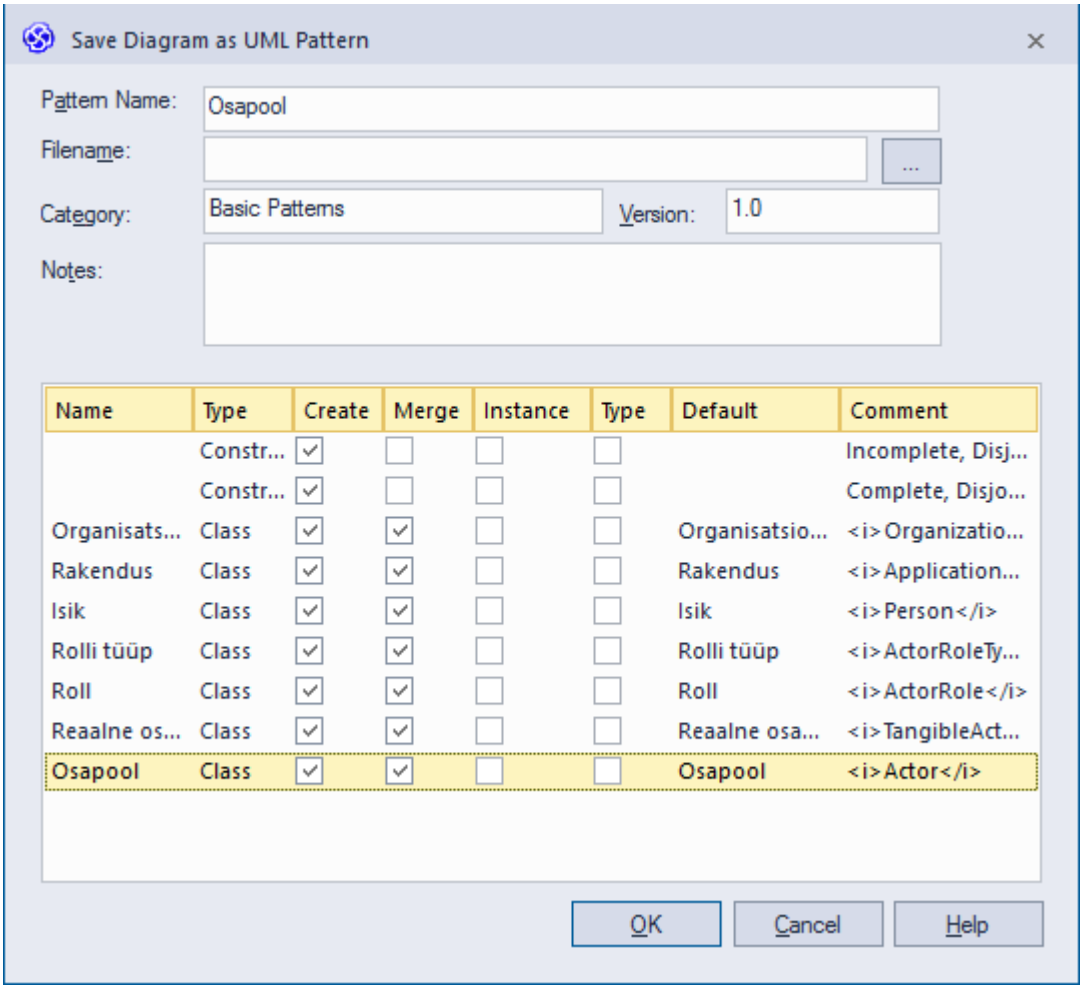
Selleks, et eristada antud mustrit teistest, määran elementidele täitevärvi vastavalt integratsiooni mustris (vt Joonis 10) vastavale arhetüübile määratud värvile. Oletame näiteks, et mustrina on loodud joonisel 11 kujutatud klassidiagramm.



Joonis 11: Mustrina loodud diagrammi näide

Selleks, et oleks võimalik loodud mustrit kasutada teistes mudelites, tuleb see eksportima UML mudelina. Selleks tuleb valida ülevalt menüüribalt *Diagram, Advanced* ning *Save Diagram as UML Pattern.*

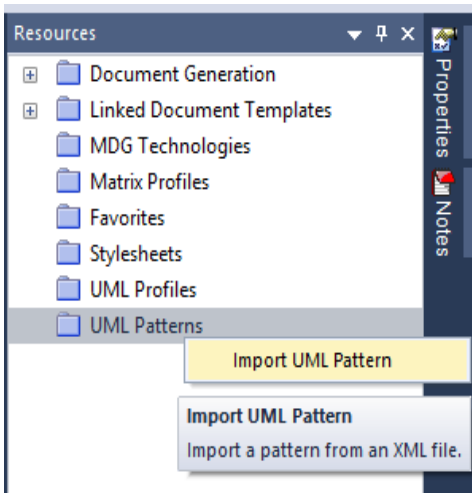
Avanenud aknas (vt Joonis 12) tuleb määrata mustrile nimi, mustri faili nimi ning asukoht ja kaust, kuhu importides muster läheb *(Category).* All osas on loetletud kõik diagrammi elemendid, mis üle viiakse. Oluline on märkida ära elementide teises mudelis loomise võimalused. Automaatselt on kõikidel elementidel *Create* võimalus, mis loob uue elemendi. Juhul, kui on võimalik, et antud element on ka teistes mustrites või see võib juba eksisteerida mudelil, tuleb märkida ka *Merge*. Paindlikkuse suurendamiseks soovitan määrata kõikidele elementidele *Merge* võimalus.



Joonis 12: Diagrammi salvestamine UML mustrina

Nüüd on muster kasutamiseks olemas, kuid edasine kirjeldus käsitleb, kuidas mustrit ka mudelisse tuua ning kuidas Enterprise Architect vahendis mudelit korrastada.

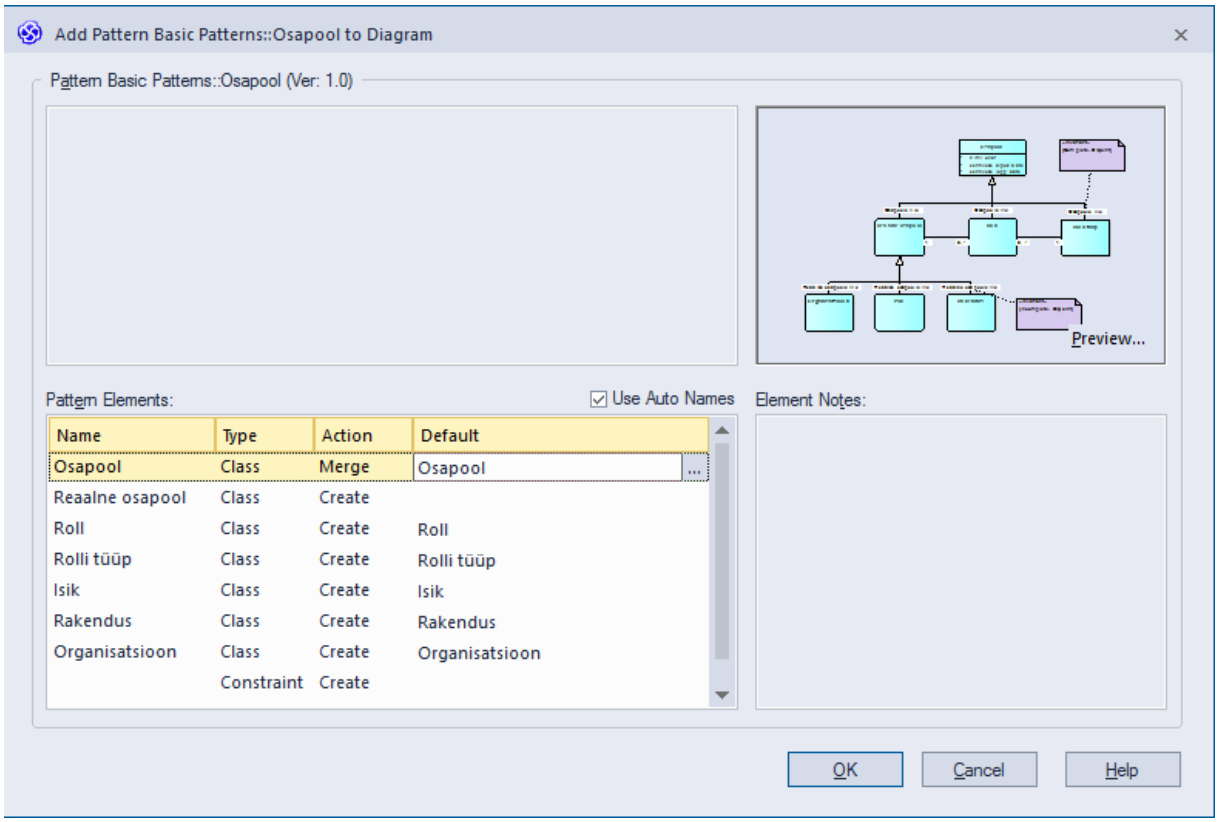
Mudelisse toomiseks tuleb *Resources* tööriistaribalt valida joonisel 13 näidatud menüüpunkt ja avada salvestatud mustri fail.



