TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl Johannes Artma 142750IABB

VALDKONNASPETSIIFILISE GRAAFILISE KEELE PROTOTÜÜBI VÄLJATÖÖTAMINE *ECLIPSE SIRIUSE* PLATVORMIL

Bakalaureusetöö

Juhendaja: Mart Roost
magistrikraad

Tallinn 2020

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.

Autor: Karl Johannes Artma

[14.05.2020]

Annotatsioon

Antud töö eesmärgiks on välja töötada valdkonnaspetsiifilise graafilise (modelleerimis)keele ning sellel põhineva tarkvara prototüüp ülikooli lõputööde juhendamise ühe alamvaldkonna jaoks *Eclipse Siriuse* platvormil.

Käsitletavaks põhiliseks probleemiks on kommunikatsioonilõhe tarkvara tellija ja tarkvaraarendaja vahel, mida üritatakse lahendada valdkonnaspetsiifilistel keeltel põhineva lähenemise kasutuselevõtuga. Töö konkreetne kontekst on ülikooli lõputööde juhendamise alamvaldkond juhendaja ja juhendatava kokkuviimiseks.

Selle töö põhilisteks tulemusteks olid lõputööde juhendamise alamvaldkonna (juhendaja ja juhendatava kokkuviimine) mudelid, graafilise keele (abstraktse ja konkreetse süntaksi) ja keele tööriistade spetsifikatsioonid ning genereeritud tarkvaraprototüüp antud platvormil.

Võib öelda, et valdkonnaspetsiifilistel keeltel põhinev lähenemine aitab arendustöös esinevate kommunikatsiooniprobleemide lahendamisele kaasa ning genereeritud prototüüp on valitud valdkonnas rakendatav ja kasulik.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 38 leheküljel, 6 peatükki, 11 joonist, 0 tabelit.

Abstract

Development of the Graphical Domain Specific Language's Prototype on the *Eclipse Sirius* Platform.

The purpose of the bachelor thesis is to develop a prototype of the domain specific modeling language of the university's theses' supervising subfield on the Eclipse Sirius software platform.

The main problem the thesis focuses on is a communication issue between the end user and the software developer. Domain specific language's approach in this matter could be one of the solutions to the stated problem. The specific context of the thesis is the university's theses' supervising subfield, which focuses on bringing together the supervisor and the student.

The main results of the thesis were the metamodels of the university's theses' supervising subfield, an initial version of specification of graphical modeling language (the abstract and concrete syntax) among with its tools specifications and a software prototype generated on the *Eclipse Sirius* platform.

It can be said that the domain specific languages' approach could be helpful in solving the communication issues that may occur in the development process and also the generated prototype in the thesis' subfield is practical and useful.

The thesis is in Estonian language and contains 38 pages of text, 6 chapters, 11 figures, 0 tables.

Lühendite ja mõistete sõnastik

Abstraktne süntaks	*Eclipse* platvormil kasutatud *Ecore* metamudelid
Domeenklass	Pärimussuhet kirjeldav klass
DOT	Valdkonnaspetsiifiline keel graafide, nende klasside, seoste ja karakteristikute kirjeldamiseks [1]
Eclipse Sirius	Modelleerimistöös kasutatav tarkvaraarendusplatvorm
Ecore	Valdkonna mudel. Valdkonnaspetsiifilise metamudeli struktuur *Eclipse Siriuse* platvormil
EMF	*Eclipse Modeling Framework*. *Eclipse* modelleerimise raamistik
GMF	*Graphical Modeling Framework*. Graafilise modelleerimise raamistik
GML	Graphical Modeling Language. Graafiline modelleerimiskeel
Graafiline keel	Programmeerimiskeel, mille süntaks koosneb visuaalsetest objektidest, mis lihtsustavad programmi loomist ja selle mõistmist
Graafiline liides	Konkreetne süntaks, kus asub loodud prototüüp
Konkreetne süntaks	*Eclipse* platvormil kasutatud graafilised *Odesign* failid
Kontseptsioon	Objekti abstraktsioon [2]
Kontseptuaalmudel	Mudel, mis koosneb kontseptsioonidest ehk objektide abstraktsioonidest [2]
Low-code/no-code	Lähenemine, milles rakenduse arendustööd tehakse visuaalselt/graafiliselt, luues mudelipõhise arendusega mingi graafiline liides [3]
(Lõpp)kasutajad	Tulevased inimesed või organisatsioonid, kes hakkavad kasutama loodud tarkvara
Metamudel	Kontseptuaalmudelite klassidest koosnev kontseptuaalmudel [2]
Valdkonnamudel	Ehk metamudel
MDA	*Model Driven Architecture* ehk mudelitega juhitav arhitektuur. Lähenemine, mis põhineb tarkvara disainil ja ühtlasi spetsifikatsioon, mis annab juhtnöörid mudelite tegemiseks [4]
Programmeerimiskeeled	Arvutites kasutatavad tööriistad laia valdkonna probleemide lahendamiseks ja programmide loomiseks
Projektsiooni redaktor	Programmi muutmise vahend, mis võimaldab näidata failide sisu projektsioonide kuvamise teel [1]
Redaktor	Programmi muutmise tööriist/vahend

Representatsioon	Diagramm või diagrammivaade
Tarkvara tellija	Inimene või organisatsioon, kes annab juhised loodava tarkvara tegemiseks. Tavaliselt ka tasub tarkvaraarendajale loodud töö eest
Tarkvaraarendaja	Inimene, kes omab programmeerimisoskust ja loob saadavate tarkvara tellija poolsete juhiste järgi funktsionaalse tarkvara
Tarkvaraprototüüp	Esmane lihtsustatud tarkvaralahendus lõppkasutajale kasutamiseks
Tekstikeel	Paljudes kontekstides kasutatav üldjuhul programmeerimiskeel [1]
Turuplats	Alamvaldkond, mis kirjeldab juhendatava ja juhendaja kokkusaamist ehk teemade ja teemavaldkondade kogumit
Valdkond	Ehk domeen. Teemavaldkond või huvipiirkond, mida saab iseloomustada temas sisalduvate objektide, nende objektide omaduste ning objektide poolt võimaldatavate funktsioonide kaudu [2]
VSK	Valdkonnaspetsiifiline keel. Spetsiifiline abivahend teatud tüüpi spetsiifiliste probleemide lahendamiseks. Valdkonnaspetsiifilisi keeli kasutatakse spetsiifilistes valdkondades, kus neid luuakse eesmärgiga toetada valdkonnas sisalduvaid kindlat tüüpi ülesandeid [1]
VSMK	Valdkonnaspetsiifiline modelleerimiskeel, mida kasutatakse graafilise liidese loomiseks [1]
VSPK	Valdkonnaspetsiifiline programmeerimiskeel [2]
ÜOMK	Üldotstarbeline modelleerimiskeel, mis on sõltumatu kindlast probleemvaldkonnast [2]
BPMN	Tüüpiline üldotstarbeline modelleerimiskeel [2]
OMG	*Object Management Group.* Objektide juhtimise/haldamise grupp [4]
PIM	*Platform Independent Model*. Platvormist sõltumatu mudel [4]
PSM	*Platform Specific Model.* Platvormispetsiifiline mudel [4]
PDM	*Platform Defined Model.* Platvormi definitsiooni mudel [4]
CORBA	*Common Object Request Broker Achitecture.* Vabavaraline sõltumatu infrastruktuur ja arhitektuur arvutirakendustega võrgu teel suhtlemiseks [5]
.Net	Vabavaraline tarkvaraplatvorm arvutirakenduste loomiseks [6]
Use Case	Kasutusjuht
QVT	Üks *OMG* standardeid [4]
ATL	*QVT* standardi mudeliteisenduse keel [4]

Tuletatud mudel	Arvutite poolt loodud mudel [4]
UML	Inimese loodud metamudeli diagramm [4]
UML Package Diagram	*UML*-i paketidiagramm
Select Solution for MDA	*MDA* tööriist, mis suudab teha enamiku *MDA* funktsioone [4]
Layer	Kiht
Java	Programmeerimiskeel
Python	Programmeerimiskeel
C#	Programmeerimiskeel
Front-end	Veebilehe kasutajaliides, millega puutub kokku lõppkasutaja [7]
Back-end	Veebilehe serveri osa, milles hoitakse ja korrastatakse veebilehe andmeid [7]
Combo-box	Tavaline ekraanivorm
JMI	*Java* metaandmete kasutajaliides [8]
Analüüsimuster	Ekspertide poolt loodud taaskasutatav (ja tarkvara realisatsiooni omav) objektmudel
Accountability	Kohustus/Vastutus – analüüsimuster
Party	Osapool – analüüsimuster
Knowledge Level	Teadmustase – analüüsimuster
Typed Relationship	Üks analüüsimustritest, võib kuuluda *Accountability* *Knowledge Level* mustrite koosseisu
Connection Rule	*Knowledge Leveli* mustri ühendamisreegli klass
Parent	Vanemosapoole ühendus/seos
Child	Lapsosapoole ühendus/seos
Design time	Disainiaeg. Üks iteratsiooni etappidest
Runtime	Kasutusaeg ehk täitmisaeg. Üks iteratsiooni etappidest.
Obeo	Firma, mille omanduses ja arendustöös on tarkvara *Eclipse Sirius* [9]
Obeo Designer Community	*ODC.* Vabavaraline ümbernimetatud *Eclipse Siriuse* versioon [9]
Github	Töös pideva varunduse tagamiseks kasutatud tarkvara [10]
Draw.io	Programm mudelite lihtsustatud eskiiside modelleerimiseks [11]
Graphiti	Valdkonnaspetsiifilise graafilise keele väljatöötamise tööriist, mis on alternatiiviks *Eclipse Siriusele* [12]

EMF Forms	Alternatiiv *Eclipse Siriusele*. Platvorm, mis järgib puhtalt *no-code* lähenemist
End-User Developer	Lõppkasutajast tarkvaraarendaja

Sisukord

Jooniste loetelu

Tabelite loetelu

No table of figures entries found.

1 Sissejuhatus

Käesolevas lõputöös luuakse valdkonnaspetsiifilise modelleerimiskeele prototüüp ülikooli lõputööde juhendamise ühe alamvaldkonna jaoks, kasutades *Eclipse Sirius* platvormi.
Selles peatükis kirjeldatakse töö teema tausta ja probleemi, metoodikat, ülesandepüstitust ning antakse ülevaade töö struktuurist.

1.1 Taust ja probleem

Tarkvaraarenduses tuleb lahendada probleemi nii koodi terminites kui ka valdkonnaga seonduvalt, sest programmeerimiskeele ja uuritava valdkonna vaheline seos on väike. Parimaks võimaluseks oleks ülesande ühel viisil lahendamine probleemvaldkonna graafilise mudeli loomise teel. Selleks läheb vaja sobivat modelleerimiskeelt, arenduskeskkonda ning metoodikat mudelist tarkvararakenduse genereerimiseks. Valdkonnaspetsiifilise modelleerimiskeele loomist vastavaid tööriistu kasutades on pakutud üheks võimalikuks lahenduseks lahendatavale probleemile [13].

Sama probleemi väljenduseks on ka kommunikatsioonilõhe tarkvara tellija ja tarkvaraarendaja vahel. Kumbki osapool suhtleb enda valdkonna spetsiifilistes terminites, seetõttu ei mõisteta üksteist nii hästi, kui arendustöö soovitud tulemuste saavutamiseks vaja oleks. Ühtlasi ei ole võimalik analüüsi- ja disainitöö etapis näha täpselt ette kõiki nõudmisi, vajadusi, ülesandeid, mida rakenduse (lõpp)kasutajad soovivad. Samas lõplik disain loodud lahendusele tekib alles rakenduse kasutamise käigus. Lõpliku rakenduseni jõudmiseks on segaduste vältimiseks vajalik saada mõlema osapoole pidevat tagasisidet. Samas tuleb arvestada, et selline mõtete edasi-tagasi tõlgendamine on aeganõudev ja kulukas protsess [1].

Lisaks kommunikatsiooniprobleemile tasub mainida ka tarkvaraarenduse hinda. Ülikoolides ja ettevõtetes kasutatakse õppetööalaselt ning põhitegevuses mitmeid tarkvaraprogramme. Need tarkvarad vajavad aga uuendusi ja aeg-ajalt ka arendustöö

väljastpoolt tellimist. Paraku on infosüsteemide arendustöö tellimine IT spetsialistidelt pigem kallis ja nimetatud spetsialistide kasvava defitsiidi tõttu piiratud.

Antud probleemidega tegelemiseks võib sobida valdkonnaspetsiifilistel keeltel põhinev lähenemine. Valdkonnaspetsiifilise (modelleerimis)keele prototüübi kasutamine lõppkasutaja poolt on ajaliselt mõttekam, odavam ning professionaalsema kasutaja puhul on temapoolsed nõuded, vajadused veelgi täpsemalt defineeritud. Sellise lähenemise kasutuselevõtuga võib õige lahenduseni jõudmine toimuda sujuvamalt [1].

Töö konkreetseks kontekstiks on ülikooli lõputööde juhendamise valdkonna konkreetne alamvaldkond – 'turuplats' juhendatava ning juhendaja kokkuviimiseks - kus eelnimetatud tüüpprobleemid on samuti aktuaalsed ning nimetatud lähenemine võiks olla rakendatav ja kasulik. Töö teema keskendub valdkonnaspetsiifilise modelleerimiskeele prototüübi väljatöötamisele. Alampeatükis 1.2 antakse lühikirjeldus valdkonnaspetsiifilise keele (VSK-e) kasutamise kohta.

1.2 Töö metoodika

Kirjeldatud probleemidele aitab lahendust leida valdkonnaspetsiifiliste keelte kasutamine arendustegevusel. Valdkonnaspetsiifilise keele (VSK-e) lähenemist hakkavad (lõpp)kasutajatena kasutama õppejõud ja tudengid, kellele antakse kätte spetsiaalsed jõukohased tööriistad, mis hõlbustavad neil „lõpliku" rakenduse loomist.

Valdkonnaspetsiifilisi keeli kasutatakse spetsiifilistes valdkondades, kus neid luuakse eesmärgiga toetada valdkonnas sisalduvaid kindlat tüüpi ülesandeid [1].

Töös on kasutusel *Eclipse Siriuse* tarkvara ehk valdkonnaspetsiifiliste modelleerimiskeelte väljatöötamise platvorm, mida arendatakse *Obeo* firma poolt. Platvorm on valitud peamiselt seetõttu, et see võimaldab keelte väljatöötajatel luua keerulisi graafilisi lahendusi ja aitab kaasa keelte kasutajatele sobiva tööriista loomisele ning selle disainimisele [9].

Platvormi kasutamine dikteerib mudelipõhise arendustöö jaoks kindla metoodika, mille kohta täpsem info esitatakse peatüki 3. Metoodika all.

1.3 Ülesandepüstitus

1.3.1 Eesmärk

Töö eesmärgiks on valdkonnaspetsiifilise graafilise (modelleerimis)keele ning sellel põhineva tööriistakasti prototüübi väljatöötamine ülikooli lõputööde juhendamise ühe alamvaldkonna jaoks *Eclipse Sirius* platvormil.

1.3.2 Oodatavad tulemused, uurimisküsimused

Kavandatavateks põhitulemusteks on lõputööde juhendamise valdkonna ühe alamvaldkonna (juhendatava ja juhendaja kokkuviimine ehk teemade ja teemavaldkondade ’turuplats’) mudelid, graafilise keele (abstraktse ja konkreetse süntaksi) ning keele tööriistade spetsifikatsioonid pluss genereeritud tarkvaraprototüüp antud platvormil.

Konkreetse disainitöö käigus ja selle kaudu otsitakse vastused järgmistele uurimisküsimustele:

- Milliste valdkondade/probleemide lahendamiseks valdkonnaspetsiifiliste modelleerimiskeelte (VSMK) lähenemine sobib?
- Kas ja kuidas saab kasutada *Eclipse Siriuse* platvormi nende valdkondade ja probleemide lahendamiseks?
- Milline on (tulemuslik) arendustöö protsess seda platvormi kasutades?
- Milliste piiride ja piirangutega peab arvestama selle platvormi tööriistade kasutamisel?

1.4 Töö struktuuri ülevaade

Antud töö koosneb 6 peatükist. Esimeses osas keskendutakse töö teoreetilistele alustele, kus põhjalikumalt selgitatakse valdkonnaspetsiifilise keele tausta, *MDA*-d ja analüüsimustreid. Sellele järgnevas peatükis on esitatud töö metoodika, milles räägitakse kasutatud meetoditest ja tarkvarast. Realisatsioonis antakse ülevaade tehtud tööst, kajastatakse realiseerimisstrateegiaid, töös loodud mudeleid ja graafilist liidest (prototüüpi). Analüüsi peatükis vaadeldakse tehtud töö tulemusi, räägitakse alternatiividest ning kirjeldatakse edasisi samme. Viimaseks osaks on kokkuvõte ja järeldused.

2 Teoreetilised alused

Selles peatükis kirjeldatakse töö teoreetilisi aluseid.

2.1 Valdkonnaspetsiifiline keel (VSK)

Valdkond (ehk domeen) on teemavaldkond või huvipiirkond, mida saab iseloomustada temas sisalduvate objektide, nende objektide omaduste ning objektide poolt võimaldatavate funktsioonide kaudu. Objektid hõlmavad füüsilisi objekte, inimtegutsejaid ja mittemateriaalseid objekte [2], kelle/mille koostöös lahendatakse valdkonnaspetsiifilisi probleeme ja ülesandeid. Valdkonnaspetsiifiline keel (VSK) on spetsiaalne abivahend teatud kindla piiratud probleemi lahendamiseks. Valdkonnaspetsiifilisi keeli kasutatakse spetsiifilistes valdkondades, kus neid luuakse eesmärgiga toetada valdkonnas sisalduvaid kindlat tüüpi ülesandeid [1].

Valdkonnaspetsiifilisi keeli on mitmesuguste sihtotstarvete ja võimalustega ning nende kasutajaskond võib varieeruda. Kasutajateks võivad olla programmeerijad, kes valdavalt toimetavad rohkem neile suunatud tehnilisemate valdkonnaspetsiifiliste keeltega, kuid on ka teisi keeli, mis leiavad rakendust teiste töötajate käe all, seal kasutatakse kergemaid süntakseid, sealhulgas graafilisi.

Üldotstarbelised programmeerimiskeeled on mõeldud programmide kirjutamiseks ja neid kasutatakse arvutile probleemi arusaadavaks tegemiseks, struktureerimiseks. Valdkonnaspetsiifilised keeled on aga piiratuma ulatusega ja seetõttu keskendutakse nendega kitsamas valdkonnas leiduvate probleemide lahendamiseks, mistõttu on nad mingites ülesannetes üldistes programmeerimiskeeltest paremad.

Näitena võib tuua siinkohal ühe VSK-na *DOT* keele, mis on sobilik tavakasutajale ja võimaldab kirjeldada graafe, kus saab ära näidata klasse, nendevahelisi seoseid ja muid karakteristikuid [1].

2.1.1 Valdkonnaspetsiifiliste keelte loomine, jagunemine

VSK loomisel on abiks tööriistade kasutamine ning sõltuvalt sellest, millise VSK-ni soovitakse jõuda, jaguneb liigitus kas tekstikeele, graafilise keele või projektsiooni

redaktori valikute vahel. Neid tööriistu kasutatakse eesmärgiga luua keel mõistliku ressursikasutusega. Edaspidi keskendutakse peamiselt graafilistele keeltele [1].

2.1.2 Valdkonnaspetsiifiline modelleerimiskeel

Valdkonnaspetsiifilisi graafilisi keeli võib lugeda valdkonnaspetsiifilisteks modelleerimiskeelteks (VSMK). Ühe levinuima VSMK redaktorina võib välja tuua *Eclipse Siriuse*, mis kasutab muuhulgas ka *GMF-i* ehk graafilise modelleerimise raamistikku ja on pigem mudelikeskne [1].

Modelleerimiskeel on keel kontseptuaalsete mudelite (edaspidi nimetame neid „mudeliteks“) klassi loomiseks. Ta võimaldab selleks vajaliku mõistete hulga. Kontseptuaalse modelleerimise keel defineeritakse (tema) süntaksi ja semantika kaudu. Abstraktne süntaks annab reeglid korrektse süntaksiga mudelite konstrueerimiseks keele mõisteid kasutades. Konkreetne süntaks defineerib sümbolid (märgid), mida kasutatakse abstraktse süntaksi esindamiseks. Kuna need sümbolid on tavaliselt graafilised, nimetatakse seda ka graafiliseks notatsiooniks (esituseks). Modelleerimiskeele semantika defineerib modelleerimise mõistete (formaalse) interpretatsiooni. Modelleerimiskeele abstraktse süntaksi ja semantika vahel puudub selge erinevus.

Modelleerimiskeel võib rohkem või vähem sobida konkreetse modelleerimisülesannete klassi jaoks. See tekitab küsimuse: kui palju valdkonnaspetsiifilist semantikat peaks üks modelleerimiskeel sisaldama? See viib meid valdkonnaspetsiifiliste modelleerimiskeelte (VSMK) ja üldotstarbeliste modelleerimiskeelte (ÜOMK) eristamiseni.

