TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatika instituut
Infosüsteemide õppetool

Animatsioonid infosüsteemi analüüsimudelite kohta

Bakalaureusetöö

Üliõpilane: Kristine Baumann
Üliõpilaskood: 120935IAPB
Juhendaja: dotsent Erki Eessaar

Tallinn
2015

Autorideklaratsioon

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

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Animatsioonid infosüsteemi analüüsimudelite kohta

Antud lõputöö eesmärgiks oli koostada animatsioon mõningate infosüsteemi analüüsimudelite kohta, mida oleks võimalik kasutada õppematerjalina. Animatsioon koostati MS PowerPointis. Eesmärgiga võrrelda graafilise kasutajaliidesega loodud animatsiooni ning programmeerimiskeeles realiseeritud animatsiooni, loodi osaline animatsioon samal teemal ka Java Swing'i kasutades.

Lõputöö raames valmis MS PowerPoint tarkvaral animatsioon, mis kujutab infosüsteemi äriarhitektuuri skeemi, kasutusjuhtude diagrammi, olemi-suhte diagrammi, seisundidiagrammi, CRUD maatriksit ja andmebaasioperatsioonide lepinguid. Samuti valmis osaline animatsioon Java Swing tarkvaral ning lõputöö koostaja järeldas, et mugavam oli animatsiooni koostada MS PowerPointis.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti koos lisadega 46 leheküljel, 6 sisulist peatükki, 31 joonist, 1 tabelit.

Abstract

Animations About Analysis Models of Information Systems

The goal of the bachelor's thesis at hand was to create animations about some analysis models of information systems with the aim that the created animation could be used as a study material. The animation was created using MS PowerPoint. With the goal to compare the process of creating animations using a graphical interface and writing actual code, a partial animation was created in Java Swing as well.

As a result of this bachelor's thesis an animation using MS PowerPoint was created. It contains the business architecture model, use case diagram, entity relationship diagram, state machine diagram, CRUD matrix, and database operation contracts of an information system. In addition to that, a partial animation was created in Java Swing and the author of the bachelor's thesis concluded that it was easier to create the said animation in MS PowerPoint than it was in Java Swing.

The thesis is in Estonian and contains 46 pages of text, 6 chapters, 31 figures, 1 table.

Sisukord

Sissejuhatus

Lõputöö teema valik tuleneb autori sügavast huvist andmebaaside projekteerimise vastu ning põhineb neljandal ja viiendal semestril läbitud ainete "Andmebaasid I" ning "Andmebaasid II" kasulikkusel. Kuna materjali omandades selgus, et tegemist on väga mahukate ainetega, mis hõlmavad suurt hulka teavet, otsustati koostada lõputöö raames õppematerjal, mis loodetavasti lihtsustab nii õpetamise kui õppimise protsesse.

Antud lõputöös esitatakse täielikud animatsioonid MS PowerPoint tarkvara põhjal loodutena ja osaliselt programmeerimiskeeles Java kirjutatuna, et võrrelda kahe erineva tarkvara võimalusi.

MS PowerPointi kõrvale on valitud Java Swing, sest MS PowerPoint on oma olemuselt üsna lihtne ja kasutajasõbralik ning seal ei ole ka päris keerulise käitumise saavutamiseks vaja programmikoodi kirjutada. Animatsiooni loomine Java programmeerimiskeeles erineb MS PowerPointi tehnoloogiast suuresti olles täiesti koodipõhine. Töös ning animatsioonides kasutatavad näitemudelid on saadud juhendaja materjalidest, kuid animatsioonid on autori omapoolne looming ja panus.

Lõputöö eesmärkidena tuuakse välja järgnev.

- Klassidiagrammide, kasutusjuhtude diagrammide, seisundidagrammide, äriarhitektuuri diagrammide ning CRUD maatriksi üldine kirjeldamine
- Ülevaade MS Powerpoint võimalustest
- MS Powerpoint tarkvaral animeeritud slaidide koostamise protsess
- Ülevaade programmeerimiskeelest Java ning Java Swingist
- Java Swing tehnoloogial animatsiooni koostamine
- Loodud animatsioonide võrdlus

1.Üldine ülevaade töös kasutavatest diagrammidest

UML-is (Unified Modeling Language) diagrammid saab jagada kahte kategooriasse: struktuuri diagrammid ning käitumist kirjeldavad diagrammid.

1.1 Klassidiagramm

Oma olemuselt kuulub klassidiagramm struktuuridiagrammi alla ning on üks selle kategooria parimaid ja olulisemaid näiteid. Klassidiagrammi eesmärk on kajastada klasse ning seoseid nendesse kuuluvate objektide vahel.

UML-is esitatakse klassi ristkülikuna. Antud ristkülik koosneb kolmest vertikaalselt järjestatud komponendist. Esimene jaotus on klassi nime jaoks – joonisel 1 käsitletakse klassifikaatorite klassi. Teises jaotuses on klassi atribuudid – siin näites klassifikaatori kood, nimetus ning kirjeldus. Kolmas jaotus on klassi operatsioonide jaoks. Antud näites operatsioonid puuduvad. Klassi luues on kohustuslik vaid esimene jaotus - klassi nimi. Teised kaks on valikulised ning võivad ka puududa.[1]

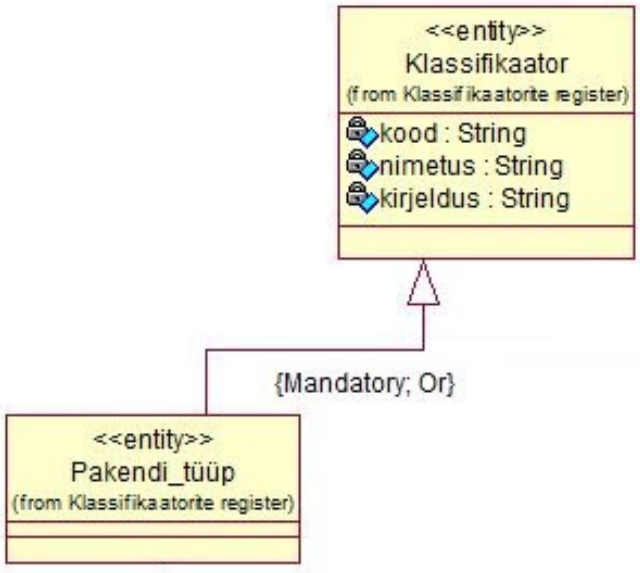
Joonis 1 Klassidiagrammil kujutatud klass *Klassifikaator*

Atribuutide jaotuses tuuakse välja nii atribuudi nimetus kui ka tüüp. Seega on atribuut *kood* tüüpi *String* nagu ka atribuudid *nimetus* ja *kirjeldus*.

Operatsioonide jaotuses tuuakses sarnaselt atribuutidele välja nii operatsiooni nimetus kui ka operatsiooni tagastatava väärtuse tüüp. Operatsioonides saab ka kirjeldada sisendparameetrid (*input*), et defineerida, milliseid andmeid antud operatsioon kasutab.[1] Iga parameeter on mingit tüüpi. Nagu ka varasemalt mainitud, selles töös operatsioone klassidiagrammis ei kasutata.

Väga oluline mõiste on klassidiagrammi juures ka pärimine. Pärimine tähendab, et alamklass (*subclass*) saab mingid andmed (atribuudid) või funktsionaalsuse ülemklassilt (*superclass*) ja

samal ajal lisab ka oma spetsiifilised atribuudid või funktsionaalsuse.[1] Seda väljendatakse joonega, mis lõppeb suletud noolega nagu võib näha joonisel 2 ning mis on suunatud ülemklassi poole.



Joonis 2 Klassidiagramm pärimissuhtega klasside *Klassifikaator* ja *Pakendi_tüüp* vahel

Teine oluline mõiste on klassidevaheline seos (*association*). Süsteemi modelleerimises tuleb arvestada, et teatud objektid on omavahel seotud. Kõige levinum seosetüüp on kahesuunaline seos (*bi-directional or standard association*). Seosed on vaikimisi kahesuunalised, mis tähendab, et mõlemad klassid on üksteisest ning neid ühendavast seosest "teadlikud", kui seost ei defineerita mõnda teist tüüpi.[1] Neist võib mõelda kui kahesuunalise liiklusega sildadest nendesse klassidesse kuuluvate objektide vahel. Seoseid kujutatakse UML-is katkematu joonena, mille mõlemasse otsa pannakse seose mitmesuste arvuline väärtus (*multiplicity values*) nagu on näha joonisel 3. Seoste mitmesuste väärtusi on erinevaid. Mitmesus näitab kui palju objekte võib osaleda antud seoses. Järgnevalt tuuakse välja mõned näited.

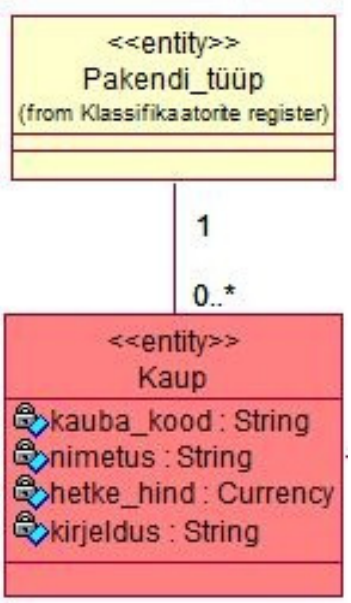
0 - 1 Null või üks

1 Täpselt üks

0..* Null või enam

* Null või enam

1..* Üks või enam

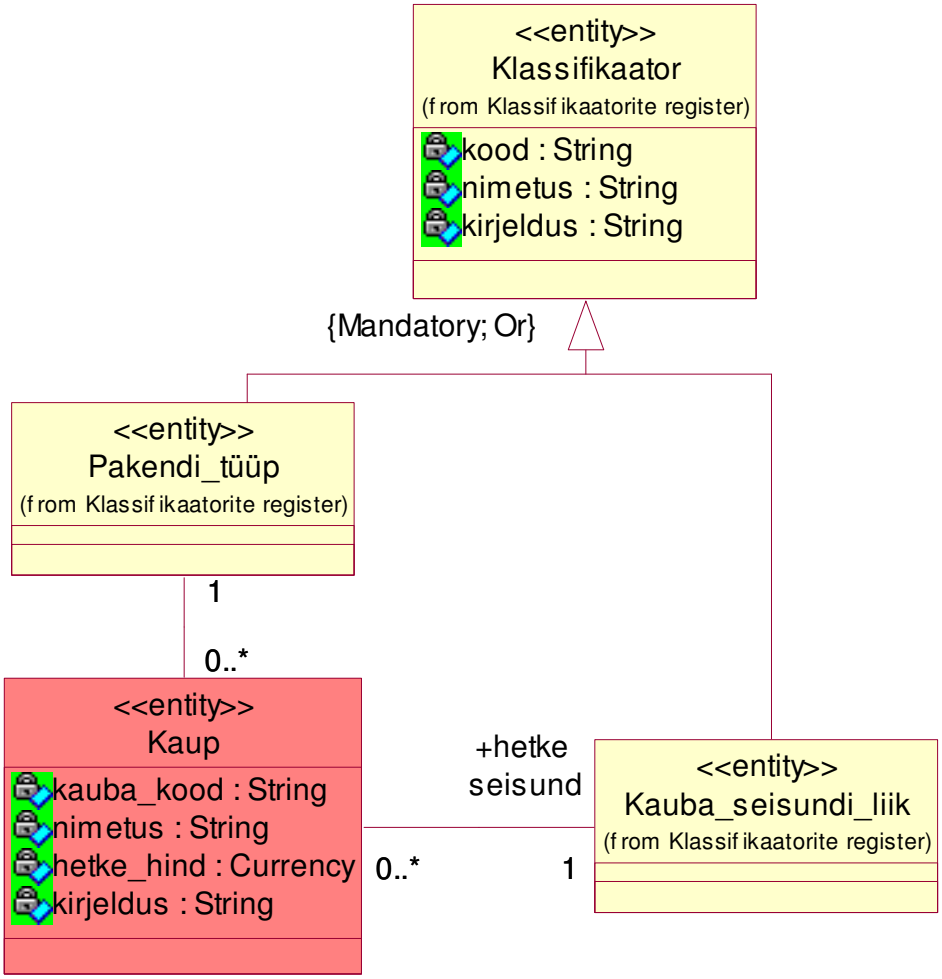


Joonis 3 Klasside vahelised seosed klassidiagrammil

Antud joonisel on esitatud seosed "täpselt üks" ning "null või enam". See tähendab, et igale kaubale (klassi *Kaup* kuuluvale objektile) vastab täpselt üks pakendi tüüp kuid ühte pakenditüüpi (klassi *Pakendi_tüüp* kuuluvat objekti) võib kasutada 0 või enam kaupa.
Lisaks kahesuunalisele seosele on võimalik UML klassidiagrammides kasutada veel:

- ühesuunalist seost, mida kujutatakse diagrammil katkematu joonena, mis lõppeb lahtise noolega, mitte kinnisega nagu kasutati pärimise puhul
- sidemeklassi, ehk lisatakse uus klass, mis sisaldab vajalikku informatsiooni kahe klassi vahelise seose kohta. Seda kujutatakse diagrammil tüüpilise klassina, mis on seotud seosejoonega katkendliku joonega
- osa-terviku (*aggregation*) seost, mille alamtüübiks on kompositsiooniline seos (*composition*)
- rekursiivne seos, mis tähendab seda, et klass saab olla seotud ka iseendaga, mis tähendab, et klassi mingi osa on seotud sama klassi teise osaga [1]

Kõike eelnevat arvesse võttes näeb antud näite täielik klassidiagramm välja selline nagu kujutatud joonisel 4. Klassidiagrammi saab kasutada erinevatel otstarvetel – näiteks mõistekaartide ja valdkonnamudelite koostamine, andmebaaside projekteerimine, objektorienteeritud tarkvara projekteerimine või veebilehtede navigatsiooni kirjeldamine. Antud töös kasutatakse klassidiagrammi andmebaasi kontseptuaalseks projekteerimiseks ning selle abil luuakse olemi-suhte diagramm, mis on osaks kontseptuaalsest andmemudelist.



Joonis 4 Kaupade arvestuse näite olemi-suhte diagramm

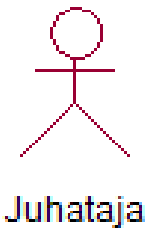
1.2 Kasutusjuhtude diagramm

Kasutusjuhtude mudeleid saab kasutada, et näidata, millist funktsionaalsust peaks loodav süsteem pakkuma või millist funktsionaalsust olemasolev süsteem juba pakub. Kasutusjuhtude modelleerimises vaadatakse süsteemi kui "musta kasti", mis pakub kasutajale

teenuseid ehk kasutusjuhte. Antud mudeli juures ei ole oluline, kuidas süsteem täpsemalt midagi teeb. Kasutusjuhtude mudel koosneb **kasutusjuhtude diagrammist** ja kasutusjuhtude tekstilistest kirjeldustest. [2]

Kasutusjuhtude modelleerimine defineerib süsteemi piirid – seda tehakse süsteemi poolt käsitletava funktsionaalsusega.[2]

Kasutusjuhtude diagrammi puhul räägitakse alati **tegutsejatest.** Tegutsejat kujutatakse UML keeles nii nagu näha joonisel 5.



