Tallinna Pedagoogikaülikool
Matemaatika-loodusteaduskond
Informaatika osakond

Liina Lang

UML-i õppematerjal

Bakalaureusetöö

Juhendaja: Juhan Ernits

Autor:....."...."....2003
Juhendaja:....."...."....2003
Osakonna juhataja:....."...."....2003

Tallinn 2003

Sisukord

Sissejuhatus

Objekt-orienteeritud programmeerimise areng tekitas vajaduse objekt-orienteeritud projekteerimismeetodite järele. Aastate jooksul tekkis erinevaid meetodeid. See raskendas projekteerijate omavahelist suhtlust. Samuti oli nende meetodite kasutajatel keerukas leida modelleerimiskeelt, mis oleks korraga rahuldanud kõiki nende vajadusi. Et olukorda parandada, otsustasid kolme teiste seast esile tõusnud meetodi loojad G. Booch, I. Jacobson ja J. Rumbaugh luua ühtse mudelikeele ehk *Unified Modeling Language (UML)*. Peamiselt liidabki UML neid kolme meetodit: *Booch*[1]*, Object-Oriented Software Engineering (OOSE)*[2] ning *Object Modeling Technique (OMT)*[3], kuid on oma olemuselt laiem. UMLi versiooni 1.0 standardiseeris *Object Management Group (OMG)* 1997. aastal [OMG][4].

OMG on rohkem kui 800 tarkvara ja tarkvaratehnika tööriistu väljatöötava, tootva, ja/või kasutava ettevõtte konsortsium, mis koordineerib objekt-orienteeritud tarkvaratehnika arengut maailmas, standardiseerib toodete spetsifikatsioone ja nende liideseid. [OMG]

Kuna UML on maailmas üha enam leviv standardiseeritud viis süsteemide modelleerimiseks, siis muutub UML-i kasutuselevõtt Eesti tarkvarafirmades aina aktuaalsemaks. Kahjuks on teemakohast eesti keelset vabalt kättesaadavat materjali siiski veel vähe. Koolitusfirmade poolt pakutavad kursusematerjalid on aga kallid.[5] [6] Ülikoolides õpetatavate ainete raames aga on loengumaterjalid sageli kas ainult konspektiivsed ning seega vastavat loengut mittekuulavale huvilisele raskesti arusaadavad, või parooliga kaitstud ja seega kättesaadavad vaid väikesele sihtgrupile.

---

[1] **Booch, G.** (1994). Object-oriented Analysis and Design With Applications. 2nd ed. The Benjamin/Cummings Publishing Company, Inc.
[2] **Jacobson, I.** (1992) Object-oriented software engineering : a use case driven approach**.** Addison-Wesley Publishing Company, Inc.
[3] **Rumbaugh, J., Blaha M., Premerlani W., etc.** (1991). Object-oriented modeling and design. Prentice-Hall International.
[4] **About the Object Management Group** (OMG)s
http://www.omg.org/gettingstarted/gettingstartedindex.htm
[5] **BCS Koolitus**
URL: http://koolitus.bcs.ee/kursused/moodul3.htm, 17.05.2003

[6] **Software Engineering Center OÜ**
URL: http://www.sec.ee/seminars.asp?lg=ee&seminarID=50, 17.05.2003

Samuti käsitletakse sageli UML-i loengukursuste raames vaid ühe osana, mitte ei pühendata sellele kogu loengukursuse aega. Seega on käesoleva bakalaureusetöö eesmärgiks luua õppematerjal, mis on elektroonilisel kujul kättesaadav vabalt ning mõeldud ka laiemale sihtgrupile sh. TPÜ ja ka teiste ülikoolide informaatikatudengitele ning tarkvaraarendajatele. Käesolev bakalaureusetöö on proseminaritöö "Ülevaade modelleerimiskeelest UML" edasiarendus.

Töö on oma ülesehituselt jaotatud neljaks osaks: uuring UML-i kasutamisest Eesti tarkvarafirmades, UML-i õppematerjal, UML-i tööriistad ning UML-i arengusuunad.

Üheks töös püstitatud ülesandeks on kirjeldada peamisi UML-i modelleerimistehnikaid, mis hõlmavad endas nii keele tähistusi kui ka semantikat. Kuna töö maht seda ei võimalda, ei ole püütud ühtlaselt katta kõiki modelleerimistehnikaid, vaid peatuda pikemalt praktikas oluliseks osutunud skeemidel. Viie modelleerimistehnika kirjelduse juurde on lisatud ka harjutusülesanded, mille hulgas on nii lihtsamaid valikvastustega küsimusi kui ka keerukamaid. Samuti on klassiskeemide ja jadaskeemide juurde lisatud koodinäited.

Ühtekokku on kirjeldatud üheksat modelleerimistehnikat: koostööskeeme, klassiskeeme, paketiskeeme, jadaskeeme, koostööskeeme, olekuskeeme, tegevusskeeme, komponentskeeme ja levitusskeeme. Siinkohal on valiku tegemisel lähtutud töö raames kaheksas Eesti tarkvarafirmas läbi viidud uurimusest.(vt peatükk 2). Kuigi UML ei ole protsessist sõltuv, omandavad modelleerimistehnikad siiski selgema mõtte protsessi kontekstis, vastavalt sellele, missuguses objekt-orienteeritud arenduse faasis ollakse. Seega on käesolevas töös kõiki modelleerimistehnikaid kirjeldatud protsessi raames ning protsessiks, millest lähtutakse on RUP-protsess ehk *Rational Unified Process (RUP)*. [RUP]

Uurimus on läbi viidud süvaintervjuudena ning eesmärgiks polnud teha statistilisi üldistusi, vaid leida kinnitust raamatust loetule ning uurida UML-iga seonduvaid kogemusi ja probleeme. Uurimus koosneb kuuest osast: intervjueeritava kogemused

UML-i ja/või objekt-orienteeritud meetodite kasutamisel, UML komponentide kasulikkus, UML tööriistad, analüütiku-kliendi suhtlus, organisatsiooni andmed ning intervjueeritava IT taust .

UML-i tööriistade valik on lai ning nii ühel kui teisel on omad plussid kui miinused. Et hõlbustada vastavalt vajadustele ja võimalustele tööriista valikut, on töös antud ülevaade, millist funktsionaalsust üks või teine tööriist pakub. Kuna primaarseks ei ole siinkohal ära kirjeldada kõiki tööriistu ning kõiki pakutavaid võimalusi, vaid pigem anda selles osas ülevaade, on paljude tööriistade seast tehtud valik ning omaduste võrdluseks tabelisse koondatud. Valiku tegemisel on lähtutud nii sellest, milliseid tööriistu eelistavad kasutada intervjueeritud kui ka sellest, millistega on käesoleva töö autor ise kokku puutunud. Käesolevas töös on skeemide joonistamisel kasutatud Rose Enterprise Edition (Rational) jaEnterprise Architect 3.5 (Sparx Systems) prooviversioone ning vabavaralist tarkvara PoseidonUML 1.6 Community Edition (Gentelware).

Tarkvaramaailm on pidevalt muutuv ja innovatiivne valdkond ning edukad on eelkõige need, kes suudavad muutustest kasu lõigata ja uuendusi oma töös rakendada. Seetõttu on käesolevasse töösse lisatud peatükk UML-i arengusuundadest. Nimetatud peatükk peaks andma ideid ning tekitama huvi ja motivatsiooni UML-iga sügavuti tutvumiseks. UML-i arengusuundades on esmalt kirjeldatud muudatusi ja parandusi, mida on tehtud peatselt ilmuvas UML-i versioonis 2.0 ning ka UML-i tähtsat rolli mudeljuhitud arhitektuuris ehk *Model Driven Architecture-s (MDA)*, mis on OMG uus tarkvara ning süsteemide arendamise metoodika.

Käesolevas töös on püütud leida inglise keelsetele sõnadele sobivaim eesti keelne vaste. Peamiselt on tõlkimisel aluseks võetud Littover, M., UML-keele sõnastik, kuid on kasutatud ka KeeleWeb-i Arvutikasutaja sõnastikku. [7] [8]

[7] **Littover, M.** UML-keele sõnastik.
URL: http://www.cc.ioc.ee/uml

[8] **Arvutikasutaja sõnastik**. KeeleWeb
URL: http://ee.www.ee/AKS/

Tekstis on kaldkirjas ära toodud tähtsamate või mitmeti tõlgitud mõistete kõrvale ka nende inglise keelsed vasted.

Objekt-orienteeritud mõttemaailm ja UML

Objekt-orienteeritud (OO) modelleerimine, analüüs ja disain on tarkvara- ja süsteemiarenduses tuntud paradigma, millest on välja kasvanud komponenttehnoloogia ja ühtne modelleerimiskeel UML .

OO lähenemine probleemidele ja programmeerimisele on erinev struktuursest lähenemisest. Kui struktuurses lähenemises, näiteks programmeerimiskeelte puhul, on pearõhk assembleri abstraheerimisel (assembler omakorda on masinkoodi abstraktsioon), siis OO lähenemise korral astutakse samm edasi ning üritatakse programmi kirja panna ümbritseva maailma struktuuriga, st objektidena, mida iseloomustavad olek, käitumine ja identiteet. [JAVA] Objekt-orienteeritud programmeerimise areng tekitas vajaduse objekt-orienteeritud projekteerimismeetodite järele. Aastate jooksul tekkis mitmeid erinevaid meetodeid. See raskendas projekteerijate ning arendajate omavahelist suhtlust. Samuti oli nende meetodite kasutajatel keerukas leida modelleerimiskeelt, mis oleks korraga suutnud kajastada kõiki arendatava süsteemi aspekte. [BOOCH, lk.38-39]

UML on modelleerimiskeel, mitte meetod. Meetodid koosnevad nii modelleerimiskeelest kui protsessist. Modelleerimiskeel on (peamiselt graafiline) tähistusvahend. Protsess määrab tegevuste järjekorra arendusprotsessis.

UML loomise eesmärgid on: modelleerida süsteeme objekt-orienteeritud (OO) tehnikaid kasutades alates nende mõistetest kuni rakenduslike tehisteni välja; võimaldata tööd ka suuremahuliste ja keeruliste süsteemidega; luua modelleerimiskeel, mida suudaksid lugeda nii inimesed kui masinad. UML kasutusalaks on kõikvõimalikud süsteemid: infosüsteemid, tehnilised süsteemid, reaalajasüsteemid, hajussüsteemid, ärisüsteemid jne. UML on keel süsteemi visualiseerimiseks, spetsifitseerimiseks, väljatöötamiseks ja dokumenteerimiseks.

1 Uuring UML-i kasutamisest Eesti tarkvarafirmades

1.1 Sissejuhatus

Käesoleva uuringu eesmärgiks ja põhilisteks uurimisprobleemideks olid:

- Uurida praktilisi kogemusi UML-i kasutamisel.
- Uurida probleeme, mis seonduvad UML-i kasutamisega.
- Saada ettekujutus, mida peetakse praktikas UML-s rohkem ning mida vähem oluliseks ja miks.
- Uurida motiive, miks UML-i kasutatakse.

Uuring aitab kohandada õppematerjali sihtgrupile (kohalike IT ettevõtete töötajad ning tulevikus neisse tööle asuvad tudengid) IT ettevõtete vajadusi ja kogemusi arvesse võttes, seades põhirõhu UML-i elujõulisematele ning praktikas kasutust leidnud osadele.

Uurimus on läbi viidud struktureeritud süvaintervjuudena ja uurimuse eesmärgiks ei ole statistiliste üldistuste tegemine, vaid reaalsetes projektides UML-i kasutamisega seotud kogemustest ning probleemidest ülevaate saamine. Kuigi enamikele küsimustest olid vastusevariandid välja pakutud, on vastuseankeetidesse lisatud ka variantidega hõlmamata asjassepuutuvad kommentaarid, mida küsitletavad andsid. Eesmärgiks polnud ainult fikseerida teatud tüüpi praktikaid, vaid leida ka nende taga peituvaid motiive.

Kuna käesoleva töö maht seda ei võimalda, on uuringu analüüsitulemused võimalikult lühidalt esitatud ning vaid enimoluline välja toodud.

1.2 Uurimismetoodika ja andmetöötluse kirjeldus

1.2.1 Valimi moodustamine ja kirjeldus

Käesolev uurimus viidi läbi 2003. aasta kevadel kaheksas Eestis tegutsevas tarkvara oma tarbeks või müügiks tootvas ettevõttes ning asutuses. Küsitletavateks olid peamiselt projektijuhid või analüütikud, kuna nende otsustel on kogu projektimeeskonna liikmetele

kaalukam mõju ning seega võib nende arvamust pidada antud ettevõtte kontekstis oluliseks. Valisin välja erialaspetsialistidega eelnevalt peetud vestlustele tuginedes kaheksa sellist ettevõtet ja asutust, kus UML on kasutusel.

1.2.2 Meetodi valik ja andmetöötluse kirjeldus

Meetodina on antud uurimistöös kasutatud struktureeritud süvaintervjuud keskmise kestusega ca 1 h. Küsitluse aluseks olid ankeedid, kus vastused osaliselt valikuvariantidena ette antud. Küsimustiku koostamisel on tuginetud bakalaureusetöö õppematerjali aluseks olnud proseminaritööle ja OMG kodulehel asuvale küsimustikule.[9] Küsimustik koosneb kuuest osast ja sisaldab 29 küsimust. Küsimustiku kuus osa on järgmised:

- intervjueeritava kogemused UML-i ja/või objekt-orienteeritud meetodite kasutamisel; UML komponentide kasulikkus;
- UML tööriistad;
- analüütiku-kliendi suhtlus;
- organisatsiooni andmed;
- intervjueeritava IT taust.

Küsimuste hilisemaks põhjalikumaks töötlemiseks ning analüüsiks on kasutatud diktofoni.

1.3 Uurimistulemuste kokkuvõte ja analüüs

1.3.1 Intervjueeritava kogemused UML-i ja/või objekt-orienteeritud meetodite kasutamisel.

UML-i kasutamine

Nagu valimi koostamisel eeldatud, on kõik küsitletud UML-i kasutanud, kuid vastanute kogemused erinevad üsna suurel määral. Mõned vastanutest kasutavad UML-i kõigis projektides, mõned peavad vajalikuks kasutada vaid teatud profiiliga, näiteks suuremate

[9] Unified Modeling Language Usage Survey
URL: http://fusion.uleth.ca/crdc/uml_survey/

ning keerukamate projektide juures. Enamus vastanutest kasutab projektides läbivalt ühte tähistuskeelt - UML-i. Üks küsitletutest kasutas andmebaaside modelleerimisel lisaks ka relatsioonilisi olem-seos ehk *Entity-relationship (ER diagram)* mudeleid. Modelleerimise maht enne programmeerima asumist on eelkõige sõltuvuses konkreetsest projektist ja selle keerukusest. Hetkel käsil olevatest projektidest kasutasid UML-i üle 50% vastanutest.

UML-i projektis mittekasutamise põhjused

Kui UML-i ei peeta vajalikuks projekti juures kasutada, siis toodi välja järgmised põhjused (järjestatult esinemise sageduse järgi):

1. *Organisatsioonil ei ole sobivaid projekte UML või objekt-orienteeritud süsteemiarenduseks*. Siinkohal toodi omakorda projektide mittesobivuse põhjustena välja, et projektid on liiga väikesed või liiga väikese eelarvega.
2. Mõnikord *ollakse sõltuvuses mingi väga spetsiifilise ala allhankijast (väljastpoolt sisse ostetavate tööde puhul), kes ei kasuta standardset keelt, vaid oma väljaarendatud tähistust*. Leiti, et sellisel juhul on lihtsam see spetsiifiline lõik ära õppida, kuna UML-is tuleks nagunii spetsiifikast tulenevalt umbes pool laiendustena juurde kirjutada
3. *Organisatsioonis on liiga vähe inimesi, kes on tuttavad UML-iga või objekt-orienteeritud meetoditega.*
4. *UML on kasutamiseks liiga keerukas.* Kasutamise keerukuse all ei peetud siinkohal silmas tähistust, vaid pigem tööriistu.

Missuguste süsteemide loomisel on UML-i kasutatud

Kõik vastanutest on UML-i kasutanud eelkõige uue süsteemi loomisel. Nimetati ka projekte, kus on olnud tegu vana süsteemi täieliku väljavahetamisega. Olemasoleva süsteemi laiendamist mainiti kõige vähem kordi.

Süsteemide valdkonnad, mille loomisel UML-i kõige sagedamini kasutati, olid:

- e-kaubandus/veeb;
- administratiivne valdkond;

- sardsüsteemid.

Administratiivse valdkonna alla klassifitseeruvad näiteks tulemüüri tarkvara, sideturbetooted, intraneti haldus, virtuaalne privaatvõrk, ekstranet jmt. Sardsüsteemide all on mõeldud arvutisüsteemi, mis on muu süsteemi lahutamatu funktsionaalne osa ja sellesse valdkonda klassifitseerusid näiteks sõjatehnoloogia, sportautode infosüsteemid, kosmosetehnoloogia, finantsarvestus, kindlustus, maksuarvestus, tööstusautomaatika, transiidisüsteem jmt. Mainiti kaks korda ka mobiilset kaubandust ning ühel korral kliendisuhete haldust ehk *Customer Relationship Management (CRM)*.

UML-i mõju projektidele

Ligikaudu 90% vastanutest leidsid, et UML-i kasutuselevõtt on projektidele mõju avaldanud. Erinevad vastajad tõstsid esile erinevaid mõjutegureid või hindasid mõjutegureid erineva olulisuse astmega, kuid üles loetleti kõik pakutud vastusevariandid. UML-i kasutuselevõtt on:

- lihtsustanud kliendi vajaduste mõistmist;
- tõstnud loodava süsteemi kvaliteeti;
- aidanud kaasa rakenduse õigeaegsele valmimisele;
- võimaldanud kokku hoida kulusid ning aega.

Lisati, et UML-i kasutamine muudab meeskonnasisese suhtluse tänu oma standardiseeritusele efektiivsemaks, seega toimub aja kokkuhoid, mis omakorda võimaldab tõsta süsteemi kvaliteeti ning aitab kaasa rakenduse õigeaegsele valmimisele ja kulude kokkuhoiule. Kaudselt on kõik need mõjutegurid omavahel seoses. Huvitav erinevus on aja ja kulutuste kokkuhoiu hinnangutes. Kui peamiselt arvati, et aja ja kulu kokkuhoid toimub süsteemi loomise käigus lihtsustunud kommunikatsiooni arvelt, siis leidus ka selliseid arvamusi, et UML-i kasutamine pigem aeglustab süsteemi loomist, kuid aitab aega ja kulusid säästa süsteemi käigushoidmisel. Selline erinevus võib olla tingitud sellest, millisel tasemel UML-is modelleerimist projekti juures rakendatakse ja milline on süsteemi enda spetsiifika. Kui mudel on vaid süsteemist parema visuaalse ülevaate saamiseks, kulub aega süsteemi loomise käigus vähem, kuid kui mudel on ka edaspidiseks kasutamiseks mõeldud ja täpsem, tekib efekt, kus mudeli loomisele kulub esialgu rohkem aega ning suuremat tulu toob see endaga kaasa hiljem.

Mitmed vastanutest tõid rääkides UML-i mõjust projektidele sisse protsessi konteksti. UML on modelleerimiskeel ja seega ei ole protsessist otseselt sõltuv, kuid modelleerimistehnikad muutuvad paremini mõistetavaks, kui need asetada protsessi konteksti. Enamikes ettevõtetes, kus vastanud töötasid, oli majasiseselt ametlikult kinnitatud arendusprotsess. Konkreetselt mainiti ära:

- RUP ehk Rational Unified Process;
- Väikeses ettevõttes kasutamiseks peeti aga paremaks lahenduseks lähtuda ekstreemprogrammeerimise ehk *Extreme Programming (XP)* põhimõtetest.

UML-i kasutamine ettevõttesiseselt polnud kohustuslik, vaid pigem vastanute isiklik initsiatiiv.

Üks vastanutest peatus pikemalt ettevõttes kasutatava tarkvaraprotsessi arendamise ja hindamise küpsusmudelil *Software Process Improvement and Capability dEtermination (SPICE)* [10], mis on ISO/IEC poolt välja töötatud mudel. SPICE küpsusmudelit kasutab vastaja nii oma asutuse kui ka partnerfirmade hindamisel. SPICE-i eeliseks alternatiivse *Capability Maturity Model-i (CMM)* [11] ees pidas vastanu SPICE paremat sobivust Eesti oludesse. Vastaja hindas enamiku Eesti tarkvarafirmadest SPICE-i skaalal kõige madalama ehk 0-taseme väärilisek ning taseme võrra kõrgmeale paigutas panganduse sektori. Küsitletu nägi hinnanguskaalal tõusmise potentsiaali ka riigiettevõtete sektoris, juhul kui muutub efektiivsemaks riigiasutuste koostöö. SPICE-i kontekstis võimaldab UML tõsta protsesside kvaliteeti, kuna standardse keelena kiirendab ja hõlbustab dokumenteerimist ning meeskonnasisest, tarkvaratootja ning tellija vahelist kommunikatsiooni.

[10] Software Process Improvement and Capability dEtermination Website
URL: http://www.sqi.gu.edu.au/spice/

[11] Capability Maturity Model for Software (SW-CMM) (2003). Carnegie Mellon University
URL: http://www.sei.cmu.edu/cmm/

Model Driven Architecture

Suurem osa vastanutest on tuttav MDA-ga, kuid pigem jäi teadmine pinnapealseks. Vaid kaks intervjueeritavat on MDA põhimõtetega väga hästi kursis ning üks neist on ka väljaspool Eestit vastavasisulisel konverentsil osalenud.

MDA-d on projektis rakendanud vaid üks vastanutest. MDA-d projektides kasutusele võtmise takistuseks peeti juhtkonna vähest huvi, kuna esialgu kulub aega rohkem ja kulusid tuleb juurde. Samuti oli argumendiks, et hetkel ei ole veel välja töötatud tööriistu, mis suudaksid MDA-le piisavat tuge pakkuda. MDA kasutuselevõtu kohta arvati, et kuna tungivat vajadust muudatuste järele ei ole, siis lähima kolme aasta jooksul MDA-d kasutusse ei võeta. Samuti leiti, et MDA-d on mõttekam kasutada just suuremates või rahvusvahelistes ettevõtetes.

Kuna reaalsetes projektides MDA kasutamise kogemused kas puuduvad või on väga vähesed, jäi ka näidete baas selles osas kitsaks. MDA-d rakendanud projektijuhi arvates oli kogemus positiivne ja plaanis on selles suunas edasi liikuda. Samas leidis vastanu, et MDA kasutamine eeldab pikaajalist kliendisuhet ja toob kasu korduvkasutamisel. Vastaja hinnanguks oli, et kui ettevõte tahab olla edukas ja püsida turul aastakümneid, siis tuleks MDA kindlasti kasutusele võtta. Vastanud, kes ei omanud MDA-st põhjalikumat arusaama, püüdsid MDA-d seostada RUP kasutamisega abstraktse analüüsimudeli loomise ja spetsiifilise mudeli loomise kontekstis.