ÜOMK on modelleerimiskeel, mis on mõeldud olema sõltumatu konkreetsest probleemvaldkonnast (diskursuse valdkonnast). Ta koosneb üldistest modelleerimismõistetest, mis ei sisalda konkreetse probleemvaldkonna ühtegi aspekti. ÜOMK annab oma kasutajatele baasmõisted, nagu „olemitüüp“, „klass“, „atribuut“ jne, mida kasutatakse vastava probleemvaldkonna oluliste mõistete rekonstrueerimiseks. Samas on aga olemas nende baasmõistete vastavused infosüsteemi disaini mõistetega. Näiteks *UML* ja *BPMN* notatsioone peetakse tüüpilisteks ÜOMK-ks [2].

VSMK on modelleerimiskeel, mis on mõeldud kasutamiseks kindlas probleemvaldkonnas. Ta rikastab üldisi modelleerimismõisteid vastavas probleemvaldkonnas kasutatavate mõistetega, mis rekonstrueeritakse (ÜOMK-te)

tehnilistest baasmõistetest. VSMK on mõeldud konkreetse valdkonna (millega keel seotud on) kontseptuaalsete mudelite loomiseks.

Arendustöö VSMK-t kasutades võib olla seotud ka valdkonnaspetsiifiliste programmeerimiskeelte ehk VSPK-ga. Arenduse ideaalsel viisil tegemisel on võimalik nt VSMK mudel konverteerida ümber VSPK programmikoodi. Ühtlasi on VSPK puhul samuti programmeerijatele kättesaadavad valdkonnaspetsiifilised kontseptsioonid, mis tagavad efektiivsema tarkvara integreerituse taseme ja tõstavad ka produktiivsust [2].

Konkreetse keele (nt *UML*, *BPMN*) üldotstarbeliseks (ÜOMK) või valdkonnaspetsiifiliseks keeleks (VSMK) olemine on suhteline – see on tõlgendamise küsimus. Näiteks *UML* ja *BPMN* keeli, mida tavaliselt peetakse üldotstarbeliseks, saab tõlgendada vastavalt tarkvara ja äriprotsesside valdkondade suhtes valdkonnaspetsiifiliste keeltena [14].

Valdkonnaspetsiifilised keeled alluvad mudelitega juhitava arhitektuuri (*Model Driven Architecture* ehk *MDA*) kontseptsioonile.

2.2 MDA – mudelitega juhitav arhitektuur

Selles peatükis räägitakse *MDA* ajaloost, sisulisest poolest, *MDA* tüüpi lähenemisest, *MDA* tööriistadest ja *MDA* kasutamisest töös.

2.2.1 MDA ajalugu, ülesehitus

MDA ehk *Model Driven Architecture* on mudelitega juhitaval tarkvara disainil põhinev lähenemine. See anti välja aastal 2001 *Object Management Groupi (OMG)* poolt. *MDA* toetab mudelipõhist tarkvarasüsteemide rakendamist andes juhtnöörid mudelitena väljendatavate tarkvarasüsteemi spetsifikatsioonide tegemiseks ja struktureerimiseks. *MDA* metoodika põhjal võib selle süsteemi funktsionaalsust kirjeldada kui platvormist sõltumatut mudelit ehk *PIM*i, kasutades kirjeldamiseks sobivat valdkonnaspetsiifilist keelt (VSK-t). *PIM* kujutab endast kontseptuaalset disainimudelit, mis realiseerib funktsionaalseid nõudeid. Süsteemile antakse järgnevalt platvormi definitsiooni mudel *(PDM)*, mis vastab näiteks *CORBA*-le ja *.Net-le*. Seejärel tõlgitakse *PIM* ümber üheks või mitmeks platvormispetsiifiliseks mudeliks *(PSM-ks)* rakendamise osa jaoks, tehes seda erinevate valdkonnaspetsiifiliste keelte või üldisemate eesmärkidega

programmeerimiskeelte *Java*, *Pythoni*, *C#* abil. *PIM*-i ja *PSM-i* vahelised tõlkeprotsessid tehakse tavaliselt automaatsete mudeli teisendamiste tööriistadega. Selleks võib olla näiteks tööriist, mis kuulub *OMG* standardi *QVT* alla. *MDA* printsiipe saab rakendada ka äriprotsesside mudelitega, kus arhitektuurid ja tehnoloogiad saavad jääda neutraalseks *PIM*-del, mida kasutatakse manuaalsete ja süsteemiprotsesside kontseptuaalsel disainimisel. Kogu protsess on kirjeldatud *MDA Guide* dokumendis, mida haldab ja toodab *Object Management Group (OMG)* [4].

2.2.2 MDA lähenemine

MDA-l põhineva lähenemise üks põhieesmärkidest on eraldada disain tehnilisest arhitektuurist ehk realiseerimistehnoloogiatest. Sellega tahetakse näidata, et disain ja arhitektuur saavad muutuda iseseisvalt. Disain kajastab funktsionaalseid nõudmisi, kasutusjuhtusid *(use case-e)*, aga arhitektuuri osa annab infrastruktuuri, kus realiseeritakse mittefunktsionaalsed nõuded, nagu usaldusväärsus, mõõdetavus ning jõudlus. Kuna realisatsioonitehnoloogiad ja tarkvaraarhitektuurid pidevalt muutuvad, siis *MDA* näeb ette, et *PIM-id*, platvormist sõltumatud mudelid, elavad muudatused edukalt üle. Eelnevalt sai mainitud, et *QVT* on üks mudeliteisendamise *OMG* standardeid. Võib veel välja tuua, et oluliseks lüliks *MDA* lähenemisel on kindlasti mudeli teisendumine, transformeerumine, millega saab edukalt hakkama näiteks *QVT* standardi keel *ATL* [4].

2.2.3 MDA tööriistad

MDA tööriistadega on võimalik mudeleid või metamudeleid tõlkida ja tõlgendada, arendada, mõõta, teisendada jne. *MDA* puhul võib täheldada kahte tüüpi mudeleid. Üheks nendest on metamudelid, mis on loodud inimeste-töötajate poolt, teiseks aga arvutite programmide poolt loodud tuletatud mudelid. Metamudel võib olla näiteks *UML* diagramm, mille analüütik loob ärisituatsiooni aluseks võttes. Samas tuletatud mudel võib olla *Java* mudel, mis on tuletatud sellestsamast *UML* diagrammist, kasutades mudeli teisendamise operatsiooni. *MDA* mudelid võivad täita üht või mitut tüüpi funktsioone, milleks võivad olla looming, analüüs, teisendumine, kompositsioon, testimine, metaandmete juhtimine [4].

2.2.4 MDA lähenemise eelised

MDA lähenemisega saab välja tuua järgnevad eelised:

- *MDA* lähenemisega võib tõlgendada koodi kirjutamist mudeli loomisena, mis võimaldab isegi mitteprogrammeerijatel antud lähenemist kergemini rakendada ja mõista.

- Erinevalt programmeerimiskeeltest, milles luuakse esmalt kood teatud mudeli saamiseks, saab nüüd programmeerija MDA tööriistadega esmalt modelleerida süsteemi, tehes hiljem vaid mõningaid vajalikke kohandusi. Ühtlasi võib siinkohal välja tuua, et osade *MDA* tööriistade kasutamisega on tööefektiivsus suurenenud lausa 40 protsendi võrra, võrreldes mõne teise programmeerimiskeele tarkvaraga.

- Arendaja poolt loodud/kasutatavad erinevad platvormispetsiifilised mudelid (*PSM*-d) on kõik omavahel ühendatud, sest need ühendused genereeritakse automaatselt.

- Teoreetiliselt ei tohiks tekkida probleeme, kui juba näiteks loodud *PIM*-i ehk platvormist sõltumatut mudelit kasutada teistel, samuti *PIM*-e loomist võimaldavatel platvormidel [15].

- *MDA* suudab edukalt eristada disaini tehnilisest arhitektuurist ehk realiseerimistehnoloogiatest.

- *MDA*-l on palju erinevate funktsionaalsustega tööriistu, mis võimaldavad mudeleid tõlkida, tõlgendada, teisendada, arendada ja mõõta [4].

2.2.5 MDA-ga kaasnevad puudused

MDA lähenemisega kaasnevad ka mõningad puudused, mis on järgnevalt loetletud:

- Praegused *UML* standardid ei pruugi vastata praeguse tarkvaraturu nõudmistele. Samuti pole lõpuni arendatud või täiesti omaks võetud mõned *MDA* standardid, nagu näiteks *Java* metaandmete kasutajaliides *JMI*.

- *UML* keerukas tehniline pool võib põhjustada modelleerimise omaksvõtuga probleeme, seega *UML* vajab üldjuhul andekat modelleerijat [8].

- Mitmed teisendamislähenemised võivad olla erinevate ülesehitustega ja seetõttu ka erinevate omadustega. Rakendustel on varieeruvaid nõudmisi, seetõttu võib nende lähenemiste sooritusvõimete võrdlemine olla raske.

- Mõnel programmeerijal võib olla keeruline kohaneda, kuna *MDA* potentsiaali kohta on erinevaid arvamusi ja kohati ka teadmisi. *MDA* lähenemisega tuleb omaks võtta innovatiivse ja võibolla senisest parema tehnoloogia kasutamine. Samas võidakse *MDA* rakendamisel võtta ka kahtlustavaid seisukohti [8].

2.2.6 MDA seos käesoleva tööga

Antud töö põhineb samuti *Model Driven Architecture (MDA)* lähenemisel, mille üheks haruks ehk arengusuunaks ongi valdkonnaspetsiifilised (modelleerimis)keeled. Valdkonnaspetsiifilise keele abstraktse süntaksi saab defineerida *PIM*-na üldotstarbelises (modelleerimis)keeles (nt *UML* klassidiagramm või sellega sarnanev *Ecore* diagramm) ning teisendada läbi vastavate *PSM*-e näiteks Java ja/või muudel platvormidel (koos)töötavate rakenduste *back-end*'iks. Vastava kasutajaliidese ehk *front-end*'i mudelipõhine arendamine on siis tihedalt seotud eelnimetatud (modelleerimis)keele konkreetse süntaksi (näiteks diagrammitüüpide) defineerimisega [14].

Järgnevalt uurime konkreetsete analüüsimustrite võimalikku kasutamist VSMK abstraktse süntaksi modelleerimisel käesoleva töö kontekstis.

2.3 Analüüsimustrid

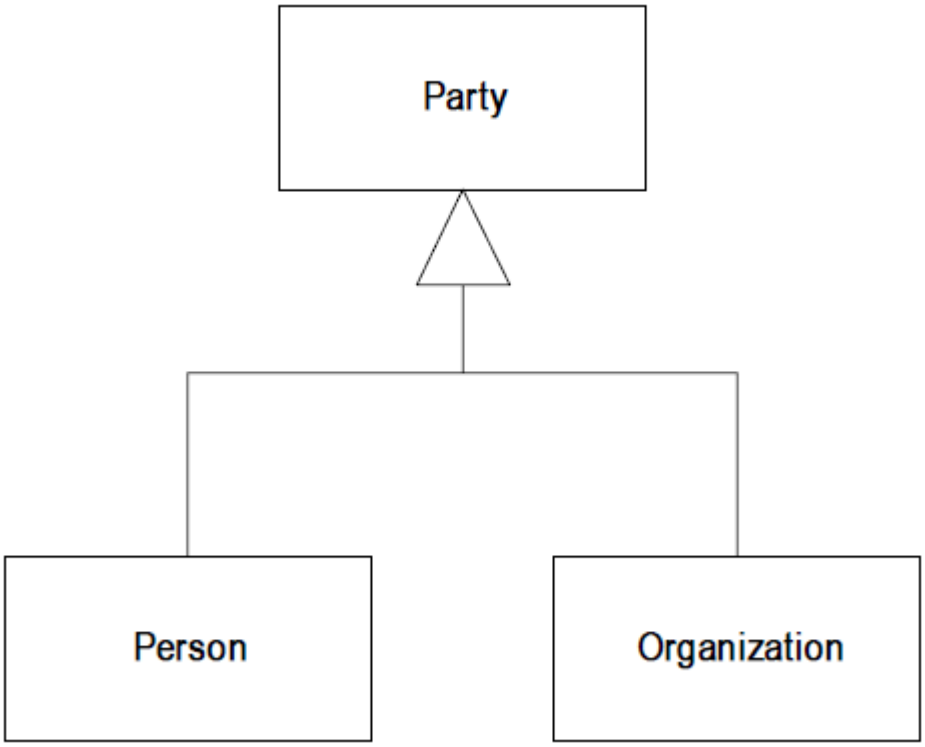
Valdkonnaspetsiifilise keele prototüübi loomine algab konkreetse valdkonna analüüsist, mis omakorda võib põhineda analüüsimustritel. Analüüsimuster on ekspertide poolt loodud taaskasutatav (ja tarkvara realisatsiooni omav) objektmudel või domeenimudeli fragment. Iga selline muster lahendab kindlat (valdkonnaspetsiifilist) ülesannet ja on seega arendatav vastava VSK (abstraktse süntaksi) mudeliks.

Antud töös kasutatakse (spetsialiseeritakse) mõnda (struktuuride modelleerimist toetavat) analüüsimustrit valdkonnaspetsiifilise modelleerimiskeele prototüübi loomiseks. Järgnevalt tutvustame selle prototüübi üheks aluseks olevat (organisatsiooni) struktuuride modelleerimist toetavat analüüsimustrit *Accountability* (Kohustus) ning sellega otseselt seotud mustreid *Party* (Osapool), *Knowledge Level* (Teadmustase).

2.3.1 *Party* (Osapool) muster

Esimese analüüsimustrina võib kirjeldada osapoole ehk *Party* mustrit. Organisatsiooni struktuuri modelleerimisel kõige tähtsamaks mõisteks on osapool (*Party*). Osapool on abstraktsioon üle inimeste (*Person*) ja organisatsiooniüksuste (*Organization*). Seda üldistust läheb sageli vaja: näiteks kontaktandmed (aadressid, telefoninumbrid jne.) on sarnased nii isikute kui ka organisatsioonide omaduste hulgas. Ettevõtete füüsilisest isikutest ja organisatsioonidest klientidel on palju ühesuguseid omadusi ja seoseid, jne.

Osapooled võivad olla omavahel seotud läbi erinevate seoste. Sarnaselt inimosapooltele (nt isikud ja organisatsioonid) võivad nendega sarnaselt seoseid ja struktuure moodustada ka muud (käegakatsutav kui ka abstraktsed) objektid, nagu tarkvarakomponendid ja -rakendused, rollid ja rollitüübid, erinevad tooted/teenused ja nende paketid, ained ja õppekavad, lõputööde teemad ja teemavaldkonnad. Seega on võimalik ja kindlates kontekstides ka vajalik osapoole mõistet laiendada ning temaga seotud mustreid üldistada (mida käesolevas töös ongi tehtud). Järgmiselt on kirjeldatud joonisel 1 kujutatud *Party* mustrit [16].

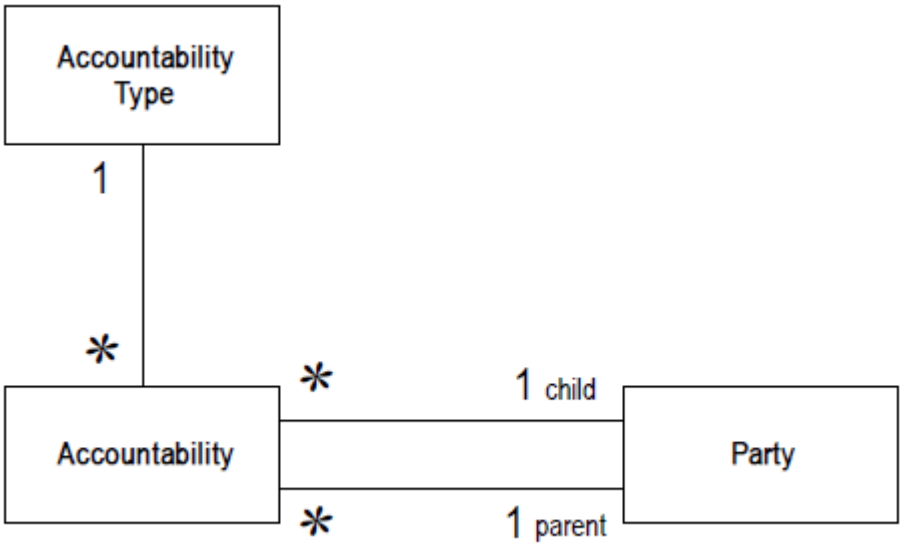


Joonis 1. Näide: *Party* mustri kirjeldus.

Allikas: *Martin Fowler, Organization Structures (Accountability)* [16].

2.3.2 *Accountability* (Kohustus) muster

Üheks põhiliseks antud töös kasutatud mustriks on *Accountability* (mis tähendab kohustust või vastutust), mida saab kasutada kõikvõimalike, sealhulgas ajas dünaamiliselt muutuvate mittehierarhiliste (ehk võrkstruktuuriga) organisatsioonide struktuuri kirjeldamiseks. Sisuliselt antud muster võimaldab kirjeldada erinevat tüüpi seoseid ja nende tähendusi erinevate osapoolte (nt isikud, organisatsioonid – vastavalt analüüsimustrile *Party* ehk osapool) vahel. Allpool leitavalt jooniselt 2 on võimalik näha, et osapool *Party* võib olla kas vanema või lapse rollis ühendatud kindla *Accountability* klassi esindaja ehk vastutusega. Sisuliselt tähendab see seost kahe erineva osapoole vahel, kusjuures vanemosapool omab vastutust (või kohustust) lapsosapoole suhtes, seega seostel on kindel suund ning igat kirjeldatavat struktuuri saab väljendada suunatud graafina. *Accountability Type* kirjeldab, mis tüüpi vastutuse ehk seosega on tegu ning neid tüüpe võib luua vastavalt vajadusele. Siinkohal on rakendatud teist analüüsimustrit nimega *Typed Relationship* [16].



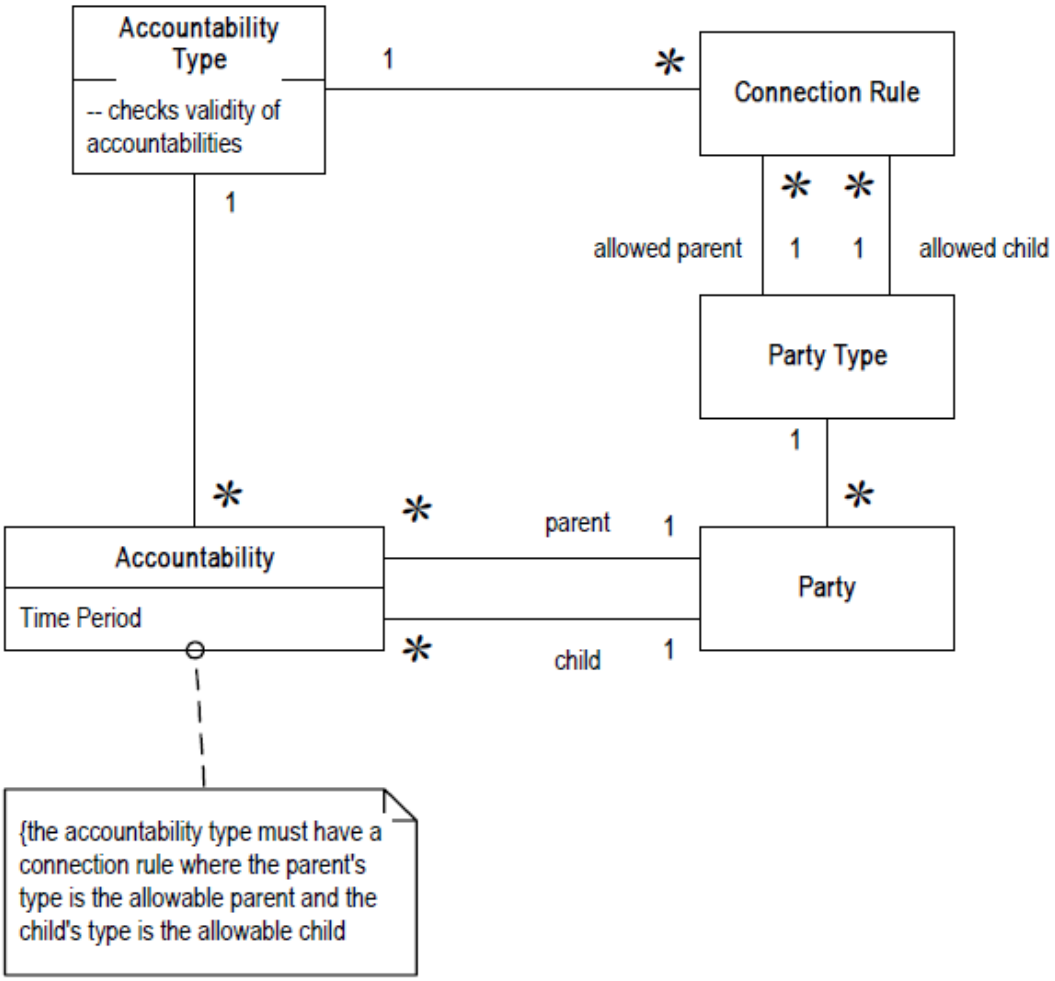
Joonis 2. Näide: *Accountability* mustri kirjeldus.

Allikas: *Martin Fowler, Organization Structures (Accountability)* [16].