Joonis 5 Tegutseja kasutusjuhtude diagrammist

Tegutseja on keegi või miski, mis suhtleb süsteemiga või kasutab seda. Tegutsejad viivad läbi kasutusjuhte. Oluline on silmas pidada, et tegutseja esitab rolli, joonisel 5 on selleks rolliks juhataja, mitte aga süsteemi üksikkasutajat. Tegutseja poolt algatatakse kasutusjuht ja see on süsteemi jaoks sündmus, mis käivitab vajaliku teenuse.[2]

Kasutusjuht on jutustav dokument, mis kirjeldab sündmuste jada, mis tekib, kui tegutseja, süsteemiväline agent kasutab süsteemi mingi protsessi läbi viimiseks (Ivar Jacobson). Kasutusjuht on mingi protsessi samm-sammuline kirjeldus. Kasutusjuht on tavaliselt võrdlemisi mahukas. [2]

Kasutusjuhu omadused on järgnevad.

- Kasutusjuht käivitatakse alati tegutseja poolt (kui tegemist on automaatselt käivitatava kasutusjuhuga, saab tegutsejaks märkida *Aeg*)
- Kasutusjuhu käivitamise tingib mingi süsteemi jaoks oluline sündmus
- Kasutusjuht peab olema täielik, st kasutusjuht annab tegutsejale ja/või laiemalt võttes huvitatud osapooltele väärtuse (aitab neil saavutada mõnda enda eesmärki)
- Kasutusjuht viiakse läbi ühel ajas ja kohas ühe tegutseja poolt [2]

Selleks, et kasutusjuht ei muutuks liiga keeruliseks, tuleks selle mõningad harutegevused teha omakorda uuteks kasutusjuhtudeks eeldusel, et need harud esinevad ka teistes kasutusjuhtudes või on liiga pikad ja keerukad ning nende lahti seletamine aitab lihtsustada algset

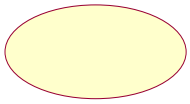
kasutusjuhtu.[2] Sellest võib mõelda kui kasutusjuhtude normaliseerimisest analoogina andmebaasi normaliseerimisele.

Kasutusjuhtude vahel eristatakse seoseid: <<include>> ja <<extend>>.

Laiendamisseos <<extend>> näitab, nagu ka näiteks programmeerimiskeeles Java objektorienteeritud programmeerimise puhul kasutatav *extends*, et laiendatava kasutusjuhu eksemplar võib sisaldada lainendavas kasutusjuhus kirjeldatud käitumist. [2]

Kasutamisseos <<include>> kasutusjuhust A kasutusjuhtu B näitab, et kasutusjuht A sisaldab käitumist, mis on kirjeldatud kasutusjuhus B.[2]

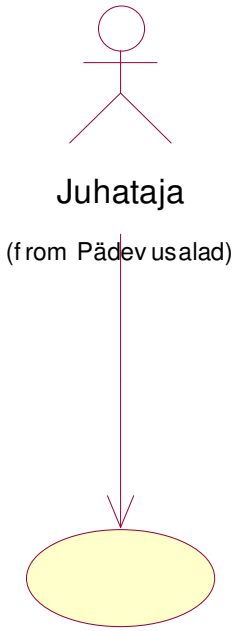
Kasutusjuhtu tähistatakse kasutusjuhtude diagrammil ellipsiga, mille all on välja toodud kasutusjuhu nimi nagu võib näha joonisel 6.



Kaupade detailaruande vaatamine

Joonis 6 Kasutusjuht "Kaupade detailaruande vaatamine" kasutusjuhtude digarammilt

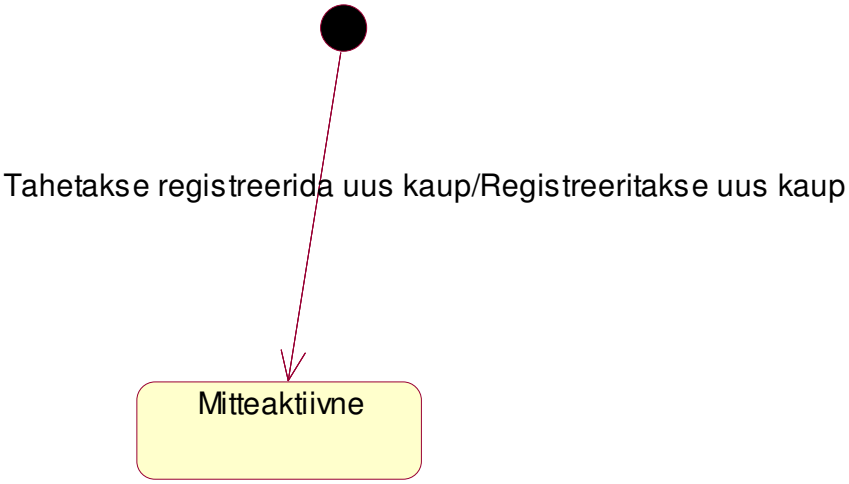
Seost kasutusjuhu ja tegutseja vahel märgitakse katkematu joonega, mille otsas on lahtine nool nagu võib näha joonisel 7.



Joonis 7 Rolli "Juhataja" kasutusjuhtude diagramm

1.3 Seisundidiagramm

UML seisundidiagramm kujutab erinevaid seisundeid, milles mingisse klassi kuuluvad objektid võivad olla ning üleminekuid seisundite vahel. Seisundidiagrammi abil saab kirjeldada klassi kuuluvate objektide kõikvõimalikud elutsüklid. Seisund näitab teatud faasi objekti käitumismudelis. Nagu tegevusdiagrammides on ka seisundidiagrammides võimalik defineerida nii algseisund kui lõpuseisund. Algseisund tähendab seda seisundit, milles objekt on siis kui ta luuakse. Lõpuseisund tähistab seda seisundit, millest enam kusagile edasi liikuda ei ole (näiteks seisund "kustutatud"). Üleminekuid seisundite vahel kutsuvad esile kas sisemised või välimised sündmused, mis omakorda algatavad süsteemis tegevusi. [10]
Algseisundit kujutatakse UML seisundidiagrammis musta ringina ning seisundit kastina, milles on seisundi nimetus. Üleminekut seisundite vahel kujutatakse suunaga joonega, mis lõppeb lahtise noolega (vaata joonis 8).

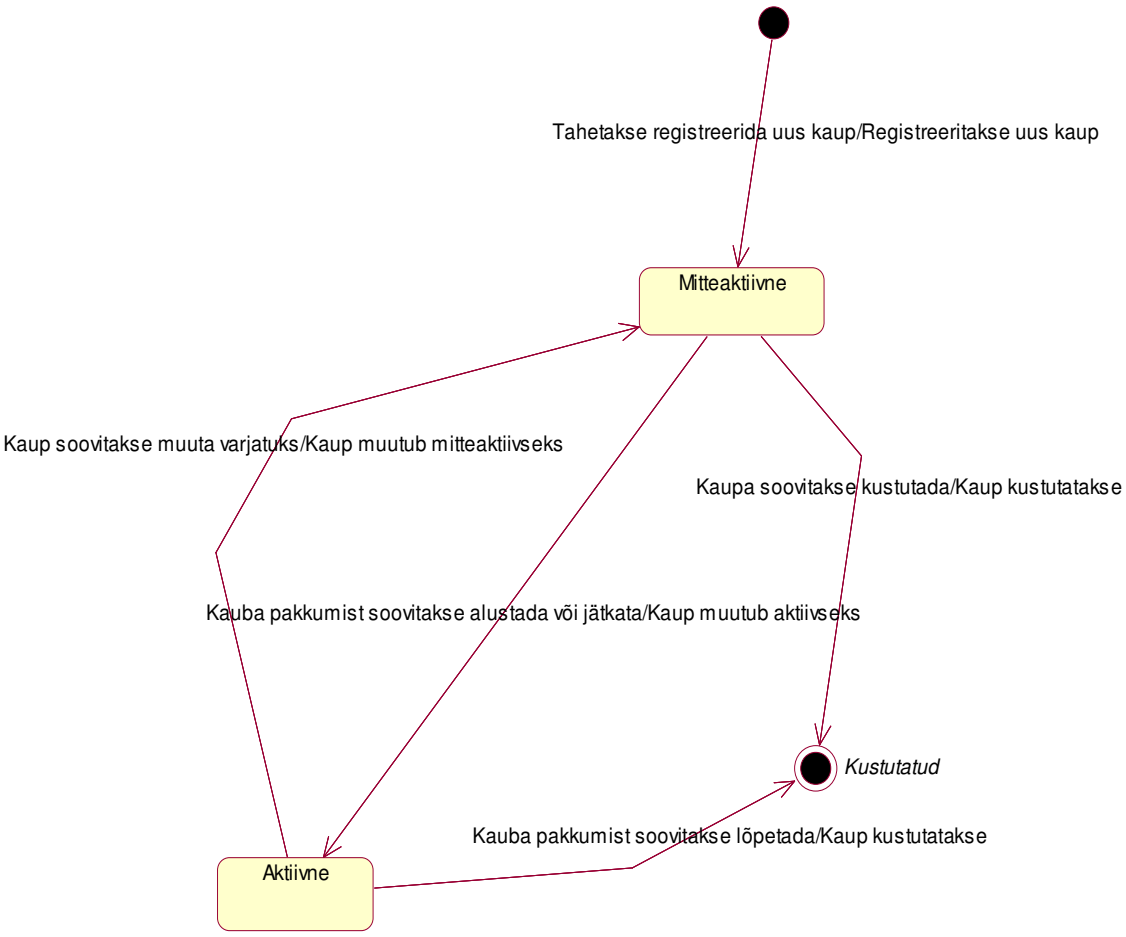


Joonis 8 Kauba algne seisund ning üleminek mitteaktiivsesse seisundisse

Üleminekujoonele lisatakse tekst, et kirjeldada, mis põhjusel antud üleminek toimub ning mis objektiga antud ülemineku käigus juhtub.
Objekti lõppseisundit kujutatakse UML diagrammis täidetut musta ringina, millel on omakorda täitmata ring ümber (vaata joonis 9).

Joonis 9 Objekti lõppseisund Kustutatud

Joonisel 10 on välja toodud antud lõputöös kasutatav kauba seisundidiagramm.



Joonis 10 Klassi *Kaup* kuuluvate objektide seisundidiagramm

1.4 CRUD maatriks

CRUD maatriks on risttabel, mille abil saab kontrollida süsteemi andmevaate ja protsesside kooskõla. Mudelite kooskõla on väga oluline, sest erinevad mudelid peavad kirjeldama sama süsteemi, mitte olema eraldiseisvad ja seostamatud pildikesed. CRUD maatriksit võib koostada erineva täpsusastmega. Antud animatsioonides tuuakse välja kuidas on omavahel seotud olemitüübid ning kasutusjuhud. CRUD maatriksi nimi tuleneb sellest, et andmete kasutamiseks on neli peamist operatsiooni, mida mõnikord kutsutakse CRUD-iks [11]:

- **C**reate (loo) – uute andmete sisestamiseks andmebaasi (SQLis vastab sellele INSERT lause)
- **R**ead (loe) – andmete lugemiseks andmebaasist (SQLis vastab sellele SELECT lause)
- **U**pdate (uuenda) – andmete muutmiseks või uuendamiseks andmebaasis (SQLis vastab sellele UPDATE lause)
- **D**elete (kustuta) – andmete kustutamiseks andmebaasist (SQLis vastab sellele DELETE lause) [11]

CRUD maatriksiga näidatakse, kuidas teatud kasutusjuhtude käigus andmebaasi osasid mõjutatakse. Järgnevalt on esitatud antud lõputöös kasutatav CRUD maatriks (Tabel 1).

1. Kauba registreerimine
   2.Kauba andmete muutmine
   3.Kauba mitteaktiivseks muutmine
   4.Kauba aktiivseks muutmine
   5.Kauba kustutamine

Kasutusjuhud **Olemitüübid**	**1**	**2**	**3**	**4**	**5**	**Kokkuvõte**
Kaubatundja						
Juhataja						
Klassifikaator		**R**	**R**	**R**	**R**	**R**
Kaup	**C**	**RU**	**RU**	**RU**	**RD**	**CRUD**
Kauba_seisundi_liik	**R**		**R**	**R**	**R**	**R**
Pakendi_tüüp	**R**					**R**

Tabel 1 CRUD maatriks

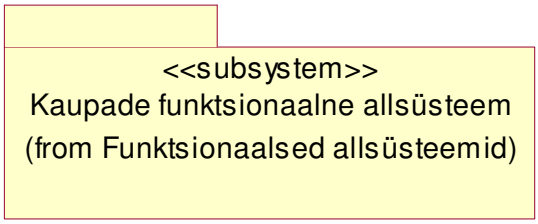
1.5 Äriarhitektuuri diagramm

Äriarhitektuuri mudel on ettevõtte plaan, mis annab üldise ettekujutuse ettevõttest ning mida kasutatakse strateegiliste eesmärkide ja taktiliste nõudmiste välja töötamiseks. Äriarhitektuuri skeem esitab ettevõtte funktsionaalset, organisatsioonilist ja andmekeskset ülesehitust, kirjeldades selle eritüübiliste alamosade sõltuvusi ning pannes ühtlasi paika süsteemi skoobi.[12]

Äriarhitektuuri mudel koosneb järgnevatest osadest.

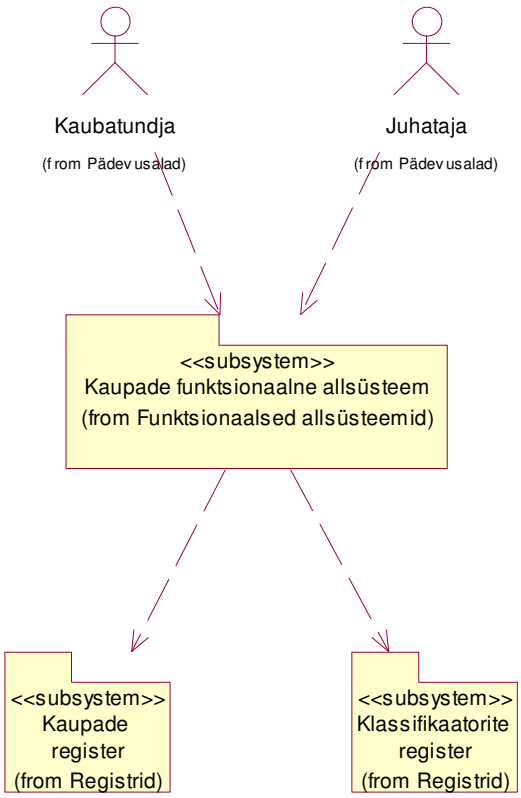
- Organisatiooniline pädevusalade vaade – pädevusala allsüsteemid
- Funktsionaalne vaade – funktsionaalsed allsüsteemid
- Informatsiooniline vaade – registrid [12]

Äriarhitektuuri skeemil kujutatakse pädevusalade allsüsteeme tegutsejatena, mis lähevad hiljem käiku kasutusjuhtude diagrammide koostamisel (vaata joonis 5). Funktsionaalseid allsüsteeme ning registreid kujutatakse UML diagrammides pakettidena, milles on kirjas funktsionaalse allsüsteemi või registri nimetus nagu kujutatud joonisel 1.5.1.