Objektitõkkekeel OCL

Objektitõkkekeel ehk *Object Constraint Language (OCL)* polnud kasutusel mitte üheski küsitletud ettevõtetest. Siiski oskasid vähemalt pooled vastanutest antud teemaga seostada teist UML-i laiendusmehhanismi stereotüüpide näol.

Kaks vastanut kasutab lisaks objekt-orienteeritud UML mudelitele ka andmebaaside kirjeldamiseks relatsioonilist IDEF1.X meetodit[12] ning ER diagramme. Teiste meetodite

[12] A structured approach to enterprise modeling and analysis. (2000). **Knowledge Based Systems, Inc.** URL: http://www.idef.com

eelistamise tingis vastanu poolt põhivahendina kasutusel olevas tööriistas Rational Rose andmebaaside modelleerimise ebamugavus. Seega kasutati andmebaaside modelleerimise tööriistana Visiot. Ka need, kes kasutasid Rationali Rose-i andmebaaside kirjeldamiseks leidsid, et selles osas on Rose-s veel arenguruumi. Enamus intervjueeritavatest aga olid seisukohal, et ei ole mõtet objekt-orienteeritud ja relatsioonilist lähenemist segi kasutada.

1.3.2 UML erinevate osade kasulikkus

Järgnevas osas on hindasid vastanud, mil määral on UML projektides olnud kasu erinevatest skeemidest. Et lihtsustada vastamist ning saada võrdluseks parem ülevaade, oli vastajatel palutud lisaks kommentaaridele hinnata skeemide kasulikkust ka kolme palli skaalal. Tabelites toodud numbrid omavad järgmist tähendust:

1. Pole olnud kasu
2. On olnud kasulik
3. On olnud hädavajalik

Kui lahtrid on jäetud tühjaks, on selle tähenduseks, et vastaja ei ole üheski projektis antud skeemi kasutanud. Kui on kasutatud numbrite vahel kaldkriipsu nt. 1/2, 2/3 vmt., tähendab see, et vastanu polnud kindel, kumba hinnangut üldistatult mitme projekti raames anda. Näiteks mõnes projektis võis skeem olla kasulik, teises hädavajalik. See võis oleneda näiteks kliendist või projekti suurusest/kestvusest. Samuti võis süsteemi nõuete täpsustamisel programmeerijale oleneda skeemi kasulikkus vastanute arvates sellest, kas tegu on näiteks kasutajaliidese programmeerijaga või süsteemi programmeerijaga. Tabelite ülaveerus toodud tähed A-st H-ni tähistavad küsitletud ettevõtteid. Tuleb ära märkida, et lisaks vastanute poolt antud numbrilistele hinnangutele oli vajalik ka arvesse võtta lisakommentaare.

Tabel 1: Kasutuslooskeemide kasulikkus

Kasutuslooskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel	2	3/ 2	2	2	3	1	2	2

Süsteemi nõuete täpsustamisel programmeerijatele	3	2/3	2	1	3	2	2	1/2
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks	3	2	2	2	1	1	2	2
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas	1	1	2	1	1	1	1	1

1.Koos kliendi esindajatega nõuete väljaselgitamisel ja määratlemisel.
2.Süsteemi nõuete täpsustamisel programmeerijatele.
3.Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks.

See on ootuspärane, et enamik kliendiga suhtlemisel kasutuslugusid kasutab, kuna see vastab kasutuslooskeemide formalismi loojate põhieesmärgile (vt. peatükk nr. 3.3.1)
Vastanu, kes ei pidanud kliendiga suhtlemisel kasutuslooskeeme vajalikuks, kasutas skeemide asemel tekstilist kuju. Üldiselt leiti, et kuna kasutuslugude tähistus ei ole väga keeruline, on see kliendile üsna arusaadav. Kasutuslugusid nimetati ka kasutusmallideks ning aktoreid tegijateks. Seejuures lähtuti Vello Hansoni poolt koostatud UML-i terminoloogiast.
"Süsteemi nõuete täpsustamisel programmeerijatele" peeti kasutuslooskeeme enamikel juhtudel vajalikuks. Leidus vaid üks vastaja, kelle arvates see nii pole ning üks vastaja pidas kasutusskeeme oluliseks olenevalt projektist või ka sellest, et näiteks kasutajaliidese loojale võib kasutuslooskeem olla hädavajalik aga süsteemi programmeerijale üsna kasulik.
"Dokumenteerimisel edaspidiseks hoolduseks ja laienduseks" oli kuus vastanut seisukohal, et kasutuslood on üsna kasulikud ja kaks vastanutest ei pidanud neid vajalikuks. Dokumenteerimisel peeti kasutuslooskeeme vajalikuks seetõttu, et kasutuslooskeemide järgi saab alati aru, millise funktsionaalsusega on tegu.
"Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas" kasutuslooskeemidest kasu ei nähtud.

Tabel 2: Klassiskeemide kasulikkus

Klassiskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel	1	1	2	1	1	3	1	1
Süsteemi nõuete täpsustamisel programmeerijatele	3	2	2	3	3	3	2/3	2/3
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks	3	2	2	3	2	3	2/3	2/3
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas	1	2	2	3	1	3	2	2/3

Tabeli 2 põhjal on klassiskeemide kasutamisel peetud oluliseks järgmist (olulisuse kahanemise järjestuses):

1.Süsteemi nõuete täpsustamisel programmeerijatele;

2.Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks;

3.Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas.

Kliendi esindajatega nõuete välja selgitamisel ja määratlemisel peeti klassiskeeme liiga keerukateks. Ka siin kujunesid vastused üsna ootuspäraseks, ehk et klassiskeemide kasulikkust nähti kõige enam süsteemi nõuete täpsustamisel programmeerijatele. Tähtsaks peeti klassiskeeme ka dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks. Nagu tabelist näha, olid klassiskeemid kõigis ettevõtetes kasutusel. Hinnanguliselt kasutati klassiskeeme sageli.

Tabel 3: Paketiskeemide kasulikkus

Paketiskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel	1	1			1	1	1	1

Süsteemi nõuete täpsustamisel programmeerijatele	3	3			1	2	2/3	2/3
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks	3	3			3	2	2/3	2/3
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas	2	3			2	1	2/3	2/3

Tabeli 3 põhjal on paketiskeemide kasutamisel peetud oluliseks järgmist (olulisuse kahanemise järjestuses):

1. Süsteemi nõuete täpsustamisel programmeerijatele;
2. Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks;
3. Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas.

Paketiskeemi võib tegelikult nimetada klassiskeemi erijuhuks, seega ka tulemused paketiskeemide erinevatele otstarvetele hinnangu andmises sarnanesid pisut klassiskeemile. Kliendi esindajatega nõuete välja selgitamisel ja määratlemisel ei peetud paketiskeeme vajalikuks mitte ühelgi korral. Küll aga peeti paketiskeeme vajalikuks nii süsteemi nõuete täpsustamisel programmeerijatele, dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks kui ka rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas. Paketiskeeme peeti vajalikuks kasutada suurte süsteemide puhul. Kui paketiskeeme ei kasutatud, siis peeti piisavaks klassiskeemide erinevaid vaadeid.

Tabel 4: Olekuskeemide kasulikkus

Olekuskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel	2		3	2/3		3	2/3	
Süsteemi nõuete täpsustamisel programmeerijatele	3		3	2/3		3	2/3	
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks	2		2	1		3	1	

Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas	1		2	2/3		3	2/3	

Tabeli 4 põhjal on olekuskeemide kasutamisel peetud oluliseks järgmist (olulisuse kahanemise järjestuses):

1. Süsteemi nõuete täpsustamisel programmeerijatele;
2. Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel;
3. Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas.

Juhul, kui olekuskeeme kasutati, siis ei peetud neid vajalikuks kasutada iga projekti käigus, vaid pigem spetsiifilisemat laadi projektides, kus on palju keerukaid ning paljude olekutega objekte. Eriti vajalikuks peeti olekuskeeme reaalajarakendustes.

Tabel 5: Tegevusskeemide kasulikkus

Tegevusskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel	2		2	2/3	2	2	1	
Süsteemi nõuete täpsustamisel programmeerijatele	3		2	2/3	2	2	2/3	
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks	3		2	2/3	1	2	1	
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas	1		3	1	1	2	2/3	

Tegevusskeemide kasulikkust on erinevad vastajad hinnanud üsna erinevalt. Näiteks kui üldjuhul projektimeeskonna tehniliste liikmete seas rakenduse paremaks mõistmiseks tegevusskeemidest kasu ei nähtud või neid ei kasutatud, siis kaks vastanutest pidasid tegevusskeeme kas hädavajalikuks või kasulikuks.

Tabel 6: Jadaskeemide kasulikkus

Jadaskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel		3		1	1	1	2/3	1
Süsteemi nõuete täpsustamisel programmeerijatele		2		2/3	2	2	2/3	2/3
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks		2		1	2	2	1	1
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas		2		1	1	2	2/3	1

Kui jadaskeem ja koostööskeem on mõneti omavahel sarnased, siis ülekaalukalt kasutati pigem jadaskeemi kui koostööskeemi. Siiski, nagu tabelist näha, peeti jadaskeeme pigem keskmiselt või vähe kasulikuks. CASE vahendid võimaldavad jadaskeeme konverteerida koostööskeemideks ning vastupidi.

Tabel 7: Koostööskeemide kasulikkus

Koostööskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel		1				1	1	
Süsteemi nõuete täpsustamisel programmeerijatele		3				2	2/3	
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks		3				2	2/3	
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas		1				2	1	

Koostööskeeme kasutanud olid alla poolte vastanutest. Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel ei olnud koostööskeemidest kasu mitte ühegi küsitletu hinnangul.

Tabel 8: Komponendiskeemide kasulikkus

Komponendiskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel		1	2		1	2	1	
Süsteemi nõuete täpsustamisel programmeerijatele		3	2		1/2	2	1	
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks		3	2		1/2	2	2/3	
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas		2	2		1/2	2	2/3	

Komponendiskeeme on kasutanud viis vastanut, kuid nende kasutamist ei peetud vajalikuks kõigis projektides.

Tabel 9: Levitusskeemide kasulikkus

Levitusskeem	A	B	C	D	E	F	G	H
Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel		2				1	1	2
Süsteemi nõuete täpsustamisel programmeerijatele		2				2	1	1/2
Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks		2				2	2/3	2
Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete seas		2				2	2/3	1

Vello Hansoni sõnaraamatu põhjal nimetati neid skeeme evitusskeemideks. Levitusskeemide kasutamist pidasid vajalikuks vaid pooled vastanutest, kuna ülejäänud leidsid, et enamasti on riistvaraline struktuur juba eelnevalt olemas ning seda muudetakse vähe.

UML-i tähistused

UML-i enda tähistuse kasutamisel ükski küsitletu häirivaid aspekte ei leidnud. Küll aga peeti häirivaks, et erinevad CASE vahendite tootjad mõnikord muudavad sümboolikat ja lisavad spetsiifilist semantikat. Leiti, et head CASE vahendid on kallid, odavad aga ei ole mugavad.

Seega ka ettepanekuteks ei olnud muuta UML tähistusi, vaid töötada välja standardid CASE vahendite tootjatele.

UML-i komponentide taaskasutatavus

Pooled vastanutest taaskasutavad uutes süsteemiarendusprojektides eelnevates UML-il põhinevatest süsteemiarendusprojektides loodud komponente sageli, kaks vastanut harva ja kaks mitte kunagi. Põhjalikumalt kommenteeriti, mida mõeldi sageli kasutamise all ning kui ei kasutata, siis miks ei kasutata. Kuid sageli taaskasutamise juures tekkis ka tõlgendamise küsimus, et kui palju taaskasutatakse UML-is kirjeldatud komponente ja kui palju UML-is modelleerimata komponente. Kahe intervjueeritava vastustest selgus, UML-i abil on kirjeldatud probleemivaldkonda, kuid mitte tehnilist realisatsiooni. Analüüsi osa ei peetud enamasti võimalikuks uuesti kasutusele võtta, kuna probleemvaldkonnad kipuvad olema liiga erinevad. Kuid madalama taseme komponentide kirjeldamist mudelitega peeti vajalikuks vaid vajadusel neid dokumenteerida kasutamiseks muudes projektides või edasiarendamiseks teiste töötajate poolt. Üks vastanutest lisas, et juurutamisel olev UML-i modelleerimisevahend Rational XDE on integreeritud põhilisse töökeskkonda Microsoft Visual Studio .NET, mis võimaldab lisaks domeenimudeli ja selle juurde kuuluvate kasutuslugude diagrammide modelleerimisele hõlpsamat pöördprojekteerimist madala taseme süsteemsete komponentidega, mis muidu jääksid disainifaasis modelleerimata, ning seeläbi tekib tehtud tööst parem dokumentatsioon. See omakorda soodustab lahenduses loodud madala

taseme süsteemikomponentide taaskasutamist selliselt, et samu komponente kasutades on võimalik ehitada uus süsteem asendades eelnevalt loodud süsteemis ainult domeenimudelit ja kasutuslugusid realiseerivad tarkvarakihid. Üldjuhul nimetati UML-is modelleerimise vajadust sageli taaskasutust leidvate teatud baasfunktsioonide juures, mida mitmes rakenduses vaja läheb. Vastanu, kes UML-i harva kasutab, tõi põhjusena välja, et korduvkasutatav tarkvarakomponent maksab märksa rohkem, kui ühekordselt kasutatav komponent. Küsimus ei ole küsitletu arvates siinkohal korduvkasutuse põhimõttes, vaid tarkvarafirma profiilis - kui suured on projektid, kui suur on eelarve ning missuguse iseloomuga on loodavad süsteemid..

1.3.3 UML tööriistad

Tabel 10: Tööriistad

Tööriistad	**Nimetatud kordade arv**
Rose-i erinevad versioonid (Rational)	7
Visio erinevad versioonid (Microsoft Visio)	5
Argo UML (Gentleware)	3
Enterprise Architect Professional (Sparx Systems)	3
MagicDraw UML Standard (MagicDraw)	2
AllFusion ERwin Data Modeler (Computer Associates International, Inc.)	2
Poseidon for UML - Community (Gentleware)	1
Control Center (Together)	1
EclipseUML (Omondo)	1
Tau UML Suite (Telelogic)	1
Real-time Modeler (ARTiSAN)	1
Oracle9*i* Designer (Oracle)	1
RefactorIt (Aqris)	1
Ettevõttesiseselt loodud programmid	2
Tekstiredaktorid nt. Word	8

Tabel 10 annab ülevaate küsitletute poolt kasutatud tööriistadest. Välja on toodud nii UML-i tööriistad kui ka tööriistad, mis ei ole mõeldud UML-i mudelite joonistamiseks. Iga tööriista järel toodud number tähistab, mitu korda antud tööriista nimetati. UML-i tööriistadest leidis enim (seitsmel korral) mainimist Rational Rose, mida kasutab igapäevatöös kuus vastanut.
Andmebaaside modelleerimiseks eelistasid mitmed vastanutest lisaks kasutada Visio-t ja Erwin-it. Tööriista RefactorIt kasutas üks vastanu koodi analüüsimiseks. Hetkel kasutusel olid veel ka Enterprise Architect, MagicDraw ning Poseidon, Eclipse ning Oracle Designer. Rose-iga võrdväärsetena nimetati ära Control Center, Tau UML Suite ja Real Time Modeller. Argo UML-i oli kasutatud, kuid enam mitte.

Rose-i kasutamine oli enamasti ettevõtte valik. Rose-i kasuks otsustamise põhjustena ettevõttes toodi välja:

- Rose-i kohta on kõige rohkem kirjandust,
- Rose on kõige usaldusväärseim,
- müügimeeste hea töö,
- katab kogu UML-i sümboolika,
- dokumentatsiooni genereerimise võimalus,
- integreeritavus (nt. .Net, oma tööriistadega jmt.),
- toetab protsessi (nt. RUP).

Enamikel juhtudest oldi ettevõtte valikuga kasutada Rose-i rahul, kuid lisaks plussidele toodi välja ka puudusi. Kõige sagedamini nimetati andmebaaside modelleerimiseks kasutatava sümboolika ebamugavust. Samuti andmebaaside projektide vahelise sünkroniseerimisega tekkida võivaid probleeme, kuna projektidevaheline integreerimine on üsna jäik. Sel põhjusel kasutati andmebaaside loomiseks pigem Visio-t või Erwin-it kuid oli ka neid, kes ka andmebaase modelleerisid Rose-is. Kasutuselolevatest versioonidest mainiti kahel korral Rose Enterprise-i ning ühel korral XDE versiooni. Ühes ettevõttes oli toimumas üleminek Rose Enterprise-i kasutamiselt Rose XDE kasutamisele, kuna selle eeliseks on integreeritus nende põhilisse töövahendisse ehk Visual Studiosse.
Sparx Enterprise-i kasuks otsustanute põhjused olid järgmised:

- mugav ja loogiline kasutajaliides:
- protsessi toetus (RUP);
- keskmisest odavam hinnaklass;
- ühilduvus Sparx Enterprise .Net platformiga.

MagicDraw kasutamise põhjuseks oli selle kordi odavam hind, kui näiteks Rose-il. Kõigi turul olevate tööriistadega võrrelduna peeti MagicDraw-d aeglaseks, ehkki ainult Java-põhiste vahendite hulgas on MagicDraw üks kiireimaid. Üks vastanutest pidas ebamugavaks MagicDraw kasutajaliidest.

Ka Poseidon-i kasutamise põhjenduseks oli tasuta versiooni kasutamise võimalus. Poseidoni puhul peeti negatiivseks, et pole võimalik tekstiredaktoris Poseidoni vahenditega genereeritud dokumentatsiooni redigeerida. Konkreetsel juhul sai määravaks ka ettevõtte profiil, kus esmatähtsad on andmebaaside rakendused ning spetsiifilised rakendused. Tulenevalt antud ettevõttega koostööd tegevast ettevõttest, ei ole spetsiifiliste rakenduste osas võimalik kasutada suhtlemisel UML-i, kuna partnerfirma kasutab mittestandardset keelt.

Argo UML-i kasutamisest olid vastanud loobunud järgmistel põhjustel:

- ei ole mugav dokumentatsiooni genereerida kliendiga suhtluseks (tekstiredaktorisse skeemide paigutamisel puudub nende redigeerimise võimalus);
- ei kata kogu UML-i.

Argo UML-i eeliseks on kättesaadavus vabavarana.

Üldiselt toodi tööriistade puudustena välja:

- paljud tööriistad pole täies ulatuses UML-i põhised;
- pole ühilduvad;
- kasutavad sageli tootjaspetsiifilisi laiendusi ja tähistusi.

Seega üheks Rose-i pooldajaskonna peamiseks argumendiks on ka vahendi standardsus. Võib öelda, et pigem ollakse valmis järgi andma tööriistade mugavuse osas, kui hinnas.

Viis vastanutest kasutab erinevaid tööriistu erinevate projektide juures eraldi. Vaid üks vastanu kasutab mitut tööriista paralleelselt: andmebaaside jaoks kasutab Visiot ning

UML-i tööriistana Rose-i. Kaks küsitletutest kasutab tööriistu eraldi või lahus olenevalt projektist.

Koodi genereerimine

Üle poole vastanutest leidis, et mudelitest genereeritud koodi tuleb suures osas muuta ning täiendada. Vaid üks vastanu pidas koodi muutmist ning täiendamist vajalikuks vähesel määral. Kaks vastanutest mudelitest automaatselt koodi ei genereeri, kuna ei peetud vajalikuks koodi ja mudeleid omavahel sünkroonis hoida. Üks vastanu leidis, et mudelid muutuvad liigselt keerukateks, kui püüda neisse kogu semantika hõlmata. Siiski peeti mudelitest koodi genereerimisel peamiseks takistuseks, et tööriistad ei ole selleks piisavalt tasemel. Mudelite abil äriloogika genereerimist ei pidanud ükski vastanu veel lähitulevikus võimalikuks. Küll aga leidsid mitmed vastanud, et tööriistad on hetkel piisavalt head selleks, et genereerida mudelitest automaatselt tühjad meetodid, mis märgatavalt hõlbustab käsitsi koodikirjutamist.

Pöördprojekteerimine

Pöördprojekteerimist kasutavad kaheksast vastanust kuus. Huvitav lähenemine, et kui pöördprojekteerimisel loetakse diagrammile palju elemente, nii et nende mõistmine raskendatud on, siis on pöördprojekteerimine vahendiks süsteemi arhitektuurivigade leidmisel ja võib olla halva arhitektuuri indikaatoriks, kuna pöördprojekteerimisel luuakse olemasolevast koodibaasist mudel, mis on tunduvalt ülevaatlikum ja semantiliselt rikkam, kui tavaline kood.

1.3.4 Analüütiku-kliendi suhtlus

Käesolevas osas on kliendi all mõeldud äriklienti, mitte ettevõtte IT partner. Ligi 90 % juhtudest leiti UML olevat kliendiga suhtlemisel üsnagi edukas. Vaid üks vastanu ei pidanud kliendiga suhtlemisel UML-i vajalikuks, kuna antud ettevõttes jäävad kliendisuhted enamasti nii lühiajaliseks, et kliendi koolitamisel pole mõtet. Ülejäänud vastanutest pidasid kliendiga suhtlemisel UML-i enim oluliseks suurte projektide puhul kus luuakse mahukat tarkvara.

Huvi pakkus analüütiku-kliendi suhtluse kontekstis ka see, kui ühtlaselt ning sügavuti on kliendid kaasatud projekti analüüsi ja disaini ehk kas eelistatakse klientidega suhtlemisel kasutada vaid teatud tüüpi skeeme või püütakse klienti kursis hoida ja kaasata ka tehnilisematesse detailidesse. Ilmnes, et kasutuslooskeeme kasutati kõige enam ehk mainiti neljal korral, kahel korral mainiti tegevusskeeme ning jadaskeeme ning ühel korral klassiskeeme.

1.3.5 Organisatsiooni andmed

Küsitletavatest ettevõtetest neljas on tarkvara arendusega seotud rohkem kui viiskümmend inimest. Ühes ettevõttes on tarkvara arendusega seotud kolmkümmend inimest ning kahes asutuses on vastava valdkonnaga seotud tööliste arv üksteist. Vaid ühes firmas küsitletutest jääb tarkvara arendamisega tegelevate töötajate arv hetkel vahemikku kolmest viieni, kuid lisaks nimetatud firmale töötavad nii küsitletu kui ka ülejäänud antud ettevõtte töötajad veel kolme asutuse meeskonnas ning seega on tegu justkui virtuaalse organisatsiooniga ning selliselt defineerituna ületab tööliste arv viiekümne piiri.

Peamiselt müügiks arendati tarkvara neljas ettevõttes ning peamiselt firmasiseseks kasutamiseks kolmes. Mitme ettevõtte heaks töötav tarkvara arendaja on seotud nii ettevõttega, kus arendatakse tarkvara peamiselt ettevõttesiseseks kasutuseks kui ka asutusega, kus tarkvara arendatakse peamiselt müügiks.

Küsitletud ettevõtete tegevusvaldkondade määratlemisel kolmel juhul leiti, et ettevõtte profiili katavad ära kõige paremini enam kui kaks pakutud valdkondadest. Kõigil kolmel juhul olid nendeks kooslusteks tarkvara arendus ja konsultatsioon. Kahel juhul oli tegu konkreetselt finantssektoriga ning kahel juhul tarkvara arendusega. Üks küsitletud organisatsioonidest klassifitseerus riigiasutuseks.

