УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Мр Славица Алексић

Методе трансформација шема база података у обезбеђењу реинжењеринга информационих система

- ДОКТОРСКА ДИСЕРТАЦИЈА -

Ментор

др Иван Луковић, ред. проф.

Нови Сад, 2013.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Докторска дисертација
Аутор, **АУ**:		мр Славица Алексић
Ментор, **МН**:		др Иван Луковић, редовни професор
Наслов рада, **НР**:		Методе трансформација шема база података у обезбеђењу реинжењеринга информационих система
Језик публикације, **ЈП**:		Српски
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Србија
Уже географско подручје, **УГП**:		АП Војводина
Година, **ГО**:		2011
Издавач, **ИЗ**:		Факултет техничких наука
Место и адреса, **МА**:		Трг Д. Обрадовића 6, 21000 Нови Сад
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		
Научна област, **НО**:		Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:		Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, **ПО**:		Реинжењеринг база података, трансформације шема база података, MDSD / CASE алати, трансформације модела
УДК		
Чува се, **ЧУ**:		Библиотека Факултета техничких наука, Трг Д. Обрадовића 6, 21000 Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		Циљ истраживања реализованих у овом раду, био је је да се формално опишу могући приступи трансформацијама различитих описа база података из једног модела података у други и практично провере кроз њихову имплементацију у оквиру једног CASE алата, намењеног пројектовању информационих система и база података. У складу са постављеним циљем дефинисан је методолошки приступ и софтверско окружење IIS*Ree које омогућава висок ниво аутоматизације процеса реинжењеринга информационих система. Окружење IIS*Ree, засновано на MDSD принципима, као крајњи резултат генерише шему базе података у изабраном циљном, концептуалном или имплементационом моделу података, као и прототип апликације.
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	др Зора Арсовски, редовни професор
	Члан:	др Зоран Марјановић, редовни професор
	Члан:	др Соња Ристић, ванредни професор
	Члан:	Др Миро Говедарица, редовни професор
	Члан, ментор:	др Иван Луковић, редовни професор

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monograph documentation	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Ph.D. thesis	
Author, **AU**:		Slavica Aleksić, M.Sc.	
Mentor, **MN**:		Ivan Luković, Ph.D., Full Professor	
Title, **TI**:		Methods of Database Schema Transformations in Support of the Information System Reengineering Process	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Serbia	
Locality of publication, **LP**:		AP Vojvodina	
Publication year, **PY**:		2011	
Publisher, **PB**:		Faculty of Technical Sciences	
Publication place, **PP**:		Trg D. Obradovića 6, 21000 Novi Sad	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)			
Scientific field, **SF**:		Software engineering	
Scientific discipline, **SD**:		Applied computer science	
Subject/Key words, **S**/**KW**:		Database Reverse Engineering, Transformation of Database Scheme, IIS*Ree tool, MDSD / CASE tool, Model Transformation	
UC			
Holding data, **HD**:		Library of Faculty of Technical Sciences, 21000 Novi Sad, Trg Dositeja Obradovića 6	
Note, **N**:			
Abstract, **AB**:		The goal of the research presented in this paper is to formally specify approaches to transformation of database specifications between different data models. The transformations are then to be implemented and tested using a CASE tool for modelling information systems and databases. Following this goal, a methodological approach is defined together with a software named IIS*Ree that provides a high level of automation of the information system reengineering process. The IIS*Ree software, developed in accordance to MDSD principles, generates database schemas specified in a target, conceptual or implementation data model, as well as application prototypes.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Zora Arsovski, Ph.D., Full Professor	
	Member:	Zoran Marjanović, Ph.D., Full Professor	
	Member:	Sonja Ristić, Ph.D., Associate Professor	Menthor's sign
	Member:	Miro Govedarica, Ph.D., Full Professor	
	Member, Mentor:	Ivan Luković, Ph.D., Full Professor	

Sadržaj

1. Uvod

Savremeno poslovno okruženje karakterišu globalizacija, sve veća konkurencija i tehnološke inovacije, naročito na polju informacionih tehnologija. Veliki broj kompanija poseduje stare, nasleđene informacione sisteme (IS) koji, sami za sebe kao izolovana softverska rešenja, često postaju neodgovarajući, jer ne zadovoljavaju poslovne potrebe kompanije. Kompanije se, takođe, brzo menjaju ili ulaze u integracione procese, zbog čega su prinuđene da pristupe reinženjeringu informacionih sistema. Čak i one kompanije koje poseduju nove informacione sisteme suočene su sa pojavom novih tehnologija i u bliskoj budućnosti će biti prinuđene da sprovedu neku vrstu reinženjeringa.

Jak izazov u reinženjeringu informacionih sistema je taj što u kompanijama obično ne postoji koherentno znanje o poslovnim procesima, pogotovo ne dovoljno detaljno i dovoljno formalno dokumentovano i centralizovano, niti osoba koja njime u potpunosti vlada. Razvojni tim je suočen sa problemom otkrivanja znanja o poslovnim procesima, problemom otkrivanja potreba zainteresovanih strana (*stakeholder*-a) i problemom reprezentovanja pribavljenih informacija.

Postoji mnogo načina i pristupa koji se mogu primeniti za unapređenje procesa reinženjeringa informacionih sistema. Jedan broj pristupa reinženjeringu informacionih sistema, koji su od interesa za istraživanja u ovoj doktorskoj disertaciji, zasniva se na analizi i transformacijama opisa baza podataka jer se smatra da baze podataka predstavljaju veoma koristan izvor informacija vezanih kako za organizacioni sistem, tako i za sam postojeći informacioni sistem. Osim toga, u procesu reinženjeringa, kao i u procesu razvoja potpuno novog informacionog sistema, postoji i potreba za integracijom šema baza podataka, što takođe zahteva primenu metoda transformacija opisa baza podataka.

U novije vreme, u žiži interesovanja naučne i stručne javnosti nalazi se inženjerstvo upravljano modelima (*Model Driven Engineering* – MDE), pravac u kojem modeli, određenog stepena formalnosti, imaju glavnu ulogu u vođenju procesa razvoja informacionog sistema. Za razliku od pristupa zasnovanih na modelima (*model based*), koji se primenjuju već decenijama, a u kojima se model koristio za potrebe analize, boljeg razumevanja i dokumentovanje sistema, u MDE se model razmatranog sistema na dovoljno visokom nivou apstrakcije i sa zadovoljavajućim stepenom formalnosti tretira kao deo implementacije, jer se automatizovanim postupcima, kroz jednu ili više transformacija, prevodi u programski kod koji se direktno može izvršavati na izabranoj ciljnoj platformi [Stahl06].

MDE ima više predstavnika, od kojih je najpoznatija *The Model Driven Architecture* – MDA inicijativa, pod vođstvom *Object Management Group* – OMG grupe [MDA03]. MDA inicijativa definiše okvir za razvoj softvera zasnovan na modelima (*Model Driven Software Development* – MDSD), u kojem je osnovna pretpostavka postojanje mogućnosti automatske transformacije modela u druge modele i programski kod [Kleppe03].

U skladu sa MDA, koja definiše tri nivoa za modelovanje [MDA03], poslovni detalji i podaci iz problemskog domena modeluju se pomoću modela nezavisnih od računarstva (*Computation Independent Models*) – CIM. Za prezentovanje funkcionalnosti i strukture sistema, bez obzira na tehnološke detalje platforme na kojoj će biti imlpementiran, koriste se

modeli nezavisni od platforme (*Platform Indepedent Model*) – PIM. Nakon opisa sistema u okviru PIM-a koriste se alati koji na osnovu definisanih pravila za transformaciju generišu jedan ili više modela zavisnih od implementacione platforme (*Platform Specific Model*) – PSM. Korišćenjem različitih PSM-ova mogu se generisati razne implementacije istog sistema.

Ima puno literaturnih navoda u oblasti primene MDE na postupak projektovanja i implementacije informacionih sistema koji obuhvata tradicionalni proces kretanja od visokog nivoa apstrakcije i logičkog, implementaciono nezavisnog dizajna do fizičke implementacije sistema (*forward engineering*) [Vara09, Lano11, Lano13, Vara12, Haina06, Beziv03, Kuster05]. Takođe, u literaturi se može naći puno referenci u kojima se autori bave primenom *Model Driven* - MD pristupa na reverzni inženjering programskog koda [Favre10, Reus06, Ulrich10], ali malo pažnje je posvećeno reinženjeringu šema baza podataka korišćenjem MDE tehnika. Istraživanja sprovedena u okviru ove doktorske disertacije odnose se na moguće pristupe rešenju problema analize i transformacije šema baza podataka u postupku reinženjeringa informacionih sistema, koji obuhvata reverzni inženjering i *forward* inženjering [Chikof90]. Ideja je da se na sličan način kao u *forward* inženjeringu, u reverznom inženjeringu iskoriste pogodnosti MDE-a, kako bi se obezbedila automatizovana transformacija šema baza podataka na različitim nivoima apstrakcije. Za razliku od *forward* inženjeringa, kod kojeg je smer primene transformacije bio od modela koji ne zavise od platforme na kojoj će biti implementirani (nivo PIM), pa do modela koji su potpuno namenski za izabranu platformu (nivo PSM), u ovom slučaju kreće se od modela koji su namenski za platformu (nivo PSM), a zatim se nizom transformacija generišu šeme baze podataka, opisane na sve višem nivou apstrakcije. Krajnji cilj reverznog inženjeringa je konceptualni opis šeme baze podataka koji ne zavisi od platforme (nivo PIM). Nakon završenog postupka reverznog inženjeringa, nastala konceptualna šema baze podataka se može restruktuirati ili menjati, a zatim se nizom transformacija ponovo može prevesti u implementacioni opis šeme baze podataka za izabranu ciljnu platformu. Ovim bi proces reinženjeringa šeme baze podataka bio kompletiran.

Iz prethodno opisanog proizilazi opšta hipoteza postavljena u ovoj doktorskoj disertaciji:

H0. Primenom MD pristupa moguće je realizovati proces reinženjeringa informacionog sistema.

Glavni cilj istraživačkog rada direktno izveden iz postavljene hipoteze je da se definiše metodološki pristup i softversko okruženje, u kojem će se primenom MDE pristupa omogućiti poluautomatski reinženjering opisa baza podataka u cilju reinženjeringa informacionog sistema.

Iz opšte hipoteze proizilazi više hipoteza koje su navedene u tekstu koji sledi.

H1. Moguća je praktična provera valjanosti razvijenih transformacija kroz njihovu implementaciju u okviru jednog MDSD alata, namenjenog projektovanju informacionih sistema i baza podataka.

Model Driven Software Development (MDSD) alat bi morao da podrži mogućnost ugradnje algoritama za transformisanje opisa šema baza podataka jednog modela podataka u drugi i da obezbedi mehanizam za definisanje korisničkih zahteva i prihvat ulaznih specifikacija, na osnovu kojih bi kao krajnji rezultat bio generisani opis šeme baze podataka u izabranom ciljnom, konceptualnom ili implementacionom modelu podataka.

IIS*Studio je softversko okruženje koje, između ostalih funkcionalnosti, pruža podršku generisanju izvršnih softverskih specifikacija i specificiranju dizajna korisničkog interfejsa. Razvijan je i unapređivan tokom dužeg niza godina najvećim delom, na Fakultetu tehničkih

nauka Univerziteta u Novom Sadu. Najvažniji alat ovog razvojnog okruženja je *Integrated Information Systems CASE Tool* (IIS*Case) koji podržava konceptualno modelovanje šema baza podataka i generisanje programskih specifikacija. Njegov razvoj započet je 90-tih godina XX veka i inicijalno je predstavljao CASE (*Computer Aided Software Engineering*) alat, na šta ukazuje i njegov naziv. IIS*Case je tokom godina unapređivan u saglasnosti sa razvojem savremenih metoda i tehnika u oblasti softverskog inženjerstva. Ideje na kojima su zasnovane aktuelne verzije alata IIS*Case i IIS*Studio u značajnoj meri konvergiraju idejama na kojima počiva MD pristup.

Analogno MDSD i MDA alatima, projektovanje informacionog sistema u IIS*Case-u zasnovano je na dva tipa modela: onom koji ne zavisi od izabrane platforme i onom koji zavisi. Sistem se prvo modeluje na relativno visokom nivou apstrakcije pomoću posebnog koncepta tipa forme, pri čemu projektant kreira model sistema bez specificiranja detalja vezanih za konkretnu implementaciju. Zbog toga ovaj model može biti okarakterisan kao PIM. Ovaj PIM model predstavlja ulaz u niz model-u-model i model-u-kod transformacija, koje se primenjuju u IIS*Case-u radi dobijanja specifikacije prototipova aplikacija informacionih sistema. Specifikacija prototipa aplikacije uključuje detalje vezane za konkretnu platformu na kojoj će prototip biti implementiran, zbog čega se može okarakterisati kao PSM. Na ovaj način je u IIS*Case-u implementiran postupak *forward* inženjeringa u kojem se na osnovu PSM-a, nastalog kao rezultat transformacije, na izlazu dobija izvršiv programski kod aplikacije i SQL opis šeme baze podataka za izabranu ciljnu platformu, koja uključuje i konkretni sistem za upravljanje bazama podataka (SUBP).

Tip forme je centralni koncept na kojem počiva modelovanje putem IIS*Case-a. On predstavlja generalizaciju poslovnih dokumenata, a samim tim i ekranskih ili izveštajnih formi koje korisnik upotrebljava u svrhu komunikacije s informacionim sistemom. Pristup zasnovan na konceptu tipa forme podrazumeva da projektant, tokom modelovanja informacionog sistema, specificira tipove formi, posredno zadajući polazni skup ograničenja, koji će, tokom procesa generisanja, biti transformisan u implementacionu šemu baze podataka u trećoj normalnoj formi, na automatizovan način. Specifikacije tipova formi su polazna tačka i za generisanje izvršnih softverskih specifikacija. Osim elemenata koji implementiraju funkcionalnosti koje omogućavaju interakciju sa korisnikom kao i komunikaciju sa odgovarajućom bazom podataka, ove specifikacije uključuju i elemente koji specificiraju strukturu korisničkog interfejsa.

Za potrebe definicije izraza ograničenja na nivou konceptualne šeme baze podataka kao PIM modela, u okviru IIS*Case-a, razvijen je i jedan namenski jezik (*Domain Specific Language* - DSL) [Lukov10a]. Primenom ovog DSL-a, IIS*Case omogućava specificiranje regularnih izraza ograničenja koristeći problemski orijentisane koncepte na grafički orijentisan način. Primena DSL pristupa u implementaciji programske logike IIS*Case-a se u potpunosti uklapa u koncepte MDSD pristupa razvoju informacionih sistema.

Koncepti i metode na kojima je IIS*Case zasnovan, kao i implementirani algoritmi opisani su, između ostalih, u radovima: [Lukov07, Lukov93, Lukov06, Aleks13, Aleks07, Aleks06, Banov10, Popov08, Obren12, Ristic07].

Uvođenje novih koncepata kao i unapređivanje postojećih, ključni su elementi aktuelnog razvoja IIS*Case, koji je usmeren ka primeni savremenih metoda softverskog inženjerstva. Budući razvoj alata IIS*Case okrenut je ka daljoj formalizaciji procesa razvoja informacionih sistema i daljem povezivanju sa MD pristupima, što je u okviru istraživanja u ovoj disertaciji dalo motiv za sledeću hipotezu:

*H2. Moguće je postojeći pristup i alat IIS*Case, u kojem je već implementiran proces forward inženjeringa, integrisati s novim pristupom i alatom u kojem bi bio implementiran proces reverznog inženjeringa. Time bi bio kompletiran proces reinženjeringa šema baza podataka.*

Da bi ova hipoteza mogla biti opravdana, potrebno je da se razvije alat kojim bi projektantu bilo omogućeno da na automatizovan način, izvrši transformaciju postojećeg modela podataka u drugi, da ispita formalnu korektnost i integriše različite šeme baze podataka i druge opise izvora podataka u jednu šemu baze podataka. Kao krajnji rezultat, ova šema baze podataka bila bi predstavljena u formi relacione šeme ili *eXtensible Markup Language* (XML) specifikacije strukture baze podataka. Alat bi, na taj način, dao projektantima mogućnost da, u procesu reinženjeirnga, generišu implementacioni opis inovirane i integrisane šeme baze podataka projektovanog informacionog sistema.

U skladu s prethodno rečenim, predložena je izgradnja MDSD alata *Integrated Information Systems Reenginering Tool* (IIS*Ree). Alat IIS*Ree je namenjen da obezbedi postupak reinženjeringa koji obuhvata reverzni inženjering, restrukturiranje, kao i *forward* inženjering šema baza podataka. IIS*Case, u kojem je podržan *forward* inženjering, biće inkorporiran u IIS*Ree, tako što će koristiti zajednički repozitorijum za čuvanje modela, čime je omogućeno povezivanje rezultata nastalih u procesu reverznog i *forward* inženjeringa.

U postupku reverznog inženjeringa šeme baze podataka, koja je u skladu sa modelom podataka namenskim za platformu na kojoj je baza podataka bila implementirana, nizom model-u-model transformacija prevodi se u PIM konceptualnu šemu. Po principima MDA, transformacije su zasnovane na meta-modelima. U skladu sa tim postavljena je sledeća hipoteza:

H3. Algoritmima za transformacije, zasnovanim na meta-modelima, moguće je sprovoditi reverzni inženjering šema baza podataka.

Kako bi navedena hipoteza bila opravdana, postavljen je cilj da se obezbedi specifikacija sledećih meta-modela koji obuhvataju koncepte za modelovanje šema baza podataka na različitim nivoima apstrakcije:

- specifikacija meta-modela relacione šeme baze podataka na PSM nivou, zasnovanog na standardu sa kojim je u solidnoj meri kompatiblna većina RSUBP-ova ([SQL:1999, SQL:2003, SQL:2011]). U daljem tekstu za ovaj meta-model biće korišćen termin RSUBP meta-model,
- specifikacija generičkog meta-modela relacione šeme baze podataka koji je u skladu sa teoretskim definicijama relacionog modela podataka ([Date06, Elmas11 i Mogin04]). U daljem tekstu za ovaj meta-model biće korišćen termin Generički meta-model. Opis relacione šeme baze podataka koji je u ovom slučaju, takođe, na PSM nivou, ali sa većim stepenom nezavisnosti od implementacione platforme i
- specifikacija meta-modela PIM koncepata alata IIS*Case kojima se opisuje konceptualna šema baze podataka putem koncepata tipa forme, koja predstavlja rezultat postupka reverznog inženjeringa. U daljem tekstu za ovaj meta-model biće korišćen termin IISCase meta-model.

Specifikacije prva dva meta-modela su rezultat istraživanja u ovoj doktorskoj tezi, dok je je specifikacija trećeg meta-modela rezultat istraživanja koje će biti prezentovano u drugoj doktorskoj tezi.

Uspešnost i brzina reinženjeringa informacionog sistema u mnogome zavisi od reinženjeringa šema baza podataka. Zbog toga je potrebno razviti formalne postupke i

algoritme, putem kojih je moguće automatizovano, uz što manju interakciju sa korisnikom, vršiti transformaciju opisa baza podataka iz jednog modela podataka u drugi. Pri tome, postoji potreba za očuvanjem semantike postojećeg modela i rezrešenjem kolizija i dvosmislenosti u izražavanju semantike. U tom cilju potrebno je da se obrati pažnja da mogući pristupi uključe postupke koji bi ovo obezbedili.

U okviru istraživanja u ovoj doktorskoj disertaciji, kao nasleđeni izvori podataka (*legacy data*), razmatrane su baze podataka zasnovane na relacionom modelu podataka, implementirane na sistemima za upravljanje bazama podataka koji poseduju rečnike podataka. Podrazumeva se da rečnici podataka sadrže osnovne podatke o šemama relacija, njihovim atributima, primarnim ključevima, ograničenjima jedinstvenosti i *check* ograničenjima, kao i ograničenjima stranog ključa.

Reverzni inženjering šeme baze podataka može se podeliti u dve faze sa odgovarajućim redosledom izvršavanja. Prva faza je ekstrakcija strukture baze podataka (modela) iz postojeće baze podataka, a druga predstavlja konceptualizaciju ekstrahovane strukture [Haina96]. Pod konceptualizacijom se podrazumeva niz model-u-model transformacija čijom se primenom, na ekstrahovani model, kao rezultat dobija konceptualna šema baze podataka.

U skladu sa navedenim fazama, formulisana je sledeća hipoteza:

H4. Ekstrakciju modela moguće je automatizovati uz minimalnu interakciju sa korisnikom-projektantom, dok je konceptualizaciju modela moguće potpuno automatizovati.

Da bi ova pretpostavka mogla biti opravdana, postavljeni su sledeći ciljevi:

- Obezbediti ekstrakciju modela iz postojeće, nasleđene baze podataka u okviru koje treba:
  - obezbediti mehanizme za čitanje rečnika podataka i samih podataka postojeće relacione baze podataka koja je predmet reinženjeringa, u cilju dobijanja reprezentacije implementacione šeme baze podataka nezavisne od konkretnog proizvođača čiju je specifikaciju potrebno zapisati u repozitorijum okruženja IIS*Studio,
  - kreirati algoritme za razrešenje kolizija u imenovanju atributa (traženje homonima i eventualno preimenovanje) i
  - obezbediti ugradnju algoritma za detekciju potencijalnih ograničenja inverznih referencijalnih integriteta.
- Formulisati sledeće specificikacije transformacija koje učestvuju u automatizovanom reverznom inženjeringu šema baza podataka:
  - specifikaciju transformacije modela relacione baze podataka, koji je u skladu sa RSUBP meta-modelom, u model relacione baze podataka koji je u skladu sa Generičkim meta-modelom i
  - specifikaciju transformacije modela koji predstavlja izlaz prethodno specificirane transformacije u konceptualni model koji je u skladu sa IISCase meta-modelom.

Imajući u vidu da se pokazuje da je kompleksnost postupka transformacije modela zavisnog od implementacione platforme u konceptualni model visoka, da je broj koncepata, veza i osobina u meta-modelima na kojima su zasnovane transformacije takođe veliki, sledi da je neophodno podeliti proces transformacije u dva koraka, odnosno u dve transformacije. U prvoj, model zavisan od implementacione platforme transformiše se u semantički bogatiji model koji svojim konceptima približava polazni model krajnjem cilju, odnosno konceptualnom modelu. Dobijeni model u sledećem koraku transformiše se u konceptualni model tipova formi.

Sledeća hipoteza odnosi se na način implementacije automatizovanog procesa konceptualizacije modela:

H5. Automatizovani proces transformacije modela u cilju konceptualizacije ekstrahovanog modela iz postojeće, nasleđene baze podataka moguće je implementirati pomoću MDA tehnologija.

Kako bi ova hipoteza bila opravdana, izabrani su za implementaciju transformacija okruženje *Eclipse Modeling Framework* (EMF) i *Atlas Transformation Language* (ATL). ATL je namenski jezik za transformacije modela. EMF okruženje je izabrano jer je trenutno najpoznatije i veoma često upotrebljavano okruženje kada je u pitanju implementacija specifikacija definisanih u okviru MDA. ATL je izabran jer je, po saznanjima autora, najpoznatija implementacija jezika QVT, koji je propisan od strane OMG MDA standarda.

Zbog potrebe za interakcijom sa korisnikom i nemogućnosti da se prva faza reverznog inženjeringa potpuno automatizuje, u okviru ove doktorske disertacije doneta je odluka da se ova faza implementira pomoću programskog jezika opšte namene (*General Programming Language* - GPL). Izabran je Java programski jezik i okruženje Oracle JDeveloper u kojem su implementirani i ostali alati koji pripadaju okruženju IIS*Studio. Što se tiče druge faze reverznog inženjeringa, ideja je da se proces transformacije PSM-u-PIM potpuno automatizuje, u okviru posebnog modula, pomoću MDA tehnologija koje su podržane u *Eclipse Modeling Framework* okruženju i pomoću ATL. Usled korišćenja dva okruženja i različitih tehnika za implementaciju čitavog procesa reinženjeringa šema baza podataka, javlja se problem povezivanja modula koji se koriste u različitim fazama procesa.

U cilju prevazilaženja prethodno pomenutih problema, potrebno je obezbediti mehanizam koji omogućava razmenu podataka između dva okruženja. Pošto je XML opšte prihvaćen kao praktično upotrebljavan format za smeštanje podataka i upravljanje podacima, potrebno je modele ekstrahovane iz postojeće, nasleđene baze podataka, kao i modele koji će nastati kao rezultat reverznog inženjeringa, serijalizovati kao XML specifikacije [XML13]. U tom cilju potrebno je ispuniti sledeće zahteve:

- definisati XML šemu na osnovu koje će biti moguće model ekstrahovan iz postojeće, nasleđene baze podataka opisati pomoću XML specifikacije,
- obezbediti serijalizaciju modela u XML specifikacije, na osnovu ove XML šeme. XML specifikacija će opisivati model koji predstavlja ulaz u automatizovani proces transformacije,
- definisati XML šemu na osnovu koje će biti moguće konceptualni model nastao nizom model-u-model transformacija, opisati pomoću XML specifikacije,
- obezbediti parsiranje XML specifikacije nastale kao izlaz iz automatizovanog procesa transformacije, u skladu sa ovom XML šemom i
- obezbediti mehanizme za prevazilaženje problema različitih tehnoloških prostora, jer model nastao ekstrhovanjem iz nasleđene baze podataka, opisan XML specifikacijom, pripada XML tehnološkom prostoru, dok se transformacija izvršava nad modelima definisanim u *Meta Object Facility* (MOF) tehnološkom prostoru [MOF13].