Joonis 13: UML mustri importimine projekti

Seejärel tuleb lohistada *Resources* alt tekkinud diagramm olemasoleva mudeli peale. Näites on kasutustes mudel, kus on olemas juba Osapoole element. Mudelis on ka eelnevalt märgitud, et aliased on nähtavad ning elementide atribuutidel näidatakse ainult nime.

Mustri üle viimisel avanevas aknas (vt Joonis 14) on näha mustri eelvaade ning võimalused, kuidas mustrit üle viia.



Joonis 14: Mustri importimine mudelisse

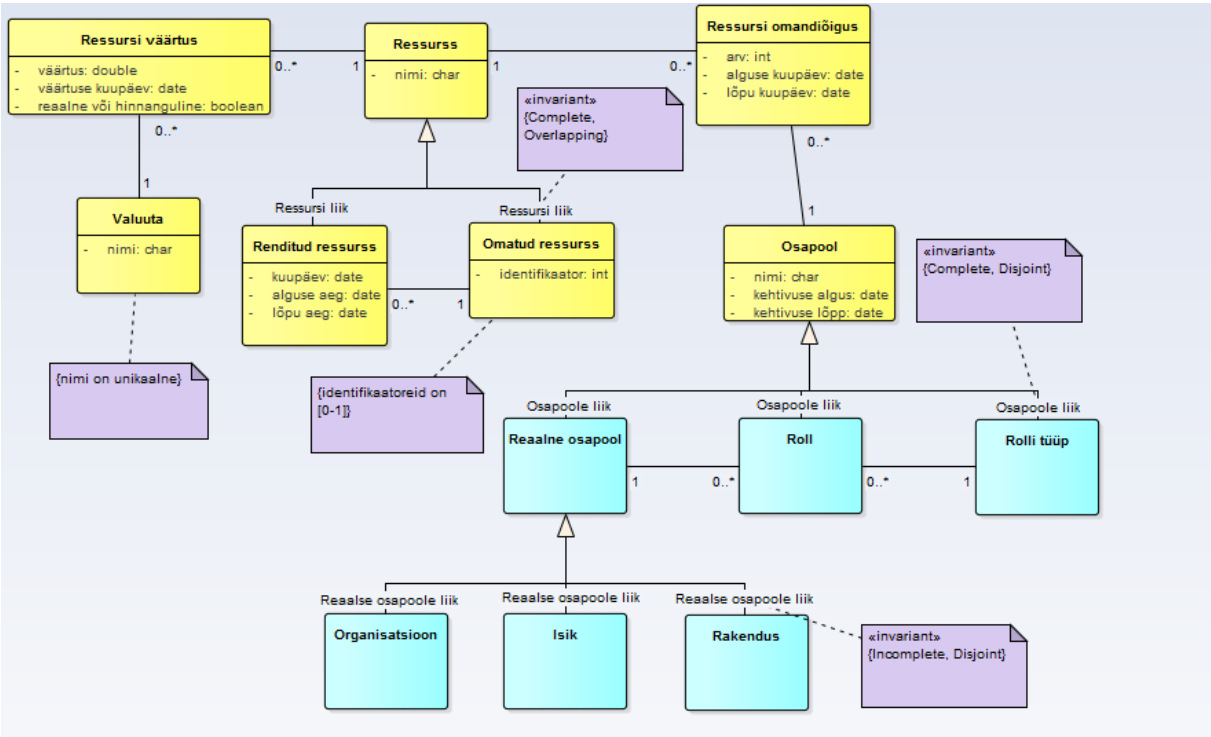
Name on elemendi nimi mustril, *Type* on elemendi tüüp, *Action* kirjeldab viisi, kuidas element üle viia ning *Default* on elemendi uus nimi mudelil. Elemendi nime on võimalik muuta, vajutades ikoonile rea lõpus.

Juhul, kui elemendi üle viimisel on vaja see kokku sulatada juba olemasoleva elemendiga, tuleb valida *Actioniks Merge* ning seejärel rea lõpus kolme punktiga ikooni kaudu määrata element, millega element kokku panna.

Kui elemendid paiknevad diagrammil ebasobilikult, saab neid automaatselt ümber paigutada ikooni (vt Joonis 15) alt, kust saab valida sobiliku paiknemisviisi. Enterprise Architect vahend on paigutamise poolest siiski veel üpris ebatõhus ning lihtsam on iseseisvalt elemente ümber paigutada.

Joonis 15: Diagrammil elementide paiknemise muutmise ikoon

Lõpptulemusena on üle toodud mustriga mudel kujutatud joonisel 16.



Joonis 16: Mustri lisamisel tekkinud mudel

5.2 Tähelepanekud, eripärad, soovitused

Järgnevalt on loetelu tähelepanekutest Enterprise Architecti diagrammide ning mustrite loomise kohta:

- Mustrite importimisel ei ole võimalik ühtegi mustrit jätta sisse toomata, mille tõttu on oluline tähistada üleliigsed elemendid teisiti, et diagrammilt oleks neid lihtsam kustutada.

- Enterprise Architect vahendis ei toimi hästi diagrammil olevate mudelielementide paigutuse muutmise funktsioon. Valides mistahes paigutuse, hakkavad seosed ristuma ning diagrammi loetavus pigem halveneb.

- Elementide atribuutide kitsendusi ei ole võimalik diagrammilt näha. Selle probleemi lahenduseks võib luua täiendava *Constraint* elemendi, kus kirjeldada kitsendus ning see siduda klassiga.

- Esineb probleem, kus alati ei pakuta varianti UML mustreid importida *Resources* all. Probleemi lahendab programmi või projekti taaskäivitamine.

- Elementide atribuutide kirjeldamisel ei ole võimalik valida mitmeid laialt levinud andmetüüpe. Näiteks on puudu kuupäevatüüp ning kellaaja tüüp. Probleemi lahendus on defineerida mudelis uus andmetüüp.

6. Realiseeritud analüüsimustrite kasutamise näide e-arvete infosüsteemi põhjal

Selles punktis luuakse realiseeritud mustrite põhjal e-arvete süsteemi alamosa mudel. See illustreerib, et need mustrid on piisavalt universaalsed, et kasutada neid vabalt valitud süsteemi modelleerimisel.

Kuna e-arvetega on seotud väga palju tegevusi ning alamosi, vaadeldakse ainult ühte osa e-arve infosüsteemis. Võimalik on ka kirjeldada kogu süsteem, kasutades realiseeritud mustreid, kuid sel juhul oleks mudel väga mahukas. Töös on käsitlemiseks valitud e-arve maksmise alamsüsteem, kuna olen selle arendamisega ise kokku puutunud.