1.3.6 Intervjueeritava IT taust

Intervjueeritavatest üks on tippjuht, kolm on projekti- või grupijuhid ning neli pidasid sobivaimaks ametinimetuseks tarkvara arendaja.

Eranditult kõigil vastanutest on kõrgharidus ning neist ühel füüsikaalane magistrikraad ning seitsmel IT alane. Kaks vastanutest omasid IT magistrikraadi ning üks õppis hetkel Tallinna Tehnikaülikoolis doktorantuuris info- ja kommunikatsioonitehnoloogia erialal.

Mis on UML-i õppimisel tähtsaim

UML-i õppimisel tähtsaimate allikate järjestus on järgmine:

- raamatud, ajakirjad ning praktiline kogemus;
- kursused ning veebilehed;
- ülikooli- või keskkooliprogrammid;
- kolleegid ja konsultandid.

Kõik vastanud pidasid õppimisel enim oluliseks raamatuid ning ka praktilist kogemust. Leiti, et abi võib olla tööriistadega kaasas olevate õppematerjalide läbi töötamisest ning näidisprojekti uurimisest. Veebilehti peeti oluliseks teemaga piisavalt kursis olles ühele või teisele arusaamale konkreetses punktis kinnituse otsimisel. Õppimist alustada veebilehtedest ei soovitatud, kuna materjal võib muutuda liiga laialivalguvaks. Küsimuses kas objekt-orienteeritud tähistuste kasutamiselt üleminek UML-ile on lihtsam või ei anna midagi juurde ning pigem segab, olid arvamused erinevad. Käesoleva töö autor jagab esimest seisukohta. Üks vastanutest on UML-i teemalise koolituse lektor ning jagas oma teadmisi UML-i õppimise ning õpetamise osas põhjalikumalt, pannes suurt rõhku sellele, et lisaks kursustele on väga oluline ka praktiline kogemus ning õpilasepoolne huvi. Huvitav on koolitaja nägemust kõrvutada poole aastase kestvusega koolitusel osalenud intervjueeritava kogemustega. Koolituse vajadus selgus intervjueeritavale olles koos UML-i rakendamisest huvitatud magistrantidega ülikoolist kogutud teadmiste baasil reaalselt projekti katsetanud, mille tulemused osutusid kesisteks. Pool aastat kestnud loengute sarja järgselt viidi järgmine projekt läbi koolitajaga koostöös ning kolmas koolitaja nõustamisel. Alles seejärel oldi valmis iseseisvalt piisavalt edukad olema. Kahjuks teisel koolitusi maininud vastajal isiklik kogemus puudus, kuna ettevõtte töötajatest olid koolitustel osalenud vähesed, kuna ettevõtte eelarves nappis selleks vahendeid.

UML-i õppematerjalide kättesaadavus

UML-i õppimise vahendeid peeti pigem kättesaadavaks, kuid oldi mõneti kõhkelval seisukohal. Esmalt toodi välja, et eesti keelset materjali peaaegu ei leidu. Samas oli siinkohal üks küsitletutest täiesti vastandlikul arvamusel, kuna leidis, et eesti keelne tõlge pigem segab tööd, kui hõlbustab. Sellel arvamusel olnud vastaja põhjendas oma seisukohta kahest aspektist, et esiteks areneb antud valdkond väga kiiresti ning tõlkijad ja/või eestikeelsete materjalide koostajad ei jõua piisavalt sammu pidada ning teiseks on valdkond piisavalt spetsiifiline, et eesti keelele püsima jäämisele see mõju ei avalda. Ka siinkohal mainiti, et kui õppimise vahendit laiemalt tõlgendada, võib õpilastel, kes pole veel IT firmas karjääri alustanud, puudu jääda praktiseerimise kogemusest. On vaja õppida nägema UML-i osa tervikus - puutuda kokku äriloogikaga, puutuda kokku projekti eelarvega jmt. Samuti leiti olevat palju tähistuse ja semantika õpetusi, kuid vähem praktika kirjeldusi ja seda nii raamatutes kui ka veebilehtedel.

Ka rahalistes vahendites võib tekkida küsimus nagu pisut eelnevalt juba kirjeldatud firma näite põhjal, kus UML-i kursusi oli seetõttu võimalik läbida vaid mõnedel töötajatest, mitte kõik, kellel asja vastu huvi. Samuti raamatute muretsemisel tuleb mõelda rahaliste vahendite peale, kuna raamatukogudes, sealhulgas ka ülikooli raamatukogudes, on vähe ja enamasti pisut vananenud raamatud. Pigem omavad kaasaegseid raamatuid IT firmad. Eraldi mainiti Martin Fowleri raamat UML Distilled[13] ning ajakirja The Rational Edge[14].

[13] **Fowler, M., Scott, K.** (1998). UML Distilled: Applying the Standard Object Modeling Language. Addison Wesley Longman, Inc.

[14] **The Rational Edge**: E-Zine for the Rational Community.
URL: http://www.therationaledge.com/

2 UML-i õppematerjal

2.1 Kuidas õppida UML-i

Soovitatav on alustada UML-i õppimist esialgu lihtsamatest skeemidest ning seejärel vastavalt vajadusele keerukamate skeemide juurde liikuda. Olles eelnevalt tuttav objekt-orienteeritud meetodiga, samuti modelleerimisvahenditega, võib see UML-ist aru saamist lihtsustada, kuid samas ei tohiks neid omavahel segamini ajada. UML-i õppides ei piisa ainult lugemisest - oluline on ka praktiline kogemus. Vahendina, milles UML-i skeeme joonistada, võiks kasutada näiteks programmi Rational Rose, kuid vastavalt soovile võib valida ka mõne teise alternatiivse programmi. (vt. tabel 11)

Vastavalt sellele, millises projekteerimisjärgus ollakse ning mis küljest soovitakse süsteemi käsitleda, tuleb valida ka skeem. UML-is on kasutusel järgmised skeemid:

- kasutuslooskeemid *(use case diagram)*
- klassiskeemid *(class diagram)*
    - pakettskeemid *(package diagram)*
- käitumuslikud skeemid *(behavior diagrams):*
    - olekuskeemid *(statechart diagram)*
    - tegevusskeemid *(activity diagram)*
    - interaktsiooniskeemid *(interaction diagrams):*
        - jadaskeemid *(sequence diagram)*
        - koostööskeemid *(collaboration diagram)*
- teostusskeemid *(implementation diagrams)*:
    - komponentskeemid *(component diagram)*
    - levitusskeemid *(deployment diagram)*

[SPETSIF, lk 1-3]

Kuigi kõik need skeemid võivad olenevalt projekti spetsiifikast ja hetkel käesolevast arenduse faasist vaadatuna olla vägagi olulised, on käesolevas õppematerjalis keskendutud siiski enim ettetulevatele skeemidele ning kõigist arenduse faasidest on välja toodud just olulisemad skeemid. Kasutuslooskeeme luuakse peamiselt kavandamise

faasis, kuid neile toetutakse läbi kogu protsessi. Klassiskeemid ning interaktsiooniskeemid omandavad suurema tähtsuse disaini faasis. Olekuskeeme kasutatakse samuti kasutatakse disaini faasis. Levitusskeemid on vajalikud süsteemist füüsilisel tasandil ülevaate andmiseks. Tegevusskeeme ning komponentskeeme on vaid lühidalt tutvustatud, kuna tuginedes käesoleva töö raames korraldatud uuringule, on nende kasutamise osatähtsus on reaalsetes projektides väiksem.

2.2 Üldine arendusprotsess

Meetod koosneb nii protsessist kui modelleerimiskeelest. UML on modelleerimiskeel ja seega ei ole protsessist sõltuv.

Siiski ei oleks modelleerimistehnikatel mingit mõtet, kui ei teaks, kuidas need kuuluvad protsessi. UML-i saab kasutada ükskõik millise protsessi puhul analüüsi ja projekteerimisotsuste üles märkimiseks.

Protsess koosneb neljast faasist: lähteuuring *(inception),* kavandamine *(elaboration),* valmistamine *(construction)* ja levitamine *(transition)* ja iga faas omakorda koosneb iteratsioonidest *(iterations)*:

- Lähteuuringu jooksul selgitatakse üldjoontes välja, kui palju projekt maksma läheb ja kui palju kasumit tagasi teenib. Samuti otsustatakse, millise ulatusega projekt saab olema. Siinjuures saadakse projekti tellijalt nõusolek projekti jätkamiseks. Selgub projekti visioon.
- Kavandamise jooksul kogutakse kokku detailsemad nõuded, tehakse üldtasemelist analüüsi, et luua baasarhitektuur ja valmistamise plaan. Selles faasis võiks püstitada järgmised küsimused:
  Mida tegelikult ehitama hakatakse?
  Kuidas kavatsetakse ehitada?
  Mis tehnoloogiat kavatsetakse kasutada?
  On oluline pöörata tähelepanu riskidele (nõuete riskid, tehnoloogilised riskid, oskuste riskid, poliitilised riskid).
- Valmistamise faas koosneb paljudest iteratsioonidest. Iga iteratsioon on üks miniprojekt. Igale kasutusloole vastab iteratsioon, mis koosneb analüüsist, disainist, kodeerimisest ja integreerimisest. Testimist ja integratsiooni ei tohi jätta kogu projekti lõppu, kuna see võtab enamasti kauem aega, kui arvati. Ja avastatud vigu on tunduvalt kallim parandada kui vastases loomisjärgus
- Levitamise faasis antakse tarkvara üle kasutajatele. See võib sisaldada ka beeta testimist, kasutajate koolitamist jmt. [RUP]

2.3 UML-i modelleerimistehnikad

2.3.1 Kastustuslood

2.3.1.1 Kasutusloo modelleerimine

Kasutusloo modelleerimistehnikat kasutatakse selleks, et näidata, mida uus süsteem peaks tegema või mida olemasolev süsteem juba teeb. Kasutusloo mudel ehitatakse iteratiivses protsessis, mille käigus toimub diskussioon süsteemi arendajate ja tellijate (ja/või lõppkasutajate) vahel, mille tulemusena saadakse vajaduste spetsifikatsioon, millega kõik osapooled nõus on.

Kasutusloo mudeli põhikomponentideks on kasutuslood, aktorid *(actor)* ning modelleeritav süsteem. Süsteemi piirid defineeritakse süsteemi poolt käsitletava funktsionaalsusega. Funktsionaalsus esitatakse hulga kasutuslugudega, millest igaüks spetsifitseerib tervikliku funktsionaalsuse. Kasutuslugu peab andma mingi väärtuse aktori jaoks, kus väärtuseks on miski, mida aktor soovib süsteemilt. Aktor on igasugune välisüksus, kes/mis soovib suhelda antud süsteemiga. See võib olla inimene, süsteem või riistvaraseade, mis peab suhtlema antud süsteemiga.

2.3.1.2 Kasutuslugude põhieesmärgid

- Otsustada ja kirjeldada süsteemi funktsionaalsed vajadused koostöös tellijate (ja/või lõppkasutajate) ja süsteemi projekteerijate/ehitajate vahel.
- Anda selge ja kooskõlaline kirjeldus sellest, mida süsteem peab tegema. Seda kirjeldust kasutavad läbi kogu arendusprotsessi süsteemi kõik osapooled, et üle anda soovitud tulemusi/funktsionaalsusi.
- Anda alus süsteemi testimisele, et saaks kindlaks teha, kas üleantud süsteem täidab esialgselt soovitud funktsionaalsust.
- Lihtsustada süsteemi muutmist ja laiendamist. [KOBRYN]

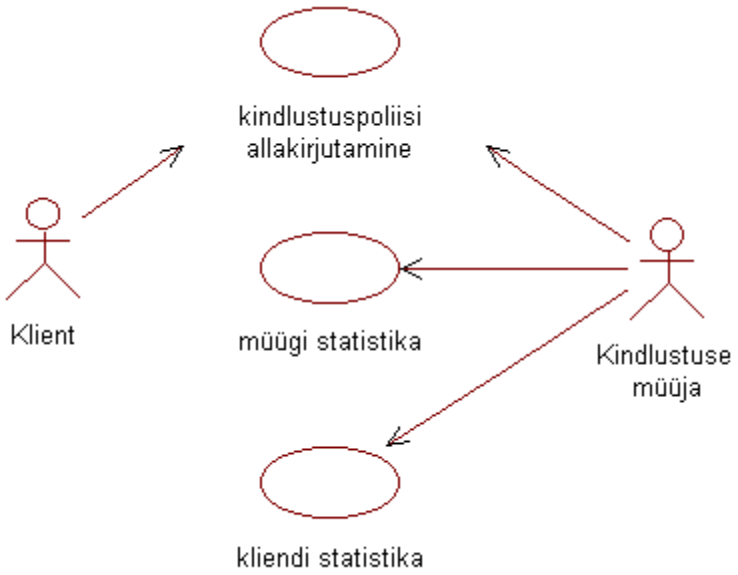
Kasutusloomudeli koostamise osadeks on süsteemi defineerimine, aktorite ja kasutuslugude leidmine, kasutuslugude kirjeldamine, seoste defineerimine kasutuslugude vahel ja mudeli õigsuse kontroll.

On hea, kui süsteemi kasutaja (esindab aktorit kasutuslooskeemis) osaleb aktiivselt kasutusloo modelleerimisel, sest see võimaldab kohandada mudelit täpselt kasutaja soovidega. Kasutuslood kirjeldatakse tellija/kasutaja keeles ning mõistetes. Nii kindlustatakse kommunikatsioon süsteemi tellijate/kasutajate ning arendajate, integreerijate, testijate ja muude osapoolte (müük, dokumenteerimine) vahel.
Kasutuslugusid võib skeemide joonistamise asemel ka lihtsalt tekstilisel kujul esitada ning sel juhul nimetatakse neid kasutusloo jutustusteks *(use case narratives)*.

2.3.1.3 Kasutuslooskeem

Kasutusloomudeli visuaalseks kujutamiseks kasutatakse kasutuslooskeeme, kusjuures kasutusloomudel võib olla jagatud paljudeks kasutuslooskeemideks. Kasutuslooskeem sisaldab mudelielemente aktorite ning kasutuslugude jaoks ning näitab seoseid nende elementide vahel.

Kasutusloo modelleerimine defineerib arendatava süsteemi piirid. Piiride täpne paikapanemine ehk süsteemi piiritlemine on eduka süsteemiarenduse esimene tingimus. Otsustatakse süsteemi esimese väljalaske suurus. See peaks olema väheste, kuid kõige tähtsamate funktsionaalsustega ning sellise arhitektuuriga, kuhu saaks hiljem soovitud funktsionaalsusi juurde lisada. Süsteemi jaoks tasuks kohe koostada ka põhimõistete sõnastik, mida kasutatakse kasutuslugude kirjeldamisel ning edasise domeenianalüüsi käigus mitmetimõistmise vältimiseks.



Joonis 1: Kasutuslooskeem

Aktorid

Aktor on keegi või miski, mis suhtleb süsteemiga ning kasutab süsteemi. Aktor saadab sõnumeid süsteemile ja/või saab sõnumeid süsteemilt. Aktorid viivad läbi kasutuslugusid. Aktor võib olla inimene või teine süsteem. Aktor on tüüp (klass), mitte eksemplar. Ta esitab süsteemis rolli, mitte üksikkasutajat. Ühe isiku kohta võib olla mitu aktorit, sõltuvalt tema rollist süsteemis. Aktori nimi peab väljendama tema rolli. Kasutusloo algatab aktor, kes saadab kasutusloole sõnumi. Kui kasutuslugu on läbi viidud, peab kasutuslugu saatma sõnumi ühele või mitmele aktorile. Need sõnumid võivad minna ka teistele aktoritele lisaks sellele, kes algatas kasutusloo. Aktiivne aktor on see, kes käivitab kasutusloo. Passiivne aktor üksnes osaleb ühes või enamas kasutusloos.

Aktorid UML-is on klassid stereotüübiga “actor” ning klassinimi on ka aktorinimeks (väljendades aktori rolli). Aktoriklass võib omada atribuute, käitumist ning dokumentatsiooniomadust. Kasutuslooskeemil on vastava stereotüübi ikooniks tavaliselt kriipsujuku.

Kasutuslood

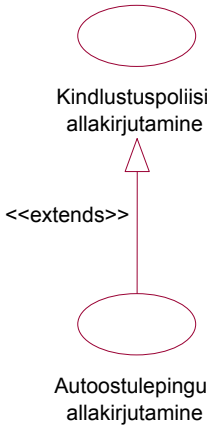
Kasutuslugu esindab terviklikku funktsionaalsust aktori jaoks. UML-is defineeritakse kasutuslugu kui süsteemi poolt läbiviidavate tegevuste järjestuste hulk, mis annab

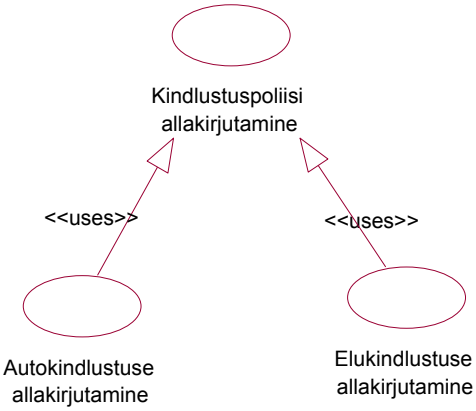
jälgitava väärtustulemuse konkreetsele aktorile. Tegevused võivad sisaldada kommunikatsiooni paljude aktoritega (kasutajad ning teised süsteemid), samuti teostada arvutusi ja töid süsteemi sees. Kasutusloo omadused:

- Kasutusloo käivitab aktor: Kasutuslugu viiakse alati läbi aktori poolelt. Aktor peab otseselt või kaudselt käskima süsteemil kasutuslugu täita. Mõnikord ei pea aktor otseselt kasutuslugu käivitama *(triggerid)*.
- Kasutuslugu annab aktorile väärtuse: kasutuslugu peab kasutajale üle andma "käegakatsutava" väärtuse. Väärtus ei pea olema nähtav, võib olla lihtsalt tajutav.
- Kasutuslugu peab olema täielik *(complete)*: Tavaline viga: jagatakse kasutuslugu väiksemateks kasutuslugudeks, mis realiseerivad üksteist nagu üksteist väljakutsuvad funktsioonid programmeerimiskeeles. Kasutuslugu pole täielik, kui ta pole tootnud lõppväärtust kasutaja jaoks. Kasutuslood ühendatakse aktoritega seoste kaudu, mis näitavad, milliste aktoritega kasutuslugu suhtleb. See on tavaliselt üks-ühele seos ilma suunata, mis tähendab, et üks aktori eksemplar suhtleb ühe kasutusloo eksemplariga ning kommunikatsioon toimub mõlemas suunas. Kasutuslugu nimetatakse tema eesmärgi järgi, nagu "Kindlustuspoliisi allakirjutamine". Tavaliselt on nimeks fraas, mitte üksik sõna. Kasutuslugu kirjeldab tervikfunktsionaalsust, kaasa arvatud võimalikud alternatiivid, vead ning erijuhtumid kasutusloo täitmisel. Kasutusloo eksemplari nimetatakse stsenaariumiks *(scenario)*, mis esindab süsteemi konkreetset täitmist. Kasutuslood UML-is tähistatakse ellipsiga, mis sisaldab kasutusloo nime.

Seosed kasutuslugude vahel

Kasutuslugude vahel on kolme liiki seoseid: laiendab *(extends)*, kasutab *(uses)* ning üldistus *(generalization)*.





Joonis 2: Kasutuslooskeem: laiendab seos **Joonis 3: Kasutuslooskeem: kasutab seos**

- Laiendab seos: on üldistusseos, kus üks kasutuslugu laiendab teist, lisades tegevusi üldisemale kasutusloole. Kui üks kasutuslugu laiendab teist kasutuslugu, siis esimene võib sisaldada laiendatava kasutusloo mingit käitumist. Ta ei pea sisaldama kogu käitumist, vaid võib valida, milliseid osi üldisema kasutusloo käitumisest ta soovib taaskasutada. Laiendatav kasutuslugu peab olema täielik. Laiendatud kasutuslugu võib käsitleda üldisema kasutusloo erijuhtumeid, mida on raske kirjeldada üldisemas kasutusloos eneses, või mis tehakse süsteemi arendusprotsessi kulgedes. Laiendusseost näidatakse (nooleots laiendatava kasutusloo poole) üldistussümboliga koos stereotüübiga “extends”.

- Kasutab seos: on samuti üldistusseos. Kui palju kasutuslugusid omavad ühist käitumist, võib see käitumine olla modelleeritud eraldi kastutuslool, mida teised kasutuslood tarvitavad. Kui üks kasutuslugu kasutab teist, peab ta kasutama teise kogu käitumist (kuigi kasutatava kasutusloo tegevused ei pea olema samas järjestuses; nad võivad olla läbisegi kasutava kasutusloo tegevustega). Kui kasutuslugu, mida kasutatakse, iseennast kunagi ei kasuta, nimetatakse teda abstraktseks kasutuslooks (*abstract use case)*. Kasutusseost näidatakse üldistussümboliga (nooleots kasutatava kasutusloo poole) koos stereotüübiga “uses”.

2.3.1.4 Millal kasutada kasutuslugusid

Kasutuslugusid tuleks kasutada alati.

Kasutustlood on hädavajalikud nõudmiste väljaselgitamisel ja planeerimisel ning iteratiivse projekti kontrollis.

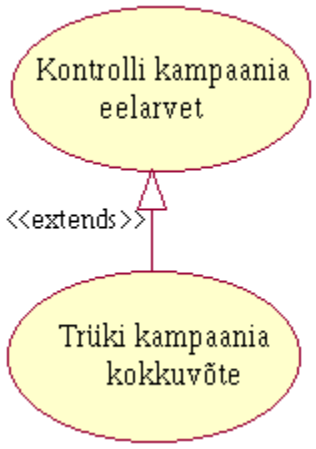
Kasutuslugude leidmine on laienduse faasi esmaseid ülesandeid.

Kasutuslugusid võib ka projekti käigus juurde tekkida ja neid peaks läbivalt kogu projekti jooksul jälgima.

Kasutuslood võiks pigem detailsemalt välja tuua, kui liiga laialt, kuid liigne detailsus võib samas liiga kirjuks minna. Seega tuleks ikkagi optimaalseim lahendus püüda leida. [FOWLER, lk. 51]

Ülesanded 1:

1. Milline vastusevariantidest kirjeldab allolevat skeemi?



- Kampaania kontrollimise eelarve laiendab kampaania trükkimise kokkuvõtet
- Kampaania kontrollimise eelarve sisaldab kampaania trükkimise kokkuvõtet
- Kampaania trükkimise kokkuvõte laiendab kampaania kontrollimise eelarvet (õige)

2. Missugune järgnevatest on aktori parim definitsioon?

- Aktor esindab süsteemis kasutajat
- Aktor esindab rolli, mida kasutaja süsteemis mängib
- Aktror esindab rolli, mida mängib süsteemi kasutaja või väline süsteem (õige)

3. Väike-ettevõtte laoseisu jälgimise süsteemi kasutusjuhud on järgmised:

- toote lisamine
- toote kustutamine
- toote muutmine

Kas kõik infosüsteemi põhitegevused on sooritatavad?