U cilju obezbeđivanja poslednje navedenog zahteva, potrebno je specifikacije transformacija koje učestvuju u automatizovanom reverznom inženjeringu šema baza podataka proširiti sa još dve specifikacije:

- specifikacijom transformacije šeme baze podataka koja je u skladu sa XML meta-modelom u model relacione baze podataka koji je u skladu sa RSUBP meta-modelom,
- specifikacijom transformacije opisa konceptualne šeme baze podataka koja je u skladu sa IISCase meta-modelom u model koji je u skladu sa XML meta-modelom.

U cilju izgradnje alata IIS*Ree, namenjenog za reinženjering šema baza podataka, postavljeni su sledeći zahtevi:

- specifikacija arhitekture alata;
- specifikacija svih potrebnih meta-modela;
- implementacija editora za interakciju sa korisnikom;
- implementacija modula namenjenog za automatsku transformaciju modela koji uključuje implementaciju svih prethodno definisanih specifikacija transformacija u tačno definisanom redosledu. Potrebno je implementirati sledeće:
  1. skup pravila za transformaciju šeme baze podataka koja je u skladu sa XML meta-modelom u model koji je u skladu sa RSUBP meta-modelom,
  2. skup pravila za transformaciju modela nastalog u prethodnom koraku u model koji je u skladu sa Generičkim meta-modelom,
  3. skup pravila za transformaciju modela nastalog u prethodnom koraku u model koji predstavlja konceptualnu šemu baze podataka zasnovanu na PIM konceptima IIS*Case alata i
  4. skup pravila za transformaciju modela nastalog u prethodnom koraku u model koji je u skladu sa XML meta-modelom;
- implementacija modula koji će omogućiti serijalizaciju modela u XML specifikacije kao i parsiranje XML specifikacije radi ponovnog zapisa specifikacije modela u repozitorijum okruženja IIS*Studio;
- proširenje repozitorijuma okruženja IIS*Studio konceptima koji će obezbediti čuvanje meta-podataka iz nasleđene baze podataka na osnovu kojih se vrši ekstrakcija modela i omogućiti njihovu analizu radi unapređenja tog modela dodatnom semantikom.

Realizacijom alata IIS*Ree i oblikovanjem odgovarajućih pristupa, biće omogućena praktična primena procesa reverznog inženjeringa šema baza podataka, kao i povezivanje sa već implementiranim alatom IIS*Case putem kojeg se konceptualni model, nastao kao rezultat reverznog inženjeringa, može restruktuirati ili menjati, i u postupku *forward* inženjeringa ponovo nizom transformacija dovesti do implementacionog opisa šeme baze podataka za izabrani ciljni model podataka. Ukoliko je predmet reinženjeringa informacioni sistem koji obuhvata više baza podataka, one se mogu integrisati u jedinstvenu šemu baze podataka uz očuvanje uslova treće normalne forme, kao i detekciju i otklanjanje velikog broja formalnih kolizija ograničenja. Na osnovu generisane šeme baze podataka može se generisati prototip aplikacije inoviranog informacionog sistema. Ovim će proces reinženjeringa informacionog sistema biti zaokružen.

Korisnici-projektanti reinženjeringa informacionih sistema dobijaju razvojno okruženje u okviru kojeg, bez visoko formalnih, naprednih znanja i iskustva u oblasti reinženjeringa i projektovanja informacionih sistema, mogu u zadovoljavajuće kratkom vremenu, na efikasan način, da dođu do željenih rezultata: implementacione šeme baze podataka i funkcionalne aplikacije informacionog sistema.

Pored Uvoda, pisani elaborat ove doktorske teze sastoji se od šest poglavlja. U drugom poglavlju, pod nazivom *Pregled stanja u oblasti*, dat je pregled aktuelnih pristupa, koncepata i alata namenjenih reinženjeringu informacionih sistema. Naročita pažnja posvećena je onim pristupima koji se bave reinženjeringom baza podataka. Takođe, u ovom poglavlju diskutovani su MD pristupi koji su najviše uticali na formulisanje hipoteze. Prikazana su pozitivna iskustva i istaknute glavne razlike u odnosu na pristup koji je predložen u okviru ove doktorske teze.

U trećem poglavlju rada, pod nazivom *Principi primene MDA u reinženjeringu IS*, ukratko je opisan postupak reinženjeringa baze podataka, kao bitne komponente koja sadrži većinu informacija jednog IS-a, primenom MDA, odnosno *Architecture-Driven Modernization* (ADM) principa [ADM]. ADM je pradigma koju je, kao i MDA, ustanovila OMG grupa, koja

se odnosi na modernizaciju softvera. ADM nastoji da reši probleme tradicionalnog reinženjeringa primenom MD principa. Reinženjering baza podataka vrši se nizom model-u-model i model-u-kod transformacija, zasnovanih na meta-modelima. U ovom poglavlju data je i klasifikacija meta-modela šema baza podataka koje učestvuju u procesima transformacija.

U četvrtom poglavlju, pod nazivom *Pregled karakteristika i mogućnosti okruženja IIS*Studio*, opisuju se osnovne funkcionalnosti alata koji pripadaju razvojnom okruženju IIS*Studio: IIS*Case i IIS*UIModeler. Posebna pažnja posvećena je alatu IIS*Case, namenjenom za sprovođenje *forward* inženjeringa, odnosno namenjenom modelovanju informacionih sistema i generisanju funkcionalnih prototipova aplikacija. Napredna znanja iz oblasti upravljanja bazama podataka i poznavanje programskih jezika ili složenih okruženja za razvoj softvera, nisu strogi preduslov za korišćenje ovih alata. Praktičan i intuitivan korisnički interfejs ovih alata omogućava projektantima koji nisu stručnjaci iz oblasti informatike i računarskih nauka, da nakon kraće obuke mogu samostalno i u kratkom roku doći do funkcionalnog prototipa aplikacije. Specificiranjem polaznih ograničenja, pravila poslovanja, funkcionalnosti, interfejsa i različitih struktura, projektant započinje proces modelovanja informacionog sistema. Tokom tog procesa, projektant je podržan različitim alatima koji obezbjeđuju vođenu izradu specifikacija i automatizaciju određenih projektantskih aktivnosti. Krajnji rezultat ovog procesa predstavlja konkretna implementacija šeme baze podataka i generisane aplikacije koje se oslanjaju na datu šemu baze podataka i implementiraju željene funkcionalnosti. U poglavlju je opisano na koji način projektant upotrebom alata IIS*Case generiše šemu relacione baze podataka. Proces generisanja buduće šeme baze podataka informacionog sistema je iterativan. On uključuje i integrisanje šeme baze podataka datog aplikativnog sistema sa podšemama koje odgovaraju aplikativnim podsistemima datog sistema, kao i konsolidaciju integrisane šeme sa podšemama. Kroz taj proces, projektant kreira šablone korisničkog interfejsa upotrebom IIS*UIModeler-a, definiše dodatna poslovna pravila i, kao krajnji rezultat, generiše željene prototipove aplikativnih sistema.

U petom poglavlju, pod nazivom *IIS*Ree: pristup i softversko rešenje za MD reinženjering IS*, predstavljene su osnove funkcionalnosti i karakteristike alata nazvanog IIS*Ree, namenjenog da obezbedi transformacije šema baza podataka u procesu reinženjeringa informacionih sistema po MD principima. Opisani su metod i algoritmi ugrađeni u alat, kojima se obezbeđuje poluautomatizovano, uz minimalnu interakciju sa korisnikom, prevođenje opisa baza podataka u izabrani ciljni model. Kao ulazni model, za sada je podržan relacioni model, kao veoma često korišćen u praksi. U ovoj verziji, pomoću IIS*Ree može se vršiti reinženjering baza podataka implementiranih na sledećim SUBP-ovima: Oracle (verzije 9i, 10g i 11g) i MS SQL Server (verzije 2000, 2005, 2008 i 2012). Kao rezultat reinženjeringa, izlaz iz IIS*Ree može biti:

- konceptualni model podataka opisan PIM konceptima IISCase-a,
- implementacioni opis šeme baze podataka za različite relacione sisteme za upravljanje bazama podataka, ili
- XML specifikacija informacionog sistema (XML specifikacija tipova formi i XML specifikacija šeme baze podataka).

Alat može da generiše implementacioni opis relacionih baza podataka za konkretne proizvođače Oracle (verzije 9i, 10g i 11g) i MS SQL Server (verzije 2000, 2005, 2008 i 2012), kao i implementacioni opis zasnovan na ANSI SQL standardu.

U šestom poglavlju, pod nazivom *Meta-modeli šema baza podataka*, data je specifikacija svih meta-modela koji učestvuju u nizu transformacija. Biće prikazani sledeći meta-modeli:

- XML meta-model,
- RSUBP meta-model,
- Generički meta-model i

- IISCase meta-model.

U sedmom poglavlju, pod nazivom *Metode transformacija šema baza podataka modula M2M Transformer*, opisani su algoritmi, specifikacija i implementacija svih transformacija kojima se model, ekstrahovan iz postojeće relacione baze podataka, translira nizom uzastopnih transformacija u novi, konceptualni model alata IIS*Case. Ovaj modul je implementiran u skladu sa MDA pristupom u EMF okruženju. Transformacije su zasnovane na meta-modelima napravljenim pomoću Ecore jezika, koji je najpoznatija implementacija MOF-a, jezika koji pripada OMG standardu. Pravila za preslikavanje (mapiranje) kocepata jednog meta-modela u drugi implementirana su pomoću ATL jezika. U daljem tekstu će ravnopravno biti u upotrebi i izraz mapiranje, koji se na ovom govornom području uobičajeno koristi u domenu transformacija modela. Niz transformacija izvršava se potpuno automatizovano, bez interakcije sa korisnikom.

U okviru Zaključka rada data je diskusija hipoteza, izvedena iz dobijenih rezultata. Istaknuta su ograničenja predloženog rešenja i definisan je doprinos disertacije. Takođe, definisani su pravci budućeg istraživanja.

Na kraju disertacije nalazi se spisak korišćene literature, kao i potrebni prilozi. Prilog A sadrži kompletne XML šeme modula *XML Transformer*, dok je prilog B posvećen specifikaciji onog dela repozitorijuma alata IIS*Studio, koji se odnosi na čuvanje podataka vezanih za IIS*Ree. Spisak često korišćenih skraćenica, spisak slika i spisak listinga prikazani su, redom, u prilozima C, D i E.

Prilikom pisanja teksta ove doktorske disertacije, u svim situacijama u kojima je bilo moguće za dati engleski terimin ili frazu naći odgovarajući termin ili frazu na srpskom jeziku, to je i činjeno, uz paraleleno zadržavanje originalnog engleskog termina. U situacijama kada bi prevođenje na srpski jezik bitno odstupalo od duha jezika ili kada bi takvo prevođenje bitno narušilo semantiku termina, prevođenje na srpski jezik nije vršeno.

2. Pregled aktuelnog stanja u oblasti istraživanja

Postoje različiti razlozi koji podstiču poslovni sistem da krene u proces reinženjeringa, kao što su: pojačana konkurencija spremna da preuzme kupce, izostanak očekivanih efekata kod pojedinih aktivnosti, uprkos stalnom nastojanju da se one unaprede, izmena izvesnih zahteva i potreba pojedinačnih potrošača, napredak u tehnologiji i fundamentalnim procesima u industriji, ili previsoki troškovi poslovanja. Razvoj tehnologije, novih znanja i potreba vrše stalan pritisak na poslovne sisteme u pravcu uvođenja sofisticirane tehologije, jednostavnije za korišćenje, kako bi se uspešno reagovalo na strateške akcije konkurencije i razvoja pogodnih softvera kojima se olakšava upravljanje sistemom.

U ovom poglavlju dat je pregled aktuelnih koncepata, pristupa i alata iz oblasti reinženjeringa uz oslonac na *Model Driven Engineering* (MDE) i *Model Driven Architecture* (MDA). Takođe, definisani su osnovni pojmovi koji se koriste u ovoj doktorskoj disertaciji: koncept modela, meta-modela i transformacije modela.

2.1. MDE

U poslednjih deset godina glavne inovacije odnose se na integraciju i interoperabilnost između individualnih sistema. Uprkos napretku u programskim jezicima i pratećim integrisanim razvojnim okruženjima, razvoj kompleksnih softverskih sistema uobičajeno zahteva ogroman napor. Glavni razlog za teškoće u razvoju složenih sistema korišćenjem aktuelne kod-orijentisane (*code-centric*) tehnologije je konceptualni jaz između problema koji se rešava i implementacije. Ovo pokreće dva pitanja koja se mogu primetiti u tekućim realizacijama: razvojni proces može biti daleko od optimalnog i često postoji neslaganje između funkcionalnih potreba koje proističu iz domena problema i isporučenog softvera. *Model-driven* pristupi razvoju sistema pomakli su interesovanje sa programskih jezika treće genderacije (3GL) na modele. Cilj *Model-driven* razvoja je da se poveća produktivnost i smanji vreme ispruke softverskih proizvoda, tako što će omogućiti razvoj sistema na višem nivou apstrakcije, korišćenjem koncepata bližim domenu problema, umesto onih koje nude programski jezici.

Model-Driven Engineering [Beziv04, FavreJ05] je pristup razvoju koji nastoji da doprinese prevazilaženju navedenih problema korišćenjem apstrakcije, premošćavajući jaz između analize projektnih zahteva i konkretne implementacije sistema. To je paradigma koja se zasniva na upotrebi modela kao ključne komponente u razvoju softverskih sistema. Dominantno se upotrebljavaju modeli koji su formalni i mašinski čitljivi. Model više nije samo sredstvo za dokumentovanje i analizu sistema, već je deo implementacije sistema, u smislu da je moguće transformisati model u programski kod za ciljnu platformu. Ova ideja nije nova. Postoji analogija sa znatno zrelijom idejom prevođenja programa napisanih na jeziku treće generacije u asemblerski kod. Navedeni pristup ne predstavlja revolucionarni obrt u softverskom inženjerstvu, već je u pitanju evolutivni pomak napred koji je u skladu sa dugogodišnjim razvojem ove discipline.

2.1.1. Modeli i meta-modeli

Upotreba modela odavno je prepoznata kao dobra praksa tokom procesa razvoja informacionih sistema. Njihovoj primeni u razvoju softverskih sistema prethodi dugogodišnja praksa primene ovakvog pristupa u razvoju složenih hardverskih sistema. Korišćenje modela u procesu razvoja informacionog sistema prisutno je već decenijama. Međutim, sa pojavom metodologija i alata koji omogućavaju kreiranje softverskih modela na visoko apstraktnom nivou, pristup koji podrazumeva upotrebu modela kao ključne i nerazdvojive komponente razvojnog procesa, postao je jedan od vodećih pristupa u razvoju softverskih sistema.

U literaturi se može naći više definicija modela. U [MDA03] modeli se definišu na sledeći način: "*Modeli se sastoje od elemenata koji opisuju neku fizičku, apstraktnu ili hipotetičku realnost*."

U [Kleppe03], u kontekstu MDA, pod modelom se podrazumevaju sistem ili deo sistema zapisan korišćenjem dobro definisanog jezika koji podržava automatsku interpretaciju od strane računara.

Prema [MDA03] modeli se mogu klasifikovati u tri kategorije.

- Računarski nezavisni modeli (*Computation Independent Model* - CIM). Ovoj klasi pripadaju modeli koji opisuju sistem sa računarski nezavisnog stanovišta. Ovo su modeli visokog nivoa apstrakcije i često se za njihovo kreiranje koriste koncepti koji su karakteristični za domen primene. U kreiranju ovih modela poželjno je uključiti i domenske eksperte. Međutim, ovi modeli nisu pogodni za proces transformacije jer su često u pitanju neformalni modeli, kada je apstrakcioni jaz prevelik;
- Modeli nezavisni od platforme (*Platform-Independent Models* - PIM). Ovoj klasi pripadaju modeli kreirani sa stanovišta koje je nezavisno od platforme na kojoj će sistem biti implementiran;
- Modeli zavisni od platforme (*Platform-Specific Models* - PSM). Ovoj klasi pripadaju modeli koji uključuju informacije koje su karakteristične za ciljnu platformu na kojoj će sistem biti implementiran.

Jezik za specifikaciju modela sistema se naziva meta-model. Saglasno tome, model je instanca meta-modela u određenom domenu.

Prema [Stahl06] meta-model se definiše na sledeći način:

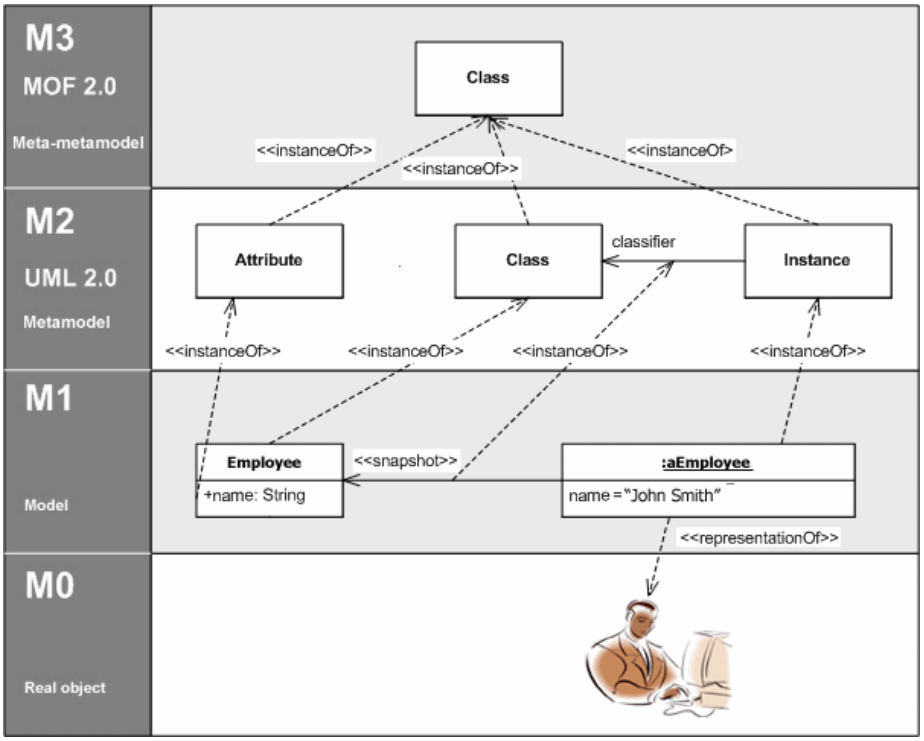
"*Meta-model opisuje strukturu modela i to na apstraktan način. On definiše osnovne konstrukte jezika za modelovanje, kao i veze, ograničenja i pravila. Pri tome se ne specificira konkretna sintaksa jezika za modelovanje.*"

Proces kreiranja meta-modela naziva se meta-modelovanje. Zadatak kreiranja meta-modela je pre svega definisanje semantike jezika za meta-modelovanje, dok se konkretna sintaksa ne mora specificirati.

Na tržištu postoje brojni jezici koji nude mogućnost meta-modelovanja. Neki od poznatijih jezika su prikazani u tabeli 1.

Tabela 1.1 Primeri poznatijih jezika namenjenih za meta-modelovanje

Jezik	Opis
MOF	Jezik uveden od strane OMG-a.
Ecore	Jedna od najpoznatijih implementacija MOF jezika.
EBNF	Jezik za definisanje sintakse programskih jezika.
GOPPRR	Jezik namenjen meta-modelovanju koji je podržan od strane alata MetaEdit+.

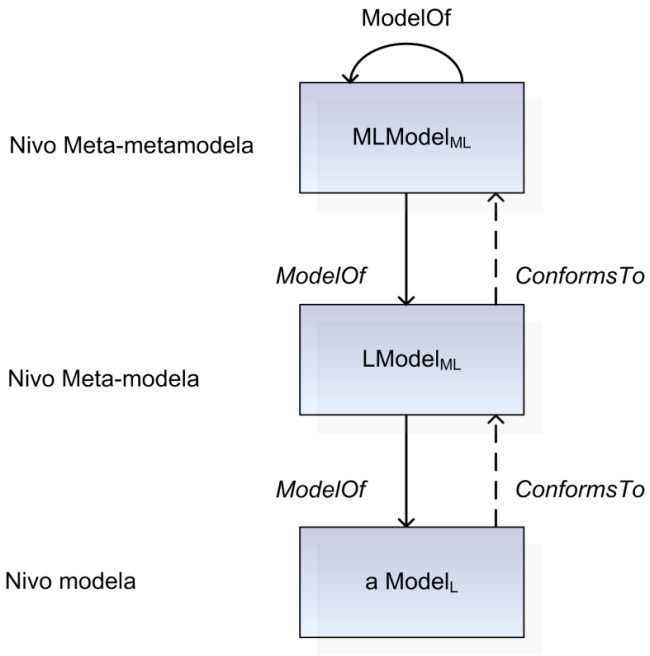


Slika 2.1. Ilustracija četvoroslojne arhitekture meta-nivoa [MDA03]

Meta-model se specificira pomoću meta-metamodela. Meta-metamodel, meta-model, model sistema i instanca modela formiraju četvoroslojnu arhitekturu meta-nivoa (slika 2.1) definisanu od strane OMG grupe [MDA03]:

- M0 – Nivo konkretnih podataka i objekata koji predstavljaju pojavu elemenata nivoa M1;
- M1 – Nivo modela, tj. predstavlja model na osnovu kojeg su kreirane instance sa nivoa M0. Na primer, ovom nivou pripada instanca neke UML klase. Sve instance ove klase pripadaju nivou M1;
- M2 – Nivo meta-modela. Na ovom nivou opisuje se jezik za izgradnju modela na nivou M1. Posmatrano u kontekstu UML-a, na ovom nivou definišu se koncepti kao što su *Class*, *Attribute*, itd. Na nivou M2 definiše se apstraktna sintaksa konkretnog jezika za modelovanje, kakav je UML;
- M3 – Nivo meta-metamodela. Služi za definisanje koncepata nivoa M2. Ovaj nivo služi za definisanje više jezika za meta-modelovanje. Jedan takav primer je UML meta-model. Ovaj nivo je samodefinišući, tj. nije potrebno uvoditi nivo M4 za opisivanje nivoa M3, već se koncepti nivoa M3 opisuju konceptima nivoa M3. Time se izbegava ulazak u problem postojanja beskonačno mnogo meta-nivoa.

Na sličan način, autor u [Kurtev05] prikazuje kako aktivnost meta-modelovanja može biti primenjivana za izgradnju hijerarhije modela koji se nalaze na različitim nivoima. Ovu organizaciju u više nivoa naziva arhitekturom meta-modelovanja. Na slici 2.2 prikazan je primer ove arhitekture.



Slika 2.2. Arhitektura meta-modelovanja [Kurtev05]

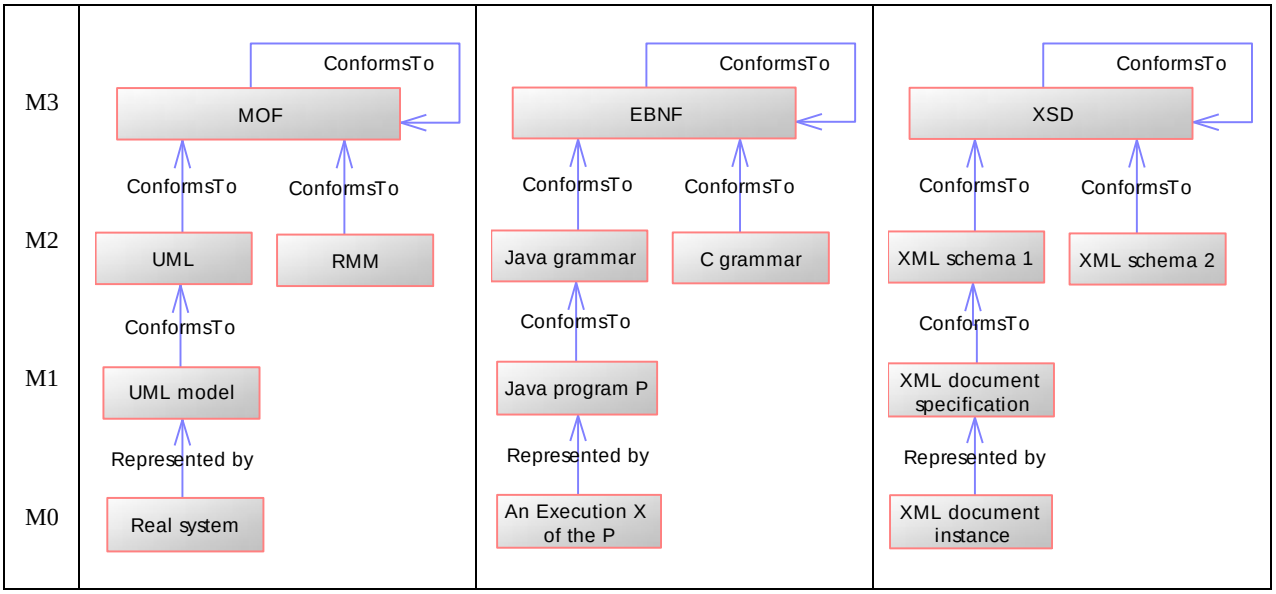
Relacija *ConformsTo* (u skladu sa) znači da model "poštuje" pravila definisana u meta-modelu [Kurtev05]. Na dnu ove arhitekture se nalaze modeli koji su predstavljeni putem različitih jezika za modelovanje. Ovaj nivo se naziva *nivo modela*. Kao primer modela na ovom nivou je $Model_L$ kreiran u jeziku za modelovanje sa nazivom *L*. Može se napraviti model za *L* (koji je meta-model) $LModel_{ML}$ predstavljen u drugom jeziku koji se zove meta-jezik (*meta-language*). Modeli jezika koji se koriste na nivou modela formiraju drugi nivo ove arhitekture, koji se naziva nivo meta-modela. Postoji *ModelOf* relacija između meta-modela jezika i modela predstavljenih u tom jeziku. Ovo se može primeniti i na modele na nivou meta-modela. Modeli jezika koji se koriste za izražavanje meta-modela se nalaze na trećem nivou, nivou meta-metamodela. Kod trećeg nivoa se *MLModel* izražava u samom *ML* jeziku. Kod ovako definisane arhitekture, nivo na vrhu je refleksivan, što znači da je ovaj nivo predstavljen u jeziku koji je modelovan od strane ovog modela. Na nivou meta-modela postoje modeli jezika za modelovanje predstavljeni u *ML*-u.

