TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Andmebaasi arendamisega seotud mudeliteisenduste täiustamine Enterprise Architect CASE vahendis

Bakalaureusetöö

Üliõpilane: Marina Nekrassova

Üliõpilaskood: 120759IAPB

Juhendaja: dotsent Erki Eessaar

Tallinn
2015

Autorideklaratsioon

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

……. ……

(*kuupäev)* *(allkiri)*

Annotatsioon

Andmebaasi arendamisega seotud mudeliteisenduste täiustamine Enterprise Architect CASE vahendis

Enterprise Architect (EA) on UML CASE vahend, mis pakub palju võimalusi tarkvara mudelipõhise arenduse (MDA) läbiviimiseks. See toetab erinevaid MDA teisendusi, mis võimaldavad teisendada platvormist sõltumatud mudelid (PIM) platvormispetsiifiliteks mudeliteks (PSM) ning genereerida viimaste põhjal koodi. Muuhulgas pakub EA sisseehitatud andmebaaside arendamisega seotud redigeeritavaid teisendusmalle, kuid nende kasutamise tulemusena saadud mudelites on terve rida probleeme. Käesoleva töö eesmärgiks on täiustada olemasolevaid teisendusi nii, et nende rakendamine viiks kvaliteetsema tulemuseni. Töös käsitletakse selliseid olulisi probleeme nagu olemasolevate teisenduste puudujääkide analüüsimine, teisendusmallide redigeerimine ja uute teisenduste loomine, ning EA lisamooduli arendamine teisendusmallide poolt mittetoetatud funktsionaalsuse realiseerimiseks. Töö tulemusena on tehtud kokku 27 täiendust ja loodud näitemudel, mis demonstreerib täienduste kasutamise tulemusi. Töös on viidatud ka mitmele EA tehnilisele piirangule, mis on saanud takistuseks mõnede algselt plaanitud täienduste realiseerimisele. See loob aluse loodud tarkvara edasiarendamiseks, sest uue EA versiooni ilmumisel võivad tekkida uued võimalused teisenduste täiustamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 74 leheküljel, 4 peatükki, 32 joonist.

Abstract

Improving Database Development Related Model Transformations in Enterprise Architect CASE Tool

Enterprise Architect (EA) is a UML CASE tool with advanced support of development according to the model-driven architecture (MDA) principles. It supports various MDA transformations that allow developers to convert Platform-Independent Model (PIM) elements to Platform-Specific Model (PSM) elements and generate code based on the result. Among other things, EA provides built-in editable transformations related to database development. On the other hand, the resulting models suffer from a range of defects. The purpose of this work is to improve existing transformations in such a way that their use would produce a higher quality result. The work deals with such important problems as analysis of drawbacks of existing transformations, editing and creation of transformation templates, and development of an EA Add-In for implementing functionality not supported by the transformation templates. The main result of this work consists of 27 improvements and the example model that is used to demonstrate the results of applying improved transformations. The work also points to some of the EA technical limitations that put obstacles on the way of implementing some of the initially planned improvements. This provides a basis for further development of the software, as new EA versions may provide additional possibilities for improving the transformations.

The thesis is in Estonian and contains 74 pages of text, 4 chapters, and 32 figures.

Lühendite ja mõistete sõnastik

MDD	***Model-Driven Development*** Mudelipõhine arendus
MDA	***Model-Driven Architecture*** Mudelipõhine arhitektuur
PIM	***Platform-Independent Model*** Platvormist sõltumatu mudel
PSM	***Platform-Specific Model*** Platvormispetsiifiline mudel
UML	***Unified Modeling Language*** Unifitseeritud modelleerimiskeel
ERD	***Entity-Relationship Diagram*** Olemi-suhte diagramm
Lisamoodul	***Add-In***
Vahekeel	***Intermediary language***
Mudeliteisendus	***Model transformation***
Teisendusmall	***Transformation template***
Klassidiagramm	***Class diagram***
Kitsendus	***Constraint***

Jooniste nimekiri

Sisukord

Sissejuhatus

Tänapäeval leiab üha laiemat kasutust mudelipõhine arendus (ingl *Model-Driven Development*), mis põhineb mudelipõhisel arhitektuuril (ingl *Model-Driven Architecture*, või lühidalt *MDA*) ja mille rakendamisel jõutakse mudelite loomise ning teisendamise kaudu lõpuks süsteemi realiseerimiseks mõeldud koodini. Esimese spetsifikatsiooni ilmumisest 2001. aastal [1] on selle töö kirjutamise ajaks (kevad 2015) möödunud 14 aastat, mille jooksul ähmane kontseptsioon arenes välja küllaltki edukaks tehnoloogiate kompletiks. Selle edukuse taga on aktiivne MDA toetuse juurutamine suurte UML CASE vahendite tootjate poolt, samuti uute MDA vahendite ilmumine. Kaasaegsed modelleerimisvahendid kuuluvad juba MDA mõttes teise generatsiooni [2], olles teinud viimase aastakümne jooksul läbi märkimisväärse arengu. On aga selge, et nendel vahenditel on endiselt palju arenguruumi, sest kuigi koodi genereerimine platvormispetsiifilistest mudelitest (PSM-mudelitest) on automatiseeritud ligilähedalt 100%, siis teisendamine platvormist sõltumatutest mudelitest (PIM-mudelitest) PSM-mudeliteks on tüüpiliselt toetatud vaid 50% kuni 70% ulatuses [3].

Seda statistikat kinnitab osaliselt ka tuntud MDA kommertsvahend Enterprise Architect, mille põhjal on tehtud käesolev töö. TTÜs võeti seda kasutusele 2014. a. sügisel [4] ning kasutatakse infosüsteemide (sh andmebaaside) alastes õppeainetes modelleerimisvahendina. Kuna andmebaaside ainetes käsitletakse muuhulgas mudelipõhise arenduse teemat, on seal vajadus tõhusa MDA vahendi järele. Enterprise Architect on küll sobilik kontseptuaalsete andmemudelite (PIM) tegemiseks, kuid nende mudelite automaatsel teisendamisel füüsilise disaini täpsusega andmemudeliks (PSM) on tulemuses suuri puudujääke. Teisendamise protsessi saab mõjutada redigeerides CASE vahendisse sisseehitatud teisendusmalle. Käesoleva töö eesmärgiks on parandada teisendusmalle, et kõrvaldada olemasolevad puudused ja saavutada korrektne ning võimalusterohke teisendamine. Seda vajavad nii andmebaaside õppijad kui ka nende arendamisega tegelejad.

Töö põhiosa on struktureeritud järgmiselt. Esimese peatükis antakse ülevaade mudelipõhisest arendamisest üldse ja seoses andmebaaside arendamisega. Teises peatükis tutvustatakse UMLi ja seda kasutamist andmebaaside projekteerimisel. Kolmandas peatükis kirjeldatakse Enterprise Architect-i ja selle pakutavaid laiendusvõimalusi. Neljas peatükk sisaldab põhilist ülevaadet tehtud tööst, kirjeldades olukorda enne täienduste lisamist, lisatud täiendusi ja nende tehnilist realisatsiooni ning esitatakse installeerimisjuhend. Samas peatükis esitatakse töö käigus

avastatud probleemid ja pakutakse loodud tarkvara edasiarendamise võimalusi. Peatüki lõpus tuuakse välja näitemudel ja selle teisendamise tulemus pärast täienduste installeerimist. Antud töös esitatud kontseptuaalne andmemudel ei ole mõeldud kirjeldama konkreetse ettevõtte andmebaasi. Selles on kasutatud võimalikult palju erinevaid kontseptuaalse modelleerimise võimalusi, et demonstreerida generaatori käitumist nende korral.

1. Mudelipõhine arendamine

Mudelipõhise arendamise (MDD) all mõistetakse üldist süsteemide (sh tarkvarasüsteemide) arendamise põhimõtet, mille kohaselt luuakse süsteemi lihtsustatud kirjeldused mudelite näol ning nende mudelite teisendamise tulemusena jõutakse lõpuks süsteemi töölepanekuks vajaliku koodini. Loodavad mudelid ei saa samas olla ka liiga suured lihtsustused, sest muidu pole nendest võimalik töötava süsteemini jõuda. MDA e mudelipõhine arhitektuur (MDA) pakub ühe võimaliku nägemuse, kuidas mudelipõhine arendamine peaks toimuma. MDA [5] on tarkvara projekteerimise lähenemisviis ja samanimeline standardite kogum, mille väljatöötamisega tegeleb aastast 2001 Object Management Group (OMG) konsortsium. Laiemas mõttes on MDA raamistik, mis toetub mudelite kasutamisele tarkvaraarenduses ning selle rakendamise tulemusena loodud mudeleid võib nimetada MDA-mudeliteks. Neid mudeleid võib luua erinevates keeltes, kuid tänapäeval on laialt levinud OMG poolt arendatav üldotstarbeline visuaalne modelleerimiskeel UML, millega koos MDA põhimõtteid sageli rakendatakse. MDA-mudelid kasutatakse väga erinevate eesmärkide saavutamiseks. Nimetame neist vaid mõned [6]:

- mudel kui rühma kommunikatsioonivahend, et soodustada diskussiooni ning luua ühine arusaam arendatavast süsteemist;
- mudeli kasutamine artefaktide e tehiste automatiseeritud loomiseks (genereerimiseks) ehk mudeli teisendamine koodiks või teiseks mudeliks;
- mudel kui otseselt käivitatav mudel, mis võimaldab loodud kavandi kiiresti sihtsüsteemis realiseerida;
- mudelitest uue informatsiooni tuletamine.

Kitsamas tähenduses kasutatakse terminit „MDA“, et tähistada MDA kasutamist oluliseimal ja levinuimal viisil – artefaktide automatiseeritud loomine spetsiaalsete teisendusreeglite abil. Edaspidi mõtleme MDA mõiste all just sellist kasutusviisi.

MDA rakendamise põhimõtteid võib lihtsustatult kirjeldada järgmiselt. Kõigepealt tuleb defineerida süsteem kontseptuaalsel tasemel, luues platvormist sõltumatu mudeli (ingl *Platform-Independent Model,* või lühidalt *PIM*). PIM-mudeli põhimõte on esitada süsteemi platvormi tehnilistest piirangutest sõltumatu kirjeldus (kontseptuaalne disain), mis on kasutatav ka siis, kui süsteemide realiseerimiseks kasutatavad vahendid ajas muutuvad. Valmis PIM-mudeli saab teisendada üheks või mitmeks platvormispetsiifiliseks mudeliks

(ingl *Platform-Specific Model*, või lühidalt *PSM*). PSM on mudel, mis kirjeldab süsteemi realisatsiooni, arvestades realiseerimise platvormist (tarkvaraline ja riistvaraline keskkond) tulenevate võimaluste ja piirangutega. Viimaseks sammuks on koodi genereerimine PSM-mudeli alusel. Kuna sisuliselt on PSM-mudel üks-üheselt teisendatav vastava platvormi koodiks, siis seda teisendust on kõige lihtsam realiseerida ja sellega on seletatav selle teisenduse lai toetus erinevate MDA vahendite poolt.

Teisendamine erinevate mudelite vahel ja koodi genereerimine toimub mudeliteisenduste abil. Mudeliteisendus on sisuliselt spetsifikatsioon, mis defineerib teisendusreeglid kasutades mingit teisenduskeelt [7]. Teisendusreegel kirjeldavad vastavusi lähtemudeli loomise keele ja sihtmudeli loomise keele elementide vahel. Võib eristada kahte põhilist teisenduskeelte perekonda:

- teisendusmudelitel põhinevad keeled,
- teisendusmallidel põhinevad keeled.

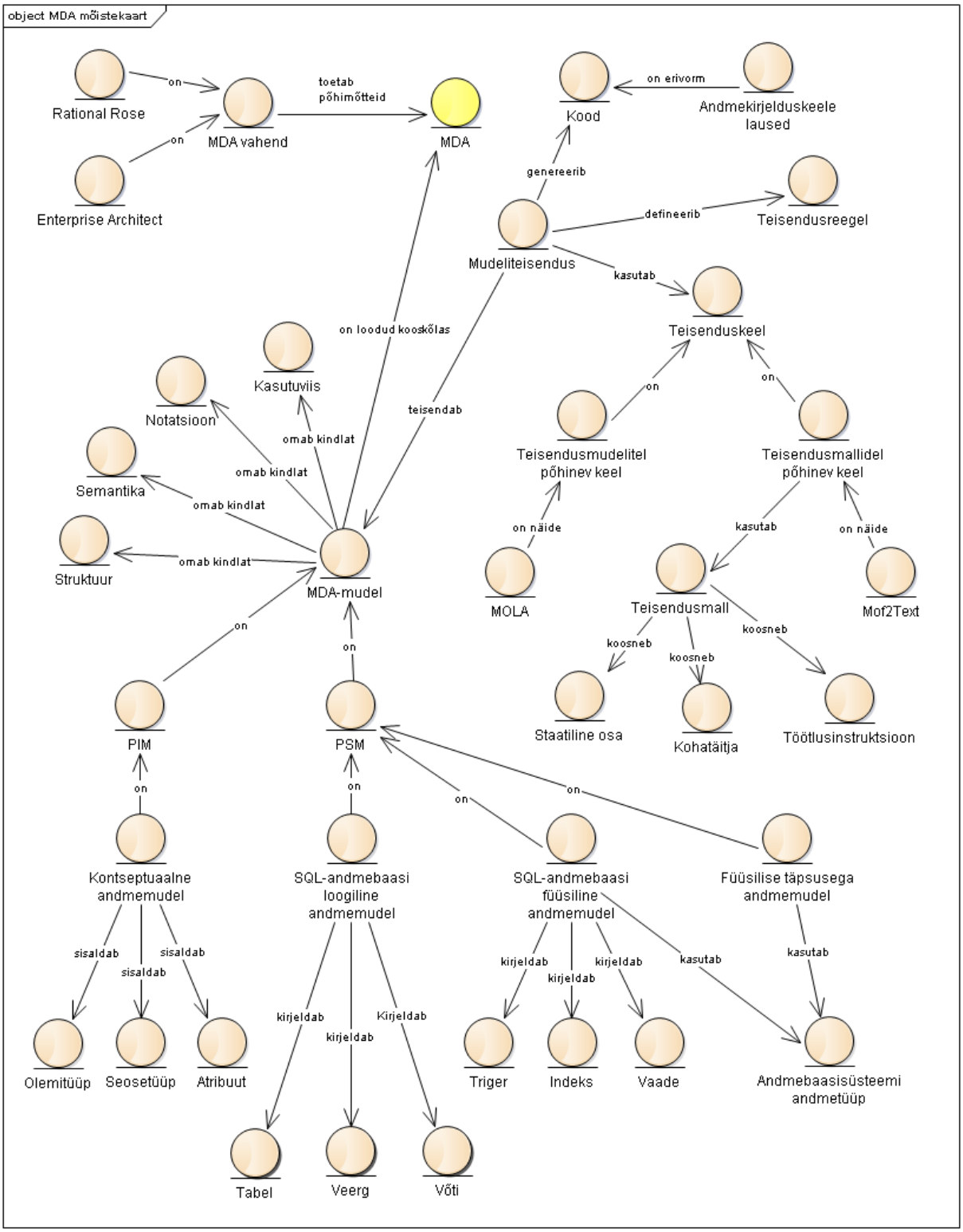
Esimeste puhul määratakse teisendusreeglid graafiliste mudelite abil, nii et võib öelda, et mudelid teisendatakse mudelitega. Teisendusmallid on lihtsustatult öeldes tekstifailid, mis koosnevad mittemuutuvast ehk staatilisest osast, mida kopeeritakse väljundisse täpselt samal kujul ja kohatäitjadest (ingl *placeholder*), mida täidetakse teisendamisel mudelist saadud andmetega. Sõltuvalt keelest, võib teisendusmallides kasutada ka erinevaid töötlusinstruktsioone (ingl *processing instructions*), mis lisavad teisenduskeelele skriptikeele omadusi. Teisendusmudelitel põhineva keele näide on MOLA [8]. Teisendusmallidel põhinev keel on näiteks Mof2Text [9].

MDA põhimõtted on edukalt rakendatavad ka andmebaaside arendamisel. Kontseptuaalsed andmemudelid, mis defineerivad olemitüübid, nendevahelised seosetüübid ja atribuudid, on PIM-mudelid, mis ei sõltu ei andmebaasisüsteemist ega andmemudeli tüübist (nt SQL, relatsiooniline, hierarhiline, võrkmudel). Kontseptuaalsete andmemudelite teisendamise tulemusena saadakse loogilised PSM-andmemudelid, mis pole veel seotud konkreetse andmebaasisüsteemiga, aga arvestavad andmemudeliga, mille põhjal andmebaas realiseeritakse. Näiteks, loogiline SQL PSM-mudel sisaldab selliste elementide kirjeldusi nagu tabel, väli, primaarvõti, välisvõti, jne. Tabelid on disainitud viisil, et andmete liiasus oleks võimalikult väike (tabelid on viiendal normaalkujul) Edaspidi toimub selle mudeli konverteerimine füüsiliseks PSM-andmemudeliks, mis arvestab andmebaasi realiseerimiseks

kasutatava andmebaasisüsteemi võimaluste ja piirangutega. Füüsiline mudel sisaldab täpsustatud andmetüüpe, mida toetab vastav andmebaasisüsteem, samuti trigereid, vaateid, indekseid ja muid realisatsioonist sõltuvaid elemente. Lisaks võib füüsilise SQL PSM-mudeli korral olla tabeleid denormaliseeritud, et andmete liiasuse suurendamise kaudu parandada mõne lugemisoperatsiooni töökiirust. Viimase sammuna genereeritakse füüsilisest PSM-mudelist andmekirjelduskeele laused – nendega saab luua andmebaasiobjekte.

Seoses eelnevaga on oluline märkida, et ülalkirjeldatud protsessi tegelik realisatsioon sõltub konkreetsest MDA vahendist. Mõlemas vahendis, millega autor on kokku puutunud – Rational Rose ja Enterprise Architect – genereeritakse kontseptuaalsest andmemudelist füüsilise disaini täpsusega andmemudel, mille puhul on juba määratud ka kasutatav andmebaasisüsteem. See mudel on peale täienduste ja paranduste sisseviimist juba aluseks andmekirjelduskeele lausete genereerimiseks.

Joonis 1 on esitatud mõistekaart, mis kirjeldab selles peatükis käsitletud mõisteid ja nendevahelisi seoseid.



Joonis 1. MDA põhimõisted ja seosed nende vahel

2. UML

Mudelipõhise arendamise aluseks võivad olla UML mudelid. UML e unifitseeritud modelleerimiskeel (ingl *Unified Modeling Language*), on tarkvaraarenduses kasutatav üldotstarbeline modelleerimiskeel, mis on mõeldud kavandatava info- või tarkvarasüsteemi visualiseerimiseks [10]. Aastast 1997 tegeleb selle keele arendamisega ja standardiseerimisega OMG [11] – sama organisatsioon, mille pädevuses on MDA standardi väljatöötamine. Oma olemuselt on UML metamudelile toetuv graafiliste tähistuste pere [12]. Graafilised tähistused moodustavad UML mudelites kasutatava notatsiooni. Metamudeli abil kirjeldatakse UML keele grammatika, mis määrab ära UML mudelite struktuuri ja elementide tähenduse.

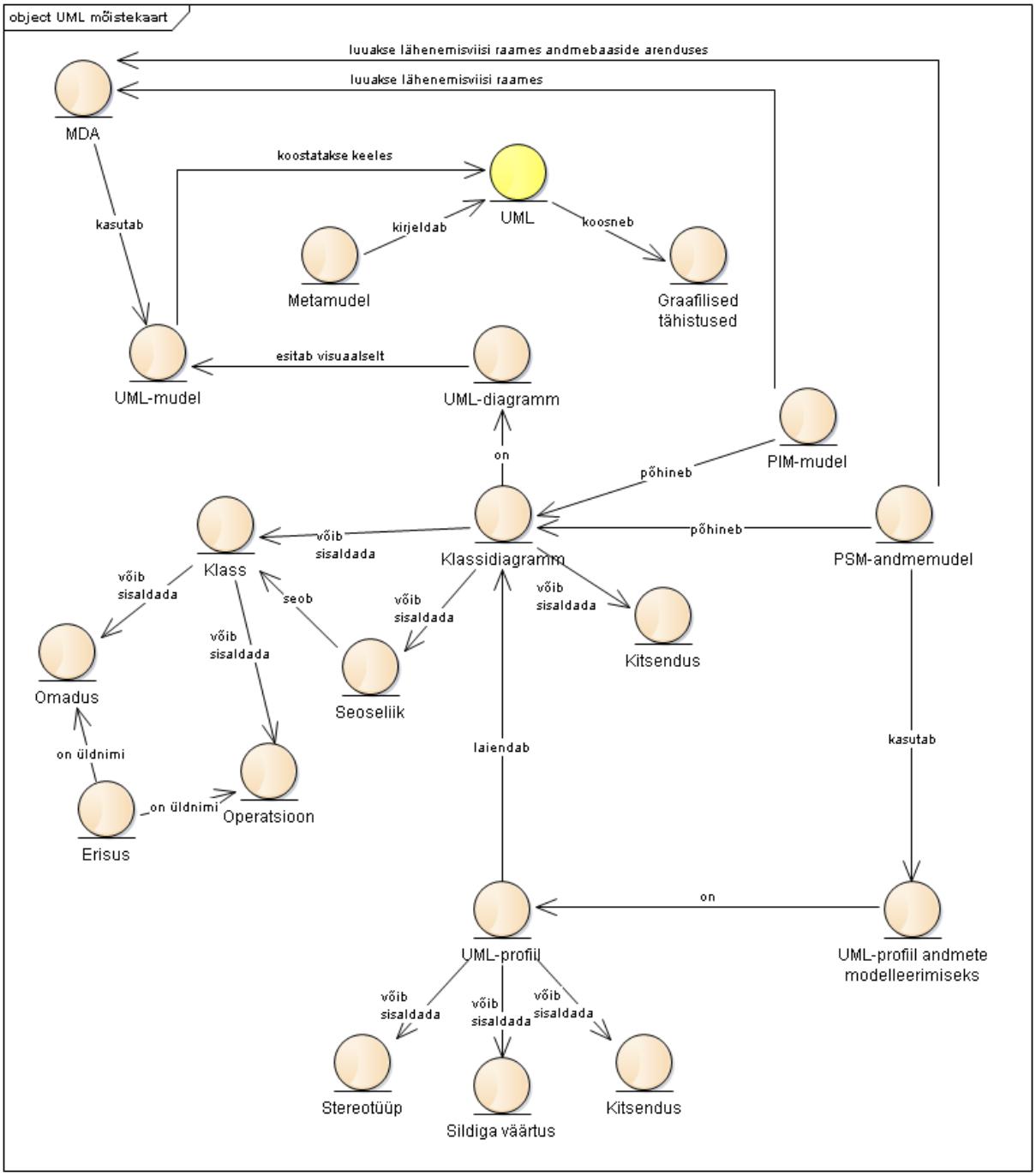
UML mudeli graafiline esitus toimub UML-diagrammidena. UML standardi hetkel (2015. aasta kevadel) kehtiv versioon 2.5 [13] defineerib kokku 14 tüüpi diagramme. Neist on kõige rohkem levinud klassidiagramm, mis kirjeldab süsteemi objektide tüüpe ehk klasse, nendevahelisi seoseliike ja kitsendusi. Samas esitatakse klassidiagrammis selliseid elemente nagu klassi omadused ja operatsioonid, mis kannavad UMLis üldnime "erisus" (ingl *feature*). MDAs omavad klassidiagrammid erilist tähtsust, sest need on paljude PIM-mudelite nurgakiviks. Andmebaaside projekteerimisel esitatakse PIM klassidiagrammis olevate klasside abil olemitüüpe, seoseliikide (assotsiatsioonid ja üldistused) abil seosetüüpe ning klassi omaduste abil atribuute; klassi operatsioone seal ei esitata.

Rääkides UMLi kasutamisest MDA-põhises arendamises, siis PIM teisendamisega saadavad PSM-mudelid pole klassidiagrammidega niivõrd seotud. Kuna nendega kirjeldatavad realisatsioonid on erisugused, siis eristuvad märgatavalt ka realisatsioonide esitusviisid. Kuna käesoleva töö fookuses on MDA rakendamine andmebaaside projekteerimises, ei peatu me sellel teemal pikemalt, sest PSM-andmemudelid esitatakse samuti klassidiagrammidena. Erinevus seisneb aga selles, et nendes PSM-diagrammides kasutatakse elementide kirjeldamiseks spetsiaalseid profiile.