Vastus: ei - tooteinfo vaatamine, toodete nimekirja vaatamine puudu CRUD-ist

Miks on/ei ole?

Millised on põhitegevused infoga?

Vastus: C-create, R-read, U-update, D-delete.

2.3.2 Klassiskeemid

Klassiskeemide kirjeldamisel on peamise allikana kasutatud Martin Fowleri raamatut UML Distilled. Klassiskeemide tähistused on väga rikkalikud ning käesoleva töö autori arvates on UML Distilled üks parimaid allikaid klassiskeemidega tutvumiseks, kuna raamatus on eraldi peatükis selgelt ja lihtsalt on välja toodud enimoluline, mida läheb reaalselt vaja 90% juhtudest. [FOWLER]

Klassiskeem kirjeldab süsteemis olevate objektide tüüpe ning mitmesuguseid nendevahelisi staatilisi seoseid. Staatilisi seoseid on kahte põhitüüpi: kooslused *(association)* - näiteks klient võib laenutada mingi hulga raamatuid; alamtüübid *(subtypes)* - näiteks poemüüja on mingit tüüpi inimene. Klassiskeemid näitavad ka klassi atribuute, operatsioone ning tõkkeid *(constraints)*. Klassiskeemi üheks eesmärgiks on defineerida alus teistele skeemidele, kus väljendatakse süsteemi muid aspekte - objektide seisundeid ja objektide koostööd *(collaboration)* väljendatakse dünaamika diagrammidega. Klassiskeemi klassi saab otseselt realiseerida objekt-orienteeritud programmeerimiskeeles (nt. Java,C++, ...), mis toetab klassi konstruktsiooni.

2.3.2.1 Kuidas leida klasse

Klasside ülesleidmisel võib abiks olla objektide üldine liigitus:

- Käegakatsutavad (füüsilised) objektid: isik, auto, maja, jne.
- Rollid: tudeng, õppejõud
- Sündmusobjektid (ühe ajaparameetriga): eksam (algusaeg)
- Tegevusobjektid (kahe või enama ajaparameetriga): õppimine (algus, lõpp), õpetamine (algus, lõpp)
- Spetsifikatsioonid: õppeaine

Klasside ülesleidmisel võivad olla abiks standardsed küsimused:

- Millist informatsiooni (asjad, mõisted, sündmused, tegevused) modelleeritavast süsteemist on tarvis salvestada või analüüsida?
- Millised on välissüsteemid, millega modelleeritav süsteem suhtleb? Millised klassid võiksid neid välissüsteeme esindada modelleeritavas süsteemis ?
- Kas on võimalik kasutada mudeleid, lahendusi või komponente varasematest

projektidest, kolleegide töödest, muudelt tootjatelt? Milliseid klasse sealt võiks kasutada?

- Milliseid seadmeid antud süsteem peab käsitlema? Iga süsteemiga ühendatud tehniline seade võib anda klassi, mis seda seadet käsitleb. Kas on olemas organisatsiooniüksusi? Organisatsioon esitatakse klasside kaudu, eriti ärimudelites.
- Milliseid rolle ärisubjektid täidavad? Rolle võib vaadata klassidena: klient, töötaja, kasutaja, jne.

Kõige enam toetavad klasside leidmist teised skeemid nagu näiteks kasutusloo- ning dünaamikaskeemid. Ilma süsteemi eesmärke ja funktsionaalsust modelleerimata pole võimalik otsustada konkreetsete klasside vajalikkuse üle.
[ROOST]

2.3.2.2 Vaated *(perspectives)*

Perspektiivist arusaamine on oluline nii klassiskeemide joonistamisel kui nende lugemisel. Iga skeem tuleb joonistada ühestainsast selgest perspektiivist lähtuvalt. Ka skeemi lugemisel tuleb eelnevalt kindlaks teha, mis perspektiivist lähtuvalt see on loodud, kuna vastasel juhul võidakse skeemi valesti tõlgendada.
Perspektiiv ei ole osa formaalsest UML-ist, kuid võib modelleerimisel ja mudelite vaatlemisel olla väga kasulik.
Klassiskeeme joonistatakse kolmest vaatest lähtuvalt.

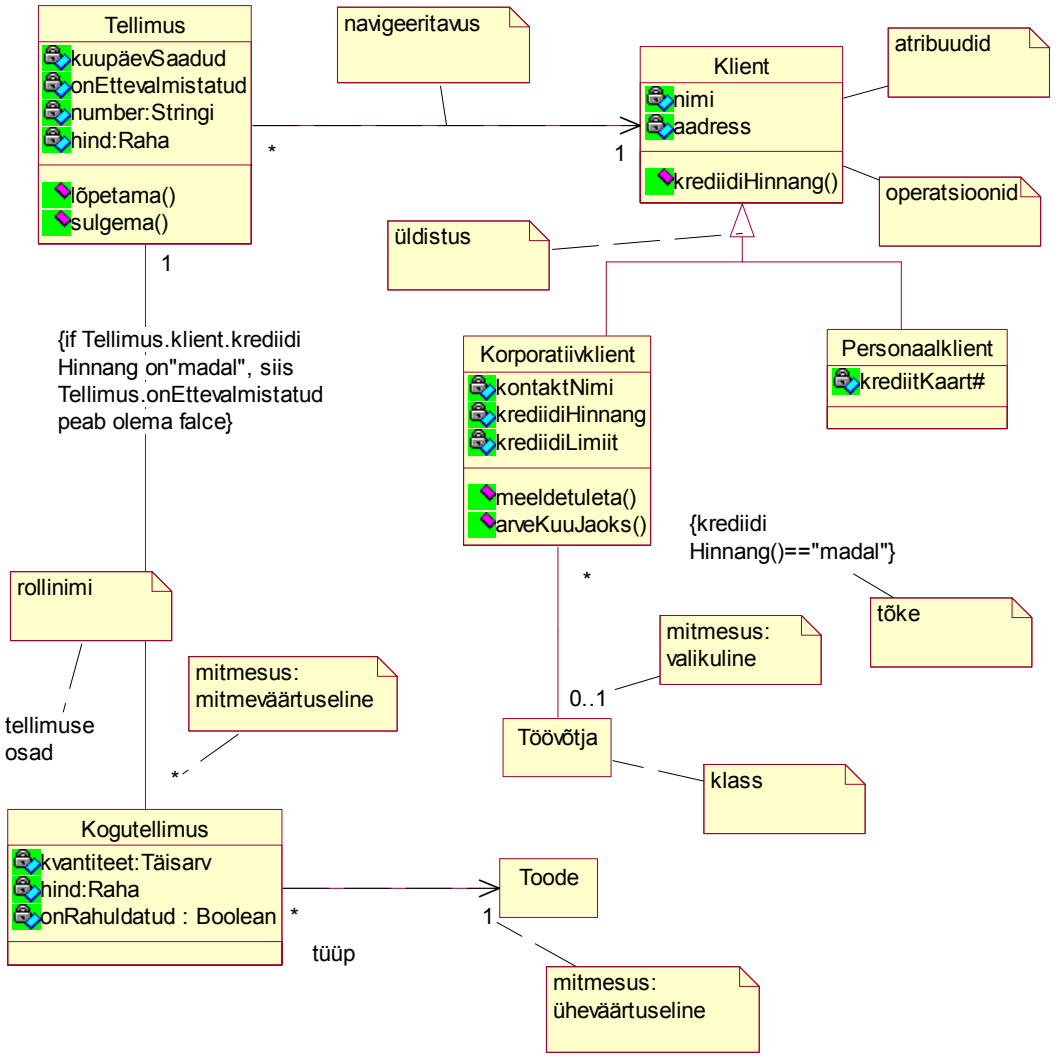
- Kontseptuaalne *(conceptual)*. Skeem, mis esindab vaatluse all oleva valdkonna mõisteid. Need mõisted saavad loomulikult ühenduses olema neid teostavate klassidega, kuid sageli ei toimu otsest seostamist. Kontseptuaalse mudeli joonistamisel tuleks tarkvarale, mis seda teostada võiks, väga vähe või üldse mitte tähelepanu pöörata. Nii et kontseptuaalskeemi võib pidada keelest sõltumatuks.

- Spetsifikatsioon *(specification)*. Siin vaadeldakse tarkvara liideseid *(interfaces)*, kuid mitte tarkvara teostust. Seega vaadeldakse pigem tüüpe kui klasse. Objekt-orienteeritud arendus asetab tugevat rõhku erinevusele liidese ja teostuse vahel, kuid sageli ei saa tegelikkuses sellest nii konkreetselt lähtuda, kuna klassi mõiste objekt-orienteeritud keeles kombineerib nii liidest kui teostust. Tihti viidatakse liidestele kui tüüpidele ja

nende liideste teostusele kui klassidele. Tüüp esindab liidest, millel võib olla mitmeid teostusi.

• Teostus *(implementation)*. Selles vaates on tõesti klassid ning teostamisel barjääri pole. See on arvatavasti enimkasutatav perspektiiv, kuigi tihti oleks parem valida spetsifikatsiooni perspektiiv. [FOWLER, lk.55-57]

2.3.2.3 Klassiskeemi elemendid



Joonis 4: Klassiskeem [FOWLER, lk. 61]

Kooslused *(associations)*

Kooslused esindavad seoseid klassi eksemplaride vahel (inimene töötab firma jaoks; firmal on hulk kontoreid jmt.).

Kontseptuaalsest perspektiivist esindavad kooslused kontseptuaalseid seoseid klasside vahel. Joonisel 4 võib näiteks näha, et tellimus peab tulema üheltainsalt kliendilt ja et üks klient esitab aja jooksul mitmeid tellimusi.

Igal kooslusel on kaks rolli *(role)*. Roll on olemi spetsiifiline nimega varustatud käitumine konkreetses kontekstis. Roll esindab antud kontekstis osaleva mudelielemendi käitumist või "nägu". Igal rollil on oma suund. Seega, kooslus kliendi ja tellimuse vahel sisaldab kahte rolli: ühte kliendist tellimusse; teist tellimusest klienti.

Joonisel 4 on rollile suunas tellimusest tellimuse sisusse nimeks antud tellimuse osad. Kui rolli nime pole skeemile kirja pandud, siis nimetatakse rolli sihtklassi järgi - seega rolli tellimusest klienti nimetatakse kliendiks.

Rollil on ka mitmesus *(multiplicity)*, mis näitab, kui palju objekte võib osaleda antud seoses. Joonisel 4 "*" kliendi ja tellimuse vahel näitab, et kliendiga võib olla seotud palju tellimusi; 1 näitab, et tellimus tuleb ainult ühelt kliendilt.

Enamasti on kõige tavalisemad seosehulga mitmesused 1, *, ja 0..1 (saab olla kas mitte midagi või üks). Üldisema mitmesuse kohta võib olla üksainus number (näiteks 11 tantsijat trupis), vahemik (näiteks 2-4 täringumängu "ümbermaailma reis" mängijat) või lõplikud kombinatsioonid arvudest ja vahemikest (näiteks 2, 4 auto ust).

Spetsifikatsiooni perspektiivist esindavad kooslused kohustusi.

Näiteks joonisel 4 on näha, et kliendiga on seotud üks või enam meetodit, mis ütleb, milliseid tellimusi antud klient on esitanud. Tellimuse meetodid näitavad, milline klient

antud tellimuse esitas ja millistest osadest see tellimus koosneb. Siiski ei saa spetsifikatsioonitasandi skeemi põhjal oletusi teha klasside andmestruktuuride kohta. Näiteks ei saa sellest perspektiivist lähtuvalt järeldada, kas tellimuse klass tegelikult sisaldab viita kliendile või mitte. Antud perspektiivist näitab skeem vaid liidest.

Kui skeemi vaadelda teostusmudelina, siis näitaks see, et viidad on seotud klasside vahel mõlemas suunas. Skeemilt võiks välja lugeda, et tellimusel on väli, mis on viitade kogu kogutellimusse ning samuti viit kliendile. Teostuse perspektiivist vaadatuna ei saa kooslusest midagi järeldada liidese kohta. Selle info annaksid klassi operatsioonid.

Kui kooslus on navigeeritav *(navigability)*, siis tähendab see seda, et ühe otsa objekti juurest võib otse ja lihtsalt jõuda teise otsa objekti juurde. Joonisel 4 on navigeeritavus tähistatud joontele lisatud nooleotstega.

Spetsifikatsioonimudeli puhul tähendaks see seda, et tellimusel on kohustus öelda, millise kliendi tellimus ta on, kuid kliendil ei ole võimalust öelda, millised tellimused tal on. Sümmeetriliste kohustuste asemel on nüüd kohustus ainult ühes joone otsas.

Teostusskeemil tähistaks see seda, et tellimus sisaldab viita kliendile, kuid klient tellimusele viita ei sisaldaks.

On mitmeid viise koosluste nimetamiseks. Enamasti objektmodelleerijad kasutavad ühele või teisele rollile nime andmiseks nimisõnu, kuna see vastab paremini kohustustele ja operatsioonidele. Kooslusele on mõtet nimi anda vaid siis, kui see aitab paremini skeemi mõista. Näiteks koosluse nimedel "on" ja "on seotud" ei ole mõtet. Kui rollil ei ole nime, siis võib rolli nimeks pidada sihtklassi nime. [FOWLER, lk. 56-62]

Atribuudid

Atribuut on klassi nimega omadus, mis kirjeldab klassi eksemplaride väärtuste vahemikku. Süsteemi eesmärk ja funktsionaalsus mõjutavad klassi kirjeldavaid atribuute. Kirjeldatakse ainult modelleeritava süsteemi kontekstis huvi pakkuvaid atribuute.

Atribuudil on tüüp:

- Primitiivsed tüübid: integer, Boolean, real, area
- Spetsiifilised (programmeerimiskeele jaoks)
- Teised klassid võivad olla atribuudi tüübiks

Atribuudi nähtavus *(visibility)*:

- Public (+): nähtav ja kasutatav väljastpoolt antud klassi
- Private (-): nähtav ja kasutatav ainult antud klassi sees
- Protected (#): kasutatakse koos üldistamise/pärimise seosega

Atribuudil võib olla vaikimisi väärtus.
Saab defineerida klassi skoobiga atribuute (muutujaid), mida jagavad kõik antud klassi objektid. Sellised atribuudid on alla joonitud.

Kontseptuaalsel tasandil kliendi nime atribuut tähistab, et klientidel on nimed ning spetsifikatsiooni tasandil tähistab, et kliendi objekt võib öelda oma nime ja omab mingit võimalust nime andmiseks. Teostustasandil kliendil on väli *(field)* nime jaoks.

Olenevalt skeemi detailsusest, võib atribuudi tähistus näidata atribuudi nime, tüüpi ja vaikeväärtust.
Atribuudi formaalne süntaks:
visibility name : type-expression = initial-value { property-string }
[ROOST]

Joonis 5: Atribuudid klassiskeemil

Koodinäide 1

```
    public class Auto

    {
   private int kiirus;
   private String värv;
   private int käik;
   private String seisund;
}
```

Operatsioonid

Operatsioon kirjeldatakse tulemustüübi *(return-type)*, nime ja 0 või enama parameetriga (parameetritele saab kirjeldada ka vaikimisi väärtusi, s.t. kui väljakutsuja ei väärtusta parameetrit, kasutatakse vaikimisi väärtust).Tulemustüüp, nimi ja parameetrid üheskoos moodustavad operatsiooni signatuuri *(signature)*. Signatuur kirjeldab kõike, mis on vajalik operatsiooni kasutamiseks.

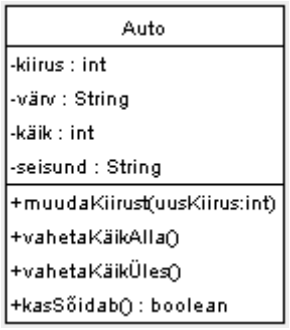
Klassi operatsioonid näitavad, mida see klass "oskab" teha, milliseid teenuseid ta osutab, seega võib operatsioone vaadata klassi liidesena. Sarnaselt atribuutidele võib operatsiooni jaoks kirjeldada nähtavuse ja kehtivuspiirkonna.

Konseptuaalsete mudelite siseselt ei peaks operatsioonid püüdma täpsustada klassi liidest. Selle asemel peaksid nad tähistama klassi põhilisi kohustusi. Kohustus on kõrgtaseme kirjeldus klassi eesmärgist.

Operatsiooni realisatsiooni nimetatakse meetodiks. Spetsifikatsiooni tasandil, vastavad operatsioonid tüübi avalikele meetoditele. Tavaliselt ei näidata neid operatsioone, mis lihtsalt käsitsevad atribuute, kuna neid saab enamasti tuletada. Küll aga võib olla vajadus

viidata, kas antud atribuut on kirjutuskaitstud või muutumatu (st. tema väärtus ei muutu kunagi). Teostusmudelil võidakse näidata ka privaatseid ja kaitstud operatsioone.
Täielik UML süntaks operatsioonide jaoks on
visibility name (parameter-list): return-type-expression{property-string}
kus
- *visibility* on public (+), protected (#), või private (-)
- name on string
- *parameter-list* sisaldab (valikulisi) argumente, mille süntaks on sama mis atribuutidel
- *return-type-expression* on valikuline, keelest sõltuv spetsifikatsioon
- *property-string* viitab omaduse väärtustele, mis kehtivad antud väärtuse kohta
[ROOST]
Tihti on kasulik vahet teha operatsioonide vahel, mis muudavad klassi olekut ja nende vahel, mis ei muuda. Päring *(query)* on operatsioon, mis saab klassilt väärtuse, muutmata seejuures klassi vaadeldavat olekut. Objekti vaadeldavat olekut *(observable state)*saab määratleda sellega seotud päringutest. Operatsioone, mis muudavad vaadeldavat objekti olekut, kutsutakse modifikaatoriteks *(modifiers)*. [LARMAN]

On mõttekas teha selgeks erinevus päringute ja modifikaatorite vahel. Päringuid saab välja kutsuda ükskõik mis järjekorras, kuid modifikaatorite järgnevus on oluline. Et hoida neid eraldi, on hea vältida modifikaatoritest väärtuste väljastamist.



Joonis 6: Operatsioonid klassiskeemil

Koodinäide 2

```
public class Auto

{
   private int kiirus;
   private String värv;
   private int käik;
   private String seisund;

   public void muudaKiirust(int uusKiirus) {
      this.kiirus = uusKiirus;
   }

   public void vahetaKäikAlla() {
      if (käik > 1)
         käik--;
   }
   public void vahetaKäikÜles() {
      if (käik < 5)
         käik++;
   }
   public boolean kasSõidab() {
      if (seisund.equals("Sõitmine"))
         return true;
      else
         return false;
   }

}
```

Üldistamine (generalization)

Üldistus on seos üldisema ja spetsiifilisema elemendi vahel. Spetsiifilisem element on täielikult kooskõlas üldisema elemendiga, kuid sisaldab täiendavat informatsiooni. Kohas, kus üldisem element on lubatud, võib kasutada spetsiifilisema elemendi eksemplari. [SONASTIK]

Tüüpilise näitena üldistamisest võib tuua talitluse personaal- ja korporatiivklientide kohta. Neil on erinevusi, kuid ka palju sarnasusi. Sarnasused võib panna üldisesse Kliendi klassi (ülemtüüp), milles PersonaalKlient ja KorporatiivKlient on alamtüübid.

Eri modelleerimistasanditel interpreteeritakse seda erinevalt. Kontseptuaalselt näiteks võib öelda, et KorporatiivKlient on Kliendi alamtüüp, kui seda on definitsiooni järgi ka kõik KorporatiivKliendi eksemplarid. Võtmeideeks on, et kõik mida öeldakse kliendi kohta (kooslused, atribuudid, operatsioonid) kehtib ka KorporatiivKliendi kohta.

Spetsifikatsioonimudelil tähendab üldistamine seda, et alamtüübi liides peab sisaldama kõiki elemente ülemtüübist st. alamtüüp ühildub *(conform)* ülemtüübi liidesega. Seda võiks nimetada ka asendatavuse *(substitutability)* printsiibiks. See tähendab, et kui ükskõik millises koodis, mis sisaldab klienti asendada see korporatiivkliendiga, töötab programm sama korralikult kui enne. Teiste sõnadega, koodi kirjutades saab seal, kus on olemas klient, vabalt kasutada ükskõik millist kliendi alamtüüpi.

Teostuse perspektiivist on üldistamine seotud pärimisega programmeerimiskeeltes. Alamklass pärib kõik ülemklassi meetodid ja väljad ning sellel võib olla meetodeid lisaks.

Võtmesisuks on siin üldistamise erinevus spetsifikatsiooni perspektiivist (alamtüübid või liides-pärimine) ja teostuse perspektiivist (alamklassid või teostus-pärimine). Alamklassideks jaotamine on üks viise alamtüüpide teostamiseks.

Ükskõik kumba neist üldistamise vormidest kasutada, peaks alati kindlaks tegema, et kehtib ka kontseptuaalne üldistamine. Seda tuleb teha kohe, sest kui hiljem muudatusi teha, siis üldistamine ei ole stabiilne.

Mõnikord on juhtumeid, kus alamtüübil on samasugune liides, nagu tema ülemtüübil, kuid see alamtüüp kasutab operatsioone erineval moel. Sellisel juhul ei pea näitama alamtüüpi spetsifikatsiooni perspektiiviga skeemil, aga olenevalt olukorrast võib.

Tõkked

Koosluste, atribuutide ja üldistuste kaudu saavad tõkked juba suures osas ära kirjeldatud, kuid mõnikord sellest ei piisa ja on tarvis veel tõkkeid lisaks defineerida. Klassiskeem on sobilik tõkete kirjeldamiseks. UML-is kasutatakse ainsa süntaksina tõkete kirjeldamisel loogelisi sulge ({}).

Joonis 4 näitab, et korporatiivklientidel on krediidilimiidid, kui personaalklientidel pole. Jooniselt võib välja lugeda ka selliseid tõkkeid, mida eraldi loogelistes sulgudes tähistada pole tarvis. Näiteks tellimuse võib esitada vaid üks klient. Samuti võib skeemilt lugeda, et igast tellimuse osast mõeldakse eraldi: öeldakse näiteks, et 10 kollast palli, 10 sinist palli ja 10 punast palli, mitte 10 kollast, sinist ja punast palli.

2.3.2.4 Millal kasutada klassiskeeme

Kuna klassiskeemid on selgrooks peaaegu kõigile objekt-orienteeritud meetoditele, kasutatakse neid kogu aeg.

Klassiskeemide probleemiks on, et nad on nii rikkalikud, et nende kasutamisega võib liiale minna.

- Ei tasu püüda kasutada kogu märgistust, mis on kättesaadav. Tuleks alusta lihtsamast: klassid, kooslused, atribuudid, ja üldistamine. Muid tähistusi tuleks hakata kasutama vaid siis, kui seda vajatakse.
- Perspektiiv, millest lähtuvalt mudeleid joonistada, peaks olema vastavuses projekti järguga.