Accountability mustrit (mis omakorda kasutab Party ja Typed Relationship analüüsimustreid) on vajadusel võimalik kasutada koos mustriga Knowledge Level (Teadmustase), nagu järgmisel joonisel 3 on näha [16].

2.3.3 *Knowledge Level* (Teadmustase) muster

Knowledge Level on muster, mis defineerib kuidas üks objektide grupp mõjutab teise objektide grupi käitumist. See laseb teatavate reeglite sissetoomise abil määrata, millist tüüpi osapooled võivad olla omavahel ühendatud. *Knowledge Level*i keskseks mõisteks on ühendamise reegel *Connection Rule*. Uue *Accountability* loomisel kontrollib *Accountability* automaatselt vastutusetüüpi (*Accountability Type*) ja selle põhjal määrab ära, kas *Accountability* on loodud korrektselt. *Accountability Type* sisaldab mingit hulka ühendusreegleid, mis määravad ära, et *Parent* ja *Child* ühendused oleksid õigesti koostatud ehk vanem- ja lapsosapool oleksid lubatud tüüpi. Seega, kui on soov mingeid reegleid muutma hakata, tuleb alustada vastutusetüübi muutmisest või asendamisest. Allpool asuval joonisel 3 on ära näidatud *Connection Rule* kasutamine [16].



Joonis 3. Näide: *Knowledge Level* mustri kirjeldus.

Allikas: *Martin Fowler, Organization Structures (Accountability)* [16].

3 Metoodika

Metoodika peatükis kirjeldatakse töös kasutatavaid tarkvarasid. Samuti antakse põhjalik ülevaade vajaminevatest etappidest prototüübi loomiseks.

3.1 Töös kasutatavad tarkvarad

Konkreetses töös on kasutuses tarkvaraplatvorm nimega *Eclipse Sirius*. See on vabavaraline toode, mille on omandanud ja mille arendusega tegeleb *Obeo* firma. *Eclipse Siriust* hakati kasutajatele pakkuma samuti vabavaralise *Obeo Designer Community* versiooni nime all. Antud töös kasutatavaks versiooniks on 11.1.0. *Eclipse Siriuse* platvorm on valitud peamiselt seetõttu, et see võimaldab luua keerulisi graafilisi lahendusi tarkvaraarendajate poolt ja annab võimaluse luua enda valdkonnale sobiv tööriist ning seda disainida. *Siriuse* platvorm on kättesaadav vabavarana iga-aasta uuendatavate versioonidena. Seetõttu tagatakse ka läbipaistvuse põhimõte ning platvormi kõrge kvaliteet. Järgmises alampeatükis räägitakse lähemalt *Ecore* valdkonna- ehk metamudelite ja sellel tugineva graafilise liidese prototüübi loomise etappidest [9].

Töös pideva turvalisuse hoidmiseks ja koopiate tagamiseks on kasutusel olnud keskkond *GitHub*, mis võimaldas tehtud töid pidevalt varundusena salvestada [10].

Mudelite lihtsustatud eskiiside modelleerimiseks sai kasutatud programmi *Draw.io* [11].

3.2 Vajaminevad etapid prototüübi loomiseks

Töö esmaseks etapiks on valdkonna analüüs, kus vaadeldakse, mis objektid peaks olema mudelis olulised, mille jaoks neid on vaja ja kuidas on nad omavahel seotud. On tarvis läbi mõelda valdkonnaspetsialisti soovid ja nõuded, et tarkvaraarendaja saaks eesmärgist ja idee sisust õigesti aru. See tuleb muidugi välja ka edaspidises tarkvaraarendaja ja kindla valdkonnaspetsialisti koostöös.

Järgmisena võetakse kasutusele *Eclipse Siriuse* tarkvara. Alustatakse valdkonna- ehk metamudeli (*Ecore* mudeli) loomisega *Eclipse Siriuse* platvormil. Sellel platvormil luuakse esmalt metamudel(id) tarkvaraarendaja poolt. Mudelis täpsustatakse konkreetsed objektid ja nendevahelised seosed. Nad on suhteliselt abstraktsed, olles aluseks lisana

hiljem loodavale graafilisele liidesele. Objektid ja seosed seovad mudeli kokku arendatavaks, abstraktseks ühtseks süsteemiks. Metamudelist tulevad metaandmed graafilise liidese jaoks, mida kasutatakse järgmistes etappides [17], [18].

Eclipse Siriuse platvormil VSMK prototüübi loomise protsess on iteratiivne, kusjuures iga iteratsioon koosneb kahest etapist, milleks on esiteks disainiaja (*design time*) tegevused ning teiseks kasutusaja ehk täitmisaja (*runtime*) tegevused ning kolmest põhitegevusest:

- Valdkonna (meta)modelleerimine ehk keele abstraktse süntaksi kirjeldamine (*Ecore* mudelina) koos vastava rakenduskoodi genereerimisega, mis on disainiaja põhitegevus (sai eelnevalt põgusalt kirjeldatud);
- Graafilise esituse (nt diagrammitüübi) kui keele konkreetse süntaksi kirjeldamine koos vastavate tööriistade (kui keele semantika komponendi) spetsifitseerimine, mis võib toimuda kas disaini või kasutuse etapis;
- Keele kasutamine, mis on täitmisaja ehk kasutusetapi põhitegevuseks ning võib toimuda kas testkasutamisena *Eclipse Sirius runtime* keskkonnas või lõppkasutamisena väljaspool nimetatud keskkonda (lõppkasutamisele eelneb keele versiooni publitseerimine).

Eelnimetatud tegevused toimuvad tarkvaraarendaja ja valdkonnaspetsialisti koostöös, kes täidavad vastavalt keele spetsifitseerija ja keele kasutaja rolle. Kuna lihtsamaid keeli on *Siriuse* platvormil võimalik spetsifitseerida ilma otseselt ise koodi kirjutamata (*MDA* lähenemist kasutades), siis põhimõtteliselt võiks valdkonnaspetsialist osaliselt täita ka keele spetsifitseerija rolli (kasutusaja, mitte disainiaja etapis).

4 Realisatsioon

Selles peatükis kirjeldatakse töös kasutatud realiseerimise strateegiat, metamudeli arhitektuuri ning samuti autori poolt loodud valdkonnamudeleid ja graafilise keele tarkvaraprototüüpi *Eclipse Siriuse* platvormil.

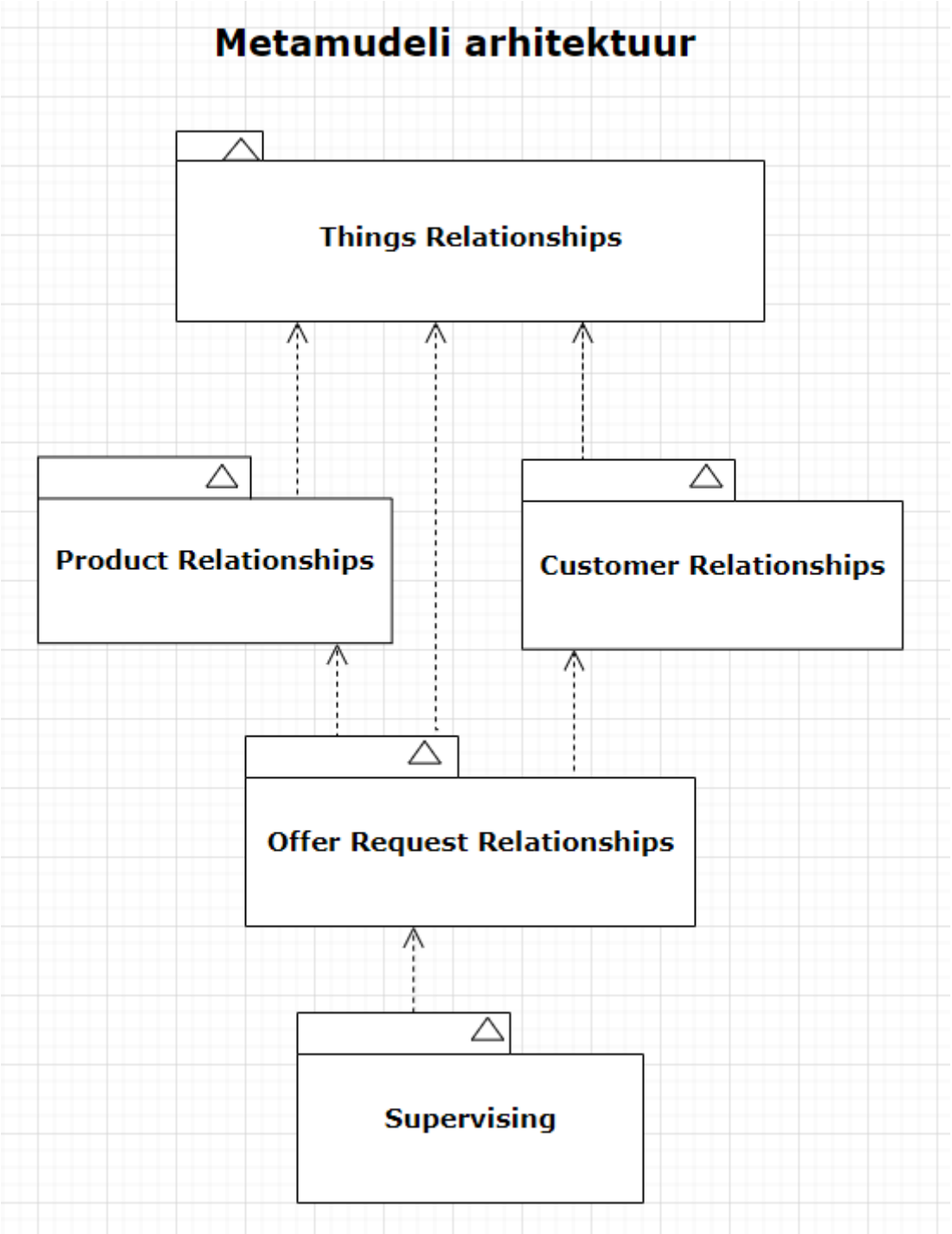
4.1 Realiseerimise strateegia

Realisatsioon toetub osas 2.3 kirjeldatud analüüsimustritele, mille hulgast keskseks on peatükis 2.3.2 tutvustatud *Accountability* (Vastutus/Kohustus) muster. Peatükis 2.3.3 sisse toodud mustri *Knowledge Level* (Teadmustase) võimalusi pole veel jõutud ära kasutada, ent põhimõtteliselt saab prototüübi edasiarendusel ka antud mustrit rakendada. Töö lähteülesande püstitamisel polnud praeguseks realiseeritud prototüübi täpne probleemvaldkond (Lõputööde juhendamine: teemade, teemavaldkondade 'turuplats') veel defineeritud. Sihiks oli võetud struktuuride graafiline modelleerimine üldisemalt, esmalt laiemas valdkonnas. Teada oli, et *Siriuse* platvormi lähenemine selleks ülesandeks kindlasti sobib. Tugineti paljuski ühele õpetlikule selle platvormi juhendile, mis kirjeldab näitevaldkonnana perekonda (*BasicFamily*), mille põhielemendid kujutavad perekonnaliikmeid ning nendevahelisi vanemlikke seoseid (nt lastevanemate ja laste vahel) [18]. Seetõttu on metamudelit arendatud sammukaupa 'ülalt-alla' ehk üldisemalt konkreetsemale valdkonnale ning selle metamudelile liikudes.

Kõige üldisema metamudelina sai esmalt loodud *Ecore* mudel nimega *Things Relationships* (asjade seosed), mis järgib ja mõistete osas pisut üldistab ('organisatsiooniline struktuur' versus 'asjade struktuur') analüüsimustrit *Accountability*. Sellest metamudelist on spetsialiseerimise teel (lõppkokkuvõttes) tuletatud ülejäänud valdkonnad ja nende metamudelid kuni kõige konkreetsema *Supervising* (juhendamine) mudelini, mille kohta sai protsessi käigus võetud eesmärgiks jõuda modelleerimiskeele prototüübi realisatsioonini. Ülejäänud üldisemate valdkondade/keelte realisatsioonideni on võimalik jõuda edaspidi 'alt-üles' ehk konkreetsemalt keelelt üldisemale keelele liikudes. Järgnevalt kirjeldatakse eelkirjeldatud protsessi esimese etapi tulemuseks oleva metamudelite süsteemi (arhitektuuri).

4.2 (Meta)mudeli arhitektuur

Selle osa eesmärgiks on anda esmane ülevaade kõigist loodud metamudelitest ja nende sõltuvusseostest. Metamudelite sõltuvuste kirjeldamiseks sai tehtud järgnev ülevaateskeem (*UML* paketidiagramm joonisel 4). Iga metamudeli ja tema seoste kohta käiv täpsem kirjeldus antakse aga järgmises 4.3 valdkonnamudelite peatükis.



Joonis 4. Metamudeli arhitektuuri kirjeldus.

Allikas: *Draw.io* [11].

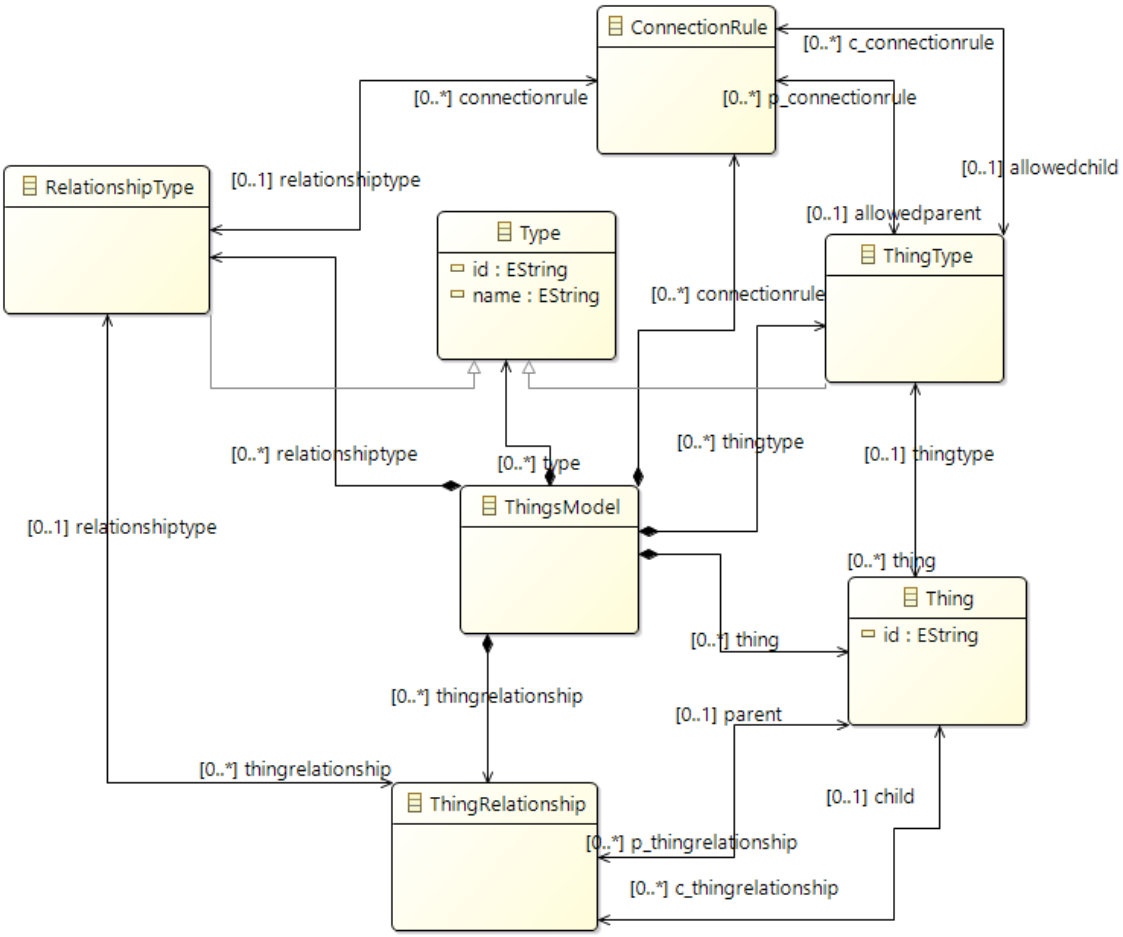
Lihtsustatud eskiisil on iga loodud *Ecore* metamudel kirjeldatud visuaalselt *UML* paketidiagrammis (*UML Package Diagram*) ühe paketina. Sõltuvusnoolte abil näidatakse, mis mudel on teisest sõltuv ehk kasutab teise mudeli elemente. Esimesena sai loodud kõige üldisemaks valdkonnaks ja metamudeliks olev *Things Relationships* (metamudel), mis väljendab kõikvõimalike asjade struktuuri ja on skeemil kujutatud

kõige ülemisel kihil. Selle alusel loodi ülejäänud spetsiifilisemad mudelid, mis spetsialiseerivad kõik Asjadeseoste osasid põhiklasse ehk elemente, igaüks oma valdkonna spetsiifiliselt (just selle valdkonna elementide seoste ja struktuuride kirjeldamiseks sobivalt). Need on koos sõltuvussuhetega näidatud järgmisel – teisel kihil. Asjadeseostel tuginevad mudelid on Toodeteseosed (*Product Relationships*) ja Kliendiseosed (*Customer Relationships*), mis kasutavadki *Things Relationships* mudeli elemente. Ühtlasi on neil mudelitel loodud veel täpsustavad klassid oma valdkonna jaoks. Sellest järgneval kolmandal kihil on kirjeldatud Pakkumiste-Nõudmiste seosed (*Offer Request Relationships*). See kasutab imporditud klasse kõikidelt eelnevatelt nimetatud mudelitelt. Mudelil on eraldi loodud täpsustavad klassid, mis hakkavad viitama juba järgmisele alamvaldkonna mudelile, milleks on Juhendamise valdkond (*Supervising*). Kuna juhendamise (turuplatsi) valdkond on tugevalt sõltuv Pakkumiste-Nõudmiste mudelist, siis siin on sisuliselt kõik klassid spetsialiseeritud imporditud klassidest, mis on pärit sealt mudelist. *Supervising* mudelis on esindatud need lõplikult loodud elemendid, mis leiavad praktiliselt kasutust juba graafilise liidese prototüübis. Sellega on üldiselt ja lihtsustatult kirjeldatud metamudelite süsteem: nende metamudelite sõltuvusseosed, millega on selgeks tehtud, kustkohast metamudelid andmeid (elemente) pärivad.

4.3 *Eclipse Siriuse* valdkonna- ehk metamudelid

Valdkonnaspetsiifilise modelleerimiskeele prototüüp on loodud *Eclipse Siriuse* platvormi kasutades. Tarkvara abil on tehtud kokku 5 *Ecore*-i modelleerimise projekti (*Ecore Modeling Project*-i), nendes sisalduvad keele abstraktse süntaksina valdkonna mudelid ehk metamudelid. Loodud on eelmises peatükis mainitud Asjadeseoste (nimega *Things_relationships*) projekt, mis kirjeldabki väga laia mõistena asjade seoseid. Selline nimi valiti põhjusel, et saaks sellele tuginedes luua mitmeid spetsiifilisi ja seda väga üldist valdkonda kitsendavaid projekte. Selle projekti metamudelile tuginedes loodi näiteks sõltuvad Toodeteseoste ehk *(Product_relationships)*, Kliendiseoste *(Customer_relationships)*, Pakkumiste-Nõudmiste seoste *(Offer_Request_relationships)* ja Juhendamise teemat väljendava *(Supervising)* projektide mudelid. Kõik nimetatud mudelid tuginevad teooria osas mainitud *Party*, *Accountability*, *Knowledge Level* mustritele. Järgnevalt kirjeldame igat nimetatud metamudelit täpsemalt.

Vastavalt osas 4.1 kirjeldatud strateegiale modelleeriti ja koodi genereerimise kaudu implementeeriti esimesena Asjadeseoste (*Things Relationships*) metamudel, milles vajalikke klasse ja seoseid sai järgnevalt teistes projektides (taaskasutada). Valdkonnamudelite (metamudelite) keerukuse tõttu on mõned järgnevad mudelid joonistel esitatud tegelikult realiseerituga võrreldes vähendatud mahus. Asjadeseoste metamudeli (ilma laiendavate klassideta tuuma) *Ecore* esitus on välja toodud allpool joonisel 5, milles olevad põhiklassid leiavad kasutust ka sõltuvates metamudelites. Lisas 1 on välja toodud ka implementeeritud metamudel täismahus (kahel joonisel).