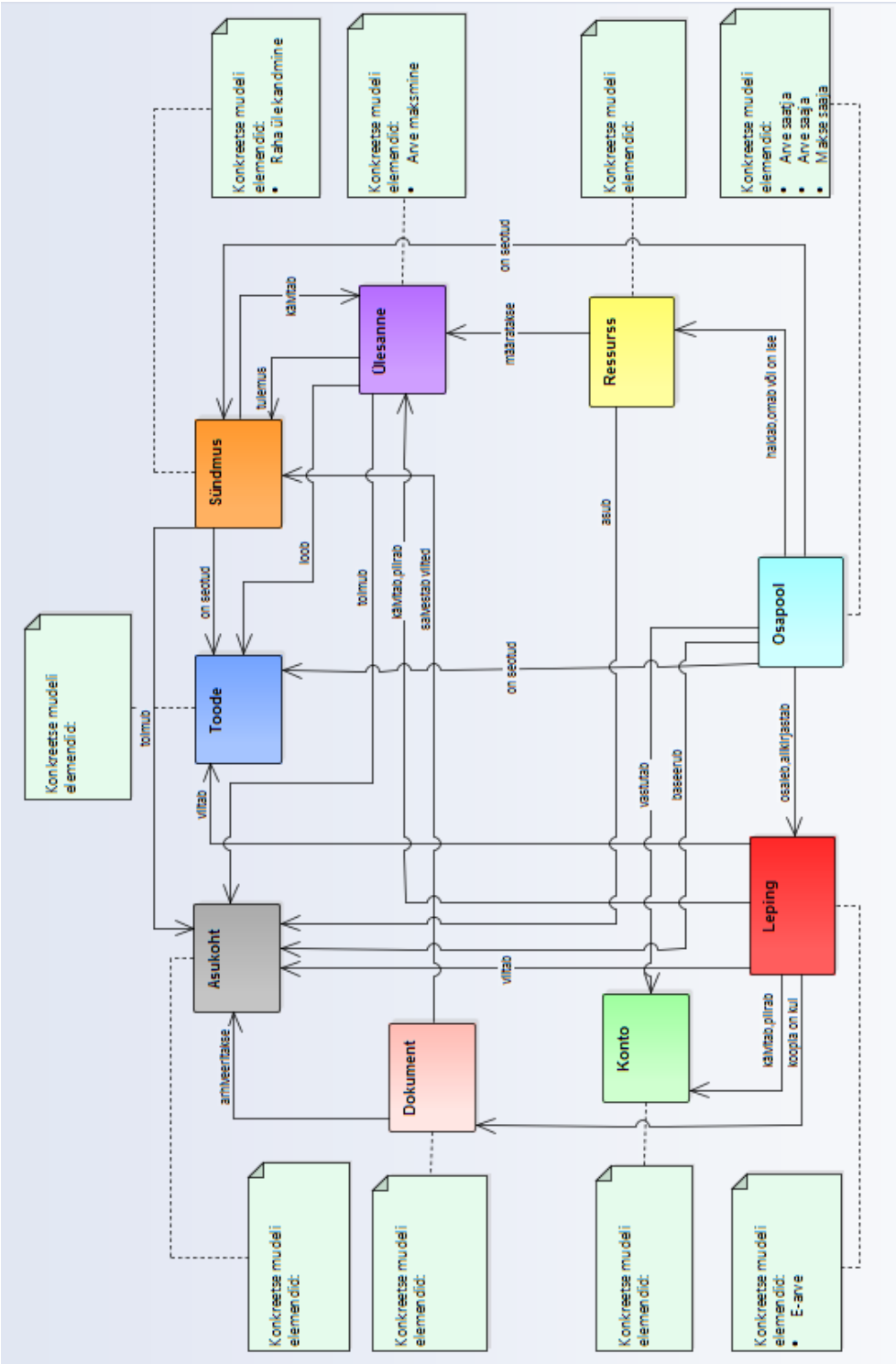
6.1 Arhetüüpide valimine integratsiooni mustri põhjal

Loodavas mudelis kasutatavad arhetüübid, mis on ühtlasi põhiolemitüüpide nimekirja esimeseks versiooniks ja millest tulenevad komponentmustrid, saab valida lähtuvalt punktis 4.10 kirjeldatud integratsiooni mustrist. Integratsiooni mustris on graafiliselt kujutatud seosed põhiolemitüüpide vahel, kuid juhul, kui loodavas mudelis ei soovita mingit olemitüüpi või seost kasutada, saab selle lihtsalt imporditud integratsiooni mustri põhjal tehtud diagrammist kustutada. Tulemuseks saadakse modelleeritava süsteemi integratsiooni mudel. E-arve maksmise integratsiooni mudelit kujutab Joonis 17.

Juhul, kui mudelis soovitakse panna ühele arhetüübile vastama mitu olemitüüpi või nimetada olemitüüp mudelisse importimise käigus ümber, tuleb olemitüübiga seotud märkmele kirjutada uued olemitüüpide nimed.

E-arve maksmise alamsüsteemi mudelis on kasutuses kõik arhetüübid, kuid mõned neist vastavad mitmele olemitüübile ning on erineva nimega. Selleks on arhetüüp Leping, millele vastab käsitletavas süsteemis E-arve ning Osapool, millele vastavad *Makse saaja*, *Arve saaja* ning *Arve saatja*.

Integratsiooni mudel on samuti sisse toodud arhetüüpide värvilegendiks, kuna sealsed värvid on vastavuses realiseeritud mustrite värvidega.



Joonis 17: E-arve maksmise integratsiooni mudel

Lepingu all mõistetakse e-arvet, kuna praeguse süsteemi kontekstis on e-arve kokkulepe osapoolte vahel kindlaks määratud toodete eest tasumiseks.

Osapoolena käsitletakse süsteemi alamosas ainult arve saajat, arve saatjat ning makse saajat. On võimalik, et arvega on seotud ka näiteks toote transportija või muud tootega seotud osapooled, kuid kuna käsitletakse ainult arve maksmise tegevust, siis pole antud mudelis need olulised. Antud süsteemis eeldatakse lihtsuse mõttes ka seda, et arve saaja on sama, kui maksja – see tähendab, et arve saaja on arve liikumise lõppsiht, kellele see on ka mõeldud.

Mudelis on ülesandeks arve maksmine. Tegelikkuses tuleb enne e-arvete maksmist arve kinnitada veel teiste osapoolte poolt, kuid kuna see protsess on komplitseeritud ning selle protsessiga on seotud mitmed objektid, käsitletakse seda eraldi e-arve kinnitamise alamsüsteemina.

Loodav valdkonnamudel kirjeldab eelkõige e-arvete maksmise olemitüüpe ning nendevahelisi seoseid, seega pole oluline tuua sisse kõrvalisi protsesse.

Ülesande tulemusena toimub Sündmus, milleks on Raha üle kandmine. Taaskord – e-arvete süsteemiga on seotud mitmed teised sündmused nagu näiteks arve kohale jõudmine, kuid neid tuleb vaadelda vastavates alamosades.

6.2 Arhetüüpide alusel loodud mudel

Kui integratsiooni mustri põhjal on kindlaks määratud mudelis kasutatavad arhetüübid (põhiolemitüübid), siis nende põhjal on võimalik tuua sisse peatükis 4 loodud mustrid. Kui tuua kõik mustrid kokku ühte diagrammi, siis on raske saada süsteemi struktuurist ülevaadet. Selle tõttu on ka e-arvete maksmise mudel jaotatud väiksemateks osadeks. Põhiline diagramm sisaldab kõiki integratsiooni mustri olemitüüpe, kuid alamdiagrammides on mudeli osad täpsemalt ära kirjeldatud.

Algses mudelis ei ole eemaldatud üldiselt liiasusi ning ei ole lisatud puuduvaid elemente. Antud mudel kirjeldab süsteemi, tuginedes puhtalt vaid integratsiooni mudelis kirjeldatud põhiolemitüüpidele ning neile vastavatele komponentmustritele. Siiski on mustrite mudelisse importimisel mõningaid elemente ümber nimetatud ja mõningaid importimata jäetud.

E-arve maksmise algse mudeli leiab Joonis 18. Joonis 19 - Joonis 24seletavad sellel välja toodud põhiolemitüüpe lähemalt lahti. Peamiseks mudeliks on E-arve maksmise algne mudel, kus on graafiliselt kujutatud maksmise allsüsteemi ning sellega seotud allsüsteemide põhiolemitüübid ning nendevahelised seosed.