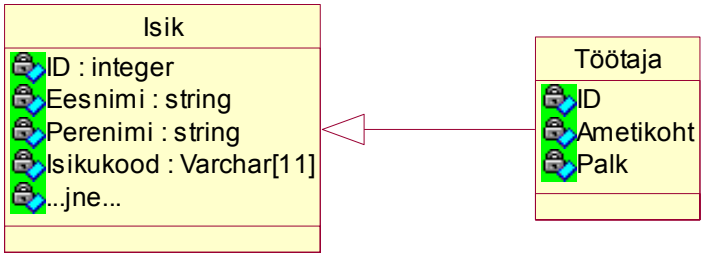
- Analüüsi faasis tuleks joonista kontseptuaalmudeleid.
- Tegeldes tarkvaraga, tuleks luua spetsifikatsioonimudeleid.
- Teostusmudeleid tuleks joonistada vaid siis kui tähistatakse mingit kindlat teostustehnikat.

• Mudeleid ei tasu joonistada kõige jaoks; selle asemel tuleks keskenduda võtmealadele.

Parem on vähe skeeme, mida kasutatakse ja uuendatakse, kui palju unustatud, kasutuid mudeleid. [FOWLER, lk 73-74]

Ülesanded 2:

1.



Kas ülalkujutatud joonisel on olemite 'Isik' ja 'Töötaja' vahel oleva seose tüübiks

- üks-mitmele seos
- mitu-mitmele seos
- üldistusseos (**õige**)
- agregatsiooniseos

2. Kas klassiskeemi objektiks võivad olla muuhulgas?

- sündmused, tegevused
- rollid ettevõttes
- mõlemad (**õige**)
- ei kumbki

3. Miks ei näidata kontseptuaalses klassiskeemis välisvõtmeid?

Vastus: Kuna kontseptuaalne klassiskeem näitab klasse ja nende suhteid organisatsiooni / IS seisukohalt. Välisvõtmete näitamine skeemis on aga realisatsiooni küsimus (relatsiooniline andmebaas).

Välisvõtmed võib näidata spetsifikatsiooni skeemis.

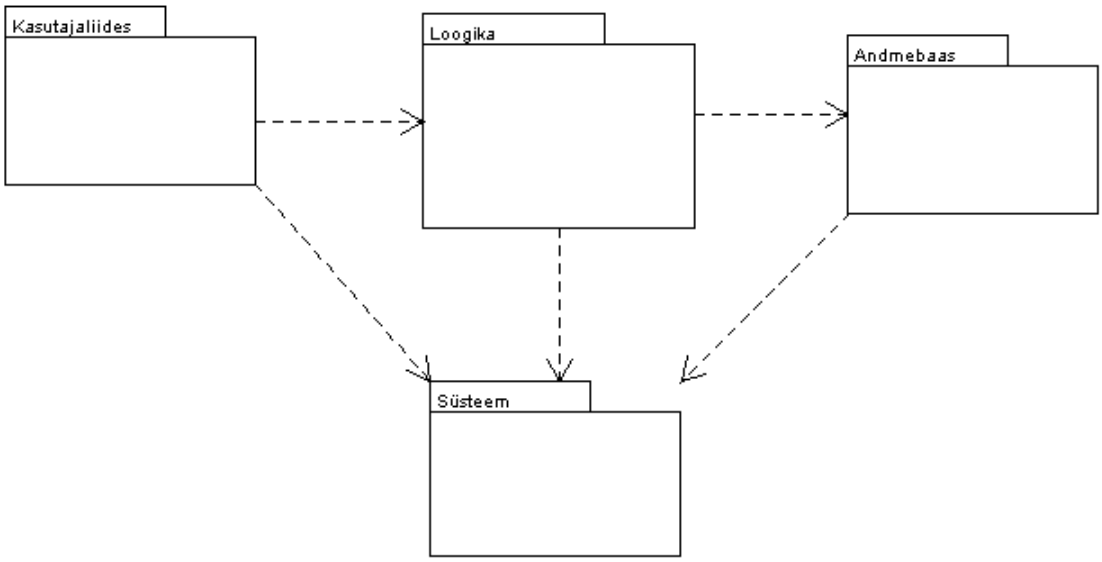
2.3.3 Paketiskeemid

Paketid on vajalikud vahendid suurte projektide jaoks. Kui süsteemid muutuvad suureks, on neist raske aru saada, samuti neisse muudatusi sisse viia. Seega tekib vajadus suurte süsteemide jagamiseks väiksemateks süsteemideks. Üheks võimaluseks on kasutada grupeerimismehhanismi, mida UML-is kutsutaks paketiks. Pakette võiks kasutada siis, kui klassiskeem, mis hõlmab kogu süsteemi, ei mahu enam ära A4 formaadis paberilehele.
Mõiste "paketiskeem" on kasutusel skeemi kohta, mis näitab klasside pakette ja nendevahelisi sõltuvusi. Paketid ja sõltuvused on elemendid klassiskeemil, nii et pakettskeem on lihtsalt üks klassiskeemi vorme.

Pakett *(package)* on üldotstarbeline nimega mehhanism mudelielementide (nt klasside, kasutuslugude, skeemide ja teiste pakettide) grupeerimiseks. Klasside grupeerimisel tuleb olla leidlik, kuna vastasel juhul muutub grupeerimine suvaliseks. Pakett on puhtalt kontseptuaalne – käitusajal pole teda olemas. Kõiki mudelielemente, mida pakett omab või millele ta viitab, nimetatakse paketi sisuks. Paketi eksemplar ei oma mõtet.

Paketid ei paku vastuseid kuidas vähendada süsteemis sõltuvusi, vaid aitavad näha, millised need sõltuvused on - ja töötada selle heaks, et sõltuvusi vähendada, saab ainult siis, kui neid on võimalik näha. Pakettskeemid on võtmevahendiks säilitamaks kontrolli süsteemi üleüldise struktuuri üle.

Paketid on eriti kasulikud testimiseks. Kuigi mõningaid teste võib kirjutada klass klassi baasil, on eelistatav teha üksuse testi pakett paketi baasil. Igal paketil peaks olema üks või enam testklassi, mis testivad paketi käitumist.



Joonis 7: Paketiskeem

Ülesanded 3

1. Milline on pakettskeemi põhifunktsioon?
    - grupeerida ühe skeemi elemente mingi kriteeriumi alusel (**õige**)
    - näidata sõltuvusi süsteemis
    - grupeerida erinevate skeemide elemente, ühendamaks neid

2. Millised järgnevatest kuuluvad klassikaliselt paketi koosseisu?
    - andmeklassid (**õige**)
    - andmeobjekti olekud
    - tegutsejad infosüsteemis

3. Mõelge, millisteks pakettideks oleks väikefirma infosüsteemi klassimudel otstarbekas jagada?

Arutage, milline pakettide jaotus võiks väljendada väikefirma (rõivapood, saekaater, raamatupood) tervikliku infosüsteemi klassimudelit kõige edukamalt?

Subjektiivse näitena .. paketid:

- Töötajate arvestus – töötajatega seotud olemid
- Müügi arvestus – müügilepingud, arved
- Materjalide arvestus – tooted, mida müüakse, laoseis
- Varade arvestus – põhivarade müük, remont, amortisatsioon, ostuarved
- Klientide arvestus – kliendibaas, püsikliendid

Pakette võiks olla näiteks 3..6. Või omal valikul (kasvõi 20).

2.3.4 Interaktsiooniskeemid

Interaktsiooniskeeme võib kõrvuti olekuskeemide ja tegevusskeemidega nimetada käitumuslikeks skeemideks. Interaktsiooniskeemide alla võib omakorda liigitada koostööskeemid ja jadaskeemid.

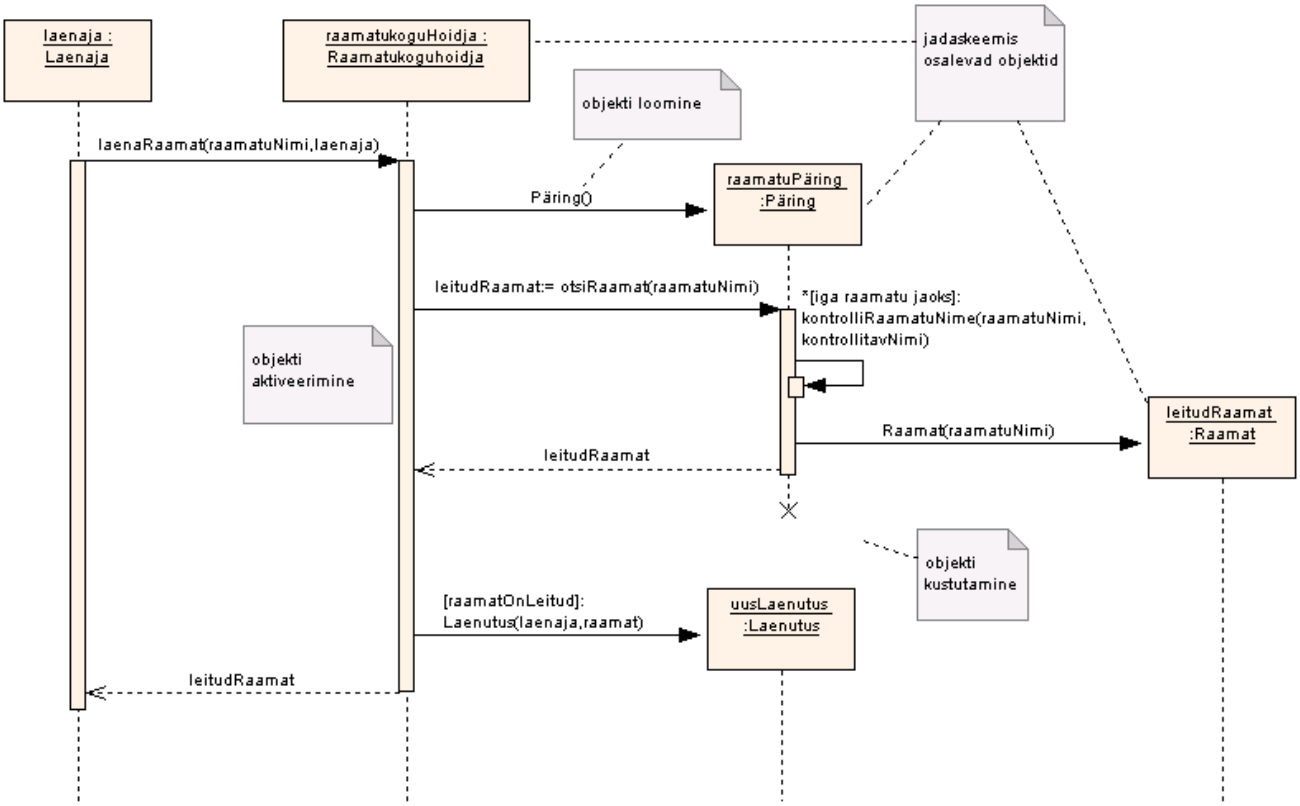
Interaktsiooniskeemid on mudelid, mis kirjeldavad, kuidas toimub objektide koostöö mingi käitumise *(behavior)* puhul. Enamasti hõlmab interaktsiooniskeem üheainsa kasutusloo käitumise. Skeem näitab objekte, ning sõnumeid, mis nende objektide vahel kasutusloo siseselt vahetatakse.

Interaktsiooniskeeme peaks kasutama mitmesuguste objektide käitumise vaatlemiseks ühe kasutusloo piires. Kui on soov vaadelda üheainsa objekti käitumist paljudes kasutuslugudes, võiks kasutada olekuskeemi. Interaktsiooniskeemid sobivad objektidevahelise koostöö näitamiseks, kuid nad ei defineeri seda käitumist põhjalikult.

2.3.4.1 Jadaskeemid

Jadaskeem võib olla üldkujul (kirjeldab kõiki võimalikke stsenaariume) või eksemplarkujul (kirjeldab ühte tegelikku stsenaariumi). Jadaskeem keskendub interaktsioonis olevate objektide ja nende vahel vahetatavate sõnumite ajalisele järjestamisele, kuid ei näita objektide vahelisi seoseid.

2.3.4.1.1 Tähistused jadaskeemidel



Joonis 8: Jadaskeem

Objekte kujutatakse jadaskeemil kastidena vertikaalsete punktiirjoonte tippudes. Vertikaalset punktiirjoont kutsutakse objekti elujooneks *(life-line)*. Elujoon kujutab objekti olemasolu teatud ajavahemikus.

Sõnumid *(messages)*

Objektid saadavad üksteisele kasutusloo täitmiseks sõnumeid. Iga sõnum on sildistatud vähemalt sõnumi nimega, kuid võib sisaldada ka argumente ja juhtinformatsiooni *(control information)* ning võib näidata endalesuunamist *(self-delegation)*. Endalesuunamine on sõnum, mille objekt saadab iseendale, saates sõnuminoole tagasi samasse elujoonde (vt. joonis 8). [FOWLER, lk 104-105]

Iga sõnumit esindab nool kahe elujoone vahel. Sõnumite järjestus kujutatakse skeemil ülevalt alla kulgevatena. Sõnum joonistatakse nooleotsaga joonena sõnumi saatja ja vastuvõtja vahel, kusjuures noole tüüp näitab sõnumi tüüpi:

- Lihtne: esitab lihtsalt juhtimisvoogu. Näitab, kuidas juhtimine edastatakse ühelt objektilt teisele ilma kommunikatsiooni detaile kirjeldamata. Kasutatakse siis, kui kommunikatsiooni detailid pole teada või neid peetakse ebaolulisteks antud diagrammis. Samuti kasutatakse sünkroonse sõnumi tagastamise näitamiseks, mis joonistatakse sõnumit käsitlevast objektist tagasi väljakutsujale näitamaks, et juhtimine antakse tagasi.
- Sünkroonne: hierarhiline juhtimisvoog, tüüpiliselt realiseeritud operatsiooni väljakutsena. Sõnumit käsitlev operatsioon viiakse lõpule (kaasa arvatud kõik teised sõnumid, mis saadetakse käsitlemise osana), enne kui väljakutsuja saab jätkata täitmist. Tagasipöördumist saab näidata lihtsa sõnumina või vaikimisi, kui sõnum on käsitletud.
- Asünkroonne: asünkroonne juhtimisvoog. Poolikud nooleotsad näitavad asünkroonseid *(asynchronous)* sõnumeid. Asünkroonne sõnum ei blokeeri väljakutsujat, nii et see saab jätkata töötlusega. Asünkroonne teade võib teha ühte kolmest asjast.
    1. Luua uue lõime, mis ühendub aktivatsiooni külge
    2. Luua uue objekti
    3. Suhelda lõimega, mis juba töötab

Lihtsat ja sünkroonset sõnumit saab kombineerida üheks sõnumijooneks koos sünkroonse sõnumi noolega ühes otsas ning lihtsa tagastamisnoolega teises otsas. See tähendab, et tagasipöördumine toimub peaaegu koheselt pärast operatsiooni väljakutset.

Juhtinformatsioon

Väärtuslikud on kaks juhtimisinformatsiooni. Esiteks saab defineerida igale sõnumile tingimuse (condition). Tingimus tähistab, millal sõnum on saadetud (nt., [raamatOnLeitud]). See sõnum saadetakse vaid siis, kui tingimus on tõene. Teine juhtmarker on iteratsiooni marker *(iteration marker)*, mis näitab, et sõnum on saadetud

palju kordi mitmetele vastuvõtjatele. Iteratsiooni saab tähistada "*" sümboliga ning olemuse saab ära näidata kandiliste sulgude sees (nagu näiteks *[iga raamtu jaoks]). Tagasipöördumisi *(return)* kasutatakse skeemil vaid juhul, kui see aitab skeemi arusaadavamaks muuta ning alati ei ole kõiki mõtet skeemile joonistada.

Aktiveerimine *(activation)* ja paralleelsed protsessid *(concurrent process)*

Jadaskeemide üheks väärtuslikuks omaduseks on ka see, et nad võimaldavad paralleelsete protsesside kirjeldamist.

Aktiveerimist tähistab skeemil laiendatud elujoon. Meetod on aktiivne, kui ta kas täidab alamoperatsiooni või ootab alamoperatsioonist tagasipöördumist. Alati ei pruugi aktiveerimised protseduurilisele täitmisele palju juurde anda. Siiski oleks aktiveerimisi ka mõnikord protseduurilise interaktsiooni puhul otstarbekas kasutada, kuna see aitab näidata palju selgemalt endalesuunamist. Aktiveerimised ja kuhitähistus *(stack notation)* aitavad selgitada, kas edasised väljakutsed toimuvad pärast endalesuunamist (nii välja kutsuva kui ka välja kutsutava meetodi puhul).

Koodinäide 3

```
public class Laenaja
{
	private String nimi;

	public Laenaja( String nimi )
	{
		this.nimi = nimi;
	}

	public String küsiNimi()
	{
		return this.nimi;
	}
```

```
}

public class Raamatukoguhoidja
{
	public Raamatukoguhoidja()
	{
	}

	public Raamat laenaRaamat( String raamatuNimi, Laenaja laenaja )
	{
		Päring päring = new Päring();
		Raamat leitudRaamat = päring.otsiRaamat( raamatuNimi );

		boolean raamatOnLeitud = (leitudRaamat != null);
		if (raamatOnLeitud)
			Laenutus uusLaenutus = new Laenutus( laenaja, leitudRaamat );

		return leitudRaamat;
	}

}

public class Päring
{

	String[] raamatuteNimed = new String[4];

	public Päring()
	{
		raamatuteNimed[0] = "Punamütsikese seiklused";
```

```
            raamatuteNimed[1] = "Tõde ja õigus";
            raamatuteNimed[2] = "UML Distilled";
            raamatuteNimed[3] = "Sõrmuste isand";
        }

        public Raamat otsiRaamat( String raamatuNimi )
        {
            Raamat leitudRaamat = null;

            for ( int i = 0; i < raamatuteNimed.length; i++ )
            {
                if ( kontrolliRaamatuNime( raamatuNimi, raamatuteNimed[i] )
                {
                    leitudRaamat = new Raamat( raamatuNimi );
                }
            }

            return leitudRaamat;
        }

        public boolean kontrolliRaamatuNime( String raamatuNimi, String
kontrollitavNimi )
        {
            if ( raamatuNimi.equals( kontrollitavNimi ) )
                return true;
            else
                return false;
        }

}
```

```
public class Laenutus
{

	private Raamat laenutatudRaamat;
	private Laenaja laenaja;

	public Laenutus( Laenaja laenaja, Raamat raamat )
	{
		this.laenutatudRaamat = raamat;
		this.laenaja = laenaja;
	}

}

public class Raamat
{

	private String raamatuNimi;

	public Raamat( String raamatuNimi )
	{
		this.raamatuNimi = raamatuNimi;
	}
```

Ülesanded 4

1. Milline on UML jadaskeemil ajatelg?
   - horisontaaltelg
   - vertikaaltelg (**õige –** elujooned - elujoon kujutab objekti olemasolu teatud ajavahemikus.)
   - diagonaaltelg
   - aeg ei puutu jadaskeemi olemusse

2.Mida kujutatakse jadaskeemi horisontaalteljel?

- objekte (**õige**)
- olekuid
- tegevusi

3. Kirjeldage, kuidas väljendab jadaskeem kasutuslugude käiku infosüsteemis?

Jadaskeemi objektide muutuse / loomise / arengu tingib kasutusjuhu käivitumine süsteemis – s.t. kasutusjuhud on need, mille toimumise tagajärjel saab jadaskeemil tegevus toimuda, kasutusjuhtude raames vahetatakse elujoonte vahel infot. Objektide ajalise muutuse tingivad süsteemi kasutusjuhud.

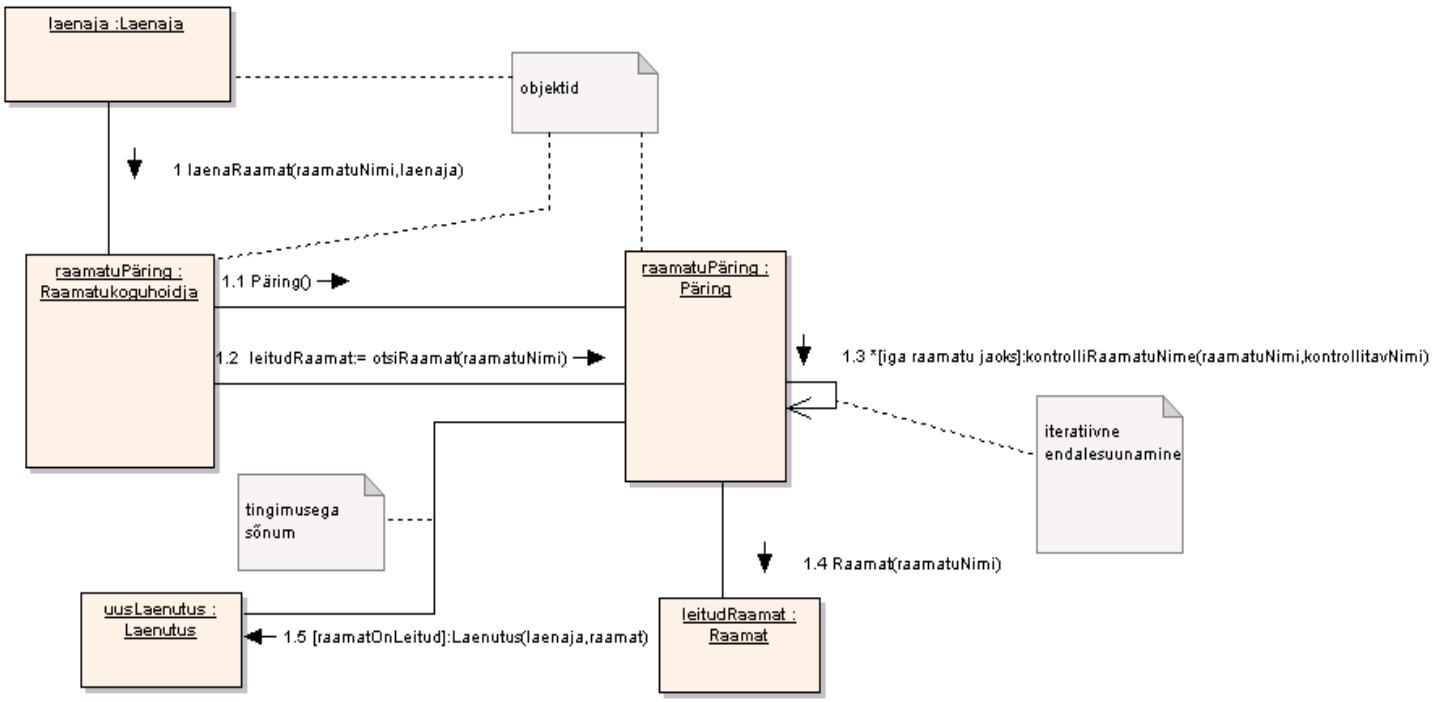
Kirjeldage, kuidas on jadaskeem seotud klassiskeemiga?

Jadaskeemi objektideks (horisontaaltelg) on klassid – jadaskeem näitab andmeklassi eksemplaride elutsüklilist arengut, nende eksistentsi ja muutumist infosüsteemi andmehoidlas.