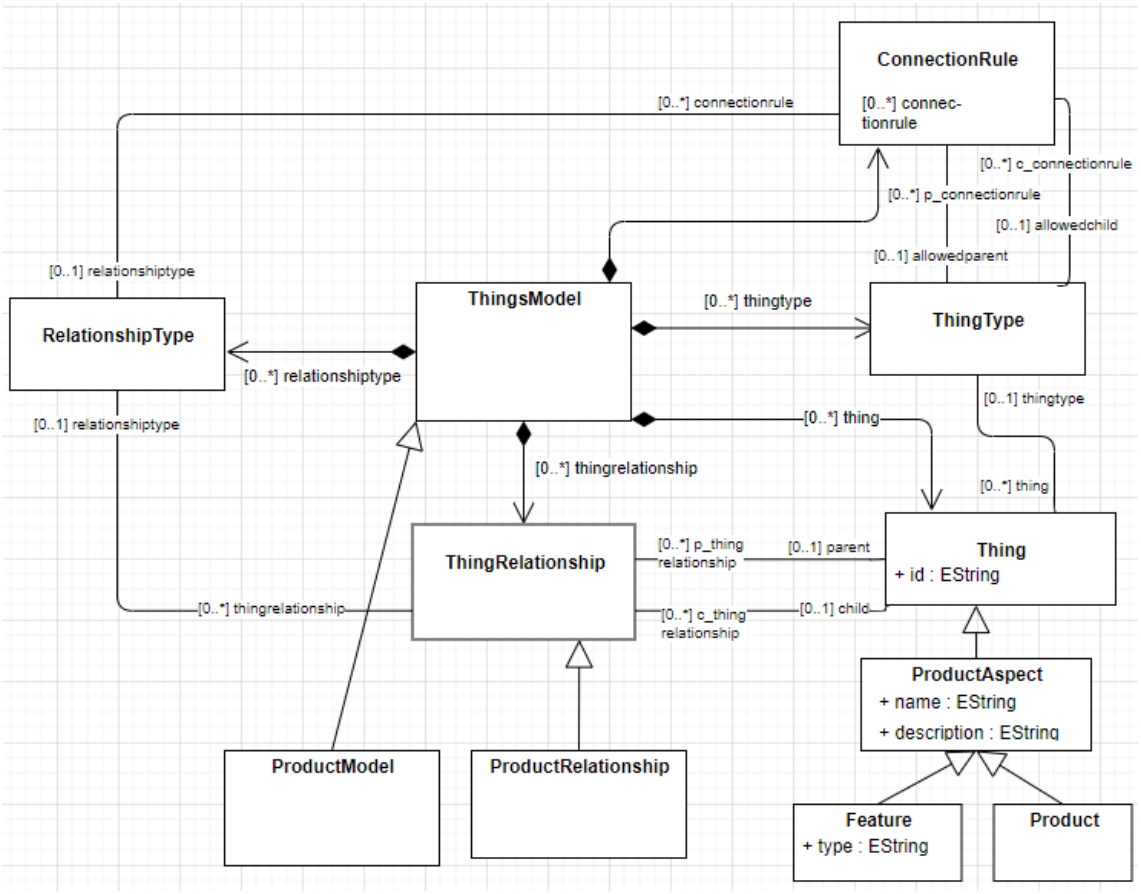
Joonis 5. *Things_relationships* metamudeli lihtsustatud kirjeldus.

Allikas: *Eclipse Sirius – Obeo Designer Community*.

See metamudel on tehtud osas 2.3 kirjeldatud analüüsimustrite alusel. Antud metamudelit esindav klass kannab nime *ThingModel*. See mudel sisaldab asju (*Thing*), nende seoseid (*ThingRelationship*), nende tüüpe (*ThingType*, *RelationshipType*), ning ühendamise reegleid (*ConnectionRule*). Asi (*Thing*) üldistab analüüsimustris *Party* defineeritud

osapoole mõistet, Asjadeseos (*ThingRelationship*) rakendab analüüsimustris *Accountability* defineeritud kohustuse/vastutuse (*Accountability*) mõistet: ’vanema’ rollis olev asi on seotud ’lapse’ rollis oleva asjaga läbi kindlat tüüpi asjadeseose (see on sisuliselt graafi mudel). Ühendamise reegel on sisuliselt sama mõiste, mis samanimeline mõiste analüüsimustris *Knowledge Level* (Teadmustase). Iga selline reegel on kolmepoolne seos konkreetse seosetüübi (*RelationshipType*) ja kahe erineva asjadetüübi (*ThingType*) vahel ning ’ütleb’, et antud tüüpi seostes (näiteks toote asendamine) on antud tüüpi ’vanema’ rollis asja (näiteks asendatav toode) ning antud tüüpi ’lapse’ rollis (näiteks asendav toode) ühendamine lubatud. Ehkki selline reeglite lahendus on implementeeritud (*Java* klassid genereeritud ning reegleid saab sisestada), pole nende reeglite kasutamiseni realiseeritud prototüübis veel jõutud (prototüüp on lahendatud ilma reeglitele häälestumata s.t. vähem paindlikult).

Üheks Asjadeseoste mudeli otseseks laienduseks (ilma vahendavaid metamudeleid kasutamata) on Toodeteseoseid kirjeldav *(Product_relationships)* metamudel, mis saab kirjeldada tooteid ja tootestruktuure näiteks e-poes (nt mööblipoes). Mudelis on loodud baasmudeli klasside spetsialiseerimise teel valdkonnaspetsiifilised toote, tootegrupi, tüüpide ja omadusteklassid ning sai tehtud ka tootespetsiifilised seoste klassid. Kõiki loodud valdkonnaspetsiifilisi klasse sisaldava valdkonnamudeli keerukuse tõttu on järgmised 2 mudelite joonist viidud lihtsustatud kujule ja illustreerivad üksnes laiendamise mehhanismi ning kõige kesksemaid spetsialiseeritud klasse. Tehtud eskiisid on loodud, kasutades tarkvara *Draw.io* veebilehel [11]. Lisas 2 on näha ka täismahus Toodeteseoste metamudel (kahel joonisel). Toodeteseoste metamudeli lihtsustatud eskiis joonisel 6 on järgmine:



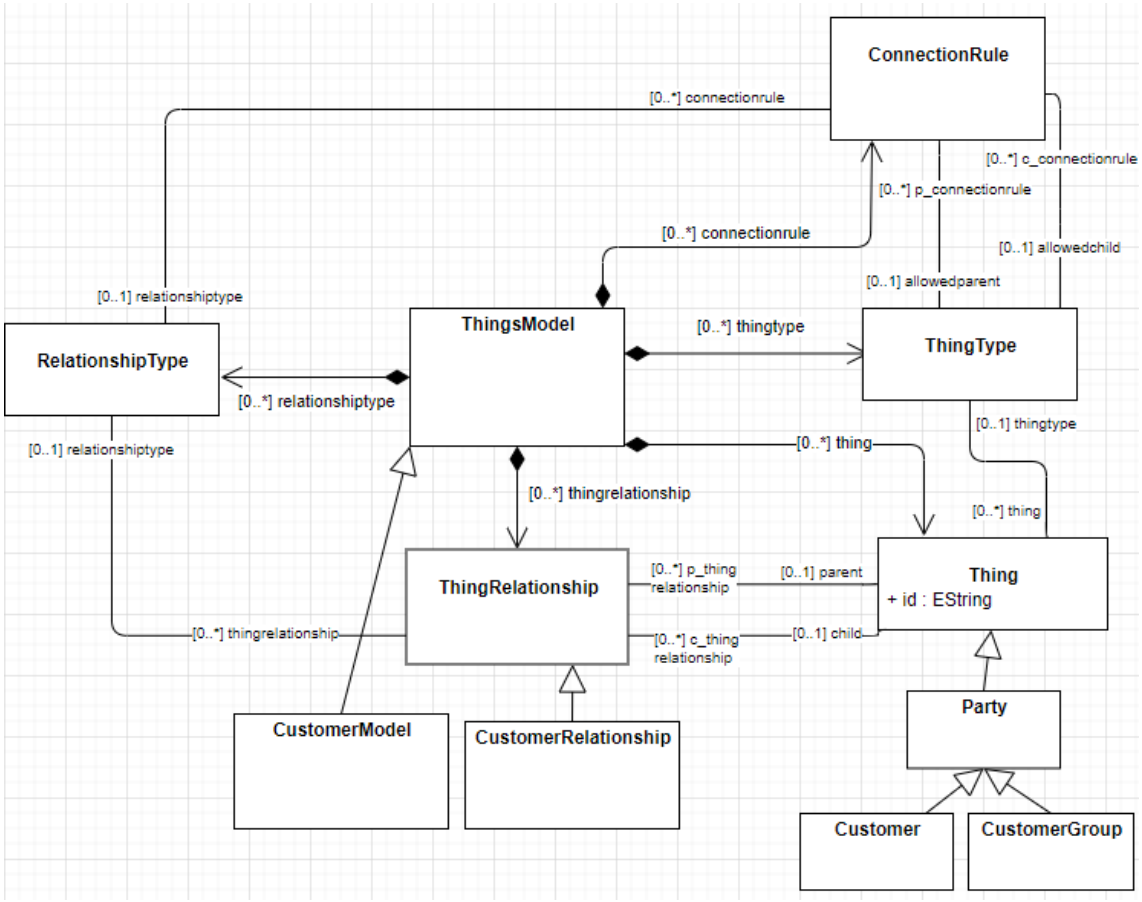
Joonis 6. *Product_relationships* metamudeli lihtsutatud kirjeldus.

Allikas: *Draw.io* [11].

Joonisel 6 on korratud eelmisel joonisel 5 nähtud Asjadeseoste (*Things Relationship*) metamudeli tuuma, mida on laiendatud toodete valdkonna spetsiifiliste klasside saamiseks. Nii näiteks toote aspekt (*ProductAspect*) laiendab baasmudeli abstraktset klassi *Thing*, tootemudelit esindav klass *ProductModel* laiendab baasmudelit esindavat klassi *ThingsModel*, Toodeteseost esindav klass *ProductRelationship* laiendab baasmudeli asjade seost esindavat klassi *ThingRelationship*. Toote Aspektiks (*ProductAspect*) võib olla kas Toode (*Product*) või selle omadus (*Feature*). Ülejäänud seni defineeritud/genereeritud valdkonnaspetsiifilisi klasse (näiteks kindlat tüüpi toodetevaheliste seoste realiseerimise lihtsustamiseks, ehkki seose tüübi kindlat klassifikaatorit kasutades saab paindlikult andmetega juhitavalt defineerida juurde uusi seosetüüpe) on illustreeritud töö lisades. Osas 3.2 kirjeldatud iteratiivset arendusprotsessi järgides on võimalik vajaduse korral siinkirjeldatud *Ecore* metamudeli lahendus (mis spetsialiseerib üldisemat baaslahendust) asendada konkreetsema ja samas vähempaindliku lahendusega (mis ei sõltu enam teistest metamudelitest). See võib olla

vajalik näiteks siis, kui soovitakse genereerida *Ecore* mudeli põhjal (objekt-)relatsioonilise andmebaasi lahendus, mida *Eclipse* modelleerimise tehnoloogia toetab.

Kliendiseoste metamudel *(Customer_relationships)* loodi vägagi sarnaselt Toodeteseoste projekti metamudeliga. Nagu teistegi sõltuvate projektide puhul, sai siin baasmudeli laiendamise teel loodud spetsiaalselt klientide kohta käivad elemendid. Olemas on baasmudeli põhiklassist (*Thing*) tuletatud osapoole (*Party*) klass, kliendi ja kliendigrupi ning samuti kliente puudutavate seoste klassid. Lisas 3 võib näha täismahus Kliendiseoste metamudelit (kahel joonisel). Lihtsustatud Kliendiseoste metamudeli eskiis joonisel 7 on järgmine:



Joonis 7. *Customer_relationships* metamudeli lihtsutatud kirjeldus.

Allikas: *Draw.io* [11].

Nagu ennist öeldud, sai tehtud ka pakkumiste-nõudmiste kirjeldamiseks ja lõputöö juhendamise valdkonna jaoks spetsiifilised projektid, millel järgnevates alampeatükkides lähemalt peatutakse.

4.3.1 *Offer-Request Relationships* metamudel

Võis ette arvata, et selle töö *Ecore* metamodelleerimise osa lõppsihiks olev lõputööde Juhendamise (*Supervising*) alamvaldkond ehk teemade, teemavaldkondade 'turuplats' allub ka üldisemale 'turuplatsi' valdkonna mudelile. Sellist valdkonda esindab selles töös metamudel nimega (*Offer Request Relationships*). Seega Pakkumisi-Nõudmisi kirjeldav metamudel *(Offer_Request_relationships)* on loodud *Eclipse Siriuses* eesmärgiga näidata ükskõik millise turuplatsi nõudluse ja pakkumise vahelisi seoseid, mitte ainult lõputööde teemade, teemavaldkondade, õppejõudude ja tudengite vahel ülikoolis. On teada, et metamudelit läheb vaja metaandmete andmiseks graafilise liidese prototüüpi. Ühtlasi kasutatakse osasid selle projekti klasse Juhendamise projekti (*Supervising*) metamudelis, seega on antud projekt vajalik koostisosa graafilise liidese hilisemaks loomiseks.

Pakkumiste-Nõudmiste metamudelisse on loodud klassid, mis spetsifitseerivad imporditud klasse. Imporditud klassid on omakorda pärit teistest varasemalt loodud metamudelitest (*Product Relationships, Customer Relationships*), mida eelnevas alampeatükis kajastati. Ühtedeks klassideks, mida kajastada, on pakkumine (*Offer)* ja nõudmine (*Request),* mis spetsialiseerivad ühist klassi (nimega *Transaction),* mis omakorda spetsialiseerib kliendiseoste klassi (*CustomerRelationship*) Kliendiseoste (*Customer Relationships*) metamudelist. Need klassid on olulised, sest need defineerivadki osaliselt pakkumise – nõudmise seoseid, mida täpsemalt spetsifitseeritakse veel sellest projektist sõltuvas Juhendamise projektis. Lisaks on oluline välja tuua ühe klassina turuosa (*Market),* mis võimaldab grupeerida turuosalisi ja spetsialiseerib omakorda kliendigrupi klassi (*CustomerGroup*) Kliendiseoste (*Customer Relationships*) metamudelist. Nö turumudeli (*MarketModel)* klass on loodud, sest see esindab vaadeldavat metamudelit tervikuna.

Lõputöö valdkonna puhul on vajalikud hilisema prototüübi elemendid nagu lõputöö teema ja teemavaldkond. Siinses mudelis kannavad need veel vastavalt abstraktsemat nime (*Inventory)* ja (*InventoryGroup),* mis spetsialiseerivad ja 'vahendavad' vastavalt toote (*Product*) ning tootegrupi (*ProductGroup*) klasse Toodeteseoste (*Product Relationships*) metamudelist. Kolmanda vajaliku suhtena prototüübi jaoks on loodud ka lõputöö teema kuuluvust teemavaldkonda abstraktsena kirjeldav seos *(InventoryGroupIncludesInventory)*. Viimaste elementidena antud metamudelis vajavad kajastamist osapooled, seega on organisatsiooni kirjeldamiseks (*BusinessMarketPlayer)*

klass ja eraisikut defineeriv klass (*PrivateMarketPlayer)*. Juhendatava ja juhendaja puhul on tegu eraisikutega ning nendest räägitakse juba täpsemas võtmes järgmises alampeatükis.

Graafilise liidese eduka toimimise jaoks on vaja metaandmetena taolisi eespool nimetatud elementide klasse. See projekt sisaldab natuke üldisemaid mõisteid Pakkumiste-Nõudmiste kohta, mille klasse hakatakse spetsifitseerima järgmises Juhendamise projekti metamudelis.

Detailne Pakkumiste-Nõudmiste teemaline metamudel (*Offer_Request_relationships)* on nähtav Lisa 4 all (kahel joonisel).

4.3.2 *Supervising* metamudel

Juhendamise teemaline *(Supervising)* metamudel on loodud kirjeldamaks otseselt ülikooli lõputöö juhendamise valdkonnaga alamvaldkonnaga ’teemade-teemavaldkondade turuplats juhendaja ja juhendatava kokkuviimiseks’ seotud elemente ja protsesse, mida kasutatakse graafilises liideses prototüübi disainimisel. Siia on sisse toodud klasse eelnevalt lahtiseletatud Pakkumiste-Nõudmiste metamudelilt. Antud metamudelis aga spetsifitseeritakse lõplikult klassid, seega siin antud nimetused kajastatakse elementide nimetustena ka graafilise liidese diagrammivaates ehk diagrammiaknas.

Juhendamise metamudelis võib esimese elemendina taaskord välja tuua ’turumudel’ klassi (*MarketModel)*, mida spetsialiseerib juhendamise mudeli klass (*Supervising_model)*. See klass on oluline, sest see leiab kasutust konfiguratsiooni graafilises liideses loodud *Siriuse* projekti metamudelit esindava elemendina. Ühtlasi leiab see element kajastust mõne elemendi domeenklassina sealses disainitavas diagrammivaates. Eelnevas alampeatükis kirjeldati ka nö turu (*Market)* klassi, mis siia mudelisse imporditi Pakkumiste-Nõudmiste metamudelilt. Selle paremaks lahtiseletamiseks on selle elemendi alla loodud täpsustavateks klassideks õppekava (*Curriculum)* ja instituut (*Institute)*. Ülikoolis on õppejõud seotud kindla instituudiga ja tudeng on konkreetse õppekava all, niimoodi ongi kaudselt seotud ülikoolis ka tudeng ja õppejõud omavahel. Seega on neid klasse vaja kirjeldamaks graafilises liideses vastavaid elemente. Abstraktne lõputöö teemavaldkonna (*InventoryGroup)* ja teema klass (*Inventory)* saavad täpsemalt defineeritumaks, luues klassid (*SubjectDomain)* ja

(*Subject)*. Need on konkreetselt selged nimetused lõputöö teemavaldkonnale ja lõputöö teemale.

Elemendid leiavad kasutust nii õppejõu kui ka tudengi poolt kasutatavas prototüübis. Pakkumiste-Nõudmiste mudelis räägiti pakkumise ja nõudmise seoste loomisest vastavalt (*Offer)* ja (*Request)* klasside näol, siis Juhendamise metamudelis on neid spetsialiseeritud. Pakkumise seosest saab klass, mis selgitab juhendamise võimalikkust kui pakkumisseost (*Supervising_capability*) õppejõult tudengitele suunatuna. Nõudmise seos on spetsialiseeritud täpsemaks seoseklassiks (*Supervising_request*). Seda saab kasutada tudeng hiljem graafilises diagrammis, näidates oma põhilist soovi saada endale lõputöö jaoks juhendaja. Seda seost võib vajadusel kasutada ka muidugi õppejõud, kes soovib endale juhendatavat tudengit, kuid kõik toimub sõltuvalt situatsioonist ja tingimustest. Kaks eelnimetatud uut seoseklassi on eelmisest mudelist imporditud neid siduva klassi (*Transaction)* alamklassid, kuid nimetatud klassile on loodud veel kolmaski alamklass, milleks on juhendamise seoseklass (*Supervising)*, mida on vaja samuti prototüübis kajastada. Kasutusele tuleb juhendamise seos, kui õppejõud ja tudeng on juhendamise soovis kokku leppinud ning seost saab kasutada ka lõputööde teemade ja teemavaldkondadega seonduvalt. Järgmise olulise elemendina on vaja välja tuua eelmises metamudelis toote sisaldumist tootegrupis spetsialiseeriva seosena klass lõputöö teema sisalduvusest kindlas lõputöö teemavaldkonnas (*SubjectDomainIncludesSubject)*. Nimetamata ei saa jätta ka vaadeldavat tüüpi ’turuplatsi turuosalisi’ esindavaid elemente tudeng (*Student)* ja juhendaja (*Supervisor)*. Need on lisatud eraisikut kirjeldava imporditud klassi (*PrivateMarketPlayer*) alla, millest eelmises alampeatükis ka juttu oli. Tudengi ja õppejõu elemente on prototüübis kindlasti vaja, sest nende klasside metaandmete baasil luuakse graafilises liideses tudengite ja õppejõudude vaatenurkade diagrammid ning määratakse ära täpsemate Tudengi- (juhendatava) ja Õppejõukesksete (juhendaja) diagrammide domeenklassid.

Sellised on vajaminevad elemendid Juhendamise metamudeli projektis, need ongi vahetult kasutatavad klassid graafilise liidese diagrammide komplektil ehk prototüübis, millest räägitakse põhjalikumalt järgmises peatükis.

Juhendamise metamudeli (*Supervising*) detailse kirjelduse (kahel joonisel) leiab Lisa 5 alt.

Osas 3.2 kirjeldatud iteratiivset arendusprotsessi järgides on võimalik vajaduse korral ka siinkirjeldatud *Ecore* metamudeli lahendus (mis spetsialiseerib üldisemat baaslahendust) asendada konkreetsema ja samas vähempaindliku lahendusega (mis ei sõltu enam teistest metamudelitest).

4.4 *Eclipse Siriuse* graafiline liides

Järgnevas peatükis kirjeldatakse lähemalt autori poolt loodud graafilise liidese prototüüpi ning tuuakse välja stsenaariumite kirjeldused, kuidas tulevased lõppkasutajad prototüüpi kasutada saavad.