E-arve vastab arhetüübile Leping, kuna seal kujutatakse kokkulepet osapoolte vahel e-arvete objektide vahetamiseks teatud tingimustel. E-arve tüübiks võib olla näiteks kreedit- või deebitarve. Tüüpe võib olla ka teisi. E-arve säilitatakse dokumendi kujul, mis peab olema algselt masinloetaval kujul. Hiljem on võimalik seda konverteerida ka paberdokumentideks ning tõlkida inimloetavale kujule. E-arvete objektid on arvega seotud tooted, mille eest oodatakse e-arve saajalt tasu. Toode on kogum objektidest, mille müüjaks on antud infosüsteemis osapool, kes saatis e-arve.

E-Arvega võib olla seotud null või rohkem osapoolt, mis ei ole päris korrektne, kuna e-arvel peab olema vähemalt kaks osapoolt. Selliseid mustris leiduvaid vigu, mis antud infosüsteemis on loogiliselt vale, muudetakse parandatud mudelis.

Integratsiooni mudelis märgitud põhiolemitüübi ning mustrite elementide vastavust on võimalik kirjeldada kahte moodi:

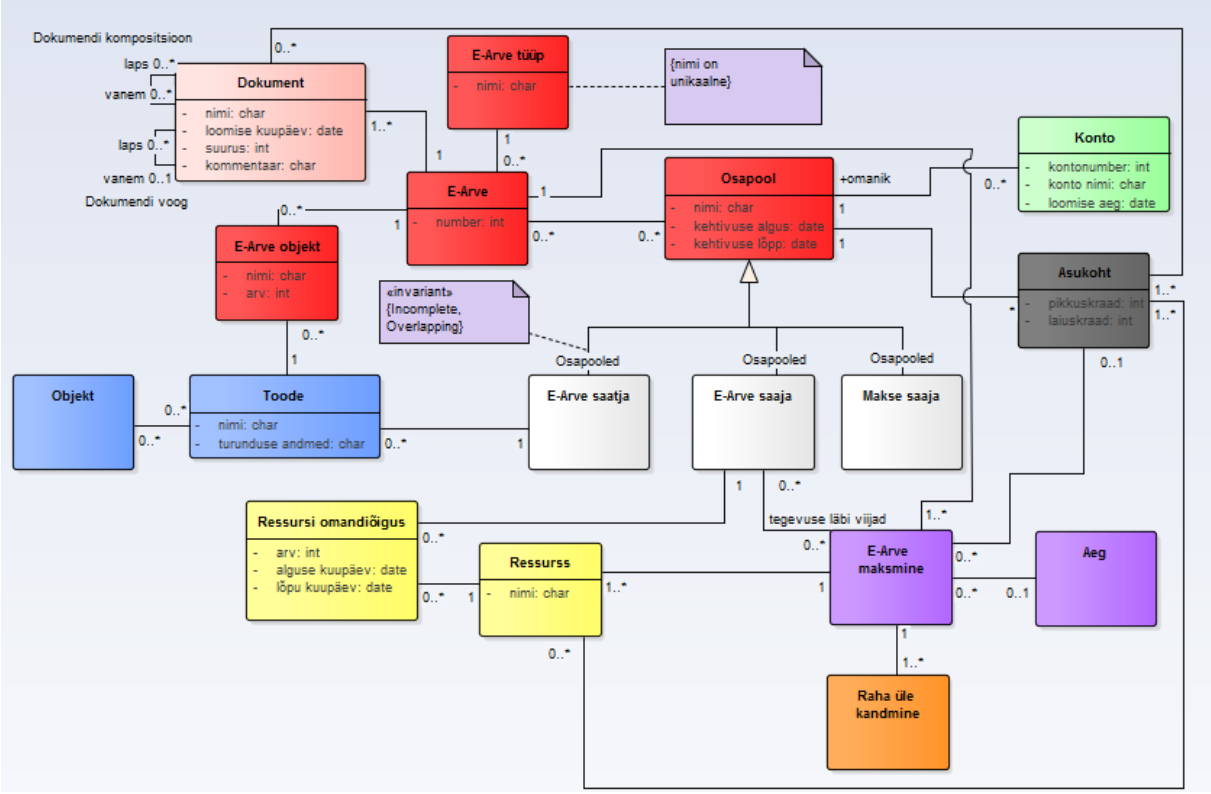
1. Tuua olemitüübile vastav muster mudelisse ning muuta selle käigus elemendi nime. Nii on toimitud olemitüübi Leping (Lepingust sai E-arve) korral.

2. Tuua olemitüübile vastav muster muutmata kujul mudelisse ning seejärel märkida see olemitüüp infosüsteemi-spetsiifiliste elementide üldistuseks. Mudelis on selliselt kirjeldatud Osapool ning selle alla kuuluvad Makse saaja, E-Arve saaja ning E-Arve saatja.

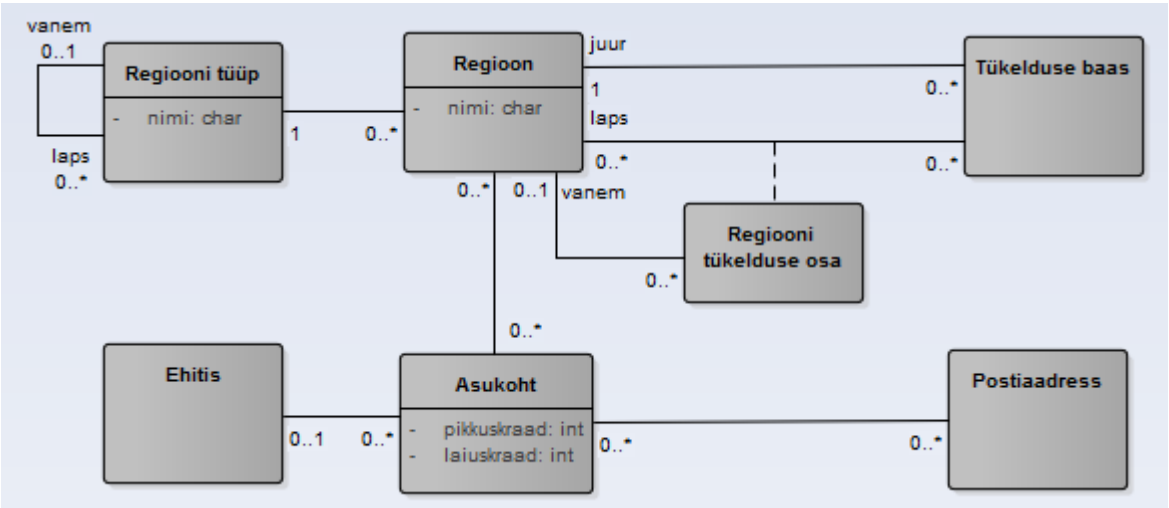
Osapooled on kirjeldatud UML kitsendustega {Incomplete, Overlapping}, mis tähendab, et osapooli võib olla ka teisi, väljaspool nimetamata tegutsejaid ning osapool võib olla samaaegselt nii näiteks arve saatja, kui makse saaja.

Kõikide osapooltega on seotud Konto ning Asukoht, kuid ainult E-Arve saaja jaoks on asjakohane ülesanne E-Arve maksmine. Ülesande teostamiseks peab E-Arve saaja kasutama ressursse, mille omanik ta on. Arve maksmise ülesanne võib olla mitmel erineval arve saajal ning maksmist võib sooritada mitmes osas.