Joonis 11 Kaupade funktsionaalne allsüsteem äriarhitektuuri skeemilt

Äriarhitektuuri skeemi esitatakse kihiti, alustades pädevusaladest, liikudes edasi funktsionaalsetele allsüsteemidele ning lõpetades registritega. Käesolevas lõputöös kasutatavat äriarhitektuuri süsteemi võib näha joonisel 12.



Joonis 12 Äriarhitektuuri skeem

2. Ülevaade MS PowerPointist ja selle võimalustest

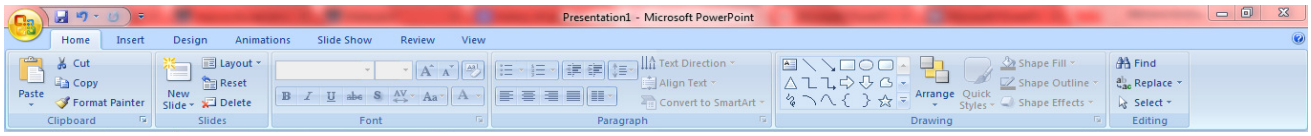
2.1 MS PowerPointi ajalugu

MS PowerPoint on virtuaalsete presentatsioonide loomiseks arendatud tarkvara. Selle looja on Berkley õpilane Robert Gaskins. Tema idee oli arendada kergesti kasutatav esitluste loomise programm, mis põhineks slaididel. Oma tööd teel PowerPointini alustas ta väikeses tarkvarafirmas Forethought, Inc koos tarkvaraarendaja Dennis Austiniga. Seda firmat ei peetud küll algselt parimaks paigaks, kus innovatiivset tarkvara luua, sest tegemist oli väikese Silicon Valley ettevõttega, millel ei läinud just kõige paremini. Selgus aga, et Forethought, Inc sai Gaskinski jaoks suurepäraseks kohaks, kus oma idee ellu viia.[4] PowerPoint, esialgse nimega Presenter anti algselt välja Apple Machintoshi jaoks 1987. aastal. Sama aasta juunis ostis Microsoft õigused PowePointile 14 miljoni USA dollari eest. PowerPointi ostmine oli üks Microsoft Corporation'i esimesi suurimaid oste. Esmalt oli PowerPoint mõeldud äriliseks kasutamiseks, kuid leidis kiiresti laialdasemat kasutust nii koolides kui tavakasutajate käes. Algselt eraldiseisvna programmina välja töötatud PowerPoint sai domineerivaks presentatsioonide loomise tarkvaraks tänu selle kaasamisele populaarsesse MS Office pakketti. [3]

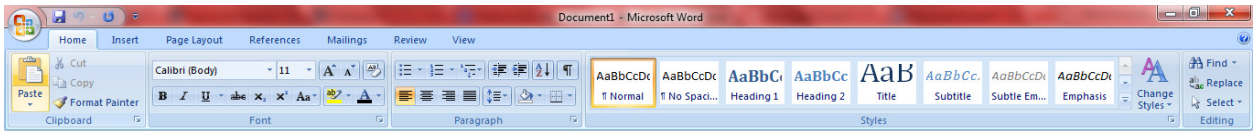
2.2 Animatsioonid MS PowerPointis

Lisaks lihtsale esitluste koostamisele pakub MS PowerPoint ka laialdast valikut animatsioone, mida saab rakendada nii slaididele kui üksikutele objektidele nendel slaididel. Antud ülevaade koostatakse MS PowerPoint 2007 näitel, kuna just seda versiooni kasutab autor oma sülearvutis. Tegelikkuses on käesoleva lõputöö kirjutamise ajal 2015. aastal väljas ka MS PowerPoint 2013.

MS PowerPoint on väga kasutajasõbralik programm. Kuuludes MS Office pakketti on sellel sarnane kujundus ka teiste, kasutajate poolt hinnatud programmidega nagu MS Word ja MS Excel. Võrdluseks on toodud pildid Ms PowerPointi 2007 ja Ms Word 2007 peamenüüdest (vaata Joonis 13 ja Joonis 14).

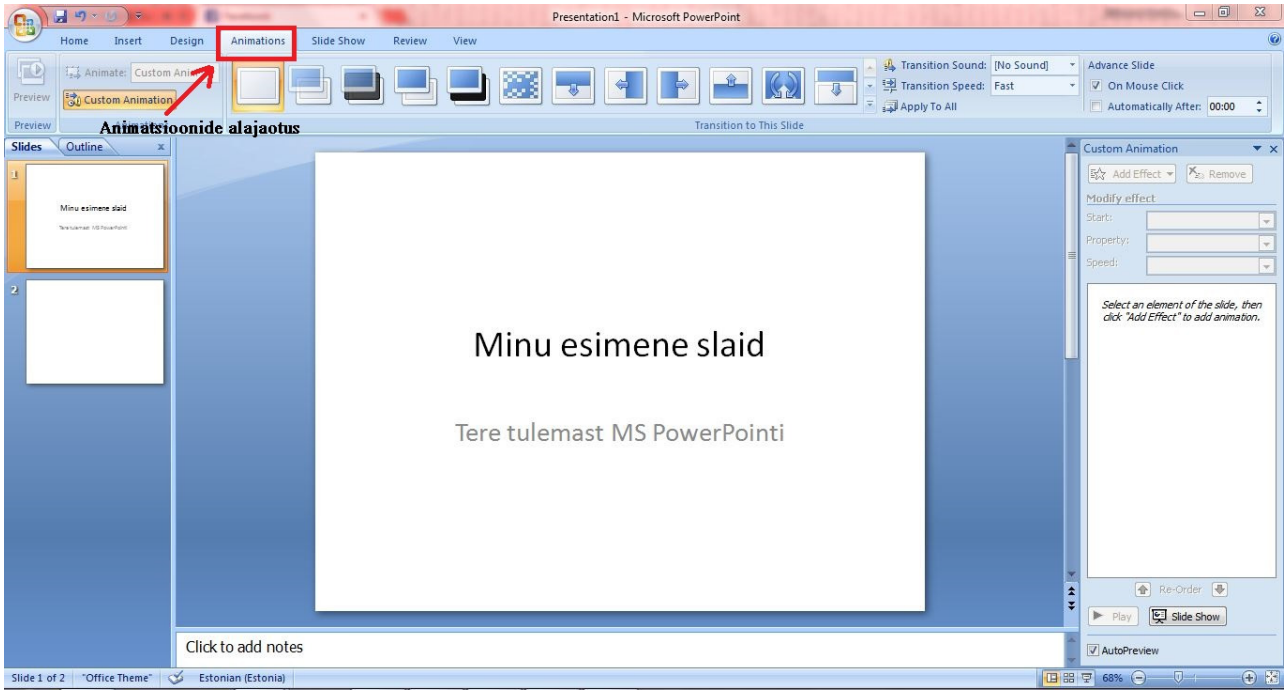


Joonis 13 MS PowerPoint 2007 peamenüü



Joonis 14 MS Word 2007 peamenüü

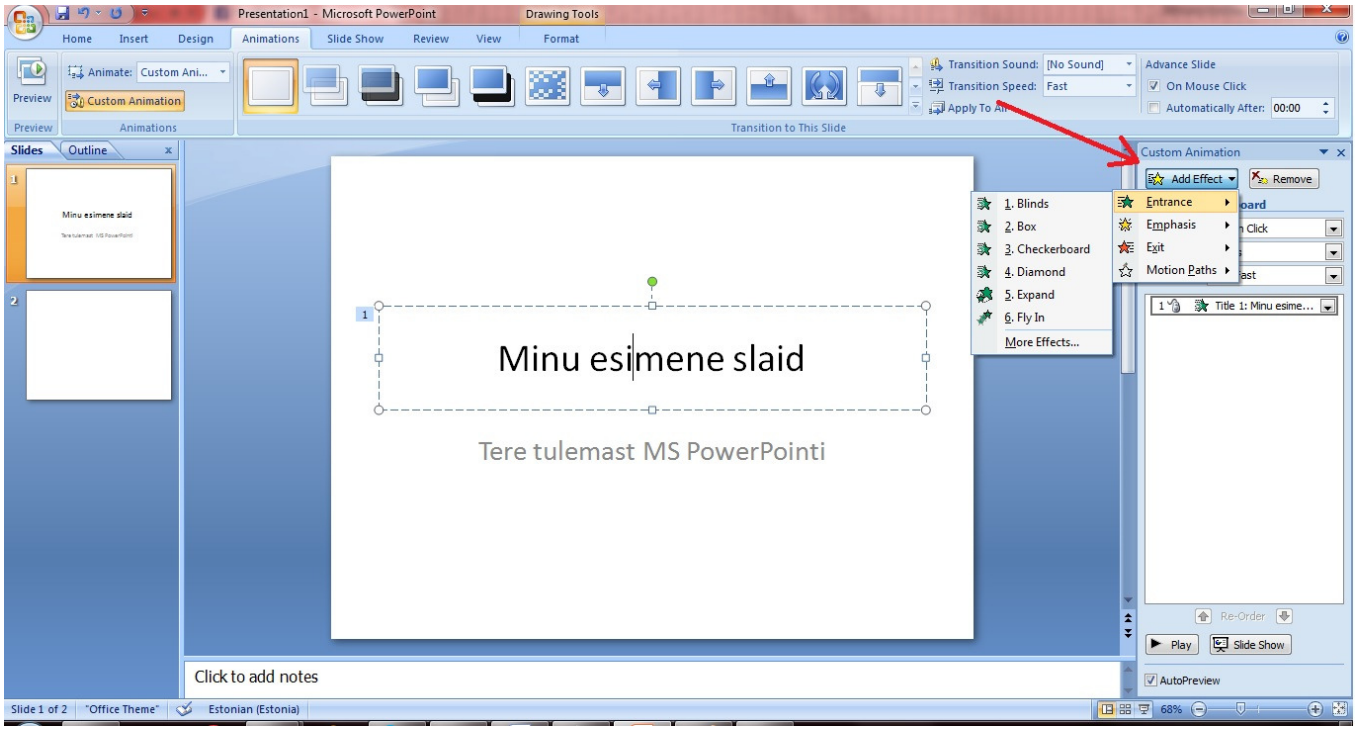
Animatsiooni lisamine oma slaidile on üsna kerge. MS PowerPoint 2007-l on selle jaoks menüüs eraldi jaotus (vaata Joonis 15).



Joonis 15 MS PowerPoint 2007 animatsioonide alajaotus

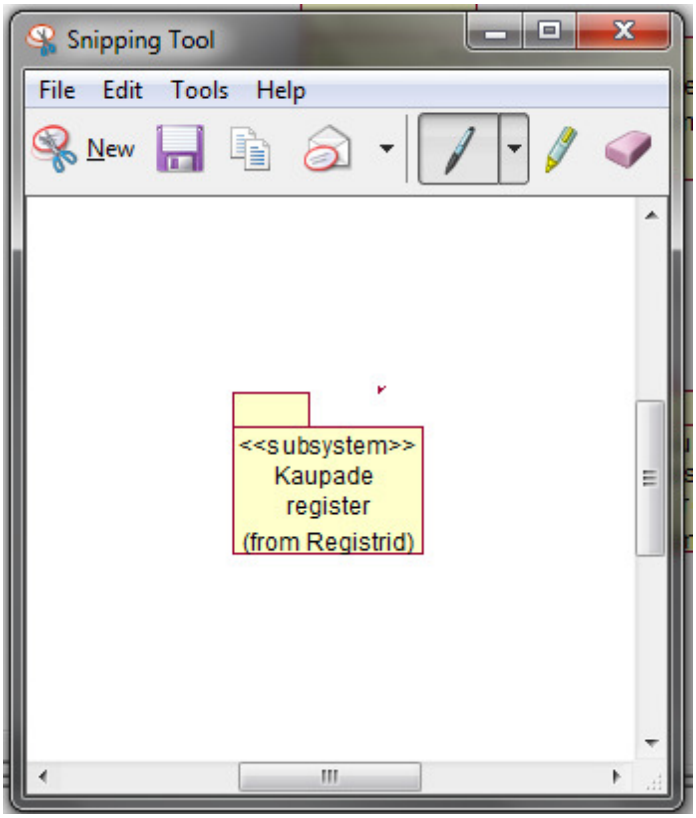
Slaidide animeerimise juures on võimalik valida 58 erineva animatsiooni vahel. Lisaks sellele saab slaidile lisada ka helisid ja määrata slaidide vaheldumise kiirust.

Objektide animeerimiseks on veel enam võimalusi. *Custom animation* menüüst on võimalik objektile valida nii sisenemise animatsioon, slaidilt kadumise animatsioon, liikumise teekond mööda slaidi kui animatsiooni kiirus (vaata Joonis 16).



Joonis 16 MS PowerPoint 2007 Custom Animation menüü objektidele

Käesoleva lõputöö raames luuakse standardsest slaidikomplektist veidi erineva animatsioon. Varasemalt valmis tehtud UML diagrammid annavad selle jaoks hea aluse. Suuremalt jaolt oli võimalik UML diagrammi MS PowerPointi lihtsalt kopeerida. Animatsiooni tegemiseks kasutatakse diagrammi osadena, mis tähendab, et ei saa korraga kopeerida nii tegutsejaid/klasse kui nende vahelisi seoseid. Selle jaoks kasutatakse Windows 7 programmi Snipping Tool, mis lubab sarnaselt Print Screen võimalusele nii öelda "välja lõigata" vajaliku pildi. Antud programmi kuva võib näha joonisel 17.