4.4.1 Graafilise liidese prototüüp

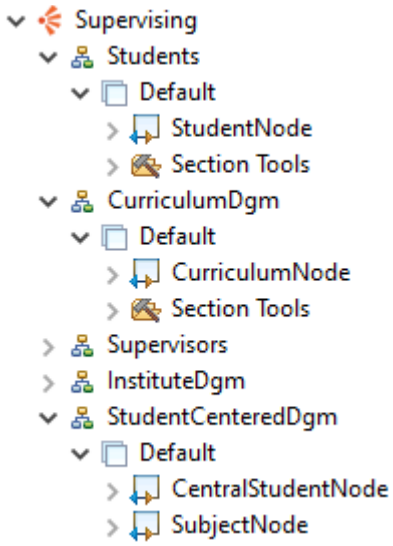
Nagu öeldud sai, siis antud töös graafilise liidese prototüüp kasutab metaandmeid kõikidest eespoolkirjeldatud projektide metamudelitest, milleks on Asjadeseoste (*Things_relationships)*, Toodeteseoste (*Product_relationships)*, Kliendiseoste (*Customer_relationships)* ja Pakkumiste-Nõudmiste seoste (*Offer_Request_relationships)* ning Juhendamise (*Supervising)* mudelid. Järgnev prototüüp viitab vahetult ainult viimatikirjeldatud Juhendamise metamudelile, mille elemendid omakorda sõltuvad teiste metamudelite elementidest. Siin alampeatükis on täpne kirjeldus antud töös valminud valdkonnaspetsiifilise modelleerimiskeele prototüübi kohta.

Lõputööde juhendamise valdkonna objektide konkreetseid andmeid (ehk mudelit), mis alluvad metaandmetele (ehk metamudelile), hoiab *Siriuse* modelleerimise projekt (*Sirius Modeling Project)*. Selles projektis on keskseks elemendiks Juhendamise metamudelit esindav klass (*Supervising model)*. Selle elemendi ehk mudeli alla saab hiljem luua graafiliste diagrammide eksemplare ning selles puustruktuuris tekivad siis nendel diagrammidel loodud graafiliste sümbolitena näidatavad andmed (*runtime* mudeli elemendid).

Teisena on loodud Vaate spetsifikatsiooniprojekt (*Viewpoint Specification Project)*, mis sisaldab Vaate mudelit (*Viewpoint Specification Model*), mis omakorda sisaldab Vaateid (*Viewpoint*). Vaade sisaldab esitluste (*Representation*) – näiteks diagrammide – kirjeldusi. Vaated väljendavad VSMK konkreetset süntaksi ja kasutavad keele abstraktset süntaksi väljendavaid metamudeleid Vaate elementide kirjeldamise alusena (Vaate

põhielemendid viitavad konkreetsetele klassidele metamudelis. Vaate spetsifikatsiooni projekti kataloogis sisalduv Vaate mudelit salvestav fail (*.odesign* laiendiga) hoiab diagrammiredaktoris loodud andmeid, mida *Siriuse* modelleerimisprojektis metamudeli instantsi kontekstis tõlgendatakse. Tänu sellele Vaate projektile sai siis diagrammide instantsid ehk esitlused elemente-seoseid õigesti näitama ja käituma panna. Vaate mudeli struktuuris (joonisel 8) asub oranži ikoonina Vaade (*Viewpoint: Supervising*), selle alla on loodud diagrammid: Õppekava (*CurriculumDgm)*, Instituudi (*InstituteDgm)*, Tudengid ehk juhendatavad (*Student*s), Juhendajad ehk õppejõud (*Supervisor*s). Samuti loodud ka juba täpsemad Tudengikeskne diagramm (*StudentCenteredDgm)* ja Juhendajakeskne diagramm (*SupervisorCenteredDgm)*.

Esimesena võib välja tuua Õppekavade ja Instituutide diagrammid, millel kummalgi on loodud üks element, vastavalt siis õppekava element (*CurriculumNode*) ja instituudi element (*InstituteNode*). Need elemendid käituvad sarnasel viisil, ent lihtsalt üks pärib andmeid õppekava klassi alt ja teine instituudi klassi alt metamudelist. Tudengite diagramm (*Students*) võimaldab lisada tudengite andmeid, mille visuaalset esitlust kontrollib diagrammielement (*StudentNode*) ning Juhendajate diagrammi (*Supervisors*) puhul vastavalt siis Juhendaja diagrammi element (*SupervisorNode*). Need lihtsad struktuuriga diagrammid on selles töös vajalikud harjumuspäraste ekraanivormide asemele *Siriuses* vastavalt tudengite ja juhendajate andmete sisestamiseks. Järgneval joonisel 8 on näidatud, milline näeb välja diagrammide struktuur elementidega diagrammiredaktoris [18].



Joonis 8. *Viewpoint*: *Supervising* alla kuuluvate diagrammide struktuur elementidega.

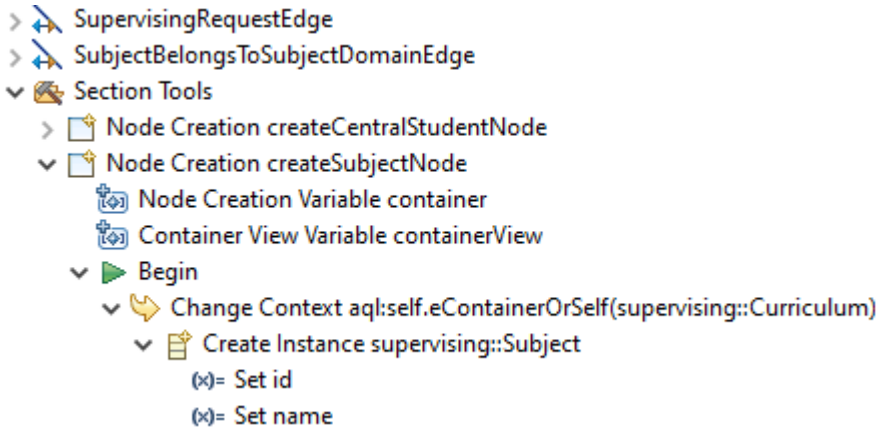
Allikas: *Eclipse Sirius – Obeo Designer Community.*

Lisaks nendele diagrammidele on loodud lõputöö juhendamise Tudengikeskne diagramm ja Juhendajakeskne diagramm. Neid diagramme võib lugeda juba kasutatavateks prototüüpideks, mida saavad hakata kasutama nii tulevased juhendatavad tudengid kui ka juhendajad. Need kaks diagrammi on üsnagi sarnaste elementidega ja ülesehitusega, seega võib kirjeldada töömahu kokkuhoiu mõttes järgmistes lõikudes Tudengikeskset diagrammi.

Selle diagrammi eesmärk on toetada lõputöö juhendamise alustamise ehk juhendatava ja juhendaja kokkuviimise protsessi; täpsemalt anda selle asjade seisust (teemade turuplatsist) graafiline ülevaade ning ühtlasi modelleerimisvõimalused situatsiooni ümberkujundamiseks (selles diagrammis põhiliselt tudengi poolt vaadatuna). Selles diagrammis sisalduvad elemendid ja seosed: lõputöö teema element (*SubjectNode)*, lõputöö teemavaldkond (*SubjectDomainNode)*, diagrammi (eksemplari) omanikuks oleva ehk keskse tudengi element (*CentralStudentNode)*, juhendaja (*SupervisorNode)*, keskse tudengiga läbi teema või teemavaldkonna seotud tudengi element (*RelatedStudentNode)*, seotud juhendaja element (*RelatedSupervisorNode)*, juhendamise nõudmise (soovimise) seos (*SupervisingRequestEdge)*, teema teemavaldkonda kuulumise seos (*SubjectBelongsToSubjectDomainEdge)*. Näiteks juhendamise nõudmisseos kirjeldab Tudengikeskses diagrammis juhendamise soovi seost (tudeng soovib juhendamist), mis võib ühendada tudengit juhendajaga (tudeng soovib konkreetset juhendajat), kuid seosenoole võib siduda ka lõputöö teema või teemavaldkonna elemendiga (tudeng soovib juhendamist konkreetsel teemal või teemavaldkonnas). Lisaks on tarvis veel juhendamise seost (*SupervisingEdge)*, kinnitamaks mingil teemal või teemavaldkonnas (tudengi) juhendamist. Selle seose loob juhendaja Juhendajakeskse diagrammi eksemplari omanikuna ja tema poolt loodud juhendamisseost näidatakse ka juhendatava tudengi keskses diagrammis. Keskseim element Tudengikesksel diagrammil on aga diagrammieksemplari omanikuks oleva ’keskse tudengi’ element, mis tekib diagrammi eksemplari ehk representatsiooni loomisel automaatselt pildile, sest tema element on varasemalt loodud Tudengite (andmete haldamise) diagrammi kaudu. Element kirjeldab tulevase lõppkasutajana tudengit, kes hakkab prototüüpi kasutades andmeid sisestama ja valima [18].

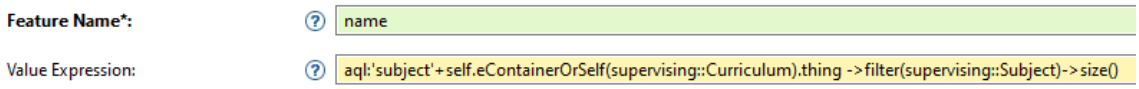
Elementide loomise ja redigeerimise Tööriistad on (*Section Tools)* kausta all. Näiteks on lõputöö teema loomiseks vaja tööriista (*Node Creation createSubjectNode)* ja juhendamise soovi seose loomiseks tööriista (*Edge Creation createSupervisingRequestEdge)*. Loomise (C*reate*) tööriistaga saab tudeng või õppejõud sisestada diagrammiaknas uusi elemente. Näiteks, tööriist keskse tudengi elemendi loomiseks (*createCentralStudentNode)* ei leia pärast diagrammiomanikuks oleva tudengi elemendi automaatse loomise realiseerimist enam kasutust, kuid sarnaste tööriistadega saab tudeng sisestada mudelile juhendajaid, lõputöö teemasid, juhendamise soovi seoseid ja ka seost teema kuuluvusest teemavaldkonda. Valikust lisamise *(Select)* tööriistaga saab valida diagrammil diagrammil näidatavaid elemente olemasolevate *Sirius* mudeli elementide hulgast, mis on loodud muudes diagrammides, kas teiste tudengite või õppejõudude poolt. Tudeng saab nende tööriistadega valida/lisada oma diagrammile juhendajaid (kes seotud teda huvitava teemavaldkonnaga), lõputöö teemavaldkondi, valitud teemavaldkondadega seotud teisi tudengeid, (teiste) tudengite juhendamissoovi seoseid ning ka teiste juhendajate juhendamise võimalikkuse (*SupervisingCapabilityEdge*) seoseid [18].

Avaldiste (*expression* tüüpi) väljadele õigete väärtuste (elemendinimede, konstantide) ja käskude (funktsioonide, operaatorite) sisestamise kontekstitundlik menüü aitab leida elementide hulgast (metamudelitest) vajaminevaid täpsustatud ja filtreerituid elemente. Näiteks on lõputöö teema tööriistale määratud esimeseks operatsiooniks kontekstimuutmise operatsioon (*Change Context*). Lõputöö teema kuulub õppekava (võiks kaaluda panemist ka instituudi) alla ning seetõttu ongi õppekava (*Curriculum)* konteksti määramise avaldisse pandud. Konteksti alla on loodud operatsioon objekti loomiseks (*Create Instance)*. Siin on täpsustatud, et luuakse lõputöö teema objekt. Nende all on veel määratud operatsioonidega (*Set)* teema objektile identifikaator ja nimi, mis on samuti päringukeeles kirjutatud ja metaandmetele viitavat avaldist kasutades. Järgnevalt on nimetatud tööriista struktuur kujutatud joonisel 9 ja selle ühe operatsiooni (*Set name)* avaldise (*Value Expression*) kirjeldus joonisel 10 [18].



Joonis 9. *Section Tools* alla kuuluva *createSubjectNode* tööriista struktuur.

Allikas: *Eclipse Sirius – Obeo Designer Community.*



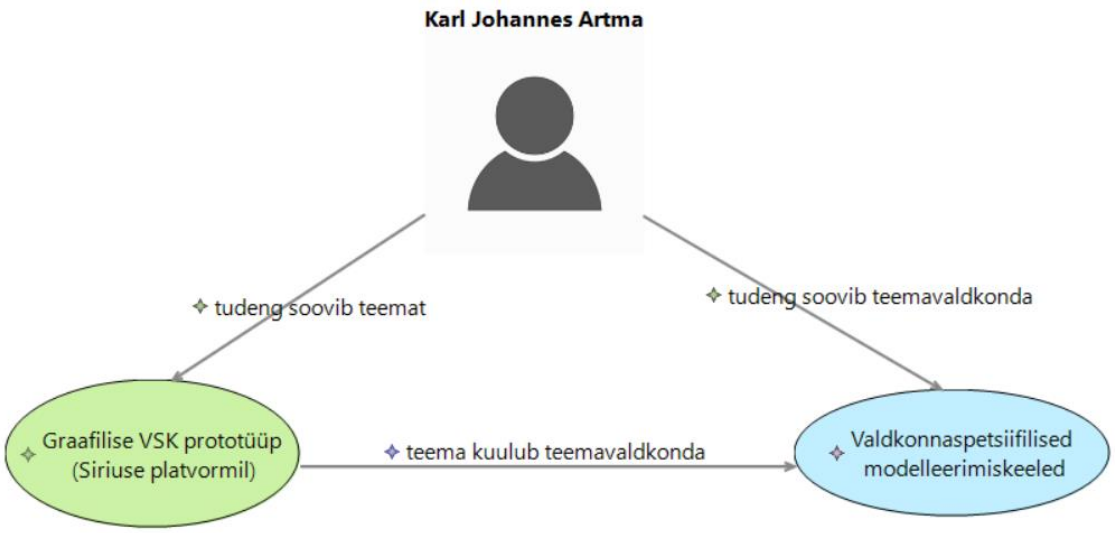
Joonis 10. *createSubjectNode* tööriista *Set name* operatsiooni kirjeldus.

Allikas: *Eclipse Sirius – Obeo Designer Community.*

Vasakus aknas on näha *Sirius* modelleerimise projektis juhendamise mudeli (*Supervising Model)* alt loodud diagrammid ehk mudelielementide graafilised esitused. Esmalt on loodud Õppekavade ja Instituutide graafilised diagrammid ehk representatsioonid. Need on loodud esimesena, sest tavaliselt on ülikoolis õppekavade all juba varasemast olemas tudengid ja instituutide all õppejõud. Tudengite ja Juhendajate diagrammidel on diagrammide disainis määratud domeenklassiks vastavalt õppekava element ja instituudi element, mistõttu pidi konkreetsete osapoolte kesksed diagrammid looma siis, kui neid osapooli esindavad elemendid *Sirius* modelleerimise projektid on juba loodud. Sarnase loogikaga loodi konkreetsete tudengite jaoks Tudengikeskne diagramm (*StudentCenteredDgm*) ning konkreetsete juhendajate jaoks Juhendajakeskne diagramm (*SupervisorCenteredDgm*).

Graafiliste diagrammide tüüpide loomise järel saab need isikukesksete diagrammide eksemplarid avada diagrammiakendes, mis väljendavadki sisuliselt prototüüpe, mida tulevased lõppkasutajad hakkavad kasutama. Diagrammile on võimalik elementide, seoste tööriistu lohistades luua endale sobiv lõputöö juhendamise protsess.

Järgmisel joonisel on toodud Tudengikeskse diagrammi näide prototüübist (mudeliredaktori tüüpi kasutajaliidesest). Nagu enne mainitud, siis diagrammil on keskne diagrammi eksemplari omava tudengi element, mis on siin ikoonina kujutatud. Tudengile on pandud selles diagramminäites nimeks selle töö autori nimi. Tudeng saab nüüd ise kujundada sobiva juhendamisskeemi tööriistade abil, mis eelnevalt diagrammiredaktoris loodud said. Järgnevalt on illustreeritud diagrammiakna prototüüp joonisel 11 [18]. Tudengikeskse diagrammi prototüübiga on kooskõlas ka loodud Juhendajakeskse (õppejõu) diagrammi prototüübi (mudeliredaktori tüüpi kasutajaliidese) näide, mis on nähtav Lisa 6 all.



Joonis 11. Tudengikeskse diagrammi prototüübi kirjeldus.

Allikas: *Eclipse Sirius – Obeo Designer Community.*

Allikas (ikoon): *eit RawMaterials Connecting Matters* [19].

Sellistest põhjalikult lahti seletatud osadest koosneb autori poolt loodud graafilise liidese prototüüp.

Järgnevates alampeatükkides on selgitatud, kuidas lõppkasutajad (tudeng ja õppejõud) saavad prototüüpi kasutada, kirjeldades seda läbi näitestsenaariumite.

4.4.2 Tudengi stsenaarium prototüübi kasutamisel

Eeltingimuseks on, et juhendajate diagrammide kaudu on loodud (mudelisse sisestatud) soovitud juhendaja ehk õppejõud. Lisaks võib olla sisestatud (nt Õppejõukeskse

diagrammi kaudu) teemavaldkond, teisi juhendajaid, juhendamise võimalikkuse seos ja ka mõne teemavaldkonnaga seotud tudengid.

Tudeng lisab Tudengikesksel (teda ennast esindaval mudelielemendil avaneval) diagrammil lisamistööriistaga soovitud juhendaja või loob tööriistaga uue juhendaja. Seejärel saab ta sisestada soovimisseose endast juhendajani. Tudeng lisab diagrammile selle juhendajaga seotud teemavaldkonna, mida juhendaja välja on pakkunud. Eelnimetatud seost saab nüüd luua ka teemavaldkonna ja tudengi vahel. Ühtlasi, kuna nii teemavaldkond kui ka juhendaja on lisatud õppejõu diagrammi kaudu, siis kandub üle ka nendevaheline juhendamise võimalikkuse seos (*SupervisingCapabilityEdge)*, kui see oli seal loodud. Oletame, et tudeng on ise tulnud lagedale uue teema mõttega, siis võib ta selle diagrammile luua uue teema elemendina tööriista (*createSubjectNode*) abil. Vastasel juhul juhendaja mõtles teema välja ja siis lisab tudeng õige (*Selection Wizard*) vahendiga lõputöö teema oma diagrammile olemasolevate mudelielementide hulgast. Tudeng loob soovimisseose lõputöö teemaga. Kui on teada, et teema kuulub kindlasse antud diagrammil esindatud teemavaldkonda, aga vastav seos on mudelielemendina veel loomata, siis saab tudeng diagrammil luua sisalduvusseose (*createSubjectBelongsToSubjectDomain)*, alustades noole tõmbamist lõputöö teemast. Juhul, kui tudeng soovib, siis võib ta diagrammile valida ka muid elemente, nagu mingi õppejõu või teemavaldkonnaga seotud tudengeid/õppejõude (*RelatedStudentNode*, *RelatedSupervisorNode)*. Näiteks mingi teine tudeng või juhendaja on seotud teemavaldkonnaga 1, siis tekivad olemasoleva teemavaldkonna ja seotud tudengi/õppejõu lisamisel nende kahe elemendi vahele automaatselt seosed, sest teemavaldkond 1 on teise tudengiga mujal diagrammis juba seotud. Sellises vaates on tudengil juba päris selge ülevaade endapoolsest lõputöö protsessist ja informatsiooni ka teiste tudengite/juhendajate seotusest oma protsessiga.

4.4.3 Juhendaja stsenaarium prototüübi kasutamisel

Eeldab, et on varasemalt loodud (mudelisse sisestatud, näiteks *Students* diagrammi kaudu) tudeng, keda soovitakse juhendada. Teiste juhendajate lisamine pildile eeldab ka nende olemasolu mudelis ehk juhendajate *(Supervisors)* diagrammil. Samuti, et oleks mudelis olemas ka teisi seotuid juhendajaid ja tudengeid. Loomulikult on võimalik vajadusel diagrammi tööriistad disainida ka nii, et kõik eelnimetatud elemendid saaks luua antud diagrammi kaudu.

Nagu Tudengikesksel diagrammil, on ka Juhendajakesksel ehk konkreetset õppejõudu esindavalt mudelielemendilt avaneval diagrammil juhendaja ikoon automaatselt diagrammil olemas. Juhendaja võib alustada juhendamise protsessi kirjeldamist näiteks teemavaldkonna loomisega diagrammil, kasutades vastavat tööriista (*createSubjectDomain*). Ühtlasi saab ta panna all asuvas omadusteaknas teemavaldkonnale sobiva nime. Järgmisena soovib ta näidata, et saab juhendada seda teemavaldkonda. Ta kasutab selleks juhendamise võimalikkuse seost, mille ühendab enda ikoonist teemavaldkonnani. Sellest edasi võib juhendaja soovida pildile lisada tudengit, kasutades selleks sobivat tööriista. Kui tudeng on diagrammile lisatud, saab ta luua ka temaga seose. Näiteks samuti juhendamise võimalikkuse seose nende vahele. Õppejõud võib teada, et selle teemavaldkonnaga on seotud veel tudengeid, sel juhul tekitab (valib või loob) ta vastava tööriistaga (*selectStudentNode* või *createStudentNode*) nad skeemile. Sarnane tööriist on ka seotud juhendajate pildile loomiseks. Kui õppejõud kindlalt teab, et juhendab teatud tudengit, saab ta kasutada juba kindlat juhendamist defineerivat seost (*createSupervisingEdge)*, näitamaks oma juhendamisseost tudengiga. Nagu ka tudengi diagrammil, saab juhendaja lisada teisi teemavaldkondi pildile (*selectSubjectDomainNode*), mis eeldab asjaolu, et keegi õppejõududest on selle oma diagrammi kaudu loonud. Lisaks sellele võib juhendaja tuua mudelile teiste õppejõududega seotud tudengeid või teiste tudengite/teemavaldkondadega seotud juhendajaid. On võimalik kasutada erinevaid mehhanisme andmete automaatseks ülekandumiseks, näiteks kui juhendaja lisab diagrammile (*Select*) teise juhendaja, siis tuleb teise juhendajaga kaasa seotud teemavaldkond ja sõltuv seos.