UML-profiil on laienduse mehhanism, mis võimaldab kohandada UML mudeleid, võttes arvesse konkreetse platvormi või domeeni spetsiifikat [14]. Profiilide ehitusplokid on stereotüübid, sildiga väärtused (ingl *tagged values*) ja kitsendused, mida saab lisada klassidele, omadustele, seostele ja operatsioonidele. Nende konstruktsioonide abil saab muuta elementide semantikat, tähistust ja omadusi ning seostada elementidega täiendavaid kitsendusi. Kuigi ei eksisteeri standardset profiili andmete modelleerimiseks [15], on välja kujunenud laialt

aktsepteeritav mitteametlik profiil [16], mida kasutavad ka suured UML CASE tarkvaratootjad [17]. Näiteks PSM-andmemudelites modelleeritakse tabelit kui klassi stereotüübiga <<table>>, indekseid ja trigereid kui vastavate stereotüüpidega klassi operatsioone ning primaarvõtit kui atribuuti stereotüübiga <<PK>> ja samanimelise stereotüübiga klassi operatsiooni.

Käesolevas peatükis esitatud mõisted ja seosed nende vahel on illustreeritud Joonis 2.



Joonis 2. UML põhimõisted ja seosed nende vahel

3. Enterprise Architect

Enterprise Architect (EA) on UML-põhine visuaalse modellerimise kommertsvahend, mida arendab austraalia firma Sparx Systems [18]. Töö tegemise alustamise hetkel (2015. aasta talv) oli viimane versioon 11 ning töö on tehtud selle versiooni põhjal. 2015. a. veebruaris ilmus versioon 12, mis toetab kõige viimast UML standardit 2.5; töös loodud teisendused ja lisamoodul töötavad ka uues versioonis. Lisaks UMLile võimaldab vahend mudeleid luua ka teistes keeltes.

MDA töövahendina pakub Enterprise Architect mitmekülgset toetust analüütikutele ja arendajatele [19]. EA peamine MDA komponent on teisendusmallidel põhinev teisendusmootor-koodigeneraator, mis võimaldab ühe käivitusega transformeerida algset PIM-mudelit mitmesse erinevasse PSM-mudelisse ja genereerida tulemuse põhjal koodi. Teisendamisel loob teisendusmootor peidetud lingid PSM- ja PIM-mudelite vahel, nii, et muudatused PIM-mudelis on sujuvalt sünkroniseeritavad PSM-mudelitega. Teisendusmallid (ka koodi genereerimise mallid) on redigeeritavad, nendest on pikem kirjutatud alapeatükis 3.1.

Mallid pole sugugi ainus viis EA sisseehitatud funktsionaalsust laiendada. Enterprise Architect-i SDK dokumentatsioon [20] kirjeldab, kuidas luua uusi UML profiile, lisada stereotüüpe ja muuta skriptikeele abil stereotüübitud elementide tähistust, defineerida sildiga väärtuste tüüpe, luua eraldiseisvaid ActiveX kliente EA mudelite manipuleerimiseks *Automation Interface*-i kaudu ning, mis on selle töö kontekstis kõige tähtsam – luua ja registreerida sisseehitatavaid lisamooduleid (ingl *Add-In*). Kuna üks selle töö tulemusena valminud artefaktidest on EA lisamoodul, siis vaatleme neid täpsemalt alapeatükis 3.2.

Enterprise Architect toetab tegevusi, mis hõlmavad andmebaasi MDA-põhise arenduse etappe, alates detailse kontseptuaalse mudeli (edaspidi nimetame seda ka *algmudeliks*) loomisest kuni andmekirjelduskeele lausete genereerimiseni [21]. Selleks vajalike mudelite kirjeldamisel saab kasutada kahte tüüpi diagramme: UMLi klassidiagrammid ja ERD-diagrammid. Sisseehitatud teisenduste abil võib konverteerida detailse kontseptuaalse andmemudeli (EA nimetab seda *Logical Data Model*) füüsilise disaini täpsusega andmemudeliks (EA terminoloogias *Physical Data Model*) – see ongi käesolevas töös parandatav PIM => PSM teisendus. Saadud PSM mudelis kasutab EA tabelite, veergude ja muude elementide kirjeldamiseks enda andmete modelleerimise UML profiili [22]. Selle mudeli põhjal saab genereerida SQL-keele laused

andmebaasi skeemi ning selles olevate skeemiobjektide loomiseks. Enterprise Architect-i versioon 12 toetab kokku 14 andmebaasisüsteemi [23].

3.1. Teisendusmallid, teisenduskeel ja vahekeel

Enterprise Architect pakub terve rida sisseehitatud teisendustüüpe [24]. Nimetame vaid mõned neist:

- PIM => C#, C++, EJB Entity Bean, EJB Session Bean, Java, PHP, VB .NET, XSD
- PIM => DDL table elements (selles töös parandatav teisendustüüp)
- Java Model => JUnit Test Model
- ja palju muid…

Arendaja saab muuta olemasolevaid teisendustüüpe ja luua juurde uusi tüüpe. See toimub teisendusmallide redigeerimise kaudu. Nii mudelist-mudeliks mudeliteisenduses, kui ka koodi genereerimisel kasutatavad mallid on kirjutatud EA poolt loodud teisenduskeeles (EA dokumentatsioonis viidatakse sellele kui "koodi genereerimise mallide keel" - ingl *code generation template language*). EA teisenduskeele abil saab kätte *piiratud* hulka mudeli andmeid, samuti pakub see mitukümmend funktsioone (ingl *function macro*), millest enamus on mõeldud lihtsamaks stringitöötluseks. Tavalistest programmeerimiskeele konstruktsioonidest on saadaval ainult `if-else`, kuid tsüklid puuduvad. Tüüpidest on olemas teksti- (*string*) ja täisarvutüüp. Täiendav arutelu keelest ja selle rakendamisest toimub alapeatükis 4.3.1, siin aga esitame koodinäide. Järgmine koodilõik (vt Joonis 3) on võetud sisseehitatud DDL-nimelise teisendustüübi mallist *Class* ja sellega defineeritakse reeglid olemitüübi teisendamiseks tabeliks (täpsemalt küll olemitüübi kirjelduse teisendamiseks tabeli kirjelduseks).

```
Table
{
  %TRANSFORM_REFERENCE("Table")%
  %TRANSFORM_CURRENT("language", "stereotype")%
  language=%qt%%genOptDefaultDatabase%%qt%
%list="Attribute" @separator="\n" @indent="  "%
%if elemType != "Association"%
  PrimaryKey
  {
    Column
    {
      name=%qt%%CONVERT_NAME(className, "Pascal Case","Camel Case")%ID%qt%
      type=%qt%%CONVERT_TYPE(genOptDefaultDatabase,"Integer")%%qt%
    }
  }
%endIf%
}
```

Joonis 3. Koodilõik DDL teisendusmallist Class

Teisendamisel interpreteerib EA seda malli, kasutades mudelis olevaid andmeid. Teisenduse väljundiks on vahefail, mis on aluseks teisendatud mudelielemendi graafilise kujutise koostamisele. Vahefail on kirjutatud spetsiaalses vahekeeles (ingl *Intermediary Language*) [25]. Teisendusmallidega määrataksegi, kuidas luua vahekeeles sihtmudeli elementide kirjeldusi, seega mallide kirjutamiseks on vaja teada ka vahekeele süntaksi [26]. Joonis 4 on esitatud vahefaili osa, mis vastab ühele võimalikest ülalpool toodud näite teisendamisega saadud väljunditest.

```
Table
{
  XRef{namespace="DDL" name="Table"
  source="{0BC360C6-C9B4-46a0-861B-4D5C020425C1}"}
  notes=""
  scope="Public"
  multiplicity=""
  version="1.0"
  author="Marina"
  alias=""
  name="Arve"
  complexity="1"
  concurrency=""
  persistence=""
  arguments=""
  phase="1.0"
  status="Proposed"
  visibility=""
  cardinality=""
  language="Oracle"
Column
{
  notes=""
  scope="Private"
  container=""
  alias=""
  lowerbound="1"
  name="email"
  default=""
  upperbound="1"
  type="CHAR"
}
  PrimaryKey
  {
    Column
    {
      name="arveID"
      type="Integer"
```

Joonis 4. Koodilõik vahefailist, elemendi Table kirjeldus

3.2. Lisamoodulid (Add-Ins)

Enterprise Architect-is on ette nähtud laiad võimalused selle kohandamiseks vastavalt ettevõtte või valdkonna vajadustele. Üks sellistest võimalustest on *Add-In*-id e lisamoodulid. Need omavad kõiki EA *Automation Interface*-i kasutavate eraldiseisvate programmide eeliseid [27] ja pakuvad hulga uusi valikuid, nagu näiteks menüüde ja kontekstmenüüde loomine, kasutajaliidese sündmustele reageerimine, lisainformatsiooni saamine hetkel avatud mudeli täpsemaks töötlemiseks jne [28]. Enterprise Architect-i ametlikul kodulehel on loetletud rohkem kui 30 kolmandate tootjate poolt valmistatud lisamoodulit, mis on jagatud kasutusotstarbe järgi kaheksaks grupiks [29]. Kuigi ametlik dokumentatsioon puudutab seda teemat ainult möödaminnes, siis lisamoodulitele leidub veel üks kasutusviis, mida käesolev töö edukalt rakendab. Nimelt saab spetsiaalse keelekonstruktsiooni abil lisamooduli meetodeid kutsuda välja teisendusmallidest. Kuna lisamooduleid kirjutatakse „traditsioonilistes“ programmeerimiskeeltes, aitab see leevendada enamust teisenduskeele piirangutest. Lisamooduli meetod realiseerib sel juhul mingit kontekstile vastavat äriloogikat ja tagastab koostatud tekstijupi, mis sobib kas otseseks kasutamiseks genereeritavas vahefailis või edasiseks töötlemiseks teisendusmallis. Siinkohal võiks lisada, et ülalkirjeldatud liidestus teisendusmallide ja lisamoodulite vahel ei ole dokumentatsioonis eriti kirjeldatud. Autor leidis ainult ühte arutelu [30], kus seda mainitakse. Ühtegi konkreetset näidet Google otsinguga leida ei õnnestunud, seega töös pakutud lahendus ei toetu mingitele väljakujunenud mustritele ja seda saaks optimeerida.

Oma sisu poolest on EA lisamoodulid ActiveX/COM objektid, mis toetavad IDispatch liidest. Neid luuakse *in-process* DLL komponentide vormis ja registreeritakse süsteemis. Registreerimise protseduur sõltub sellest, kuidas kirjutati / kompileeriti DLLi – kui Win32 DLL või .NET DLL [31]. Selleks, et Enterprise Architect saaks uuest DLL-ist teada ja seda kasutada, luuakse registris spetsiaalne võtit, mille väärtuseks on DLL-projekti nimi ja avalikke meetodeid sisaldava klassi nimi. Kuidas see kõik funktsioneerib .NET DLL abil loodud lisamooduli juhul, on hästi illustreeritud ühes leitud artiklis komponentdiagrammina [32].

Lisamoodulite arendamiseks soovitab EA dokumentatsioon kasutada selliseid arendusvahendeid nagu Borland Delphi, Microsoft Visual Basic või Microsoft Visual Studio .NET [33]. Antud töös loodud lisamoodul on tehtud kahes vahendis: Microsoft Visual C# 2010 Express ja SharpDevelop 4.3.

4. Enterprise Architect-i andmebaaside arendamisega seotud mudeliteisenduste täiustamine

Enterprise Architect 12 pakub kolm andmebaaside arendamisega seotud sisseehitatud teisendustüüpi, mis on mõeldud erinevat tüüpi andmemudelite konverteerimiseks [34]:

- **Data Definition Language**. Teisendab detailse kontseptuaalse andmemudeli, mis on esitatud UMLi klassidiagrammina, füüsilise disaini täpsusega andmemudeliks.
- **Entity Relationship Diagram to Data Model**. Teisendab detailse kontseptuaalse andmemudeli, mis on esitatud ERD-diagrammina kasutades Chen'i notatsiooni [35], füüsilise disaini täpsusega andmemudeliks.
- **Data Model to Entity Relationship Diagram**. Eelmise teisendustüübi vastupidine versioon. Teisendab füüsilise andmemudeli detailseks kontseptuaalseks andmemudeliks, esitades selle ERD-diagrammina.

Vastavalt tööülesandele parandatakse käesolevas töös teisendustüüpi *Data Definition Language* kuuluvaid teisendusi (edaspidi *DDL teisendused* või lihtsalt *teisendused*), sest TTÜs kasutatakse andmebaaside õppeainetes andmemudelite loomiseks UMLi klassidiagramme [36]. Nagu nimigi vihjab, siis DDL teisenduste rakendamisel algmudeli suhtes saadakse füüsiline andmemudel, mis on valmis DDL lausete genereerimiseks. Selle oluliseks eelduseks on vaikimisi andmebaasisüsteemi määramine enne teisendamise käivitamist [22]. Teisendamise käigus leiab Enterprise Architect andmetüüpide tabelist vastavused kontseptuaalse mudeli üldiste andmetüüpide ja sihtsüsteemi konkreetsete andmetüüpide vahel ning kasutab füüsilise andmemudeli loomisel just selle andmebaasisüsteemi poolt toetavate tüüpide nimetusi. See aitab vähendada käsitsi parandamise vajadust disaini mudeli ülevaatamisel ja tõstab seeläbi genereeritud DDL SQL koodi kvaliteeti.

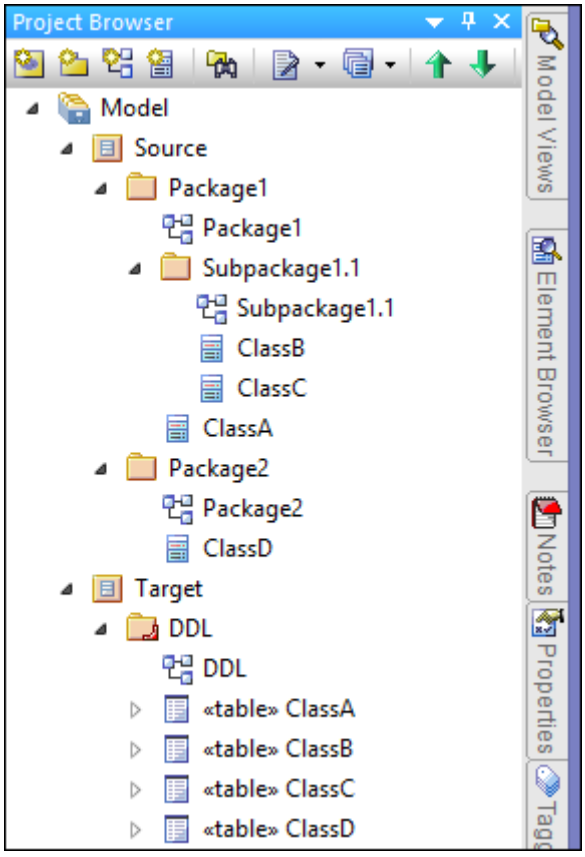
4.1. Olukord enne täienduste lisamist

Järgnevalt antakse ülevaade sisseehitatud DDL teisendustest ja nende võimalustest. Ülevaate koostamiseks kasutati nii ametlikku Enterprise Architect-i dokumentatsiooni [37], proovimudeleid, kui ka teisenduste koodi uurimist, mis aitas saavutada maksimaalselt usaldusväärset ettekujutust olemasolevatest teisendustest.

Algmudeli klassidiagrammi elementide teisendamine sihtmudeli elementideks on suhteliselt sirgjooneline. Iga UMLi klass, millele ei ole määratud loetelu (ingl *enumeration*) stereotüüpi,

teiseneb tabeliks, mille nimi on sama kui klassil. Iga klassi atribuut saab tabeli veeruks. Kõik atribuudid töödeldakse ühesuguselt: veeru nimi on sama kui atribuudi nimi, veeru tüübiks on atribuudi üldisele tüübile vastavusse määratud andmebaasisüsteemi andmetüüp. Iga klassi alusel tekkinud tabeli (v.a. n-aarse seose alusel loodava tabeli) jaoks luuakse primaarvõti. Primaarvõtmena kasutatakse igal pool surrogaatvõtmeid. Tabeli primaarvõtme veeru nimi tuletatakse vastava klassi nimest – nimi kirjutatakse camelCase stiilis ja lisatakse sellele sufiks *–ID* (näiteks, *OrderItems -> orderItemsID*).

Teisendamisel loob Enterprise Architect sihtpaketis (kataloogis) alampaketi nimega *DDL* ja paneb sinna kõik genereeritud tabelite kirjeldused (vt Joonis 5). Kõigi tabelite esitus on koondatud ühte klassidiagrammi nimega *DDL*.



Joonis 5. Projekti algmudeli (Source) ja teisendatud mudeli (Target) kataloogipuud

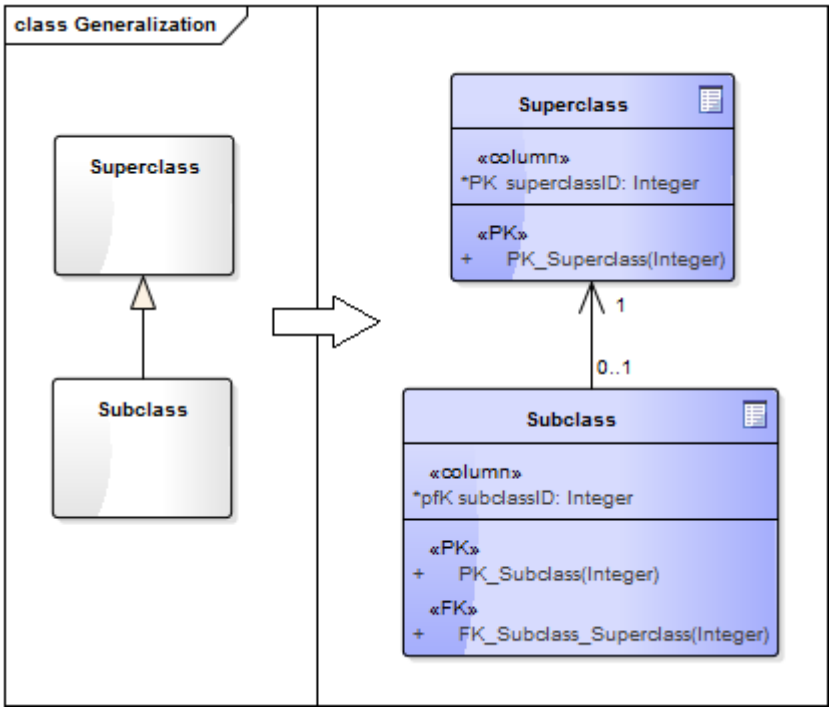
Klasside vaheliste seoste teisendamine toimub iga seoses osaleva klassi jaoks eraldi. See tähendab, et iga seost teisendatakse kaks korda [38]. Tulemuseks on, et vahekeele failis ilmub iga seose kohta kaks kirjet. Kuna need on identsed, siis see ei tekita probleemi: Enterprise Architect oskab sellised kirjed omavahel seostada ja näidata neid teisendus tulemusena loodud mudelis kui ühte välisvõtme kitsendust.

Kehtiv seoste teisendamise toimemehhanism vajab eraldi põhjalikku käsitlemist. Jagame seoste teisendamise realisatsiooni seosetüüpide kaupa ja vaatleme iga seosetüüpi teisendamisprintsiipi alljärgnevates alapeatükkides. Lisame ka ettepanekud, kuidas saaks neid teisendusi parandada. Soovitatud teisendusreeglid pärinevad raamatust „*Andmebaaside projekteerimine*“ [39] ning samuti autori enda ja juhendaja tähelepanekutest. Enamus parandamise ettepanekuid oli antud töö praktilises osas realiseeritud. Neid ja muid tehtud muudatusi käsitleb alapeatükk 4.2.

4.1.1. Üldistusseos

Iga üldistusseoses osaleva üla- ja alamtüübi alusel luuakse füüsilise disaini mudelis eraldi tabel. Kõikidesse tabelitesse genereeritakse primaarvõtmed, mis on surrogaatvõtmed. Iga alamtüübi tabel on seotud ülatüübi tabeliga välisvõtme kaudu, mis hõlmab alamtüübi tabeli primaarvõtme veergu. Määratud võimsustikud kehtestavad reegli, et igale ülatüübi olemile *UO* vastab null või üks antud alamtüübi olem *AO* ja iga alamtüübi olemi *AO* jaoks leidub täpselt üks ülatüübi olem *UO*.

Joonis 6 esitatakse näide üldistusseost sisaldavast mudelist ja selle teisendamise tulemusest.



Joonis 6. Üldistusseose teisendamine

Olemasoleva teisenduse suurimaks puudujäägiks võib nimetada liiga pealiskaudset lähenemist probleemile. Üldistusseost realiseerivate tabelite soovituslik disain sõltub suurimal määral üldistusseosega seotud kitsendustest, mis võivad esineda neljas kombinatsioonis [40].

Enterprise Architect küll võimaldab määrata kontseptuaalses andmemudelis üldistuste hulka [41] ning sellega seotud osaluskohustuse (ingl *covering constraint*) ja kuuluvuse (ingl *disjoint constraint*) kitsendusi, aga sisseehitatud DDL teisendused ei erista neid kitsendusi ega jõusta neid tabelite disaini kaudu.

Ettepanekuks on realiseerida lisaks olemasolevale üldisele üldistusseose teisendusele variant, mis jõustab osaluskohustuse {Mandatory} ja kuuluvuse {Or} kitsenduste kombinatsiooni. Osaluskohustus {Mandatory} tähendab, et iga ülatüübi olemi kohta leidub vähemalt üks vastav suvalise alamtüübi olem. Kuuluvus {Or} tähendab, et iga ülatüübi olem võib olla seotud ainult ühe konkreetse alamtüübiga. Teisisõnu, niisuguse kitsenduste kombinatsiooni tähendus on selline, et iga ülatüübi olem peab kuuluma mingi ühte konkreetsesse alamtüüpi. Autori hinnangul on see kõige enamlevinum üldistusseose variant.

Uus teisendus võiks tuvastada üldistuste hulgaga seotud kitsendust {Mandatory; Or} ja teostada järgnevat:

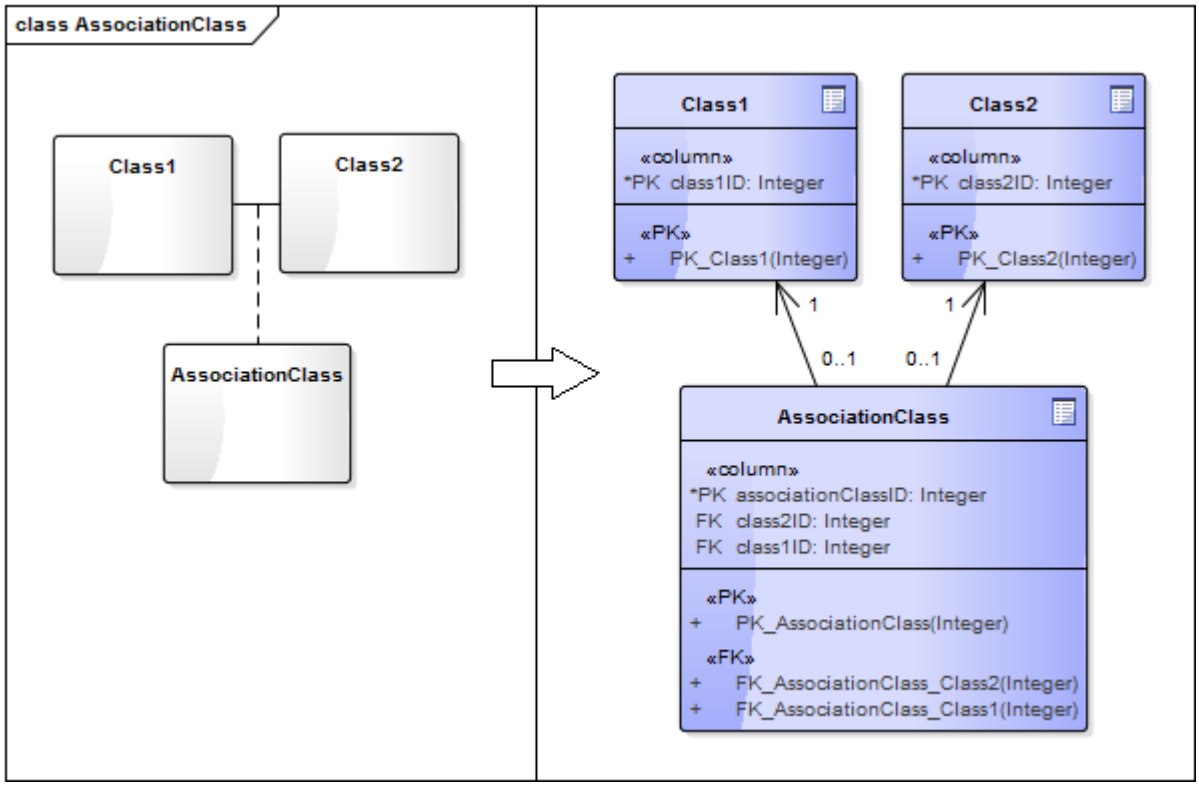
- luua ainult alamtüüpidele vastavad tabelid;
- luua igas tabelis ülatüübi atribuutidele vastavad veerud;
- luua igas tabelis vastava alamtüübi atribuutidele vastavad veerud;
- määrata iga tabeli primaarvõtmeks veerud, mis vastavad ülatüüpi unikaalselt identifitseerivale atribuutide hulgale;
- määrata iga tabeli alternatiivvõtme(te)ks veerud, mis vastavad alamtüüpi unikaalselt identifitseeriva(te)le atribuutide hulga(de)le;
- realiseerida ülatüübi seoste *copy-down inheritance*, ehk lisada vajadusel igasse tabelisse välisvõtme veerud ja kitsendused, mis tulenevad ülatüübi seostest teiste olemitüüpidega;
- arvestada ka mitmetasemeliste pärimisseostega.