2.3.4.2 Koostööskeem

Koostöökeem esitab eksemplaride ja nendevaheliste linkide ümber kujundatud interaktsioone. Erinevalt jadaskeemist näitab koostööskeem seoseid eksemplaride vahel.

2.3.4.2.1 Tähistused koostööskeemidel



Joonis 9: Kümnendnumeratsiooniga koostööskeem

Klasside ja eksemplaride tähistamine

UML-is on ühtne lähenemine tüüpide ja eksemplaride eristamiseks skeemidel:

Igasuguse UML-i elemendi korral (klass, aktor jne.) kasutab eksemplar samasugust graafilist sümbolit kui tüüp, kuid nimestring joonitakse alla. Seepärast näidatakse interaktsiooniskeemil objekti (klassi eksemplari) tavalise klassi sümboliga (ristkülik), kus nimi on alla joonitud. Välja võib jätta nii klassi kui objekti nime. Koostööskeemil peab klassi nimele alati eelnema koolon (:Laenaja).

Linkide tähistamine

Link *(link)* on ühendustee kahe eksemplari vahel, mis näitab navigatsiooni ja viidatavuse *(visibility)* vormi nende eksemplaride vahel. Link on koosluse eksemplar. Vaadates kahte eksemplari klient-server seoses, tähendab navigatsioonitee kliendist serverini, et klient saab saata sõnumeid serverile. Näiteks on link ehk navigatsioonitee raamatukoguhoidjast (raamatuPäring:Päring) raamatuni (uusLaenutus:Laenutus) koos sellel võimalike sõnumitega, näiteks [RaamatOnLeitud]:Laenutus(laenaja, raamat). Link tähistatakse joonega (vt. joonis 9).

Sõnumite tähistamine

Sõnumeid objektide vahel tähistatakse märgistatud noolega ühendusjoonel. Nagu jadaskeemilgi näitab noole tüüp sõnumi tüüpi (vt. peatükk 2.3.4.1 Sõnumid *(messages)*). Ühe lingiga võib olla seotud palju sõnumeid. Järgnevust näidatakse sõnumite nummerdamise abil. Koostööskeemil on võimalik kasutada kahte tüüpi numeratsiooni. Lihtsam numeratsioon näitab vaid sõnumite järgnevust ja üleüldise järgnevuse nägemine on seega lihtsam. Kuid soovitav on kasutada keerukamat kümnendnumeratsiooniga skeemi, kuna see võimaldab ka näidata, milline operatsioon kutsub välja millise teise operatsiooni. (vt. joonis 9)

Juhtinformatsioon

Koostööskeemile on võimalik lisada täpselt samasugust juhtinformatsiooni, kui jadaskeemilegi (vt. peatükk 2.3.4.1 Juhtinformatsioon).

Tingimuslik käitumine

Tingimusliku käitumise näitamise jaoks on võimalik märkida tingimused sõnumite ette. Teine võimalus on kasutada iga stsenaariumi jaoks eraldi skeeme. Soovitatav on kasutada eraldi skeeme, kuna nii jada- kui koostööskeemide puhul on nende üheks tugevaks küljeks just väljenduse lihtsus. Komplekssema käitumise puhul kaotavad nad kiiresti selguse. Kui on soov kirjeldada kompleksset käitumist ühelainsal skeemil, võiks selleks kasutada tegevusskeemi.

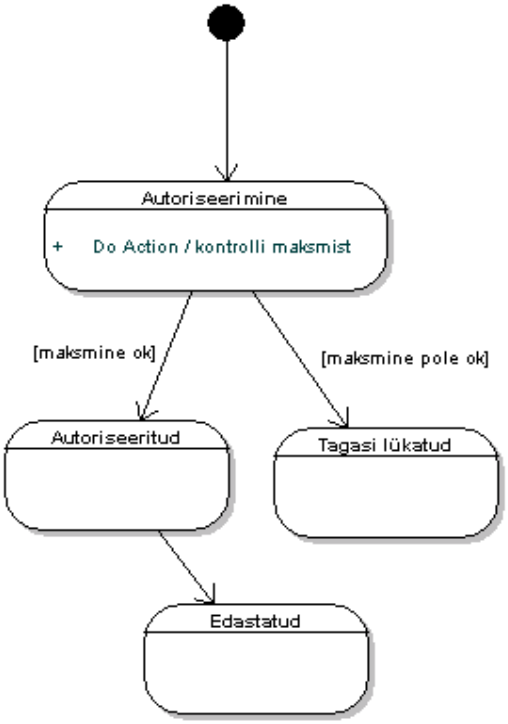
2.3.5 Olekuskeemid

Olekuskeemi kasutatakse klassi (või allsüsteemi või kogu süsteemi) sisemise käitumise ja olekute kirjeldamiseks. Olekuskeem keskendub sellele, kuidas objektid muudavad oma olekut ajas, sõltuvalt toimuvatest sündmustest, olekus teostatavast käitumisest ning tegevustest, ning ajast, millal sündmus toimub. Sündmuseks võib olla tingimuse tõeseks saamine, signaali vastuvõtt või operatsiooni väljakutse, või lihtsalt etteantud ajaperioodi möödumine. Olekuskeeme (statechart diagrams) võib samastada "klassikaliste" olekuskeemidega (state-transition diagrams).

2.3.5.1 Tähistused olekuskeemil

Olekud (states) ja siirded (transitions)

UML-is iseloomustab olek tingimust või olukorda objekti elutsükli jooksul, mis määrab objekti võime sooritada teatud tegevusi või rahuldada konkreetseid nõudeid. Olek on objekti omaduste teatud kombinatsioon, nagu näiteks olek=abielus. Objektid reageerivad sündmustele, seega sündmus “laulatus” viib objekti olekust “vallaline” olekusse “abielus”. Ebasobivas olekus saabuvad sündmused tähendavad viga (nt laulatus olekus “abielus”).



Joonis 10: Olekuskeem

Olekuskeem võib omada lähtepunkti (algolek) ning mitut lõpp-punkti (lõppolekud). Olekut näidatakse ümara kastiga, algolekut väikese täidetud ringiga, lõppolekut "härjasilma" sümboliga. Oleku siiret joonega, mille ühes otsas nool ülemineku suuna näitamiseks. Oleku siirded võivad olla märgistatud siiret põhjustava sündmusega. Sündmuse toimumisel viiakse oleku siire tegelikult läbi (siire käivitatakse).

Olek võib sisaldada kolme liiki osasid. Esimene osa näitab oleku nime, näiteks joonisel 10 kontrolliv, edastav. Teine osa on mittekohustuslik seisundimuutujate osa, kus võivad olla loetletud ning väärtustatud atribuudid (muutujad). Kolmas osa on mittekohustuslik tegevuse osa, kus võivad olla näidatud sündmused ja tegevused. Terminit "action/toiming" kasutatakse siirde kohta ja terminit "activity/tegevus" oleku kohta. Kuigi nad mõlemad on protsessid, mida tüüpiliselt teostavad teatud meetodid, käsitletakse neid erinevalt. Toimingud on seotud siiretega ja neid loetakse protsessideks, mis toimuvad

ruttu ja mida ei saa segada. Tegevused on seotud olekutega ja need võivad kauem aega võtta. Tegevusi võib mõni sündmus segada.

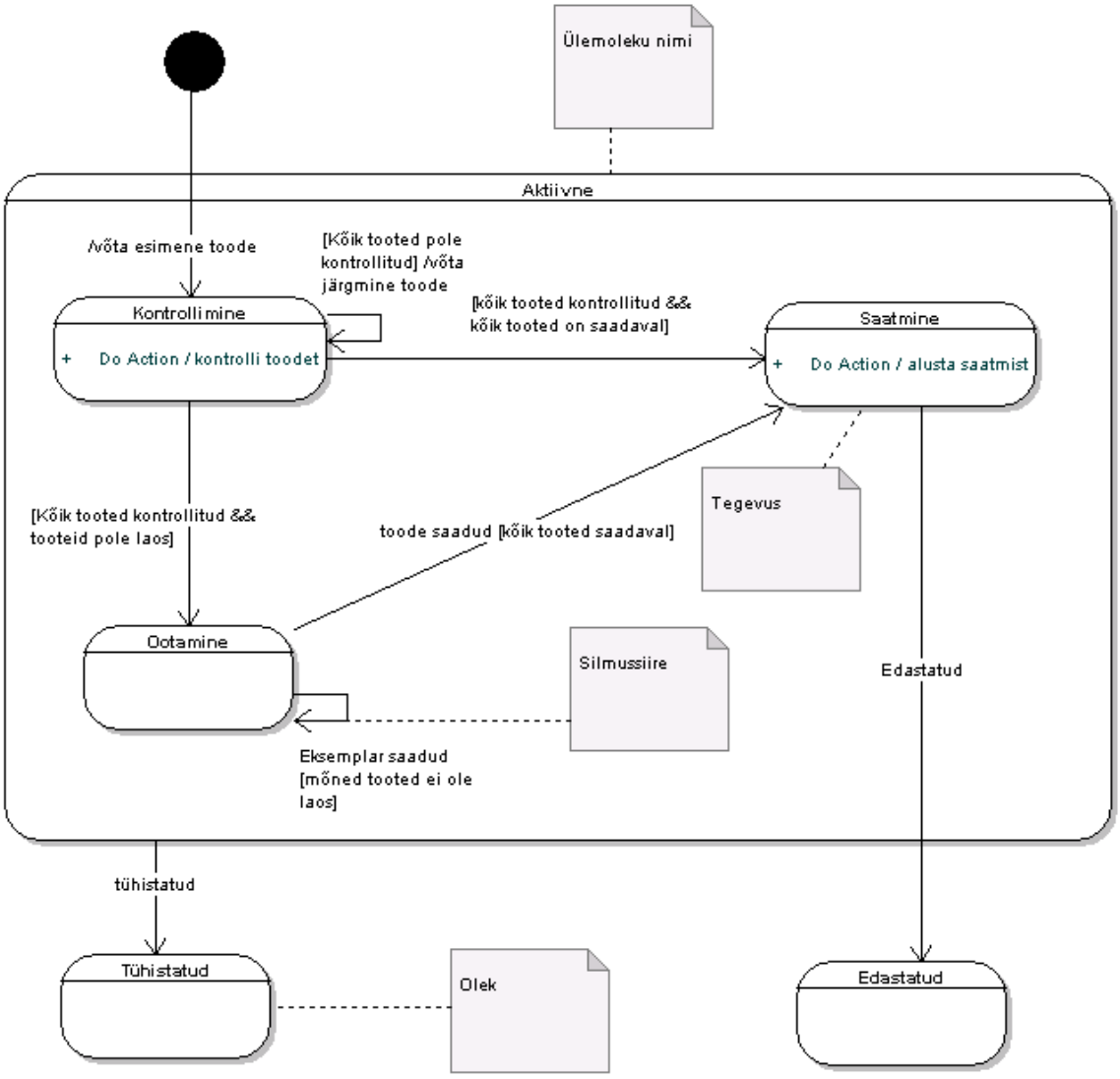
Kasutatakse kolme standardsündmust: *entry, exit, do:*

- Entry sündmust kasutatakse olekusse sisenemisel täidetavate toimingute kirjeldamiseks, näiteks atribuudi väärtustamine või sõnumi saatmine.
- Exit sündmust kasutatakse olekust väljumisel täidetavate toimingute kirjeldamisel.
- Do sündmust saab kasutada oleku kestel (kuni objekt on antud olekus) täidetavate tegevuste kirjeldamiseks (vt. joonis 10 nt. do/kontrolli maksmist).

Tegevuse osa formaalne süntaks: *event-name argument-list ‘ / ‘ action-expression*

Sündmuse nimi võib olla suvaline sündmus, kaasa arvatud standardsündmused. Tegevusavaldis näitab, milliseid tegevusi tuleb täita (operatsiooni väljakutsed, atribuutide väärtustamine, jne.). Sündmusel võivad olla argumendid. Standardsündmustel ei saa argumente olla.

Oleku siirdega on tavaliselt seotud sündmus, kuid mitte tingimata. Do-tegevus oleku sees võib olla pidev protsess, mida teostatakse senikaua, kui objekt on antud seisundis. Do-tegevuse võib katkestada väline sündmus, mis põhjustab antud oleku siirde järgmisesse olekusse. Kui oleku siire ei oma sündmust, siis lähteolek muutub automaatselt hetkel, kui temas kirjeldatud sisemised tegevused on täidetud (juhul kui seal on kirjeldatud entry, exit, do või kasutaja-defineeritud tegevusi).
Formaalne süntaks oleku siirde spetsifitseerimiseks:
event-signature ‘’[‘ guard-condition ‘]’ ‘/’ action-expression



Joonis 11: Ülemolekuga olekuskeem

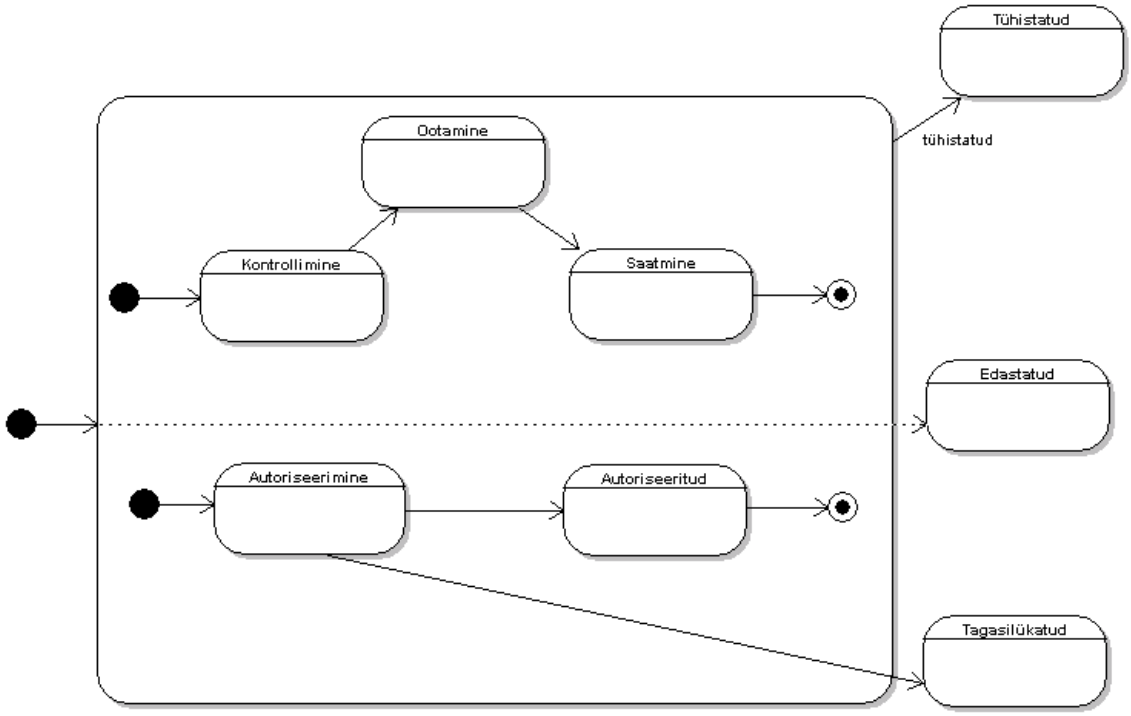
Ülemolekud

Kui mitut siiret teostavad mitu olekut, on üheks võimaluseks igast olekust eraldi siire joonistada. Kuid see vähendab skeemi lihtsat ja selget väljanägemist. Seega oleks kasulikuks alternatiiviks luua ülemolek kõigist kolmest olekust ja siis joonistada üksainus siire sellest. Alamolekud lihtsalt pärivad ükskõik millised siirded ülemolekust. Joonisel 11 on näha, et siiret nimega "tühistatud" saab ülemoleku kaudu teostada nii Kontrollimine, Ootamine kui ka Edastatud olekutest.

Üleminekutingimus (guard-condition)

Üleminekutingimus on kahendavaldis, mille väärtus peab enne siirde käivitamist olema tõene. (näiteks joonisel 11 [Kõik tooted pole kontrollitud]). Üleminekutingimuses esinevad nimed peavad olema vastava päästiku ja/või objekti atribuutide nimed, millele asjakohane olekumasin kuulub. Kui see tingimus on kombineeritud sündmuse signatuuriga, siis sündmus peab toimuma ja tingimus peab olema tõene, et üleminek toimuks. Kui üleminekuga on seotud ainult üleminekutingimus, siis üleminek toimub, kui tingimus saab tõeseks.

2.3.5.2 Paralleelsed olekuskeemid



Joonis 12: Paralleelne olekuskeem [FOWLER, lk 127]

Paralleelsed olekuskeemid on kasulikud, kui antud objektil on sõltumatute käitumiste komplektid. Samas ei tohiks ühesainsas objektis olla liiga palju paralleelseid käitumise kogumeid. Kui ühe objekti jaoks on mitmeid keerukaid paralleelseid olekuskeeme, tuleks kaaluda objekti poolitamist eraldi objektideks.

2.3.5.3 Millal kasutada olekuskeeme

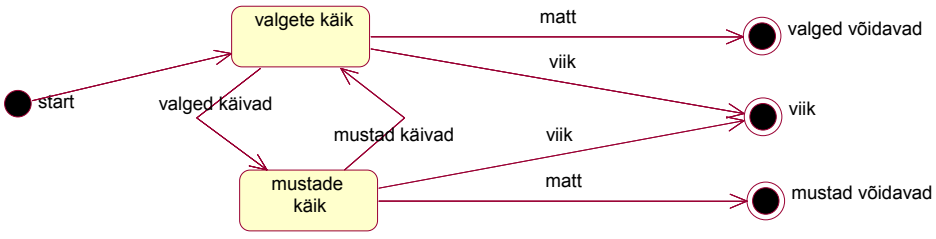
Olekuskeemid sobivad kirjeldamaks objekti käitumist üle mitmete kasutuslugude. Nad ei sobi hästi kirjeldamaks käitumist, mis sisaldab mitmeid objekte, mis teevad koostööd. On kasulik kombineerida olekuskeeme muude tehnikatega. Näiteks interaktsiooniskeemid on head kirjeldamaks mitmesuguste objektide käitumist ühes kasutusloos ja tegevusskeemid on head näitamaks üdist tegevuste järgnevust mitmesuguste objektide ja kasutuslugude jaoks.

Olekuskeeme ei ole mõtet püüda joonistada süsteemis iga klassi jaoks, kuna see on enamasti liigne jõu raiskamine. Olekuskeeme võiks kasutada vaid nende klasside jaoks, mis esindavad keerukat käitumist ja kus olekuskeemi loomine aitab aru saada, mis toimub. [FOWLER, lk 127-128]

Ülesanded 5:

1. Olek olekuskeemil on keeleliselt
    - omadussõna
    - nimisõna (ka verb) (**õige**)
    - tingimuslause
2. Mida kujutab endast üleminek (transition) olekuskeemil?
    - tegevust (andme)objekti(de)ga (**õige**)
    - valvurtingimuse täitmist
    - vaheolekut
3. Kirjeldage malemängu / kabemängu UML olekuskeemiga..

Vastus: näiteks..



Insipreeritud:
http://www.ics.uci.edu/~ebert/teaching/fall2000/ics121/lect/topic11/sld013.htm

2.3.6 Tegevusskeemid

Tegevusskeem on olekuskeemi erijuht, kus kõik või enamik olekuid on toimingolekud ning kus kõik või enamik siirdeid käivitatakse toimingute sooritamise teel lähteolekutes. Tegevusskeem sarnaneb klassikalisele programmi plokkskeemile ning väljendab töö kulgemist ajas ja organisatsioonis. Tegevusskeemi saab kasutada operatsioonide, klasside, või use case'ide jaoks, kuid ka lihtsalt töövoogude näitamiseks. Tegevusskeem sobib hästi äriprotsesside kirjeldamiseks.

Tegevusskeemid on kasulikud töövoo modelleerimisel ning käitumise kirjeldamisel, millel on palju paralleelset töötlemist. Vooskeemide ja tegevusskeemide erinevuseks on see, et vooskeemid on enamasti piiritletud järjestikuste protsessidega, kuid tegevusskeemidel saab käsitleda ka paralleelseid protsesse. Seega võimaldab tegevusskeem vältida käitumises ebavajalikke järgnevusi ning see parandab omakorda efektiivsust ja kiirust talitluse töötlemises. [ROSE, lk 13]

Tegevusskeeme võidakse kasutada ka näiteks kasutuslugude analüüsimiseks. Selles staadiumis ei olda huvitatud tegevuste jaotamisest objektide külge; on vaja lihtsalt aru saada, millised sündmused peavad toimuma ja millised on käitumuslikud sõltuvused. Meetodid jaotatakse objektidele hiljem ja näidatakse neid jaotusi interaktsiooniskeemiga.

Tegevusskeemid aitavad töövoost aru saada üle mitmete kasutuslugude. Kui kasutuslood on üksteisega interaktsioonis, siis tegevusskeemid on heaks vahendiks esitamaks seda käitumist ja sellest aru saamiseks.

Tegevusskeemid on kasulikud ka mitmelõimeliste rakendustega tegelemisel, kuna võimaldavad graafiliselt näidata, millised lõimed on olemas ja millal nad peavad sünkroniseeruma.

Nagu enamikul käitumuslikel modelleerimistehnikatel, on tegevusskeemidel lisaks tugevustele ja nõrkusi, nii et nad on parimad kombineeritult teiste tehnikatega. Tegevusskeemide suureks puuduseks on, et neil ei ole väga selgeid linke tegevuste ja

objektide vahel. Tegevusskeem ütleb, mis juhtub, kuid ei ütle, kes mida teeb. Programmeerimises tähendab see seda, et skeem ei edasta milline klass on vastutav millise tegevuse eest. Tegevusskeeme ei tasuks kasutada püüdmaks näha, kuidas objektid omavahel interaktsioonis on. Interaktsiooniskeem on lihtsam ja annab koostööst selgema pildi.

Tegevusskeemid ei sobi ka sel juhul, kui on soov näha, kuidas objekt käitub oma eluaja jooksul. Selleks võiks kastutada olekuskeemi. [FOWLER, lk. 141-142]

Ülesanded 6:

1.Kirjeldage, mis antud triviaalsel näiteskeemil toimub.

Kauba ostu-müügi protsess. Osalevad klient, klienditeenindaja ning ladu. Kauba väljastab ladu kliendi esitatud tellimuse peale ning kaup antakse kliendile pärast selle eest maksmist üle.