Ova arhitektura meta-modelovanja može se primeniti na više različitih tehnoloških prostora. Na slici 2.3 prikazani su primeri: MOF arhitekture, organizacije programskih jezika, veza između XML dokumenata i XML šema, respektivno. Svi primeri slede isti, prethodno opisani princip prikazan na slici 2.2. [Aksit02]

Svrstavanje modela u PIM ili PSM kategoriju zavisi od konteksta izabrane platforme [MDA03]. Na primer, šema baze podataka iskazana putem koncepata relacionog modela podataka može biti posmatrana kao model tipa PSM, jer se koristi relaciona tehnologija, a ne recimo objektno-relaciona, XML, ili neka druga tehnologija. S druge strane, može se reći da je u pitanju model tipa PIM, jer je ovakav model nezavisan od konkretnog sistema za upravljanje bazama podataka. Modeli tipa PSM kreiraju se na osnovu modela tipa PIM, najčešće putem alata koji automatizovano obavljaju transformacije. Svaka transformacija obavlja se na osnovu modela tipa PIM i dodatnih informacija koje su specifične za konkretnu platformu, u svrhu generisanja koda za ciljnu platformu.

Model kreiran putem alata IIS*Case može se svrstati u PIM kategoriju, jer tokom modelovanja projektant ne navodi detalje koji su vezani za neku konkretnu platformu [Lukov13]. Pored toga, podržane su transformacije u druge PSM modele. Na primer, na osnovu modela kreiranog putem koncepta tipa forme i pratećih koncepata moguće je

generisati implementacioni opis šeme baze podataka. Na sličan način, putem DSL jezika IIS*CfuncLang [Popov13], manenjenog za specificiranje netipičnih funkcionalnosti, i dalje se kreira model koji pripada PIM kategoriji, jer koncepti ovog jezika nisu vezani za neku konkretnu platformu. Podržana je i transformacija u druge PSM modele, kao što su pseudo-asemblerski kod i *PL/SQL* programski kod.



Slika 2.3. Primeri arhitekture meta-modelovanja za različite tehnološke prostore

2.1.2. MDA inicijativa

Krajem 2000. godine, organizacija *Object Management Group* (OMG) je predstavila *Model Driven Architecture*, arhitekturu za MD razvoj softvera, *Model-Driven Software Development*. MDA je jedan od pristupa koji daju modelima vodeću ulogu u razvojnom procesu [Kleppe03, Booch04, MDA03]. Upotreba ovog pristupa predstavlja mogući način da se umanji kompleksnost razvojnog procesa, da se postignu visoki nivoi ponovne upotrebljivosti postojećih programskih modula i značajno smanji napor uložen u projekte razvoja softvera.

MDA predstavlja takav pristup modelovanju informacionih sistema koji razdvaja specifikaciju funkcionalnosti i struktura sistema od specifikacije konkretne implementacije na izabranoj platformi. Specifikacije sistema opisane su modelima. Glavna karakteristika upotrebe modela u MDA pristupu je prenosivost, tj. mogućnost da jedan model može biti realizovan na različitim platformama, odnosno implementiran upotrebom različitih tehnologija. MDA koncept omogućava povezivanje modela različitih aplikativnih sistema, njihovu integraciju i interoperabilnost. Na taj način, razvoj apstraktnih specifikacija sistema postaje nezavisan od konkretnog izbora platforme, a samim tim i od stalnih promena i inovacija na polju informacionih tehnologija. Upotreba novih tehnologija za potrebe implementacije aplikativnog sistema ne zahteva dodatni napor za prilagođavanje specifikacija modela.

MDA je zasnovana na nizu specifikacija, poznatih pod nazivom OMG standardi [Mellor04]:

- *Unified Modeling Languange* (UML) – standardizovani jezik za dokumentovanje i specifikaciju softverskih sistema, [UML11],
- *Meta Object Facility* (MOF) – jezik namenjen specifikaciji meta-modela, [MOF11],

- XMI (*XML Metadata Interchange*) – format za razmenu modela putem XML dokumenata, [XMI11],
- *Query/View/Transformation* (QVT) – jezik za specifikaciju model-u-model transformacija između modela, [QVT09] i
- *Object Constraint Language* (OCL) – jezik namenjen definisanju izraza nad modelima, [OCL12].

U nastavku će biti dat kratak opis OMG standarda koji se koriste u ostatku teksta.

2.1.2.1. MOF

Specifikacija MOF-a smatra se najznačajnijom u okviru MDA pristupa. U [MOF11] MOF je definisan na sledeći način: "*Specifikacija MOF definiše apstraktni jezik i okruženje za specificiranje, kreiranje i održavanje meta-modela koji su nezavisni od konkretne platforme*."

Jezik UML brzo je postao jedan od najpopularnijih jezika za modelovanje različitih aspekata sistema. Radi se o opštem jeziku za modelovanje namenjenom širokom spektru domena. Ozbiljan nedostatak UML-a jeste činjenica da je zbog opšte prirode koncepata ovog jezika nemoguće izraziti specifičnosti konkretnog domena. Ovo naročito dolazi do izražaja kada za konkretan domen postoji specijalizovan jezik čiji koncepti su iz datog domena. Umesto nemogućeg zadatka kreiranja jezika primjenljivog na sve domene, OMG se orijentiše na kreiranje MOF-a kao mehanizma za formalno definisanje jezika za modelovanje (meta-modela) nad konkretnim domenima. MOF se koristi za modelovanje meta-modela, pa se MOF naziva i meta-metamodel. Putem MOF-a definisan je meta-model UML-a, kao i sam MOF. MOF definiše četvoroslojnu arhitekturu, opisanu u tački 2.1.1 i sam pripada nivou M3. Da bi neki jezik za modelovanje postao deo MDA pristupa nije potrebno menjati konkretnu sintaksu jezika, već je potrebno opisati njegovu apstraktnu sintaksu putem MOF-a.

Koncepti MOF-a pozajmljeni su iz modela klasa UML-a. U verziji UML-a 2.0 postoji usklađenost tog jezika i MOF-a, pa je moguće iskoristiti neki od alata za UML modelovanje, pa potom transformisati kreirani model u MOF model.

Postoji nekoliko implementacija MOF-a, međutim većina podržava samo deo ove specifikacije. Platforma *Eclipse Modeling Framework* (EMF) je najdalje otišla u ovom pravcu. Njihov Ecore meta-metamodel smatra se jednom od najuspelijih implementacija MOF-a, što je imalo uticaja na dalje specifikacije OMG-a. U skladu sa tim, OMG je definisao dva podskupa jezika MOF i to (i) *essential* MOF (EMOF) i (ii) *complete* MOF (CMOF). U poslednjoj verziji meta-metamodel Ecore je u potpunosti usklađen sa EMOF specifikacijom. Specifikacija CMOF je obimnija nego specifikacija EMOF i koristi se za složenije meta-modele kakav je UML.

Verifikacija rezultata nastalih u okviru ove doktorske teze obavlja se u okviru alata IIS*Ree, koji je deo integrisanog okruženja IIS*Studio. U ovom alatu meta-modeli koji se koriste u procesu transformacije, u cilju reinženjeringa informacionih sistema, opisani su putem MOF-a. Takođe, meta-model koji opisuje PIM koncepte alata IISCase, koji je takođe deo okruženja IIS*Studio, iskorišćen je za kreiranje konkretne sintakse jezika namenjenog za domen projektovanja informacionih sistema. Više o ovoj temi može se naći u [Čelik12].

2.1.2.2. XMI

OMG koristi XMI za definisanje, razmenu, manipulaciju i integraciju XML podataka i objekata [XMI11]. XMI je zasnovan na *Extensible Markup Language* (XML) [XML13]. XMI definiše pravila po kojima se može generisati šema za svaki validni XMI-prenosiv MOF meta-model. XMI omogućava transformaciju MOF specifikacija u XML specifikacije.

U pogledu opisivanja objekata putem XML-a XMI omogućava sledeće:

- predstavljanje objekata putem XML elemenata i atributa,
- kreiranje veza između objekata koji se nalaze u istom ili različitim fajlovima,
- postojanje jedinstvenih identifikatora koji omogućavaju međusobno referenciranje objekata i
- validiranje XMI dokumenata putem šeme.

Kada god je jezik za modelovanje opisan putem MOF-a, onda je definisan i format za razmenu modela (sa nivoa M1) kreiranih u tom jeziku. Često se XMI koristi za razmenu modela kreiranih putem UML-a.

Do sada je OMG objavila više verzija XMI-a. Poslednja verzija je 2.4.1. Verzija 2.0.1 je formalno objavljena kao ISO standard ISO/IEC 19503:2005 [XMI05].

Jedan modul, pod nazivom *M2M Transformer* alata IIS*Ree, implementiran je u EMF okruženju, dok su ostali moduli alata implementirani pomoću Java GPL-a kao i čitavo okruženje IIS*Studio. Korišćenje dva različita okruženja za implementaciju alata zahtevalo je razmenu podataka između modula. Za razmenu podataka korišćen je XMI standard.

2.1.2.3. OCL

OCL predstavlja formalni, tekstualni jezik koji omogućava precizno definisanje ograničenja i upita na bilo kojem MOF modelu. Iako je kreiran kao formalan jezik, OCL je lako čitati i pisati. U pitanju je jezik za kreiranje specifikacija, i zato nije moguće, kao kod programskih jezika, njime kreirati (programirati) poslovnu logiku sistema, pokrenuti proces ili aktivirati operaciju koja nije tipa upita. Ovo znači da OCL izraz ne može da utiče na stanje sistema, tj. ne može da promeni model uz koji je specificiran. [OCL12]

OCL je prvobitno kreiran kao deklarativni jezik za opisivanje pravila koja se primjenjuju nad UML modelima i u suštini je i sam predstavljao deo UML specifikacije, odnosno formalnu ekstenziju jezika UML. Danas se OCL primjenjuje na sve MOF meta-modele, uključujući i UML.

OCL je ključna komponenta jednog od najvažnijih standarda MDA pristupa - vezanog za transformacije modela. Transformacije modela biće detaljnije objašnjene u narednoj tački. U [Cari04] je predstavljen jedan pristup implementaciji transformacije modela koji uvodi pojam ugovora za transformaciju (originalni naziv na engleskom: *model transformation contracts*). Ugovorima je omogućeno specificiranje transformacije modela nezavisno od odabrane platforme. Specifikacija ugovora za transformaciju je zasnovana na standardnim konceptima UML-a i OCL-a. Cilj specificiranih ugovora je da definišu šta radi transformacija modela, pod kojim uslovima može biti primenjena i šta se može očekivati kao rezultat transformacije. U [Cari04] su ugovori predstavljeni kao važna komponenta procesa razvoja softvera, zasnovanom na modelima, jer njihova primena može da unapredi specificiranje i dokumentovanje transformacija, validaciju i testiranje transformacija.

Ugovori za transformaciju modela definisani su kao strukture nad sledećim skupovima:

- skup ograničenja koja moraju biti ispunjena kako bi model mogao biti kandidat za izvorni model transformacije,
- skup ograničenja koja moraju biti ispunjena kako bi se model mogao smatrati validnim ciljnim modelom transformacije i
- skup ograničenja na vezama između različitih elemenata izvornog i ciljnog modela.

OCL nije programski jezik, tj. putem njega nije moguće definisati akcije i tok izvršavanja na način koji je svojstven jezicima treće generacije. Primarna namena ovog jezika nije kreiranje izvršnih specifikacija, već je namenjen pre svega modelovanju, ali danas se pojavljuju alati koji su u stanju da validiraju model na osnovu specifikacija zadatih putem OCL-a.

Istraživanja u okviru ove doktorske teze, između ostalog, imaju za cilj definisanje meta-modela šema baza podataka na različitim nivoima apstrakcije. Meta-modeli opisani putem MOF-a, dodatno su precizirani putem pravila specificiranih u OCL jeziku.

2.1.3. Transformacije modela

Transformacije modela različitih nivoa apstrakcije predstavljaju jedan od glavnih koraka u primeni MDA pristupa. Postoji više definicija transformacije modela koje se pojavljuju u literturi vezanoj za MDE.

U MDE Guide, [MDA03] autori Miller i Mukerji daju svoju definiciju transformacije modela:

„*Transformacija modela je proces konvertovanja jednog modela u drugi model istog sistema*.“

U [Kleppe03] data je sledeća definicija transformacije modela:

„*Transformacija je automatsko generisanje ciljnog modela iz izvornog, na osnovu definicije transformacije. Definicija transformacije je skup pravila za transformaciju koja zajedno opisuju kako se model u jeziku izvora može transformisati u model na ciljnom jeziku. Pravilo transformacije je opis kako se jedna ili više konstrukcija na jeziku izvora može transformisati u jednu ili više konstrukcija na ciljnom jeziku*.“

U [Mens05], autori Mens, Czarnecki i Van Gorp predlažu generalizaciju ove definicije dodavanjem odrednice po kojoj je dozvoljeno postojanje više modela izvora i/ili više ciljnih modela. Primer za ovo je spajanje modela (*model merging*), gde se želi kombinacija ili spajanje nekoliko modela, koji se razvijaju paralelno, u jedan model kao rezultat. Drugi primer je transformacija koja prevodi PIM u nekoliko PSM modela.

Tratt koristi sledeću definiciju u [Tratt05]:

„*Transformacija modela je program koji mutira jedan model u drugi, nešto slično kompajleru.*“

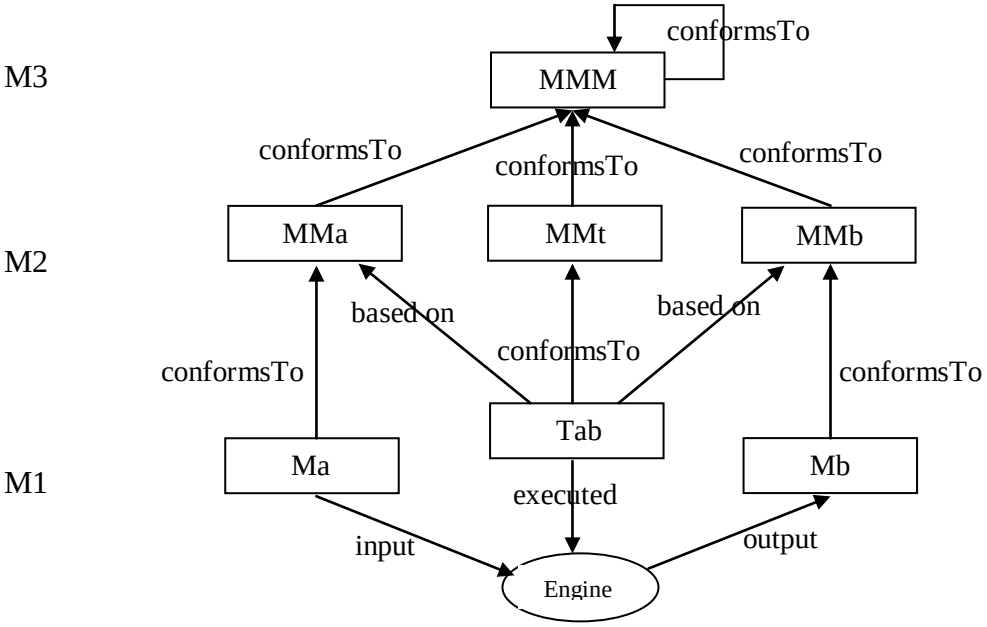
Autori Sendall i Kozaczynski u [Senda03] smatraju da je transformacija modela automatizovani proces, koji uzima jedan ili više izvornih modela kao ulaz i proizvodi jedan ili više ciljnih modela kao izlaz, pomoću skupa transformacionih pravila.

U [Kurtev05], navedena je sledeća definicija:

„*Transformacija modela je proces automatskog generisanja ciljnog modela iz izvornog modela, u skladu sa definicijom transformacije, koja je izražena u modelu jezika za transformacije*.“

Na slici 2.4 prikazan je opšti MDE obrazac transformacije modela i informacije koje su potrebne za takvu transformaciju. Izvorni model *Ma* se transformiše u ciljni model *Mb* u skladu sa definicijom transformacije *Tab*. Definicija transformacije je takođe model. Izvorni i ciljni model kao i definicija transformacije formalno odgovaraju svojim meta-modelima *MMa*, *MMb* i *MMt*, respektivno. Sva tri spomenuta meta-modela su u skladu sa meta-metamodelom *MMM*. U kontekstu OMG standarda meta-metamodel *MMM* je MOF.

U narednim tačkama biće data klasifikacija tipova transformacija modela i jezici koji podržavaju postupke transformacije modela.



Slika 2.4. Obrazac transformacije modela

2.1.3.1. Klasifikacija transformacija modela

Transformacije modela različitih nivoa apstrakcije predstavljaju jedan od glavnih koraka u primeni MDA pristupa. Uopšte, transformacije se mogu podeliti u dve široke kategorije. Prva kategorija obuhvata transformacije modela u druge modele (*model-to-model* - M2M), dok druga obuhvata transformacije modela u izvršne specifikacije, tj. programski kod (*model-to-code* - M2C) [Ruscio07].

Posmatrano u kontekstu MDA, pri specifikaciji M2M transformacija, u osnovi se definiše mapiranje između dva meta-modela: specificira se koji elementi u izvornom meta-modelu se mapiraju na koje elemente u ciljnom meta-modelu.

Mens i Van Gorp u radu [Mens05] klasifikovali su transformaciju u nekoliko dimenzija. U zavisnosti od toga „šta treba transformisati u šta“, tj. u zavisnosti od izvornih i ciljnih artefakata transformacije, razlikuju se sledeće vrste:

- transformacija programa i modela,
- endogena i egzogena transformacija i
- horizontalna i vertikalna transformacija.

Ako se transformišu programi, tj. izvorni kod, *bytecode* ili mašinski kod, koristi se termin transformacija programa (*program transformation*). Ukoliko se radi o modelima koristi se termin transformacija modela (*model transforamtion*). Po mišljenu ovih autora, druga vrsta transformacije obuhvata i prvu pošto se model može protezati od apstraktnog modela analize, preko konkretnog modela dizajna, do veoma konkretnog modela izvornog koda. Transformacija modela, takođe, uključuje transformaciju od više apstraktnog ka više konkretnom modelu (od dizajna do koda) i obrnuto (u kontekstu reverznog inženjeringa). Primeri alata u kojima se primenjuje ova vrsta transformacije modela su generatori koda i parseri.

Pri transformaciji modela, modele treba predstaviti u nekom jeziku za modelovanje. Sintaksa i semantika jezika za modelovanje izražava se pomoću meta-modela. U zavisnosti od jezika kojim su opisani izvorni i ciljni modeli koji učestvuju u transformaciji razlikuju se endogena i egzogena transformacija. Endogena transformacija je transformacija modela

opisanih istim jezikom. Egzogena transformacija je transformacija modela opisanih različitim jezicima.

Akehurst i Kent u radu [Akehu02] predlažu u osnovi istu podelu, samo što se u njihovoj taksonomiji umesto endogena transformacija koristi termin *rephrasing*, a termin *translation* za egzogenu. Tipični primeri *rephrasing* tj. endogene transformacije su:

- optimizacija čiji je cilj unapređenje nekih operativnih osobina (performansi) pri čemu se čuva semantika softvera,
- *refactoring*, menja internu strukturu softvera radi poboljšanja nekih karakteristika kvaliteta softvera (razumevanje, modularnost, adaptivnost, ponovno korišćenje) pri čemu treba da očuva ponašanje softvera [Fowler99],
- pojednostavljenje i normalizacija, koristi se za smanjenje sintaksne kompleksnosti.

Tipični primeri translacije, tj. egzogene transformacije su:

- sintetizovanje (*synthesis*) specifikacije na višem apstraktnom nivou (model analize ili dizajna) u niži, konkretan, na pr. model Java programa. Tipičan primer sintetizovanja je generisanje koda, gde se izvorni kod prevodi u bajt kod ili izvršni kod ili kada se model dizajna prevodi u izvorni kod,
- reverzni inženjering, koji je suprotan od sintetizovanja, ekstrahuje specifikaciju na višem nivou apstrakcije iz nižeg nivoa,
- migracija programa napisanih u jednom jeziku u drugi pri čemu se mora očuvati isti nivo apstrakcije.

Horizontalna transformacija je transformacija gde su izvorni i ciljni model na istom nivou apstrakcije. Tipičan primer je *refactoring*.

Vertikalna transformacija je transformacija gde su izvorni i ciljni model na različitim nivoima apstrakcije. Tipičan primer je *refinement*, gde se specifikacija postepeno unapređuje u punu implementaciju sukcesivnim koracima koji dodaju konkretnije detalje.

Lano i ostali u radu [Kolah11 i Lano12] za klasifikaciju transformacija koristili su kriterijume koji se baziraju na očuvanju semantike modela i semantike veza između izvornog i ciljnog modela. Takođe, kriterijumi su bili korišćeni jezici kao i nivoi apstrakcije. Oni razmatraju sledeće tri uopštene kategorije transformacije modela:

- *refinements*,
- poboljšanje kvaliteta (*quality improvements*) i
- *re-expressions*.

Transformacije koje se svrstavaju u kategoriju r*efinements* unapređuju model do implementacije. Primeri uključuju transformaciju PIM u PSM u MDA pristupu ili generisanje koda iz PSM. Semantika modela može da se menja, ali sve osobine originalnog modela treba da budu podržane u novom modelu pomoću neke interpretacije. Transformacije ovog tipa su obično egzogene, okarakterisane kao vertikalne transformacije (od višeg ka nižem nivou) sa ciljem da model na izlazu bude semantički ekvivalentan sa izvorom.

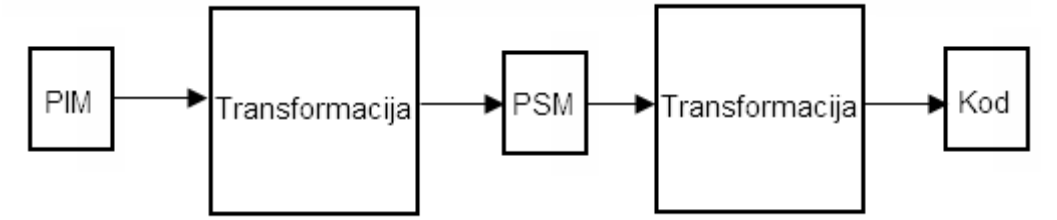
Transformacije koje se svrstavaju u kategoriju poboljšanja kvaliteta ne menjaju nivo apstrakcije modela, obično očuvaju njegovu semantiku, ali unapređuju njegovu strukturu i organizaciju. One obično funkcionišu nad istim jezikom tj. pripadaju endogenim i horizontalnim transformacijama koje čuvaju sematiku.

Transformacije koje se svrstavaju u kategoriju *re-expressions* prevode model u jednom jeziku u njegov „najbliže ekvivalnetan“ u drugom jeziku, kao što su različite verzije jezika izvora. Ovo je korisno za reinženjering, migracije, validaciju i integraciju alata. Ovo su egzogene i horizontalne transformacije koje čuvaju sematiku.

U MDA Guide [MDA03] i [Singh09], životni ciklus softverskog sistema predstavljen je transformacijama modela koje se ostvaruju kroz faze životnog ciklusa i koje, na kraju, rezultuju u konkretnoj implementaciji sistema. Transformacije jednog PIM modela u drugi PIM model, tzv. PIM-u-PIM transformacije modela, prisutne su tokom faza analize i

projektovanja. Transformacije PIM modela u PSM modele, tzv. PIM-u-PSM transformacije, izvršavaju se tokom transformacije specifikacija projekta u implementaciju sistema. Takođe, postoje i transformacije PSM modela u PIM modele, tzv. PSM-u-PIM transformacije koje se primjenjuju u procesu reverznog inženjeringa i koje su predmet istraživanja ove doktorske disertacije.

Specifikacija složenog sistema sastoji se od mnogo međusobno povezanih modela različitih nivoa apstrakcije. Model može imati ulogu modela tipa PSM na jednom nivou, dok na drugom nivou može predstavljati osnovu za dalju transformaciju. Uopšte, moguće je imati niz transformacija koje počinju od inicijalnog modela tipa PIM i završavaju se generisanjem programskog koda i obrnuto. Na slici 2.5 prikazana su tri glavna koraka MDA pristupa u *forward* inženjeringu.



Slika 2.5. Tri glavna koraka razvojnog procesa MDA pristupa

U radu [Mens05], Mens i Van Gorp definišu sledeće važne karakteristike transformacije modela.