Sellist stsenaariumit rakendades on juhendajal väga hea ülevaade enda juhendatavatest tudengitest, nendega seotud teemadest, teemavaldkondadest ning lisaks saab juhendaja näha endaga seotud teisi tudengeid, juhendajaid ja teemavaldkondi.

On loodud ja lahti seletatud mudelid, mis kirjeldavad nii õppejõu kui ka tudengipoolsest vaatepunktist ülikooli lõputööde juhendamise esimese etapi (juhendatava ja juhendaja kokkuviimine) stsenaariumeid ja protsesse.

4.4.4 Töö põhitulemus

Vahekokkuvõtteks võib öelda, et saavutati töö sissejuhatuse osas kavandatud põhitulemus ehk loodi lõputööde juhendamise valdkonna ühe alamvaldkonna (teemade ja teemavaldkondade ’turuplats’ juhendatava ja juhendaja kokkuviimiseks) mudelid,

vastava valdkonnaspetsiifilise graafilise keele abstraktse ja konkreetse süntaksi ning tööriistade spetsifikatsioonid. Realiseeriti eelnevates osades kirjeldatud tarkvaraprototüüp *Eclipse Siriuse* platvormil.

5 Analüüs – Järeldused

Analüüsi peatükis tuuakse välja järeldused eelmistes osades esitatud töötulemuste, nendeni jõudmise protsessi ning edasise töö suundade kohta.

Töö põhitulemuseks olev prototüüp põhimõtteliselt võimaldab juhendataval tudengil ning juhendaval õppejõul teemade/valdkondade ’turuplatsil’ kokku saada ning koostöös kavandada (graafiliselt modelleerida) tulevase juhendamise protsessi põhistruktuuri (ehk karkassi) – põhiosapooled (juhendaja ja juhendatav), neid huvitavad teemad ja teemavaldkonnad, seotud osapooled (samade või sarnaste teemade ja teemavaldkondadega seotud teised õppejõud ja tudengid), praegused (miks mitte ka eelnevad) juhendamise soovid ja tegelikud juhendamised, jne.

Selline konkreetse juhendamise struktuur on dünaamiline (ajas kiiresti muutuv) ning vajab jätkuvalt ümbermodelleerimist juhendamise protsessi käigus. Seepärast on mudeliredaktori tüüpi kasutajaliidesed senise testkasutamise käigus osutunud põhimõtteliselt sobivaks lahenduseks antud konkreetse (juhendamise) probleemvaldkonna jaoks. Ka nende kasutajaliideste ’all olev’ suhteliselt abstraktsetest analüüsimustritest spetsialiseeritud metamudel on seni hästi sobinud, sest arenduse praeguses/varases faasis toimuvad prototüübi testkasutamine, probleemvaldkonna (uuesti)analüüsimine ja selle tulemustele vastav metaandmete korrigeerimine sageli ’paralleelselt’ (s.t. ühe kiire arendustsükli/iteratsiooni jooksul).

Iteratiivses arendusprotsessis modelleeriti *Ecore* mudelite näol ja analüüsimustreid kasutades üldine Asjadeseoste ja struktuuride valdkond. Seda valdkonda spetsialiseeriti sammhaaval ülalt-alla strateegiaga (ehk üldisemast konkreetsemaks) liikudes konkreetsemateks ja kitsamateks valdkondadeks (tootestruktuurid, kliendistruktuurid jne). Lõpuks jõuti antud töö ’lõppsihini’ ehk Juhendamise (*Supervising*) struktuuride valdkonna ja vastava keele lahenduseni. Põhimõtteliselt oleks saanud ka kohe luua

juhendamise valdkonna struktuuri, kui oleks kasutatud alt-üles lähenemist (konkreetsemast üldisemale valdkonnale liikumist) või ainult lõputööde juhendamise spetsiifilise valdkonna juurde jäädudki. Siis oleks jõutud kasutatava tarkvaraprototüübini kindlasti kiiremini, ent samas oleksid selle edasiarendusvõimalused olnud piiratumad ja lahenduse korduvkasutamine teistes sarnastes valdkondades vähemtõenäolisem. Praegust töös loodud lahendust saab nimetada arhitektuurseks ja ka pidevalt edasiarendatavaks prototüübiks, mida juhendamiste kavandamise koostööprotsessis saavad nii lõppkasutajad (tudengid, õppejõud) kui ka keele spetsifitseerijad (arendajad) omavahelises koostöös suhteliselt lihtsalt muuta (ehk transformeerida). Paraku jäi selle prototüübi testkasutamiseks vähe aega, mistõttu on seda seni testinud/kasutanud ainult lõputöö autor ja tema juhendaja. Mõlemad senised testkasutajad juhtumisi sobivad täitma nii loodava keele lõppkasutaja (probleemvaldkonna spetsialisti) kui ka spetsifitseerija (lahendusvaldkonna spetsialisti) rolle/ülesandeid. See ei ole aga tavapärane olukord. Tavaliselt on (*Eclipse Siriuse* platvormi kasutamisel) keele spetsifitseerijaks tarkvaraarenduse spetsialist, kes teeb tihedat koostööd probleemvaldkonna spetsialistist (lõpp)kasutajaga, kes ei oma tarkvaraarendaja kompetentse. Kuna aga lihtsamaid keeli on *Siriuse* platvormil võimalik spetsifitseerida ilma otseselt ise koodi kirjutamata (*MDA* lähenemist kasutades), siis põhimõtteliselt võiks valdkonnaspetsialist osaliselt täita ka spetsifitseerija rolli. *Eclipse Siriust* kasutades saaks ta sellist lõppkasutajast arendaja (*End-User Developer*) rolli ehk *MDA* vaates lõppkasutajast modelleerija rolli täita pigem kasutusaja, mitte disainiaja etapis (vt täpsemalt Metoodika peatükist 3.2).

Kindlasti aitab valdkonnaspetsiifilistel keeltel põhinev (VSMK) lähenemine kaasa töös põhiprobleemina kajastatud kommunikatsioonilõhe lahendamisele. Tehtud prototüüp on lõputöö juhendamise valdkonnas kasutoov ja rakendust leidev.

Lisaks nendele järeldustele võib veel välja tuua võrdluskoha graafilise mudeliredaktori (põhineb graafi mudelil) ja tavaliste ekraanivormide (kus on nt *combo-box'id* jms.) kasutamise vahel antud töö alamvaldkonnas. Graafilise modelleerimise kasutamine on parem lahendus konkreetse isiku (tudengi või õppejõu) juhendamisseoste kavandamise kasutajaliideste jaoks, sest vajalikke kujunduselemente ja seoseid saab *Eclipse Siriuses* luua ja disainida kasutamise ajal ja ilma programmeerimata, taaskasutades olemasolevaid standardseid või oma loodud kujundeid või pilte. Elementide ja seoste käitumist on valemilahtris kerge luua ja muuta, kuna pakutakse valemikomponente (sh metamudelite elementide nimetusi), millega luua kindla elemendi tööriista jaoks töötava valemi.

Tavalisi ekraanivorme kasutades (kus on nt *combo-box'id* jne) tegeldaks sisuliselt samade andmetega, kuid puuduks juhendamisseoste visualiseeritud struktuur. Ka ei pruugi kasutajaliidese kujunduse ja käitumise muutmine kasutusaja (*runtime*) keskkonnas enam võimalik olla. Samuti võib elementide, seoste ning mõlema kujunduse ja ka käitumise loomine olla põhjalikuma protsessiga ning üldjuhul on vaja programmeerimisoskust, mis nõuab eelteadmisi ja on kokkuvõttes aeganõudvam. Samas oleks mitmete antud töö kontekstis mittepõhiliste kasutajaliideste – näiteks tudengite ja õppejõudude esmaseks registreerimiseks – lahendamine tavaliste ekraanivormidega mõistlik ja ilmselt parem. Antud töö jätkuna oleks lihtne kombineerida näiteks *Eclipse Siriust* ja *EMF Forms-i* erinevat tüüpi kasutajaliideste rakendamiseks samadele andmetele ehk *Sirius* modelleerimise projektile, kuna mõlemad tööriistad kuuluvad *Eclipse* modelleerimise tehnoloogiasse ning toetuvad *EMF*-ile.

5.1 Alternatiivsed keskkonnad

Eclipse Siriuse kõrval on ka teisi valdkonnaspetsiifilisi graafilise keele loomise tööriistu. Nendest ühena võib kirjeldada *Eclipse* ettevõtte poolt pakutava *Graphiti* programmi, mis sisuliselt kasutab ka *Eclipse Modeling Frameworki* (*EMF*i) infrastruktuuri graafiliseks modelleerimiseks. *Graphiti* saab hakkama seega *EMF-l* põhinevate valdkonnamudelitega ja *Java* objektidega domeenides samuti. *Graphiti* raamistik on avatud ka teistele platvormidele ja integreeritav, ent praegu siiski on vaid *Eclipse* keskne. Eesmärkide saavutamiseks *Graphiti* tööriistaga pakub see graafilise liidese loomiseks selget ja hästi ülesehitatud *Java API*-t ning tagab ka piisava dokumentatsiooni mudelite ehitamise abistamiseks. Võrreldes *Eclipse Siriuse*ga, kus on metamudeliteks *Ecore* mudelid abstraktse süntaksi kompositsiooniks, siis *Graphitil* on kasutuses abstraktse süntaksi jaoks kas *Ecore* või *Java* metamudelid. Mudelite representatsioonide kuvamise jaoks on ka erinev konkreetne süntaks e graafiline liides, kuna *Eclipse Sirius* kasutab *Odesign* vaadet, ent *Graphiti Draw2D* oma.

Sarnasuseks nende kahe programmi puhul võib tuua, et mõlemad on vabavaralised tööriistad ning nad on ka ühtedest kõige funktsionaalsematest ja täiuslikumatest VSK keele redaktorite loomisel [12], [20].

Eclipse Siriuse puhul on erinevalt *Graphitist* vähemalt natuke metodoloogilist tausta paremini kajastatud. Tausta eri parameetrite alustele tuginedes on metamudelid, süntaksid ja graafilised objektid arendajale *Siriuse*s veidi selgemini arusaadavamad [12].

Põgusalt võib mainida ka üht käesoleva töö tulemuste edasiarendamist puudutavat teemat, milleks on mudeli koostamise mugavdamine kasutajale. Nimelt pole *Siriuse* platvormi poolt pakutavad standardväärtused mudelite koostamisel praegu veel eriti paindlikud kasutaja suhtes, s.t platvorm ei suuda praeguses staadiumis kohanduda kasutaja varasema käitumisega, mis tegelikult oleks kasutajale heaks abimeheks ja milleni tulevikus võiks jõuda [21].

Lisaks *Graphityle* on *Eclipse Siriuse* alternatiiviks veel *GML (Graphical Modeling Language)*. *Siriuses* kasutatakse *Low-code/no-code* lähenemist, seega ei ole seal üldjuhul, erinevalt *Graphity*-st ja graafilisest modelleerimise raamistikust (*GMF*-st), vajadust programmeerida lihtsamate mudeliredaktorite loomiseks. *Siriuses* on siiski tehniliselt võimalik programmeerida, kasutades *Javas* kirjutatud teenuseid. Sisuliselt tähendavad *Low-code/no-code* lähenemised, et rakenduse arendustööd tehakse visuaalselt/graafiliselt, luues mudelipõhise arendusega mingi graafiline liides [3]. Selle lähenemise puhul on ka arendustöö kiirused erinevad.

5.2 Lõppkasutajad

Kuna prototüübi testkasutamiseks jäi vähe aega, on seda seni testinud/kasutanud ainult lõputöö autor ja tema juhendaja. Järgnevalt on tarvis tagasiside saamise ja seda arvesse võttes prototüübi parendamise eesmärgil (test)kasutajate ringi laiendada.

Võimalikud edasised kasutajad loodud prototüübile oleksid ülikooli liikmed (õppejõud ja tudengid). Põhilisteks kasutajateks oleks ülikoolis juhendajad ehk õppejõud, kes soovivad juhendada või kellele esitatakse juhendamise soov tudengite poolt. Teine võimalik sihtgrupp on tudengid, kes soovivad oma juhendamise protsessi visuaalselt modelleerida, saades samal ajal ka huvitava valdkonnaspetsiifilise modelleerimiskeele tööriista kasutamise kaudu kogemust juurde. Juhendajad samamoodi võivad saada kogemust ja inspiratsiooni, jagades oma teadmisi teiste kolleegidega või õpetada tudengeid prototüüpi katsetama.

5.3 Töö piirangud

Üheks piiranguks on kindlasti see, et autoril puudub varasem *Eclipse Siriuse* kasutamise kogemus. Seeläbi oli programmi kasutamine teatud piiridesse jäädes uudne kogemus ja toimus jooksvalt tarkvara mõistmine ja teadmiste omandamine, mis võttis oma aja. Infomaterjale oli piisavalt *Siriuse* enda kohta, ent samas siiski piiratud hulgal, seetõttu oli juhendaja teadmistest ja soovitustest palju mõtteainet ja kasu töö tegemisel. Samas *Eclipse Siriuse* kasuks otsustati, sest varasemalt on olnud programmeerimise aines *Eclipse*-i tekstikeele redaktoriga kokkupuude *Java* arendamisel.

Tehnilise poole pealt võib tekkida vajadus arendajapoolse kasutajatoe järele. Seda vajataks näiteks tarkvara installimiseks või alustavale tööriista lõppkasutajale nõu andmiseks võib olla tarvis arendaja abi. Prototüüp on hetkel piiritletud ka ühes valdkonnas, juhendamise alal, rakendamiseks. See piiritletus võib tähendada aga taaskord arendaja kaasamist prototüübi mingi teise valdkonnaspetsiifika täiendamiseks või muutmiseks.

5.4 Edasised arengu suunad

Selle töö edasiarenduse üheks suunaks oleks kaasata tulevane lõppkasutaja ja temalt saadud tagasiside alusel näha parendusvõimalusi, et temale prototüübi kasutamist lihtsustada ja selgemaks teha. Saab kaasata erinevaid hilisemaid lõppkasutajaid (õppejõud ja tudengid) ja vaadata, millised on nende ühtsed ja eraldiseisvad soovitused tööriista parandamiseks. Sel juhul on võimalik koostöös tudengite ja õppejõududega või siis tarkvaraarendaja enda poolt muudatused sisse viia. Arendaja võib lisaks anda mõtteterasid töö efektiivsemaks muutmisel. Üldiselt võib arvata, et koostöö paraneb õppejõudude ja tudengite vahel, sest mõlemad saavad luua oma skeemi ja teineteisega tulemusi jagada. Kindlasti on see prototüüp või tema edasiarendus mõeldav kasutusele võtta, sest väheneb kommunikatsiooni vajadus ja nõudmised, soovid on selgemini arusaadavamad.

Teiseks arengu suunaks peaks kindlasti olema prototüübi tehnilise poole edasiarendamine. Saab paigutada elemente konteineritesse nende paremaks grupeerimiseks, et näidata näiteks juhendaja poolt juhendatavate tudengite kuuluvust konkreetsete õppekavade alla. Filtrite kasutuselevõtuga saab määrata, millised elemendid

ja seosed kindlates situatsioonides võiksid olla diagrammil peidetud. Võib kasutusele võtta sellise tööriista graafilisel liidesel, mis võimaldab elemendil topeltklikkides minna konkreetse elemendiga seotud diagrammi juurde. Nagu näha on võimalusi arenemiseks prototüübiga veel palju, sest *Eclipse Sirius* on väga laiahaardeline graafilise keele platvorm [22].

6 Kokkuvõte

Lõputöö teemavaldkonnaks olid valdkonnaspetsiifilised modelleerimiskeeled. Selliste keelte loomise eesmärgiks on vähendada arendustöö praktikas esinevat kommunikatsioonilõhet tarkvara tellija ja arendaja, teisisõnu äri- ja IT osapoolte vahel. Põhiprobleem seisneb siin selles, et mõlemad osapooled suhtlevad oma terminites ega saa üksteisest piisavalt hästi aru. Seetõttu lahendatakse sama probleemi kaks korda – enne probleemvaldkonna ja seejärel lahendusvaldkonna (nt programmeerimiskeele) terminites. Olukorra muutmise üheks lahenduseks on valdkonnaspetsiifilistel keeltel põhinev lähenemine.

Antud töö eesmärgiks oli välja töötada valdkonnaspetsiifilise graafilise (modelleerimis)keele ja sellel põhineva tööriista prototüüp ülikooli lõputöö juhendamise valdkonna ühe alamvaldkonna (ühe etapi) jaoks *Eclipse Siriuse* platvormil. Eesmärgi täitmiseks on kasutatud ja kombineeritud *Fowler'i* sobivaid analüüsimustreid (*Accountability* jt.), *Eclipse* modelleerimisraamistiku (*EMF*) *Ecore* metamudelite modelleerimist ja koodi genereerimist. Sellele järgnes *Siriuse runtime* modelleerimine ja Vaadete (*Viewpoint*) ehk täpsemalt diagrammitüüpide modelleerimine/tõlgendamine. Samuti oli vaja järgida *Siriuse* platvormi iteratiivset arendusprotsessi, mis eeldab tarkvaraarendaja (kui keele spetsifitseerija) ning (lõpp)kasutaja tihedat koostööd.

Töö põhitulemusena loodi lõputööde juhendamise valdkonna ühe alamvaldkonna (juhendatava ja juhendaja kokkuviimise ehk teemade ja teemavaldkondade 'turuplatsi') mudelid. Samuti loodi graafilise keele abstraktse ja konkreetse süntaksi (ehk vastavalt *Ecore* ja *Viewpoint* mudelid) ning keele tööriistade spetsifikatsioonid. Lisaks genereeriti ja konfigureeriti *Siriuse* platvormil esmane töötav tarkvaraprototüüp, milleks on sisuliselt

graafilised mudeliredaktori tüüpi kasutajaliidesed juhendatava ja juhendaja kokkuviimiseks, mis ’käivitavad’ ühe esmastest etappidest juhendamisprotsessis.

Iteratiivses arendusprotsessis modelleeriti *Ecore* mudelite näol ja analüüsimustreid kasutades üldine Asjadeseoste ja struktuuride valdkond. Seda valdkonda spetsialiseeriti sammhaaval ülalt-alla strateegiaga (ehk üldisemast konkreetsemaks) liikudes konkreetsemateks ja kitsamateks valdkondadeks (tootestruktuurid, kliendistruktuurid jne). Lõpuks jõuti antud töö ’lõppsihini’ ehk Juhendamise (*Supervising*) struktuuride valdkonna ja vastava keele lahenduseni. Põhimõtteliselt oleks saanud ka kohe luua juhendamise valdkonna struktuuri, kui oleks kasutatud alt-üles lähenemist (konkreetsemast üldisemale valdkonnale liikumist) või ainult lõputööde juhendamise spetsiifilise valdkonna juurde jäädudki. Siis oleks jõutud kasutatava tarkvaraprototüübini kindlasti kiiremini, ent samas oleksid selle edasiarendusvõimalused olnud piiratumad ja lahenduse korduvkasutamine teistes sarnastes valdkondades vähemtõenäolisem. Praegust töös loodud lahendust saab nimetada arhitektuurseks ja ka pidevalt edasiarendatavaks prototüübiks, mida juhendamiste-kavandamise koostööprotsessis saavad nii lõppkasutajad (tudengid, õppejõud) kui ka keele spetsifitseerijad (arendajad) omavahelises koostöös suhteliselt lihtsalt muuta (ehk transformeerida). Paraku jäi selle prototüübi testkasutamiseks vähe aega, mistõttu on seda seni testinud/kasutanud ainult lõputöö autor ja tema juhendaja.

Võib öelda, et valdkonnaspetsiifilistel keeltel põhinev lähenemine aitab kommunikatsiooniprobleemi lahendamisele kaasa ning genereeritud prototüüp on lõputöö juhendamise valdkonnas rakendatav ja kasulik.