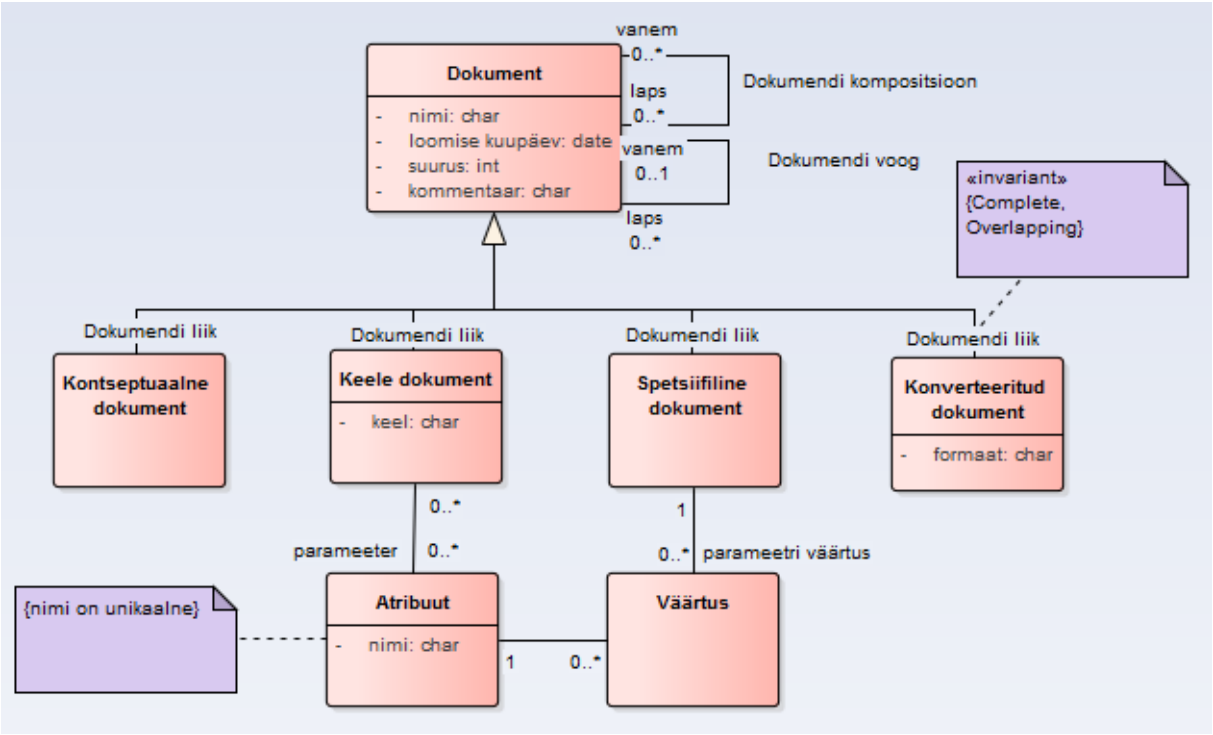
Raha üle kandmine toimub läbi E-Arve maksmise ülesande täitmise ning raha üle kandmisi võib olla mitu, ehk arve eest tasutakse mitmes osas. Raha üle kandmine võib esile kutsuda mitmeid teisi sündmusi, kuid kuna käsitletakse vaid maksmise toimingut, siis selle põhjustatud sündmused pole antud kontekstis olulised. Raha üle kandmine võib samas jaguneda alamsündmusteks nagu raha ülekanne maksjalt panka ühte panka ning raha ülekanne ühest pangast teise.



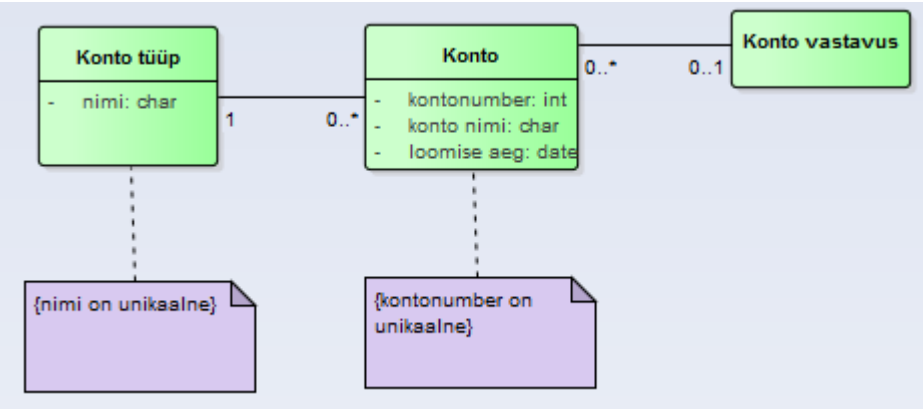
Joonis 18: E-arve maksmise allsüsteemi algne mudel



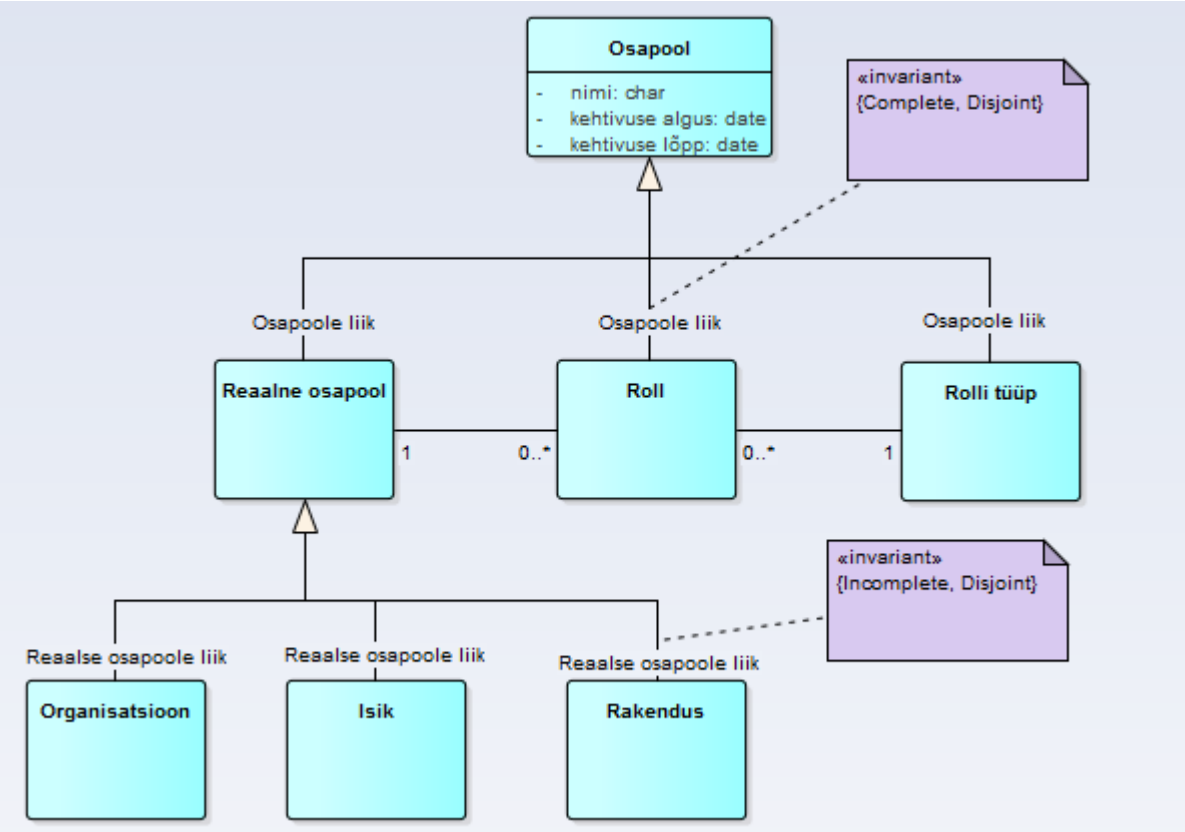
Joonis 19: Objekti Asukoht täpsustav mudel