- **Nivo automatizacije.** Može se napraviti razlika između transformacija modela koje mogu da se automatizuju i transformacija koje je potrebno raditi manuelno, ili je barem neophodna neka ručna intervencija. Primer ove druge je transformacija specifikacije zahteva u model analize. Za takvu transformaciju manuelna intervencija je potrebna da razreši dvosmislenost, nekompletnost i nedoslednost u zahtevima koji su, često samo parcijalno, izraženi u prirodnom jeziku.
- **Kompleksnost transformacije.** Neke transformacije, kao što je *refactoring*, mogu se smatrati jednostavnim, dok se druge mogu smatrati veoma kompleksnim. Primer kompleksnih su parseri, kompajleri i generatori koda. Razlike u kompleksnosti mogu biti veoma velike tako da ovo obično zahteva potpuno različit skup tehnika i alata.
- **Očuvanje (*preservation*).** Iako postoji širok spektar različitih tipova transformacija koje se koriste u MD razvoju, svaka transformacija čuva izvesne aspekte izvornog modela u transformisani ciljni model. Osobine koje se čuvaju mogu se razlikovati značajno u zavisnosti od tipa transformacije. Na primer u *refactorings* ili *recstructurings* treba biti očuvano ponašanje, dok se struktura menja. Kod *refinements* potrebno je da se očuva korektnost.

U radu [Czarn03], Czarnecki i Helsen, takođe, definišu važne osobine transformacije modela od kojih će biti nevedene neke koje su bitne za praćenje daljeg teksta.

- **Praćenje (*traceability*).** Transformacije mogu zapisivati veze između svojih izvornih i ciljnih elemenata. Ove veze mogu biti korisne u vršenju analize uticaja, odnosno kako promena jednog modela može uticati na druge povezane modele, pri sinhronizaciji modela i traženju grešaka (*debugging*).
- **Smer transformacije (*directionality*).** Transformacija može biti jednosmerna ili dvosmerna. Jednosmerna transformacija se može izvršavati samo u jednom smeru. U tom slučaju se ciljni model generiše ili menja na osnovu izvornog modela. Dvosmerna transformacija se može izvršavati u oba smera, što je korisno u kontekstu sinhronizacije modela. Dvosmerna transformacija može se postići korišćenjem

dvosmernih pravila ili definisanjem dva odvojena komplementarna jednosmerna pravila, po jedno za svaki smer.

Istraživanja u oblasti transformacija modela važna su za razvoj alata IIS*Ree, razvojnog okruženja IIS*Studio, koji predstavlja jedan od postignutih rezultata ove doktorske teze. U okviru alata omogućeno je da se prvo izvrši niz transformacija koje počinju od specifikacije modela koji reprezentuju šeme baze podataka, tipa PSM i završavaju se specifikacijom IIS*Case PIM-a, a zatim se ponovo nizom transformacija dobijeni modeli prevode u implementacioni opis šema baza podataka. Na osnovu prethodno navedenih klasifikacija transformacija modela, transformacije primenjene u IIS*Ree pripadaju M2M, egzogenim, vertikalnim transformacijama, PSM-u-PIM.

Transformacije modela su bile, takođe, predmet istraživanja u ranijem razvoju alata IIS*Case, u *forward* inženjeringu. Specifikacije modela kreiranih alatom IIS*Case, na PIM nivou, transformišu se u specifikacije modela namenskog za platformu, pri čemu se kao krajnji rezultat može dobiti programski kod ili SQL kod specifičan za određenog proizvođača RSUBP-a. Na osnovu prethodno navedenih klasifikacija transformacija modela, transformacije primenjene u IIS*Case-u pripadaju M2M i M2C, egzogenim, vertikalnim transformacijama, PIM-u-PSM. Postupak transformacije u *forward* inženjeringu biće ukratko prezentovan u poglavlju 4. Detalji se mogu naći u mnogim radovima autora okruženja IIS*Studio, a između ostalih u [Lukov08, Lukov10, Lukov13, Aleks06, Aleks07a, Pavić05, Banov10, Lukov07, Ristic07, Lukov03, Lukov10a].

Transformacije modela su, takođe, bitne za budući razvoj alata razvojnog okruženja IIS*Studio. Njihovom primenom, bilo bi omogućeno da se specifikacije modela kreiranih alatom IIS*Case transformišu u specifikacije koje se mogu uvoziti u druge MDA alate. I obrnuto, specifikacije modela kreiranih upotrebom nekih od alata, koji su danas u širokoj upotrebi tokom procesa projektovanja informacionih sistema, bilo bi moguće preuzimati i dalje razvijati upotrebom alata okruženja IIS*Studio.

2.1.3.2. Jezici za transformacije modela

U ovoj tački opisani su karakteristični pristupi i klasifikacije jezika za transformacije modela na osnovu više različitih kriterijuma, kao i neki konkretni jezici za transformacije modela.

Czarnecki i Helsen u radu [Czarne06] daju taksonomiju transformacije modela zasnovanu na *feature* modelu. Takođe, oni diskutuju o pristupima koji se koriste za klasifikaciju jezika za transformacije tipa M2M. Neki od tih pristupa opisani su u narednom tekstu.

Pristup direktne manipulacije (*Direct manipulation approach*) nudi internu reprezentaciju modela i mogućnost da se manipuliše tom reprezentacijom pomoću skupa proceduralnih API-ja. Obično je implementiran kroz objektno-orijentisano okruženje. Korisnici moraju da implementiraju pravila transformacije, uglavnom od početka, pomoću nekog programskog jezika.

Operativni pristup (*Operational approach*) je sličan prethodnom, ali nudi više podrške za transformaciju modela. Tipično rešenje u ovoj kategoriji je proširenje korišćenih formalizama za meta-modelovanje karakteristikama kojima se mogu predstaviti izračunavanja. Jedan primer bi bio kada bi se upitni jezik kao što je OCL proširio imperativnim konstruktima. Primer konkretnih jezika u ovoj kategoriji su QVT *Operational mappings* [QVT09] i Kermeta [Kermeta].

U relacionom pristupu (*Relational approach*) glavni koncept je matematička relacija. U ovom pristupu se transformacija modela opisuje tako što se specificiraju relacije između izvornih i ciljnih tipova elemenata modela. Skup relacija precizira *šta* transformacija menja u modelu, a ne *kako* se izvršava ta promena. Relacioni pristup je u tom smislu, sličan

deklarativnim jezicima kao što je programski jezik Prolog [Kuster04]. U skladu sa svojom matematičkom prirodom, relacije ne impliciraju smer transformacije. Iz tog razloga relacioni pristup prirodno podržava dvosmerna pravila. Takođe, ponekad obezbeđuje i tehniku praćenja transformacije (*backtracking*). Većina relacionih pristupa zahtevaju striktnu podelu između izvornih i ciljnih modela. Primer relacionog pristupa je QVT *Relations* [QVT09].

Hibridni pristup (*Hybrid approach*) kombinuje različite tehnike prethodno prezentovanih pristupa. Primer primene ovog pristupa je jezik ATL [ATL10].

Pristup transformacije zasnovane na grafovima (*Graph-transformation based approach*) koristi prednosti teoretske osnove i sličnosti između grafova i modela. Pri opisu transformacije modela pomoću transformacije grafa, izvorni i ciljni model moraju biti predstavljeni pomoću grafa. Obavljanje transformacije modela pomoću transformacije grafa znači apstraktnu sintaksu grafa modela transformisati prema pravilima transformacije u sintaksu grafa ciljnog modela. Najpoznatiji predstavnici ovog pristupa su AGG [Taent03], VIATRA2 [Csert02] i AToM3 [Lara02]. Iako su ovi jezici prilično privlačni zbog svoje matematičke osnove, po mišljenju autora nisu pogodni za složene transformacije.

U radu [Senda03], autori Sendall i Kozaczynski posebnu pažnju posvećuju karakteristikama koje bi jezik za transformacije trebalo da ima. Pored toga oni daju klasifikaciju pristupa transformaciji modela primenjenih u alatima za transformaciju. Za razliku od prethodno navedenih, oni definišu samo tri pristupa:

- pristup direktne manipulacije modelom (*Direct Model Manipulation*),
- pristup reprezentacije modela za razmenu (*Intermediate Representation*) i
- pristup podrške jezicima za transformaciju.

Prvi pristup je identičan onome koji je prezentovan u [Czarne06]. Alati koji ga primenjuju nude korisniku pristup internoj reprezentaciji modela i mogućnost manipulacije tom reprezentacijom pomoću skupa proceduralnih API-ja.

Alati koji korsite drugi pristup izvoze model u standardizovanom formatu, tipično XML. Spoljni alat preuzima izveženi model i transformiše ga. U [Vara09a] autor ovaj pristup nazva *XML-based* pristupom.

Alati, koji primenjuju pristup podrške jezicima za transformaciju, nude jezike koji obezbeđuju skup konstrukata ili mehanizama za eksplicitno izražavanje, komponovanje i primenu transformacija. Ovi jezici spadaju u namenske jezike za opis transformacija. U ove jezike autori ubrajaju i jezike zasnovane na grafovima, koji su već opisani u prethodmom tekstu, i jezike sa deklarativnim pristupom.

Jezik za transformacije je deklarativan ako definicije transformacija napisane u tom jeziku određuju veze između elemenata u izvornim i ciljnim modelima, bez detalja o redu izvršavanja. Za razliku od deklarativnih, imperativni jezici za transformacije određuju sekvencu koraka koje treba izvršiti da bi se proizveo rezultat. Takođe, tu postoje i hibridni transformacioni jezici koji sadrže skup pravila koji određuju korake koje treba izvršiti da bi traženi rezultat bio dobijen.

Autori rada [Kolah11, prošireno u Lano12] definišu koje osobine treba da podrži jezik za specifikaciju transformacije modela:

- modularnost,
- validacija,
- verifikacija i
- implementacija transformacije.

Jezik za specifikaciju transformacije modela podržava modularnost ukoliko je moguće organizovati pravila transformacije modela u module.

Validacija je podržana od strane jezika za specifikaciju transformacije modela ukoliko je moguće analizirati specifikaciju radi utvrđivanja da li ona korektno reprezentuje nameru transformacije. Jedan od načina validacije je testiranje.

Verifikacija je podržana od strane jezika za specifikaciju transformacije modela ukoliko je moguće dokazati da je transformacija konzistentna, zadovoljiva i semantički korektna, tj. kada su sva ograničenja izvornog modela zadovoljena u ciljnom modelu, eventualno pod određenom interpretacijom.

Ukoliko se specifikacija transformacije može koristiti za automatsko generisanje izvršne implementacije transformacije, koja je korektna u odnosu na specifikaciju, tada se može reći da jezik za specifikaciju transformacije modela podržava implementaciju transformacije.
Isti autori koriste tri uopštena stila specifikacije za definisanje transformacije modela:

- deklarativni, kod kojeg su transformacije opisane apstraktno, tj. matematičkim relacijama između izvornih i ciljnih modela [Akehu02, Akehu05 i Boron05],
- imperativni, kada su transformacije definisane kao programi koji eksplicitno definišu detalje kako se izvorni model transformiše u ciljni model [Kermeta] i
- hibridni, kao kombinacija deklarativnog i imperativnog [Kolov08].

Deklarativni stil ima prednost u tome što transformacije mogu biti opisane jasnije i konciznije, izostavljanjem detalja strategije za selektovanje i modifikaciju elemenata modela, i izbegavanjem eksplicitnog redosleda primene pravila na specifične elemente. Imperativni stil, s druge strane, olakšava implementaciju transformacije u izvršnom obliku. Hibridni stil pokušava da spoji ova dva stila u cilju objedinjavanja prednosti od oba.

Ključni problem specifikacije transformacije modela je taj što, semantički, transformacije modela predstavljaju relacije između jednog ili više čitavih modela sa jednim ili više čitavih modela. Takav uopšteni opis je nepraktičan za netrivijalne jezike i transformacije. Umesto toga, transformacije modela se specificiraju kao relacije između individualnih elemenata u izvornom modelu (modelima), i individualnih elemenata u ciljnom modelu (modelima). Model-u-model relacija se sastoji od kompozicije tih individualnih relacija. Ovo dovodi do potencijalne dvosmislenosti i nekompletnosti usled mogućih alternativnih kompozicija.

U radu [Mens05], predloženi su kriterijumi za efikasnost jezika i alata za transformaciju, uključujući mogućnost kompozicije transformacija i demonstracije sintaksne i semantičke korektnosti.

Pred jezike (*transformation language*) i alate (*transformation tools*) za transformaciju modela postavlja se više važnih funkcionalnih zahteva koje oni treba da zadovolje [Mens05]:

- mogućnost za kreiranje/čitanje/izmenu/brisanje transformacije,
- mogućnost da predloži kada, tj. u kom slučaju treba primeniti koju transformaciju,
- mogućnost prilagođavanja ili ponovnog korišćenja transformacija,
- mogućnost garantovanja korektnosti transformacija,
- mogućnost rada sa nekompletnim i nekonzistentnim modelima,
- mogućnost grupisanja, kompozicije i dekompozicije transformacija,
- mogućnost testiranja, validacije i verifikacije transformacije,
- mogućnost za specifikaciju opštih i transformacija višeg reda,
- mogućnost specifikacije dvosmernih transformacija i
- mogućnost praćenja (*traceability*) i propagacije izmene.

Neki jezici dozvoljavaju specifikaciju definicije transformacije tako da ona može biti primenjena samo u jednom smeru: od izvornog do ciljnog modela. Ove transformacije su poznate pod nazivom jednosmerne transformacije. Drugi jezici dozvoljavaju definicije koje mogu biti izvršene u oba smera. Ove transformacije se nazivaju dvosmerne transformacije. Ova mogućnost je korisna kada dva modela treba da budu sinhronizovana. Ako jezik za transformacije ne podržava definicije za dvosmerne transformacije, onda se mogu koristiti dve odvojene definicije, po jedna za svaki smer. Transformacija obično uzima jedan model kao ulazni i proizvodi jedan model kao izlazni (1-u-1 transformacije). Postoje i slučajevi kada se više modela stvara od jednog ulaznog modela (1-u-N transformacije), kao npr. jedan model u Java kod (prvi izlazni model) i u XML šemu (drugi izlazni model). Kada se više modela

integriše u jedan, to se naziva kompozicija modela i primer je N-u-1 transformacije. Takođe, postoje i M-u-N transformacije.

U skladu sa svim što je prethodno navedeno, za postupak transformacije šema baza podataka u postupku reverznog inženjeringa, koji je jedan od predmeta istraživanja ove doktorske teze, korišćen je hibridni pristup. Razlog, zašto nije korišćen čist deklarativni pristup, su složene transformacije u kojima je vrlo često bilo potrebno eksplicitno definisati detalje kako se neki element izvornog modela transformiše u jedan ili više elemenata ciljnog modela. Algoritmi transformacija i njihova implementacija detaljno su opisani u poglavlju sedam. Za implementaciju transformacija korišćen je jezik ATL koji predstavlja implementaciju jezika QVT propisanog od strane OMG-a. Specifikacija ova dva jezika ukratko je opisana u tekstu koji sledi.

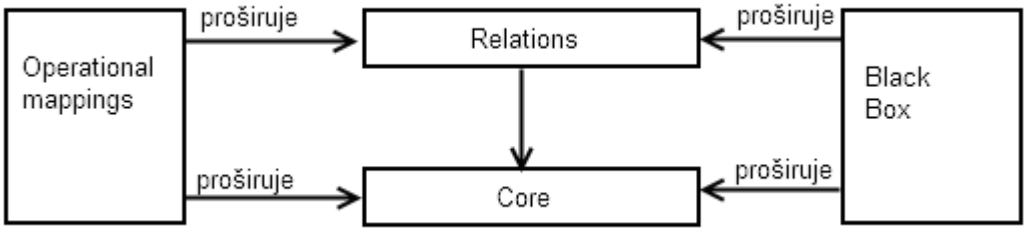
Za transformacije u *forward* inženjeringu korišćen je pristup direktne manipulacije, tako da su implementirane pomoću GPL Java u okviru objektno-orijentisanog okruženja Oracle JDeveloper. Postoje dva glavna razloga zašto je korišćen ovaj pristup. U vreme razvoja alata IIS*Case, putem kojeg se vrši *forward* inženjering, ostali navedeni pristupi nisu bili na zadovoljavajućem stepenu razvijenosti. Drugi, ali ne i manje važan razlog, predstavlja činjenica da su transformacije modela u izvršne specifikacije u okviru IIS*Case alata izrazito složene. Najveći deo opisa tih transformacija dat je u algoritamskoj formi, kao i formi predikatskih formula i teorema koje su dokazane u [Mogin04]. Po mišljenju autora, na sadašnjem stepenu razvoja ovih jezika, realizacija takvog zadatka bila bi izazov koji se planira u daljim istraživanjima.

2.1.3.3. QVT

OMG grupa ustanovila je standardizovani skup jezika *Query/View/Transformation* (QVT), namenjenih za definisanje transformacija modela. QVT obuhvata tri međusobno povezana jezika:

- *Relations*,
- *Core* i
- *Operational Mappings*.

Za sve jezike je karakteristično da se transformacije izvršavaju nad modelima koji se zasnivaju na MOF meta-modelima. Arhitektura QVT-a, definisana od strane OMG-a, prikazana je na slici 2.6. U ovoj tački biće ukratko objašnjeni svi jezici i komponente prikazane na slici 2.6, dok je detaljna specifikacija QVT-a data u [QVT09].



Slika 2.6 Arhitektura QVT-a

QVT-*Relations* je deklarativni jezik čija notacija je na relativno visokom nivou apstrakcije, bliska formi zahteva, tako da specifikacija u notaciji može biti direktno validirana. Svako QVT- *Relations* pravilo ima matematičku interpretaciju kao predikat, pa dozvoljava, u

principu, formalnu verifikaciju. Jezik omogućava kreiranje jednosmernih i dvosmernih transformacija modela.

Modularnost QVT-*Relations* notacije je zastupljena samo na nivou pojedinačnih pravila, koja mogu da budu međusobno zavisna i čak uzajamno rekurzivna. Pošto je notacija veoma bliska UML-u i OCL-u, i koristi MOF meta-modele, interoperabilnost sa drugim razvojnim alatima zasnovanim na UML-u je olakšana.

Ovaj jezik podržava validaciju transformacije putem poređenja uzoraka (*ptterna*) kompleksnih objekata. Uzorci su fragmenti modela sa nula ili više promenjljivih. Takođe se implicitno kreiraju *trace* modeli koji sadrže *trace* klase (klase za praćenje transformacije) i njihove instance kako bi se evidentirali događaji nastali prilikom izvršenja transformacije. Transformacije se izvršavaju pomoću interpretera. QVT-*Relations* sadrži i tekstualnu i konkretnu grafičku sintaksu.

QVT-*Core* je uveden sa namerom da postavi osnove za jezik QVT-*Relations*, ali implementacija ovog jezika nije zaživela u praksi. To je deklarativni jezik čija specifikacija je jednako moćna kao i kod jezika QVT-*Relations*, ali je nešto jednostavnije sintakse. S druge strane, specifikacije transformacije pomoću QVT-*Relations* su složenije, ali su deskriptivnije. Kod QVT-*Core* transformacija, *trace* modeli se specificiraju eksplicitno, kao MOF modeli, a kreiranje i brisanje *trace* objekata je definisano na uobičajen način, kao kod standardnih klasa.

Za pokretanje proceduralne implementacije transformacija iz QVT-*Relations* ili QVT-*Core* jezika postoje dva mehanizma: standardni jezik *Operational Mappings* i nestandardni *Black-box*.

Operational Mappings predstavlja proceduralni jezik koji proširuje QVT-*Relations* i QVT-*Core*. Njegova sintaksa obuhvata konstrukcione elemente koji se tipično koriste u proceduralnim programskim jezicima kao što su npr. petlje, uslovi i sl. Jezik *Operational Mappings* omogućava definisanje transformacija primenom potpuno proceduralnog pristupa koji uključuje specificiranje operativnih transformacija (*operational transformations*). Takođe u tzv. hibridnom pristupu, *Operational Mappings* omogućava nadogradnju transformacija proceduralnim operacijama koje implementiraju relacije. Operacije transformacije definišu na koji način se elementi jednog ili više izvornih modela transformišu u elemente jednog ili više ciljnih modela. Jezik omogućava kreirnje samo jednosmernih transformacija. *Operational Mappings* jezik koristi isti model za praćenje transformacije kao i QVT- *Relations*.

Black-box predstavlja deo arhitekture QVT-a koji omogućava transformacije zasnovane na drugim jezicima, kao što su npr. XSLT i XQuery. Koncept Crne kutije omogućava pozivanje izvršavanja eksternog koda prilikom vršenja transformacija. Ovo omogućava ponovno korišćenje postojećih biblioteka. Ovaj mehanizam, takođe, može predstavljati i opasnost, jer izvršavanje nije više u potpunosti kontrolisano od okruženja koje sprovodi transformaciju.

2.1.3.4. ATL

Jedna od najpoznatijih implementacija jezika QVT jeste *Atlas Transformation Language* (ATL). ATL specifikacija data je u [ATL10].

ATL je proizvod Eclipse fondacije, neprofitabilnog društva čiji su projekti usmereni ka razvoju sistema i alata otvorenog koda (*open-source*). U oblasti inženjerstva zasnovanog na modelima, ATL, kao *open-source* proizvod, ima veliki broj korisnika.

Transformacione specifikacije zadate putem ATL-a organizuju se u vidu modula. Osim osnovne funkcije transformacije modela, ATL pruža mogućnost specifikacije različitih upita nad modelima. Svaki modul sastoji se od sledećih elemenata:

- sekcije zaglavlja (*Header*),
- sekcije za uvoz (*Import*),
- skupa pomoćnih metoda (*Helpers*) i

- skupa pravila (*Rules*).

U sekciji zaglavlja navodi se ime modula i nazivi odgovarajućih promenljivih koje definišu izvorni i ciljni model kao i odgovarajuće meta-modele. Sekcija *Import* je opciona sekcija, gde je moguće naznačiti dodatne ATL biblioteke koje će biti uvežene. Pomoćne metode se smatraju ATL ekvivalentom Java metodama. Namenjene su za implementaciju dodatne logike korišćenjem imperativnih koncepata. Skupom pravila definiše se način kako se ciljni modeli generišu od izvornih. Pomoćne metode i pravila ne pripadaju određenoj sekciji u modulu, već se mogu definisati u proizvoljnom redosledu u skladu sa određenim uslovima.

Platforma Eclipse podržava jezike Ecore i *Kernel Meta Meta Model* (KM3) za specifikaciju meta-modela. Jezik KM3 je jezik razvijen od strane instituta *Institut national de recherche en informatique et en automatique* (INRIA) i u pitanju je opšti jezik za definisanje meta-modela i jezika namenskih za domen [Kernel]. Kako bi transformacija bila izvršena potrebno je definisati sledeće specifikacije:

- meta-model izvornog modela, kreiran putem Ecore ili KM3 specifikacije,
- meta-model ciljnog modela, kreiran putem Ecore ili KM3 specifikacije,
- model, zadat putem XMI specifikacije i
- ATL transformacija.

Kao rezultat transformacije dobija se ciljni model u XMI formatu.

Izvršavanje svakog ATL transformacionog programa prolazi kroz dva koraka, prevođenje i izvršavanje.

- U koraku prevođenja, ATL prevodilac prevodi transformacioni program u tzv. bajt-kod, koji predstavlja skup instrukcija spremnih za izvršavanje. Bajt-kod specifikacija je u XML formatu i slična je asemblerskom programu, ali ne sadrži detalje vezane za implementacionu platformu.
- U koraku izvršavanja, ATL virtuelna mašina podržava izvršavanje bajt-kod specifikacija, dobijenih u prvom koraku, kako bi transformacija modela bila realizovana. Virtuelna mašina podržava skup instrukcija za upravljanje modelima i može biti pokrenuta nad različitim sistemima za upravljanje modelima, kao što su npr. Eclipse EMF i NetBeans MDR.

ATL je hibridni jezik za transformaciju. On se sastoji od mešavine deklarativnih i imperativnih konstrukcija. Jouault i Kurtev u radu [Jouau06] predlažu deklarativni stil specifikacije transformacije. Ponekad je teško obezbediti kreiranje kompletno deklarativne specifikacije za dati problem transformacije. U tom slučaju, korisnici mogu pribeći impreativnim karakteristikama jezika.

ATL transformacije su jednosmerne, izvršavaju se nad izvornim modelima koji mogu samo da se čitaju, a ne i da se menjaju (*read-only*) i proizvode ciljni model koji može samo da se čita (*write-only*). U toku izvršavanja transformacije, na izvorni model može se primeniti navigacija, dok to ne važi za ciljni model. Dvosmerna transformacija implementira se kao par transformacija, po jedna za svaki smer.

Helper i pravila transformacije su konstrukcije koje se koriste za specifikaciju funkcionalnosti transformacije.

U ATL-u postoje različite vrste pravila, u zavisnosti od načina na koji se pozivaju i kako specificiraju rezultat. Pravilo se može pozivati eksplicitno putem naziva sa parametrima (*called rule*), ili se mogu izvršavati kao rezultat prepoznavanja ulaznog uzorka u izvornom modelu (*matched rule*). Rezultat izvršavanja pravila može biti deklarisan korišćenjem izlaznog uzorka, ili implementacijom u imperativnom bloku, ili pak korišćenjem oba načina.

Pravila sa ulaznim i izlaznim uzorcima nazivaju se deklarativnim pravilima. Ako pravilo ne sadrži imperativni blok onda je ono potpuno deklarativno. Pravila koja imaju naziv, formalne parametre, imperativni blok i nemaju ulazne i izlazne uzorke, naziva se procedurom. Ostale kombinacije nazivaju se hibridnim pravilima.

Imperativna sekcija, koja je opciona, počinje ključnom rečju *do* i omogućava specifikaciju imperativnog programskog koda koji će se izvršavati nakon inicijalizacije ciljnih elemenata generisanih pravilom kome imperativna sekcija pripada.

Primeri primene ATL jezika mogu se naći u literaturi [Beziv03a], [Jouau08] i [Jouau06].