4.1.2. Sidemeklassina modelleeritud M:N (mitu-mitmele) seos

Iga seoses osaleva olemitüübi alusel luuakse eraldi tabel. Kõikidesse tabelitesse, sealhulgas sidemeklassi tabelisse, genereeritakse surrogaatvõtmed. Sidemeklassi tabelis on kaks välisvõtit, mille abil luuakse seosed teiste tabelitega. Määratud võimsustikud kirjeldavad reegli, et igale mitu-mitmele seoses osalevale olemile *O* vastab null või üks sidemeklassi olemit *SO* ja iga

sidemeklassi olemi *SO* jaoks leidub täpselt üks olem *O*. Samas tabelitega seotud kitsendused lubaksid andmebaasis iga *O* kohta null või rohkem sidemeklassi olemit.

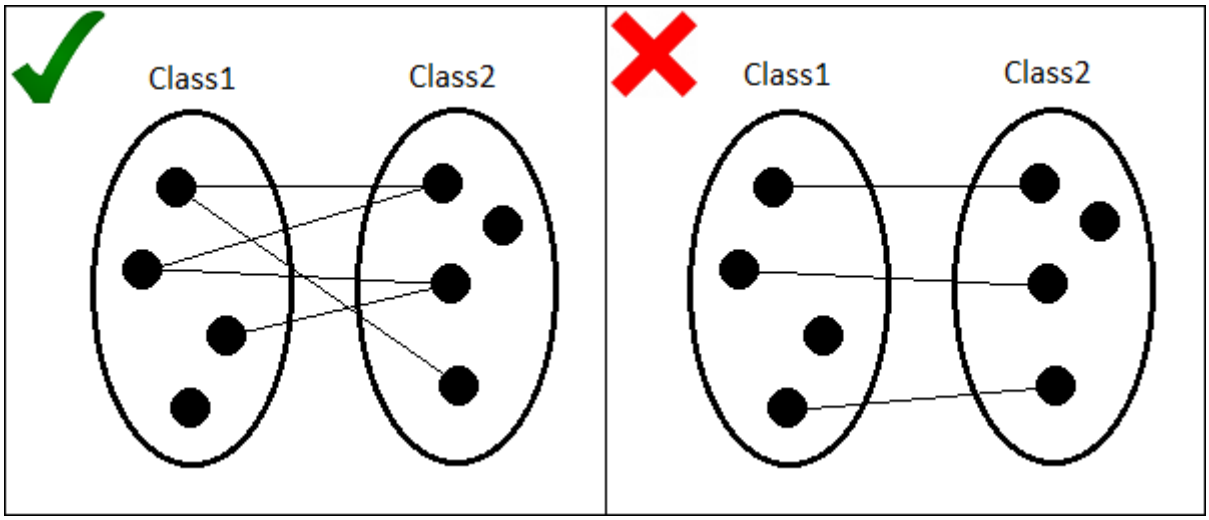
Joonis 7 esitatakse näide sidemeklassi sisaldavast mudelist ja selle teisendamise tulemusest.



Joonis 7. Sidemeklassi teisendamine

Olemasoleva teisenduse peamiseks probleemiks on see, et sidemeklassi tabeli (edaspidi ka lihtsalt sidetabeli) disain ei jõusta sidemeklassi põhiomaduseks olevat kitsendust, mille kohaselt iga *class1ID* ja *class2ID* väärtuste kombinatsioon selles tabelis peab olema unikaalne. Selle unikaalsuse kitsenduse peab inimkasutajast disainer tabeli kirjeldusse käsitsi juurde lisama. Vaadates võimsustikke, võib saadud teisenduse tulemust interpreteerida hoopis sedamoodi, et sidetabelis iga *class1ID* väärtust võib kasutada kombinatsioonis mingi *class2ID* väärtusega maksimaalselt üks kord – ja vastupidi, iga *class2ID* väärtus võib esineda kombinatsioonis mingi *class1ID* väärtusega null või üks kord. Pealegi on määratud võimsustikud vastuolus sidetabeli välisvõtmete poolt hõlmatud veergude kitsendustega. Need veerud peaksid sel juhul olema kohustuslikud (NOT NULL) ja unikaalsed (UNIQUE).

Joonis 8 illustreerib õiget sidemeklassi abil realiseeruvat binaarset seost ja seda seost, mida kajastab praeguse teisendusega saadud mudel.



Joonis 8. Sidemeklassiga realiseeruv binaarne seos (vasakul) ja valele teisenduste realisatsioonile vastav seos (paremal)

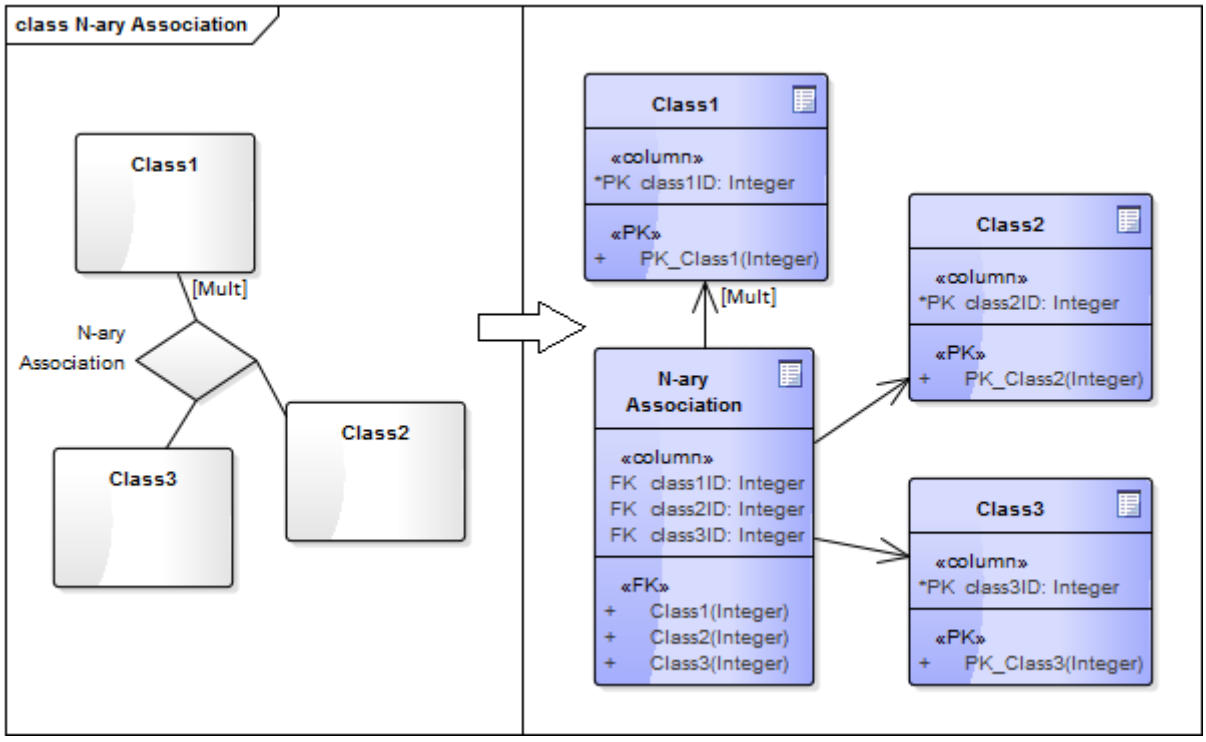
Ettepanekuks on parandada teisenduse niimoodi, et see ei looks sideklassis surrogaatvõtit, vaid määraks selle asemel sideklassi primaarvõtmeks liitvõtme, mis hõlmab mõlema välisvõtme veerge. See aitab tagada, et iga *class1ID* ja *class2ID* väärtuste kombinatsioon on unikaalne. Samas võib iga *class1ID* (*class2ID*) väärtus esineda kombinatsioonis erinevate *class2ID* (*class1ID*) väärtustega rohkem kui üks kord.

Samuti peaks parandatud teisendus määrama õige võimsustiku: 0..1 asemel 0..*.

4.1.3. N-aarne seos

Iga seoses osaleva olemitüübi alusel luuakse eraldi tabel. Surrogaatvõtmed luuakse kõikidesse tabelitesse, v.a seosetüübi esitava olemitüübi alusel loodud tabelisse (edaspidi nimetame seda *seosetabeliks*). Seosetabelis on N välisvõtit, mille abil luuakse seosed teiste tabelitega. Kui algmudelil oli määratud võimsustik seose otsas, mis on kaugemal seosetüübi esitavast olemitüübist (vt Joonis 9), siis see kopeeritakse sihtmudelisse.

Joonis 9 esitatakse näide n-aarset seost sisaldavast mudelist ja selle teisendamise tulemusest.



Joonis 9. N-aarse seose teisendamine

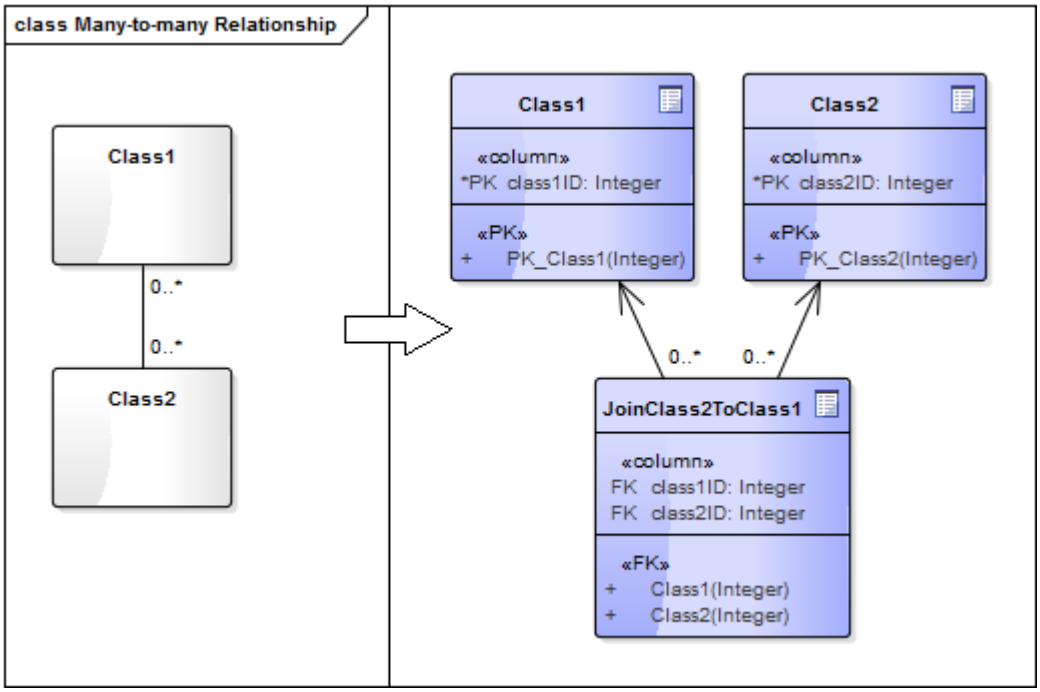
See teisendus on eelmistest mõnevõrra parem, sest teisenduse autorid otsustasid mitte luua seosetabelis surrogaatvõtit. Üldiselt sellise seose realiseerimine eeldab, et lõppotsus primaarvõtme kohta tehakse vastavalt seose semantikale loogilise disaini etapis. Seosetabeli primaarvõti peab igal juhul hõlmama kõiki välisvõtmete veerge, aga puudub kitsendus, et iga välisvõtmete väärtuste kombinatsioon peab seosetabelis olema unikaalne (nagu sidetabelis, mida kirjeldati eelmises alapeatükis).

Ettepanekuks on täiendada teisenduse niiviisi, et see lisaks seosetabelisse primaarvõtme, mis hõlmaks kõik N välisvõtme veergu. Vajadusel võib disainer täpsustada pärast teisendamist primaarvõtme käsitsi. Võiks ka määrata seosetabelist väljuvate seoste juures võimsustiku 0..*.

4.1.4. M:N (mitu-mitmele) seos

Mõlema seoses osaleva olemitüübi põhjal luuakse eraldi tabel, koos surrogaatvõtmetega. Lisaks luuakse vahetabel, mis seob need tabelid omavahel välisvõtmete kaudu. Uue vahetabeli nimi on kujul Join*T1*To*T2*, kus *T1* ja *T2* on vastavalt esimese ja teise tabeli nimed. Primaarvõtme vahetabelisse ei looda. Võimsustikud kopeeritakse muutmata kujul.

Joonis 10 esitatakse näide mitu-mitmele seost sisaldavast mudelist ja selle teisendamise tulemusest.



Joonis 10. M:N seose teisendamine

Nagu n-aarse seose teisenduses, ei looda siduvas tabelis surrogaatvõtit. Siiski oleks asjakohane luua teisendamise käigus primaarvõti, mis hõlmaks mõlema välisvõtme veerge. Mitu-mitmele seos, mis on modelleeritud sel viisil, ei jäta palju ruumi interpreteerimiseks. Vahetabel peab jõustama kitsenduse, et iga välisvõtmete väärtuste kombinatsioon peab olema unikaalne ja seda saab tagada ülalkirjeldatud primaarvõtmega.

Parandatud teisenduse versioon võiks lisada vahetabelisse primaarvõtme, mis hõlmaks mõlema välisvõtme kõik veerud.

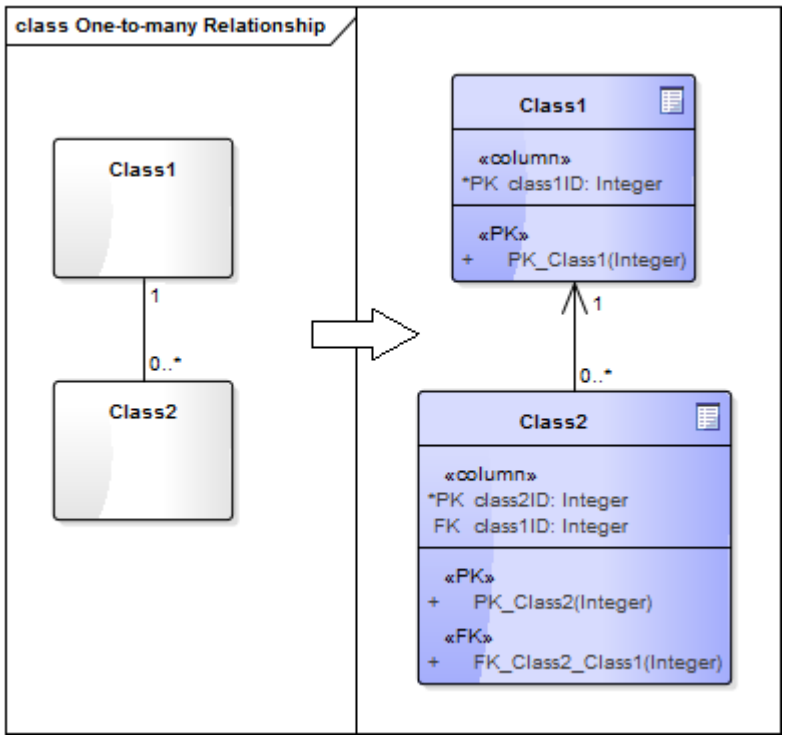
4.1.5. 1:M (üks-mitmele) seos

Seose teisendamisel käsitleb Enterprise Architect algmudelil määratud võimsustikke järgnevalt:

- 1, 0, 0..1 ja mittemääratud võimsustiku võrdsustatakse võimsustikuga 1..1;
- ülejäänud võimsustikud, s. h. *, 0..*, 1.., 1..*, võrdsustatakse võimsustikuga 0..*.

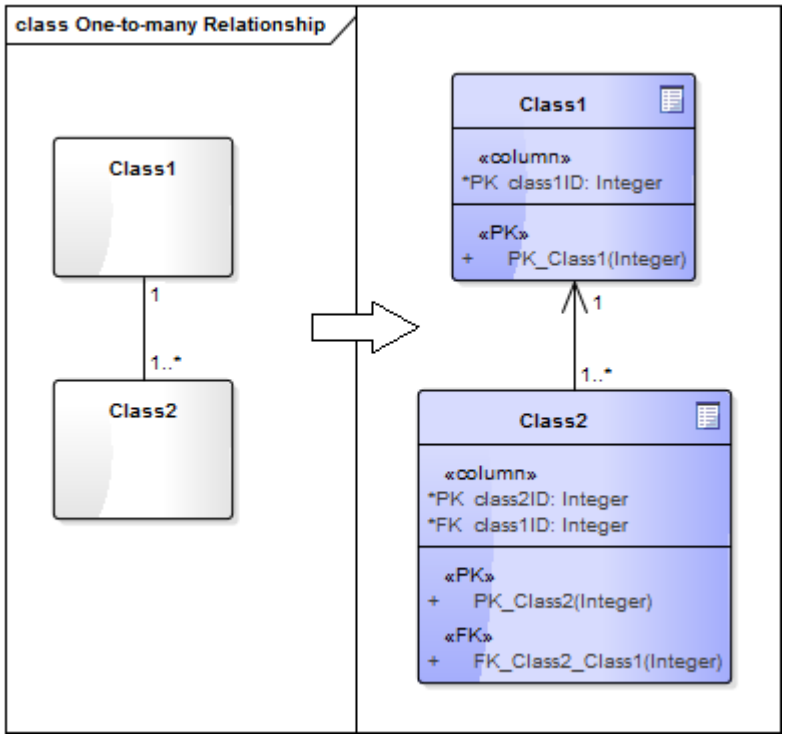
Mõlema seoses osaleva olemitüübi põhjal luuakse eraldi tabel, koos surrogaatvõtmetega. Tabelid on seotud välisvõtme kaudu. Olgu *OT* olemitüüp, mille võimsustik antud seosetüübi kontekstis on võrdsustatav võimsustikuga 1..1. Välisvõti lisatakse tabelisse, mis on loodud olemitüübi *OT* alusel.

Joonis 11 esitatakse näide üks-mitmele seost sisaldavast mudelist ja selle teisendamise tulemusest.



Joonis 11. 1:M seose teisendamine

Näitena toonud teisenduse ainsaks probleemiks on, et tabeli *Class2* välisvõtme veerule *class1ID* pole lisatud NOT NULL kitsendust (edaspidi ka *kohustuslikkuse kitsendus*). Kuna olemitüübi *Class2* osaluskohustus seosetüübis on kohustuslik, siis peab ka vastava tabeli välisvõtme veerg *class1ID* olema kohustuslik veerg. Siinkohal tuleb mainida, et testimine erinevate võimsustike väärtustega tõi välja veel ühe tähelepaneku, mis on näidatud Joonis 12.



Joonis 12. 1:M seose teisendamine ja NOT NULL kitsendus

Nimelt, teatud juhul Enterprise Architect ikkagi lisas tabeli *Class2* välisvõtme veerule *class1ID* kohustuslikkuse kitsenduse. Sama toimub ka siis, kui asendada selle näite algmudelil võimsustik 1..* võimsustikuga 1.. . Ükski teise kombinatsiooni puhul kitsendust ei lisata. See tähendab, et veerule *class1ID* kohustuslikkuse kitsenduse deklareerimisel teeb Enterprise Architect otsuse lähtudes valedest eeldustest, sest selle kitsenduse lisamine peab sõltuma nii olemitüübi *Class2*, kui ka olemitüübi *Class1*, osaluskohustusest. Kuigi taoline otsustusviis on ebaloogiline, siis ei tekita see probleemi, sest kuna olemitüübi *Class2* osaluskohustus on antud juhul kohustuslik, siis NOT NULL kitsendus peabki olema lisatud. Tasub ka ära mainida, et koodi uurimisel selgus, et vaadeldava kitsenduse deklareerimine ei ole teisendusmallides ära määratud, vaid seda teeb teisendusmootor ise. Seega pole ka võimalik seda käitumist mõjutada.

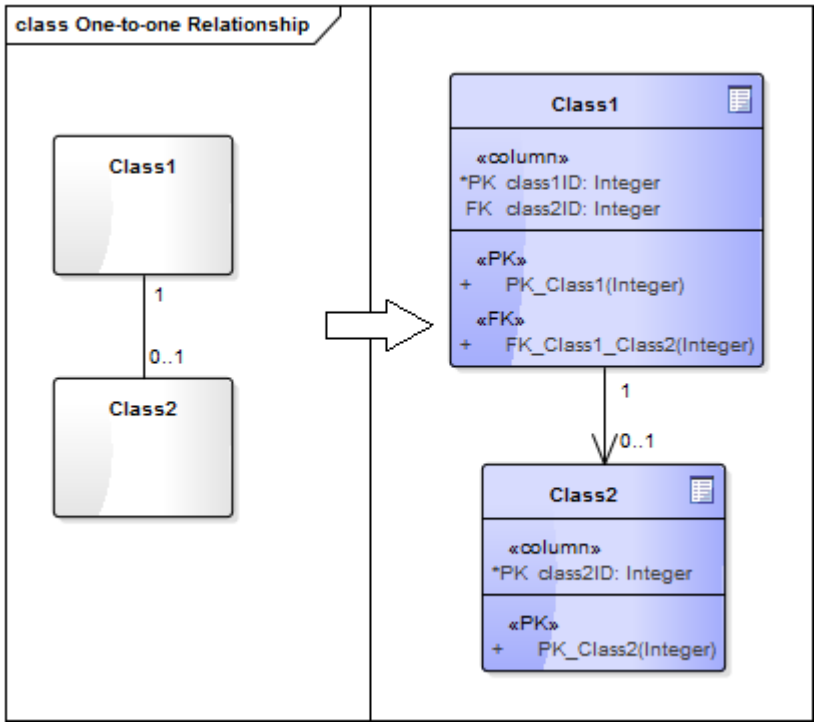
Võttes eelneva kokku, siis teisenduse võimalik parandamine seisneb NOT NULL kitsenduse lisamises veergudele, mis on hõlmatud seost realiseeriva välisvõtme poolt, kui vastava olemitüübi osaluskohustus seosetüübis on kohustuslik.

4.1.6. 1:1 (üks-ühele) seos

Mõlema seoses osaleva olemitüübi põhjal luuakse eraldi tabel, koos surrogaatvõtmetega. Tabelid on seotud välisvõtme kaudu. Tabeli valik, kuhu paigutada välisvõti, sõltub sellest, mis

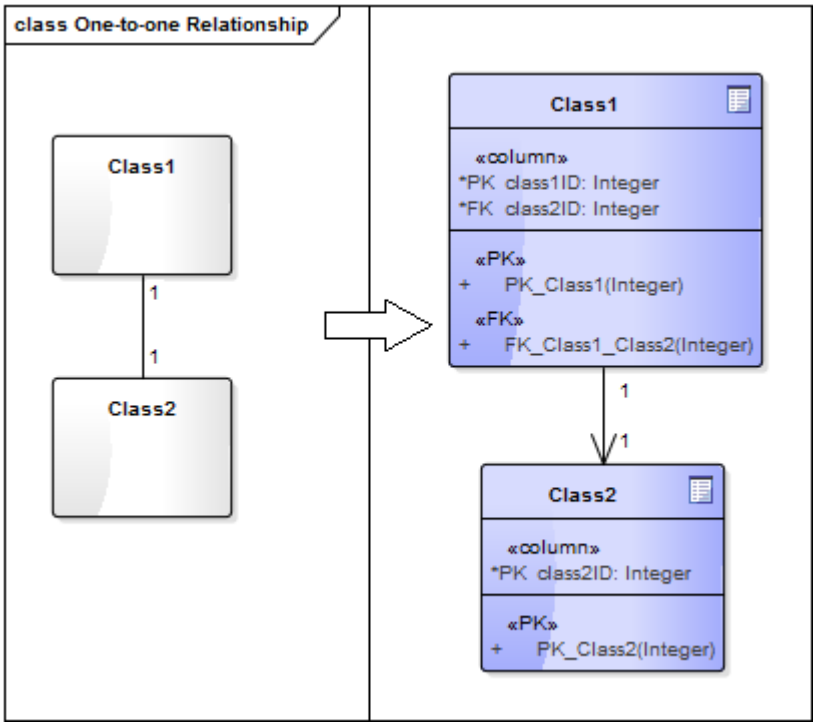
suunas on tõmmatud algmudelil seos kahe klassi vahel. Välisvõti paigutatakse alati tabelisse, mis vastab sellele klassile, mis asub seose *destination* otsa pool.