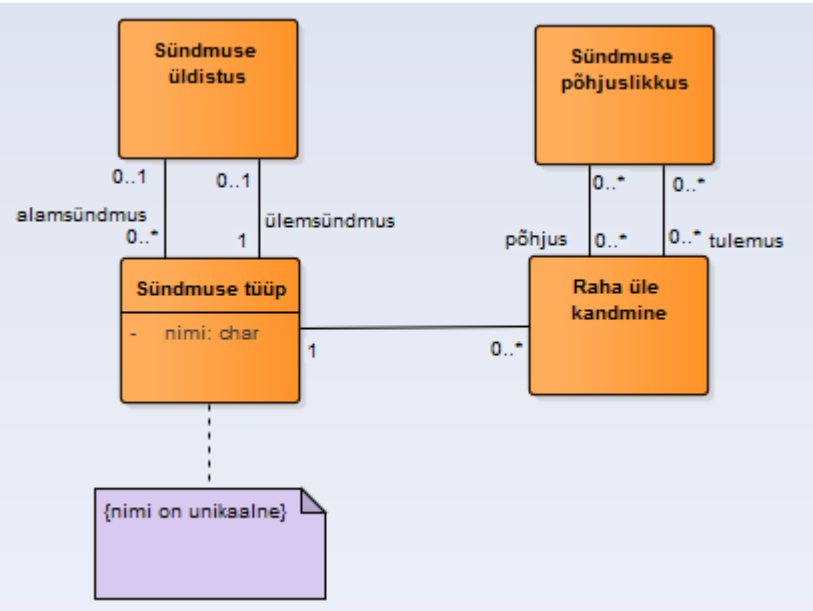
Joonis 20: Objekti Dokument täpsustav mudel



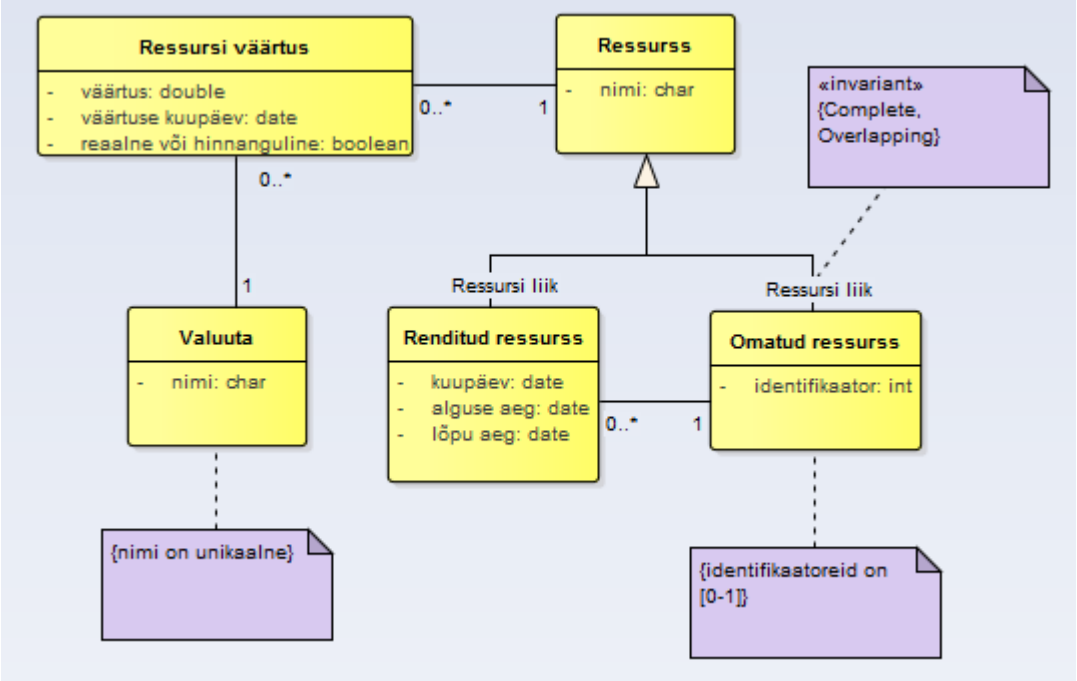
Joonis 21: Objekti Konto täpsustav mudel