Kao rezultat istraživanja ove doktorske disertacije nastao je niz transformacija modela koje su implementirane pomoću ATL jezika. U transformacijama su korišćena deklarativna i hibridna pravila. U najvećem broju slučajeva korišćena su hibridna pravila. U transformacijama je korišćeni i velik broj *helper*-i koji se pozivaju iz mnogih pravila. Specifikacije svih pravila i *helper*-a, nastalih u okviru ove doktorkse disertacije, biće detaljno prikazani u sedmom poglavlju.

2.1.3.5. Eclipse Modeling Project (EMP)

Projekat *Eclipse Modeling Project* (EMP) namenjen je obezbeđivanju podrške razvoju softverskih sistema, koji je zasnovan na modelima [Gronb09]. Radi se o skupu pristupa, alata i okruženja koji se grupišu oko platforme Eclipse. Osnovu predstavlja okruženje *Eclipse Modeling Framework* (EMF), koje je primarno namenjeno definisanju apstraktne sintakse jezika za modelovanje, tj. meta-modela. Osim definisanja apstraktne sintakse jezika za modelovanje, projekat EMP uključuje i okruženja koja podržavaju sledeće aktivnosti:

- kreiranje konkretne sintakse jezika za modelovanje,
- kreiranje transformacija tipa *model-to-model* (M2M) i
- kreiranje transformacija tipa *model-to-text* (M2T).

Platforma *Eclipse Modeling Framework* (EMF) trenutno je vodeća kada je u pitanju implementacija specifikacija definisanih u okviru MDA. Međutim, EMF je postao toliko značajan u ovoj oblasti da *defacto* počinje i da diktira pravce razvoja u okviru MDA. Ovo je jedan od razloga zašto je ova platforma posebno obrađena u sekciji posvećenoj MDA pristupu.

Glavni gradivni element platforme EMF je meta-metamodel Ecore koji predstavlja implementaciju specifikacije EMOF. Njegova namena je zadavanje apstraktne sintakse jezika za modelovanje. Specifikacije meta-modela čuvaju se u XMI formatu, pa je pored podrazumevanog editora, omogućeno kreiranje meta-modela upotrebom UML-a, kao i XSD šema. Takođe, putem okruženja za razvoj editora moguće je kreirati sopstveni editor za tu namenu.

Za definisane apstraktne sintakse novog jezika za modelovanje mogu se definisati i konkretne sintakse. Pored toga, moraju se obezbediti odgovarajući softverski alati za zadavanje specifikacija putem te sintakse. U okviru platforme EMF postoje okruženja koja su namenjena u ovu svrhu. Dodatno, ova okruženja podržavaju automatsko generisanje grafičkih i tekstualnih editora. U pogledu definisanja konkretne sintakse, izdvajaju se dva pristupa. Prvi se zasnivaju na upotrebi tekstualne notacije, dok su drugi orijentisani ka upotrebi vizuelne notacije.

Okruženje *Graphical Modeling Framework* (GMF) namenjeno je automatizovanom generisanju editora za vizuelno zadavanje specifikacija. Kao ulazne parametre za postupak generisanja editora potrebno je zadati:

- meta-model kreiran putem EMF-a i
- dodatne specifikacije koje se tiču samog izgleda i ponašanja editora.

Rezultat generisanja je vizuelno orijentisan alat, tj. editor za kreiranje i modifikaciju modela. Moguća je dalja nadogradnja vizuelnog editora u skladu sa individualnim potrebama.

U okviru projekta EMP, razvija se i projekat *Model to Model Transformation* koji treba da pruži okvir za razvoj transformacija tipa M2M. Moguće je koristiti implementaciju jezika QVT, kao i jezik ATL.

U okviru istraživanja ove doktorske disertacije korišćeno je EMF okruženje za implementaciju modula *M2M Transformer* alata IIS*Ree, u kojem se izvršava niz M2M transformacija, namenjenih za reverzni inženjering šema baza podataka, počev od ekstrahovanog logičkog modela baze podataka do modela opisanog PIM konceptima alata IIS*Case. Meta-modeli na kojima su zasnovane transformacije specificirani su Ecore jezikom, dok je za implementaciju transformacija korišćen ATL jezik. Primeri, korišćeni za testiranje transformacija kreirani su putem vizuelnih editora nastalih na osnovu meta-modela u GMF okruženju.

2.2. Reinženjering

Postoji više različitih pristupa u reinženjeringu informacionih sistema. Neka rešenja zastupaju pristup kompletne promene sistema, neka upućuju na delimične i evolutivne promene, dok neka rešenja polaze od potrebe prestrukturiranja potojećih opisa podataka i migracije samih podataka.

U nekim slučajevima izvorna i ciljna platforma sistema nad kojim se sprovodi reinženjering je ista. U suprotnom, reinženjering se koristi i za proizvodnju nove verzije sistema, prilagođene novom okruženju i paradigmama kao što su *web*, objektna orijentacija, itd.

U ovom odeljku dat je pregled nekih od najvažnijih pristupa, koncepata, tehnika i alata koji se koriste u reinženjeringu.

2.2.1. Osnovni koncepti reinženjeringa

U literaturi se najčešće citira rad [Chikof90], u kojem su autori Chikofsky i Cross definisali osnovne koncepte kao što su: reinženjering, restrukturiranje, reverzni inženjering i inženjering prema napred (*forward engineering*).

U navedenom radu, reinženjering predstavlja analizu i izmenu posmatranog sistema radi rekonstituisanja i implementacije u novi oblik, koji uključuje: reverzni inženjering, iza kojeg sledi neki oblik *forward* inženjeringa ili restruktuiranja.

Autori rada restrukturiranje definišu kao transformaciju jednog reprezentacionog nivoa u drugi na istom nivou apstrakcije, pri čemu treba da bude očuvano eksterno ponašanje sistema (funkcionalnost i semantika). Restrukturiranje ili *refactoring* ima za cilj poboljšanje kvaliteta sistema bez promene eksternog ponašanja. To smanjuje kompleksnost i povećava mogućnost održavanja.

Reverzni inženjering je proces analize posmatranog sistema radi:

- identifikacije komponenata sistema i njihovih veza i
- kreiranja reprezentacije sistema u drugoj formi ili na drugom nivou apstrakcije.

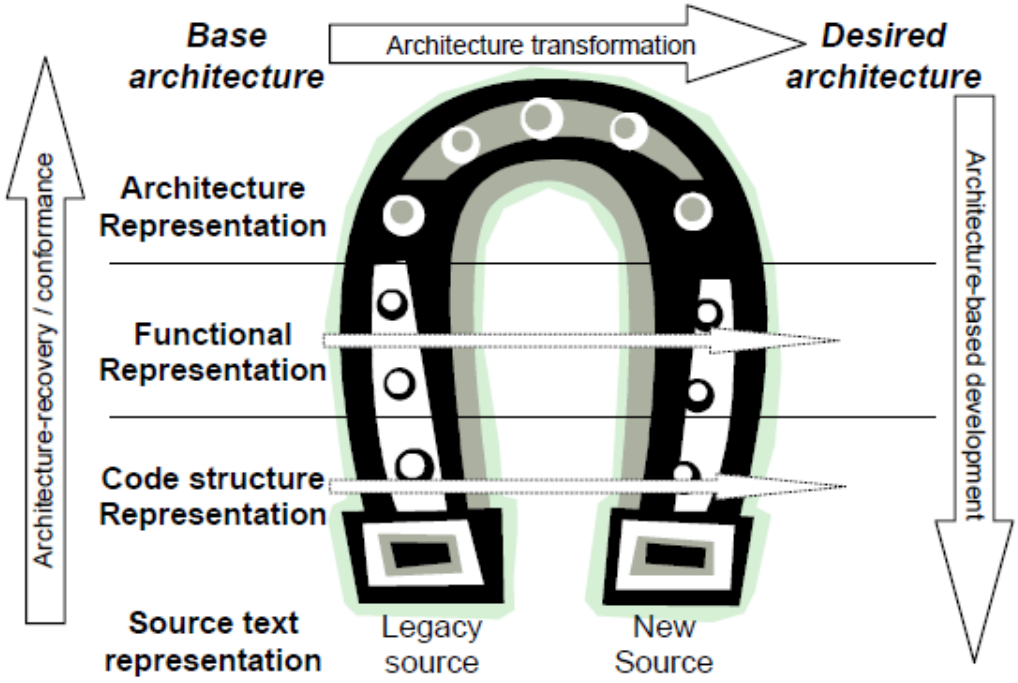
Forward inženjering autori rada definišu kao tradicionalni proces kretanja od visokog nivoa apstrakcije i logičkog implementaciono nezavisnog dizajna do fizičke implementacije sistema. Pridev *forward* je potreban iz razloga da bi se ovaj proces razlikovao od reverznog inženjeringa.

2.2.2. Životni ciklus reinženjeringa

Prema [Chikov90], reinženjering uključuje dve komplementarne aktivnosti: reverzni inženjering i *forward* inženjering. Na slici 2.7 prikazan je poznati model potkovice koji je

objavio SEI institut u [Berg99]. Po ovom modelu reinženjering se sastoji od tri osnovna koraka:

1. analiza postojećeg sistema i izgradnja skupa logičkih opisa njegove strukture,
2. transformacija ovih opisa za dobijanje nove, unapređene strukture i
3. restrukturiranje sistema na osnovu ovih novih opisa. [Duge06]



Slika 2.7. SEI model potkovice za reinženjering nasleđenih sistema – preuzeto iz [Berg99]

Softverska industrija primenjuje reinženjering u raznim domenima.

- Tradicionalno, reinženjering se sprovodi za tehnološke promene ili migraciju na drugi programski jezik. Jedan primer je transformacija nasleđenog sistema razvijenog pomoću strukturne paradigme u objektno-orijentisani sistem. Proces reinženjeringa, takođe, uključuje ponovno pisanje koda pisanog strukturnim programskim jezikom kao što je COBOL ili C u drugi objektno-orijentisani jezik kao što su Java ili C++.
- Reinženjering se, takođe, koristi za izmenu dizajna nasleđenog sistema u cilju poboljšanja održavanja, efikasnosti i drugih sistemskih funkcija. Na primer, nasleđeni sistem može biti zasnovan na Java tehnologiji. U fazi reverznog inženjeringa vrši se analiza postojećeg izvornog koda i oporavak dizajna nasleđenog sistema, tj. skup dijagrama klasa koji opisuju sistem. Nakon toga, u fazi restruktuiranja, može se modifikovati dizajn sistema. Konačno, u fazi *forward* inženjeringa generiše se novi izvorni kod iz unapređenih dijagrama klasa.
- Još jedna široka oblast gde se reinženjering primenjuje u softverskoj industriji je reinženjering baza podataka. U tom smislu u [Haina96] predlažu reinženjering kao mehanizam za dizajniranje relacionih baza podataka. Dok se u *forward* inženjeringu zahteva konceptalni dizajn, logički dizajn i fizički dizajn, u reverznom inženjeringu vrši se ekstrakcija strukture podataka i konceptualizacija strukture podataka.

2.2.3. Architecture-Driven Modernization (ADM)

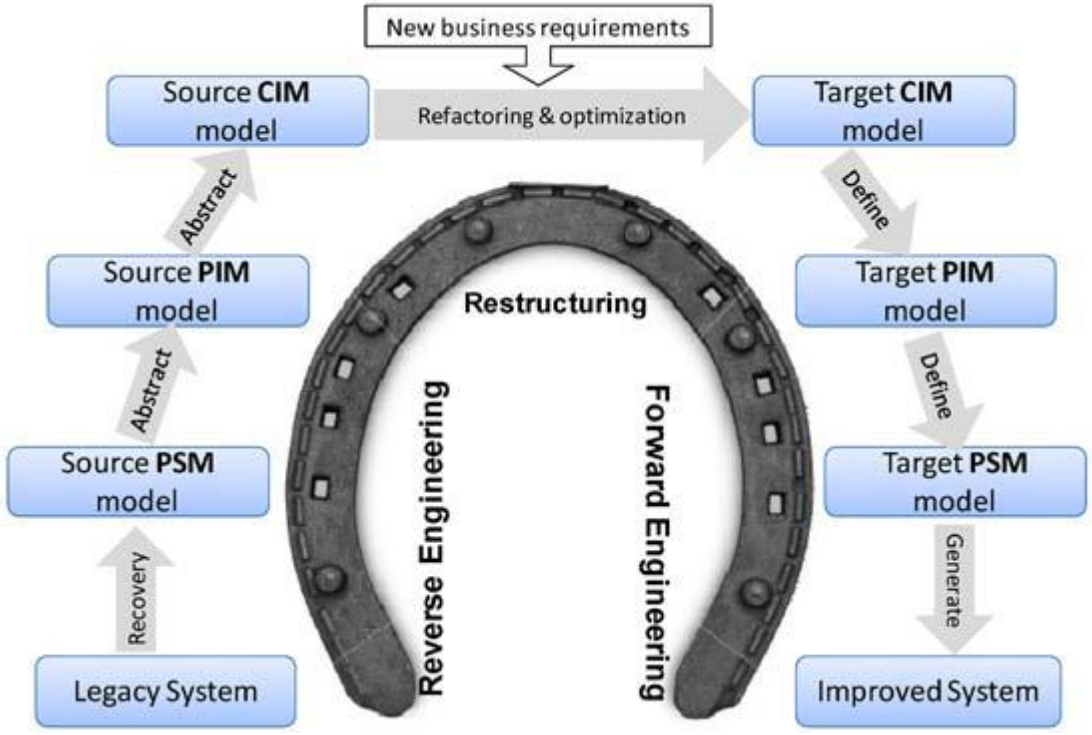
ADM je pradigma koju je ustanovila OMG grupa, koja se odnosi na modernizaciju softvera. ADM rešava problem tradicionalnog reinženjeringa uzimajući u obzir MD principe. ADM ne zamenjuje reinženjering već ga unapređuje.

Model reinženjeringa tipa potkovice prilagođen je ADM-u i poznat je kao model potkovice za modernizaciju softvera, prikazan na slici 2.8 [Ulrich10]. Model definiše tri faze modernizacije:

- faza reverznog inženjeringa koja identifikuje komponenete sistema i njihove veze i gradi jednu ili više apstraktnih reprezentacija postojećeg sistema;
- faza restruktuiranja koja restruktuira apstraktnu reprezentaciju u drugu reprezentaciju sistema na istom nivou apstrakcije, koji unapređuje izvorni nasleđeni sistem, ali čuvajući eksterno ponašanje sistema; i
- faza *forward* inženjeringa koja generiše fizičku implementaciju ciljnog sistema na niskom nivou apstrakcije.

Osim toga, ADM model potkovice koristi MDA nomenklaturu koja se odnosi na različite nivoe apstrakcije:

- CIM, poslovni pogled na sistem sa računarski nezavisne tačke gledišta na visokom nivou apstrakcije;
- PIM, pogled na sistem sa platformski nezavisne tačke gledišta na srednjem nivou apstrakcije, tako da PIM modeli apstrahuju sve implementacione detalje vezane za platformu; i
- PSM, tehnološki pogled na sistem sa platformski specifične tačke gledišta na nižem nivou apstrakcije.



Slika 2.8. ADM model potkovice – preuzeto iz [Perez10]

Do sada su MD principi obično korišćeni u *forward* inženjeringu. Međutim, *model-driven* razvojni principi mogu se primeniti i na reverzni inženjering i restrukturiranje.

U reverznom inženjeringu informacije oporavljene iz softverskih artefakata mogu se reprezentovati modelima koji su u skladu sa odgovarajućim meta-modelima. Pored toga, tehnike restruktuiranja i refaktoringa mogu se primeniti na te modele.

ADM obezbeđuje nekoliko prednosti kao što su unapređivanje *Return on Investment* (ROI) postojećeg sistema, redukovanje cene razvoja i održavanja, produžavanje životnog ciklusa postojećeg sistema i lakša integracija sa ostalim sistemima. [Perez10]

ADM je koncept modernizacije postojećeg sistema koji se može odnositi na sve aspekte arhitekture tekućeg sistema i mogućnost transformacije tekuće arhitekture u ciljnu arhitekturu. [Perez10]

U tom cilju, ADM podrazumeva takvo sprovođenje procesa reinženjeringa, u kojem se svi uključeni artefakti tretiraju kao modeli na određenom nivou apstrakcije.

ADM definiše nekoliko standarda vezanih za modernizaciju softvera. Temelj tih standarda je *Knowledge Discovery Meta-Model* (KDM). Poslednja verzija je 1.3, koja je objavljena 2012. godine i kao ISO standard ISO/IEC19506 [KDM12]. KDM obezbeđuje meta-modele za reprezentovanje i razmenu informacija o softverskim artefaktima postojećeg sistema. KDM omogućava reprezentaciju, na različitim nivoima apstrakcije, svih softverskih artefakata kao što su: izvorni kod, korisnički interfejs, baze podataka, poslovna pravila, itd.

U ovoj doktorskoj disertaciji definisan je originalni metodološki pristup koji se zasniva na analizi i transformacijama opisa baza podataka. U metodološkom pristupu primenjen je model potkovice kojim je obuhvaćen kompletan životni ciklus reinženjeringa opisa baze podataka. Primenjeni model potkovice obuhvata reverzni inženjering, restrukturiranje i *forward* inženjering zasnovan na upotrebi MDA principa i tehnika. Jedna interpretacija modela potkovice predložena u ovoj doktorskoj disertaciji biće prikazana u trećem poglavlju.

2.2.4. MD transformacije

Ima puno literaturnih navoda koji se bave transformacijama modela, koje mogu biti interesante u istraživanjima ove doktorske disertacije.

Autori rada [Guerr10] prezentuju familiju jezika *trans*ML, kao podršku razvoja transformacija modela, korišćenjem dobro utemeljenih inženjerskih principa. Oni su prezentovali meta-model platforme, meta-model jezika za specifikaciju i meta-model za mapiranje.

Za razliku od prethodnog rada, autori u radu [Kolah11] smatraju da je pogodnije koristiti standardizovane notacije kada god je to moguće, a naročito UML. Za UML se smatra da je sa njim većina, od onih koji razvijaju modele transformacija, familijarna.

U radovima [Atzeni06, Atzeni07, Atzeni07a i Atzeni08], autori predlažu okruženje pod nazivom MIDST, zasnovano na pretpostavci da svaki postojeći model podataka može da se reprezentuje sa konačnim skupom konstrukata. Jedna od karakteristia MIDST-a je da omogućava definiciju konstrukata opšte namene nazvanih metakonstrukti (*metaconstructs*). Još jedan važan koncept u MIDST okruženju je meta-metamodel nazvan Supermodel (*Supermodel*). Supermodel predstavlja najopštiji model koji uključuje sve moguće metakonstrukte. Autori smatraju da je svaki drugi model njegova specijalizacija. Supermodel je komponenta koja omogućava translaciju šema nezavisno od modela (*model-independent*) koja se može formulisati u tri faze: šema (instanca izvornog modela) kopira se u supermodel; translacija u ciljni modela odvija se u ovom okruženju i iz toga se generiše ciljni model. Transformacije su specificirane pomoću DATALOG pravila definisanih nad MIDST metakonstruktima. Upotreba *Supermodel*-a kao meta-metamodela u ovom rešenju, koji se razlikuje od MOF-a koji je standardizovan od strane OMG-a, otežava razvoj povezivanja ovog rešenja prema MOF kompatibilnim predlozima.

Sličan pristup imaju autori u radu [Neško07]. Oni se bave transformacijom XML šeme u relacioni model. Autori predlažu pristup u kojem se transformacija modela obavlja u dva koraka, preko specifičnog modela nazvanog Apstraktni model (AM). AM je definisan kao uopštenje strukturnih modela. Pod strukturnim modelom autori smatraju relacioni model, objektno-relacioni, objektni ili XML model. AM ima ulogu međumodela, tako da je za svaki model dovoljno definisati transformacije u i iz AM, umesto za svaki par modela. U prvom koraku transformacije modela vrši se transformacija izvornog modela u međumodel koji je izražen preko AM. Zatim, u drugom koraku obavlja se transformacija dobijenog međumodela u ciljni model. U prvom koraku transformacije primenjuje se princip apstrakcije, dok se u drugom primenjuje princip konkretizacije. Osnovni nedostatak predloženog pristupa transformacije modela je što se modeli mogu definisati samo preko koncepata AM, tj. transformacije se definišu samo za strukturne modele. Isto tako, nedostatak ovog pristupa je i što apstrakcija uvek dovodi do gubitka informacija, tj. preslikavanjem XML šeme u apstraktni model nije moguće sačuvati sve informacije koje su sadržane u konkretnoj šemi. U drugom koraku transformacije, AM se konkretizuje u relacionu šemu pri čemu se koriste samo koncepti AM. Zbog toga se u transformacijama ne mogu koristiti informacije specifične za XML šemu. Međutim, AM omogućava iskazivanje svih fundamentalnih koncepata strukturnih modela, tako da je za potrebe prevođenja u relacioni (ili neki drugi strukturni model) opisani pristup transformaciji XML šeme, po tvrđenju autora, sasvim zadovoljavajući.

U okruženju IIS*Ree predloženom u ovoj doktorskoj disertaciji, za razliku od prethodno opisana dva pristupa, ne koristi se poseban interni meta-metamodel koji je specifičan samo za IIS*Ree, već se koristi MOF kao standardizovani meta-metamodel. Takođe, za razliku od prethodna dva pristupa, korišćeno je direktno mapiranje između dve vrste modela. Na taj način nema transformacije u međumodel već se za oba modela, ulazni i izlazni, definišu odgovarajući meta modeli, a zatim se definišu pravila za transformaciju između njih.

U radu [Vara09], autori kroz studiju slučaja podržanu alatom M2DAT, predlažu MD razvoj objekto-relacione baze podataka. U tom cilju, autori su implementirali ATL transformaciju modela koja generiše objektno-relacioni model baze podataka od konceptualnog modela baze podataka reprezentovanog pomoću UML dijagrama klasa. Takođe, implementirali su i transformaciju modela u tekst, kojom se generiše SQL kod za modelovanu šemu baze podataka u MOFScript alatu. U okviru predloga definisali su DSL za modelovanje objektno-relacionih baza podataka kao i grafički editor za taj DSL. Prezentovali su Oracle 10g meta-model koji se može klasifikovati kao meta-model fizičke šeme baze podataka, namenske za određenog proizvođača.

Autori Lano i Kolahdouz-Rahimi u radovima [Lano11] i [Lano13] prezentuju studiju slučaja transformacije UML modela u model relacione baze podataka. Meta-model relacione baze podataka, korišćen u transformaciji, veoma je pojednostavljen u odnosu na meta-modele relacionih šema baze podataka nastalih kao rezultat ove doktorske disertacije.

2.2.5. Reinženjering baza podataka

U koracima reverznog inženjeringa uglavnom je glavna polazna tačka izvorni kod, mada taj proces može biti zasnovan na upotrebi mnogih drugih izvora kao što su tragovi izvršavanja programa [Chan03], datoteke sa podacima [Chikof90] ili relacione baze podataka [Ander94, Preme94, Haina96, Henra98, Jesus98, Alhaj01, Blaha01, Blaha01a, Alhaj03, Henra07].

Reverzni inženjering baza podataka je predmet velikog broja istraživanja kao bitan deo reinženjeringa, verovatno zato što baze podataka čuvaju sve relevantne informacije kompanija i jedan su od najosetljivijih elemenata u poslovanju koji mogu postati zastareli (*out-of-date*).

Reverzni inženjering baze podataka (*Database Reverse Engineering*-DBRE) bavi se zadacima razumevanja nasleđene baze podataka i ekstrakcije specifikacije njihovog dizajna

(semantike domena) [Chiang97 iz Lurd99]. U novijim i dobro projektovanim sistemima, DBRE zahteva analizu rečnika podataka baze podataka i ispitivanje specifičnog koda, kao što su trigeri korišćeni za implementaciju ograničenja koja nisu mogla biti implementirana deklarativnim mehanizmima i ne nalaze se u rečniku podataka. U starijim sistemima scenario može biti raznovrsniji. U svim slučajevima DBRE zahteva analizu rečnika podataka, *data mining* baze podataka, interakciju sa korisnicima, npr. programerima i analitičarima koji poznaju sistem, i u izvesnoj meri pokazuju razumevanje izvornog koda aplikacije.

Jedna od veoma često korišćenih paradigmi u izgradnji baza podataka informacionih sistema je relacionioni model podataka, koji će biti predmet metoda reinženjeringa i u ovoj doktorskoj disertaciji.

Do pre nekoliko godina većina istraživanja vezana za reinženjering baza podataka bavila se samo reverznim inženjeringom. Većina radova bavi se ekstrakcijom konceptualne šeme baze podataka, tako da je rezultat reverznog inženjeringa često ER ili EER dijagram.

Autori rada [Haina96] opisuju glavne korake reverznog inženjeringa baze podataka, kao suprotnog procesa od njenog nastajanja: ako *forward* inženjering baze podataka zahteva konceptualni dizajn, logički dizajn, fizički dizajn i prezentacioni dizajn, tada njen reverzni inženjering zahteva ekstrakciju i konceptualizaciju strukture baze podataka. Ekstrakcija strukture baze podataka obezbeđuje kompletan opis strukture podataka u skladu sa modelom podataka na kojem je zasnovan SUBP-a, dok je konceptualizacija strukture baze podatka pokušaj dobijanja semantike logičke šeme. Prema ovim autorima, ekstrakcija strukture baze podataka je inverzan proces od fizičkog dizajna *forward procesa*, dok je konceptualizacija u velikoj meri inverzan proces logičkog dizajna *forward* procesa.

Koristeći rad [Haina96] kao pogodan opis procesa reverznog inženjeringa, nekoliko autora je napravilo konkretne predloge za formiranje konceptualne šeme baze podataka.