Joonis 13 esitatakse näide üks-ühele seost sisaldavast mudelist ja selle teisendamise tulemusest. Teisendamine toimub identselt nii 1 / 0..1, kui ka 0..1 / 0..1 võimsustikega seoste jaoks.



Joonis 13. 1:1 seose teisendamine

1 / 1 seose puhul (vt Joonis 14) rakendub jälle loogika, mida kirjeldati eelnevas alapeatükis.



Joonis 14. 1:1 seose teisendamine ja NOT NULL kitsendus

Olemasolev teisenduse teostus on puudulik. Iga erinevat tüüpi seose (1 / 0..1, 0..1 / 0..1, 1 / 1) teisendamine vajab oma strateegiat. Vaatleme neid lähemalt ja võrdleme strateegiatega, mida kasutavad sisseehitatud teisendused.

1 / 0..1 seos: Mõlema seoses osaleva olemitüübi põhjal peab looma eraldi tabeli. Tabelid seostatakse välisvõtme kaudu. Teisenduse põhiline viga on selles, kuidas valitakse tabel, kuhu lisatakse välisvõti. Välisvõtme veerud ja kitsendus peavad olema paigutatud tabelisse, mis on loodud selle olemitüübi põhjal, mille osaluskohustus vaadeldavas seosetüübis on kohustuslik (ehk Joonis 13 illustreeritud näite puhul tabelisse *Class2*). Lisaks tuleb määrata välisvõtmega hõlmatud veerud alternatiivvõtmeks, deklareerides vastava unikaalsuse (UNIQUE) ja kohustuslikkuse (NOT NULL) kitsenduse.

0..1 / 0..1 seos: Mõlema seoses osaleva olemitüübi põhjal peab looma eraldi tabeli. Tabelid seostatakse välisvõtme kaudu. Küsimus on jälle selles, mis tabelisse peab see välisvõti olema lisatud. Vastus sellele küsimusele sõltub seose semantikast: välisvõtme veerud ja kitsendus peavad olema paigutatud tabelisse, millesse lisatakse ridu hiljem. Järelikult, lõppotsus on disaineri käes. Välisvõtmega hõlmatud veergudele tuleb deklareerida unikaalsuse kitsendus.

1 / 1 seos: Sisseehitatud teisendus loob ka sel juhul kaks eraldi tabelit. Tegelikult tuleks luua üks ühine tabel, mis sisaldab mõlema olemitüübi atribuutidele vastavaid veerge. Põhiline

küsimus on selles, kuidas määrata selle tabeli primaarvõtit. Selle otsuse tegemise peab jätma disainerile. Disainer peab hindama, milline seostatud olemitüüpide unikaalsetest identifikaatoritest sobib kõige rohkem ühendatud tabeli primaarvõtmeks ja kinnitama oma valiku. Mittevalitud unikaalsetest identifikaatoritest saavad tabeli alternatiivvõtmed.

Teisendused tuleks viia vastavusse eeltoodud strateegiatega. Uus lahendus eeldab samuti kasutajalt sisendit küsimist teisendamise käigus.

4.2. Mudeliteisenduste täiendused

Käesolevas töös on parandatud olemasolevaid teisendusi ning lisatud rida uusi teisendusi. Seoste teisendamise realisatsiooni parandamisel on järgitud alapeatükis 4.1 kirjeldatud ettepanekuid. Mõnda algselt plaanitud parandustest ei õnnestunud teostada tehniliste piirangute tõttu. Seda käsitletakse detailselt alapeatükis 4.3.3. Kõik tehtud täiendused on klassifitseeritud erinevate aspektide järgi ja esitatakse järgnevates alapeatükkides. Ühtlasi tähendab sellise täienduse olemasolu järgnevas nimekirjas, et algne teisendus sellise asjaga hakkama ei saanud.

4.2.1. Primaarvõtmed ja alternatiivvõtmed (PK)

PK1: Kui algmudeli klassidiagrammis määratakse mingi atribuut või atribuutide kogum unikaalseks identifikaatoriks (määrates atribuutide seadete aknas neile stereotüübi <<*id*>>), siis tabeli kirjelduses tekib selle alusel primaarvõti või alternatiivvõti. Viimane tekitatakse juhul, kui tabel luuakse üldistusseoses osaleva alamtüübi põhjal (vt alapeatükk 4.1.1). Vastavalt valitud atribuutide arvule võib primaarvõti olla lihtvõti või liitvõti. Surrogaatvõtit kasutakse tabeli kirjeldamisel vaid juhul, kui mitte ükski atribuut pole määratud unikaalseks identifikaatoriks.

PK2: Sidemeklassi alusel tehtud tabelis luuakse õige primaarvõti (vt alapeatükk 4.1.2).

PK3: N-aarse seose seosetabelis luuakse õige primaarvõti (vt alapeatükk 4.1.3).

PK4: Mitu-mitmele seose vahetabelis luuakse õige primaarvõti (vt alapeatükk 4.1.4).

4.2.2. Veerutaseme kitsendused (CLC)

CLC1: Kui algmudeli klassidiagrammis määratakse mingi atribuut unikaalseid väärtusi sisaldavaks atribuudiks (täites märkeruudu *isID* atribuudi seadete aknas), siis vastava veeru kirjelduses tekib üldjuhul selle alusel UNIQUE kitsendus. Kitsendust ei looda, kui sama atribuut on märgitud ka unikaalseks identifikaatoriks (parandus PK1). Vastasel juhul tekiks olukord, kus samal veerul on nii PRIMARY KEY kui UNIQUE kitsendus. See ei anna midagi juurde, sest primaarvõti tähendab unikaalsust, kuid samas võib tekitada probleeme, kui mõlema kitsenduse kohta luuakse andmebaasisüsteemi poolt automaatselt eraldi indeks.

CLC2: Kui algmudeli klassidiagrammis pole atribuudil määratud võimsustikku või võimsustik on suurem kui 0 (atribuudi seadete aknas on võimsustiku väärtuse alampiir >0), siis vastava veeru kirjelduses tekib selle alusel NOT NULL kitsendus.

CLC3: Kui algmudeli klassidiagrammis on atribuudil defineeritud algväärtus, siis vastava veeru kirjelduses tekib selle alusel DEFAULT klausel lähtemudelis määratud väärtusega.

4.2.3. Välisvõtmed, kompenseerivad tegevused ja seoste võimsustikud (FK)

FK1: Kui välisvõti viitab sisulise tähendusega primaarvõtmele, siis välisvõtme kirjelduses tekib kompenseeriv tegevus ON UPDATE CASCADE.

Märkus: Enterprise Architect ei esita seda kompenseerivat tegevust sihtmudelis juhul, kui teisendus kasutas sihtsüsteemiks vaikimisi andmebaasisüsteemi, mis seda ei toeta (nt Oracle).

FK2: Kompositsiooniseose alusel loodavate välisvõtmete puhul tekib välisvõtme kirjelduses kompenseeriv tegevus ON DELETE CASCADE.

FK3: Sidemeklassi teisendamisel saadud mudelis määratakse vastavatele seostele õiged võimsustikud (vt alapeatükk 4.1.2).

FK4: N-aarse seose teisendamisel saadud mudelis määratakse vastavatele seostele õiged võimsustikud (vt alapeatükk 4.1.3).

FK5: 1:M seose teisendamisel loodava välisvõtme veeru/veergude kirjelduses on vastavalt vajadusele deklareeritud kitsendus NOT NULL (vt alapeatükk 4.1.5).

FK6: 1 / 0..1 seose teisendamisel loodavad tabelite kirjeldused kajastavad õiget otsust välisvõtme paigutuse kohta (vt alapeatükk 4.1.6). Kui tegemist on lihtvõtmega, siis deklareeritakse vastava veeru kirjelduses UNIQUE ja NOT NULL kitsendus. Liitvõtme puhul sellega hõlmatud veergude kirjeldustes tekivad ainult NOT NULL kitsendused. See on seotud Enterprise Architect-i piiranguga: vahekeeles ei ole võimalik kirjeldada tabelitaseme UNIQUE kitsendust (hõlmab rohkem kui ühte veergu). Täpsemalt on sellest kirjutatud alapeatükis 4.3.3.

FK7: 0..1 / 0..1 seose teisendamisel küsitakse kasutajalt, mis tabelisse peab lisama välisvõtme. Lihtvõtme puhul deklareeritakse vastava veeru kirjelduses UNIQUE kitsendus. Kui välisvõti on liitvõti, siis eespool kirjeldatud põhjusel kitsendust ei deklareerita.

FK8: 1 / 1 seose teisendamisel kasutatakse täpselt sama loogikat, kui 0..1 / 0..1 seose korral. Erinevalt alapeatükis 4.1.6 kirjeldatud ettepanekust, otsustati loobuda ühise tabeli loomisest realiseerimise keerukuse tõttu. Selle otsuse tagamaad on käsitletud alapeatükis 4.3.3.

4.2.4. Nimetused (N)

N1: Surrogaatvõtme veeru nimi tuletatakse vastava klassi nimest. Nimi kirjutatakse väiketähtedena ja lisatakse sellele sufiks *_id* (näiteks, klassile *OrderItems* vastavas tabelis võib tekkida primaarvõti nimega *orderitems_id*).

N2: Primaarvõtme ja välisvõtme kitsenduste nimed on ühtlustatud. Neid kitsendusi nimetatakse vastavalt mallidele, mida saab seadistada siin: *Tools – Options – Source Code Engineering – Code Editors – DDL Name Templates*. Erandiks on välisvõtmed, mis luuakse juhul, kui on määratud vastava olemitüübi roll 1:M seosetüübi kontekstis; need välisvõtme kitsendused saavad nime kujul FK_*RolliNimi*.

N3: Mitu-mitmele seose vahetabeli uus nimemall on kujul *T1_T2*, kus *T1* ja *T2* on vastavalt esimese ja teise tabeli nimed.

N4: Tabelite, veerude ja kitsenduste nimetamisel asendatakse vastavates klasside ja atribuutide nimedes eesti tähestiku diakriitiliste märkidega tähed lähimate alustähtedega (nt. *ä -> a, Õ -> O*). Erandiks on tähed *Ü* ja *ü*, mida asendatakse vastavalt tähtedega *Y* ja *y*.

N5: Kõik märgid (sh tühikud), mida vastavalt SQL standardile ei või kasutada identifikaatorites [42], asendatakse tabelite, veerude ja kitsenduste nimede tuletamisel alakriipsuga (_).

N6: Kui klassi nimi algab numbriga, siis vastava tabeli nime tuletamisel pannakse selle ette prefiks *t*. Kui atribuudi nimi algab numbriga, siis vastava veeru nime tuletamisel pannakse selle ette prefiks *c*.

N7: Kui klassi või atribuudi nimi langeb kokku mõne SQLi reserveeritud sõnaga, siis vastava tabeli või veeru nime tuletamisel pannakse selle lõppu alakriips. Nimekiri teisenduse poolt arvestatavatest võtmesõnadest esitatakse Lisa 1. Nimekirja koostamisel on autor arvestanud sellega, milliseid võtmesõnu võidakse kontseptuaalses andmemudelis kasutada.

N8: Tabeli, veeru või kitsenduse nime maksimaalne lubatud pikkus on 128 märki, mis vastab SQL standardile [43]. Kui tuletatud nimi on pikem, siis seda lühendatakse.

4.2.5. Muu (G)

G1: Kui üldistusseose korral määratakse ülaklass abstraktseks klassiks (täites märkeruudu *Abstract* klassi seadete aknas), siis tõlgendatakse seda nii nagu kuuluksid kõik sellega seotud alamklassid üldistuste hulka, millega on seotud kitsendus {Mandatory; Or}. Taolise üldistusseose teisendamine toimub vastavalt alapeatükis 4.1.1 kirjeldatud ettepanekutele. Tasub mainida, et õigem oleks teha kindlaks kitsenduse tüüp, lugedes algmudelist vaadeldava üldistuseoste hulga omadusi, mitte kasutades klassi abstraktsuse omadust. Kahjuks ilmnes aga, et Enterprise Architect-i praegusel versioonil puudub toetus vajalike atribuutide mudelist teada saamiseks. Sellest piirangust on kirjutatud detailsemalt alapeatükis 4.3.3.

G2: Teisendamisel säilitatakse lähtepaketi struktuuri. Iga genereerimise tulemus pannakse eraldi paketti, mille nimi tuletatakse selle paketi nimest, mille põhjal teisendus tehti (paketi nime järgi on lisatud sulgudes sõna *transformed*). Kui genereerimise sihtkohaks olevas paketis on juba sellise nimega pakett, siis veateadet jms ei tule. Selle asemel üritab süsteem lisada uue genereerimise tulemuse olemasolevasse paketti ning muudatusi sünkroniseerida. Kahjuks mõnikord see ei tööta nii nagu peab, sellest on kirjutatud alapeatükis 4.5.2.

G3: Kui algmudeli klassidiagrammis on atribuudile määratud võimsustik suurem kui 1 (atribuudi seadete aknas on võimsustiku väärtuse ülempiir > 1) või atribuut on märgitud kollektsiooniks (atribuudi seadete aknas on täidetud märkeruut *isCollection*), loetakse seda atribuuti mitmeväärtuseliseks. Sellise atribuudi alusel luuakse eraldi tabel ja välisvõtme kitsendus. Loodava tabeli nimi on kombinatsioon atribuuti sisaldava olemitüübi nimest ja atribuuti nimest.

Mitmeväärtuselise atribuudi teisendamisel arvestatakse atribuudi juures määratud *Allow Duplicates* väärtusega. Selle väärtuse mõju illustreerib järgmine näide. Olgu meil olemitüüp *Klient*, millel on mitmeväärtuseline atribuut *email*. Olemitüübi *Klient* unikaalseks identifikaatoriks on atribuut *klient_id*.

Kui *Allow Duplicates* on märkimata, loodava uue tabeli struktuur on:

```
Klient_email (klient_id NOT NULL, email NOT NULL)
Primaarvõti (klient_id, email)
Välisvõti (klient_id) Viitab Klient (klient_id) ON DELETE CASCADE
```

Kui *Allow Duplicates* on märgitud, luuakse järgmist tabelit:

```
Klient_email (klient_email_id NOT NULL, klient_id NOT NULL, email
NOT NULL)
Primaarvõti (klient_email_id)
Välisvõti (klient_id) Viitab Klient (klient_id) ON DELETE CASCADE
```

G4: Kui algmudeli klassidiagrammis on olemitüübile määratud stereotüüp <<*singleton*>>, siis luuakse selle alusel tabel, milles tohib olla maksimaalselt üks rida. Selle saavutamiseks kasutatakse ära primaarvõtme omadust, mis ei luba tabelis mitut sama primaarvõtme väärtusega rida ning CHECK kitsendust, mis piirab primaarvõtme veerus lubatud väärtuste hulka ühega.

4.3. Mudeliteisenduste tehniline realisatsioon

Järgnevalt antakse ülevaade teisendamise toimimise printsiipidest. Ülevaate koostamisel on kasutatud Enterprise Architect-i ametlikku dokumentatsiooni [20], kuid suur osa informatsioonist pärineb praktilise töö käigus saadud kogemusest ja ei ole mujal kirjeldatud, olles seega üheks iseseisva töö tulemuseks.

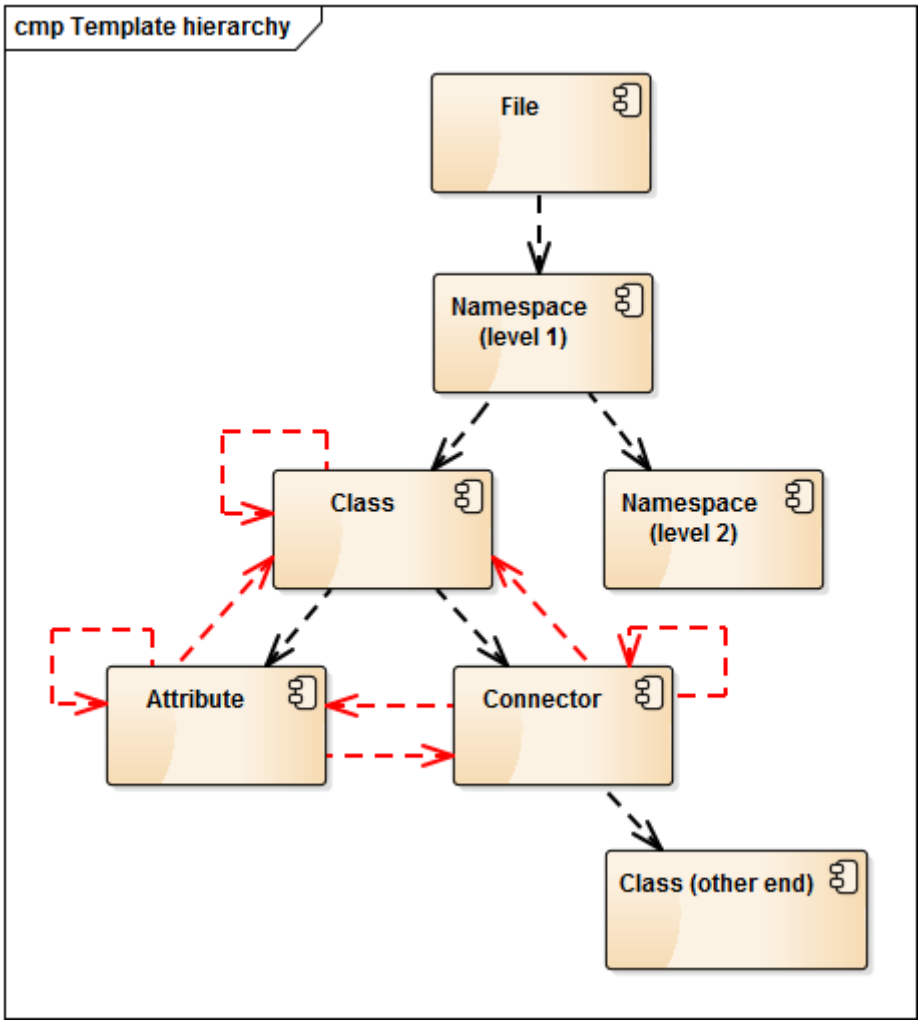
4.3.1. Teisendusmallid

Üldiselt teisendamise protsess toimub järgmiselt. Paketi teisendamine algab alati teisendusmallist nimega *File*, see on nagu tavalise programmi sisenemispunkt (ingl *entry point*). Edasine teisendamise järjekord on määratud teisendusmallides endis. Iga mall (v.a *File*) on seotud mingi konkreetse UML elemendi tüübiga ja oskab teisendada ainult sellesse tüüpi kuuluvaid mudelielemente. Nagu UML elemendid moodustavad omavahel hierarhilise struktuuri [44], nii on olemas ka varjatud hierarhia neid elemente teisendatavate mallide vahel. Teisendusmall võib üle anda juhtimist ainult teatud tüüpide teistele mallidele (vt Joonis 15). Mõned võimalikest üleminekutest on enamikul juhtudel ebamõistlikud või tekitavad lõputu rekursiooni ja neid tuleb üldjuhul vältida. Joonisel 15 on need üleminekud märgitud punaste nooltega.

On kolm viisi kuidas üle anda teisendamise juhtimist teisele mallile.

- Kasutades juhtimistüüpi keelekonstruktsiooni (ingl *Control Macro*) `list`. Süntaks: `%list=<TemplateName> @separator=<string> @indent=<string> [<conditions>]%` Teisendab kõik elemendid, mis on antud skoobis kättesaadavad ja mille tüüp langeb kokku *TemplateName* malli poolt toetava tüübiga. Näiteks, kui hetkel teisendatakse mingi klassi, siis selle konstruktsiooni abil võib teisendada ka selle atribuudid (kas kõik või need, mis vastavad mingile tingimusele).
- Kasutades malli asenduse keelekonstruktsiooni (ingl *Template Substitution Macro*), mille süntaks on `%TemplateName(<param1>, <param2>, ...)%`. Rakendab hetkel teisendatava elemendi või teda sisaldava elemendi suhtes *TemplateName* malli. Pakub ainsana võimalust edastada argumente ühest mallist teisse. „Vastuvõtvas“ mallis saab need argumendid kätte kasutades spetsiaalseid muutujaid `$parameter1`, `$parameter2`, jne.

- Kasutades funktsioonitüüpi keelekonstruktsiooni (ingl *Function Macro*) `PROCESS_END_OBJECT`, mille süntaks on `%PROCESS_END_OBJECT(<TemplateName>)%`. Seda konstruktsiooni kasutatakse ainult mallides, mis teisendavad seosed klasside vahel ehk *Connector*-tüüpi mallides. Selle idee on anda võimalus teisendada seose teises otsas asuvat klassi hetkel teisendatava klassi kontekstis (vt Joonis 15). Seda konstruktsiooni kasutavad näiteks praktilises töös lisatud teisendused, mis tegelevad {Mandatory; Or} üldistusseose korral ülatüüpi atribuutide ja seoste *copy-down inheritance* realiseerimisega.



Joonis 15. Teisendusmallide hierarhia ja üleminekud

Selleks, et saada ligipääsu mudeli andmetele, kasutavad teisendusmallid välja täitmise keelekonstruktsiooni (*Field Substitution Macros*). Need konstruktsioonid kujutavad endast sisuliselt kohatäitjaid, mida asendatakse teisendamise käigus tegelike muutujate (ehk teisendatava elemendi andmekildude) väärtustega. See, milliseid konstruktsioone on võimalik rakendada antud teisendusmallis, sõltub selle malli tüübist. Näiteks, konstruktsiooni `attType`,

mis on mõeldud selleks, et asendada see teisendatava atribuudi tüübiga, on võimalik kasutada ainult *Attribute*-tüüpi mallides, ent konstruktsioon `className` on saadaval nii *Class*-tüüpi, *Attribute*-tüüpi, kui ka *Connector*-tüüpi mallides. Erandiks on siin mallitüüp *File*, mis hoolimata nimetusest võimaldab teisendada ka paketi tähestikulises järjekorras esimese klassi. Selline kasutusviis on ebatavaline ning selle tegelik otstarve seisneb siiski teisendamise juhtimise üleandmises *Namespace*-tüüpi mallidele.

Lisas 2 esitatakse täielik nimekiri teisendusmallidest, mida antud töös parandati või lisati, koos viidetega realiseeritavatele täiendustele.

4.3.2. Lisamoodul

Mudeliteisenduste täiendamisel on üritatud maksimaalselt ära kasutada teisenduskeele võimalusi. Ainult teisendusmallide kasutamine, ilma lisamooduli kaasamiseta, annab selliseid ilmseid eeliseid nagu suurem täitmise kiirus ja arendamise mugavus. Esimene seisneb selles, et puuduvad lisakulud (ingl *overhead*), mis on seotud meetodite väljakutsumisega IDispatch liidese kaudu. Arendamise mugavuse all on mõistetud seda, et muudatuste tegemisel puudub lisamooduli ümberkompileerimise vajadus, mis omakorda nõuab Enterprise Architect-i sulgemist ja uuesti avamist. Sellest on täpsemalt kirjutatud ühe kogenud EA lisamoodulite arendaja artiklis [45], kus on pakutud ka lahendus probleemile (testimiseks mõeldud lisamoodul), mis kahjuks selles situatsioonis eriti ei sobi. Veel üheks probleemiks, mis on seotud lisamooduli kasutamisega, on mõningane äriloogika dubleerimine teisendusmallide ja lisamooduli vahel, mida valminud lahenduses ei saanud täielikult vältida.

Vaatamata eelpool kirjeldatud puudustele, lisamoodulid annavad teisendusmallidega võrreldes oluliselt suuremaid võimalusi. Antud töös lisatud 27-st täiendustest, koguni 20 on tehtud lisamooduli abil, seega ilma selleta poleks seda tööd võimalik teha. Järgnevalt on loetletud vaid mõned lisamooduli eelised teisendusmallide ees:

- itereerimine üle elementide massiivi ja elementide töötlemine;
- kasutajaga suhtlemine dialoogiakna kaudu;
- lisaandmete meelespidamine / hoidmine teisendamise „sessiooni“ jooksul;
- regulaaravaldiste kasutamine stringitöötlusel;
- teisendatava mudeli muutmine;
- ligipääs suuremale hulgale lähtemudelis olevatest andmetest;

- ligipääs suvalisele mudeli elemendile, sõltumata sellest, mis elementi parasjagu teisendatakse;
- igasuguste arvutuste tegemine;
- võimalus kasutada tavapärasemat objektorienteeritud lähenemist;
- jne...