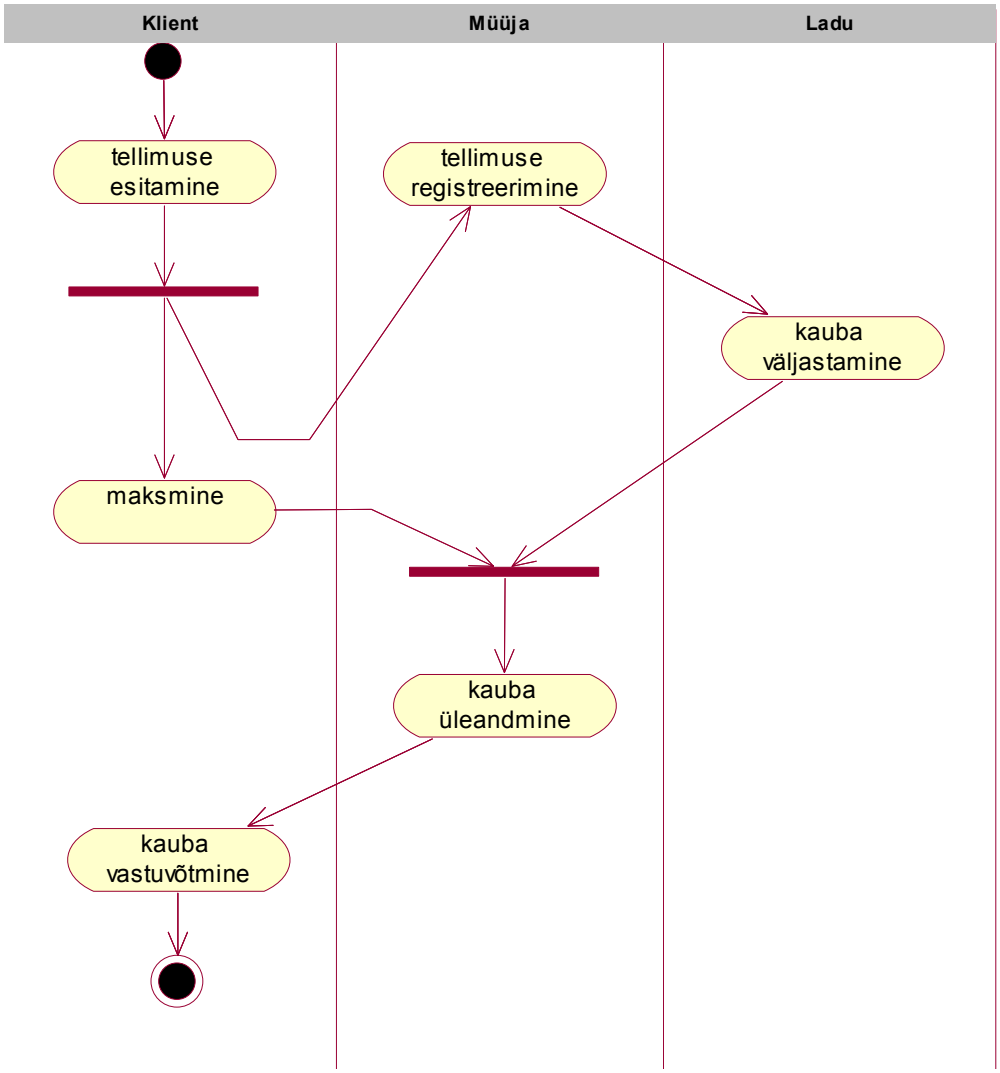
Millised võiksid olla tellimuse täitmisel kasutatavad andmeklassid? Millised nende olekud?

Tellimus – olekud --> vastu võetud, registreeritud, makstud, väljastatud

Kaup – olekud --> klienditeenindajale väljastatud, kliendile väljastatud

Töötaja ..

jne..



2. Mida tähistavad tegevusskeemil 'ujumisrajad' (swimlanes)?

- tegevuste jaotust tegutsejate kaupa (**õige**)
- eristavad olekuid ja tegevusi
- annavad tegevuste ajalise jaotuse

3. Milleks peamiselt kasutatakse tegevusskeemi?

Tegevusskeemi saab kasutada operatsioonide, klasside, või use case'ide jaoks, kuid ka lihtsalt töövoogude näitamiseks. Tegevusskeem sobib hästi äriprotsesside kirjeldamiseks.

Milles seisnevad tegevusskeemi peamised tugevused ja nõrkused?

Tegevusskeemid on kasulikud töövoo modelleerimisel ning käitumise kirjeldamisel, millel on palju paralleelset töötlemist. Tegevusskeeme võidakse kasutada ka näiteks

kasutuslugude analüüsimiseks. Tegevusskeemid aitavad töövoost aru saada üle mitmete kasutuslugude.

Tegevusskeemide suureks puuduseks on, et neil ei ole väga selgeid linke tegevuste ja objektide vahel. Tegevusskeem ütleb, mis juhtub, kuid ei ütle, kes mida teeb.

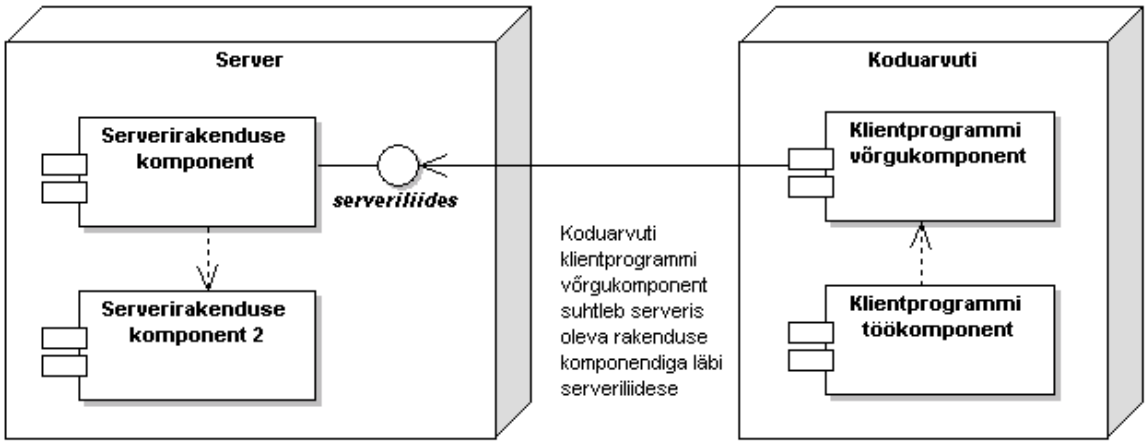
2.3.7 Komponendiskeemid ja levitusskeemid

Komponentskeeme ja levitusskeeme võib ühise nimega liigitada teostusskeemideks.

Komponendiskeem kirjeldab ära tarkvarakomponendid ja nende vahelised seosed. Komponent *(component)* on süsteemi füüsiline asendatav nimega osa, mis tähistab muidu loogiliste elementide paketti ning omab üht või mitut liidest.

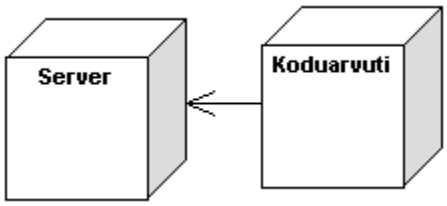
Levitusskeem kirjeldab ära võrgu sõlmed *(nodes)* ja ühendused *(connections)* nende vahel.

Nii komponent- kui ka levitusskeeme võib esitada eraldi, kuid on ka tehnikaid, et neid ühele skeemile kokku kombineerida ja nimetada endiselt levitusskeemiks, kuna komponendid on selle üheks osaks. Sellise määratluse järgi näitab levitusskeem loodavas süsteemis füüsilist suhet tarkvara ja riistvarakomponentide vahel. Levitusskeem esitab käitusaegsete töötlussõlmede konfiguratsiooni ning neis sisalduvad komponendid, protsessid ja objektid. Komponendid esindavad koodiüksuste käitusajailminguid.



Joonis 13: Komponente sisaldav levitusskeem

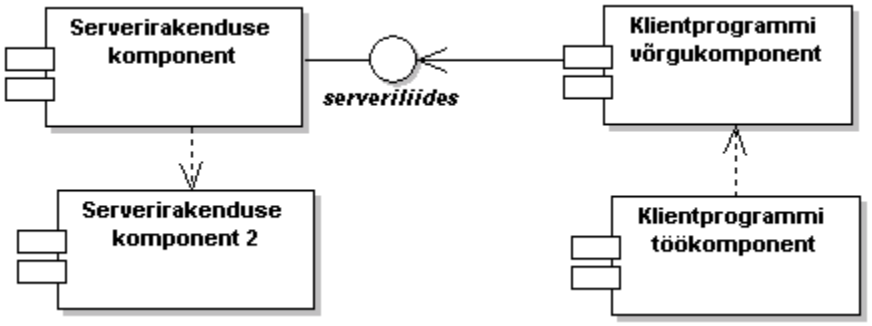
Iga sõlm levitusskeemil esindab mingit riistvarakomponenti. Riistvara võib olla lihtne seade või sensor või ka suurarvuti.



Joonis 14: Levitusskeem

Seosed *(connections)* sõlmede vahel näitavad kommunikatsiooniteid, üle mille toimub süsteemi interaktsioon.

Komponendid *(components)* levitusskeemil esindavad füüsilisi koodi mooduleid. Need vastavad täpselt pakettidele paketiskeemil, nii et levitusskeem näitab, kus iga pakett süsteemis töötab.



Joonis 15: Komponendiskeem

Sõltuvused *(dependencies)* komponentide vahel peaksid olema samad, mis pakettsõltuvused. Need sõltuvused näitavad, kuidas komponendid suhtlevad teiste komponentidega. (Näiteks joonisel 15 serverirakenduse komponent 2 sõltub serverirakenduse komponedist).

3 UML-i tööriistad

Tööriistade kirjeldused

ArgoUML (http://argouml.tigris.org/)

UML-i versiooni 1.3 toetav modelleerimisvahend, mis toetab põhilisi UML modelleerimiseks vajaminevaid diagramme ja ka pöördprojekteerimist keelega Java. Samuti võimaldab lugeda XMI-faile ning genereerida Java koodi. Miinuskülgedena võib nimetada kohati aeglast ja ebaintuitiivset kasutajaliidest. Võrreldes teiste vahenditega võiks sisaldada rohkem võimalusi, kuid hinna ja kvaliteedi suhe on paigas, sest tegemist on vabatarkvaraga. Reaalseks tööks sobib siiski kasutada ArgoUML-i baasil arendatud tarkvara Poseidon for UML, millest on olemas ka vabavaraline versioon.

Borland Together ControlCenter 6.0 (http://www.borland.com/)

Kõige võimalusterikkaim UML modelleerimis-tarkvara, mis eksisteerib. Kuna omab sisemist tarkvara-arenduskeskkonda ja ka eriversioone muude tarkvara-arendusvahendite jaoks, on modelleerimine ja otsene arendustegevus tihedalt omavahel seotud. Plusskülgedena võib tuua head integreeritust ja laia funktsionaalsust, mis võimaldab hõlpsamini süsteemides rakendada MDA printsiipe. Miinuskülgedeks võib pidada mitmekülgse funktsionaalsuse keerulist õpikurvi ja kallist hinda.

Eclipse UML Free Edition (http://www.eclipseuml.com/)

Tegemist on Eclipse UML Enterprise vabavaralise versioonina, mida erineb tasulisest versioonist vaid tiimitöö toe, versioonihalduse, koodimallide ja tarkvaramustrite toe puudumise poolest. Eclipse UML on mõeldud kasutamiseks Eclipse tarkvara-arenduskeskkonnas. Plusspoolena võib nimetada kasutamislihtsust ning head integratsiooni arenduskeskkonnaga. Miinusteks võib pidada ebamugavat installeerimisprotsessi ning tavapärasest veidi teistsugust UML elementide graafikat. Soovitav on seda modelleerimis-tarkvara kasutada ainult siis, kui on plaan arendustegevust läbi viia just Eclipse keskkonnas.

Sparx Enterprise Architect Professional v3.5 (http://www.sparxsystems.com.au/)

Laia UML diagrammide- ja mudel elementide valikut omav tarkvara. Plussideks võib pidada suhteliselt soodsat hinda, kiiret ja intuitiivset kasutajaliidest, paljude ka Windows-platvormil arendatavate keelte tuge ning pakutavaid mitmekülgseid võimalusi. Miinusteks võib lugeda diagrammielementide kapriisset käitumist mõningatel juhtudel, tihti ka tülikaid programmivigu ning veidi ebastandardseid mudeleid.

MagicDraw Professional 6.0 (http://www.magicdraw.com/)
Mitmekesiste võimalustega ja lihtsalt kasutatav tarkvara. Plussküljed: paljude võimalustega kuid samas lihtne modelleerimine, disainimustrite tugi, hea diagrammide graafika ja paljude programmeerimiskeelte genereerimise ja pöördprojekteerimise tugi. Miinusküljed: vajab tööks suhteliselt võimast arvutit ja pöördprojekteerimise toega versioon on suhteliselt kallis. Sobib hästi kasutamiseks igapäevatöös, kuna on üsna praktiline ja tarkvara pole üle kuhjatud liigsete võimalustega, need mis on, on väga praktilised. Meenutab funktsionaalsuselt ja ülesehituselt kõige rohkem tarkvara Rational Rose.

Poseidon for UML Community Edition 1.6.1(http://www.gentleware.com/)
Eespoolmainutud ArgoUML-ist arendatud ja täiustatud versioon, mis toetab eeskujulikult põhilisi UML diagramme ning kasutab diagrammide loomiseks üsnagi tavatut, klassikalistest modelleerimis-tarkvaradest erinevat, kuid mugavat ja intuitiivset meetodit. Omab üsnagi keeruliselt hallatavat kasutajaliidest ning koostab graafiliselt korrektseid diagramme. Plussküljed: vabavaraline, toetab eeskujulikult kõik UML diagramme ning omab head Java lähtekoodi generaatorit. Miinusküljed: konkurentidega võrreldes vähem võimalusi, puudub dokumentatsioonigeneraator näiteks. Java-põhise tarkvara modelleerimisel vägagi arvestatav vahend.

Rational Rose 2002 Enterprise Edition (http://www.rational.com)
Tegemist on ühe maailma esimese UML modelleerimisvahendiga, toetades UML versiooni 1.4. Rose on üsna hästi integreeritud väliste vahenditega (andmebaasid, teised tarkvara-arenduse tooted), kuid modelleerimisvõimaluste mitmekülgsuse osas jääb konkurentidele alla. Rose diagrammid jälgivad täpselt UML standardit ja on selged olles

standardikehtestajateks kogu tööstusele. Kuna Rose on suhteliselt muutumatuna püsinud juba mitmeid aastaid, on uue kontseptsiooniga tooted tunduvalt mugavamad modelleerimiseks. Plussküljed: tugev kontseptuaalsus, mitmekesine pöördprojekteerimine ja koodigenereerimine, hea kolmandate osapoolte tugi ja lihtne keskkond. Miinusküljed: aegunud liidesega ja oma pakutavate võimaluste kohta põhjendamatult kallis, sama modelleerimistöö saab tehtud 10 korda odavamate vahenditega. [ROSE] [15][16]

[15] **UML Products by Company.** Objects by Design, Inc.
URL: http://www.objectsbydesign.com/tools/umltools_byCompany.html

[16] **UML Tools (CASE & Drawing).** (2003)

http://www.jeckle.de/umltools.htm

Tabel 11: UML-i tööriistad

	Argo UML	Together Control Center	EclipseUML Free Edition	Enterprise Architect Professional	MagicDraw UML Professional	Poseidon for UML - Community	Rose-i Enterprise Edition
Hind	tasuta	6000$	tasuta	149$	899$	tasuta	4194
UML versioon	1.3	1.4	1.4	1.4	1.4	1.4	1.4
OCL tugi	+	-	-	-	-	-	-
Toetatud diagrammide arv	7*	8	9	10	9	7*	7
Toetatud protses	teadmata	teadmata	teadmata	teadmata	teadmata	teadmata	RUP
Skeemide eksportimise formaadid	GIF, PS, SVG	WMF, GIF, SCG	SVG, WMF, GIF, JPEG	BMP, PNG, JPEG, TGA, GIF, WMF, EMF	JPG, PNG, WMF, EMF, DXF, EPS, SVG	GIF, PS, SVG, png, JPEG	-
Dokumentat-siooni genereerimine	Java koodis	+	osaliselt	+	+	Javadoc automaatselt koodis	+
Koodi ja andmebaaside genereerimine	Osaline Java genereeri-mine	DDL - jah	Java genereeri-mine	Java, C++, C#, Visual Basic, Visual Basic.Ne, andmebaasid	Java, C++, C#, CORBA IDL, SQL DDL	Java	Java, Visual Basic, C++, Ada, Asp, erinevad andmebaasid
Pöördprojek-teerimine sh andmebaaside struktuur	Java	andmebaasid	Java	Java, C++, C#, Visual Basic, Visual Basic.Net, paljud andmebaasid	Java, C++, C#, CORBA IDL, SQL DDL	Java	Nii koodi- kui baasi-strktuuri pöördprojek-teerimine
XMI tugi - mis versioon / UML versioon	1.0/UML 1.3	+	-	XMI1.1/UML1.3	XMI1.1/UML1.4	XMI1.1/UML1.4	-
Andmehoidla või mitme kasutaja tugi	-	+, (versiooni-haldustark-vara kaudu)	-	+	+	-	+
Katab kogu UML sümboolika	+	+	osaliselt	+	+	+	+
Kasutajaliide-se mugavus (10 palli skaalal)	6	-	8	8	8	8	7
Integreerita-vus teiste toodetega / arenduskesk-kondadega	-	IBM Websphere, Jbuilder jpt	Integreeritud Eclipse keskkonda	-	NetBeans, Websphere, Eclipse	-	WebGain Visual Cafe, Sun Forte, Borland Jbuilder
Üldhinnang	**6**	**9**	**7**	**8**	**8**	**7**	**7**

4. UML-i arengusuunad

4.1 UML2.0

UML-i esimene versioon ilmus 1997 ning seda on mitmel korral parandatud ning täpsustatud. Aastal 2003 on ilmumas UML-i versioon 2.0, mis kätkeb endas mitmeid olulisi muudatusi.

Aastatel 2000 ja 2001 andis OMG välja neljaseerialise uuringu ettepanekute ehk *Request for Proposals (RPFs)* saamiseks UML2.0-i jaoks. Need neli esitist on:
UML2.0 infrastruktuur, mille tulemusteks oli: muuta lihtsamaks UML osade kasutamine, ilma, et peaks taaskasutama koos ühtede osadega ka teisi osi, parandused laienduste mehhanismis, UML ja *Meta Object Facility (MOF*) vahelise halva kokkusobivuse parandamine;
UML2.0 diagrammide vahetus, mille tulemuseks oli: mehhanismide definitsioon, mille abil tööriistad saavad vahendada skeemidel olevat informatsiooni;
UML2.0 objektitõkkekeel, mille tulemuseks oli: OCL-i metamudeli defineerimine;
UML2.0 ülemstruktuur, mille tulemuseks oli: teiste keelte poolt katmata keelte kaasajastamine. Kuid need on vaid mõned probleemid, millelele püütakse UML2.0 versioonis lahendus leida. [MDA2003 lk.76-77]

4.2 Model Driven Architecture ja UML

UML 2.0-st ja arengusuundadest rääkides ei saa jätta nimetamata ka tähtsat rolli, mida UML mängib MDA-s (Model Driven Architecture), mis on OMG uus nägemus süsteemide modelleerimisest. MDA arengu eest seisab hea Object Management Group (OMG), samuti nagu ka UML-i eest alates 1997 aastast (enne seda haldas UML-iga seonduvat Rational). Ehkki UML ei ole ainus keel, mis kuulub MDA-sse, on UML-l MDA-s tähtis roll.

MDA vaatenurga alt on UML-i peamisteks tugevusteks:

- UML eraldab abstraktse süntaksi konkreetsest süntaksist;
- UML ei ole üks fikseeritud keel, vaid pigem toetab laiendatud UML keelte definitsiooni, mida kutsutakse ka profiiilideks;
- teeb võimalikuks tõsta tarkvaraarenduse abstraktsioonitaset;
- UML-i hallatakse avatud standarditega üksuse poolt;

Mõningateks märkimisväärseteks puudusteks aga on:

- metamudeli elementide vahel on liiga palju ebavajalikke ristsõltuvusi;
- UML-il on erinevatele vaatenurkadele nõrk tugi;
- UML ja MOF, mis peaksid olema tihedalt koordineeritud, on natuke üksteise suhtes koordinatsioonist väljas;
- ei ole skeeme vahetavate tööriistade jaoks korralikku standardit. [MDA2003, lk. 77]

Siiani on harjutud mõtlema tarkvaramudelitest kui disainivahenditest ning 3GL programmidest kui arendusvahenditest. Paljud ettevõtted eraldavad modelleerija ja programmeerija rollid täielikult.

Selle tulemusel mudelid, nagu ka UML mudelid, on sageli mitteformaalsed, mis tähendab seda, et neid ei saa masintöödelda. Programmeerija kasutavad neid kui juhtnööre ja spetsifikatsiooni, kuid mitte kui midagi, mis otsestelt annab panuse tootmise protsessi.

MDA kasutab formaalseid mudeleid, mida saab masintöödelda. Sellised mudelid on tootmisprotsessi otsene osa. Sellises keskkonnas ei ole modelleerija ning programmeerija rollid enam kaugeltki nii erinevad. Modelleerimistegevus on samas ka programmeerimistegevus.

See ei tähenda, et enam ei ole mingit kasu mitteformaalsest modelleerimisest. Mitteformaalne modelleerimine jääb kasulikuks inimestevahelises kommunikatsioonis arhitektuuri ja disaini üle. Mitteformaalne modelleerimine on samuti kasulik UML-i kasutamine, kuid mitteformaalsed mudelid ei suuda koodigeneraatoreid või virtuaalarvuteid juhtida. [MDA2003], [MDACONF]

MDA alustab rakenduse platvormist sõltumatu mudeli ehk *Platform-Independent Model (PIM)* määratlemisega. PIM sisaldab arhitektuuriliselt olulisi tunnuseid, laskumata spetsiifilistesse rakendustehnoloogiatesse. Näiteks PIM pakub muuhulgas abstraktset viisi disainimaks kataloogiteenuste, turvalisuse, tehingute tegemise võimaluste ja püsivuse kasutamist. Kui säärane rakenduse ülevaade on välja joonistatud, on disaineritel lihtsam luua platvormispetsiifilisi detaile ja tehnoloogiaid, mis seda disaini optimaalselt rakendavad. Ühiselt on need detailid kokku kogutud platvormispetsiifilisse mudelisse ehk *Platform-Specific Model (PSM)*. Lõpuks loodetakse, et koodi on võimalik genereerida platvormist sõltumatutest mudelistest spetsiifilise platvormi disainirakenduste jaoks, kombineerides seda platvormist sõltuvate mudelite informatsiooniga. Kui soovid suunduda teisele platvormile, siis too PIM-i sisse teine aspekt ja regenereeri kood. Hetkel ei ole seda taasloomist võimalik teostada, kuid see on eesmärk. Selle eesmärgi saavutamine võimaldaks analüütikutel keskenduda vaid disainile, äriloogikale ja arhitektuurile. Need aspektid peaksid muutuma ainult selleks, et peegeldada uusi arenguid äritasandil. Ja koodi regenereeritaks alati automaatselt, kui disaini tehakse sääraseid kohendusi. Mõned unistavad kahepoolsest koodi projekteerimisest, nii et lihv, mida programmeerijad alati genereeritud koodile peavad andma, kajastuks ka kõrgematel disaini tasanditel. Siiski on selleni veel pikk tee. Tegelikkuses alles nüüd hakatakse rakendama kõiki MDA osi. [MDACONF, lk .14],

Millised siis on need MDA elemendid?