Andersson u radu [Ander94] predlaže metod za dobijanje ERC+ specifikacije, iskazane pomoću proširenog ER modela, iz relacione baze podataka korišćenjem osnovnih podataka koji se odnose na nazive tabela i kolona, indeksa, kao i definicije pogleda, a koji se mogu naći u rečniku podataka postojeće (*legacy*) baze podataka. Ovi podaci su pronalaženi u DML izrazima koji se ekstrahuju iz programskog koda aplikacije, napisanom u 4GL okruženju.

Autori Alhajj i Polat u radu [Alhaj01] predstavljaju pristup transformaciji relacione baze podataka u objektno-orijentisanu, zasnovan na rečniku podataka i ekspertskom znanju. Autori se, takođe, bave migracijom podataka u novo okruženje.

Autori Premerlani i Blaha u radu [Preme94] prezentuju neke zaključke o svom radu na nekoliko studija slučaja. Autori koriste kao izvore informacija šemu baze podataka i podatke za dobijanje tzv. *Object Modelling Technique* (OMT) reprezentacije baze podataka. Oni sumiraju proces u pet koraka, započinjući sa razmatranjem klasa za svaku tabelu i završavajući sa određenim transformacijama za unapređenje performansi vremena i prostora.

Autori rada [Perez02 i Boron04] kreiraju OASIS objektno-orijentisanu konceptualnu šemu iz rečnika podataka relacione baze podataka. Rad se bavi migracijom podataka iz stare u novu realacionu bazu podataka, iako koriste objektno-orijentisanu šemu.

Većina radova čiji su glavni rezultati prezentovani u ovom odeljku doktorske teze, deli isti principijelni cilj dobijanja novog modela koji reprezentuje originalnu bazu podataka, nekad u objektno-orijentisanoj notaciji, nekad kao ER šemu baze podataka, ili čak kao novu relacionu bazu podataka, i u ovoj tački završavaju proces.

Slično prethodno opisanim pristupima postupak reverznog inženjeringa u IIS*Ree, koji je nastao kao rezultat istraživanja u ovoj doktorskoj disertaciji, obuhvata analizu rečnika podataka, analizu samih podataka zatečenih u nasleđenoj bazi podataka i interakciju sa korisnicima. Programski kod nije razmatran.

Za razliku od prethodno opisanih pristupa u alatu IIS*Ree podržan je čitav proces reinženjeringa baze podataka na čijem izlazu može biti:

- konceptualni model podataka opisan PIM konceptima IISCase-a,
- implementacioni opis šeme baze podataka za različite relacione sisteme za upravljanje bazama podataka, ili
- XML specifikacija informacionog sistema (XML specifikacija tipova formi i XML specifikacija šeme baze podataka).

U radu [Polo07] autori prikazuju metod i alat *Relational Web* koji integriše kompletni process reinženjeringa, uključujući reverzni inženjering, restrukturiranje i *forward* inženjering. Polazna tačka je relaciona baza podatka, čija fizička šema je u reverznom inženjeringu ekstrahovana u dijagram klasa koji predstavlja konceptualnu šemu. U restruktuiranju, dijagram klasa se obrađuje od strane korisnika i prosleđuje kao ulaz u automatizovani proces generisanja koda za višeslojne sisteme koristeći različite programske jezike i platforme. Alat može da generiše četiri različita tipa aplikacije za četiri različita SUBP-a: *Cache Intersystems*, *Microsoft Access*, *Oracle* i SQL *Server*. Takođe, alat uključuje mogućnost migracije baze podataka jednog proizvođača u drugi. Arhitektura alata omogućava dodavanje novih generatora koda za druge programske jezike. Polazna tačka za reinženjering je fizička šema baze podataka koja je prevedena na meta-model nezavisan od proizvođača, tj. u logičku šemu, a zatim prevedena u dijagram klasa koji predstavlja moguću konceptualnu šemu baze podataka. Ovaj dijagram se zatim uzima kao polazna tačka za generisanje programskog koda izvršne aplikacije za četiri različite platforme.

2.2.6. MDE i reinženjering

Model Driven Engineering nije primenljiv samo za kreiranje novih softverskih sistema, već se može koristiti, takođe, za modernizaciju ili reinženjering postojećih, nasleđenih sistema [ADM, Ulrich10]. Ima puno istraživanja i radova koji se bave reinženjeringom programskog koda, između ostalih u [FavreL10, FavreL10a, Reus06a, Barbi10]. Razni meta-modeli su već standardizovani na način da podržavaju nasleđene softvere, izrađene npr. u jezicima kao što su COBOL ili C. Ovi standardizovani meta-modeli pripadaju CWM standardu [CWM].

Reinženjering zahteva ekstrakciju modela (*model harvesting*), tj. proces u kojem se modeli dobijaju iz softverskih artefakata kao što su programski kod ili baza podataka. Kada se ekstrahuje model, MDE tehnike transformacije modela mogu biti korišćene za modernizaciju sistema.

U MD reinženjeringu, za modernizaciju nasleđenih (*legacy*) sistema primenjuje se MDA pristup, što podrazumeva da se jedan ili više modela, koji predstavljaju reprezentaciju nasleđenih sistema, prevode nizom transformacija u nove modele, svaki put bliže ciljnom sistemu.

Već je pomenuto da je OMG, takođe, pokrenula novu radnu grupu pod nazivom *Architecture Driven Modernization* (ADM), koja ima za cilj da integriše reverzni inženjering i MDA [ADM, Ulrich10, Guzm07].

Autori rada [Boron05], primenjujći MD pristup, definišu okruženje za upravljanje modelima pod nazivom MOMENT ("*MOdel manage-MENT*"). U MOMENT okruženju, meta-modeli i transformacije specificirani su pomoću algebre tzv. *Maude* sistema. Meta-model MOMENT-a je apstraktna reprezentacija neke vrste meta-podataka. MOMENT je implementacija tehnike prezentovane u radovima [Perez02 i Boron04].

U radu [Reus06a] predstavljen je skup alata koji su razvijeni za ekstrakciju modela iz postojećeg programskog koda. Dobijeni modeli se prikazuju u komercijalnom MDA-alatu ArcStyler.

MoDisco (*Model Discovery*) je proširivo okruženje za MD reverzni inženjering, napravljen kao *Eclipse Generative Modeling Technology* (GMT) komponenta [Ulrich10]. Namenjen je da olakša razvoj alata „pronalazača" („*discoverers*" u terminologiji MoDisco-a)

za dobijanje modela iz nasleđenih sistema pri modernizaciji. Dostupni su „pronalazači“ za Javu i C#.

Autori rada [Diaz11] predstavljaju *Schemol*, namenski jezik za ekstrakciju modela iz baze podataka za Web2.0 aplikacije, kao što su baze podataka blogova ili *web* sajtova socijalnih mreža. *Schemol* je dodatni MoDisco „pronalazač“ za podatke u relacionim bazama podataka.

2.2.7. Rezime prezentovanog stanja u oblasti istraživanja

U ovom poglavlju dat je pregled aktuelnih koncepata, pristupa i alata iz oblasti MDE. Predstavljena je MDA i niz njenih specifikacija, poznatih pod nazivom OMG standardi, bitnih za istraživanja u ovoj doktorskoj disertaciji: MOF, XMI, QVT i OCL. Prezentovani su pojmovi model, meta-model i transformacija modela. Posebna pažnja posvećena je transformacijama modela koje imaju ključnu ulogu u MDE. Prikazana je klasifikacija pristupa i jezika za transformaciju modela, kao i detaljnije karakteristike nekih od njih. Transformacije modela su osnovni predmet istraživanja u ovoj doktorskoj tezi.

U ovom poglavlju, takođe su prikazani koncepti, pristupi i alati iz oblasti reinženjeringa. Obuhvaćeni su tradicionalni pristupi reverznom inženjeringu, kao i pristupi koji se oslanjaju na MDE i MDA. Predstavljen je životni ciklus reinženjeringa, koji obuhvata reverzni inženjering, restrukturiranje i *forward* inženjering, kao i ADM model potkovice. Posebna pažnja posvećena je reverznom inženjeringu baza podataka.

Pristup u ovoj doktorskoj disertaciji zasniva se na analizama pronađenih pristupa u literaturi i odabira onih koji odgovaraju postavljenim ciljevima.

U okviru ove doktorske disertacije biće predložen metodološki pristup i softversko okruženje IIS*Ree, namenjeni unapređenju procesa reinženjeringa informacionih sistema. Izabrani pristup se zasniva na analizi i transformacijama opisa baza podataka.

U trećem poglavlju biće prezentovan način primene principa MDE u reinženjeringu IS-a, predložen u okviru ove doktorske disertacije. Biće predložena jedna interpretacija modela potkovice koja obuhvata čitav proces reinženjeringa: reverzni inženjering, restrukturiranje i *forward* inženjering. Primenjeni model potkovice je zasnovan na upotrebi MDA principa i tehnika.

Postupak *forward* inženjeringa biće detaljno opisan u četvrtom poglavlju, dok je u petom poglavlju opisan softverski alat IIS*Ree, originalno nastao kao rezultat istraživanja ove doktorske disertacije. IIS*Ree je zasnovan na MDSD principima, u cilju omogućavanja visokog nivoa automatizacije procesa reinženjeringa informacionih sistema.

U poglavlju sedam biće detaljno prikazan metod, takođe originalno nastao u okviru istraživanja ove doktorske teze, koji se primenjuje u postupku konceptualizacije modela ekstrahovanog iz nasleđene baze podataka koja je predmet procesa reinženjeringa IS-a. Metod se sastoji od niza transformacija zasnovanih na meta-modelima. Kompletna specifikacija svih meta-modela koji učestvuju u transformacijama biće data u šestom poglavlju.

3. Principi primene MDA u reinženjeringu IS

Model Driven Software Engineering osim što je primenljiv za kreiranje novih softverskih sistema, može se, takođe, primeniti za modernizaciju ili reinženjering postojećih sistema. Reinženjering je pristup nastao sa ciljem da se obezbede neophodni koncepti, metode i alati za proces evolucije informacionih sistema u pogledu migracije i ponovne upotrebe njegovih artefakata.

Osnovna hipoteza ove doktorske disertacije je da se proces reinženjeringa IS može realizovati pomoću *Model Driven* principa. U skladu sa MDA, kao najzrelije formulacije MDSE paradigme, kompletan proces reinženjeringa koji, prema [Chikov90], obuhvata reverzni inženjering, restrukturiranje i *forward* inženjering, može se posmatrati kao niz transformacija modela, koji uključuje transformacije od više konkretnog ka više apstraktnom modelu (u kontekstu reverznog inženjeringa) i obrnuto (od dizajna do programskog koda u kontekstu *forward* inženjeringa). Na početku reinženjeringa zahteva se ekstrakcija modela tj. proces u kojem se model kreira iz drugih softverskih artefakata, na primer programskog koda ili baze podataka. Kada se model ekstrahuje, mogu se koristiti tehnike transformacije modela za obezbeđivanje modernizacije IS-a.

Kao referentni model modernizacije, u ovoj doktorskoj tezi je korišćen ADM model potkovice prikazan na slici 2.8. Po modelu potkovice, u procesu reverznog inženjeringa, sistem se predstavlja postepeno na sve višim nivoima apstrakcije. Niz transformacija započinje transformacijom ekstrahovanog modela, koji opisuje specifikaciju implementacije sistema na konkretnoj platformi i predstavlja PSM model. Nakon toga slede transformacije čiji ulaz i izlaz predstavljaju modeli koji opisuju sistem sa sve višim stepenom nezavisnosti od platforme. Niz transformacija završava se transformacijom koja generiše model funkcionalnosti sistema koji je nezavisan od platforme na kojoj će biti implementiran i koji predstavlja PIM model. PIM koji je korišćen u ovoj doktorskoj disertacji je model zasnovan na konceptu tipa forme. Skup tipova formi je platformski nezavisan pogled na IS iz perspektive krajnjeg korisnika. Zbog toga model IS-a kreiran pomoću IIS*Case-a, reprezentovan skupom specifikacija tipova formi, može biti klasifikovan kao PIM. Detaljan opis alata IIS*Case i njegovih koncepata dat je u narednom poglavlju.

U postupku *forward* inženjeringa, počinje se od modela, nastalog kao rezultat reverznog inženjeringa, koji je nezavisan od specifične platforme za implementaciju (PIM). Ovaj model se može menjati i unapređivati (restruktuirati), a zatim, ponovo nizom transformacija transformisati u programski kod ili implementacioni opis šeme baze podataka inoviranog IS (PSM). Prema MDA principima svaki model mora biti opisan u skladu sa nekim meta-modelom.

Posebna pažnja u istraživanjima ove doktorske disertacije posvećena je transformacijama šema baza podataka, primenom MDA, odnosno ADM principa, u cilju reinženjeringa IS, jer se baze podataka smatraju jednim od najosnovnijih artefakata sistema, koje sadrže značajne informacije vezane za postojeći informacioni sistem. Izabrana je relacionia baza podataka kao najčešće korišćen nasleđeni izvor u reinženjeringu IS.

U ovom poglavlju ukratko je opisan MDA proces reinženjeringa zasnovan na integraciji tradicionalnih tehnika reverznog inženjeringa, naprednih tehnika meta-modelovanja i transformacijama zasnovanim na meta-modelima. U odeljku 3.1 biće dat predlog klasifikacije meta-modela šema baza podataka na osnovu koje su klasifikovani meta-modeli na kojima su zasnovane transformacije modela nastale kao rezultat istraživanja u ovoj doktorskoj disertaciji. U odeljku 3.2 predložena je interpretacija modela potkovice, dok je princip primene MDA na proces reverznog inženjeringa detaljno prikazan u odeljku 3.3.

3.1. Klasifikacija meta-modela šema baza podataka

Osnovni predmet istraživanja ove doktorske teze su transformacije šema baza podataka u cilju reinženjeringa IS, primenom MDA principa i tehnika, koje podrazumevaju korišćenje meta-modela. U ovom odeljku biće dat predlog klasifikacije meta-modela šema baza podataka na različitim nivoima apstrakcije.

Šema baze podataka mora biti u skladu sa nekim modelom podataka. Prema definiciji Date-a i Darwen-a u [Date06] koja je revidirana u [Eass07]: „*Model podataka je apstraktna, samostalna, nezavisna od implementacije definicija elemenata četvorke skupova (T, S, O, C) koji zajedno čine apstraktnu mašinu sa kojom korisnici baza podataka vrše interakciju, gde je: T skup tipova podataka; S skup tipova strukture podataka; O skup tipova operacija nad podacima; i C je skup tipova ograničenja integriteta.*"

U [Elmas11] definišu kategorizaciju modela podataka na osnovu tipova koncepata koji se koriste za opis strukture baze podataka. Pod strukturom baze podataka podrazumevaju tipove podataka, veze i ograničenja koja se primenjuju na podatake. U prvu grupu spadaju konceptualni modeli podataka ili modeli na visokom nivou (*high-level*) koji obezbeđuju koncepte koji su bliski načinu na koji mnogi korisnici percipiraju podatake. Neki od dobro poznatih konceptualnih modela podataka su: ER (*entity-relationship*), EER (*extended* ER) ili objektni model. U ovu grupu modela na visokom nivou spada i model tipova formi koji se u ovoj doktorskoj disertaciji koristi kao model podataka na kojem je zasnovana šema baze podataka, cilj procesa reverznog inženjeringa.

U drugu grupu spadaju fizički modeli podataka ili modeli na niskom nivou (*low-level*) koji obezbeđuju koncepte koji opisuju detalje kako su podaci smešteni na kompjuterskom mediju za čuvanje podataka. Koncepti modela niskog nivoa su namenjeni stučnjacima iz oblasti računarstva, a ne krajnjim korisnicima.

Treću grupu modela podataka, koja se nalazi između prve dve, čini klasa reprezentacionih ili implementacionih modela podataka koji obezbeđuju koncepte koji mogu biti lako razumljivi krajnjim korisnicima, ali nisu ni suviše udaljeni od načina kako su podaci organizovani u memoriji. Reprezentacioni modeli podataka sakrivaju mnogo detalja vezanih za smeštanje podataka na disk, ali se mogu implementirati na kompjuterskim sistemima direktno. Ovde je termin implementacioni korišćen u smislu ukazivanja na dostupne modele podataka u komercijalnim sistemima baza podataka koji su, takođe, zasnovani na nekom modelu podatka. Reprezentacioni ili implementacioni modeli podataka su modeli koji se najviše koriste u tradicionalnim komercijalnim sistemima za upravljanje bazama podataka. Najčešće korišćeni su relacioni model ili, u poslednje vreme, objektno-relacioni model.

Relacioni model podataka je, u fokusu kontinualnog procesa standardizacije. *Structured Query Language* (SQL), koji je trenutno dostupan u većini komercijalnih i *open-source* sistema za upravljanje bazama podataka, još je od 1986. godine postao standardni jezik za komercijalne relacione sisteme za upravljanje bazama podataka. Kontinualni proces standardizacije je rezultovao SQL standardima počevši od SQL-86 pa do tekuće verzije SQL:2011 [SQL:2011]. Od SQL:1999 standard je podeljen na osnovnu specifikaciju i

specijalizovane ekstenzije. Osnovnu specifikaciju bi trebalo da implementiraju svi proizvođači relacionih sistema za upravljanje bazama podataka koji su SQL kompatibilni, što bi omogućilo prenosivost SQL koda. Međutim, problemi prenosivosti SQL koda između relacionih sistema za upravljanje bazama podataka još uvek postoje, usled nedostatka potpune usaglašenosti sa standardom i sopstvenih proširenja koje relacioni sistemi za upravljanje bazama podataka podržavaju.

Iz ovoga se može zaključiti da čak i mapiranja između SQL opisa šema baza podataka ekstrhovanih iz rečnika podataka relacionih sistema za upravljanje bazama podataka različitih proizvođača mogu biti ozbiljan problem. Ovo je jedan od razloga zbog kojeg je za rešenje problema reinženjeringa postojećih, nasleđenih baza podataka, korišćeno meta-modelovanje, odnosno definisanje apstraktnih sintaksi jezika za modelovanje šema baza podataka i transformacije modela zasnovane na meta-modelima.

Zbog velike rasprostranjenosti korišćenja relacionih SUBP u prethodnim decenijama, kao predmet reinženjeringa u ovoj doktorskoj diseratciji uzeta je baza podataka zasnovana na relacionom modelu podataka. U prvoj fazi reinženjeringa, reverznom inženjeringu, cilj je da se od modela baze podataka čija specifikacija je zapisana u rečniku podataka postojeće, nasleđene, baze podataka nizom model-u-model transformacija dobije konceptualni model zasnovan na tipovima formi, a zatim da se od tog modela, nakon izmene ili restruktuiranja, u postupku *forward* inženjeringa opet nizom model-u-model transformacija generiše implementacioni opis ili fizička šema inovirane baze podataka. U transformacijama učestvuje nekoliko modela baza podataka na različitim nivoima apstrakcije. Svaki od modela mora da bude u skladu sa odgovarajućim meta-modelom.

U tabeli 3.1 prikazan je jedan predlog klasifikacije meta-modela šema baza podataka na različitim nivoima apstrakcije, prilagođen iz [Ristic13]. U skladu sa Elmasrijevom podelom modela podataka ([Elmas11]) na kojima su šeme baze podatka zasnovane, meta-modeli (MM) su klasifikovani u tri grupe:

i) meta-modeli konceptulanih šema baze podataka;
ii) meta-modeli implementacionih šema baze podataka:
    a. meta-modeli logičkih šema baze podataka,
    b. meta-modeli standardnih šema baze podataka i
iii) meta-modeli fizičkih šema baze podataka zavisnih od konkretnog proizvođača.

Ovde je uveden termin generičke šeme baza podataka koji predstavlja šeme baze podataka koje su u skladu sa teoretskim definicijama modela podataka. Neki od generičkih meta-modela šema baza podataka opisuju konceptualne šeme baza podataka, kao što su ER ili EER meta-modeli šema baze podataka, dok drugi opisuju logičke šeme baze podatka, kao što su relacioni ili OR meta-modeli šema baze podataka.

Pošto usklađenost sistema za upravljanje bazama podataka sa SQL standardom po pravilu nije potpuna, osim standardnih meta-modela šema baza podataka za specifične SQL standarde, uvedena je i klasa meta-modela fizičke šeme baze podataka koji su konkretni komercijalni ili *open source* SUBP.

U tabeli 3.1 dat je pregled raspoređivanja meta-modela i modela po nivoima četvoroslojne MOF arhitekture. Svaka vrsta tabele predstavlja jedan meta-nivo. Na M0 nivou se, u zavisnosti od nivoa apstrakcije modela na M1 nivou koji ih reprezentuje, nalazi instanca baze podataka ili logička struktura baze podataka. Na M1 nivou nalaze se modeli, odnosno šeme baze podataka koje su u skladu sa meta-modelima na M2 nivou. Na primer, na M1 se nalaze šema baze podataka implementirana na Oracle 10g SUBP-u koje su u skladu sa Oracle 10g meta-modelom ili ER šema baze podataka koja je u skladu sa meta-modelom zasnovanom na teoretskim definicijama ER modela podataka. Na vrhu se nalazi MOF meta-metamodel, propisan od strane OMG standarda. U ovoj doktorskoj disertaciji korišćena je njegova najpoznatija implementacija, Ecore.

Tabela 3.1 Klasifikacija meta-modela šema baze podataka

<table>
<tr><th>MOF nivo</th><th colspan="4">MOF arhitektura</th></tr>
<tr><td>M3</td><td colspan="4">EMOF/CMOF/Ecore</td></tr>
<tr><td rowspan="3"></td><td rowspan="2">MM konceptualne šeme bp</td><td colspan="2">MM implementacione šeme bp</td><td>MM fizičke šeme bp</td></tr>
<tr><td>MM logičke šeme bp</td><td rowspan="2">MM šeme bp zasnovane na standardu</td><td rowspan="2">MM fizičke šeme bp zasnovane na spec. proizvođača SUBP-a</td></tr>
<tr><td colspan="2">MM generičke šeme bp</td></tr>
<tr><td></td><td>(1)</td><td>(2)</td><td>(3)</td><td>(4)</td></tr>
<tr><td>M2</td><td>MM ER šeme bp,
MM EER šeme bp,
MM O šeme bp,
MM PIM koncepata IIS*Case-a,
…</td><td>MM OR šeme bp,
MM Relacione šeme bp,
…</td><td>MM SQL:1999 šeme bp,
MM SQL:2003 šeme bp,
MM SQL:2011 šeme bp
…</td><td>MM Oracle10g šeme bp,
MM MySQL šeme bp,
MM dBase III+ šeme bp,
…</td></tr>
<tr><td>M1</td><td>ER šema bp 1,
ER šema bp 2,
Objektna šema bp 1,
Objektna šema bp 2,
Model tipova formi 1,
Model tipova formi 2,
…</td><td>Relaciona šema bp 1,
Relaciona šema bp 2,
OR šema bp 1,
…</td><td>SQL:2011 šema bp 1,
SQL:2003 šema bp 1,
SQL:2003 šema bp 2,
SQL:1999 šema bp 1,
…</td><td>Oracle10g šema bp 1,
MySQL šema bp 1,
MySQL šema bp 2,
…</td></tr>
<tr><td>M0</td><td colspan="3">Logička struktura podataka baze podataka</td><td>Instanca baze podataka</td></tr>
</table>

Iz tabele 3.1 vidi se da modeli baza podataka mogu biti na nekoliko različitih nivoa apstrakcije: nivo konceptualne šeme baze podataka, nivo logičke šeme baze podataka, nivo implementacione šeme baze podataka, kao i nivo fizičke šeme baze podataka.

Posmatrano iz ugla MDA inicijative, gde se prema [MDA03] modeli klasifikuju na:

- računarski nezavisne modele (CIM),
- modele nezavisne od platforme (PIM) i
- modele specifične za određenu platformu (PSM),

modeli se mogu razlikovati u tome koliko sadrže informacija specifičnih za platformu. Platforma se ne treba posmatrati samo kao izvršna infrastruktura. Atkinson i Kuhne u [Atkin05] platformu vide "kao neki sistem sposoban za podršku ispunjenja nekog cilja vezanog za softversku aplikaciju". Oni naglašavaju da platformska nezavisnost nije binarna osobina. Ako se posmatraju modeli na ovaj način, tada može da postoji više modela koji su PSM, ali sa različitim stepenom platformske nezavisnosti.

Takođe, u [MDA03] se smatra da svrstavanje modela u PIM ili PSM kategoriju zavisi od konteksta izabrane platforme. Na primer, šema baze podataka iskazana putem koncepata relacionog modela podataka može biti posmatrana kao model tipa PSM, jer se koristi relaciona tehnologija, a ne recimo, objektno-relaciona, XML, ili neka druga tehnologija. S druge strane, može se reći da je u pitanju model tipa PIM, jer je ovakav model nezavisan od konkretnog sistema za upravljanje bazama podataka. Modeli tipa PSM kreiraju se na osnovu modela tipa PIM, najčešće putem alata koji automatizovano obavljaju transformacije. Svaka transformacija obavlja se na osnovu modela tipa PIM i dodatnih informacija koje su specifične za konkretnu platformu, u svrhu generisanja koda za ciljnu platformu.