Liidestamine lisamooduliga toimub funktsioonitüüpi keelekonstruktsiooni `EXEC_ADD_IN` abil. Selle süntaks on `EXEC_ADD_IN("addin_name", "function_name", <param1>, <param2>, ...)`. Keelekonstruktsioon kutsub välja COM-lisamooduli *addin_name* avaliku meetodi nimega *function_name*, kasutades IDispatch liidest. Iga lisamooduli meetodi signatuur peab järgima kindlat struktuuri. Meetodil on kaks parameetrit: esimene on `EA.Repository` tüüpi, mida väärtustatakse väljakutsumisel viitega projekti andmeid sisaldavale objektile; teine on objekti tüüpi, mis saab argumendina stringide massiivi. See massiiv koosneb `EXEC_ADD_IN` poolt edastatud parameetrite väärtustest (*param1*, *param2*, jne). Meetod tagastab väärtuse, mis peaks olema string, kuid ei ole keelatud tagastada mõnda muud tüüpi väärtust (sel juhul teisendusmallid seda lihtsalt ignoreerivad). Näitena toome töös loodud lisamooduli ühe meetodi signatuuri (C# keeles): `public object Initialize(Repository Repository, object Args)`. Selle meetodi väljakutsumine toimub teisendusmalli *File* alguses ja näeb välja niimoodi: `%EXEC_ADD_IN("EADDLEnh", "Initialize", genOptDefaultDatabase)%`, kus *EADDLEnh* on lisamooduli nimi ja `genOptDefaultDatabase` on seadistatud vaikimisi andmebaasisüsteemi nimi.

Lisamoodul on kirjutatud C# keeles. See polnud ainus võimalik variant, kuna ActiveX/COM tehnoloogia toetus on sisse ehitatud paljudesse keeldesse. ActiveX komponente on võimalik luua näiteks sellistes keeltes nagu Delphi, C++, VB6, Visual Basic .NET ja C#. Viimased kaks keelt põhinevad .NET tehnoloogial, mis realiseerib COM toetust kasutades spetsiaalset vahevara COM Interop [46]. ActiveX/COM komponentide loomine nendes keeltes ei nõua midagi enamat, kui lihtsat projekti seadistamist ja vajalike signatuuridega meetodite loomist. Valides Visual Basic .NET ja C# vahel lähtuti nii keele populaarsusest [47], tuttavlikkusest kui ka näidete kättesaadavusest EA lisamooduli loomise kontekstis. Kõiki neid kaalutlusi arvestades langetati lõplik otsus C# kasuks.

Lisas 3 esitatakse täielik lisamooduli funktsioonide nimekiri, koos viidetega realiseeritavatele täiendustele.

4.3.3. Probleemid ja võimalikud edasiarendused

Järgnevalt kirjeldatakse mõningaid probleeme, mis on jäänud selles täienduste komplektis lahendamata. Enamus neist on seotud Enterprise Architect-i vahekeele piirangutega ja ei ole hetkeseisuga põhimõtteliselt lahendatavad. Mõnedest algselt plaanitud täiendustest loobuti praktilistel kaalutlustel ja nende puhul antakse vihjeid, kuidas neid saaks realiseerida.

Mitut veergu hõlmavad UNIQUE-kitsendused. Sellist tüüpi kitsendust ei ole võimalik vahefailis defineerida, sest EA vahekeeles puudub element tabelitaseme UNIQUE-kitsenduse kirjeldamiseks. Vahekeeles saab defineerida ainult veerutaseme UNIQUE-kitsendusi (need hõlmavad ainult ühte veergu). Sellest probleemist tingituna ei ole teisendatud mudelis tabelitaseme UNIQUE-kitsendused esitatud klassi operatsioonina ja need tuleb mudelisse käsitsi lisada.

Samanimelised välisvõtmete veerud. Probleemi illustreerib Joonis 21 alapeatükis 4.5.3. Tabelis *Tellimus* peaks tekkima kaks veergu nimedega, näiteks, *kood1* ja *kood2*, mis vastavad kahele erinevale välisvõtme kitsendusele. Mõlemad kitsendused on küll õigesti esitatud klassi operatsioonidena, kuid atribuutide hulgas on ainult üks veerg, nimega *kood*. Seda probleemi võiks lahendada seose teisendamisel kõikide teiste klassi seoste analüüsimise kaudu, kuid see aeglustaks oluliselt teisendamise protsessi ning pole seetõttu otstarbekas.

1 / 1 seose teisendamine. Ideaalis tuleks panna osalevate olemitüüpide atribuudid kokku ja moodustada uus tabel (vt alapeatükk 4.1.6). Reaalselt nõuab selle realiseerimine kas algse mudeli programmilist muutmist teisendamise alguses või täiendava seoste analüüsimise loogika lisamist iga klassi ja seose teisendamisel, mis samuti pole otstarbekas. Võivad esineda ka probleemid mudelite sünkroniseerimisega (vt märkus alapeatüki 4.5.2 lõpus).

Üldistuste hulgaga seotud kitsendused. Praegu töötavad teisendused nii, et kui üldistusseose ülaklass on määratud abstraktseks, siis interpreteeritakse kõik temaga seotud alamklasse nii, nagu need kuuluksid ühte üldistuste hulka, millel on kitsendus {Mandatory; Or}. Tegelikult võimaldab Enterprise Architect küll määrata erinevaid üldistuste hulki ja deklareerida neile erinevaid kitsendusi, kuid need andmed ei ole API kaudu kättesaadavad. On oodata, et seda parandatakse järgmistes EA versioonides.

Piiritletud indentifikaatorid. Kui mõni identifikaator langeb kokku SQLi reserveeritud sõnaga, võiks ümbritseda seda jutumärkidega (praegu pannakse identifikaatori lõppu alakriips). Praegu

seda ei tehta, sest EA vahekeele süntaks ei näe ette jutumärkide kasutamise võimalust elementide nimede kirjeldamisel.

Mitmeväärtuselised atribuudid ülaklassis. {Mandatory; Or} üldistusseose teisendamisel ei arvestata ülaklassi mitmeväärtuseliste atribuutidega.

Rekursiivsed seosed. Sellise seosetüübi teisenduste täiustamine ei ole realiseeritud vähese kasutatavuse tõttu.

Välistav kaar. Sama, mis eelneval.

4.4. Täiendatud mudeliteisenduste installeerimine

Loodud tarkvara installeerimine toimub kahes etapis: lisamooduli installeerimine ja uue teisendustüübi importimine projekti. Allpool toodud juhendid selgitavad vajalikke toiminguid ja teostamise järjekorda.

Lisamooduli installeerimine

Enne installeerimist veenduge, et:

- arvutisse on installitud .NET Framework 4;
- Enterprise Architect ei ole käivitatud;
- kasutaja on sisse logitud administraatori õigustes;
- (soovitavalt) kasutajakonto kontroll (UAC) on installeerimise ajaks välja lülitatud.

Installeerimispaketi võib alla laadida aadressilt
http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/eaddlenh_setup.msi

Vajadusel saab installeerida lisamooduli käsitsi. Selleks on vaja:

1) Laadida alla ja lahti pakkida ZIP-arhiivi aadressilt http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/manual_setup.zip.
2) Kopeerida faili EADDLEnh.dll mugavasse asukohta: näiteks luua kataloogis `%programfiles(x86)%\Sparx Systems` kataloogipuu `EAAddins\EADDLEnh` ja kopeerida faili sinna.
3) Kataloogis, kuhu kopeeriti DLL-faili, käivitada käsurealt järgmine käsk:
   `%WINDIR%\Microsoft.NET\Framework\v4.0.30319\regasm EADDLEnh.dll /codebase`
4) Võtta lahtipakitud arhiivist reg-fail ja avada see topeltklõpsuga. Nõustuda Windowsi hoiatusega.

Teisendustüübi importimine projekti

Teisendustüübi importimine toimub projektipõhiselt. Teisendustüübi definitsioonifail asub siin: http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/DDLenhanced.xml (laadida alla). EA menüü *Project* all valida *Model Import / Export -> Import Reference Data*. Avanenud aknas vajutada nupule *Select File* ja navigeerida kataloogi, kus on allalaaditud XML-fail. Valida nimekirjas *Select Datasets to Import* esimene rida ja kinnitada valik klõpsates *Import* peale.

Deinstalleerimine

Enne deinstalleerimist veenduge, et Enterprise Architect ei ole käivitatud.

Kui lisamooduli installeerimisel kasutati installeerimispaketi, siis deinstalleerimine toimub Windows 7-s (ja uuemates versioonides) akna *Control Panel -> Programs and Features* abil. Loendis on kuvatud programm nimega EADDLEnh, mis tuleks valida ja klikkida nupul *Uninstall*. Eemaldamise lõpetamiseks järgige juhendeid ekraanil.

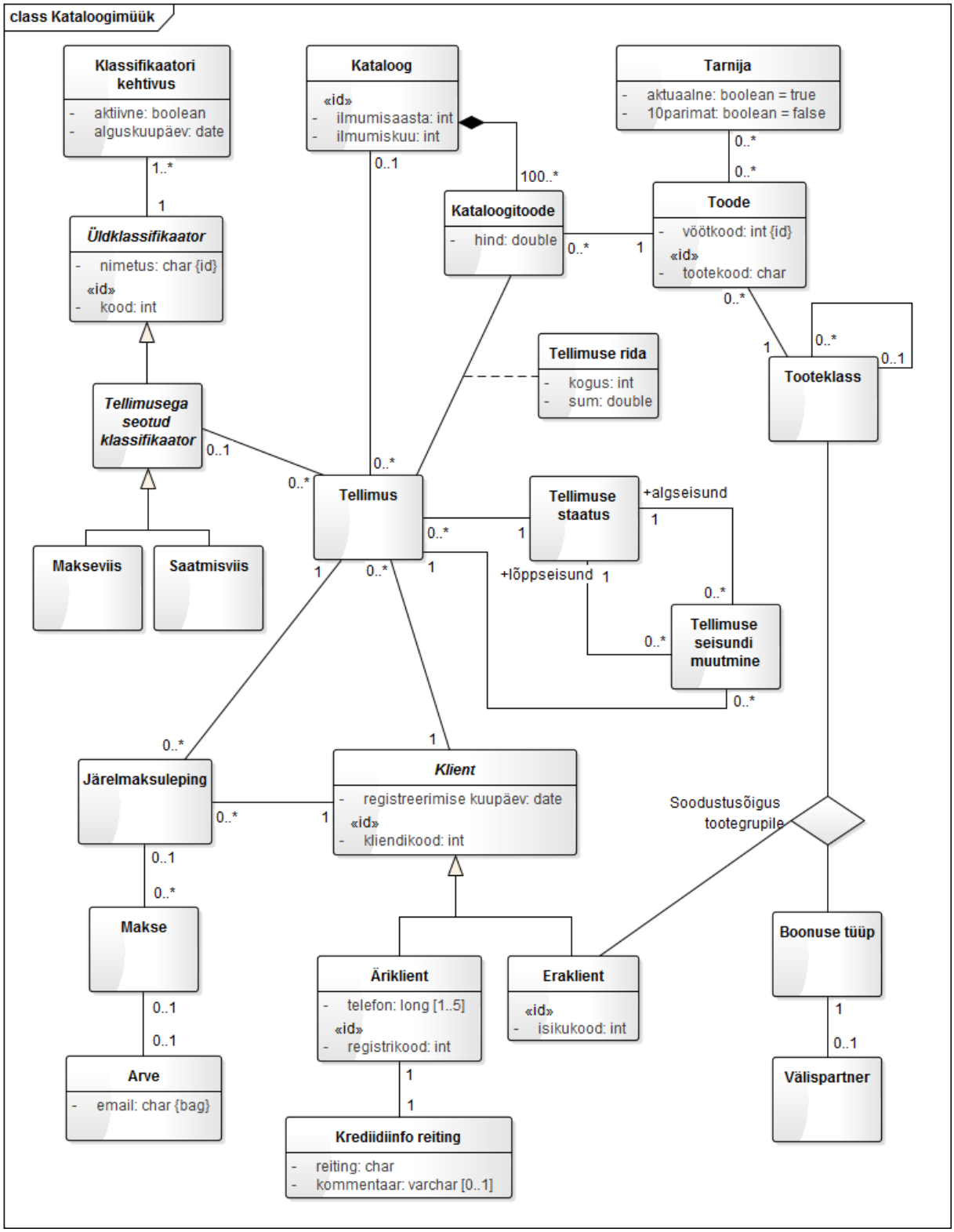
Kui lisamooduli installeerimine toimus käsitsi, siis vajaliku informatsiooni DLL-i deinstalleerimiseks leiate käsu `regasm` kirjeldusest [48]. Pärast seda võib soovi korral kustutada faili EADDLEnh.dll.

Alternatiivina võib keelata lisamooduli, kasutades EA dialoogiakna *Add-In Manager* [49].

Pange tähele, et kord imporditud teisendustüüpi pole hiljem võimalik projektist eemaldada.

4.5. Täiendatud mudeliteisenduste kasutamise näide

4.5.1. Kontseptuaalne näidismudel



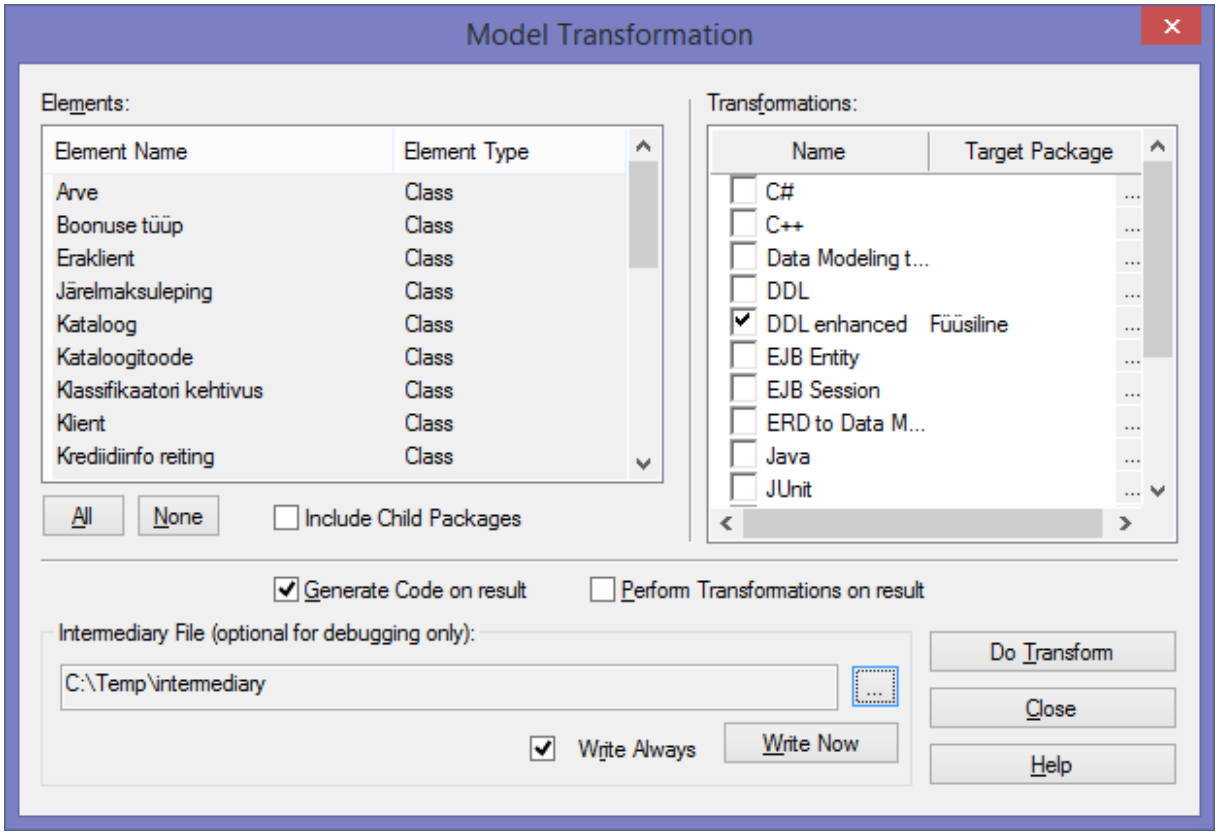
Joonis 16. Kontseptuaalse andmemudeli klassidiagramm

4.5.2. Teisendamine

Järgnevalt on illustreeritud teisendamise rakendamist näidismudelile. Näidismudeli võib alla laadida aadressilt http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/example.zip.

Algmudel asub paketis nimega „Kontseptuaalne“. Enne teisenduse käivitamist tuleb määrata vaikimisi andmebaasisüsteem. Selleks tuleb valida menüüst *Settings -> Database Datatypes*… Avanenud aknas tuleb nimekirja *Product Name* alt valida soovitud andmebaasisüsteem ja tähistada märkeruut *Set as Default*. Soovitatav on ka üle vaadata andmetüüpide vastavustabel ja veenduda, et iga üldtüübi jaoks, mida kasutati kontseptuaalses mudelis atribuutide defineerimiseks, on määratud vastav andmebaasisüsteemi-spetsiifiline andmetüüp. Näiteks, EA 11 ei määra atribuudi üldtüübile *int* vastavat Oracle DBMS andmetüüpi ja seda tuleks teha käsitsi eelpool mainitud seadistuste aknas. Antud näites kasutatakse teisendamiseks Enterprise Architect versiooni 11. Vaikimisi andmebaasisüsteemiks on valitud PostgreSQL.

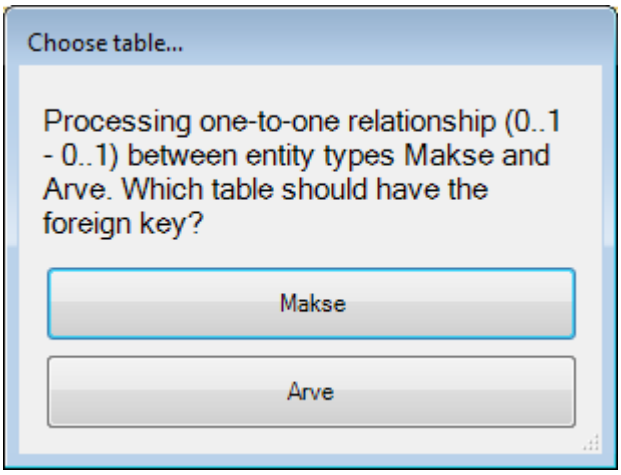
Kõige lihtsam viis käivitada teisendamist on valida projektivaates (*Project Browser*) soovitav pakett ning vajutada klahvikombinatsiooni <Ctrl> + <Left Shift> + <H>. Avaneb aken:



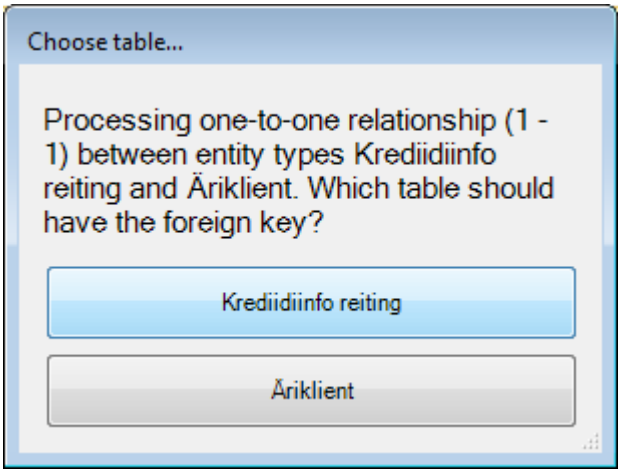
Joonis 17. Enterprise Architect-i teisendamise seadistamise aken

Aknas tuleb määrata soovitud teisendustüüp (*DDL* – vana sisseehitatud versioon või *DDL enhanced* – selles töös tehtud versioon) ja sihtpakett, kuhu panna teisendamise tulemus. Ülejäänud valikud võib jätta muutmata. *Do Transform* nupule vajutamine käivitab teisendamise protsessi.

Kuna kontseptuaalses andmemudelis on olemas 0..1 / 0..1 ja 1 / 1 tüüpi seoseid, siis teisendamise käigus küsitakse kasutajalt täiendavat sisendit (vt Joonis 18 ja 19).



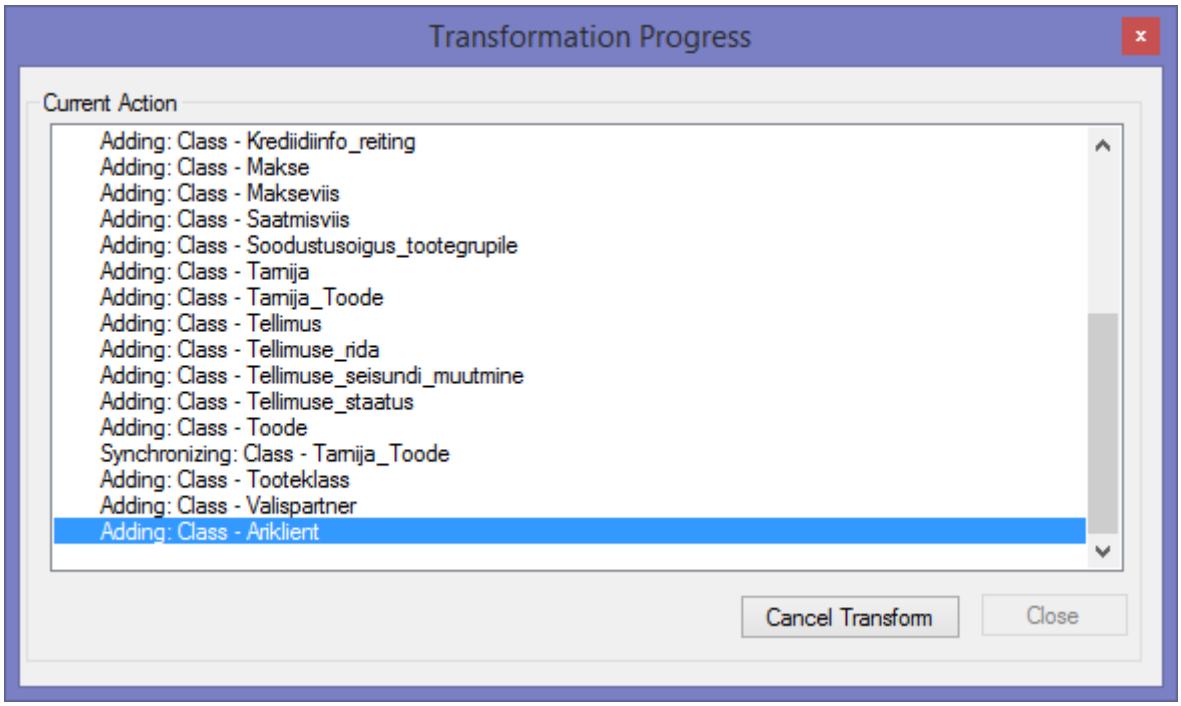
Joonis 18. Lisamooduli dialoogiaken 0..1 / 0..1 seose teisendamisel



Joonis 19. Lisamooduli dialoogiaken 1 / 1 seose teisendamisel

Võib katsetada erinevaid kombinatsioone, kuid selle näite jaoks on tehtud vastavalt valikud „Makse“ ja „Krediidiinfo reiting“.