Joonis 17 Windows 7 Snipping Tool

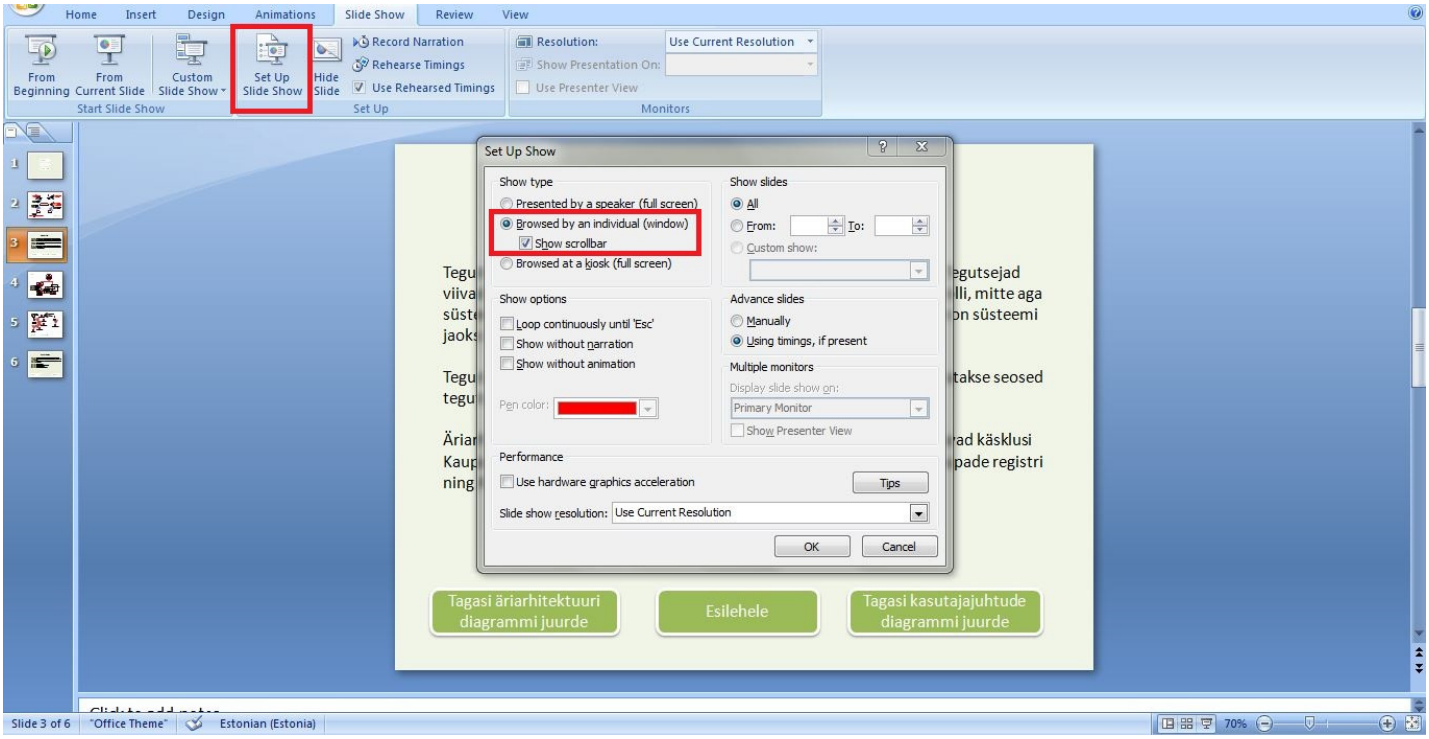
Seega kasutatakse lõputöös lisaks MS PowerPointi võimalustele ka teisi Windows operatsioonisüsteemi poolt pakutavaid.

Tänu MS PowerPointi võimalusele kasutada slaididel ka hüperlinke, on ühelt slaidilt teisele liikumine kerge ja slaidide järjekord esitluses ei oma erilist tähtsust. Joonisel 18 on võimalik näha hüperlingi sisestamise kohta MS PowerPoint menüü *Insert* alajaotuse all.

Joonis 18 MS PowerPoint Hyperlink võimalus

Funktsiooni *Hyperlink* on võimalik kasutada nii pildi kui teksti puhul. Hüperlink annab võimaluse liikuda slaidilt slaidile, suunata kasutaja veebilehele või liikuda presentatsioonide vahel.

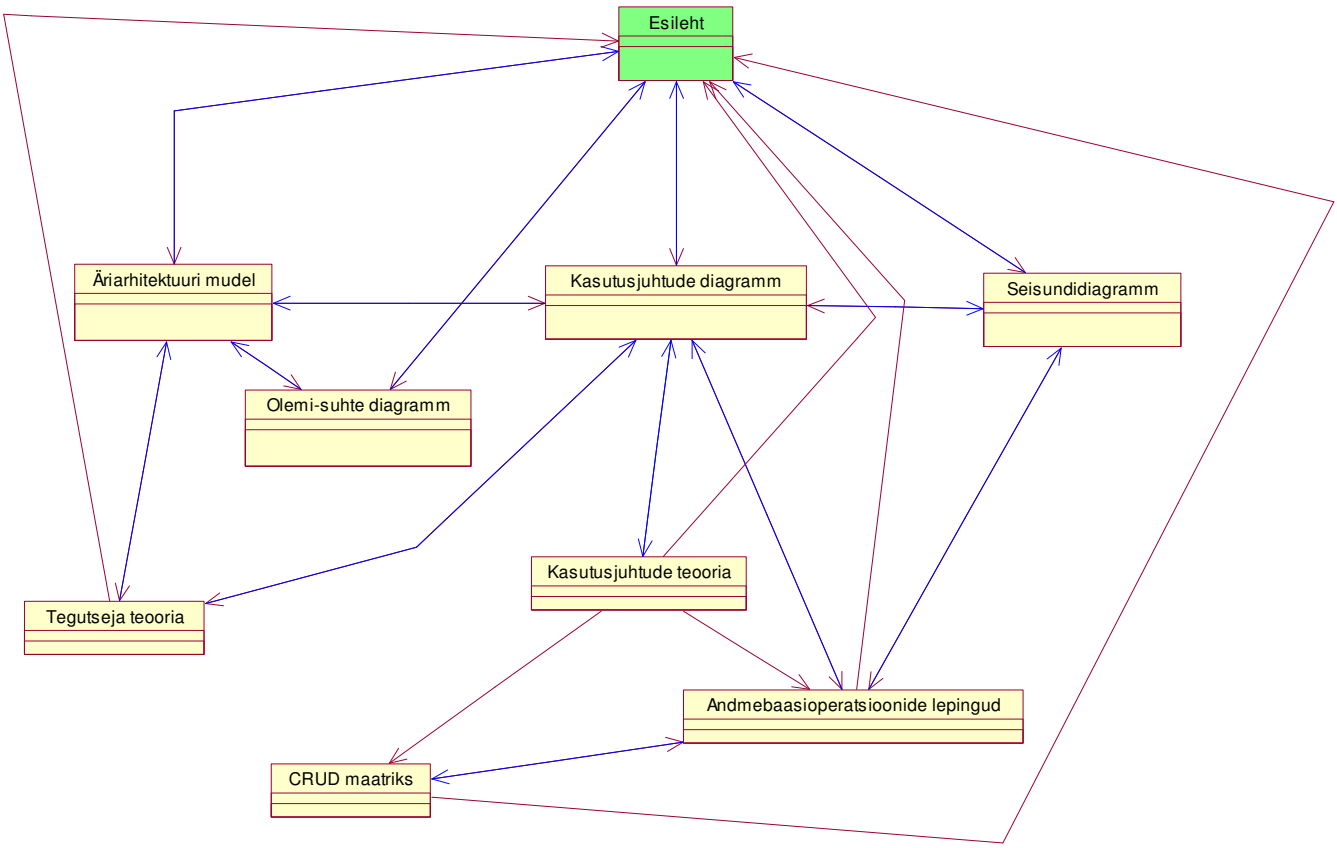
Antud lõputöös kasutatakse palju hüperlinkide võimalust. Selleks, et slaidid ei vahelduks hiireklõpsuga, mis animatsiooni ebatäpseks muudaks, kasutati MS PowerPoint'i *Set Up Show* menüüs võimalust *Browsed By an individual window*. See keelab slaidide vaheldumise hiireklõpsul, kuid varasemalt määratud hüperlinkide funktsionaalsus jääb alles. Antud menüüd võib näha joonisel 19.



Joonis 19 MS PowerPoint Set Up Show menüüjaotus

3. Animatsioonide loomine MS PowerPointis

Käesoleva lõputöö raames alustati animatsioonide loomist sellest, et loodi UML klassidiagrammi baasil slaidide navigatsiooni kirjeldav mudel. Igale slaidile vastab üks klass. Ühest klassist läheb teise nool, kui nendelt slaididelt on võimalik sellises suunas ühelt teisele liikuda. Sinisega on esitatud kahesuunalised nooled, mis tähendab, et mõlemalt slaidilt on võimalik navigeerida teineteisele. Diagramm on esitatud joonisel 20.



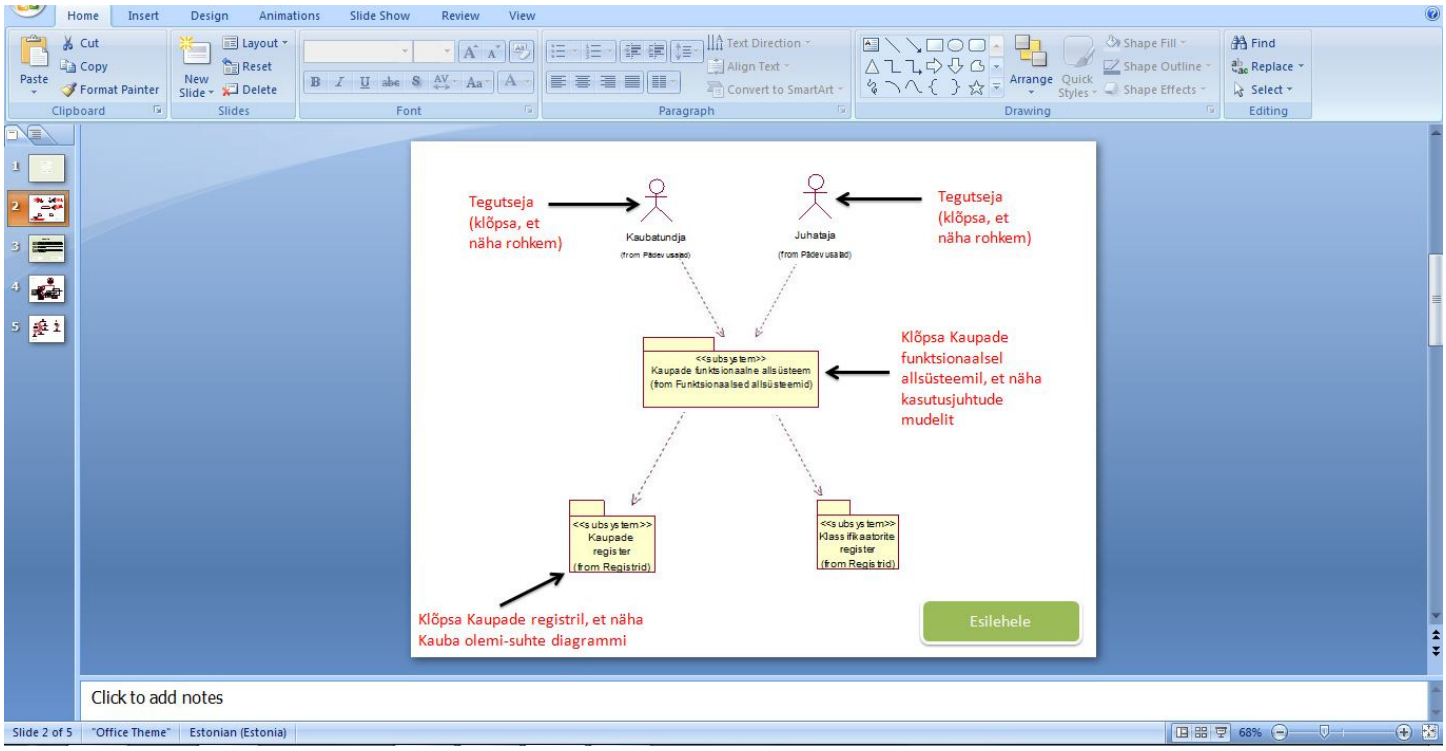
Joonis 20 Slaidide navigatsioonidiagramm

Slaidide loomist alustas autor äriarhitektuuri diagrammist. Autor valis selle, kuna antud mudeli kaudu oli mugav suunata kasutajat nii olemi-suhte diagrammile kui ka kasutusjuhtude diagrammile. Nagu punktis 2.2 mainitud, kasutati Snipping Tool'i, et UML diagramm nii öelda osadeks jagada ja need osad kleepida MS PowerPoint slaidile. MS PowerPoint pakub sarnaselt MS Wordile võimalust lisada oma slaidile erinevaid kujundeid (ring, ristkülik, ruut, ellips jms) ja jooni ning nooli. Selleks, et ühendada vajalikud viidad neile vastavate diagrammiosadega, kasutati selliseid jooni. Joonise 21 võib näha esialgset slaidi koos

viitadega, mis annavad märku, et ühel või teisel diagrammi osal klõpsates on võimalik liikuda vastavalt tegutseja rolli kirjeldusele, olemi-suhte diagrammile või kasutusjuhtude diagrammile.

Slaidile lisati ka nupp "Esilehele", mis varustati esitlusesisese hüperlingiga slaidile Esileht, kuhu lisati sarnased nupud, mis suunavad kasutaja teda huvitavatele diagrammidele ilma ebavajalike vaheslaididete. Selline lähenemisviis muudab antud slaidikomplekti rohkem animatsiooni ning vähem tavalise esitluse sarnaseks.

Klõpsates tegutsejal, mida esitav UML diagrammi osa on samuti hüperlingiga varustatud, suunatakse kasutaja slaidile, kus on lühidalt kirjeldatud mõistet „tegutseja“. Sealt on võimalik navigeerida kolme kohta - esilehele, äriarhitektuuri diagrammi juurde ning kasutusjuhtude diagrammi juurde, kahte viimasesse seetõttu, et just nende diagrammidega on tegutseja seotud.

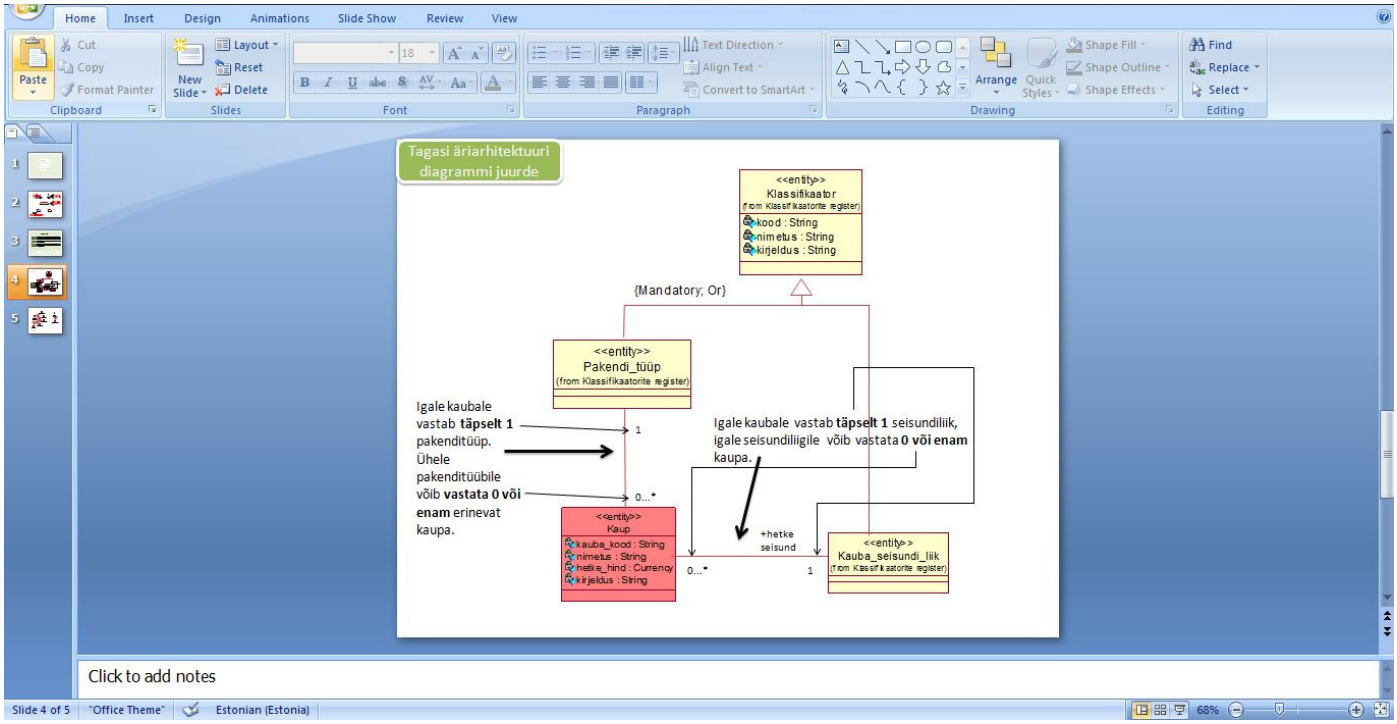


Joonis 21 Äriarhitektuuri slaid koos viitadega teistele diagrammidele

MS PowerPoint pakub slaidi tausta kohandamiseks erinevaid lahendusi. Võimalik on taustaks seada pilt, värv, muster ja palju muud. Taustavärvi muutmine on kiire ja kerge – parem hiireklõps slaidil avab menüü, milles viimaseks valikuks on *Format Background*. Tegutsejat tutvustavale slaidile on määratud diagramme kujutavatest slaididest erinev taustavärv. Eesmärk on eristada vaid teooriat sisaldavaid slaide diagramme sisaldavatest slaididest. Lisaks sellele on diagramme kujutavate slaidide taustavärviks valge, et diagrammid paremini

esile tuleksid ning taustavärv kasutajat segama ei hakkaks. Sarnaselt tegutsejat tutvustavale slaidile luuakse ka mõistet „kasutusjuht“ tutvustav slaid.