Ako se tabela 3.1 posmatra po kolonama, u skladu sa OMG klasifikacijom apstraktnih nivoa, modeli se mogu svrstati na sledeći način:

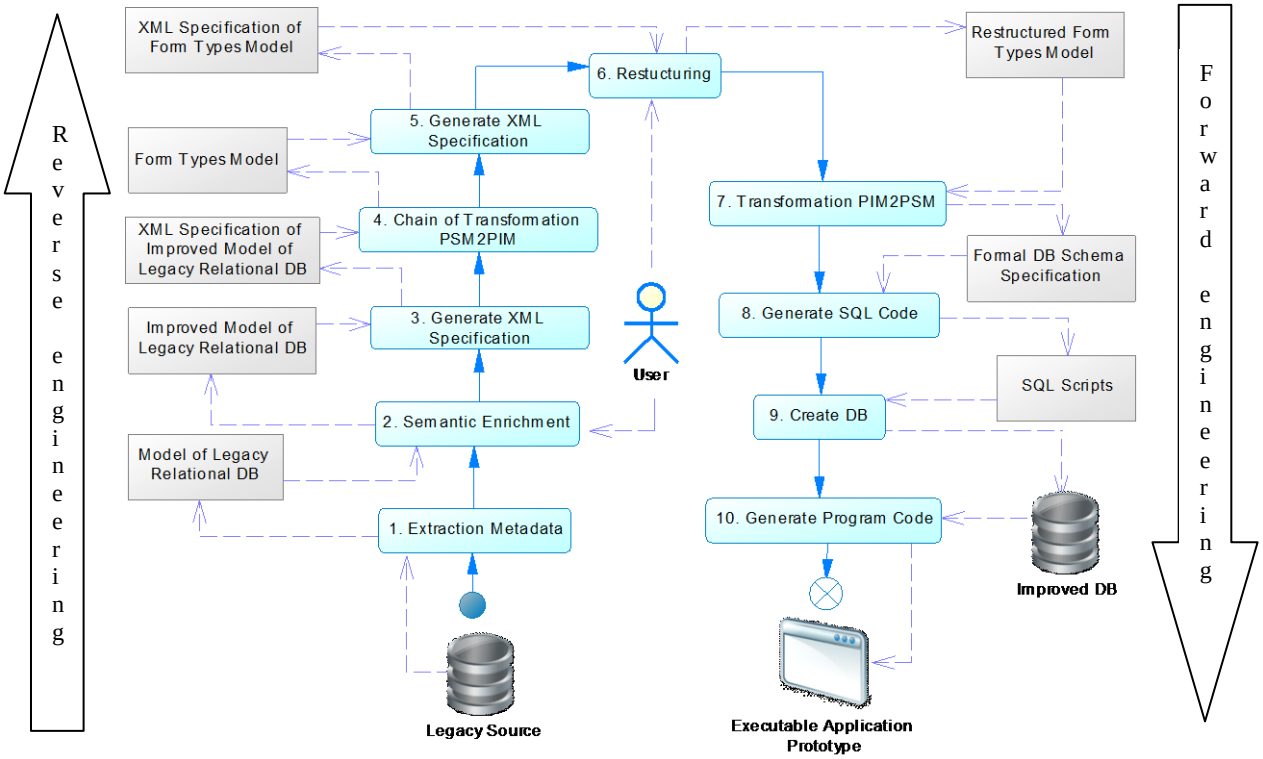
- modeli iz četvrte kolone mogu se smatrati PSM modelima, jer su specifični za određenu platformu i određenog proizvođača,
- modeli iz treće kolone, zasnovani na standardu, mogu se smatrati, takođe, PSM ali sa višim stepenom nezavisnosti od platforme jer su nezavisni od konkretnog proizvođača,
- modeli iz druge kolone, zasnovani na teoretskim principima modela podatka, mogu se posmatrati kao PSM ili PIM, u zavisnosti od konteksta izabrane platforme i
- modeli iz prve kolone mogu se posmatrati kao PIM.

Nivoi apstrakcije će, u postupku reveznog inženjeringa, omogućiti postepenu transformaciju modela od nižeg ka višem apstraktnom nivou, kao i obrnuto, u postupku *forward* inženjeringa. U narednom odeljku ukratko je opisan čitav navedeni postupak.

3.2. Model potkovice reinženjeringa IS

Klasifikacija i distribucija modela baza podataka u smislu višeslojne MOF arhitekture će omogućiti sistematičan pristup za mapiranje specifikacija između različitih modela/meta-modela i razvoj odgovarajućih M2M ili M2C transformacija u cilju reinženjeringa tj. modernizacije postojeće nasleđene baze podataka.

Na slici 3.1 prikazan je uprošćeni dijagram aktivnosti, koji opisuje proces transformacije šeme baze podataka u cilju reinženjeringa IS, predložen u okviru ove doktorske disertacije. Predloženi model prati ADM model reinženjering potkovice [ADM].



Slika 3.1. Dijalgram aktivnosti u obliku reinženjering potkovice predložen u ovoj doktorskoj disertaciji

Na slici 3.1 prikazan je kompletan proces reinženjeringa koji se sastoji od tri glavne faze:

1. faze reverznog inženjeringa za dobijanje modela koji opisuje izvornu bazu podataka,
2. faze restruktuiranja za transformaciju ovog izvornog modela radi izmene i unapređenja, kao i
3. faze *forward* inženjeringa za generisanje prototipa aplikacije inoviranog IS-a.

Polazna tačka procesa reinženjeringa je fizička šema baze podataka, smeštena u rečniku podataka RSUBP-a nasleđene baze podataka. Ova fizička šema baze podataka u skladu je sa meta-modelom konkretnog SUBP-a i pripada modelima iz četvrte kolone tabele 3.1. Ova šema baze podataka predstavlja ulaz u proces reverznog inženjeringa.

Kako je *forward* inženjering realizovan u ranijim istraživanjima autora doktorske disertacije i drugih autora tima koji se bavi razvojem okruženja za projektovanje i implementaciju IS-a, IIS*Studio, u ovoj doktorskoj disertaciji posebna pažnja posvećena je postupku reverznog inženjeringa.

Reverzni inženjering ima dve faze. Prva faza je ekstrakcija modela iz postojeće, nasleđene baze podataka, a druga je njegova konceptualizacija tj. dobijanje konceptualnog modela izvorne baze podataka.

Prve dve aktivnosti sa slike 3.1, *Extraction Metadata* i *Semantic Enrichment*, obavljaju zadatak ekstrakcije modela. Prva aktivnost ima za cilj da kreira model koji reprezentuje ulaznu bazu podataka. Ova aktivnost se obavlja tako što se pristupa rečniku podataka RSUBP-a nasleđene baze podataka i čitaju meta-podaci. Podaci koji se ekstrahuju iz rečnika podataka su o: tabelama i njihovim kolonama, primarnim ključevima, *check* ograničenjima, ograničenjima jedinstvenih vrednosti obeležja, stranim ključevima i podržanim tipovima podatataka izabranog SUBP-a. Pročitani meta-podaci smeštaju se u repozitorijum alata putem kojeg se obavlja reinženjering, čija realizacija će biti predložena u poglavlju 5.

Model, nastao nakon ekstrakcije meta-podataka iz rečnika podataka, u skladu je sa delom standarda koji podržava većina SUBP-ova i pripada modelima iz treće kolone tabele 3.1.

Druga aktivnost *Semantic Enrichment* unapređuje ovaj model, tako što vrši analizu ekstrahovanih meta-podataka i samih podataka koji postoje u nasleđenoj bazi podataka. Ovim postupkom obezbeđuju se informacije koje se ne mogu dobiti jednostavnim čitanjem iz rečnika podataka. Na osnovu toga, model se proširuje informacijama o ograničenjima inverznog referencijalnog integriteta i otklanjaju se neusaglašenosti vezane za homonime. U okviru ove aktivnosti neophodna je pomoć korisnika-projektanta koji obavlja reverzni inženjering. Projektant interaktivno kontroliše proces ekstrakcije modela tj. relacione šeme baze podataka iz postojeće, nasleđene baze podataka, dajući neophodne dodatne informacije. Ove aktivnosti biće detaljno objašnjene u petom poglavlju.

Zbog potrebe za interakcijom sa korisnikom, ove dve aktivnosti, koje obavaljaju zadatak ekstrakcije modela iz postojeće baze podataka, obavljaju se poluautomatski. Nemogućnost potpune automatizacije ove faze procesa reinženjeringa značajno je uticala na njenu implementaciju, koja je realizovana pomoću tradicionalnih tehnika odnosno pomoću GPL Java.

Unapređeni model, nastao nakon prethodno opisane dve aktivnosti, predstavlja ulazni model u niz transformacija koje obavljaju zadatak konceptualizacije modela. Za razliku od ekstrakcije modela, ovaj zadatak se može potpuno automatizovati i realizovati u skladu sa MDA principima, pomoću transformacija zasnovanih na meta-modelima, implementiranih jezikom ATL.

Usled korišćenja dva okruženja i različitih tehnika za implementaciju čitavog procesa reinženjeringa šema baza podataka, javlja se problem povezivanja modula u kojima se različite faze reinženjeringa odvijaju. U cilju prevazilaženja ovog problema, potrebno je obezbediti mehanizam koji omogućava razmenu podataka između dva okruženja. Pošto je XML prihvaćen kao standardni format za smeštanje i upravljanje podacima, modeli ekstrahovani iz postojeće, nasleđene baze podataka, koji predstavljaju ulaz u automatizovani proces konceptualizacije, kao i modeli koji će nastati kao rezultat ovog procesa, serijalizuju se kao XML specifikacije (aktivnosti 3 i 5 sa dijagrama na slici 3.1). Ovi modeli se serijalizuju u XML specifikacije na osnovu predefinisanih XML šema koje se nalaze u prilogu A.

U skladu sa prehtodnom činjenicom, proces reverznog inženjeringa se nastavlja tako što se za unapređeni model ekstrahovan iz postojeće baze podataka, smešten u repozitorijumu alata za reinženjering, generiše XML specifikacija koja ga opisuje. Ovaj model, opisan XML specifikacijom, koji je u skladu sa delom standarda koji podržava većina SUBP-ova, predstavlja ulaz u automatizovani proces niza transformacija koji je na dijagramu sa slike 3.1 prikazan kao četvrta aktivnost, *Chain of Transformation PSM2PIM*. Ova aktivnost translira tehnološki specifičan model, zavisan od relacione tehnologije na kojoj su zasnovani RSUBP-ovi, u tehnološki nezavisan model zasnovan na konceptu tipa forme. Model nastao kao rezultat ove aktivnosti pripada modelima iz prve kolone tabele 3.1. Ova aktivnost je složena jer se sastoji od nekoliko transformacija putem kojih se postepeno od PSM višeg stepena platformske zavisnosti, preko PSM-ova nižeg stepena platformske zavisnosti, konačno dolazi do PIM. Da bi model mogao da se koristi u daljem postupku reinženjeringa, kao i ulazni model, mora biti predstavljen kao XML specifikacija. Sve transformacije niza transformacija *Chain of Transformation PSM2PIM* detaljnije će biti prikazane u narednom odeljku.

Nakon dobijenog konceptualnog modela zasnovanog na konceptu tipa forme, sledi aktivnost restruktuiranja, prikazana na dijagramu kao šesta aktivnost, *Restucturing*. Projektant, takođe, može da menja model, tako što će dodavati nove elemente, usled novih poslovnih zahteva, ili briše nepotrebne koncepte nastale u automatskom procesu transformacije.

Nakon toga, slede aktivnosti koje pripadaju fazi *forward* inženjeringa. Cilj *forward* inženjeringa je da se dobije prototip aplikacije inoviranog IS-a. Pomoću aktivnosti *Transformation PIM2PSM*, koja predstavlja sedmu aktivnost na dijagramu sa slike 3.1, izmenjena konceptualna šema transformiše se u model koji predstavlja formalni opis nove relacione šeme baze podataka. Na osnovu ovog modela ponovo se generiše implementacioni opis šeme baze podataka za izabranu platformu u vidu SQL koda, na osnovu kojeg se kreira nova baza podataka (aktivnosti 8 i 9 sa dijagrama na slici 3.1). U finalnom koraku, projektantu se pruža mogućnost generisanja transakcionih programa, odnosno izvršnih softverskih specifikacija aplikativnih sistema nad novokreiranom bazom podataka (aktivnost 10 na dijagramu sa slike 3.1).

Opisani postupak se odnosi na slučaj kada je samo jedna baza podataka na ulazu. Ukoliko postoji više, moguća je integracija u jednu šemu baze podataka. Integracija šema se ne vrši prostim uniranjem, već se vrši konsolidacija i analiza usaglašenosti podšema sa integrisanom šemom.

Postupak *forward* inženjeringa detaljno je opisan u narednom poglavlju, dok je u poglavlju pet predloženo rešenje alata koji implementira prethodno opisani postupak.

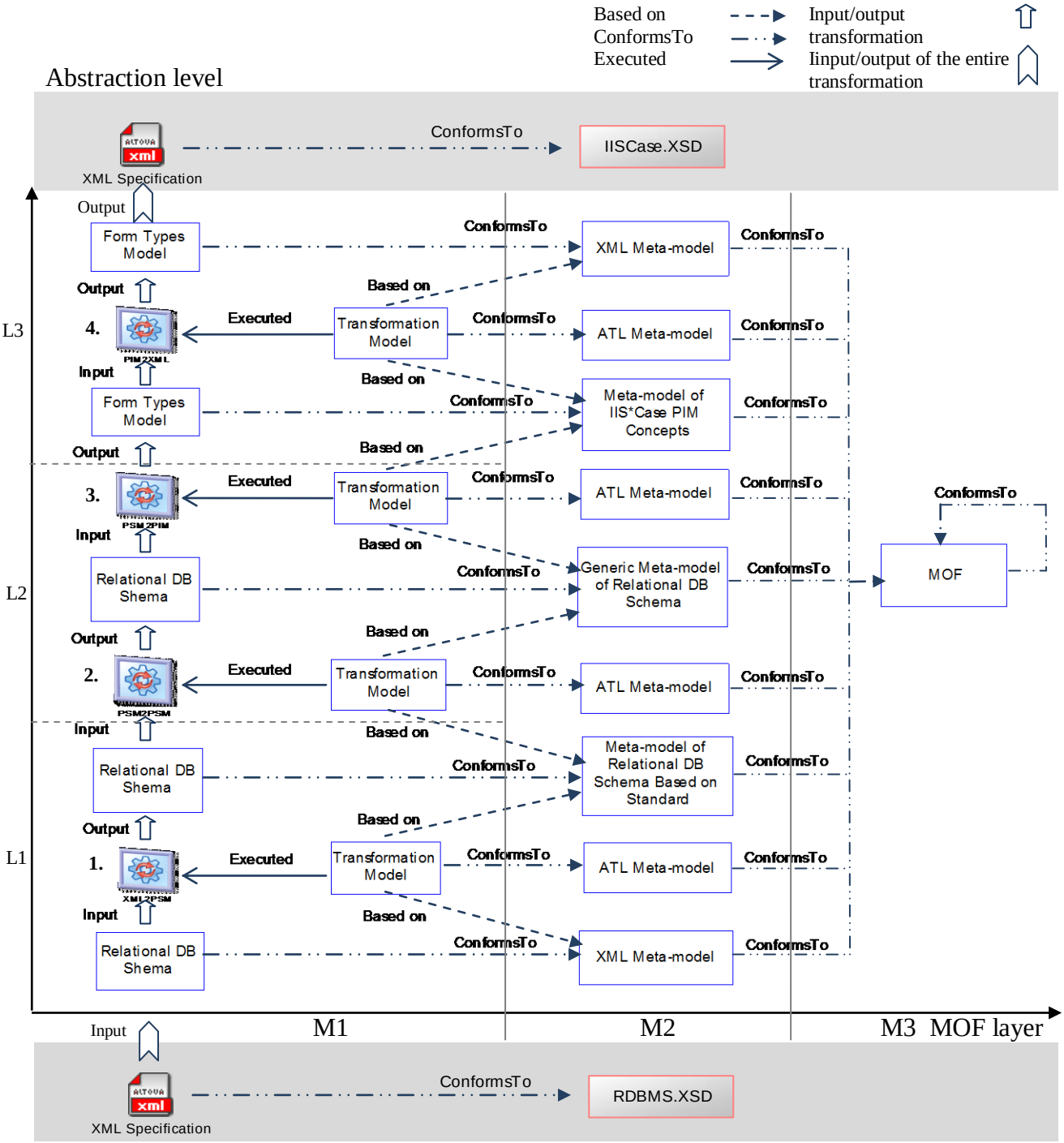
3.3. Konceptualizacija modela

Glavni predmet istraživanja ove doktorske disertacije su transformacije šema baza podataka u cilju reinženjeringa IS. Od posebnog interesa su transformacije kojima se model ekstrahovan iz postojeće, nasleđene baze podataka prevodi u konceptualni model.

U prethodnom odeljku postupak konceptualizacije modela, kao dela procesa reverznog inženjeringa, prikazan je na dijagramu aktivnosti sa slike 3.1, u vidu složene aktivnosti koja obuhvata niz transformacija (*Chain of Transformations PSM2PIM*). Transformacije su zasnovane na meta-modelima i ne zahtevaju interakciju sa korisnikom-projektantom. Postupak je potpuno automatizovan.

Modeli i transformacije koje pripadaju zadatku konceptualizacije prikazani su na slici 3.2 u koordinatnom sistemu u kojem se na vertikalnoj osi nalaze nivoi (L1-L3) u koje su svrstani modeli na istim ili različitim MDA nivoima apstrakcije sa različitim stepenom zavisnosti od

platforme, dok su na horizontalnoj osi meta-nivoi MOF arhitekture, od M1 do M3. Na slici su, takođe, prikazane i sve transformacije, putem kojih se dolazi do konceptualnog modela tipova formi. Ovaj model, koji predstavlja cilj reverznog inženjeringa, opisuje šemu baze podataka koja je nezavisna od platforme.



Slika 3.2. Deo modela potkovice koji se odnosi na reverzni inženjering predložen u ovoj doktorskoj disertaciji

Postupak počinje korišćenjem modela, nastalog ekstrakcijom iz postojeće baze podataka koji je smešten u repozitorijumu alata za reinženjering. Model je u skladu sa meta-modelom relacione šeme baze podataka, zasnovanom na konceptima SQL standarda sa kojim je kompatibilna većina RSUBP-ova (treća kolona iz tabele 3.1), zavisan je od platforme (PSM),

ali ne zavisi od konkretnog proizvođača SUBP-a. Ovaj model, koji predstavlja ulaz u automatizovani proces transformacije, serijalizuje se u XML specifikaciju, na osnovu predefinisane XML šeme (RDBMS.XSD). Na slici 3.2, deo koji se odnosi na ulaz i izlaz iz automatizovanog postupka transformacije osenčen je sivom bojom.

Model nastao ekstrhovanjem iz nasleđene baze podataka, opisan XML specifikacijom, pripada XML tehnološkom prostoru, dok se niz transformacija izvršava nad modelima definisanim u MOF tehnološkom prostoru. Iz ovog razloga prva transformacija niza transformacija, na slici 3.2 označena rednim brojem 1 (XML2PSM), potrebna je za prevazilaženje problema različitih tehnoloških prostora. Model koji je u skladu sa XML meta-modelom transformiše se u model koji je u skladu sa meta-modelom relacione šeme baze podataka, zasnovanom na konceptima SQL standarda sa kojim je kompatibilna većina RSUBP-ova. Transformacija je horizontalna i egzogena jer se modeli nalaze na istom nivou apstrakcije, L1, samo su opisani različitim jezikom.

Model nivoa L1, odnosno šema baze podataka, transformiše se u šemu baze podataka, koja je u skladu sa generičkim meta-modelom relacione šeme baze podataka (druga kolona tabele 3.1) i nalazi se na nivou L2. Na nivou L2 se nalaze modeli, takođe zavisni od platforme (PSM), ali semantički bogatiji od modela na nivou L1 i sa manjim stepenom zavisnosti od platforme. Na osnovu ovoga, transformacija se može smatrati PSM-u-PSM, odnosno vertikalnom egzogenom transformacijom. Transformacija je na slici 3.2 označena pod rednim brojem 2 (PSM2PSM).

U sledećem koraku, dobijena šema nivoa L2 transformiše se u model tipova formi koji se nalazi na nivou L3. Model tipova formi je nezavisan od platforme (PIM) i predstavlja konceptualnu šemu baze podataka. Ova šema baze podataka je u skladu sa meta-modelom PIM koncepata alata IIS*Case (prva kolona tabele 3.1). Transformacija je vertikalna, egzogena, tipa PSM-u-PIM. Na slici 3.2 označena je pod rednim brojem 3 (PSM2PIM).

Na kraju, radi prelaska u XML tehnološki prostor, potrebno je ponovo dobijeni model na novou L3 transformisati u model koji je u skladu sa XML meta-modelom i serijlizovati ga u XML specifikaciju koja je u skladu sa odgovarajućom XML šemom (IISCase.XSD). Na slici 3.2, transformacija je označena rednim brojem 4 (PIM2XML).

Klasifikacije po nivoima apstrakcije ne odnose se na modele transformacija (*Transforamtion Model*) prikazane na slici 3.2. Modeli transformacija su opisani pomoću ATL jezika, odnosno u skladu su sa njegovim meta-modelom.

Relacija **u skladu sa** (*ConformsTo*), koja označava da je neki model u skladu sa odgovarajućim meta-modelom na slici 3.2 je predstavljena isprekidanom usmerenom linijom sa tačkama. Relacija **zasnovana na** (*Based on*), koja označava da je transfromacija zasnovana na nekom meta-modelu, na slici je označena isprekidanom usmerenom linijom. Legenda je prikazana u gornjem desnom uglu slike 3.2.

Detaljne specifikacije meta-modela biće opisane u šestom poglavlju, dok su specifikacije transformacija modela opisane u sedmom poglavlju.

3.4. Zaključak

U ovom poglavlju ukratko je prikazan proces reinženjeringa šema baza podataka u cilju reinženjeringa IS, s posebnom pažnjom posvećenom procesu reverznog inženjeringa. Postupak počinje sa modelom, nastalim ekstrakcijom iz postojeće baze podataka, koji je zavisan od platforme, i kroz niz model-u-model transformacija, transformiše seriju modela specifičnih za platformu sa sve većim stepenom nezavisnosti, tj. sa sve manjim i manjim specifičnostima vezanim za platformu. Od modela sa velikim stepenom zavisnosti od platforme se, nizom model-u-model transformacija dolazi do konceptualnog modela

zasnovanog na konceptu tipa forme. Ovaj model, nezavisan od platforme (PIM), zatim se uzima kao polazna tačka u procesu *forward* inženjeringa u kojem se opet sprovodi niz složenih transformacija koje rezultuju programskim kodom transakcionih programa, kao i SQL kodom za kreiranje baze podataka.

Postupak *forward* inženjeringa detaljno je opisan u narednom poglavlju.

4. Pregled karakteristika i mogućnosti okruženja IIS*Studio

Kao rezultat dugogodišnjeg istraživanja iz oblasti projektovanja i automatskog generisanja šema baza podataka i transakcionih programa nastao je metodološki pristup i razvojno okruženje koje ga podržava, IIS*Studio. IIS*Studio je namenjen projektovanju informacionih sistema i odgovarajuće šeme baze podataka, čiji je krajnji proizvod funkcionalni prototip informacionog sistema.

Poslednja verzija okruženja IIS*Studio organizovana je kroz dva glavna alata:

- IIS*Case - alat za projektovanje informacionih sistema i
- IIS*UIModeler - alat za modelovanje šablona korisničkog interfejsa aplikacija informacionih sistema.

Alat IIS*UIModeler namenjen je izradi šablona korisničkog interfejsa aplikacija za rad nad bazama podataka. Kako se istraživanja u okviru ove doktorske disertacije ne tiču ovog alata, to mu u daljem izlaganju neće biti posvećena pažnja. Više informacija o ovom alatu može se naći u [Banov10].

IIS*Case je alat zasnovan na MDSD pristupu, koji omogućava visok nivo automatizacije procesa *forward* inženjeringa informacionih sistema. Kako je *forward* inženjering deo čitavog procesa reinženjeringa, koji je predmet istraživanja ove doktorske disertacije, alat IIS*Case će biti detaljnije opisan u tesktu koji sledi.

Prvo će biti ukratko prikazane mogućnosti i osnovne funkcionalnosti alata IIS*Case. Zatim, biće opisani osnovni koncepti alata, gde će posebna pažnja biti posvećena definiciji koncepta tipa forme, kao glavnog koncepta za modelovanje. Nakon toga će biti ukratko opisane faze *forward* inženjeringa čiji je krajnji rezultat implementacioni opis relacione šeme baze podataka i programski kod ekranskih formi transakcionih programa.