Kui sisend on antud, siis ülejäänud protsess toimub automaatselt:

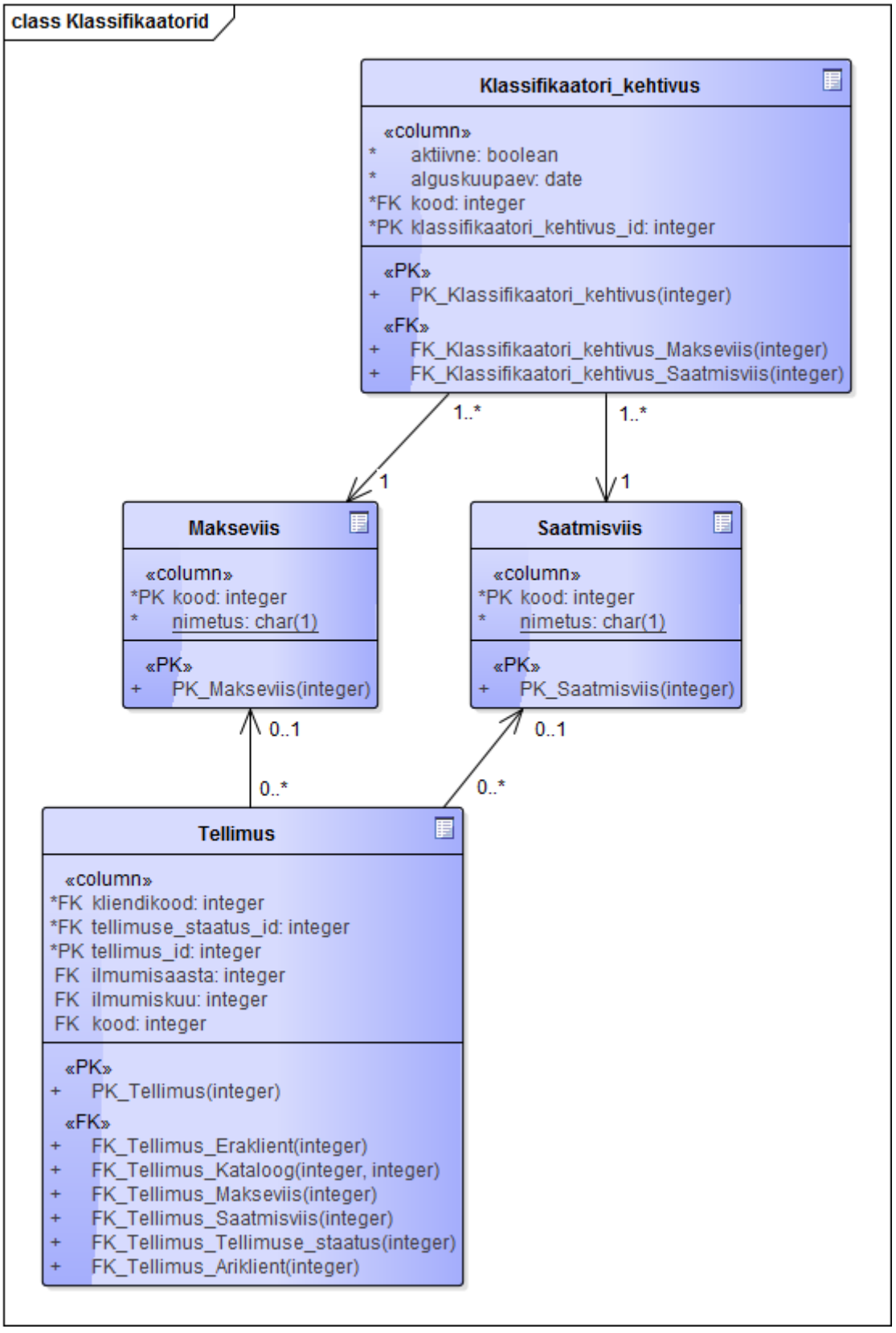


Joonis 20. Enterprise Architect-i teisendamise protsessi edenemise aken

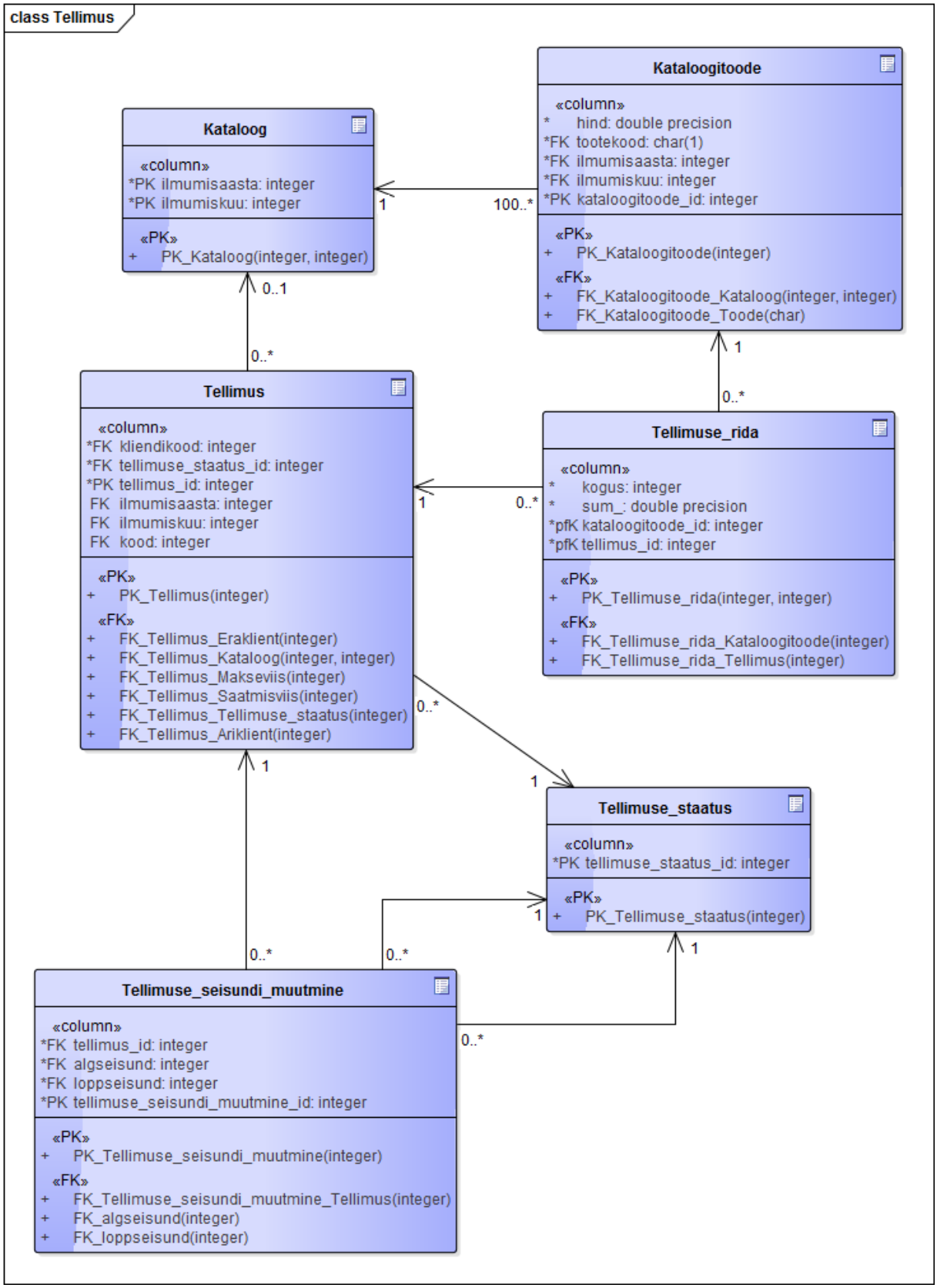
Teisendamise tulemusena on sihtpaketis „Füüsiline“ on loodud alampakett nimega „Kontseptuaalne (transformed)“. Kõigi loodud tabelite kirjeldused koondatakse automaatselt ühele paketiga samanimelisele klassidiagrammile. Kuna saadud diagrammil on kokku 24 tabelit, siis parema loetavuse eesmärgil on see diagramm käsitsi jagatud kuueks väiksemaks alamdiagrammiks, mis esitatakse järgmises alapeatükis. Lisas 4 esitatakse võrdluseks vana teisenduse abil genereeritud mudel.

Märkus sünkroniseerimise kohta: Kuigi Enterprise Architect toetab automaatset sünkroniseerimist algmudeli ja teisendamisega saadud mudelite vahel, siis mõningatel juhtudel see ei tööta nagu peab. See probleem esineb aeg-ajalt nii sisseehitatud teisenduste kui ka selles töös valminud *DDL enhanced* teisenduse kasutamisel ja see on tingitud teisendusmootori töö eripärast. Enne korduvat genereerimist on soovitatav eelmise teisendusega loodud mudel võimalusel kustutada või selle paketi nime muuta. Kui seda ei teha, siis korduvalt teisendatud mudel võib sisaldada tabelite kirjeldustes üleliigseid välisvõtme kitsenduste kirjeldusi või üleliigseid seosejooni. *DDL enhanced* teisenduse puhul kerkib see probleem eriti sageli esile, kui algmudel sisaldab {Mandatory; Or} tüüpi üldistusseoseid.

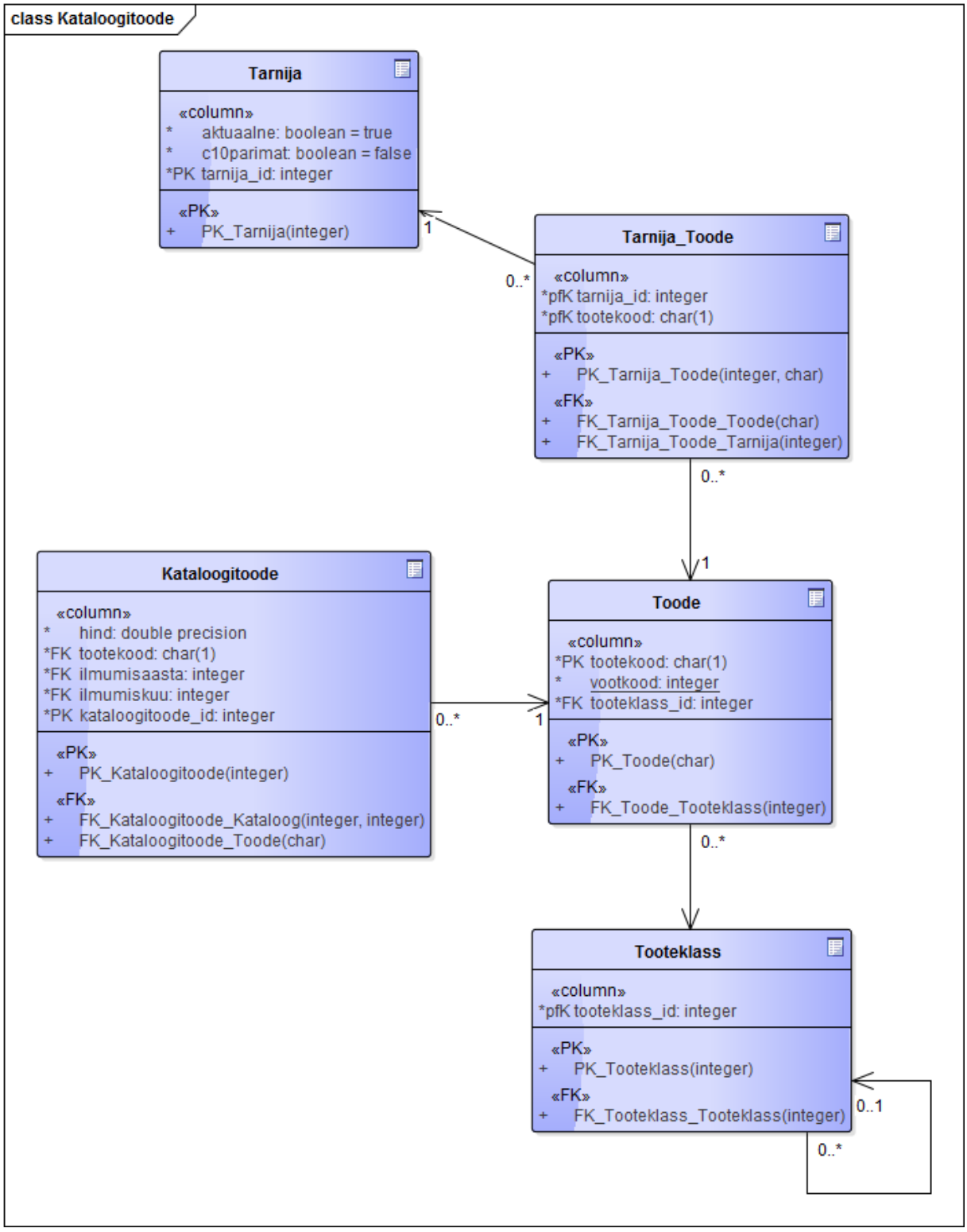
4.5.3. Teisenduse tulemusena saadud mudel



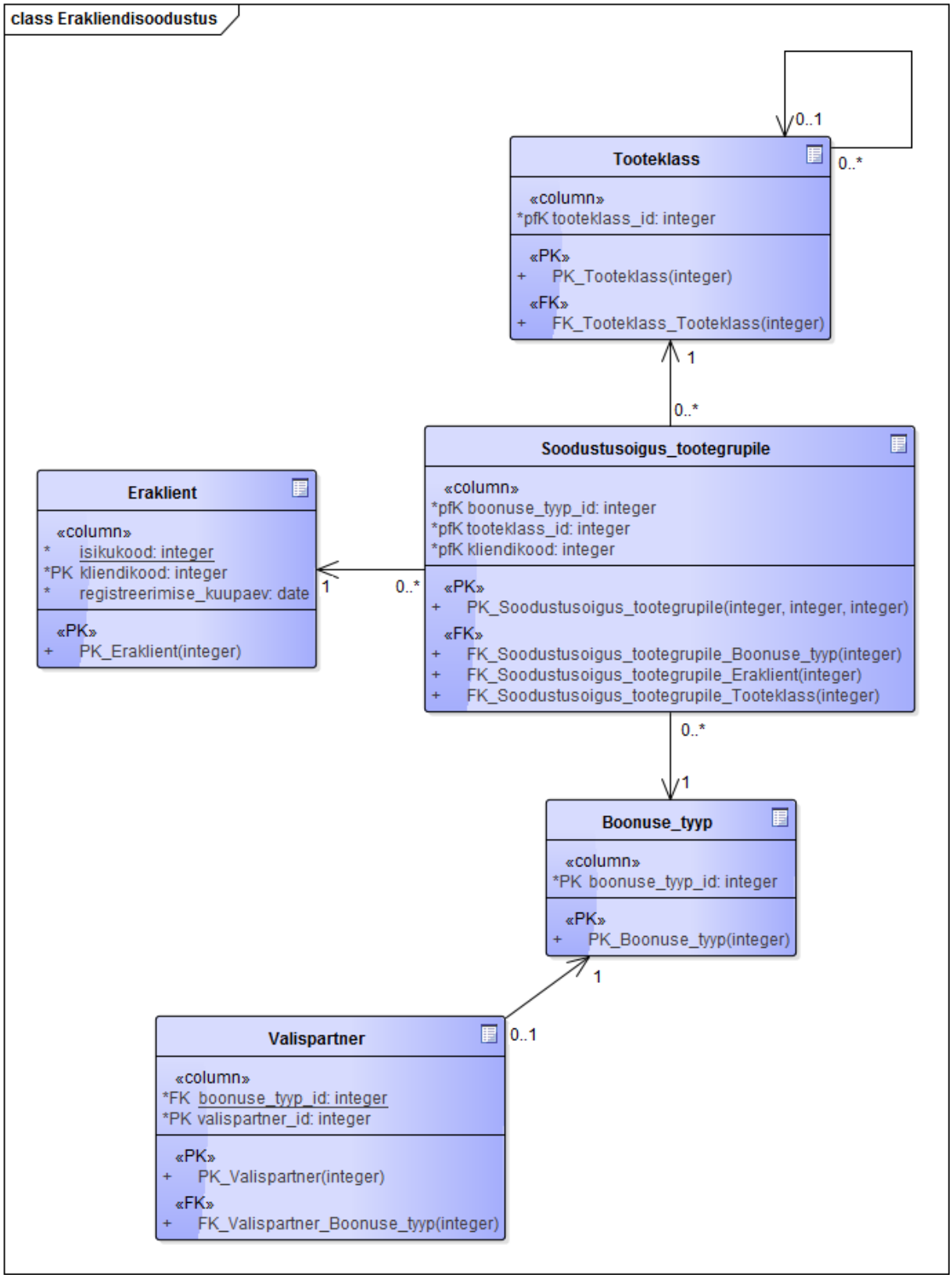
Joonis 21. Teisendatud mudel. Klassifikaatorid



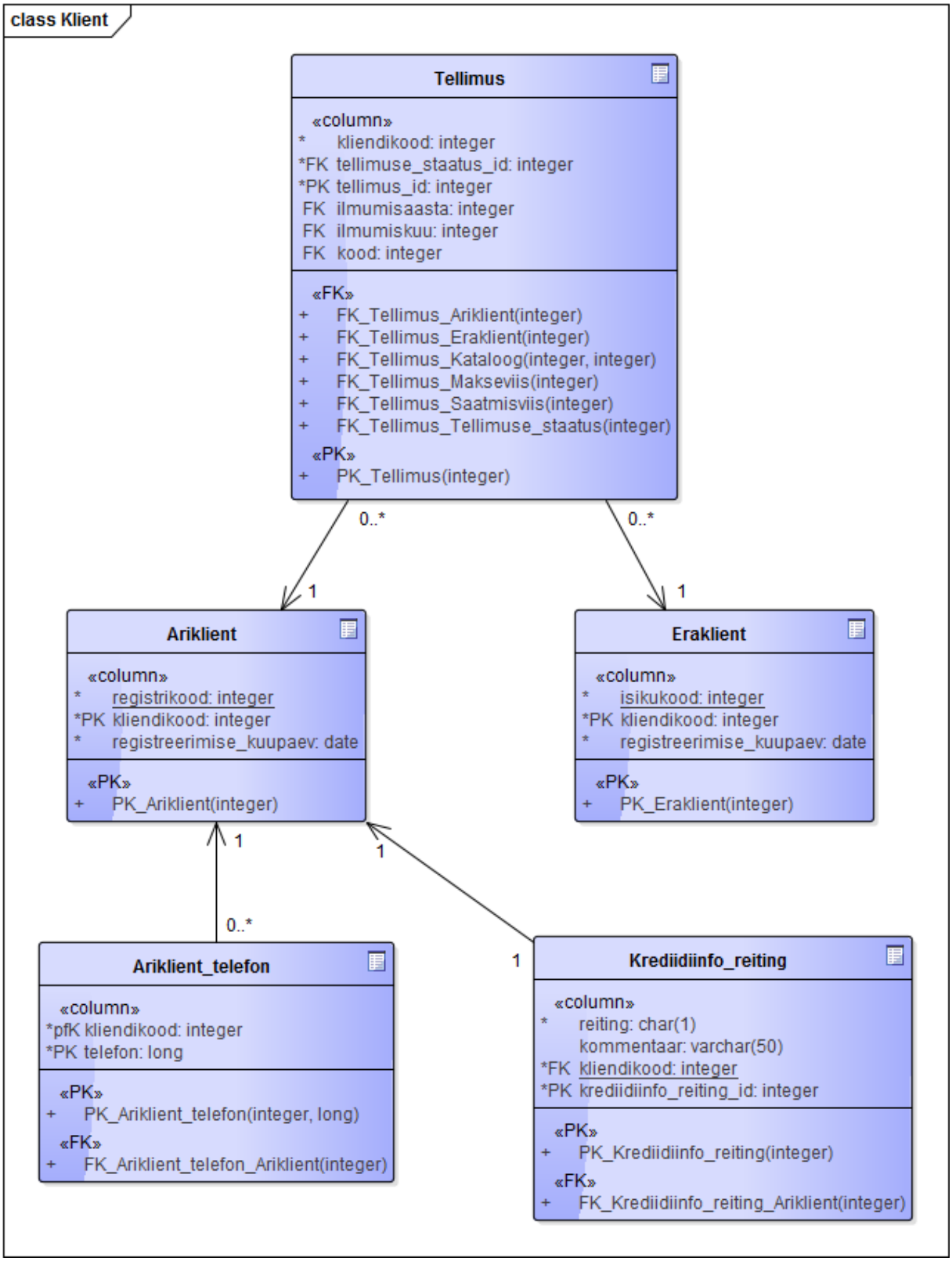
Joonis 22. Teisendatud mudel. Tellimus



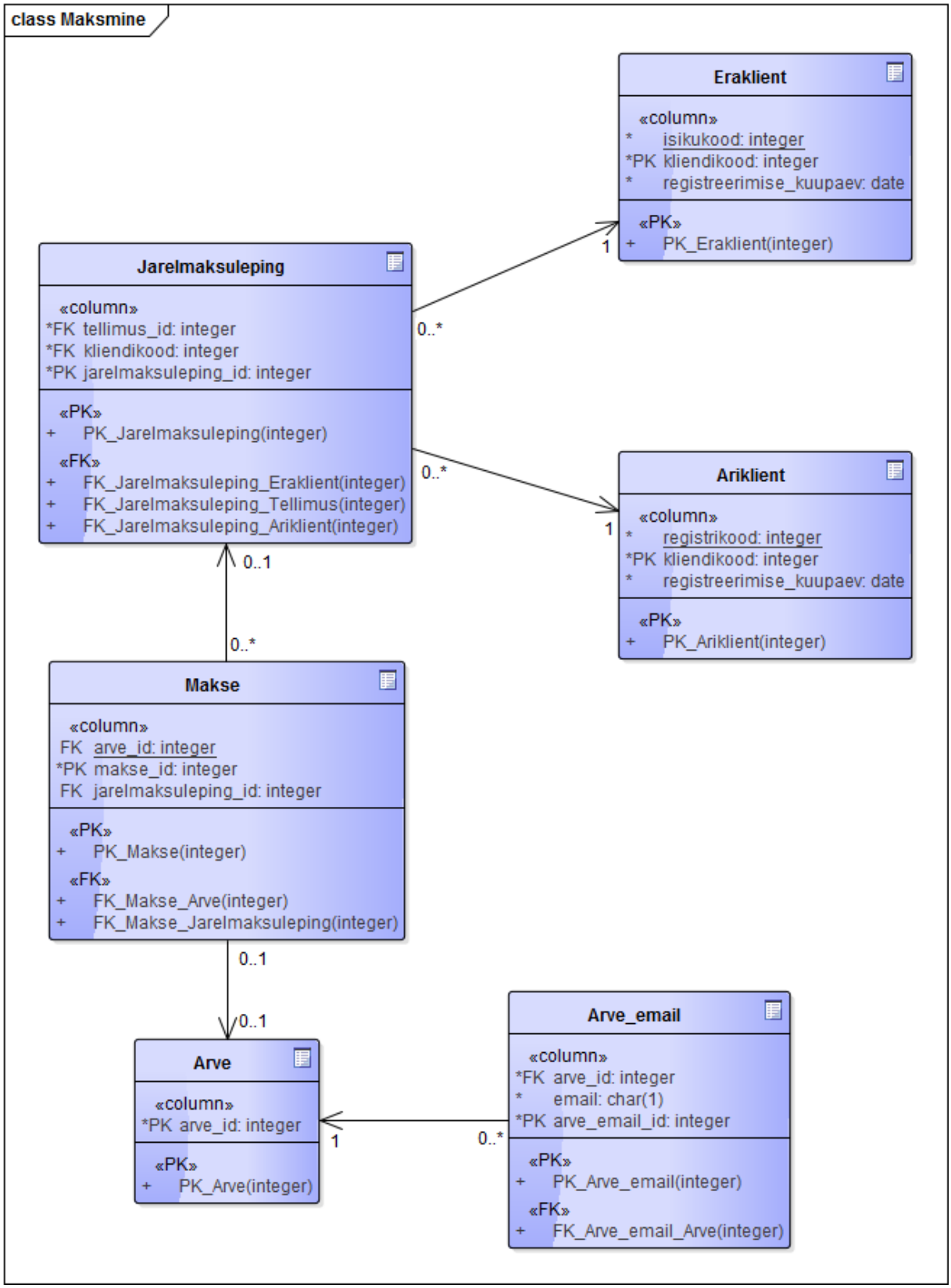
Joonis 23. Teisendatud mudel. Kataloogitoode



Joonis 24. Teisendatud mudel. Erakliendisoodustus



Joonis 25. Teisendatud mudel. Klient



Joonis 26. Teisendatud mudel. Maksmine

Vana teisenduse abil loodud mudelit võib huviline vaadata Lisast 4.

Kokkuvõte

Käesoleva töö eesmärgiks oli parandada Enterprise Architect CASE vahendisse sisse ehitatud mudeliteisendusi, mida kasutatakse andmebaaside MDA-keskses arenduses, et automaatselt teisendada UML klassidiagrammina esitatud detailne kontseptuaalne andmemudel (PIM) füüsilise disaini täpsusega andmemudeliks (PSM). Kavandatava eesmärgi saavutamiseks oli vaja täita selliseid ülesandeid nagu võimalikult täpse informatsiooni kogumine hetkeolukorra kohta, projekti skoobi määramine, Enterprise Architect-i teisenduskeele ja vahekeele uurimine, sobiliku lahenduse leidmine ning tulemuste kontrollimine ja hindamine koostatud näitemudeli abil. Töö käigus lisati kokku 27 täiendust, mille rakendamist demonstreeriti ning kontrolliti loodud näitemudeli teisendamisega. Teisenduste toimimist kontrolliti Enterprise Architect versioonide 11 ja 12 abil. Teisendatud mudelis esinevad endiselt üksikud kõrvalekalded soovituslikust disainist [39], mille kõrvaldamine ei ole praegu võimalik Enterprise Architect-i tehniliste piirangute tõttu. Sellele vaatamata vajab saadud andmemudel enne koodi genereerimist minimaalset käsitsi sekkumist. Seega võib järeldada, et töös püstitatud eesmärk on edukalt saavutatud.