MDA baseerub kolmel modelleerimistehnoloogia, millest kõik omakorda baseeruvad UML-il. Need on:

Meta-Object Facility (MOF), mis pakub PIM-ile standardset andmehoidlat ja aitab identifitseerida, kuidas grupid näevad PIM-i, st. liideseid, mida kasutajategrupid saavad selles omada.

Common Warehouse Metamodel (CWM), mis OMG sõnades "standardiseerib aluse andmemodelleerimise ühtsustamiseks ettevõtte siseselt, üle andmebaaside ja andmehoidlate. Rajanedes alusmetamudelile, lisab see metamudeleid relatsiooniliste, salvestatud ja multidimensiooniliste andmete jaoks, teisendustele, OLAP-ile, andmete otsimisele ja andmeaida funktsioonidele kaasa arvatud protsess ja operatsioon. CWM

teisendab olemasolevaid skeeme, toetades automatiseeritud skeemi generatsiooni ja andmebaasi laadimist. See teeb sellest andmeotsingu ja OLAP-i jaoks baasi üle terve ettevõtte.

XML Metadata Interchange (XMI), on teisendus, mis väljendab UML-il põhinevaid mudeleid XML-is. Seeläbi võimaldab XMI UML-i dokumentide vahetada eri tööriistade vahel (näiteks mudeli koodi generaatorisse importimiseks).

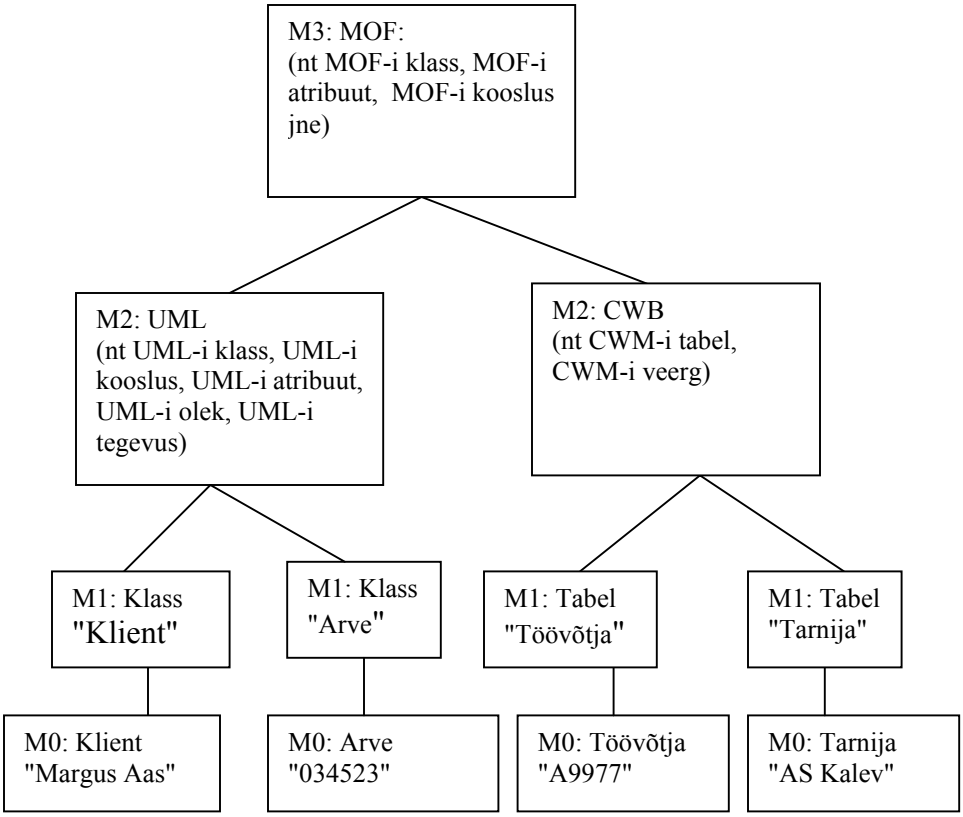
MOF-i, CWB, UML-i ja XMI, peaksid võimaldama MDA-s täpselt määratleda objektid, andmed ja disaini ükskõik millise konkreetse projekti jaoks. Kui seda kombineerida platvormi informatsiooniga, siis on selge, et kõik need andmed, mis on rakenduse jaoks vajalikud, on nüüd kokku kogutud ja täpselt määratletud. See on MDA laiem eesmärk.

MDA on veel arendamisel, kuigi suur osa tehnoloogiast on juba kasutatav. Kui arendajad ja IT-juhid peavad ühest tükist koosnevaid või klient-server rakendusi portima näiteks veebiteenuste arhitektuuri, siis muutub väga oluliseks moodus, kuidas kirjeldada programmi funktsionaalsust ja rakendust sõltumata konkreetse rakenduse detailidest. Analüütikute jaoks, kes on juba tuttavad UML-iga, ei tohiks MDA õppimine väga raskeks osutuda. [1]

Tabel 12: MDA metatasemed

Metatase	Kirjeldus	Elemendid
M3	MOF, näiteks konstruktsioonide kogu, mida kasutatakse metamudelite defineerimiseks	MOF-i klass, MOF-i atribuut, MOF-i kooslus, jne.
M2	Metamudelid, mis koosnevad MOF-i konstruktsioonide eksemplaridest	UML-i klass, UML-i kooslus, UML-i atribuut, UML-i olek, UML-i tegevus, jne. CWM-i tabel, CWM-i veerg, jne.
M1	Mudelid, mis koosnevad M2 metamudeli konstruktsioonide eksemplaridest	Klass "Klient", klass "Arve" Tabel "Töövõtja", Tabel "Kaupmees", jne.
M0	Objektid ja andmed, näiteks M1mudeli konstruktsioonide eksemplarid	Klient Margus Aas, klient Annika Urb, arve 5799, töövõtja A9977, kaupmees 957444, jne.

Joonis 16: MDA metatasemed



Kokkuvõte

Käesoleva töö eesmärgiks oli luua eesti keelne UML-i õppematerjal, mis oleks elektroonilisel kujul vabalt kättesaadav kõigile huvilistele.

Õppematerjali loomisel püstitatud ülesanneteks olid:

- uurida UML-i kasutamisega seonduvaid kogemusi ja probleeme Eesti tarkvarafirmade näitel, saada ettekujutus, mida peetakse praktikas UML-s rohkem ning mida vähem oluliseks ja miks, uurida UML-i kasutamise motiive.;
- kirjeldada peamisi UML-i modelleerimistehnikaid;
- anda valikuline ülevaade UML-i tööriistadest;
- tuua välja UML-i arengusuunad.

Kuna proseminaritöö maht ning ajakava võimaldasid anda UML-ist vaid põgusa ülevaate, siis andis bakalaureusetöö hea võimaluse teema uurimist jätkata. Esmaseks tööetapiks oli küsimustiku koostamine ning seejärel süvaintervjuude läbi viimine. Ettevõtted ja asutused, kellega ühendust võtsin, olid küsitluse suhtes positiivselt meelestatud ning vastutulelikud. Eranditult kõigist ettevõtetest oldi nõus oma kogemusi antud valdkonnas jagama.

Edasiseks töö etapiks oli küsitluste analüüs ning selle põhjal järelduste tegemine. Selgus, et modelleerimistehnikate kasutamise osas oli kokkulangevus üsna suur raamatutes kirjutatavaga. Lisaks uuringu läbiviimisele oli bakalaureusetöösse kavandatud lisada tööriistu kirjeldav osa. Ka uuringu tulemus andis kinnitust tööriistadel pikemalt peatumise vajadusele. Samuti selgus, et ehkki tööriistade osas on valik lai, on neil mitmeid puudusi. Sagedamini mainiti, et mugavamad tööriistad on liiga kallid, vabavaralisi aga peeti pisut kohmakateks. Siiski õppimise seisukohalt on töö autori meelest piisavalt palju häid tööriistu kättesaadavad.

UML arengusuundade kirjeldamise olulisusest tekkis arusaam esmalt veebimaterjalide käsitlemisel, kuna uudiskirjanduse hankimine antud valdkonnas ei pruugi olla lihtne. Raamatute hankimisel osutasid abi intervjueeritavad IT firmadest. Töö käigus selgus

tõsiasi, et UML-i õppida soovijatel ei ole raamatukogudes pakutav antud temaatikat käsitlevate raamatute valik just lai. Töö autori arvates peitub UML-i arengus veelgi palju huvitavaid võimalusi ning seega võiks ka käesolev töö olla baasiks, et antud teemakäsitlust veelgi jätkata.

Tabelid

Joonised

Kasutatud kirjandus

1. [1] **Binstock, A.** MDA gives us reason to believe in methodology ... at last URL: http://www.devx.com/SummitDays/Article/7803

2. [ROSE] **Boggs, W., Boggs M.** (2002). UML wiht Rational Rose 2002 . SYBEX Inc.

3. [BOOCH] **Booch, G.** (1994). Object-oriented Analysis and Design With Applications. 2nd ed. The Benjamin/Cummings Publishing Company, Inc.

4. [JAVA] **Eckel, B**. (2000) Thinking in Java 2, Prentice Hall, lk 30-32.

5. [FOWLER] **Fowler, M., Scott, K.** (1998). UML Distilled: Applying the Standard Object Modeling Language. Addison Wesley Longman, Inc.

6. [MDA2003] **Frankel, D. S.** (2003). Model Driven Architecture Applying MDA to Enterprise Computing.Wiley Publishing, Inc.

7. [KOBRYN] **Kobryn, C.** Introduction to UML: Structural and Use Case Modeling
   URL:http://www.omg.org/news/meetings/workshops/presentations/embedded-rt2002/02-1_Kobryn_UML_Tutorial.pdf

8. [LARMAN] **Larman, C.**(2002). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. 2nd ed. Prentice Hall PTR.

9. [SONASTIK] **Littover, M.** UML-keele sõnastik.
   URL: http://www.cc.ioc.ee/uml, 17.05.2003

10. [SPETSIF] OMG Unified Modeling Language Specification: Version 1.3 (1999) URL: http://www.rational.com/media/uml/post.pdf, 17.05.2003

11. [ROOST] **Roost, M.** (2001). Objektorienteeritud modelleerimine. Loengukonspekt.

12. [RUP] **Seeba, A.** (2001). Unifitseeritud tarkvaraarendusprotsess ja selle rakendamise juhtumianalüüs. Magistritöö.
URL: http://www.math.ut.ee/~asko/tarkvaratehnika/magistritoo.pdf

13. [MDACONF] **Watson, A.** (2003). Introduction to Model Driven Architecture Conference Proceedings: OMG Information Day on Integrating the Enterprise.

14. [OMG] About the Object Management Group (OMG)s
URL: http://www.omg.org/gettingstarted/gettingstartedindex.htm

Summary

UML Learning Material

Liina Lang

Tallinn Pedagogical University

The development of object-oriented programming created a necessity for object-oriented designing methods. Different methods developed during years. This made it more difficult for the designers to communicate with each other. It was also complicated for the users of the methods to find a modeling language, which would have satisfied all of their needs at the same time. In order to improve the situation G. Booch, I. Jacobson and J. Rumbaugh, three creators of the method standing out from among the others, decided to create the *Unified Modeling Language (UML)*. UML combines mainly these three methods: *Booch, Object-Oriented Software Engineering (OOSE)* and *Object Modeling Technique (OMT)*, but it is wider in its essence. UML version 1.0 was standardized by the *Object Management Group (OMG)* in 1997.

UML is a modeling language, not a method. Methods consist of both, a modeling language and a process. A modeling language is a (mainly graphical) means of marking. A process determines the sequence of actions in a development process.

The aims of creating UML are: to model systems by using object-oriented (OO) techniques starting from their concepts up to applications; to enable operation also with large-scale and complicated systems; to create a modeling language, which could be read by both, people and machines. The field of the application of UML includes all kinds of systems: information systems, technical systems, real-time systems, distributed systems, business systems, etc. UML is a language for visualizing, specifying, developing and documenting of a system.

Since UML is a standardized way for system modeling used more and more widely in the world, the introduction of UML in the Estonian software companies becomes more and

more topical. Unfortunately there is little material in Estonian, that is freely available in this subject. The course materials offered by training companies are expensive. In the framework of the subjects taught in the universities, the lecture materials are often either only brief and difficult to understand for the interested people, who have not listened to the relevant lecture, or the materials are protected by a password and thus available only to a small target group. Also, UML is handled as one part in the framework of the lectures and not the whole time the lecture course is dedicated to it. Thus, the objective of this bachelor thesis is to create learning material, which would be freely available in an electronic form and would be meant also for a wider target group, including the informatics students of TPU and other universities and software developers. This bachelor thesis is the advancement of the pro-seminar work "Survey of the Modeling Language UML".

The thesis has been divided structurally into four parts: study of the use of UML in Estonian software companies, UML learning materials, UML tools and UML development directions.

One of the tasks set in the thesis is to describe the main modeling techniques of UML, which include both, notation and semantics. Altogether nine modeling techniques have been described: use case diagrams, class diagrams, package diagrams, sequence diagrams, collaboration diagrams, state diagrams, activity diagrams, component diagrams and deployment diagrams. Here the selection is made on the basis of the research carried out in eight Estonian software companies.

The research has been carried out in the form of a profound interview and the aim was not to make statistic generalizations, but to find confirmation to the issues read from books and to study the experiences and problems connected with UML. The research consists of six parts: the experiences of the interviewee at the use of UML and/or object-oriented methods, usefulness of UML components, UML tools, analyst-client communication, data of the organization and IT background of the interviewee.

There is a wide range of UML tools and each of them has its own pluses and minuses. In order to simplify the selection of a tool in accordance with the needs and possibilities, there is a survey presented in the thesis describing, which function each tool offers. Since the priority here is not to describe all the tools and all the possibilities offered, but rather to give a survey in this part, a selection has been made from among several tools and they have been summarized in a table for the comparison of the qualities. The selection is based on the fact, which tools are preferred by the interviewees and which tools the author of the work himself has come across.

The software world is a constantly changing and innovative field and those, who can make profits from the changes and apply the innovations in their work, are successful in the first place. Therefore, a chapter about the development directions of UML has been added to this thesis. The abovementioned chapter should give ideas and arouse interest and motivation for getting deeply acquainted with UML. In the UML development directions the modifications and corrections made in the UML version 2.0, which will be issued soon, have been described first and also the significant role of UML in the *Model Driven Architecture (MDA)*, which is a new method for the development of software and systems.

Lisad

Lisa 1

Osa 1: Intervjueeritava kogemused UML-i ja/või objekt-orienteeritud meetodite kasutamisel

1. Olete Te osalenud süsteemiarendusprojektides, milles kasutatakse/kasutati UML-i (nt. kasutuslooskeemid, klassiskeemid, olekuskeemid, tegevusskeemid jne.)

- *Jah*
- *Ei*

2. Miks Te ei ole süsteemiarendusprojektides kasutanud UML-i või objekt-orienteeritud arendusmeetodeid?

- *Minu organisatsioonil ei ole sobivaid projekte UML või objekt-orienteeritud süsteemiarenduseks;*
- *UML on kasutuselevõtuks liiga kallis;*
- *Organisatsioonis kasutusel olevad töövahendid ei toeta UML-i kasutamist;*
- *Minu organisatsioonis on liiga vähe inimesi, kes on tuttavad UML või objektiorienteeritud meetoditega;*
- *Minu organisatsioon on pühendunud teistsugus(t)ele lähenemis(t)ele arenduses (palun täpsustage, millis(t)ele);*
- *Isiklikud põhjused:*
  Ma ei osale süsteemiarendustöös;
  Ma ei näe mingit kasu UML-i kasutamisest (nt. skeemide põhjal genereeritav kood pole tootmiseks valmis jmt.);
  UML-i on liiga keeruline õppida;
  UML on kasutamiseks liiga keerukas;
- *Muud põhjused (täpsustage palun)*

3. Kui suur osa nendest objekt-orienteeritud/UML projektidest, millesse Te olete olnud kaasatud, sobiksid järgnevatesse kategooriatesse?

- *Uus süsteem;*
- *Vana süsteemi täielik väljavahetamine;*
- *Olemasoleva süsteemi laiendamine*

4. Millised kategooriad kirjeldavad nende süsteemide valdkondi kõige paremini

- *Administratiivne*
- *Tootmine*
- *E-kaubandus/veeb*
- *Kliendisuhete haldus (CRM)*
- *Mobiilne kaubandus*
- *Muu (palun täpsustage)*

5. Missugust mõju on UML-i kasutuselevõtt projektidele avaldanud?

- *On lihtsustanud kliendi vajaduste mõistmist*
- *On tõstnud loodava süsteemi kvaliteeti*
- *Aidanud kaasa rakenduse õigeaegsele valmimisele*
- *On võimaldanud kokku hoida kulusid ning aega*
- *Muu (palun täpsustage)*

6. Kas olete kursis OMG Model Driven Architecture (MDA) olemusega?
   - *Jah*
   - *Ei*

7. Kas Te olete kasutanud või kasutate MDA-d projektis?
   - *Jah*
   - *Ei*

8. Kas Teile järgmised lühendid on tuttavad: Common Warehouse Metamodel (CWM)
   XML Metadata Interchange (XMI),
   Meta-Object Facility (or MOF)

9. Milline mulje on jäänud MDA kasutamise mõjust projektile?

10. Kui sageli ja kas Te kasutate UML-i laiendusi (nt. Object Constraint Language, OCL)?

Osa 2: UML komponentide kasulikkus

11.1. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu kasutuslooskeemidest järgnevates osades:

	Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*				
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*				
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*				
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*				
▪ *Muu (palun täpsustage)*				

11.2. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu klassiskeemidest järgnevates osades:

	Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*				
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*				
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*				
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*				
▪ *Muu (palun täpsustage)*				

11.3. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu paketiskeemidest järgnevates osades:

Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*			
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*			
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*			
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*			
▪ *Muu (palun täpsustage)*			

11.4. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu olekuskeemidest järgnevates osades:

Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*			
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*			
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*			
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*			
▪ *Muu (palun täpsustage)*			

11.5. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu tegevusskeemidest järgnevates osades:

Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*			
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*			
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*			
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*			
▪ *Muu (palun täpsustage)*			

11.6. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu jadaskeemidest järgnevates osades:

Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*			
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*			
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*			
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*			
▪ *Muu (palun täpsustage)*			

11.7. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu koostööskeemidest järgnevates osades:

Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*			
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*			

- *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*
- *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*
- *Muu (palun täpsustage)*

11.8. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu komponentskeemidest järgnevates osades:

	Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*				
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*				
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*				
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*				
▪ *Muu (palun täpsustage)*				

11.9. Mil määral on UML projektides, kus Te olete osalenud, olnud kasu levitusskeemidest järgnevates osades:

	Pole olnud kasu	*On olnud kasulik*	*On olnud hädavajalik*	*Pole kasutanud*
▪ *Koos kliendi esindajatega nõuete välja selgitamisel ja määratlemisel*				
▪ *Süsteemi nõuete täpsustamisel programmeerijatele*				
▪ *Dokumenteerimisel edaspidiseks hoolduseks ja laiendusteks*				
▪ *Rakenduse paremaks mõistmiseks projektimeeskonna tehniliste liikmete/riistvara seas*				
▪ *Muu (palun täpsustage)*				

12. Kas tähistuse kasutamisel on mingeid häirivaid aspekte?
    - *Jah (palun täpsustage, milliseid)*
    - *Ei*

13. Mida võiks tähistuses muuta ja millised on ettepanekud?

14. Kui sageli on olnud võimalik taaskasutada komponente UML-il põhinevatest süsteemiarendusprojektidest uutes süsteemiarendusprojektides?
    - *Mitte üldse*
    - *Harva*
    - *Sageli*

Osa 3: UML tööriistad

15. Kas Teie organisatsioon kasutab UML või muid tööriistu süsteemianalüüsiks ja -disainiks?
    - *UML tööriistu (palun täpsustage)*
    - *Muid analüüsi- ja disainitööriistu (palun täpsustage)*
    - *Tööriistu ei kasutata*

16. Kui Te kasutate enam kui ühte tööriista, kas kasutate neid paralleelselt või pigem eraldi?

- *Kasutan paralleelselt*
- *Kasutan eraldi*

17. Miks eelistate valitud tööriista/tööriistu teis(t)ele?

18. Kas tööriistade osas mida Te kasutate, on mingeid häirivaid aspekte?
    - *Jah (täpsustage palun, milliseid)*
    - *Ei*

19. Kui suures osas on mudelitest genereeritav kood valmis kasutuselevõtuks?
    - *Koodi tuleb muuta ning täiendada ainult vähesel määral*
    - *Koodi tuleb suures osas muuta ning täiendada*
    - *Selliselt genereeritud kood on kasutamiseks kõlbmatu*

20. Kas pöördprojekteerimise (reverse engineering) võimalust kasutate?
    - *Jah*
    - *Ei*

Osa 4: Analüütiku-kliendi suhtlus

21. Kui edukas on olnud UML-i kasutamine klientidega suhtlemisel?
    - *Ei ole olnud edukas*
    - *Üsnagi edukas*
    - *Väga edukas*

22. Milliste skeemide arendamisse on kliendid olnud kaasatud ning kui suurel määral? (kasutuslooskeemid, klassiskeemid, pakettskeemid, koostööskeemid, jadaskeemid, olekuskeemid, tegevusskeemid, levitusskeemid, komponentskeemid)

Osa 5: Organisatsiooni andmed

23. Mitu inimest on ettevõttes, kus Te töötate, seotud tarkvara arendamisega?
    - *Kuni 2*
    - *3-5*
    - *6-12*
    - *13-50*
    - *Rohkem kui 50*

24. Ettevõttes, kus Te töötate, arendatakse tarkvara?
    - *Peamiselt müügiks*
    - *Peamiselt firmasiseseks kasutamiseks*

25. Milline järgnevatest tüüpidest kõige paremini iseloomustab organisatsiooni, kus Te hetkel töötate?
    - *Konsultatsioonifirma*

- *Finants*
- *Kommunikatsioon, meelelahutus, meedia*
- *Hariduslik institutsioon*
- *Tarkvara arendus*
- *Tööstus*
- *Riigiasutus*
- *Tervishoid*
- *Turism*
- *Transport*
- *Muu (täpsusta palun)*

Osa 6: Intervjueeritava IT taust

26. Kes Te olete?
    - *Tippjuht*
    - *Projekti- või grupijuht*
    - *Tarkvara arendaja*
    - *Muu (täpsustage palun)*

27. Milline on Teie haridus?
    - *Kõrgem IT*
    - *Kõrgem*
    - *IT eri*
    - *Muu*

28. Milline järgnevatest allikatest olid teile tähtsad UML-i õppimisel?
    - *Ülikooli- või keskkooliprogrammid*
    - *Kursused*
    - *Raamatud, ajakirjad ja ajalehed*
    - *Veebilehed*
    - *Kolleegid (mitte ülemused) ja konsultandid*
    - *Ülemused*
    - *Muu (palun täpsustage millised)*

29. Kas Teie arvates on UML-i õppimise vahendid piisavalt kättesaadavad?
    - *Jah*
    - *Ei (täpsustage palun, miks)*