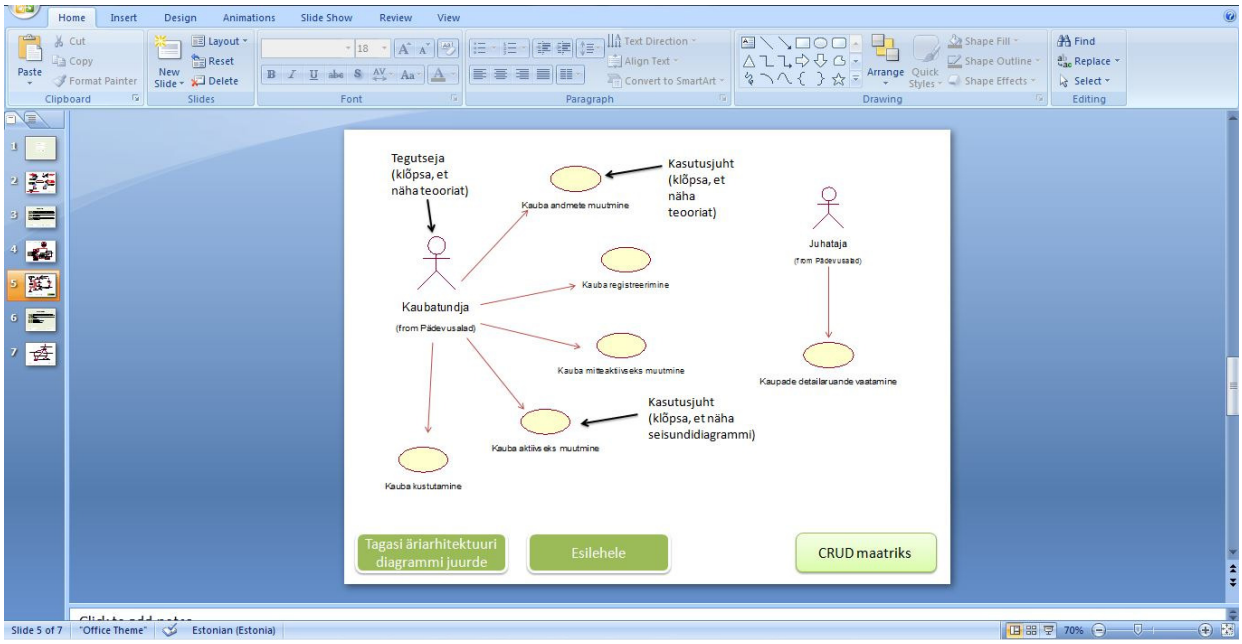
Järgmise sammuna loob autor slaidi olemi-suhte diagrammile. Ka sellele slaidile lisatakse MS PowerPoint poolt pakutavaid animatsioonivõimalusi. Olemi-suhte diagrammi slaidile lisatakse ka selgitavaid kommentaare diagrammi kohta. Animatsioonid lisatakse just selgitavatele kommentaaridele. Antud slaidi võib näha Joonis 22.



Joonis 22 Olemi-suhte diagrammi slaid koos kommentaaridega

Olemi-suhte diagrammi slaidi animeerimise juures kasutakse *Custom Animations* valikut. Nii kommentaaride tekstile kui ka suunavatele nooltele *Entrance* (sisenemine) lisatakse animatsioonid ja määratakse nende animatsioonide toimumiskiiruseks *Fast* (kiire). Samuti muudetakse ära see, et animatsioonid toimuvad vaid hiireklõpsuga, mis on animatsioonide loomisel vaikimisi seadistus. Muudatuse tulemusel algab iga animatsioon eelmise lõppedes automaatselt.

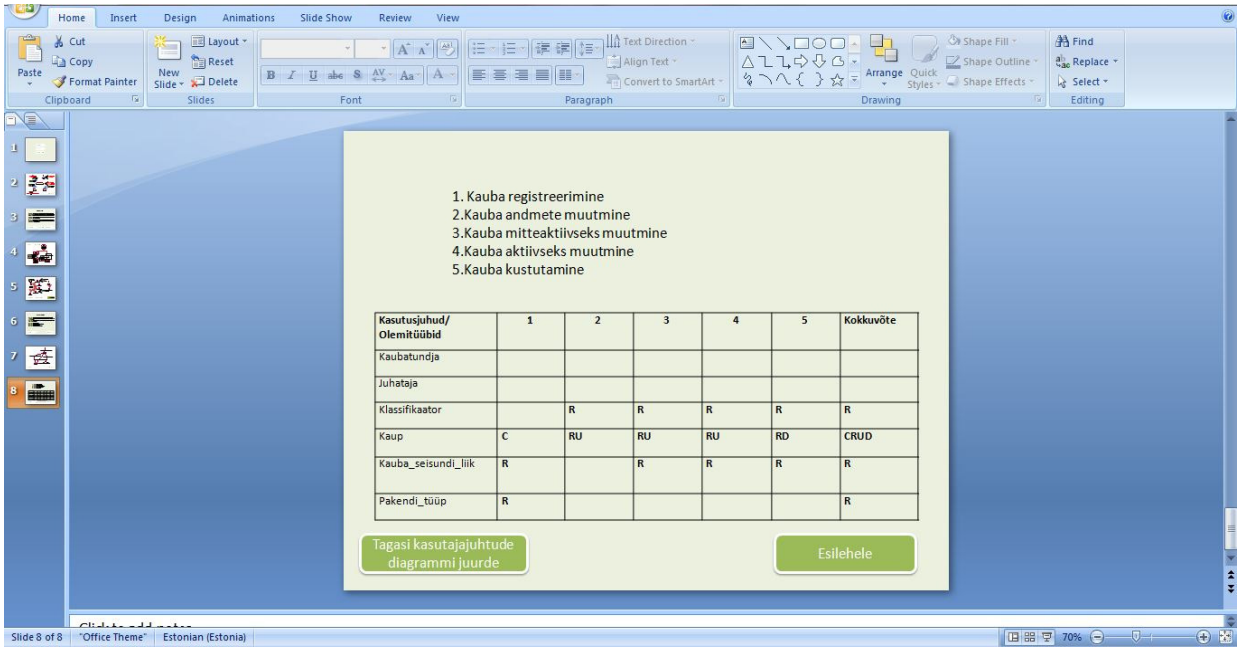
Järgmiseks luuakse slaid kasutusjuhtude diagrammile. Kasutusjuhtude diagrammil kasutatakse mitmel korral hüperlinkide võimalust. Antud diagrammilt on võimalik navigeerida slaidile, mis kuvab teooriaosa tegutseja kohta, slaidile, mis kuvab teooriaosa kasutusjuhtude kohta, seisundidiagrammile, andmebaaside lepingutele ning CRUD maatriksile. Seda slaidi on võimalik näha joonisel 23.



Joonis 23 Kasutusjuhtude diagram

CRUD maatriksile suunav nupp on erinev navigeerimisnuppudest, mida võib leida igalt slaidilt, eesmärgiga tuua sellele kasutaja tähelepanu. Kasutusjuhtude diagrammi slaidile, sarnaselt äriarhitektuuri ning olemi-suhte diagrammi slaididega, on lisatud *Entrance* (sisenemine) animatsioonid kiirusega *Fast* (kiire). Animatsioonid on lisatud kasutajat suunavatele viidetele. Nuppudele ning diagrammile endale animatsioone ei lisata.

Sarnaselt varasemalt loodud tegutsejat ja kasutusjuhte tutvustavatele slaididele saab ka CRUD maatriksit kujutav slaid diagrammislaididest erineva taustavärvi. Ühelegi teooriaslaidile animatsioone lisatud ei ole, et mitte kasutaja tähelepanu teooria sisult liigselt kõrvale tõmmata. CRUD maatriksit kujutavale slaidile on lisatud kaks nuppu ehk kaks hüperlingiga varustatud ristkülikut, mis suunavad kasutaja tagasi kasutusjuhtude diagrammile või esilehele. CRUD maatriksit kujutavat slaidi on võimalik näha joonisel 24.



Kasutusjuhud/ Olemitüübid	1	2	3	4	5	Kokkuvõte
Kaubatundja						
Juhataja						
Klassifikaator		R	R	R	R	R
Kaup	C	RU	RU	RU	RD	CRUD
Kauba_seisundi_liik	R		R	R	R	R
Pakendi_tüüp	R					R

Joonis 24 CRUD maatriksit esitav slaid

4. Ülevaade programmeerimiskeelest Java ja Java Swing'ist

4.1 Ülevaade programmeerimiskeelest Java

Programmeerimiskeel Java (edaspidi Java) on objektorienteeritud programmeerimiskeel. Erinevalt varasematest programmeerimiskeeltest, mis on üldiselt disainitud komplieeruma masinkoodiks, on Java disainitud komplieeruma baitkoodiks, mida jooksutatakse seejärel kasutades Java virtuaalmasinat (JVM). Java süntaks sarnaneb paljuski C ja C++ programmeerimiskeeltele. JavaScript ja Java on oma vahel vaid põgusalt seotud, vaatamata nende sarnastele nimedele ja C-keeltega sarnanevale süntaksile.[5]

Java sai alguse 1991. aasta juunis, kui James Gosling Sun Microsystem'ist alustas projektiga "Oak", eesmärgiga luua virtuaalmasin ja programmeerimiskeel, mis oleks tuttavliku C-sarnase süntaksiga, kuid ühilduvam ning lihtsam. Esimene avalik Java versioon, Java 1.0 avaldati aastal 1995. See oli võrdlemisi turvaline ning turvalisusesätted konfigureeritavad. Suuremad veebibrauserid kaasasid Java varsti pärast seda oma vaikimis konfiguratsiooni applet vormis. Uued versioonid väikestele ja suurtele platvormidele (J2EE ja J2ME) tõi kaasa Java 2. [5]

Java loomisel lähtuti viiest põhiprintsiibist:

1. Kasutama peaks objektorienteeritud metoodikat

2.Sama programmi peaks saama käivitada erinevatel operatsioonisüsteemidel

3.See peaks sisaldama sisseehitatud tuge arvutivõrkude kasutamiseks

4.See peaks olema disainitud koodi turvaliselt käivitama

5.Seda peaks olema kerge kasutada, valides Java jaoks need osad teistest objektorienteeritud keeltest, mida peeti heaks [5]

Objektorienteeritus Javas tähendab seda, et kõike esitatakse Javas objektidena (andmed ja kood on kombineeritud) välja arvatud atomaarsed andmetüübid nagu tähed, numbrid, tõeväärtustüüpi väärtused. Javas kirjutatakse kogu kood klasside sisse. [5]

Joonisel 26 tuuakse välja väga lihtne Java koodinäide.

```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello world, this is Java!");
        System.out.println("Check out my method");
```

```
        doSomething("method");
    }
    public static void doSomething(String word){
        System.out.println(word);
    }
}
```

Joonis 26 Koodinäide keeles Java

`public class HelloWorld` on siin klass, kuhu sisse defineeritakse vajalikud atribuudid ja meetodid. `public static void main(String[] args)` on Java klassi põhiline (main) meetod, mida programmi käivitamisel tavaliselt jooksutatakse. Seal kutsutakse välja ehk käivitatakse teised meetodi nagu on tehtud meetodiga `doSomething("method")`.

`public static void doSomething(String word)` on Java meetod, mis prindib lihtsalt argumendina ette antud sõna konsooli. Sellise programmijupi puhul ei käivituks ilma *main* meetodita ka *doSomething* meetod. Koodi käivitamisel trükitakse konsooli tulemused. Esitatud näite tulemusi võib näha joonisel 27.

Joonis 27 Näiteprogrammi tulemused konsoolis

4.2 Ülevaade Java Swing'ist

Swing on graafilise kasutajaliidese teek Java SE platvormi jaoks. Swing projekti üldine eesmärk oli ehitada laiahaardelisi graafilise kasutajaliidese komponente, et anda arendajatele võimalus kiiresti arendada Java baasil kasutajaliideseid. Seetõttu pani Swingi

arendusmeeskond tarkvaraarenduse varajastes etappides paika eesmärgid, mille poole liikuda. [6] Need eesmärgid ütlesid järgmist.

1. Swing peab olema täielikult realiseeritud Java keeles, et säilitada platvormide vahelist sobivust ning kergemat hallatavust

2. Swing peab andma ühe programmiliidese (API) mitme erineva disainivõimalusega (*look-and-feel*), et arendajad ei peaks piirduma ühe disainiga

3. Swing peab andma võimaluse kasutada mudel-vaade-kontroller (MVC) stiilis programmeerimist ilma, et see oleks kõrgeima taseme API-s nõutud

4. Swing peab kasutama JavaBeans disainipõhimõtteid, et tagada komponentide töötamine integreeritud arenduskeskkondades ning kompileerimistööristades

5. Swing peab võimaldama sobivust AWT (*Abstract Window Toolkit*) API'dega, et portimist lihtsustada [6]

Swing API kasutab MVC arhitektuuri järgnevalt.