Kasutatud kirjandus

[1] F. Tomassetti. The complete guide to (external) Domain Specific Languages. – *Blogi*, 2017. [WWW] https://tomassetti.me/domain-specific-languages/ (08.05.2019)

[2] F. Ulrich. Multi-Perspective Enterprise Modelling: Background and Terminological Foundation. – *ICB-Research Report No. 46*, 2011. [WWW] https://www.econstor.eu/obitstream/10419/70904/1/738560111.pdf (15.05.2020)

[3] Mendix Tech BV. What is Low-Code? A Quick Introduction to Low-Code Development – *E-artikkel*, 2020. [WWW] https://www.mendix.com/low-code-guide/ (16.05.2020)

[4] Select Business Solutions, Inc. What is Model Driven Architecture? (*MDA*). – *E-artikkel*, 2019. [WWW] http://www.selectbs.com/analysis-and-design/what-is-model-driven-architecture-mda (27.12.2019)

[5] Object Management Group, Inc. Welcome To CORBA Web Site! – *E-artikkel*, 2020. [WWW] https://www.corba.org/ (16.05.2020)

[6] Microsoft. What is .NET? An open-source developer platform. – *E-artikkel*, 2020. [WWW] https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet (16.05.2020)

[7] P. Singhal, GeeksforGeeks. Frontend vs Backend. – *E-artikkel*, 2019. [WWW] https://www.geeksforgeeks.org/frontend-vs-backend/ (16.05.2020)

[8] Y. Singh, M. Sood. Model Driven Architecture: A Perspective. – *IEEE International Advance Computing Conference*, 2009, 1644–1652. [*Online*] IEEE (10.05.2020)

[9] Obeo. Eclipse Sirius: A graphic model is worth a thousand words. – *E-artikkel*, 2019. [WWW] https://www.obeodesigner.com/en/product/sirius (29.11.2019)

[10] GitHub, Inc. The world's leading software development platform. – *GitHub, Inc*, 2020. [WWW] https://github.com/ (04.11.2019)

[11] Draw.io. Flowchart Maker and Online Diagram Software. – *Draw.io*, 2020. [WWW] https://app.diagrams.net/ (10.05.2020)

[12] D. Granada. Comparing tools to build graphical modeling editors. – *E-artikkel*, 2016. [WWW] https://modeling-languages.com/comparing-tools-build-graphical-modeling-editors/ (25.04.2020)

[13] A. Saabas, E. Tõugu. Skeemikeelte kasutamine – implementatsioon. – *Microsoft Powerpoint*, 2004. [WWW] http://www.cs.ioc.ee/~jaan/pedase/Slaidid/pedase_ando.pdf (11.05.2020)

[14] Gronback, Richard C. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. 1st ed. Addison-Wesley Professional, 2009 [*Online*] ACM Digital Library (15.05.2020)

[15] R. V. D. Krogt. Model Driven Architecture (MDA). – *AMIS, Data Driven Blog*, 2006. [WWW] AMIS Technology Blog (16.05.2020)

[16] M. Fowler. Organization Structures (Accountability). – *E-artikkel*, < 2002. [WWW] https://martinfowler.com/apsupp/accountability.pdf (14.12.2019)

[17] J. Dupont, F. Madiot, Eclipse Foundation, Inc. Sirius/Tutorials/DomainModelTutorial – Eclipsepedia. – *Eclipse Wiki: Sirius E-artikkel*,

2019. [WWW] https://wiki.eclipse.org/Sirius/Tutorials/DomainModelTutorial (06.05.2020)

[18] F. Madiot, Annamalai, Obeo, P.-C. David jt, Eclipse Foundation, Inc. Sirius/Tutorials/ StarterTutorial. – Eclipsepedia. – *Eclipse Wiki: Sirius E-artikkel,* 2018. [WWW] https://wiki.eclipse.org/Sirius/Tutorials/StarterTutorial (05.05.2020)

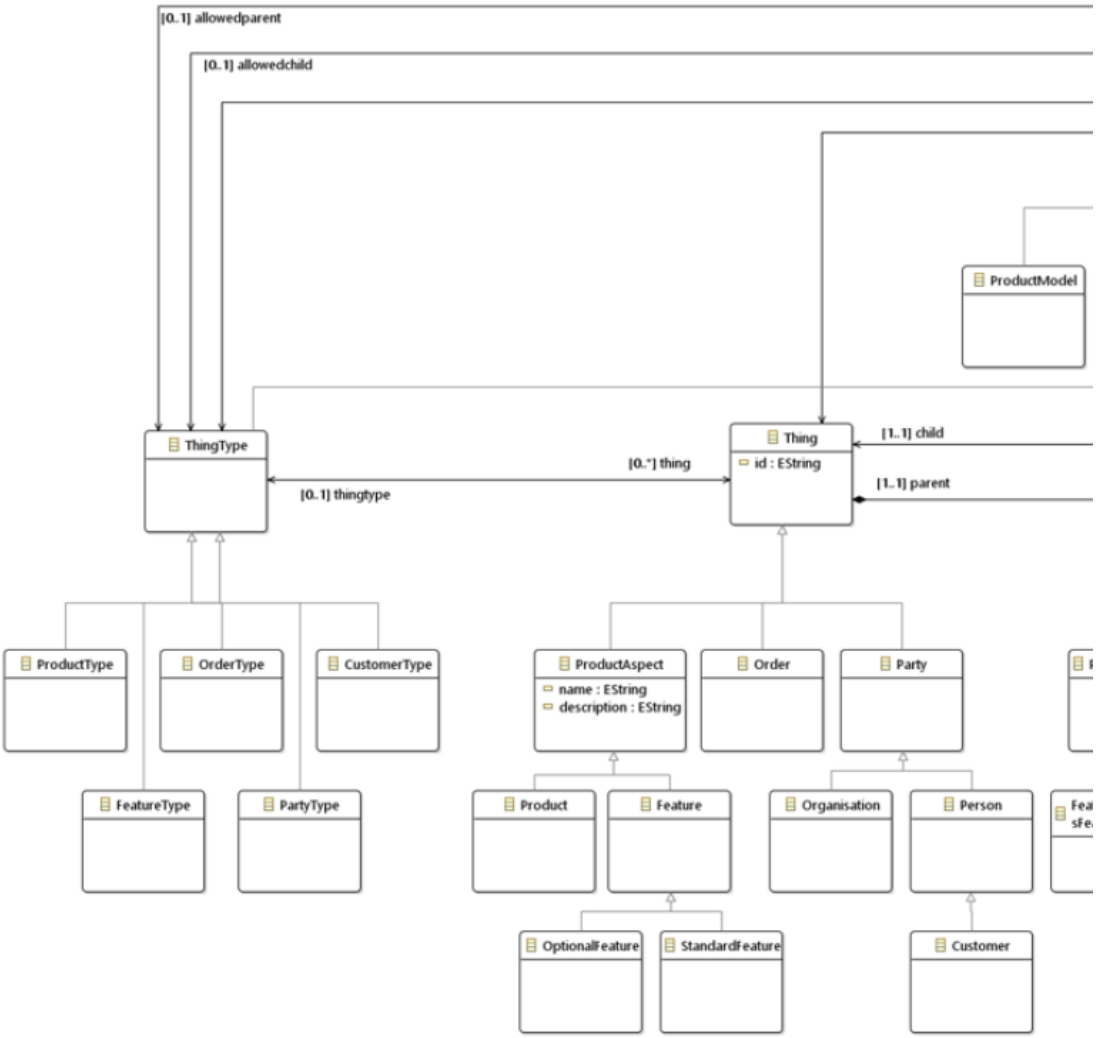
[19] K. Vorobiova. person icon. – *EIT RawMaterials,* 2020. [WWW] https://eitrawmaterials.eu/clc-location/clc-west/person-icon/ (14.05.2020)

[20] M. Wenz, Eclipse Foundation, Inc. Graphiti – a Graphical Tooling Infrastructure. – *E-artikkel,* 2020. [WWW] https://www.eclipse.org/graphiti/ (04.05.2020)

[21] J. Cabot. Sirius – The easiest way to get your own modeling tool. – *E-artikkel,* 2016. [WWW] https://modeling-languages.com/sirius-eclipse-obeo-graphical-modeling-tool/ (07.05.2020)

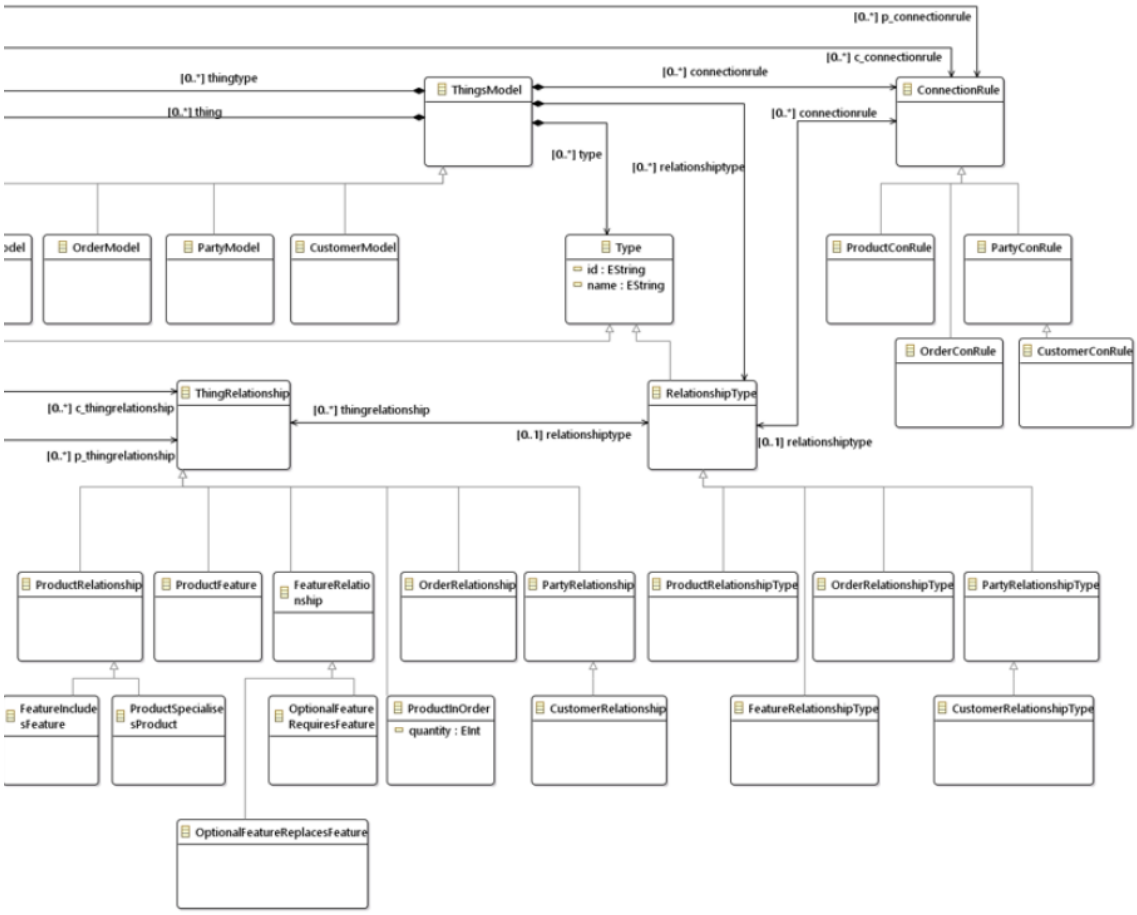
[22] F. Madiot, P.-C. David, Eclipse Foundation, Inc. Sirius/Tutorials/AdvancedTutorial - Eclipsepedia. – *Eclipse Wiki: Sirius E-artikkel,* 2018. [WWW] https://wiki.eclipse.org/Sirius/Tutorials/AdvancedTutorial (14.05.2020)

Lisa 1 *Things_relationships Ecore* metamudel



Lisa 1: *Things_relationships Ecore* metamudel.

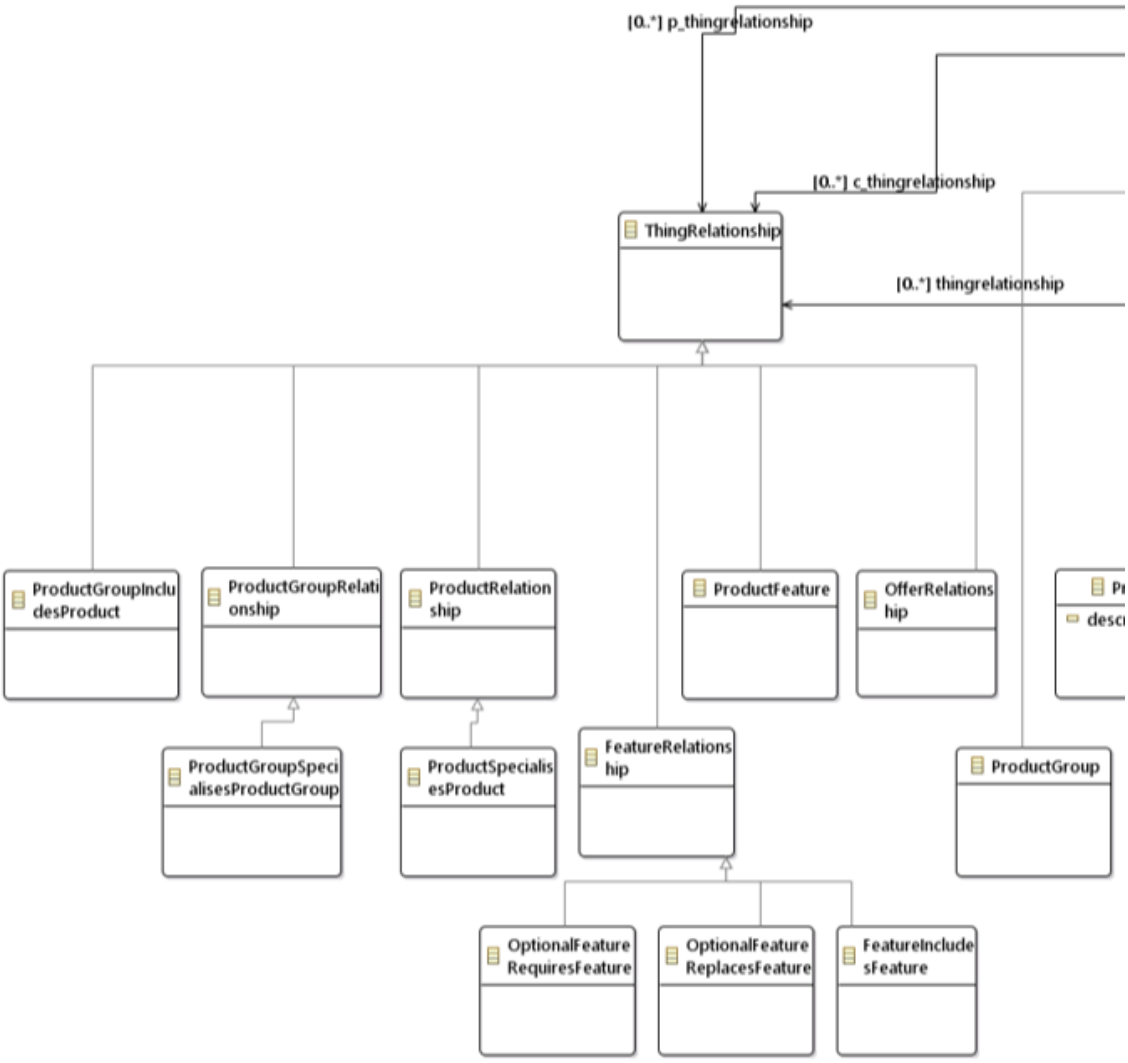
Allikas: *Eclipse Sirius – Obeo Designer Community*.



Lisa 1: *Things_relationships Ecore* metamudel.

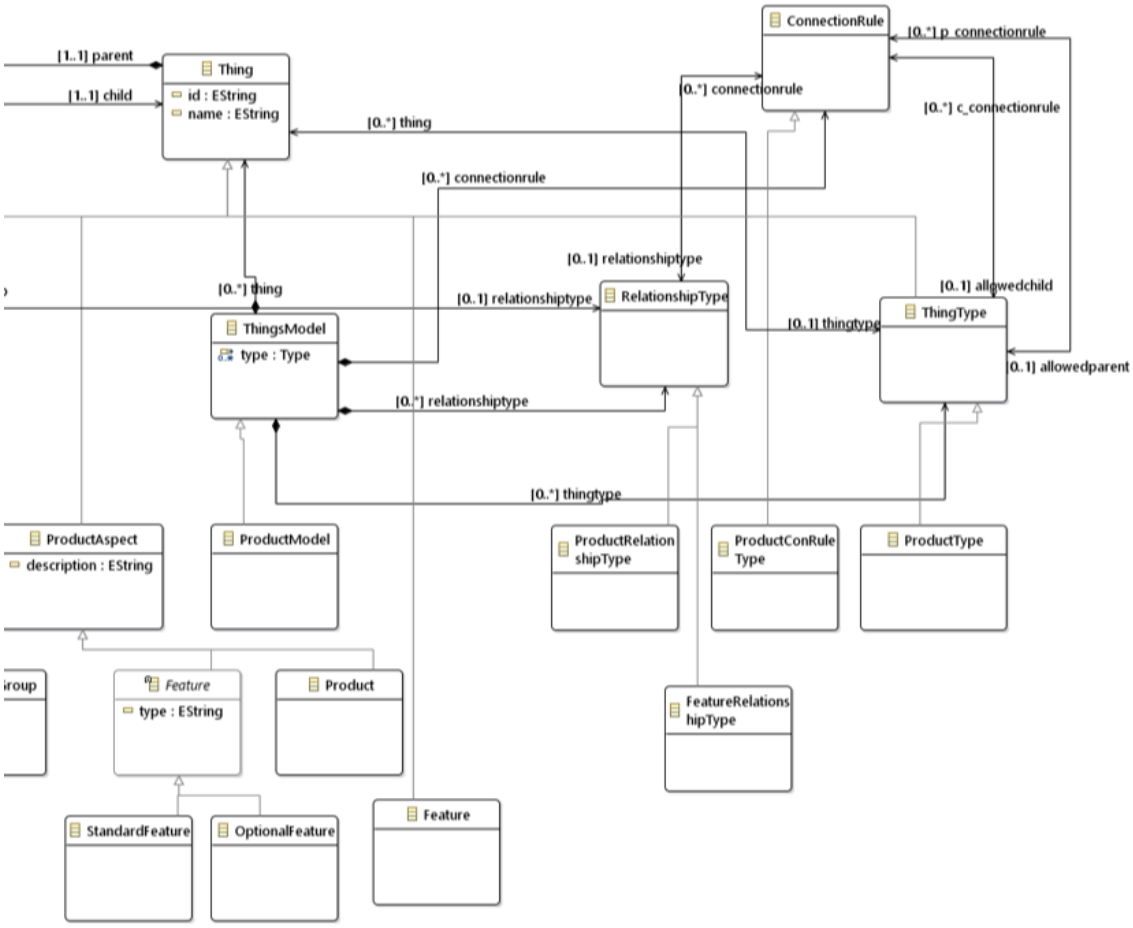
Allikas: *Eclipse Sirius – Obeo Designer Community*.

Lisa 2 *Product_relationships Ecore* metamudel



Lisa 2: *Product_relationships Ecore* metamudel.

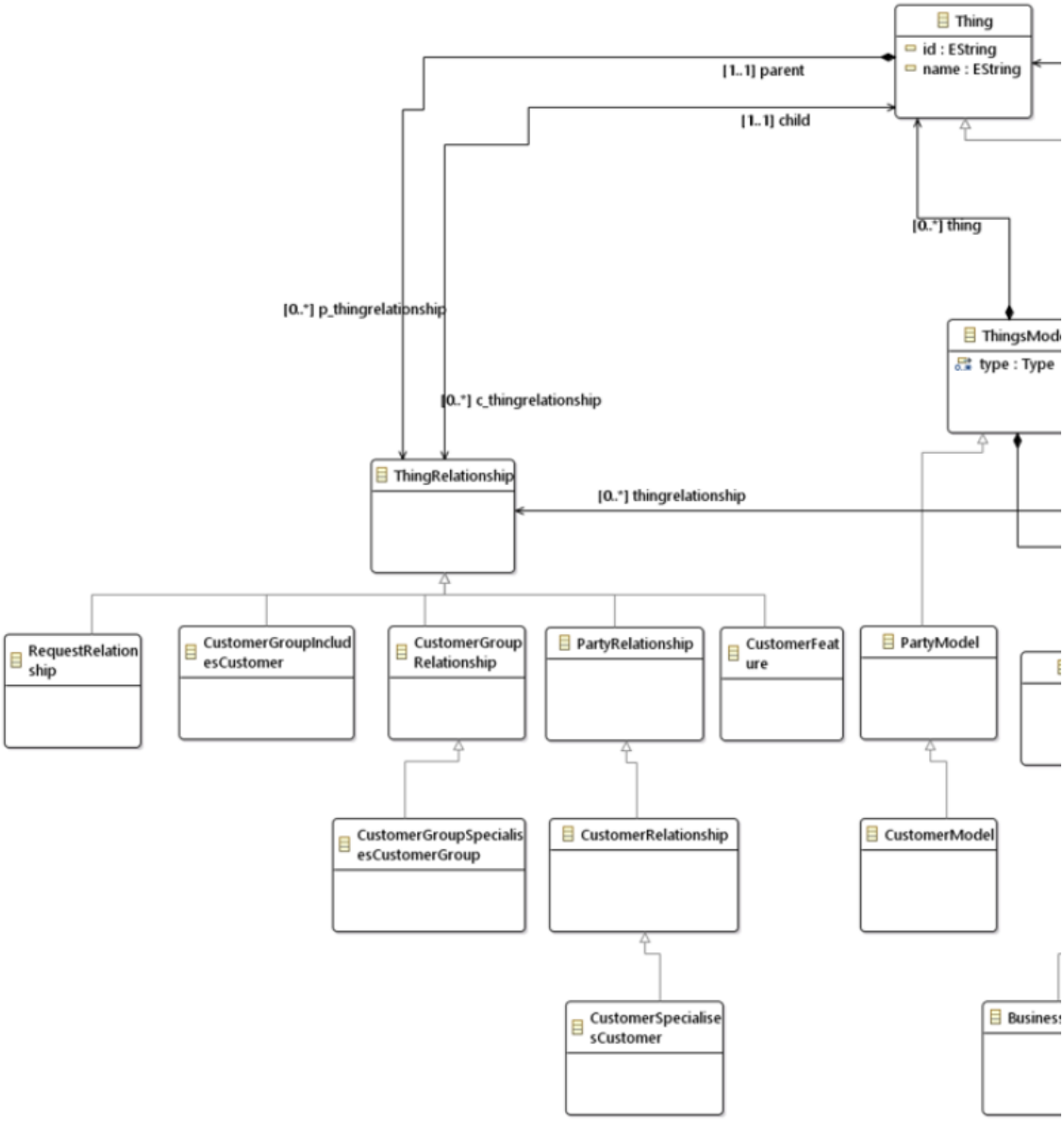
Allikas: *Eclipse Sirius – Obeo Designer Community*.