Töö tulemusena on loodud uue teisendustüübi *DDL Enhanced* definitsiooni fail, mida saab alla laadida siit: http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/DDLenhanced.xml ja Enterprise Architect-i lisamoodul, mille installeerimispaketi võib alla laadida aadressilt http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/eaddlenh_setup.msi. Ühtlasi demonstreerib see töö, kuidas ületada EA teisenduskeele puudujääke, kombineerides selles keeles kirjutatavad teisendusmallid mõnes kõrgtaseme progammeerimiskeeles kirjutatud lisamooduliga. Võib järeldada, et selline liidestamine on väga oluline keerulisemate teisenduste realiseerimiseks, sest 27-st täiendustest 20 on tehtud lisamooduli kaasamise abil. Samas on esitatud lahenduses ka mõningad puudused, mis on seotud äriloogika dubleerimisega teisendusmallides ja lisamoodulis. Seega teisenduste edasiarendamise üheks kaasaskäivaks tegevuseks võiks olla koodi refaktoreerimine ja parema arhitektuurse lahenduse leidmine. Uute täienduste lisamine sõltub palju EA vahekeele võimalustest. Seega uute Enterprise Architect-i versioonide ilmumine võib luua soodsad tingimusi teisenduste edasiarendamiseks ja tööd võib selles suunas jätkata.

Summary

The goal of this work was to improve the built-in model transformations of Enterprise Architect CASE tool that are used in MDA-centric database development process in order to automate the PIM-to-PSM conversion of data models represented by UML class diagrams. Accomplishment of this goal required performing such tasks as gathering the accurate information about current situation, determining the project scope, learning about Enterprise Architect's transformation and intermediary language, finding an appropriate solution, and verification and assessment of results by designing an example model. We demonstrated and verified the 27 improvements added by this work by transforming the example model. The transformations were made by using the Enterprise Architect versions 11 and 12. The transformed model still exhibits certain deviations from the recommended design practices [39], which cannot be eliminated due to Enterprise Architect's technical limitations. However, despite of this fact the produced model needs only minimal manual intervention before generating the code. Thus, we can conclude that the planned goal was successfully achieved.

The outcome of this work is a definition file of the new transformation type *DDL Enhanced* (available at http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/DDLenhanced.xml), and an add-in for Enterprise Architect deployed with the MSI installer package (available at http://www.tud.ttu.ee/web/Marina.Nekrassova/loputoo2015/eaddlenh_setup.msi). This work also demonstrates how to overcome the obstacles of EA transformation language by combining the usage of transformation templates and add-in modules written in some high-level programming language. One can conclude that this bundle is of utter importance in implementing some more advanced transformations, because out of 27 added improvements 20 are done with the aid of the add-in. Still, the presented solution is not without shortcomings that are associated with duplication of business logic between transformation templates and the add-in. Therefore, the further development of transformations should involve code and architecture refactoring. The potential for further improvements strongly depends on the syntax of EA's intermediary language. This means that future Enterprise Architect releases may offer favorable conditions for the further development of transformations and one can continue the work in this direction.

Kasutatud kirjandus

[1] Wikipedia. Model-driven architecture. [WWW] http://en.wikipedia.org/wiki/Model-driven_architecture (21.04.2015)

[2] Krogmann, K., Becker, S. (2007). A Case Study on Model-Driven and Conventional Software Development: The Palladio Editor. – *Software Engineering (Workshops)*, 106, 169-176. [E-ajakiri] (http://subs.emis.de/LNI/Proceedings/Proceedings106/gi-proc-106-020.pdf) (21.04.2015)

[3] OMG. MDA FAQ. [WWW] http://www.omg.org/mda/faq_mda.htm (21.04.2015)

[4] TED Tenders Electronic Daily. Tarneleping - 180689-2014. [WWW] http://ted.europa.eu/udl?uri=TED:NOTICE:180689-2014:TEXT:ET:HTML (21.04.2015)

[5] OMG. MDA specifications. [WWW] http://www.omg.org/mda (21.04.2015)

[6] OMG. Model Driven Architecture (MDA). MDA Guide rev. 2.0. [WWW] http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf (21.04.2015)

[7] Wikipedia. Model transformation language. [WWW] http://en.wikipedia.org/wiki/Model_transformation_language (21.04.2015)

[8] MOLA Project. Ametlik koduleht. [WWW] http://mola.mii.lu.lv (22.04.2015)

[9] OMG. MOF Model to Text Transformation Language, v1.0. [WWW] http://www.omg.org/spec/MOFM2T/1.0/PDF (22.04.2015)

[10] Wikipedia. Unified Modeling Language. [WWW] http://en.wikipedia.org/wiki/Unified_Modeling_Language (22.04.2015)

[11] OMG. UML specifications. [WWW] http://www.omg.org/spec/UML (22.04.2015)

[12] Fowler, M. UML Distilled : A Brief Guide to the Standard Object Modeling Language. Boston : Addison-Wesley Longman Publishing Co., Inc, 2003

[13] OMG. The current UML specification. [WWW] http://www.omg.org/spec/UML/Current (22.04.2015)

[14] Wikipedia. Profile (UML). [WWW] http://en.wikipedia.org/wiki/Profile_(UML) (22.04.2015)

[15] OMG Formal Specifications. [WWW] http://www.omg.org/spec/ (22.04.2015)

[16] Ambler, Scott W. A UML Profile for Data Modeling. [WWW] http://www.agiledata.org/essays/umlDataModelingProfile.html (22.04.2015)

[17] Sparx Systems official site. Database Modeling in UML. [WWW] http://www.sparxsystems.com/resources/uml_datamodel.html (22.04.2015)

[18] Sparx Systems official site. Enterprise Architect product page. [WWW] http://www.sparxsystems.com.au/products/index.html (22.04.2015)

[19] Sparx Systems. MDA Overview. [WWW] http://www.sparxsystems.com.au/bin/MDA%20Tool.pdf (22.04.2015)

[20] Sparx Sysems. Enterprise Architect Software Developers' Kit. [WWW] http://www.sparxsystems.com.au/downloads/resources/booklets/enterprise_architect_sdk.pdf (22.04.2015)

[21] Nordquist, M. Data modeling and SQL generation with Enterprise Architect. [WWW] http://www.marcusnordquist.com/?p=112 (22.04.2015)

[22] Sparx Systems. Physical Data Model. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/database_engineering/physical_data_model.html (22.04.2015)

[23] Sparx Systems. Supported DBMSs. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/database_engineering/supported_databases.html (22.04.2015)

[24] Sparx Systems. Model Transformation. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/model_transformation/mdastyletransforms.html (22.04.2015)
[25] Sparx Systems. Intermediary Language. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/model_transformation/intermediarylanguage.html (22.04.2015)
[26] Sparx Systems. MDA Transformations User Guide. [WWW] http://www.sparxsystems.com.au/downloads/resources/booklets/model_driven_architecture.pdf (22.04.2015)
[27] Sparx Systems. Enterprise Architect Object Model. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/automation_and_scripting/theautomationinterface.html (22.04.2015)
[28] Sparx Systems. Enterprise Architect Add-In Model. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/automation_and_scripting/addins_2.html (22.04.2015)
[29] Sparx Systems. Third Party Extensions for Enterprise Architect. [WWW] http://www.sparxsystems.com.au/products/3rdparty.html (22.04.2015)
[30] Google Groups. Sparx Enterprise Architect General. [WWW] https://groups.google.com/forum/m/#!topic/sparx-enterprise-architect-general/TaXdSJ3UgLk (22.04.2015)
[31] Sparx Systems. Deploy Add-Ins. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12.0/automation_and_scripting/deployingaddins.html (22.04.2015)
[32] Bellekens, G. Tutorial: Create your first C# Enterprise Architect addin in 10 minutes [WWW] http://bellekens.com/2011/01/29/tutorial-create-your-first-c-enterprise-architect-addin-in-10-minutes (22.04.2015)
[33] Sparx Systems. Create Add-Ins. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12.0/automation_and_scripting/creatingaddins.html (22.04.2015)
[34] Sparx Systems. Built-in Transformations. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/model_transformation/builtintransformations.html (22.04.2015)
[35] Wikipedia. Entity–relationship model. [WWW] http://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model (22.04.2015)
[36] Eessaar, E. Teema 7. Andmebaaside projekteerimine: strateegiline analüüs ja detailanalüüs. [WWW] http://maurus.ttu.ee/ained/IDU0220_2014/doc/3/Teema_IDU0220_7_2014.pdf (22.04.2015)
[37] Sparx Systems. DDL Transformation. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/model_transformation/ddltransformation.html (22.04.2015)
[38] Sparx Systems. Transform Connectors. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/model_transformation/transform_connectors.html (22.04.2015)
[39] Eessaar, E. Andmebaaside projekteerimine. Tallinn : Tallinna Tehnikaülikooli Kirjastus, 2008
[40] Generalization in UML. [WWW] http://www.uml-diagrams.org/generalization.html (22.04.2015)

[41] Sparx Systems. Generalization Sets. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/modeling_basics/generalization_sets.html (22.04.2015)
[42] PostgreSQL 9.4.1 Documentation. Chapter 4 – SQL Syntax. 4.1. Lexical Structure. [WWW] http://www.postgresql.org/docs/9.4/static/sql-syntax-lexical.html (22.04.2015)
[43] Eessaar, E. Teema 12. Andmebaasi füüsiline disain. [WWW] http://maurus.ttu.ee/ained/IDU0220_2012/doc/4/Teema_IDU0220_12_2012.pdf (22.04.2015)
[44] UML Core Elements. [WWW] http://www.uml-diagrams.org/uml-core.html (22.04.2015)
[45] Bellekens, G. Testing and debugging your Enterprise Architect C# Add-In. [WWW] http://bellekens.com/2011/02/08/testing-and-debugging-your-enterprise-architect-csharp-add-in (22.04.2015)
[46] Wikipedia. COM Interop. [WWW] http://en.wikipedia.org/wiki/COM_Interop (22.04.2015)
[47] Tiobe Software: TIOBE Index for April 2015. [WWW] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (22.04.2015)
[48] Regasm.exe (Assembly Registration Tool). [WWW] https://msdn.microsoft.com/en-us/library/tzat5yw6%28v=vs.110%29.aspx (25.05.2015)
[49] Sparx Systems. The Add-In Manager. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12.0/automation_and_scripting/addinmanager.html (23.05.2015)
[50] Microsoft TechNet. Command-Line Syntax Key. [WWW] https://technet.microsoft.com/en-us/library/cc771080.aspx (22.04.2015)
[51] Sparx Systems. Cross References. [WWW] http://www.sparxsystems.com/enterprise_architect_user_guide/12/model_transformation/crossreferences2.html (22.05.2015)

Lisa 1. SQLi reserveeritud võtmesõnad

CALL	HOUR	PARTITION	TABLE
CHECK	IDENTITY	PERIOD	TIME
COLUMN	LANGUAGE	POSITION	TIMESTAMP
CONDITION	LOCALTIME	RANK	UNION
COUNT	LOCALTIMESTAMP	RELEASE	VALUE
DATE	MEMBER	RESULT	WINDOW
DAY	MINUTE	SECOND	XML
ELEMENT	MODULE	SET	XMLDOCUMENT
GROUP	MONTH	SUM	YEAR
GROUPS	ORDER	SYSTEM	

Lisa 2. Teisendusmallid

Allpool olevas tabelis on loetletud kõik käesolevas töös parandatud või lisatud teisendusmallid koos viidetega täiendustele, mida iga mall realiseerib. Uued mallid, mis olid lisatud töösse, on märgitud tärniga (*).

Malli nimetus	**Seotud täiendused**
File	–
Namespace	**G2**
Class	**PK1, PK2, PK3, N1, N4, N5, N6, N7, N8**
Attribute	**CLC1, CLC2, CLC3, N4, N5, N6, N7, N8**
Connector	**PK4, FK1, FK2, FK4, FK6, FK7, FK8, N2, N3, N4, N5, N8**
Attribute__Multivalue*	**FK1, N4, N5, N6, N8, G3**
Connector__Generalization*	**FK1, G1**
Connector__AssociationClass*	**FK1, FK3**
Connector__ForeignKeyColumns*	**FK5, FK6, FK7, FK8**
Connector__InheritedAttributes*	**G1**
Class__Abstract_Attributes*	**G1**
Class__Abstract_Connectors*	**G1**
Class__Singleton*	**G4**
Class__IsSingleton*	**G4**

Lisa 3. Lisamooduli funktsioonid

Allpool on toodud kõikide töö käigus valminud lisamooduli funktsioonide kirjeldused. Iga funktsiooni kirjeldus on esitatud järgmises vormis (kasutades notatsiooni [50]):

<Funktsiooni nimi> ([@<parameetri nimi>, ...]) **[*seotud täiendused**]**

<parameetri nimi>	<parameetri kirjeldus>
...	...

<Funktsiooni kirjeldus, s. h. tagastatav väärtus>

[Seotud vahekeele elemendid ja omadused: `Element{omadus}`.]

---

seotud täiendused* on formaadis: {~** | <**täienduse kood**> [, ...]}, kus:

~ (*tilde*)	tähendab, et antud funktsioon on üldine ning kasutatakse enamustes teisendustes,
täienduse kood	alapeatükis 4.2 kirjeldatud täienduse kood (nt. **N1**, **PK1**, jne)

Funktsioonide nimekiri

Initialize (@DefaultDatabase) **[~]**

DefaultDatabase	andmebaasisüsteem, mille jaoks koostatakse füüsiline andmemudel

Valmistab ette andmed, mida on vaja teiste funktsioonide korrektseks tööks (nt initsialiseerib andmetüüpide sõnastikku). Seda funktsiooni kutsutakse välja iga teisendamise alguses.

ConvertName (@OldName, @IsAttribute) **[N4, N5, N6, N7, N8]**

OldName	algmudeli atribuudi või klassi nimi
IsAttribute	näitab, kas parameetri `OldName` väärtusena edastatud nimi on atribuudi või klassi nimi. Juhul, kui see argument on kas null või tühi string "", loetakse `OldName` argumendi klassi nimena; mistahes muu väärtus tähendab, et `OldName` argument on atribuudi nimi.

Võtab identifikaatori ja viib selle kooskõlla SQL standardiga, kasutades täiendustes N4-N8 kirjeldatud teisendusreegleid. Tagastab standardiseeritud identifikaatori, mis sobib kasutamiseks tabeli või veeru nimena vahekeele failis.

Seotud vahekeele elemendid ja omadused: `Table{name}, Column{name}`.

ConvertDBDataType (@GenericDataType) **[~]**

GenericDataType algmudeli atribuudi üldtüüp

Võtab üldise andmetüübi ja leiab tüüpide sõnastikust konkreetse andmebaasisüsteemi andmetüübi. Tagastab andmetüübi, mis sobib kasutamiseks veeru tüübina vahekeele failis. Kui sõnastikus vastet ei ole, tagastab üldise andmetüübi.

Seotud vahekeele elemendid ja omadused: Column{type}.

GetPrimaryKeyColumnsString (@ClassGUID) **[PK1, N1]**

ClassGUID algmudeli klassi GUID

Võtab mudelist teisendatava olemitüübi (klassi) andmeid, sh selle eksemplare unikaalselt identifitseerivate atribuutide hulga ja koostab vastava tabeli primaarvõtme kitsenduse. Kui tegemist on üldistusseose alamtüübi tähistava klassiga, siis liigub mööda hierarhiat ülespoole, et tuletada kitsendust õige ülatüübi klassi andmetest. Kui alusklassis pole määratud olemitüübi unikaalselt identifitseerivate atribuutide kogumit, siis koostab sobiva veeru nimega surrogaatvõtme. Tagastab stringi, mis koosneb kõikidest primaarvõtme poolt hõlmatud veergude kirjeldustest, ning sobib kasutamiseks vahekeele failis primaarvõtme elemendina.

Seotud vahekeele elemendid ja omadused: PrimaryKey{}.

GetAssociationClassPrimaryKeyColumnsString (@ClassGUID) **[PK2]**

ClassGUID algmudeli sidemeklassi GUID

Võtab mudelist andmeid mõlemast klassist, mis on seotud teisendatava sidemeklassiga ja koostab teisendusel saadava sidetabeli primaarvõtme kitsenduse, kasutades mõlema klassi puhul tavalist primaarvõtme leidmise strateegiat (vt funktsiooni **GetPrimaryKeyColumnsString** kirjeldus). Tagastab stringi, mis koosneb sidetabeli kõikidest primaarvõtme poolt hõlmatud veergude kirjeldustest, ning sobib kasutamiseks vahekeele failis primaarvõtme elemendina.

Seotud vahekeele elemendid ja omadused: PrimaryKey{}.

GetNAryAssociationPrimaryKeyColumnsString (@ClassGUID) **[PK3]**

ClassGUID algmudeli n-aarse seoseklassi GUID

Võtab mudelist andmeid kõikidest klassidest, mis on seotud teisendatava n-aarse seoseklassiga ja koostab teisendusel saadava seosetabeli primaarvõtme kitsenduse, kasutades kõikide klasside puhul tavalist primaarvõtme leidmise strateegiat (vt funktsiooni **GetPrimaryKeyColumnsString** kirjeldus). Tagastab stringi, mis koosneb seosetabeli kõikidest primaarvõtme poolt hõlmatud veergude kirjeldustest, ning sobib kasutamiseks vahekeele failis primaarvõtme elemendina.

Seotud vahekeele elemendid ja omadused: PrimaryKey{}.

GetLinkTablePrimaryKeyColumnsString (@ConnectorGUID) **[PK4]**

ConnectorGUID — algmudeli klassidevahelise seose GUID. Eelduseks on, et seos realiseerib M:N seosetüüpi, mis on kirjeldatud alapeatükis 4.1.4.

Võtab mudelist andmeid mõlemast klassist, mis asuvad teisendatava seose otstes ja koostab teisendusel saadava vahetabeli primaarvõtme kitsenduse, kasutades mõlema klassi puhul tavalist primaarvõtme leidmise strateegiat (vt funktsiooni **GetPrimaryKeyColumnsString** kirjeldus). Tagastab stringi, mis koosneb vahetabeli kõikidest primaarvõtme poolt hõlmatud veergude kirjeldustest, ning sobib kasutamiseks vahekeele failis primaarvõtme elemendina.

Seotud vahekeele elemendid ja omadused: PrimaryKey{}.

GetForeignKeyColumnsStringWithConstrs (@ClassGUID, @Alias, @NotNull, @Unique)

[FK5, FK6, FK7, FK8, G3]

ClassGUID — algmudeli selle klassi GUID, mille vastava tabeli primaarvõtme poolt hõlmatud veergude kirjeldust on vaja kätte saada sellele viitava välisvõtme veergude kirjeldamiseks

Alias — alias veergude nimetamiseks. Sisuliselt on see rolli nimi, mis kirjeldab olemitüübi rolli seosetüübi kontekstis. Kui on antud, siis töötab järgmiselt. Kui välisvõti hõlmab ühte veergu, nimetab selle ümber aliaseks. Kui välisvõti hõlmab mitu veergu, kasutab aliast nende veergude nimede prefiksina.

NotNull — näitab, kas välisvõtme poolt hõlmatud veergudel peab olema deklareeritud NOT NULL kitsendus. Juhul, kui see argument on kas null või tühi string "", siis kitsendust ei lisata; mistahes muu väärtuse korral peab lisama NOT NULL kitsenduse.

Unique — kui välisvõti hõlmab ainult ühe veergu, näitab, kas sellel veerul peab olema deklareeritud UNIQUE kitsendus. Juhul, kui see argument on kas null või tühi string "", siis kitsendust ei lisata; mistahes muu väärtuse korral peab lisama veerule UNIQUE kitsenduse. Kui välisvõti hõlmab mitu veergu, siis selle argumendi väärtus ei oma tähtsust ja seda ignoreeritakse, sest sellisel juhul peaks tegelikult deklareerima tabelitaseme UNIQUE kitsenduse, mida Enterprise Architect-i vahekeele piirangute tõttu teha ei saa (vt selle kohta ka alapeatükk 4.3.3).

Koostab antud klassile vastava tabeli primaarvõtmele viitava välisvõtme veergude kirjelduse. Võtab vaadeldava tabeli primaarvõtme poolt hõlmatud veergude kirjelduse ja viimistleb seda kirjeldust, lisades veergudele vastavalt vajadusele NOT NULL ja/või UNIQUE kitsenduse. Töötab korrektselt ainult juhtudel, mil vaadeldava tabeli primaarvõtme leidmiseks on rakendatav funktsioon **GetPrimaryKeyColumnsString**. Näiteks, ei tööta õigesti juhul, kui argumendina edastatakse sidemeklassi GUIDi, mille puhul vastava tabeli primaarvõtme leidmiseks kasutatakse funktsiooni **GetAssociationClassPrimaryKeyColumnsString**.

Tagastab viimistletud stringi, mis sobib kasutamiseks tabelisse lisatud välisvõtme veergude kirjeldamiseks.

Seotud vahekeele elemendid ja omadused: `Table{}`.

`GetForeignKeyColumnsString` `(@ClassGUID, @Alias)` **[~]**

`ClassGUID` vt. **`GetForeignKeyColumnsStringWithConstrs`**

`Alias` vt. **`GetForeignKeyColumnsStringWithConstrs`**

Töötab samal põhimõttel nagu funktsioon **`GetForeignKeyColumnsStringWithConstrs`**, ainult et ei lisa veergudele kitsendusi. Tagastab stringi, mis sobib kasutamiseks tabelisse lisatud välisvõtme kitsenduse kirjeldamiseks.

Seotud vahekeele elemendid ja omadused: `ForeignKey{Source{}}`.

`MarkUniqueIfApplicable` `(@AttrGUID)` **[CLC1, PK1]**

`AttrGUID` algmudeli atribuudi GUID

Võtab mudelist andmeid atribuudist ja teeb kindlaks, kas peab deklareerima teisenduse tulemusena tekkivale veerule UNIQUE kitsenduse, või mitte. Otsuse tegemisel võtab vajadusel arvesse ka andmed atribuuti sisaldava klassi kuuluvusest mingi üldistusseoste hierarhiasse. Tagastab vastavalt otsusele vahekeele markeri „Unique“ või tühja stringi.

Seotud vahekeele elemendid ja omadused: `Column{}`.

`MarkNotNullIfApplicable` `(@AttrGUID)` **[CLC2, PK1]**

`AttrGUID` algmudeli atribuudi GUID

Võtab mudelist andmeid atribuudist ja teeb kindlaks, kas peab deklareerima teisendusel saadavale veerule NOT NULL kitsendust, või mitte. Otsuse tegemisel võtab vajadusel arvesse ka andmed atribuuti sisaldava klassi kuuluvusest mingi üldistusseose hierarhiasse. Tagastab vastavalt otsusele vahekeele markeri „NotNull“ või tühja stringi.

Seotud vahekeele elemendid ja omadused: `Column{}`.

`MarkUpdateDeleteCascadeIfApplicable` `(@ClassGUID, @ConnectorGUID)` **[FK1, FK2, G3]**

`ClassGUID` algmudeli selle klassi GUID, mille vastavale tabelile viitab vaadeldav välisvõti

`ConnectorGUID` algmudeli selle seose GUID, mille alusel luuakse välivõti. Juhul, kui see argument on kas null või tühi string "", siis seda interpreteeritakse nii, et vaadeldav välisvõti tekib mitmeväärtuselise atribuudi teisendamise alusel, mille korral välisvõtmega peab alati seostama kompenseeriv tegevus ON DELETE CASCADE.

Võtab mudelist andmeid klassist ja seosest ja teeb kindlaks, kas vaadeldava välisvõtmega peavad olema seotud kompenseerivad tegevused ON UPDATE CASCADE ja/või ON DELETE CASCADE, või mitte. Otsuste tegemisel lähtub reeglitest, mis on kirjeldatud täiendustes FK1, FK2, G3. Tagastab vastavalt tehtud valikutele ühe järgmistest vahekeele omadus-väärtus paaridest:

cascade="update", cascade="delete", cascade="update,delete"

või tühja stringi.

Seotud vahekeele elemendid ja omadused: `ForeignKey{cascade}`.

`ChooseFKTable` `(@ConnectorGUID)` **[FK7, FK8]**

`ConnectorGUID` algmudeli klasside vahelise seose GUID. Eelduseks on, et teisendatav seos realiseerib 1:1 seosetüübi 0..1 / 0..1 või 1 / 1 varianti. Seda on kirjeldatud alapeatükis 4.1.6.

Kuvab dialoogiakna, mis laseb valida tabeli (klassi), kuhu paigutada välisvõtme veerud ja kitsendus. Jätab kasutaja valiku meelde. Tagastab valitud klassi GUID stringi.

`GetLastChosenFKTableGUID` `(@ConnectorGUID)` **[FK7, FK8]**

`ConnectorGUID` algmudeli klassidevahelise seose GUID. Vt. ka funktsiooni **`ChooseFKTable`** kirjeldus.