- Mudel (*model*) esitab komponendi andmeid
- Vaade (*view*) esitab komponendi andmeid visuaalselt
- Kontroller (*controller*) võtab kasutaja sisendi vaatest ja peegeldab selle muutusteks komponendi andmetes
- Swingi komponentide puhul asetseb mudel eraldi ning vaade ja kontroller on sisalduvad kasutajaliidese elementides. Seetõttu on Swingis võimalik kasutada erinevaid *look-and-feel* teemasid [7]

Swingi komponendid on operatsioonisüsteemi API-st eraldiseisvad, sest Swingi väljakutsed põhinevad peaagu täielikult Java koodil. Swing pakub palju erinevaid võimalusi ja mooduleid, mida kasutajaliidesele lisada ja need on kergelt kohandatavad, kuna Swingi visuaalne välimus on eraldiseisev sisemisest esitusest. [7]

Järgnevalt toob autor välja joonisel 28 mõne lihtsa näite Java Swing'ist.

```
package animation;

import java.awt.Color;
import java.awt.SystemColor;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class MainFrame extends JFrame {

	public static void main(String args[]) {
```

```
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                mainGUI();
            }
        });
    }

    public static void mainGUI(){
        final JFrame frame = new JFrame("Java Swing");
        frame.setSize(400, 300);
        frame.setBackground(Color.PINK);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel welcomePanel = new JPanel();
        welcomePanel.setBackground(SystemColor.PINK);
        frame.getContentPane().add(welcomePanel);
        frame.setVisible(true);

        JLabel welcomeText = new JLabel("Welcome to Java Swing!");
        welcomePanel.add(welcomeText);


    }

}
```

Joonis 28 Java Swing koodinäide

Antud näide on üks kõige põhilisemaid Swing'i näiteid. Meetod `public static void mainGUI()` sees defineeritakse JFrame, mis on sisuliselt kasutajaliidese põhi - sinna hakatakse lisama kõiki ülejäänud komponente. JPanel on paneel, mis lisatakse JFrame'ile ja mille abil saab seada erinevate komponentide paigutust kasutajaliidesel. JLabel on nii öelda silt, mille abil lisada kasutajaliidesele teksti.

Nagu Javale omane, kutsutakse meetodeid välja main meetodis. Selles programmijupis luuakse main meetodis uus *Java Runnable* objekt ja käivitatakse see. *run()* meetodit kasutatakse peamiselt Java lõimede (*thread*) puhul. Milline näeb välja antud koodiga implementeeritud kasutajaliides saab näha joonisel 29.

Joonis 29 Swing'is realiseeritud lihtne kasutajaliides

Kasutajaliidesele on võimalik lisada veel palju erinevaid komponente. Esitatud näide on väga põhiline ja hõlmab vaid üliväikest osa Java Swingi võimalustest.

5. Animatsiooni loomine Java Swing tehnoloogial

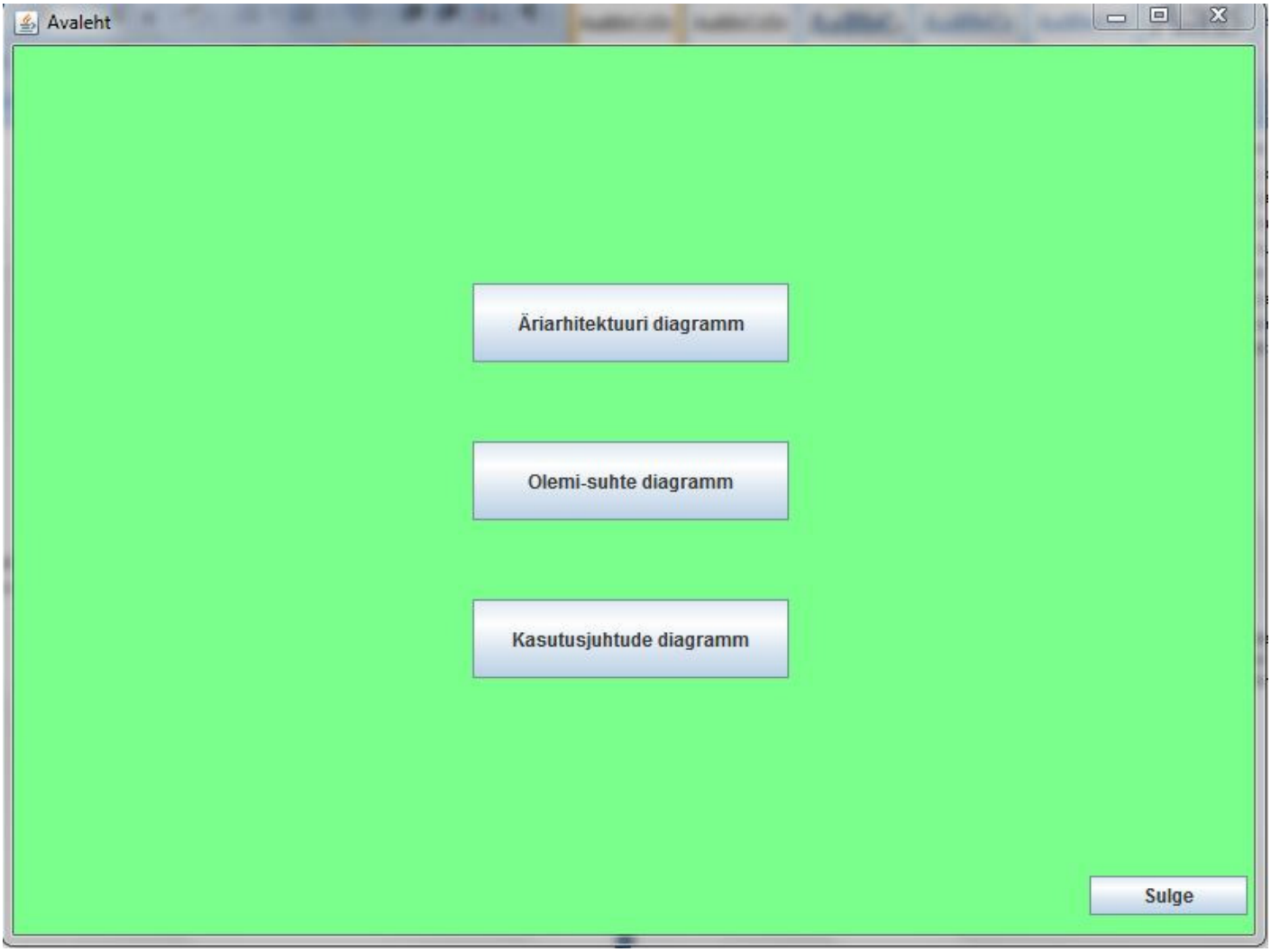
Käesolevas töös kasutatakse programmeerimiseks Eclipse Java EE IDE versiooni Luna (4.4.0). Selleks, et kasutada Swing API't, ei ole vaja Javale eraldi teeke lisada ning saab kohe programmeerimise juurde asuda. Eclipse pakub Swing'iga arendamiseks ka sellist lisa nagu *WindowBuilder*, mis hõlbustab komponentide paigutamist ekraanil. Käesoleva lõputöö raames aga *WindowBuilderit* ei kasutata ning kogu kood kirjutatakse käsitsi. Autor alustab programmi kirjutamist esilehest. Eesmärk on teha käesolev animatsioon võimalikult sarnane MS PowerPointis realiseeritule. Luuakse uue Java klassi nimega *MainFrame*, kuhu lisatakse järgemööda kõik vajalikud meetodid ning komponendid esilehe jaoks. Nagu ka varem realiseeritud MS PowerPointi animatsioonil, on esilehel nupud äriarhitektuuri diagrammi, kasutusjuhtude diagrammi ning olemi-suhte diagrammi jaoks. Lisaks on esilehel ka nupp "Sulge", mis programmi töö lõpetab ning Swing akna sulgeb. Iga diagrammi jaoks on tarvis luua uus aken sarnaselt esilehe loomisele. Navigatsioon nende lehtede vahel toimub nuppudele lisatud ActionListener'ide abil. Näiteks on nupul "Äriarhitektuuri diagramm" küljes järgnev ActionListener:

```
buttonAri.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                      frame.dispose();
                      try {
                            Ariarhitektuur.main(null);
                      } catch (Exception e1) {
                            // TODO Auto-generated catch block
                            e1.printStackTrace();
                      }

                }
});
```

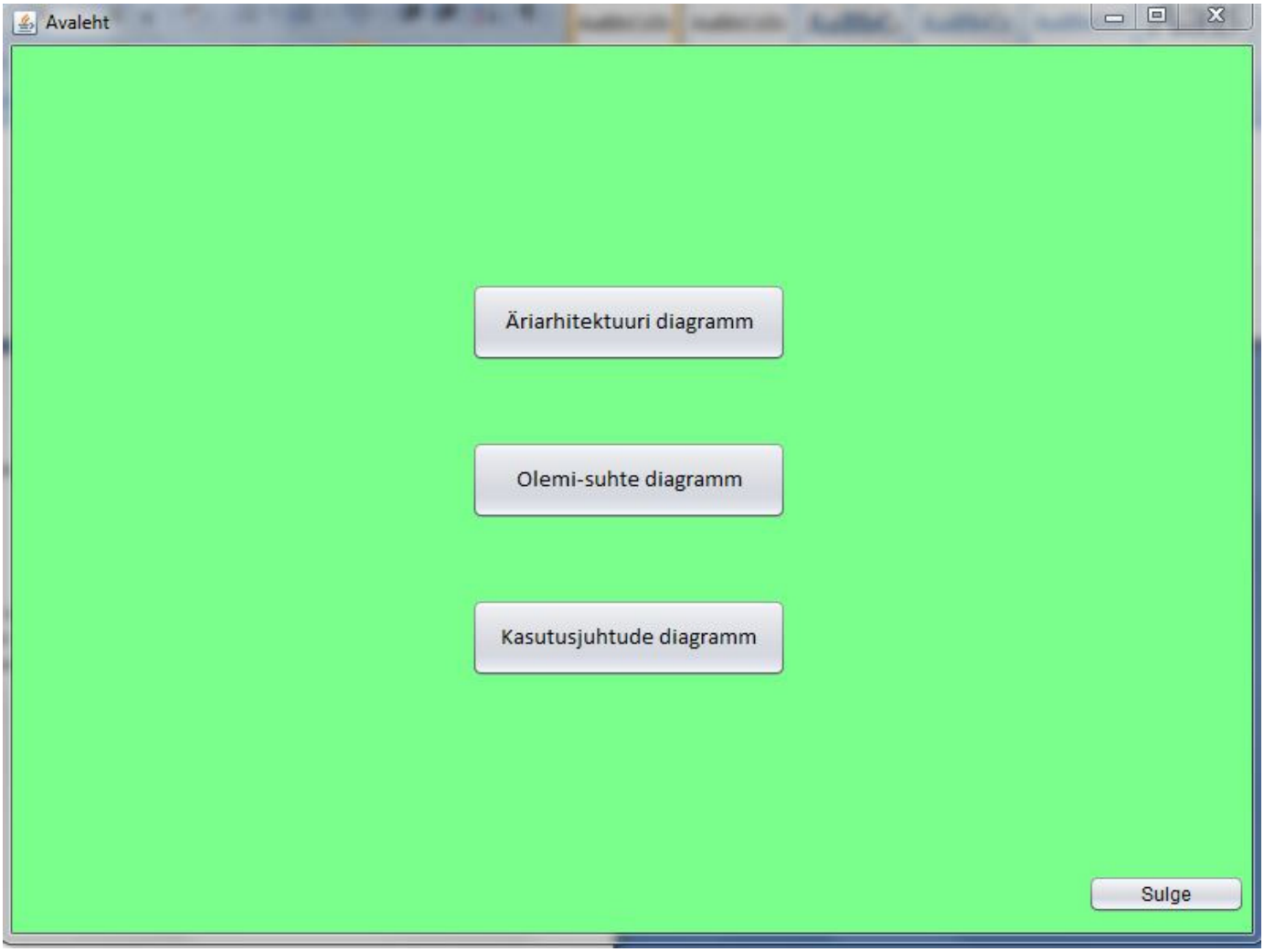
Esitatud kood ütleb programmile, et nupu `buttonAri` vajutamisel tuleb käesolev JFrame nimega frame sulgeda ning üritada käivitada klassis *Ariarhitektuur* defineeritud *main* meetod, millesse on lisatud käsklus uue raami avamiseks. Kui see mingil põhjusel ei õnnestu, siis kuvatakse konsooli teade erandi kohta. Antud esilehele on lisatud ka Java Swing'i *look-and-feel* (kujundus) nimega *Nimbus* [8]:

```
try {
for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
                    if ("Nimbus".equals(info.getName())) {

      UIManager.setLookAndFeel(info.getClassName());
                            break;
                    }
                }
            } catch (Exception e) {
                System.out.println("Theme Nimbus not available!");
                                                        }
```

Sellise koodi lisamine programmi *main* meetodisse kutsub Java Swing'i vaikimisi kujunduse asemel välja kujundusega nimega "Nimbus". Esilehe üldist kujundust võib ilma Nimbuseta ning Nimbusega näha joonistel 30 ja 31.



Joonis 30 Esilehe kujundus ilma kujunduseta (*look-and-feel*)



Joonis 31 Esilehe kujundus koos kujunduseta (*look-and-feel*)

Äriarhitektuuri diagrammi loomiseks ei piisa nuppudele ActionListener'ide külge panemisest, kuna tarvis on kuvada jpg formaadis pilte, mis esitavad diagrammi erinevaid osasid. Kuna siiani on kogu programmi sisuline osa kirjutatud meetodisse *mainGUI()*, on lihtsuse ja loetavuse huvides mõttekas piltide kuvamiseks luua uus meetod. Et seda meetodit kasutada, tuleb muuta *frame* globaalseks muutujaks, ehk selle defineerimine tõsta kõigist realiseeritud meetoditest väljapoole. Nii on võimalik sama JFrame'i kasutada rohkem kui ühel meetodil. Piltide kuvamiseks äriarhitektuuri aknas on sobilik kasutada Java Swing'i komponenti JLabel, millele on võimalik lisada *ImageComponent*, mis tähendab, et *JLabel'i* ehk tavaliselt teksti kuvamiseks kasutatava komponendi sisse pannakse nüüd pilt. Sellisele *JLabel'ile* saab lisada *MouseListener'i*, mis toimib sisuliselt nagu eelnevalt JButton'ile lisatud *ActionListener*. Järgneva koodinäitega luuakse *JLabel* nimega „kaubatundja“, mis esitab tegutsejat

äriarhitektuuri diagrammis ja pannakse sellele külge *MouseListener*, mille eesmärk on avada vastav aken, kus on toodud välja teoreetilist informatsiooni tegutseja kohta.

```
JLabel kaubatundja = new JLabel(new
ImageIcon("src/animation/kaubatundja.JPG"));
          kaubatundja.setBounds(170, 50, 83, 115);
          kaubatundja.addMouseListener(new MouseAdapter() {
               public void mouseClicked(MouseEvent e) {
                    try {
                         Tegutseja.main(null);
                    } catch (Exception e1) {
                         // TODO Auto-generated catch block
                         e1.printStackTrace();
                    }


               }
});
```

Antud tegutseja akna loomisel on peetud silmas, et aken avaneks väiksemana, kui põhiaken ning et selle sulgemisel ei sulgetaks kogu programmi. Selleks, et JFrame sulgemisel lõpetataks kogu programmi töö, kasutatakse Java Swing puhul käsklust `frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`. Tegutseja akna puhul on see käsklus lisamata, mis tähendab, et kui raam suletakse ei sulgu automaatselt lahti olev põhiraam ning programmi töö jätkub.