Lisa 2: *Product_relationships Ecore* metamudel.

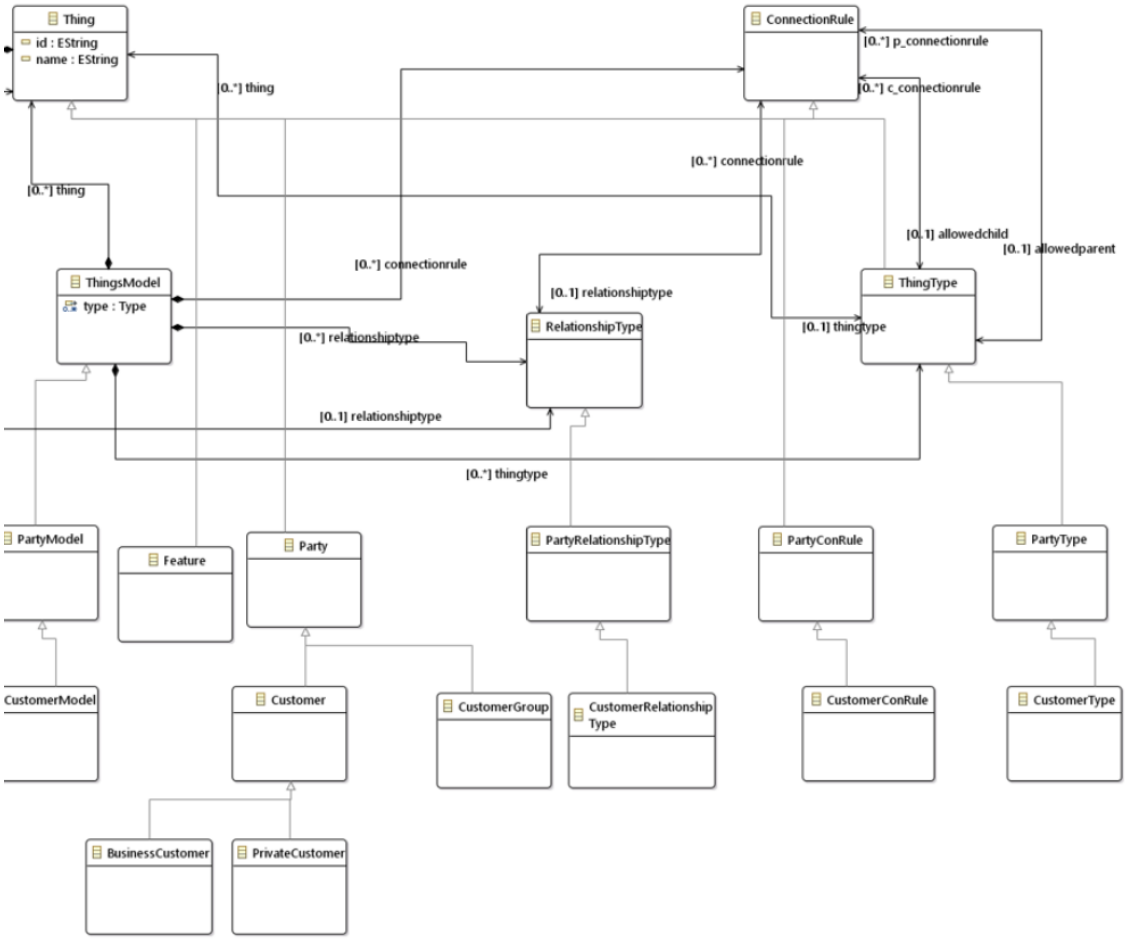
Allikas: *Eclipse Sirius – Obeo Designer Community*.

Lisa 3 *Customer_relationships Ecore* metamudel



Lisa 3: *Customer_relationships Ecore* metamudel.

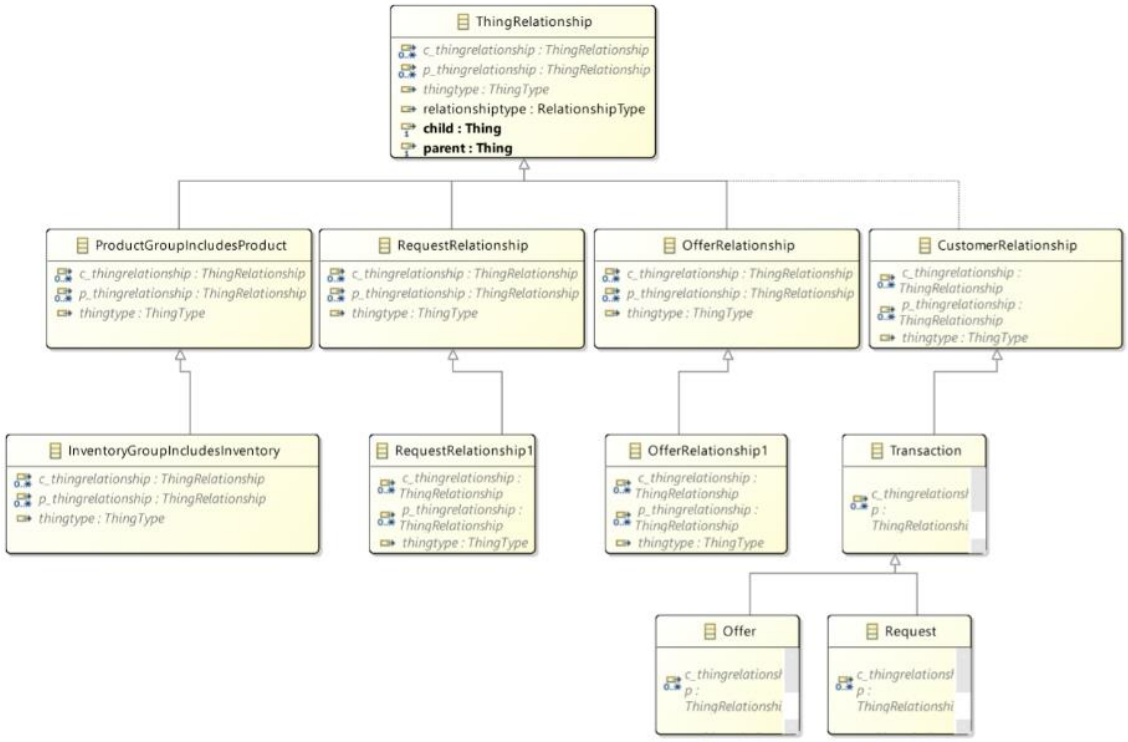
Allikas: *Eclipse Sirius – Obeo Designer Community*.



Lisa 3: *Customer_relationships Ecore* metamudel.

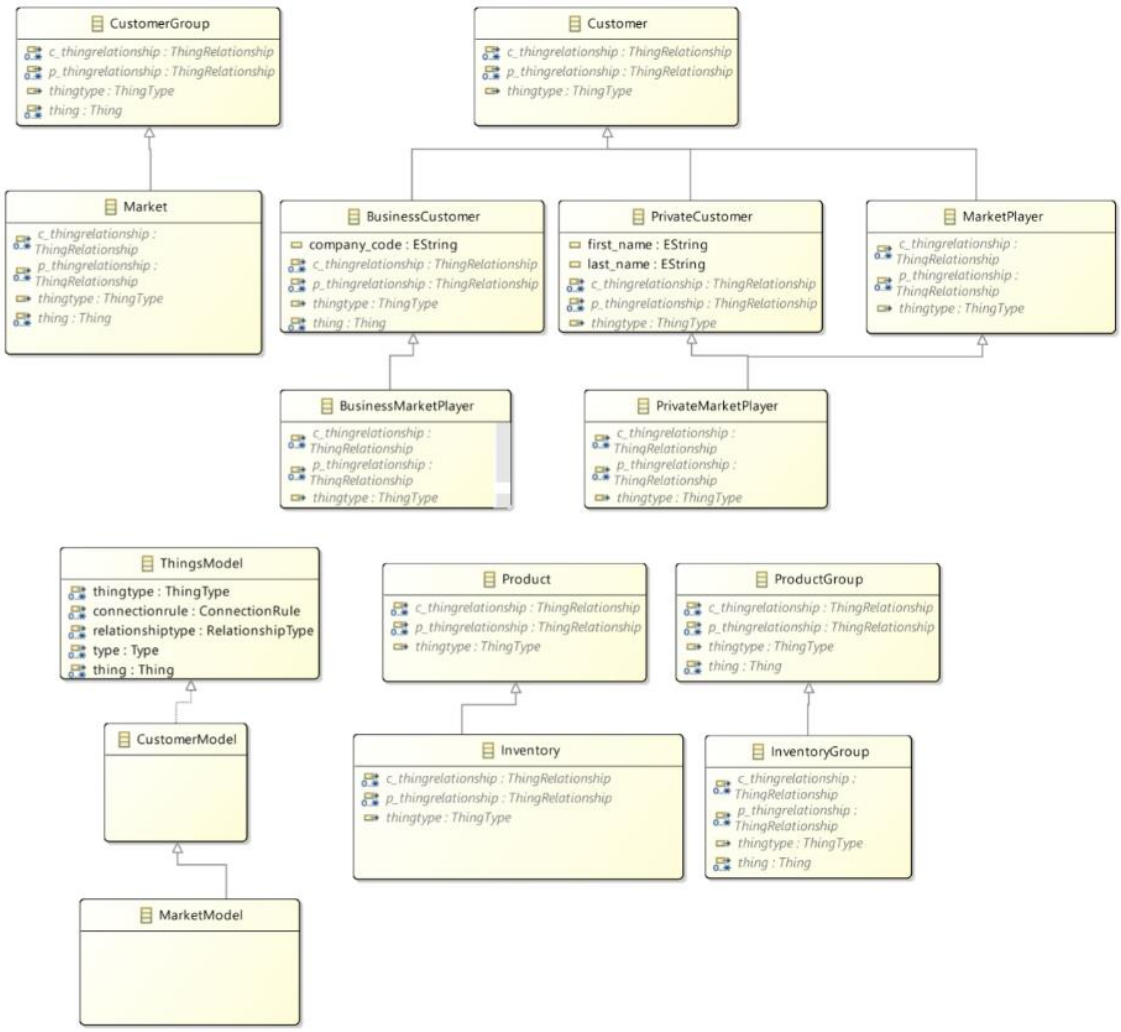
Allikas: *Eclipse Sirius – Obeo Designer Community*.

Lisa 4 *Offer_Request_relationships Ecore* metamudel



Lisa 4: *Offer_Request_relationships Ecore* metamudel.

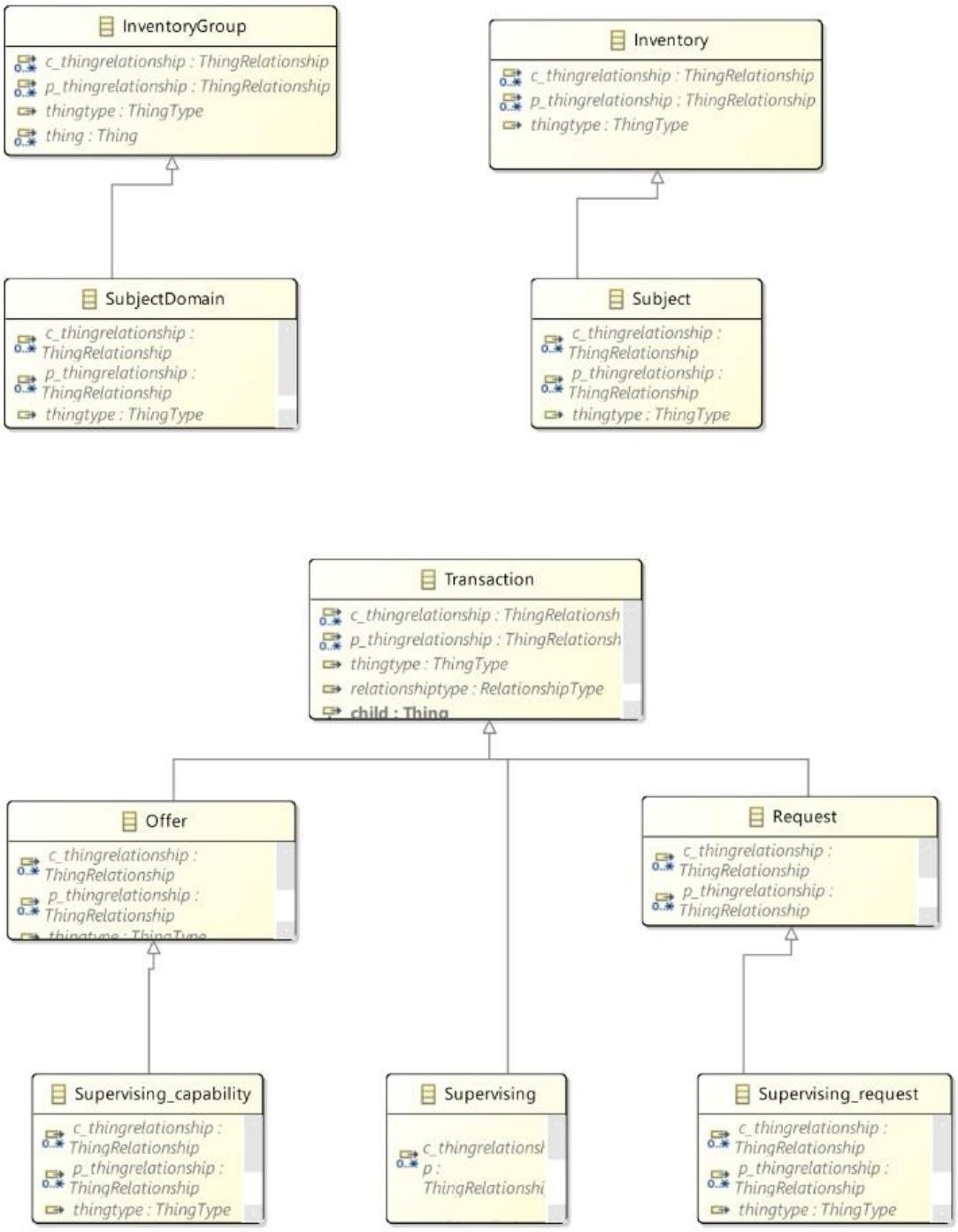
Allikas: *Eclipse Sirius – Obeo Designer Community*.



Lisa 4: *Offer_Request_relationships Ecore* metamudel.

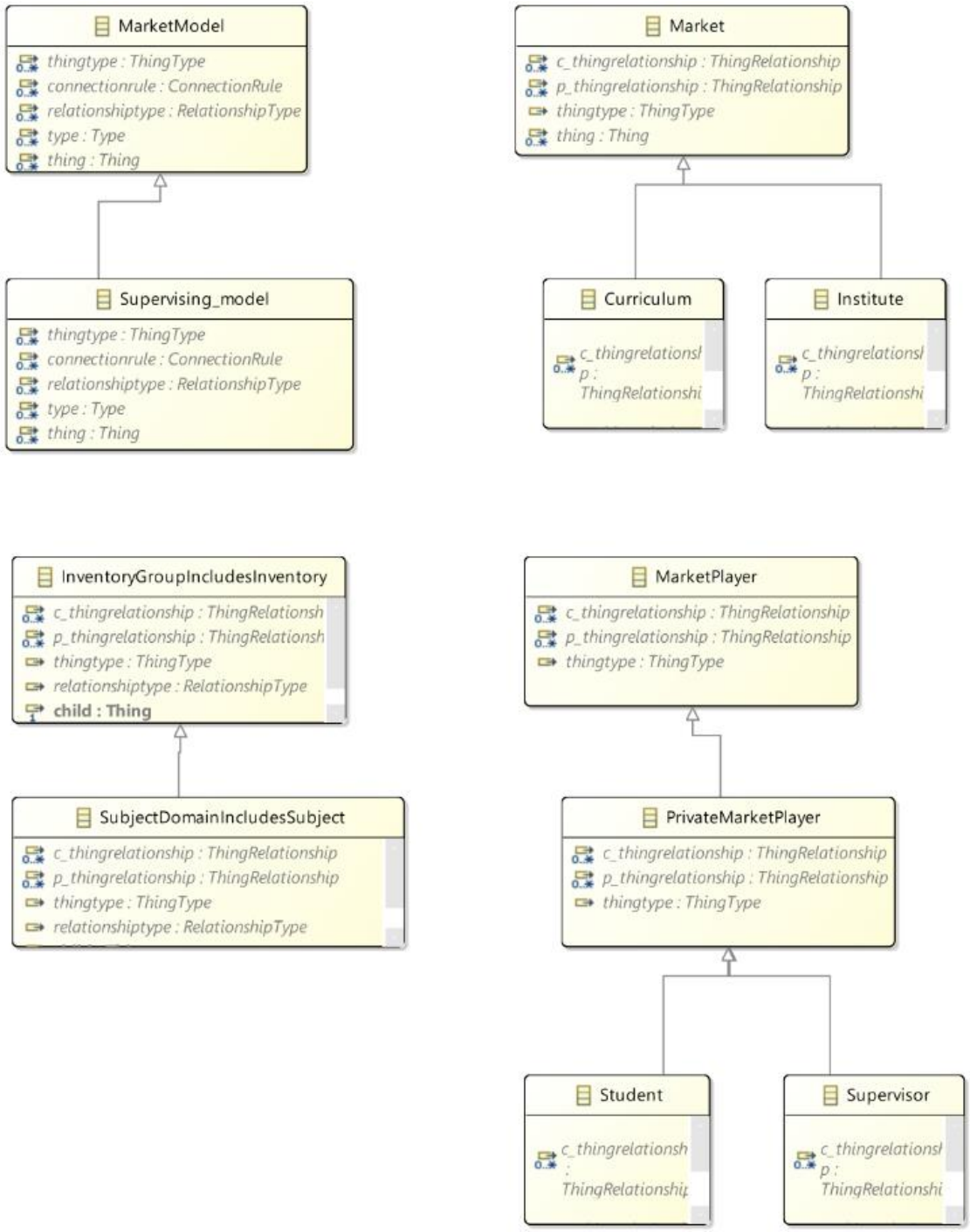
Allikas: *Eclipse Sirius – Obeo Designer Community*.

Lisa 5 *Supervising Ecore* metamudel



Lisa 5: *Supervising Ecore* metamudel.

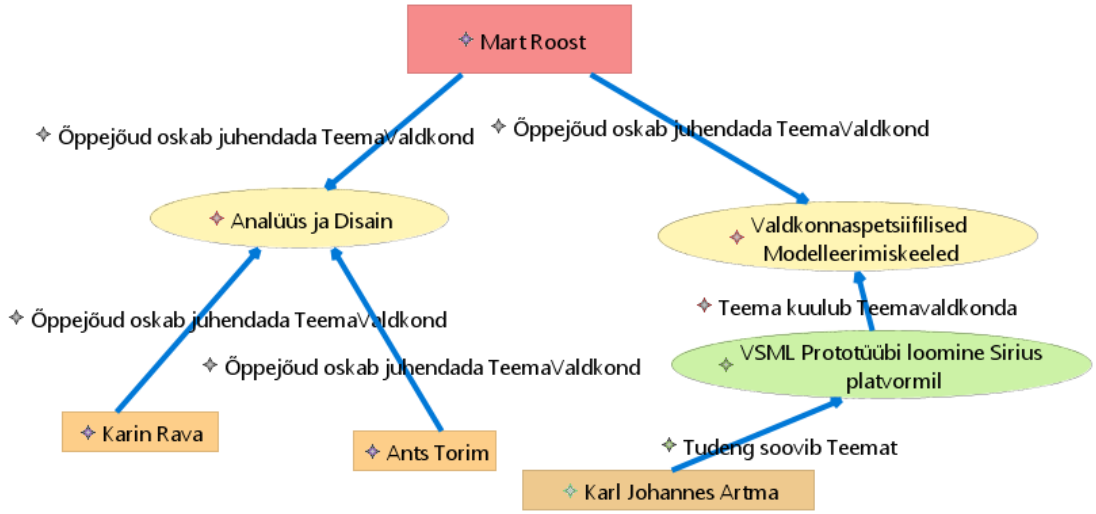
Allikas: *Eclipse Sirius – Obeo Designer Community*.



Lisa 5: *Supervising Ecore* metamudel.

Allikas: *Eclipse Sirius – Obeo Designer Community*.

Lisa 6 Juhendajakeskse mudeliredaktori tüüpi kasutajaliidese näide



Lisa 6: Juhendajakeskse mudeliredaktori tüüpi kasutajaliidese näide.

Allikas: *Eclipse Sirius – Obeo Designer Community*.