Joonis 22: Objekti Osapool täpsustav mudel



Joonis 23: Objekti Raha üle kandmine täpsustav mudel

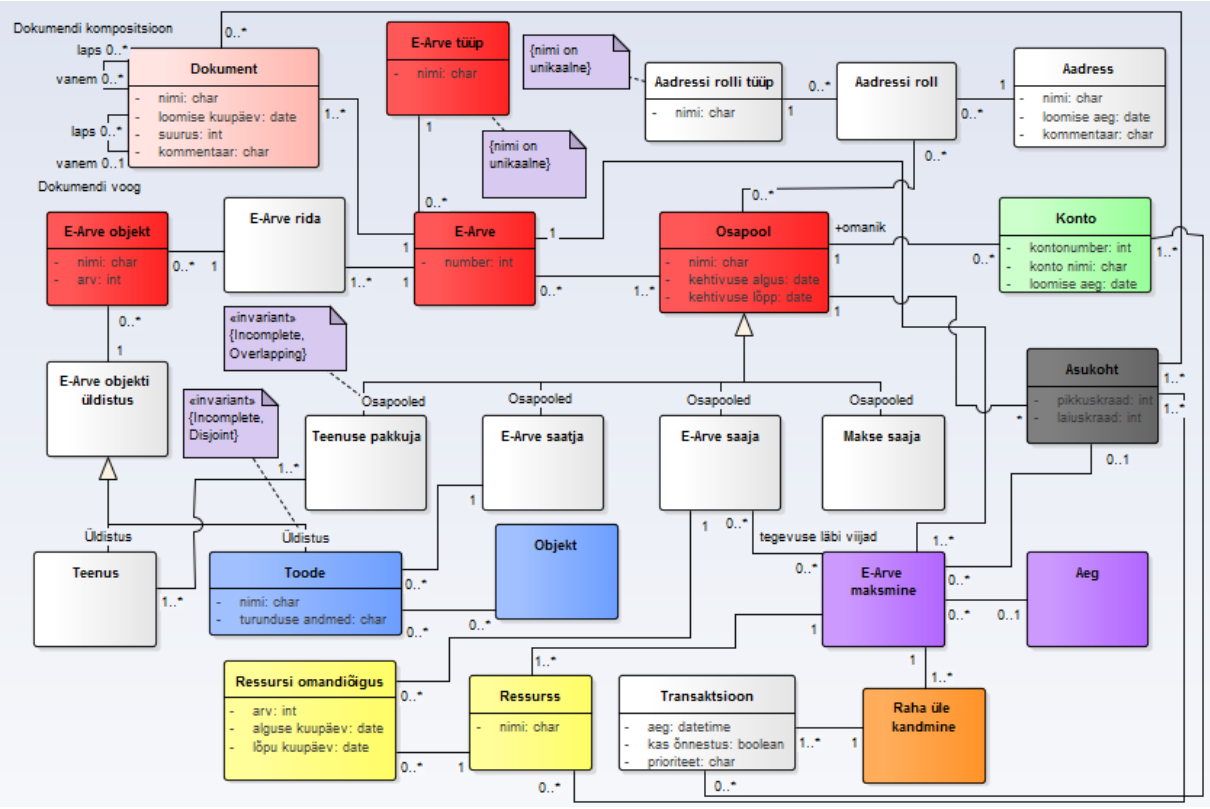


Joonis 24: Objekti Ressurss täpsustav mudel

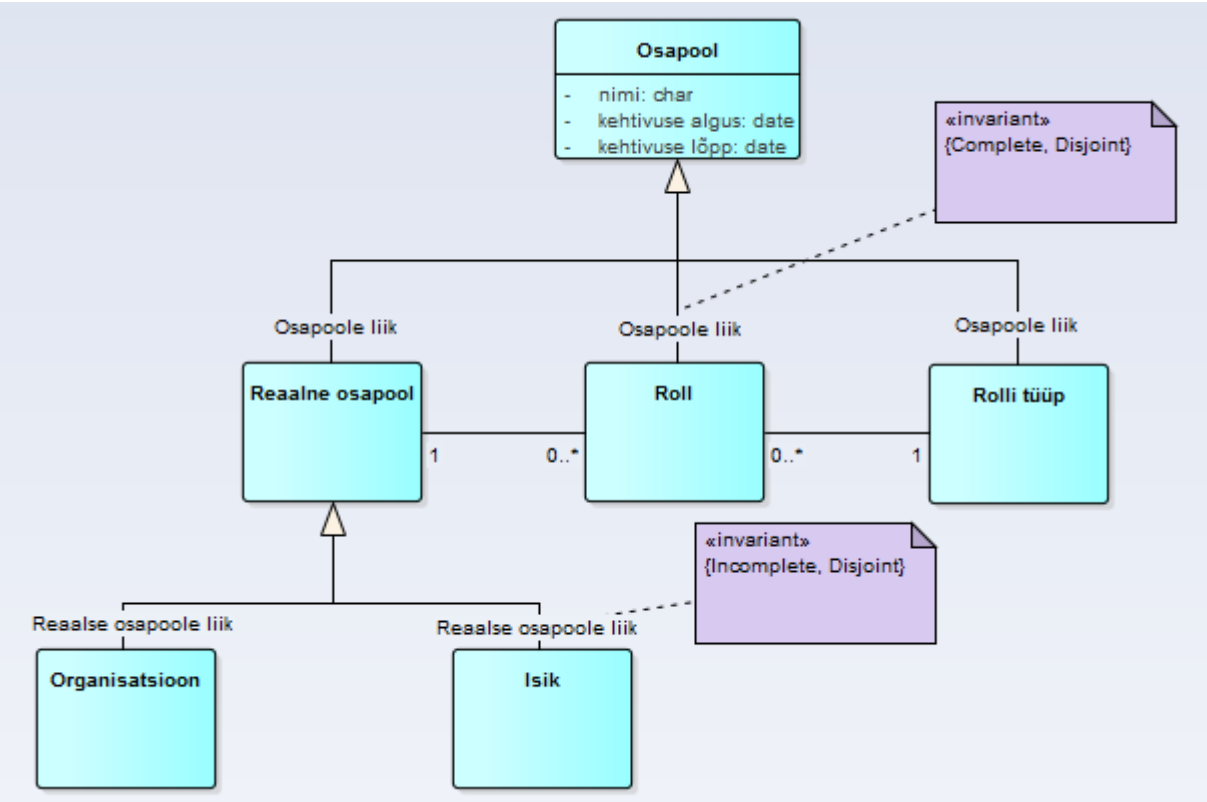
6.3 Parandatud mudel

Eelnevas punktis kirjeldatud mudel sobib selleks, et saada üldine pilt süsteemi struktuurist. Sellegi poolest puuduvad sealt mitmed olemitüübid, mis on e-arvete maksmisel olulised, esineb seoseid, mis pole päris korrektsed ning võib esineda elementide liiasusi vaatamata sellele, et arhetüüp on võimalikult lihtsustatud kujul.

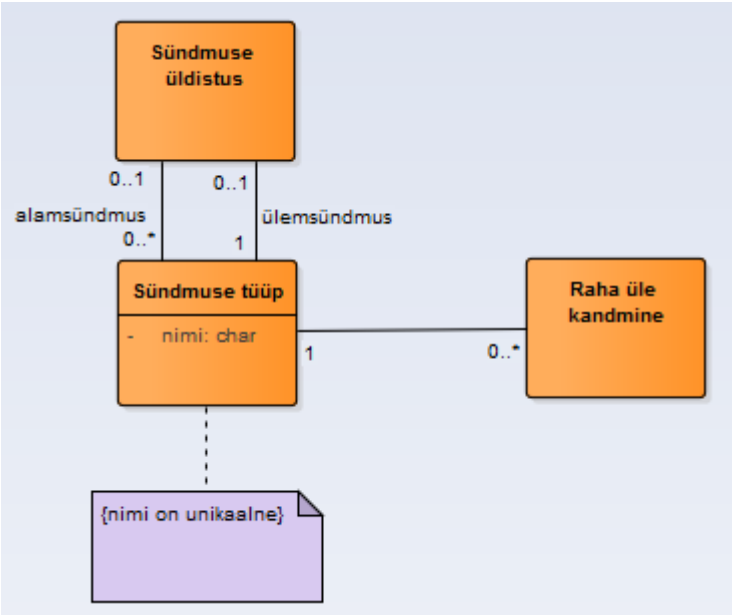
Parandatud mudeli joonised on Joonis 25 - Joonis 27. Joonised on ainult mudelitest, kus on muudatusi toimunud. Ülejäänud mudelid jäid samale kujule. Seega, muutunud on mudelid: E-arve maksmise mudel, Osapoole mudel ning Raha üle kandmise mudel.



Joonis 25: E-arve maksmise allsüsteemi parandatud mudel



Joonis 26: Objekti Osapool parandatud mudel



Joonis 27: Objekti Raha üle kandmine parandatud mudel

Osapoole mudelist on eemaldatud liigne olemitüüp *Rakendus*, mis oli Reaalse osapoole alamtüübiks põhjusega, et e-arve maksmisel ei saa osaleda rakendus, vaid maksmise otsused peavad vastu võtma konkreetselt kas organisatsioon või isik. Kuna üldistuste hulga kitsendus on „Incomplete“, siis pole teised alamklassid siiski täielikult keelatud.

Raha üle kandmise (Sündmus) mudelist on eemaldatud element *Sündmuse põhjuslikkus*. Antud süsteemis on raha ülekandmise tegevus lõplik, millele järgnevad sündmused pole olulised.

E-arve maksmise mudelis ei sobinud, et e-arvele saab vastata null või mitu osapoolt. E-arvega peab olema seotud vähemalt üks osapool ning selle tõttu on seose kordsus nüüd üks või mitu. E-arved ei ole seotud otseselt e-arvete objektidega. Vastavalt standarditele, on arvel seos objektiga läbi ühe või enama arve rea.

Lepingu mustris (vt Joonis 4) on *Lepingu objektil* seos ainult olemitüübiga *Toode*. Reaalsuses võib olla arve objektiks ka osutatav teenus. E-arve objektil ei ole seega seos otse tootega, vaid hoopis e-arve objekti üldistusega. Üldistuse alla kuuluvad *Teenus* ning *Toode*. Üks ja sama e-arve objekt ei saa olla korraga teenus ja toode. Teenuse sisse toomisel on vajalik kirjeldada osapoolena ka Teenuse Pakkuja, kuna arve saaja ei pruugi olla see, kes osutab teenust, vaid võib teiste lepingute alusel palgata teenuse pakkuja.

E-arves on alati kirjeldatud osapoolte kontaktandmed. Selleks on mudelisse sisse toodud Blaha (2010, 124) kirjeldatud arhetüüp *Address* (eesti keeles Aadress), mis kirjeldab osapoolega ühenduse võtmise meetodeid. *Aadressi rolli tüüp* kirjeldab, mis sorti kontaktandmetega on tegemist – arve saatmise aadressiga, kauba saatmise aadressiga või muu sarnasega. Aadressiandmed võivad käia nii telefoninumbri, veebiaadressi, postiaadressi kui e-maili aadressi kohta.

Konto ning *Raha üle kandmise olemitüüpidega* on seoses olemitüüp *Transaktsioon*, mis kirjeldab rahasumma kontole kandmise protsessi andmeid. See seob maksmise sündmuse kontoga ning võimaldab süsteemis jäädvustada kontoga seoses toimuvaid tehinguid.

Mudelit on võimalik veel spetsiifilisemaks muuta edasise modelleerimise käigus, kuid oluline on hetkel see, et kontseptuaalne mudel annaks ülevaate süsteemi põhiolemitüüpidest ning nende seostest.

6.4 Mudelite analüüs

Esimesena realiseeritud mudel e-arvete maksmise funktsionaalsest allsüsteemist ei erine suurel määral parandatud mudelist. Põhiolemitüübid jäid üldjoones samaks (peale *Osapool* ning *Raha üle kandmine* olemitüüpide).

Algses mudelis oli võimalik olemasolevate mustrite alusel kirjeldada süsteemi põhistruktuuri ning tekkis hea ülevaade e-arvete maksmise valdkonna olemitüüpide struktuurist. Enne mudeli loomist integratsiooni mustri põhjal arhetüüpide valimine annab hea ülevaate vaja minevatest mustritest ning aitab mudeli välja töötamisel luua seoseid mudeli osade vahel. Kui on põhiolemitüübid mustri põhjal mudelisse tekitatud, siis edasi saab aluseks võtta süsteemi spetsiifika ning lisada olemitüüpe, mida kõikidel süsteemidel ei esine.

Selles töös realiseeritud mustrite korral tuleks paremate mudelite loomise huvides täiendada *Konto* ja *Asukoha* arhetüüpe, lisades nendele vastavalt juurde *Transaktsiooni* ja *Aadressi* arhetüübid.