Selleks, et pilte, silte ja muid komponente liigutada, kasutatakse Swing'is *setBounds(x,y,width,height)* meetodit. See meetod määrab ära kujutise x koordinaadi, y koordinaadi ning kujutise laiuse ja kõrguse ning lubab kujutist paigutada meelepärasesse kohta terve *JFrame'i* laiuses.

Kõik vajalikud *JFrame'id* luuakse sarnaselt, kasutades üht meetodit raami loomiseks ning teist meetodit vajalike piltide raamile asetamiseks. Igale loodud raamile lisatakse "Sulge" nupp ning "Esilehele" nupp, mis on varustatud *ActionListener'itega*. Seetõttu on kõige aega nõudvam esimese raami loomine – järgnevate raamide puhul saab kasutada suurt osa juba varasemalt kirjutatud koodist.

6. Loodud animatsioonide võrdlus

MS PowerPoint on tarkvara, mis pakub kasutajale täielikku graafilist liidest animatsioonide koostamiseks ja esitamiseks. MS PowerPointi puhul ei kirjuta kasutaja üldjuhul programmikoodi, vaid kasutab talle tarkvara poolt pakutavaid lahendusi. Seetõttu on MS PowerPointi puhul töö tulemus kohe näha, mis teeb kergemaks ka testimise ja komponentide lisamise ning muutmise. Java puhul tuleb iga muutuse jaoks uuesti koodi poole pöörduda, samuti ei ole töö tulemus nii kiiresti nähtav kui MS PowerPointi puhul. Iga kord, kui soovitakse visuaalselt näha koodis tehtud muudatuse tulemust, on tarvis kood uuesti käivitada, mis on tegelikkuses ka suurem ajaline kulu kui MS PowerPoint'i slaidikomplekti käivitamine, sest Java puhul kompileeritakse kood iga kord enne selle käivitamist.

Java Swing'i kasutamine on küll ajakulukam, kuid reaalsuses paindlikum kui MS PowerPoint. Vaatamata sellele, et MS PowerPoint pakub esitluse realiseerimiseks palju komponente, on ühe slaidikomplekti sisuline ülesehitus siiski suhteliselt piiritletud. Java puhul on oskuslikul programmeerijal võimalik kasutajaliidest disainida paljudel erinevatel viisidel. Antud lõputöö raames avaneb erinevus näiteks selles, et navigeerides kasutusjuhtude diagrammilt tegutseja või kasutusjuhu teoorialehele, avaneb Swing'i puhul uus aken ilma vana akent sulgemata. See tähendab, et kasutajal on võimalus samal ajal näha nii teooriaosa kui reaalset diagrammi. MS PowerPointi puhul lahkutakse diagrammi kujutavalt slaidilt ning liigutakse uuele slaidile, mis sisaldab soovitud teooriaosa. Reaalsuses on see küll väike erinevus, kuid kasutajakogemuse puhul võib saada plussiks. Lõputöö autor katsetas antud erinevust Java Swing'i ja MS PowerPointi vahel kahel kursusekaaslasel, kes mõlemad tundsid, et Java Swing'is realiseeritu oli mugavam.

Teisalt on Java Swing'i realiseerida palju keerukam kui esitlust MS PowerPointis. Java Swing'i puhul on tarvis palju rohkem spetsiifilisi teadmisi ning tavakasutaja kindlasti Java poole ei pöörduks. Samuti on programmeerimiskeele puhul palju ebamugavam komponentide asukoha määramine liidesel. Käesolevas lõputöös ei kasutatud Eclipse'i poolt pakutavat *WindowBuilder'it*, mis tähendab, et iga väiksemagi asukoha muudatuse jaoks oli vajalik muuta koodis komponendi koordinaate. Kuigi *WindowBuilder'i* kasutamine oleks olnud komponentide asetuse suhtes lihtsam lähenemine, muudab see omakorda koordi üsna segaseks, mis tähendab, et uue funktsionaalsuse lisamine programmile oleks muutunud keerukamaks.

Ajalises mõttes on animatsiooni realiseerimine kiirem MS PowerPointi kasutades. Java koodi kirjutamine võtab rohkem aega, kui kujundite paigutamine Ms PowerPoint'i slaidile. Samuti

on koodi kirjutamise puhul kordades rohkem ruumi eksimiseks – üks väike viga kirjutatud meetodites võib programmeerijale maksma minna mitu tundi.

Järeldusena arvab autor, et taolise animatsiooni loomiseks on otstarbekam kasutada MS PowerPointi. Java Swing on kindlasti paindlik ning mitmekülgne, kuid selles kasutatava tarkvara loomine ajakulukam ning spetsiifilisem. MS PowrPoint on kasutajasõbralik ning pakkus antud animatsiooni realiseerimiseks rohkem kui küllaldaselt vajalikke komponente. Samuti leiab autor, et MS PowerPoint'i slaidikomplekti on kergem kasutada õppematerjalina ning mugavam kasutada tudengitel, kellele see slaidikomplekt suunatud on.

Töö käigus realiseeritud Java kood on esitatud Lisas 1.

Autor otsis võrdluse eesmärgil ka teisi sarnasel teemal animatsioone, kuid ei suutnud neid leida.

Kokkuvõte

Käesoleva bakalaureusetöö sisuks oli animatsioonide koostamine mõningate infosüsteemi analüüsi mudelite kohta. Täielikud animatsioonid loodi MS PowerPointis ning võrdluseks loodi osalised animatsioonid programmeerimiskeeles Java, täpsemalt kasutades Java Swing'i. Vaatluse all olid järgmised analüüsi mudelid:

- Äriarhitektuuri diagramm
- Olemi-suhte diagramm
- Kasutusjuhtude diagramm
- Seisundidiagramm
- CRUD maatriks

Teel lõppeesmärgini valmisid järgnevad vahetulemused, mis on esitatud käesoleva lõputöö peatükkides:

- ülevaade tööst kasutatavatest UML diagrammitüüpidest,
- ülevaade MS PowerPointist ja selle võimalustest,
- animatsioonide loomine MS PowerPointis,
- ülevaade programmeerimiskeelst Java ja Java Swing'ist,
- animatsioonide realiseerimine programmeerimiskeeles Java,
- võrdlus MS PowerPointis loodud animatsiooni ning programmeerimiskeeles Java realiseeritud animatsiooni vahel.

UML ehk *unified modeling language* võimaldab koostada visuaalseid mudeleid (diagramme), mis kirjeldavad infosüsteemi ja lihtustavad infosüsteemide projekteerimist. Diagrammide hulka, mida saab UMLi diagrammi tüüpide põhjal koostada kuuluvad muuhulgas äriarhitektuuri diagramm, olemi-suhte diagramm, seisundidiagramm ja kasutusjuhtude diagramm.

CRUD maatriks võimaldab esitada andmete kasutamist kirjeldavaid seoseid olemitüüpide ning kasutusjuhtude vahel.

MS PowerPoint on MS Office paketti kuuluv esitluste koostamise programm, mis on tavakasutajate seas laialt levinud ja pakub mitmeid võimalusi erinäoliste esitluste koostamiseks, mistõttu näeb käesoleva lõputöö raames koostatud õppematerjal välja rohkem animatsiooni kui standartse slaidikomplektina.

Java on objektorienteeritud programmeerimiskeel. Antud lõputöö raames oli eesmärgiks realiseerida kogu Java kood käsitsi, et mõista ja välja tuua kui erinev on animatsiooni loomine graafilise kasutajaliidesega ning koodi kirjutamisega.

Kasutajasõbralikku ja visuaalselt meeldivat animatsiooni on kergem ning kiirem luua MS PowerPoint'iga. Programmeerimiskeel Java nõuab rohkem spetsiifilisi teadmisi ning oskusi. Ometi on Java kogenenud programmeerijate jaoks paindlikum kui MS PowerPoint.

Kõik püstitatud eesmärgid on täidetud. Lõputöö kirjutaja sai rakendada enda teadmisi UML diagrammidest ning infosüsteemi analüüsimudelitest, laiendada oma teadmisi levinud MS PowerPointist ja realiseerida animatsioon osaliselt Java Swing meetoditel.

Praktilises elus võiks MS PowerPointis loodud animatsioon kasutust leida õppematerjalina töös käsitletud UML mudelite teema juures. Töö edasiseks arenduse suunaks võiks olla veel täiuslikum animatsioon, mis juhendab vaatajat samm-sammult läbi nende mudelite loomise võimaliku protsessi.

Summary

The goal of the bachelor's thesis in hand was to create animations about some analysis models of information systems. The complete animation was created in MS PowerPoint and a partial reconstruction of the animation was created in Java Swing to compare the two. The thesis concentrated on the following analysis models.

- Business architecture diagram
- Entity relationship diagram
- Use case diagram
- State diagram
- CRUD matrix

The main results of the thesis, which correspond to the main chapters, are as follows:

- an overview of the UML diagram types used in the thesis,
- an overview of MS PowerPoint and its possibilities,
- creating animations in MS PowerPoint,
- an overview of Java and Java Swing and their possibilities,
- creating a partial animation in Java Swing,
- a comparison between the two techniques.

UML or Unified Modeling Language enables the user to create visual models (diagrams) to depict an information system and simplify the development of information systems. Diagrams that could be created based on UML diagram types include but are not limited to use case diagrams, business architecture diagrams, state diagrams, and entity-relationship diagrams.

CRUD matrix could be used to show the relations between use cases and entities.

MS PowerPoint is a program that is a part of the MS Office package. It enables users to create slideshows with animations. It is very common with regular users but also offers many opportunities to create customized slideshows, which is why the slideshow created in the course of the thesis in hand looks more like an animation and not as much as a regular slideshow.

Java is an object-orientated programming language. A goal of this bachelor's thesis was to implement the code by hand without using the help of WindowBuilder provided by Eclipse. It was done to understand the differences between creating an animation using a graphical user interface and implementing manually methods using a programming language.

Creating a user friendly and aesthetically appealing animation is easier with the help of MS PowerPoint. Java requires a lot more specific knowledge and skill. Nevertheless, Java is more flexible for a seasoned software developer compared to MS PowerPoint.

All the goals set have been met. The author of the bachelor's thesis was able to use her knowledge about UML diagrams and analysis models of information systems, expanded the knowledge of MS PowerPoint, and implemented the animation partially by using Java Swing.

The animation created in MS PowerPoint could be used as a study material when covering analysis models of information systems. Further work on this topic could include even more advanced animation that guides the user step-by-step through the creation process of models.

Kasutatud kirjandus

1. UML Basics: The class diagram [WWW]
http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/ (18.04.2015)
2. Eessaar, E. (2012) Andmebaaside projekteerimiseks kasutatavad mudelid, Õppematerjal, 3-12 [WWW]
http://maurus.ttu.ee/ained/IDU0220_2012/doc/11/Modelleerimine_IDU0220_2012_ver_2_14.pdf (25.05.2015)
3. Microsoft PowePoint [WWW]
http://www.britannica.com/EBchecked/topic/1491611/Microsoft-PowerPoint (20.04.2015)
4. History of PowerPoint: The Amazing Facts You Did Not Know [WWW] http://www.free-power-point-templates.com/articles/history-of-powerpoint-the-amazing-facts-you-did-not-know/ (20.04.2015)
5. History of Java programming language [WWW]
http://www.freejavaguide.com/history.html (24.04.2015)
6. A Swing Architecture Overview [WWW]
http://www.oracle.com/technetwork/java/architecture-142923.html (24.04.2015)
7. SWING - Overview [WWW] http://www.tutorialspoint.com/swing/swing_overview.htm (24.04.2015)
8. Nimbus Look and feel [WWW]
https://docs.oracle.com/javase/tutorial/uiswing/lookandfeel/nimbus.html (25.04.2015)
9. Preventing Mouse-clicks from Advancing Slides [WWW]
http://notes.indezine.com/2009/06/prevent-mouse-clicks-from-advancing.html (14.05.2015)
10. UML2 State Machine DIagrams: An Agile Introduction [WWW]
http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm(17.05.2015)
11. Background to a CRUD Matrix [WWW]
http://www.databaseanswers.org/data_arch/crm/crud_matrix.htm (17.05.2015)
12. Roost, M. (2014) Ettevõtte äriarhitektuurid(IDU 1321, 1322). Infosüsteemide strateegiline analüüs(IDU 0021, IDU 0022) Loengumaterjalid
[WWW] http://maurus.ttu.ee/ained/IDU0021/doc/7/IS_strat_analyys_2014_loengud_3detsembri_seisuga.docx (25.05.2015)

Lisad

Lisa 1: Java Swing rakenduse kood

Klass MainFrame.java

```
package animation;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;

public class MainFrame extends JFrame {
	private static final long serialVersionUID = 1L;
	public static void mainGUI() {
		final JFrame frame = new JFrame("Avaleht");
		frame.setSize(800, 600);
		frame.setResizable(false);
		frame.setBackground(new Color(213, 212, 212));
		frame.setLocationRelativeTo(null);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JPanel welcomePanel = new JPanel();
		welcomePanel.setBackground(new Color(150, 255, 140));
		welcomePanel.setLayout(null);
		frame.getContentPane().add(welcomePanel);
		frame.setVisible(true);

		JButton buttonAri = new JButton("Äriarhitektuuri diagramm");
		buttonAri.setBounds(290, 150, 200, 50);
		buttonAri.setFont(new Font("Calibri", Font.PLAIN, 15));
		welcomePanel.add(buttonAri);
		JButton buttonOlem = new JButton("Olemi-suhte diagramm");
		buttonOlem.setBounds(290, 250, 200, 50);
		buttonOlem.setFont(new Font("Calibri", Font.PLAIN, 15));
		welcomePanel.add(buttonOlem);
		JButton buttonKasutaja = new JButton("Kasutusjuhtude diagramm");
		buttonKasutaja.setBounds(290, 350, 200, 50);
		buttonKasutaja.setFont(new Font("Calibri", Font.PLAIN, 15));
		welcomePanel.add(buttonKasutaja);

		buttonAri.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				frame.dispose();
				try {
					Ariarhitektuur.main(null);
				} catch (Exception e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
				}

			}
		});
		buttonKasutaja.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				frame.dispose();
				try {
					Kasutusjuht.main(null);
				} catch (Exception e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
				}
			}
		});
		buttonOlem.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				frame.dispose();
				try {
```

```
                                Olemisuhte.main(null);
                            } catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                            }

                        }
                });
                JButton sulgeBtn = new JButton("Sulge");
                sulgeBtn.setBounds(680, 525, 100, 25);
                welcomePanel.add(sulgeBtn);
                sulgeBtn.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                frame.dispose();

                        }
                });
        }

        public static void main(String args[]) {
                try {
                        for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
                                if ("Nimbus".equals(info.getName())) {
                                        UIManager.setLookAndFeel(info.getClassName());
                                        break;
                                }
                        }
                } catch (Exception e) {
                        System.out.println("Theme Nimbus not available!");
                }
                javax.swing.SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                mainGUI();
                        }
                });
        }
}
```