Kui vaadeldav seos oli juba mudeli teisendamise käigus kord teisendatud (nagu oli juba mainitud, iga seos teisendatakse kaks korda) ja kasutaja tegi valiku, kuhu paigutada välisvõti, siis tagastab valitud klassi GUID stringi. Vastasel juhul tagastab tühja stringi.

`StoreCurrentTable` `(@CurrentClassGUID)` **[G1]**

`CurrentClassGUID` viimasena teisendamiseks võetud algmudeli klassi GUID

Jätab meelde hetkel teisendatava klassi GUIDi. Seda kasutab funktsioon **`ProcessAbstractConnectors`** üldistusseoste hierarhia teisendamisel.

`ProcessAbstractConnectors` `(@ConnectorsString, @AbstractClassGUID)` **[G1]**

`ConnectorsString` vahekeele välisvõtmete kirjeldustest koosnev sõnaline väärtus, mida on vaja töödelda

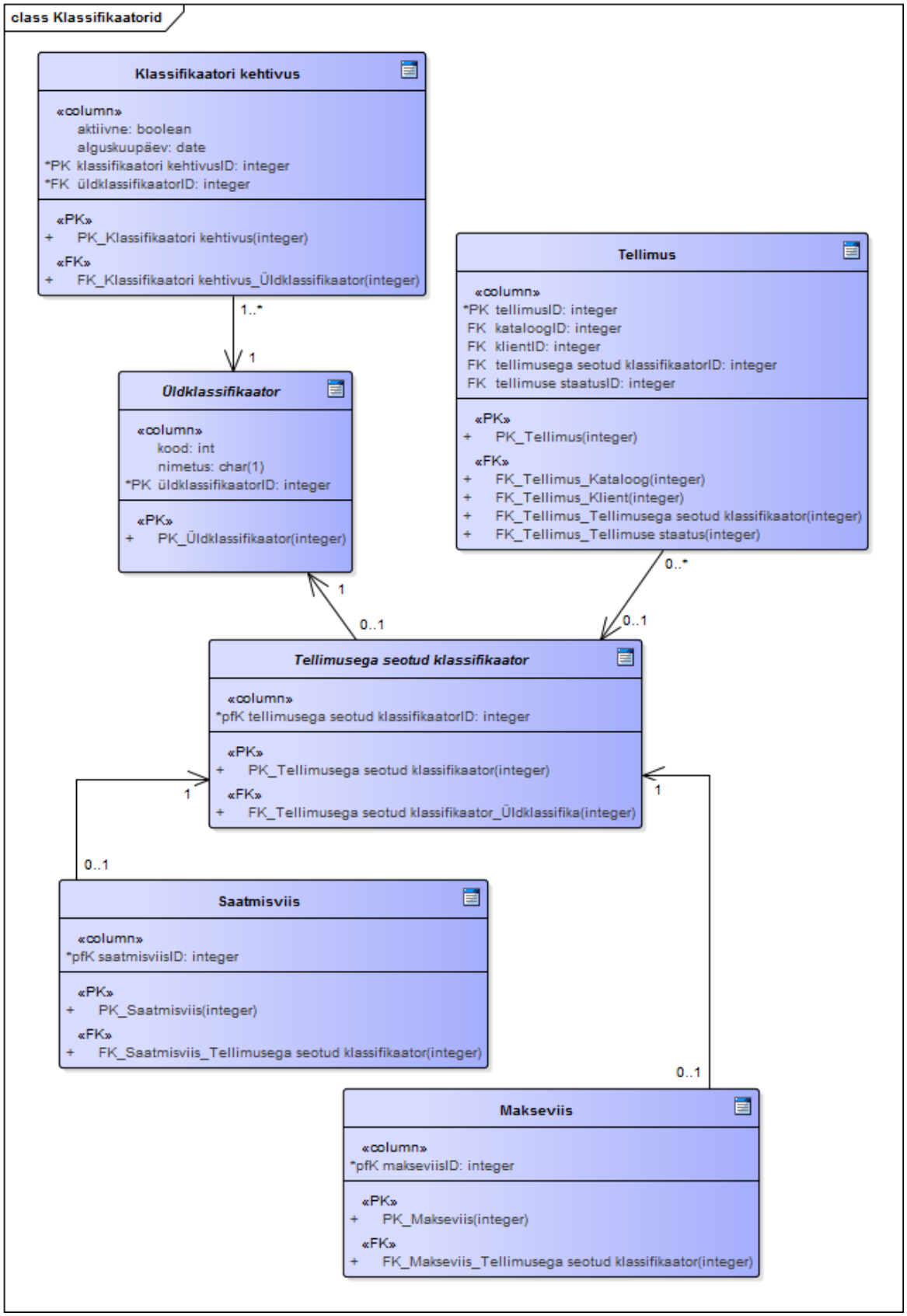
`AbstractClassGUID` abstraktse klassi GUID, mis tuleb asendada `ConnectorsString` argumendis hetkel teisendatava klassi GUIDi väärtusega

Abifunktsioon, mida kasutatakse {Mandatory; Or} üldistusseose korral ülatüübi seoste *copy-down inheritance* realiseerimisel. Asendab etteantud välisvõtmete kirjeldustest koosnevas stringis kõik GUIDid, mis langevad kokku etteantud abstraktse klassi GUIDiga, hetkel teisendatava klassi GUIDi väärtusega. Samuti genereeritakse iga välisvõtme kirjelduse jaoks

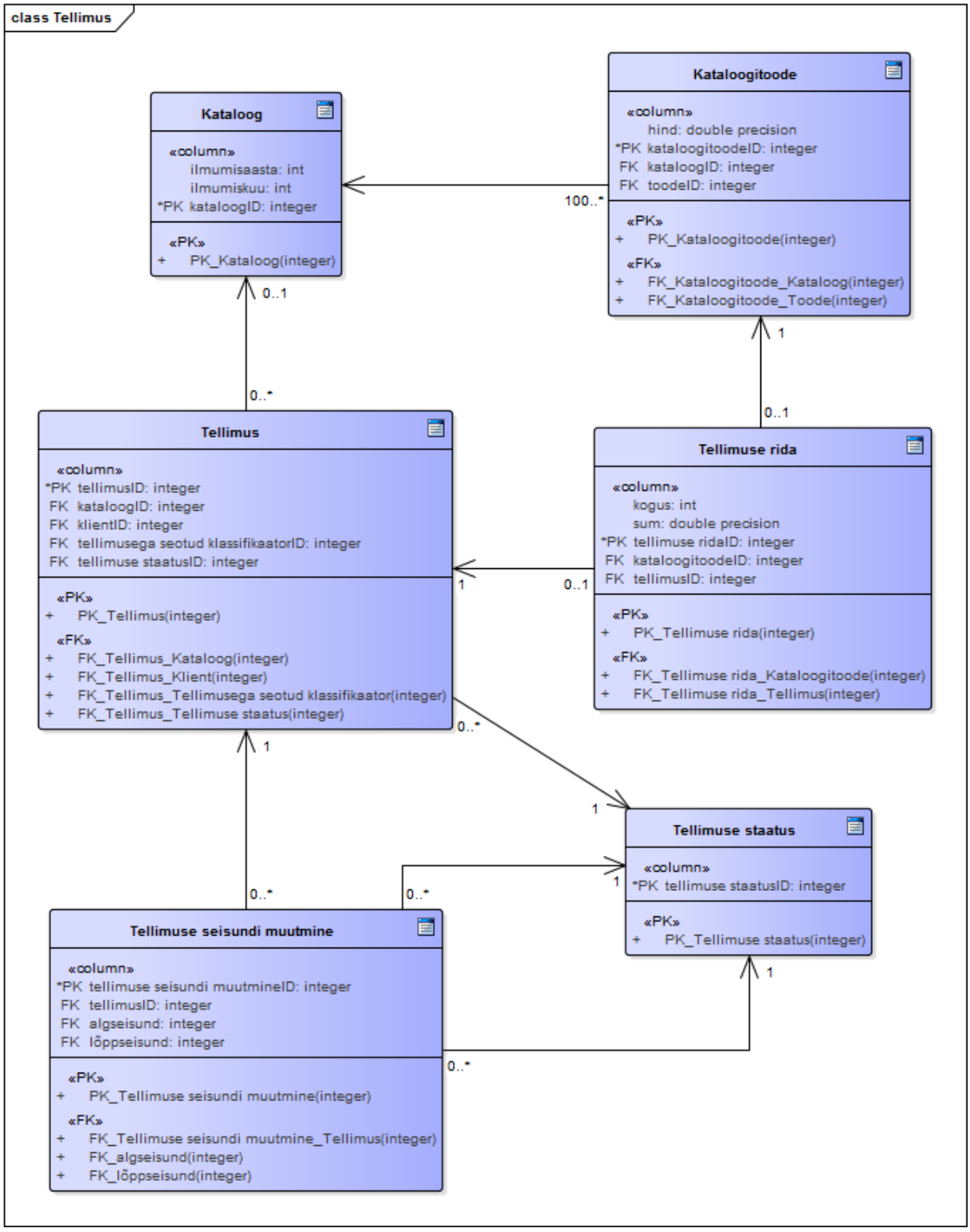
uus XRef element [51], et tagada uue välisvõtme loomine. Funktsiooni väljundiks on string, mis kujutab ennast päritud välisvõtmete kirjeldust vahekeeles.

Seotud vahekeele elemendid ja omadused: `ForeignKey{}`.

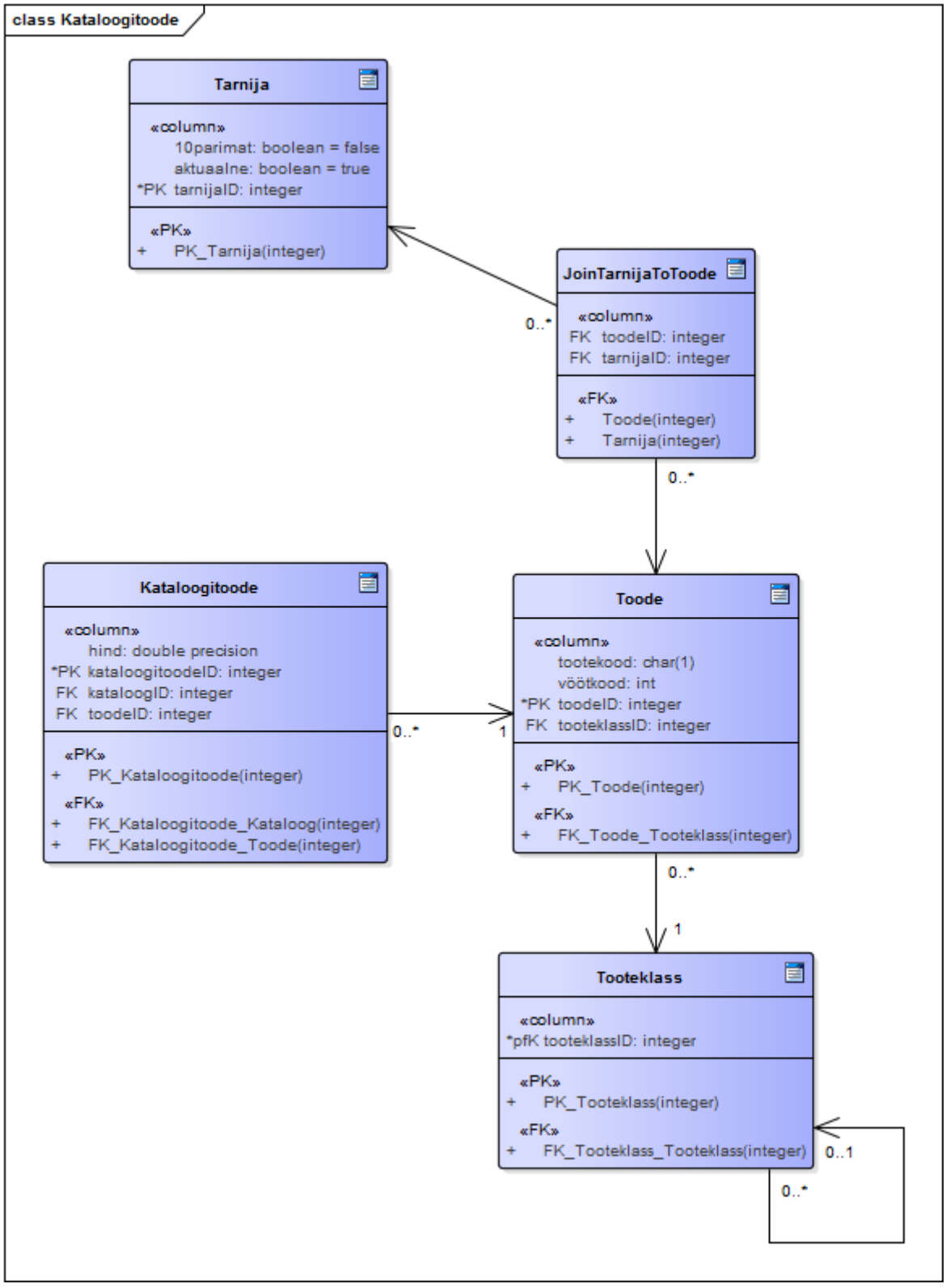
Lisa 4. Vana teisenduse abil genereeritud mudel



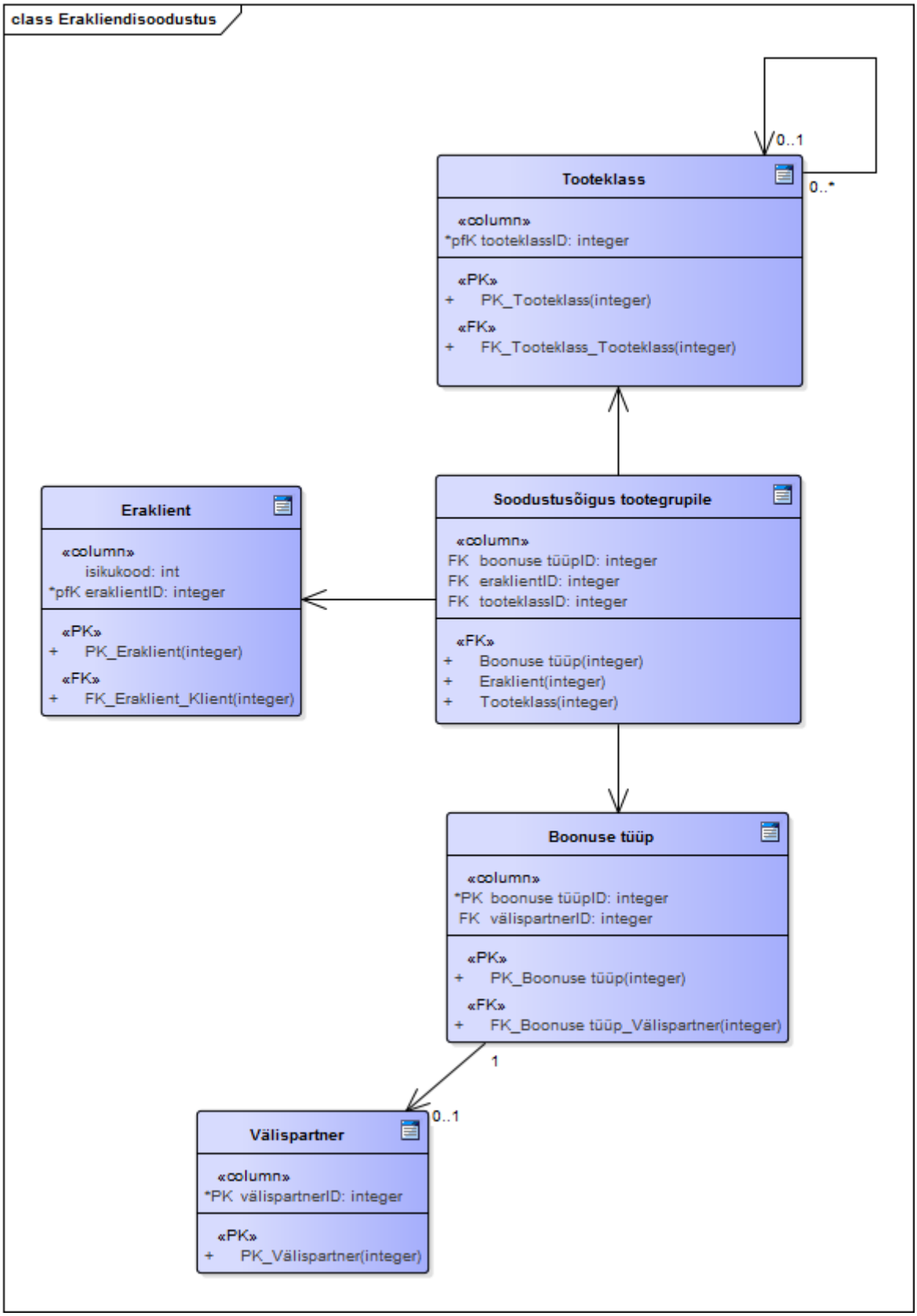
Joonis 27. Teisendatud mudel (vana teisendus). Klassifikaatorid



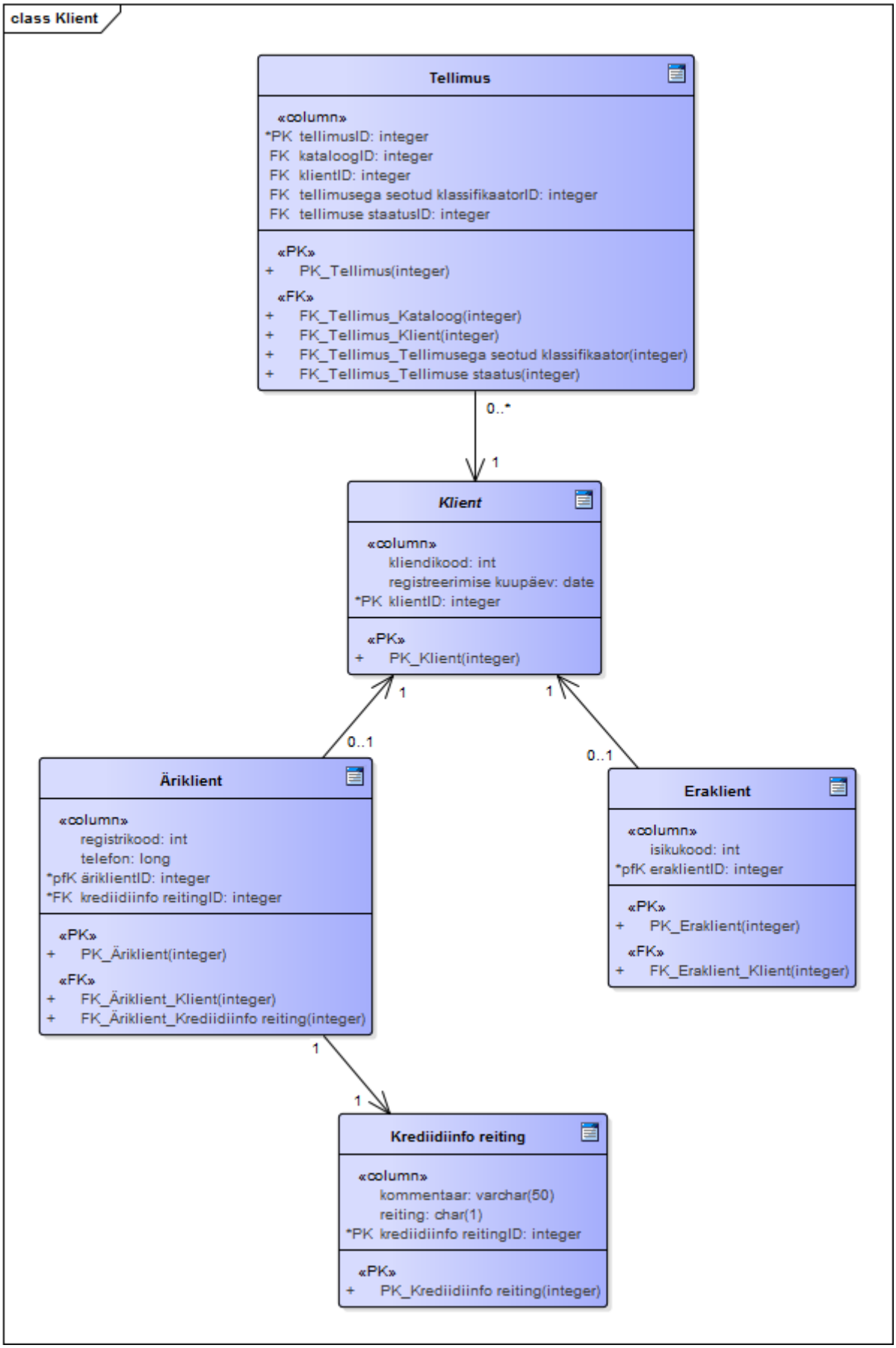
Joonis 28. Teisendatud mudel (vana teisendus). Tellimus



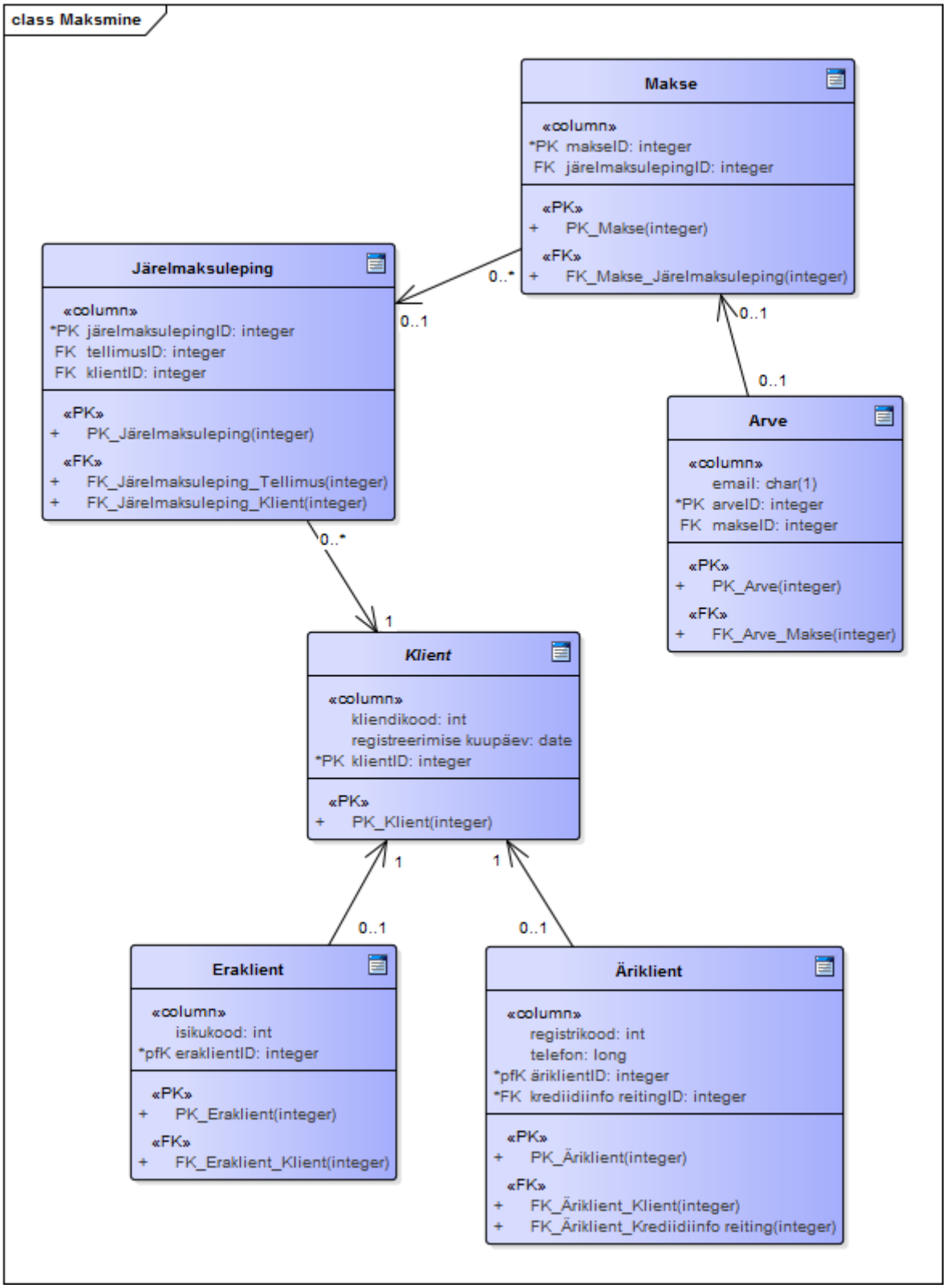
Joonis 29. Teisendatud mudel (vana teisendus). Kataloogitoode



Joonis 30. Teisendatud mudel (vana teisendus). Erakliendisoodustus



Joonis 31. Teisendatud mudel (vana teisendus). Klient



Joonis 32. Teisendatud mudel (vana teisendus). Maksmine