Kuna arhetüübid on kirjeldatud väga suurelt üldistades, siis algne mudel on juba suhteliselt lihtsustatud kujul. Samas liigne lihtsustus osutus ka probleemseks Sündmuse mustri (vt Joonis 7) korral, kus ei ole kirjeldatud seost, mis võib esineda mitme konkreetse sündmuse vahel, vaid pigem abstraktset olemitüüpi *Sündmuse põhjuslikkus*.

Sama mudeli loomine ilma mustrite kasutamiseta võtaks rohkem aega ning tuleks taasleiutada mitmed kontseptuaalsed struktuurid, mis tegelikult on kasutustes väga paljudes süsteemides. Mustrite kasutamine suurendab arendamise kiirust ja parandab tulemuse kvaliteeti.

7. Kokkuvõte

Töö eesmärgiks oli realiseerida Enterprise Architect (EA) CASE vahendis üheksa taaskasutatavat komponentmustrit ning nende integratsiooni muster (tuntud ka kui mustrite muster), mis lihtsustaksid edaspidiselt suvalise valdkonna kontseptuaalsete mudelite loomist. Töö ülesandeks oli ka kasutada loodud mustreid vabalt valitud valdkonna näitesüsteemi kontseptuaalse andmemudeli loomisel, et näidata nende mustrite kasutamise võimalikkust ja kasulikkust.

Autori hinnangul eesmärk täideti. Kuna mustritest esineb palju variatsioone, siis valiti realiseerimiseks Blaha (2010) arhetüüpe ja paaril korral ka Fowleri (1996) välja pakutud analüüsimustrid. Kõigepealt kirjeldati töös need mustrid struktureeritud formaati kasutades ning sealjuures analüüsiti nende puuduseid. Seejärel realiseeriti mustrid UML klassidiagrammidena ning tehti võimalikuks nende taaskasutamine EA vahendis. Selleks kasutati EA-sse sisseehitatud mustrite loomise mehhanismi. Loodud tulemid on kättesaadavad avalikult veebilehelt: http://apex.ttu.ee/archetypes Näitesüsteemi valdkonnaks valiti e-arvete infosüsteemi maksete allsüsteem ja sellega demonstreeriti loodud mustrite kasutamist. Töös kirjeldati ka EA vahendis mustrite loomise protsessi, et huvilistel saaksid täiendavaid mustreid luua.

Järgnevalt on loetletud töö sooritamisest tulenevad järeldused:

- Suhteliselt väikese hulga analüüsimustrite abil on võimalik luua süsteemi kontseptuaalse mudeli esialgne versioon.
- Süsteemi põhiobjektide ehk põhiolemitüüpide valikuks on efektiivne kasutada integratsiooni mustrit.
- Realiseeritud mustrid ei ole täiesti piisavad infosüsteemi täielikuks kirjeldamiseks.

- Enterprise Architect vahendis on mitmeid vigu, mis võivad teha diagrammide loomise keeruliseks, kuid neid puudujääke arvesse võttes saab selle töövahendiga mugavalt luua mudeleid ning mustreid.

- Viies mustrid üle spetsiifilisse konteksti, ei pruugi need alati olla piisava täpsusega kirjeldatud, vaid edasiste arendussammude käigus tuleb mustrite alusel loodud mudelit täpsustada.

Käesolevat tööd saab edasi arendada, realiseerides CASE vahendis kasutamiseks täiendavaid mustreid. Näiteks võib tuua e-arvete maksmise mudelis ilmnenud *Transaktsiooni* ning *Aadressi* mustrite vajaduse. Vajalik oleks uurida, milline täpselt on minimaalne, vajalik ning piisav komponentmustrite hulk, selleks et erinevates valdkondades opereerivaid süsteeme modelleerida. Kindlasti oleks vaja erinevate autorite kirjeldatud komponentmustreid ning neid variatsioone lähemalt analüüsida, leidmaks nende puudujääke ja teha neid vajadusel paremaks. Arhetüüpide põhise modelleerimise toetuseks tuleb valida ainult kõige paremad variatsioonid.

Summary

The goal of the work was to implement nine reusable component patterns and their integration pattern (known also as "pattern of patterns") in Enterprise Architect EA CASE tool to simplify conceptual modeling in any domain. One of the tasks in this work was to use the created patterns in the creation of a conceptual data model of a system to demonstrate the possibility and usefulness of using said patterns.

The author thinks that the goal was fulfilled. Because there are many variations of patterns, archetypes offered by Blaha (2010) and sometimes also analysis patterns offered by Fowler (1996) were selected for the implementation. First of all, the patterns were described in a structured manner and the problems of the patterns were shortly analyzed. Next, the patterns were implemented as UML class diagrams and the arrangements were made to reuse these in the EA CASE tool. The pattern-creation mechanism that is built in to the EA was used for that purpose. The produced artifacts are available at the public web page: http://apex.ttu.ee/archetypes The example system that was used to illustrate the use of the patterns was the payment subsystem of electronic invoices. The work also described the process of creating patterns so that interested parties could create additional patterns.

The conclusions of the thesis are as follows:

- It is possible to create the initial version of a system's conceptual model with the use of a relatively small number of analysis patterns.
- An effective way of choosing a system's main objects or entity types is using an integration pattern.
- The realised patterns are not sufficent to describe an information system fully.
- The Enterprise Architect tool has several errors, that can render creating diagrams difficult, however when these problems are accounted for, then EA is a convenient tool for creating models and patterns.

- When patterns are put into a specific context, they might not be sufficently described. In that case the constructed model must be specified in subsequent steps of development.

It is possible to further expand this work by creating additional patterns in the CASE tool. For instance the necessity for patterns of Transaction and Address, as seen from the payment subsystem of electronic invoices. It would be essential to explore, what is the exact minimal and sufficent amount of component patterns to model systems from various domains. Analysing component patterns and their variations from different authors is a definite necessity to find the patterns' shortcomings and to improve them, if necessary. Only the best variations must be chosen to support archetype-based modeling.

Kasutatud kirjandus

1. Blaha, M. (2010). Patterns on Data Modeling. Boca Raton: CRC Press.

2. Doble J., Meszaros G. A Pattern Language for Pattern Writing. - *The Hillside Group.* [WWW] http://hillside.net/a-pattern-language-for-pattern-writing (05.01.2016)

3. Eessaar, E. (2015). Andmebaasid I/II õppematerjalid.

4. Eesti keele seletav sõnaraamat. (2009). Eesti Keele Sihtasutus.Tallinn. [WWW] http://www.keeleveeb.ee/ (05.01.2016)

5. Fernandez, E. B. (1998). Building Systems Using Analysis Patterns. - *Proceedings of the 3rd International Software ArchitectureWorkshop (ISAW3)* : November 1998, Orlando, FL.

6. Fowler, M. (1996). Analysis Patterns – Reusable Object Models. Menlo Park, Calif. : Addison-Wesley.

7. Geyer-Schulz A., Hahsler M. (2001). Software Engineering with Analysis Patterns. - *Working Papers on Information Systems, Information Business and Operations, 01/2001.* [WWW] http://epub.wu.ac.at/id/eprint/592 (05.01.2016).

8. Giles, J. (2012). The Nimble Elephant: Agile Delivery of Data Models Using a Pattern-Based Approach. Westfield, NJ : Technics Publications, LLC.

9. Piho, G. (2011). *Archetypes Based Techniques for Development of Domains, Requirements and Software. Towards LIMS Software Factory.* Doktoritöö TTÜ Informaatikainstituut.

10. Silverston, L., Agnew, P. (2009). The Data Model Resource Book.Vol. 3,Universal Patterns for Data Modeling. Indianapolis, IN: Wiley Publishing, Inc.

11. Unified Modeling Language – Wikipedia. [WWW] https://en.wikipedia.org/wiki/Unified_Modeling_Language (05.01.2016)

12. Vallaste, H. (2000). e-Teatmik: IT ja sidetehnika seletav sõnaraamat. [WWW] http://www.vallaste.ee (05.01.2016)