Klass Ariarhitektuur.java

```
package animation;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;

public class Ariarhitektuur extends JFrame {

        /**
         *
         */
        private static final long serialVersionUID = 1L;
        public static final JFrame frame = new JFrame("Äriarhitektuuri diagramm");

        public static void main(String args[]) {
                try {
                        for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
                                if ("Nimbus".equals(info.getName())) {
                                        UIManager.setLookAndFeel(info.getClassName());
                                        break;
                                }
                        }
                } catch (Exception e) {
                        System.out.println("Theme Nimbus not available!");
                }
```

```
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                        ariGUI();
                }
        });
}

public static void ariGUI(){

        frame.setSize(800, 600);
        frame.getContentPane().setBackground(Color.white);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);
        frame.setResizable(false);
        frame.setVisible(true);

        JButton sulgeBtn = new JButton("Sulge");
        sulgeBtn.setBounds(680, 525, 100, 25);
        frame.getContentPane().add(sulgeBtn);

        JButton frontBtn = new JButton("Esilehele");
        frontBtn.setBounds(550, 525, 100, 25);
        frame.getContentPane().add(frontBtn);
        sulgeBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        frame.dispose();

                }
        });
        frontBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        frame.dispose();
                        try{
                                MainFrame.main(null);
                        }
                        catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }

                }
        });
        JLabel kasutajast = new JLabel("Klõpsa tegutsejal, et näha teooriat");
        kasutajast.setBounds(35,30,250,50);
        frame.getContentPane().add(kasutajast);
        JLabel olemist = new JLabel("Klõpsa, et näha olemi-suhte diagrammi");
        olemist.setBounds(115,460,250,50);
        frame.getContentPane().add(olemist);
        JLabel kasutajad = new JLabel("Klõpsa, et näha kasutusjuhtude diagrammi");
        kasutajad.setBounds(45,235,280,70);
        frame.getContentPane().add(kasutajad);
        addImages();
}

public static void addImages(){
        JLabel juhataja = new JLabel(new ImageIcon("src/animation/juhataja.JPG"));
        juhataja.setBounds(510, 50, 83, 115);
        juhataja.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent e) {
                        //frame.dispose();
                        try {
                                Tegutseja.main(null);
                        } catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }


                }
        });
        frame.getContentPane().add(juhataja);
        JLabel kaubatundja = new JLabel(new ImageIcon("src/animation/kaubatundja.JPG"));
        kaubatundja.setBounds(170, 50, 83, 115);
```

```
		kaubatundja.addMouseListener(new MouseAdapter() {
			public void mouseClicked(MouseEvent e) {
				//frame.dispose();
				try {
					Tegutseja.main(null);
				} catch (Exception e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
				}


			}
		});
		frame.getContentPane().add(kaubatundja);
		JLabel kaupadereg = new JLabel(new ImageIcon("src/animation/kaupadereg.JPG"));
		kaupadereg.setBounds(160, 370, 110, 105);
		kaupadereg.addMouseListener(new MouseAdapter() {
			public void mouseClicked(MouseEvent e) {
				//frame.dispose();
				try {
					Olemisuhte.main(null);
				} catch (Exception e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
				}


			}
		});
		frame.getContentPane().add(kaupadereg);
		JLabel kfunsys = new JLabel(new ImageIcon("src/animation/kfunsys.JPG"));
		kfunsys.setBounds(290, 220, 200, 105);
		kfunsys.addMouseListener(new MouseAdapter() {
			public void mouseClicked(MouseEvent e) {
				//frame.dispose();
				try {
					Kasutusjuht.main(null);
				} catch (Exception e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
				}


			}
		});
		frame.getContentPane().add(kfunsys);
		JLabel klasreg = new JLabel(new ImageIcon("src/animation/klasreg.JPG"));
		klasreg.setBounds(520, 400, 110, 105);
		frame.getContentPane().add(klasreg);

		JLabel noolRight = new JLabel(new ImageIcon("src/animation/nool_parem.JPG"));
		noolRight.setBounds(210,110,100,120);
		frame.getContentPane().add(noolRight);


		JLabel noolRight2 = new JLabel(new ImageIcon("src/animation/nool_parem.JPG"));
		noolRight2.setBounds(450,290,100,120);
		frame.getContentPane().add(noolRight2);

		JLabel noolLeft = new JLabel(new ImageIcon("src/animation/nool_vasak.JPG"));
		noolLeft.setBounds(480,140,100,120);
		frame.getContentPane().add(noolLeft);


		JLabel noolLeft2 = new JLabel(new ImageIcon("src/animation/nool_vasak.JPG"));
		noolLeft2.setBounds(250,290,100,120);
		frame.getContentPane().add(noolLeft2);


	}
}
```

Klass Tegutseja.java

```
package animation;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;

public class Tegutseja extends JFrame{
	private static final long serialVersionUID = 1L;
	public static void main(String[] args) {
		try {
			for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
				if ("Nimbus".equals(info.getName())) {
					UIManager.setLookAndFeel(info.getClassName());
					break;
				}
			}
		} catch (Exception e) {
			System.out.println("Theme Nimbus not available!");
		}
		javax.swing.SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				mainGUI();
			}
		});
	}
	public static void mainGUI(){
		final JFrame frame = new JFrame("Tegutseja");
		frame.setSize(500, 500);
		frame.setLocationRelativeTo(null);
		frame.setResizable(false);
		//frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JPanel welcomePanel = new JPanel();
		welcomePanel.setBackground(new Color(150,255,140));
		welcomePanel.setLayout(null);
		frame.getContentPane().add(welcomePanel);
		frame.setVisible(true);

		JLabel head = new JLabel("Tegutseja");
		head.setBounds(220, 10, 200, 50);
		head.setFont(new Font("Calibri", Font.BOLD, 20));
		welcomePanel.add(head);

		String text ="Tegutseja on keegi või miski, mis suhtleb süsteemiga või kasutab"+" seda.
Tegutsejad viivad läbi kasutusjuhte. Oluline on"+" silmas pidada, et tegutseja esitab rolli, mitte
aga"+" süsteemi üksikkasutajat. Tegutseja poolt algatatakse kasutusjuht"+
				" ja see on süsteemi jaoks sündmus, mis käivitab vajaliku teenuse.\n "
+
				"Tegutsejaid kasutatakse peamiselt kasutusjuhtude diagrammil, kus"+
				" esitatakse seosed tegutseja ja kasutusjuhu"+" vahel.\n
Äriarhitektuuri diagrammil on näidatud, et tegutsejad"+
				" kasutavad ehk saadavad käsklusi Kaupade"+
				"  funktsionaalset allsüsteemile, mis omakorda suhtleb edasi Kaupade
registri ning Klassifikaatorite registriga.";

		JLabel body = new JLabel("<html>"+text+"</html>");
		body.setBounds(5, 10, 450, 400);
		body.setFont(new Font("Calibri", Font.PLAIN, 16));
		welcomePanel.add(body);
		JButton sulgeBtn = new JButton("Sulge");
		sulgeBtn.setBounds(380, 425, 100, 25);
		welcomePanel.add(sulgeBtn);
```

```
		sulgeBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				frame.dispose();
			}
		});
	}

}
```

Klass Kasutusjuht.java

```
package animation;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;

public class Kasutusjuht {

	public static final JFrame frame = new JFrame("Kasutusjuhtude diagramm");

	public static void main(String[] args) {
		try {
			for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
				if ("Nimbus".equals(info.getName())) {
					UIManager.setLookAndFeel(info.getClassName());
					break;
				}
			}
		} catch (Exception e) {
			System.out.println("Theme Nimbus not available!");
		}
		javax.swing.SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				kasutusGUI();
			}
		});
	}
	public static void kasutusGUI() {
		frame.setSize(800, 600);
		frame.getContentPane().setBackground(Color.white);
		frame.setLocationRelativeTo(null);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.getContentPane().setLayout(null);
		frame.setResizable(false);
		frame.setVisible(true);

		JButton sulgeBtn = new JButton("Sulge");
		sulgeBtn.setBounds(680, 525, 100, 25);
		frame.getContentPane().add(sulgeBtn);
		JButton frontBtn = new JButton("Esilehele");
		frontBtn.setBounds(550, 525, 100, 25);
		frame.getContentPane().add(frontBtn);
		sulgeBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				frame.dispose();
			}
		});
		frontBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				frame.dispose();
				try {
					MainFrame.main(null);
				} catch (Exception e1) {
					// TODO Auto-generated catch block
					e1.printStackTrace();
```

```
                        }
                }
        });
        JButton ariBtn = new JButton("Tagasi");
        ariBtn.setBounds(10, 525, 100, 25);
        frame.getContentPane().add(ariBtn);
        ariBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        frame.dispose();
                        try {
                                Ariarhitektuur.main(null);
                        } catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }

                }
        });
        addImages();
    }
    public static void addImages() {
        JLabel user = new JLabel("Tegutseja(klõpsa, et näha rohkem)");
        user.setFont(new Font("Calibri", Font.BOLD, 13));
        JLabel usecase = new JLabel("Kasutusjuht(klõpsa, et näha rohkem)");
        usecase.setFont(new Font("Calibri", Font.BOLD, 13));
        usecase.setBounds(540, 460, 220, 50);
        user.setBounds(550, 5, 200, 50);
        frame.getContentPane().add(user);
        frame.getContentPane().add(usecase);
        JLabel arrowUp = new JLabel(new ImageIcon("src/animation/arrow_up.JPG"));
        arrowUp.setBounds(630, 410, 15, 60);
        frame.getContentPane().add(arrowUp);
        JLabel arrowDown = new JLabel(new ImageIcon(
                        "src/animation/arrow_down.JPG"));
        arrowDown.setBounds(630, 40, 15, 73);
        frame.getContentPane().add(arrowDown);
        JLabel diagramm1 = new JLabel(new ImageIcon(
                        "src/animation/kasutajadgr1.JPG"));
        diagramm1.setBounds(10, 70, 450, 450);
        frame.getContentPane().add(diagramm1);

        JLabel juhataja = new JLabel(
                        new ImageIcon("src/animation/juhataja.JPG"));
        juhataja.setBounds(600, 120, 83, 115);
        juhataja.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent e) {
                        // frame.dispose();
                        try {
                                Tegutseja.main(null);
                        } catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }
                }
        });
        frame.getContentPane().add(juhataja);
        JLabel kasutusjuht = new JLabel(new ImageIcon(
                        "src/animation/kastusujuht.JPG"));
        kasutusjuht.setBounds(540, 220, 200, 180);
        kasutusjuht.addMouseListener(new MouseAdapter() {
                public void mouseClicked(MouseEvent e) {
                        // frame.dispose();
                        try {
                                UseCase.main(null);
                        } catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }
                        }
        });
    frame.getContentPane().add(kasutusjuht);
    }
}
```

Klass UseCase.java

```
package animation;
import java.awt.Color;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;

public class UseCase extends JFrame{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public static void main(String[] args) {
		try {
			for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
				if ("Nimbus".equals(info.getName())) {
					UIManager.setLookAndFeel(info.getClassName());
					break;
				}
			}
		} catch (Exception e) {
			System.out.println("Theme Nimbus not available!");
		}
		javax.swing.SwingUtilities.invokeLater(new Runnable() {
			public void run() {
				mainGUI();
			}
		});
	}
	public static void mainGUI(){
		final JFrame frame = new JFrame("Kasutusjuht");
		frame.setSize(500, 500);
		frame.setLocationRelativeTo(null);
		frame.setResizable(false);
		//frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		JPanel welcomePanel = new JPanel();
		welcomePanel.setBackground(new Color(150,255,140));
		welcomePanel.setLayout(null);
		frame.getContentPane().add(welcomePanel);
		frame.setVisible(true);
		JLabel head = new JLabel("Kasutusjuht");
		head.setBounds(200, 10, 200, 50);
		head.setFont(new Font("Calibri", Font.BOLD, 20));
		welcomePanel.add(head);
		String text ="Kasutusjuht on jutustav dokument, mis kirjeldab sündmuste jada,"+" mis
tekib, kui tegutseja, süsteemiväline agent kasutab süsteemi mingi protsessi läbi viimiseks (Ivar
Jacobson).\n" +
				"Kasutusjuht on mingi protsessi samm-sammuline kirjeldus." +  "
Kasutusjuht on tavaliselt võrdlemisi mahukas.\n";

		String line = "Kasutusjuhu omadused on:";
		String line1 = "Kasutusjuht käivitatakse alati tegutseja poolt";
		String line2 = "Kasutusjuht peab olema täielik";
		String line3 = "Kasutusjuht annab tegutsejale väärtuse";
		JLabel body = new
JLabel("<html>"+text+"<br>"+line+"<br>"+line1+"<br>"+line2+"<br>"+line3+"</html>");
		body.setBounds(5, 5, 450, 400);
		body.setFont(new Font("Calibri", Font.PLAIN, 18));
		welcomePanel.add(body);
		JButton sulgeBtn = new JButton("Sulge");
		sulgeBtn.setBounds(380, 425, 100, 25);
		welcomePanel.add(sulgeBtn);
		sulgeBtn.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
```

```
                                frame.dispose();
                        }
                });
        }
}
```

Klass Olemisuhte.java

```
package animation;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.UIManager;
import javax.swing.UIManager.LookAndFeelInfo;

public class Olemisuhte {
        public static final JFrame frame = new JFrame("Olemi-suhte diagramm");
        public static void main(String[] args) {
                try {
                        for (LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
                                if ("Nimbus".equals(info.getName())) {
                                        UIManager.setLookAndFeel(info.getClassName());
                                        break;
                                }
                        }
                } catch (Exception e) {
                        System.out.println("Theme Nimbus not available!");
                }
                javax.swing.SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                olemiGUI();
                                addImage();
                        }
                });
        }

        public static void olemiGUI(){
                frame.setSize(800, 650);
                frame.getContentPane().setBackground(Color.white);
                frame.setLocationRelativeTo(null);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.getContentPane().setLayout(null);
                frame.setResizable(false);
                frame.setVisible(true);

                JButton sulgeBtn = new JButton("Sulge");
                sulgeBtn.setBounds(680, 575, 100, 25);
                frame.getContentPane().add(sulgeBtn);

                JButton frontBtn = new JButton("Esilehele");
                frontBtn.setBounds(550, 575, 100, 25);
                frame.getContentPane().add(frontBtn);
                sulgeBtn.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                frame.dispose();

                        }
                });
                frontBtn.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                frame.dispose();
                                try{
                                        MainFrame.main(null);
                                }
                                catch (Exception e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }
```

```
                }
        });
        JButton ariBtn = new JButton("Tagasi");
        ariBtn.setBounds(10, 575, 100, 25);
        frame.getContentPane().add(ariBtn);
        ariBtn.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        frame.dispose();
                        try{
                                Ariarhitektuur.main(null);
                        }
                        catch (Exception e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }

                }
        });
    }
    public static void addImage(){
        JLabel diagram = new JLabel(new ImageIcon("src/animation/olemisuhte.jpeg"));
        diagram.setBounds(70,1,500,600);
        frame.getContentPane().add(diagram);
    }
}
```

Lihtlitsents lõputöö üldsusele kättesaadavaks tegemiseks ja reprodutseerimiseks

Mina, Kristine Baumann (sünnikuupäev:12.03.1993.)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

Animatsioonid infosüsteemi analüüsimudelite kohta

mille juhendaja on Erki Eessaar,

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas TTÜ raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas TTÜ raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

_____ (*allkiri*)

26.05.2015 (*kuupäev*)