4.1. Alat IIS*Case

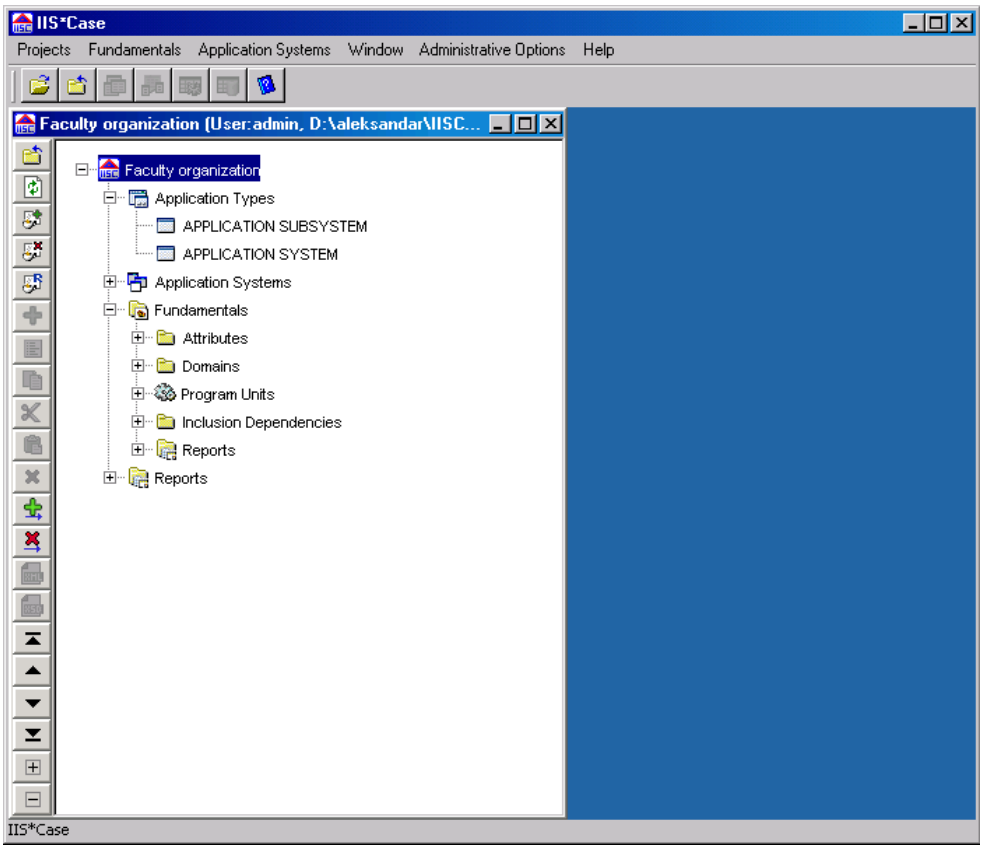
Osnovni koncepti i formalizmi na kojima je zasnovan metodloški pristup i alat IIS*Case nastajali su od kraja osamdesetih godina na Fakultetu tehničkih nauka u Novom Sadu. Izrađene su brojne verzije IIS*Case-a, u različitim programskim okruženjima. Prva verzija IIS*Case-a bila je zasnovana na tipovima formi i skupu funkcionalnih zavisnosti, kao polaznom skupu ograničenja. Tokom razvoja, ovaj skup se proširuje i nefunkcionalnim odnosima i tzv. specijalnim funkcionalnim zavisnostima [Mogin04, Pavić05 i Lukov09].

Inicijalno, IIS*Case je razvijen da podrži automatsko projektovanje šema baza podataka. Motiv razvoja ovakvog alata bila je pomoć projektantu u prevazilaženju problema definisanja ograničenja baza podataka. To se može postići automatskim generisanjem uz minimalnu interakciju sa projektantom. Zadatak projektanta je da dobro specificira pravila korišćenja podataka u realnom sistemu, a na alatu je da izvede zaključke o postojanju ograničenja baze podataka. IIS*Case je zasnovan na pretpostavci o postojanju šeme univerzalne relacije i

atributi buduće šeme baze podataka definišu se potpuno nezavisno od šeme relacije kojoj će pripadati. Daljim razvojem, skup koncepata ugrađenih u alat IIS*Case se širi, pa je na sadašnjem stepenu razvoja moguće modelovati većinu aspekata jednog informacionog sistema.

Poslednja verzija alata V.7.1. implementirana je u programskom jeziku *Java* i na slici 4.1 prikazana je glavna forma ovog alata. Na sadašnjem stepenu razvoja alat podržava:

- konceptualno projektovanje šeme baze podataka, transakcionih programa i informacionog sistema [Lukov07] i [Pavić06],
- automatizovano projektovanje podšema relacione baze podataka u trećoj normalnoj formi [Lukov93] , [Lukov06] i [Lukov03],
- automatizovanu integraciju podšema u jedinstvenu šemu baze podataka u trećoj normalnoj formi [Lukov93] i [Lukov06],
- automatizovanu detekciju formalnih kolizija ograničenja [Ristic07],
- generisanje SQL opisa šeme baze podataka za različite sisteme za upravljanje bazama podataka [Aleks05], [Aleks06], [Aleks07] i [Aleks07a],
- generisanje XML specifikacije informacionoog sistema [Pavić05],
- projektovanje aplikacija informacionog sistema i vizuelnih atributa ekranskih formi [Popov08] i [Popov08a],
- zadavanje *check* ograničenja, transformacija i generisanje SQL koda za proveru važenja tih ograničenja na nivou konceptualne šeme baze podataka [Obren12] i [Lukov10a] i
- generisanje programskog koda ekranskih formi transakcionih programa [Banov10].



Slika 4.1. Glavna forma alata IIS*Case

U pristupu na kojem se IIS*Case zasniva striktno je razdvojena specifikacija sistema i implementacija na konkretnoj platformi. Modelovanje se odvija na visokom nivou apstrakcije,

jer projektant putem koncepata iz problemskog domena kreira model informacionog sistema ne navodeći detalje vezane za implementacionu platformu. Počevši od ovog modela informacionog sistema koji je na viskom stepenu nezavisnosti od platforme (PIM), nizom transformacija modela, putem IIS*Case-a, može se dobiti više modela sa različitim stepenom zavisnosti od platforme (PSM). Model koji je potpuno specifičan za platformu je konkretni implementaciono orijentisan opis informacionog sistema ili implementacioni opis šeme baze podataka za izabranog proizvođača.

Postoji nekoliko alata koji su integrisani u alat IIS*Case. Neki od njih omogućavaju model-u-model transformacije, dok drugi omogućavaju model-u-kod transformacije. Na apstraktnom nivou PIM-a, IIS*Case obezbeđuje konceptualno modelovanje šeme baze podataka koje uključuje specifikaciju raznih vrsta ograničenja baze podataka, kao što su: ograničenje domena, ograničenje nula vrednosti, c*heck* ograničenje, ograničenje ključa i jedinstvenosti, kao i razne tipove ograničenja zavisnosti sadržavanja. Takav model se automatski transformiše u model relacione baze podataka, koji je u izvesnom stepenu platformski zavisan (zasnovan je na relacionom modelu), ali je nezavisan od konkretnog proizvođača. Dobijeni model se može transformisati model-u-kod transformacijom u SQL opis šeme baze podataka, koji je u skladu sa standardom i koji još uvek ima određeni stepen implementacione nezavisnosti ili u SQL opis šeme baze podataka za izabranog proizvođača koji predstavlja implementacioni opis šeme baze podataka sa stepenom 0% platformske nezavisnosti [Lukov10]. Ovakav pristup razvoju informacionih sistema saglasan je sa MDSD pristupom razvoja [Lukov08].

U daljem tekstu ovog poglavlja prezentovane su osnovne karakteristike i koncepti ovog alata. Detaljnije informacije o IIS*Case alatu i pristupu, kao i konceptima na kojima se on zasniva mogu se naći i u velikom broju radova, kao što su: [Lukov07], [Lukov93], [Lukov06], [Aleks07], [Aleks06], [Banov10], [Popov08], [Obren12], [Ristic07].

4.2. Platformski nezavisni koncepti u alatu IIS*Case

IIS*Case je zasnovan na konceptima koji su bliski krajnjem korisniku iz problemskog domena. Skup IIS*Case koncepata koji su tehnološki nezavisni mogu se podeliti u dva skupa:

- osnovni koncepti koji uključuju: domen, atribute, funkcije, pakete i događaje; i
- koncepte specifične za aplikaciju koji uključuju: projekat, aplikativni sistem, poslovnu aplikaciju, tip forme i tip komponente.

Osnovni koncepti se definišu na nivou projekta, a nezavisno od aplikativnih sistema. Njih čine:

- atributi (obeležja),
- domeni (primitivni i korisnički definisani),
- programske jedinice (paketi, funkcije i događaji) i
- zavisnosti sadržavanja, definisane na nivou univerzalnog skupa obeležja.

Detaljna specifikacija koncepta programskih jedinica iz klase osnovnih koncepata neophodnih za izražavanje kompleksnih aplikativnih funkcionalnosti data je u [Lukov10a] i [Mosti09].

Paket je kolekcija odabranih funkcija definisanih u IIS*Case repozitorijumu. Koncept funkcije koristi se za specifikaciju kompleksnih funkcionalnosti. Funkcije u IIS*Case-u definisane su na nivou projekta i mogu biti referencirane iz različitih IIS*Case specifikacija. Koncept događaja koristi se na nivou platformski nezavisnog modela, da predstavi bilo koji softverski događaj koji može pokrenuti neku akciju pod određenim uslovima. Događaji se dele na:

- događaje baze podataka (trigeri i izuzeci),

- događaje aplikativnog servera (događaj tastature, događaj miša i izuzeci) i
- događaje klijenta (događaj tastature, događaj miša i izuzeci).

Svaki događaj trebalo bi da bude povezan sa softverskom specifikacijom na nivou platformski nezavisnog modela.

Pomoću zavisnosti sadržavanja projektant može zadati ograničenja univerzalne šeme relacije tipa: skup vrednosti niza atributa je podskup skupa vrednosti drugog niza atributa. Ovakva ograničenja će uzrokovati pojavu odgovarajućih međurelacionih ograničenja implementacione šeme. Preimenovani atributi su specijalni slučajevi zavisnosti sadržavanja i definisanje preimenovanja je analogno zadavanju odgovarajuće zavisnosti. Detaljna specifikacija zavisnosti sadržavanja data je u [Pavić05].

Primena IIS*Case-a zasnovana je na pretpostavci o postojanju univerzalne šeme relacije, pa se atributi, pri projektovanju, definišu nezavisno od šeme relacije kojoj će pripadati. Definisanje potrebnih domena i funkcija je aktivnost koju je neophodno obaviti pre same definicije atributa.

Za atribute definisane na nivou osnovnih koncepata projektant je u mogućnosti da definiše:

- pridruživanje domena atributu,
- da li atribut učestvuje u izgradnji buduće implementacione šeme baze podataka,
- da li je atribut izveden iz nekog drugog atributa,
- da li je atribut nastao preimenovanjem nekog drugog atributa i
- vizuelne osobine atributa.

Detaljna specifikacija osnovnih koncepata atributa i domena data je u poglavlju 5.

Sve projektantske specifikacije vezane za jedan informacioni sistem organizovane su u okviru projekta koji ima strukturu tipa stabla. Osnovni činilac svakog projekta je aplikativni sistem. Projekat može da sadrži više aplikativnih sistema. Aplikativni sistem može da referencira više drugih aplikativnih sistema, koji time postaju njegovi potomci, tj. podsistemi.

Sve specifikacije vezane za projekte čuvaju se u jedinstvenom repozitorijumu okruženja IIS*Studio. Detaljnije informacije o strukturi repozitorijuma mogu se naći u [Pavić05] i [Banov10]. Komunikacija sa repozitorijumom odvija se putem ODBC protokola, što omogućava konekciju na različite sisteme za upravljanje bazama podataka.

Glavni koncept za modelovanje, koji je platformski nezavisan, u IIS*Case-u je *tip forme*. Jedan aplikativni sistem može da sadrži proizvoljan broj tipova formi, a takođe može da referencira i tipove formi koje pripadaju nekom drugom aplikativnom sistemu. Koncept tipa forme je objašnjen u narednom odeljku.

4.3. Tip forme

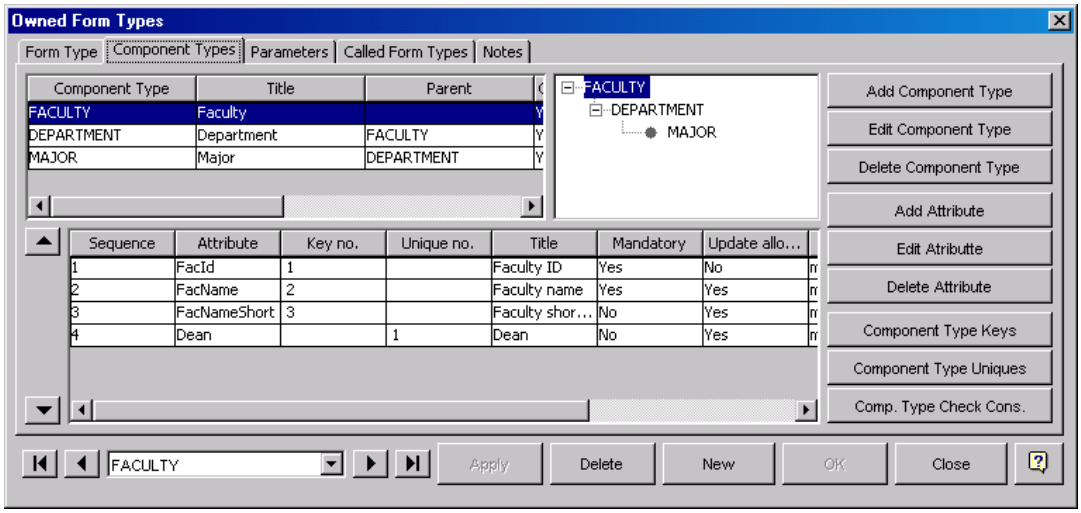
Tip forme je centralni koncept pristupa i alata IIS*Case. Tip forme predstavlja generalizaciju ekranskih formi koje korisnik upotrebljava u svrhu komunikacije s informacionim sistemom. Koristeći ovaj alat projektant specificira ekranske forme transakcionih programa, a indirektno, kreira inicijalni skup atributa i ograničenja.

Tradicionalni pristup podrazumeva projektovanje šeme baze podataka, a potom i izradu ekranskih formi transakcionih programa. Projektant koristeći IIS*Case, putem kojeg zadaje specifikacije vezane za tipove forme, istovremeno dizajnira šemu buduće baze podataka, kao i strukturu ekranskih formi koje će biti korišćene za unos, brisanje i izmenu podataka nad tom bazom. Takođe, projektant ne mora da poseduje napredna znanja, koja se tiču dizajna relacionih baza podataka. Projektant treba dobro da definiše pravila korišćenja podataka jednog realnog sistema, a alat je u stanju da takve specifikacije transformiše, između ostalog, u opis implementacione šeme baze podataka sa svim odgovarajućim ograničenjima.

Koncept tipa forme je dovoljno semantički bogat da omogući iskazivanje elemenata statičke strukture realnog sistema, kao i semantike, međuzavisnosti komponenata i pravila poslovanja. Projektovanje baze podataka u IIS*Case-u, putem koncepata tipa forme, daje kao rezultat formalan opis šeme baze podataka u trećoj normalnoj formi sa definisanim:

- skupovima svih ključeva šema relacija,
- ograničenjima nula (NULL) vrednosti,
- ograničenjima jedinstvene vrednosti (UNIQUE) i
- ograničenjima referencijalnog i inverznog referencijalnog integriteta.

Jedan tip forme može biti označen kao *menu* ili *program*. Ako je tip forme označen kao *menu*, onda će taj tip forme u budućem generisanju aplikacije predstavljati osnovu za generisanje menija ekranskih formi. Ako je označen kao program, onda će taj tip forme biti osnova za generisanje ekranskih formi transakcionih programa. Na slici 4.2 prikazana je forma putem koje se zadaje specifikacija tipa forme i njegove strukture. Za jedan tip forme koji je označen kao *program* može se specificirati da li učestvuje u dizajnu buduće baze podataka. Ako tip forme učestvuje u dizajnu, onda će on biti uključen u skup tipova formi koji se koristi pri generisanju implementacione šeme baze podataka.



Slika 4.2. Forma za definisanje tipa forme

Tip forme predstavlja imenovanu strukturu tipa stabla, čiji čvorovi se nazivaju tipovi komponenti. Svaki tip komponente ima jedinstveno ime u opsegu tipa forme koji ga sadrži, kao i neprazne skupove atributa i ključeva. Za određeni tip komponente je neophodno specificirati i skup operacija koje su dozvoljene nad tim tipom komponente. U okviru aplikativnog sistema projektant unosi nove, menja i briše postojeće tipove formi, kopira ih i premešta u druge aplikativne sisteme tekućeg projekta. Jedan aplikativni sistem može referencirati tipove formi koji pripadaju drugim aplikativnim sistemima. Ovakav tip forme će biti ravnopravno uključen u kreiranje polaznog skupa ograničenja.

Tipovima komponenti pridružuju se atributi iz univerzalnog skupa atributa. Zadavanje atributa na tipu komponente podrazumeva izbor atributa iz skupa svih definisanih atributa i zadavanje dozvoljenih operacija nad datim atributom. Jedan atribut sme da pripada najviše jednom tipu komponente istog tipa forme, a može biti uključen u različite tipove formi. Zadaje se i redosled atributa na tipu komponente, podrazumevana vrednost atributa i njegovo ponašanje na tipu komponente. Za svaki tip komponente projektant zadaje osnovne podatke kao što su:

- ime,
- nadređeni tip komponente,

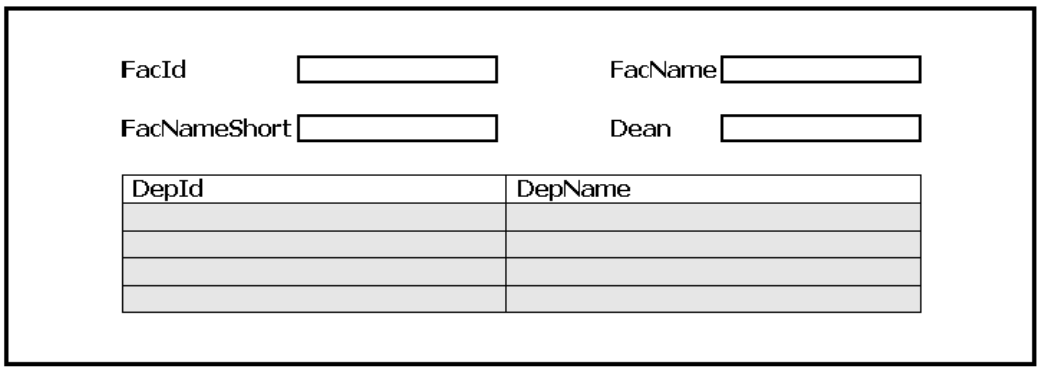
- dozvoljene operacije,
- kardinalnost u okviru nadređenog tipa komponente,
- atribute za vizualizaciju i
- grupe polja.

Jedan tip komponente je koreni, a svi ostali su podređeni tom tipu komponente u hijerarhijskoj strukturi tipa forme. Svaki tip komponente može posedovati proizvoljan broj podređenih tipova komponenti. Kardinalnost pojave tipa komponente u okviru pojave nadređenog tipa komponente, definiše se izborom jedne od opcija *0-N* i *1-N*. Ova činjenica će imati uticaja i na ograničenja u implementacionoj šemi baze podataka.

Za svaki tip komponente, mogu se definisati ograničenja ključa i ograničenja jedinstvene vrednosti. Ključ predstavlja neprazan skup atributa koji jedinstveno identifikuje pojavu tog tipa komponente u okviru jedne pojave nadređenog tipa komponente. Za jedan tip komponente moguće je definisati proizvoljan broj ključeva, a neki atribut može učestvovati u proizvoljnom broju ključeva. Atribut koji nije označen da učestvuje u kreiranju implementacione šeme baze podataka, ne može biti deo nekog ključa. Svaki tip komponente mora imati definisan makar jedan ključ, koji ga jedinstveno identifikuje u okviru jedne pojave njemu nadređenog tipa komponente. Za koreni tip komponente ključ definiše samo projektant. Za neki podređeni tip komponente, ključ predstavlja uniju po jednog ključa od korenog do datog tipa komponente.

Na sličan način, definišu se i ograničenja jedinstvene vrednosti. Jedno ograničenje jedinstvene vrednosti predstavlja neprazan skup atributa. Ne postavljaju se nikakva ograničenja vezana za broj ograničenja jedinstvene vrednosti koji će biti definisani na jednom tipu komponente.

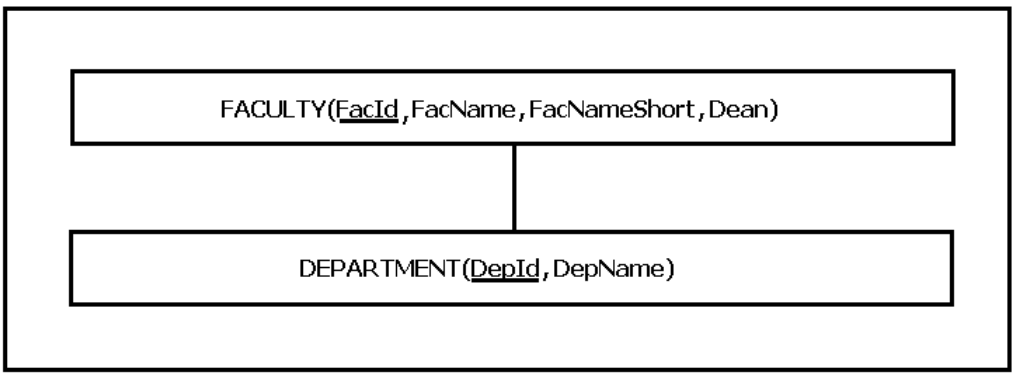
Primer 4.1. U ovom primeru prikazano je kako je moguće uz pomoć alata IIS*Case na osnovu dokumenata koji se koriste u realnom sistemu, doći do opisa tipova formi i tipova komponenti.



Slika 4.3. Dokument koji sadrži informacije o fakultetima

Na slici 4.3 prikazan je jedan dokument koji služi za evidenciju fakulteta i njegovih departmana. U gornjem delu dokumenta unose se osnovne informacije o fakultetu, dok je tabela u donjem delu namenjena za unos departmana koji pripadaju tom fakultetu.

Na osnovu ovog dokumenta, projektant može lako da definiše odgovarajući tip forme. Tip forme koji odgovara navedenom dokumentu sadrži dva tipa komponente *Faculty* i *Department*. Tip komponente *Faculty* je koreni tip komponente i nadređen je tipu komponente *Department*. Na slici 4.4 dat je grafički prikaz navedenih tipova komponenti.



Slika 4.4. Specifikacija tipa forme

Ključni atributi tipova komponenti su podvučeni. Atribut *FacId* jedinstveno identifikuje svaku pojavu korenog tipa komponente. Jednu pojavu tipa komponente *Department* jedinstveno identifikuje atribut *DeptId* u okviru jedne pojave tipa komponente *Faculty*.

4.4. Forward inženjering u alatu IIS*Case

U tradicionalnom pristupu projektovanja IS, projektovanje šeme baze podataka uobičajeno prethodi specifikaciji ekranskih formi i izveštaja transkacionih programa. Za razliku od tog pristupa, u IIS*Case-u projektant prvo specificira ekranske forme i indirektno kreira incijalni skup atributa i ograničenja. Skup tipova formi je platformski nezavisan pogled na IS iz perspektive krajnjeg korisnika. Zbog toga model IS-a kreiran pomoću IIS*Case-a, reprezentovan skupom specifikacija tipova formi, može biti klasifikovan kao PIM.

IIS*Case koristi skup tipova formi za generisanje relacionih šema baza podataka i njegov graf zatvaranja. Na ovaj način, kreiranjem tipova formi, projektant specificira:

- buduću šemu baze podataka,
- funkcionalne karakteristike budućeg transakcionog programa i
- izgled interfejsa za krajnjeg korisnika, sve u isto vreme.

Navedeni postupci počivaju na algoritmima koji su opisani u [Lukov93], [Lukov02] i [Lukov07]. Alat IIS*Case je dovoljno "inteligentan" da na osnovu specifikacija tipova formi sam izvede zaključke o ograničenjima, kandidatima za primarne ključeve, itd. i time uštedi značajno vreme projektantu.

U postupku *forward* inženjeringa putem IIS*Case-e projektovanje IS-a i generisanje prototipa aplikacije je iterativni proces koji se vrši u nekoliko sledećih osnovnih koraka:

1. konceptualno modelovanje,
2. generisanje šeme baze podataka,
3. konsolidacija šeme baze podataka,
4. integracija relacione šeme baze podataka,
5. generisanje implementacionog opisa šeme baze podataka i
6. automatsko generisanje transakcionih programa.

U narednim tačkama opisan je svaki od ovih koraka.

4.4.1. Konceptualno modelovanje

Konceptualno modelovanje vrši se kreiranjem skupova tipova formi, po jedan za svaki aplikativni sistem. Pre nego što započne postupak zadavanja specifikacija tipova formi projektant mora da kreira novi projekat koji sadrži bar jedan aplikativni sistem.

Sledeći korak u konceptualnom modelovanju IS-a je zadavanje osnovnih koncepata, koji su opisani u odeljku 4.2. Osnovni koncepti se definišu na nivou projekta, nezavisno od aplikativnog sistema.

U sledećem koraku, na osnovu dokumenata, projektant kreira tipove formi. Na njemu je da korektno izrazi pravila upotrebe podataka u realnom sistemu koristeći koncepte tipa forme. Tipovi formi predstavljaju osnovu na kojoj IIS*Case bazira algoritme za generisanje početnog skupa atributa i izdvajanje funkcionalnih zavisnosti kao ulaznih parametra postupka generisanja šeme baze podataka.

Na tipovima formi zadaju se tipovi komponenti i njihova hijerarhijska struktura tipa stabla kao što je prikazano na slici 4.2. Jedan tip komponente je u vrhu stabla, a svi ostali su hijerarhijski pod njim. Svaki tip komponente može imati proizvoljan broj podređenih tipova komponenti.

Zadavanje atributa na tipu komponente podrazumeva izbor atributa iz skupa osnovnih koncepata i zadavanje dozvoljenih akcija nad datim atributom. Jedan atribut može biti, u okviru tipova komponenti, uključen na različitim tipovima formi, ali ne sme pripadati različitim tipovima komponenti istog tipa forme.

Nakon definisanja atributa zadaju se ključevi i ograničenja jedinstvene vrednosti na tipu komponente.

Na osnovu specifikacija tipova formi u ovom koraku, moguće je identifikovati polazni skup ograničenja sledećih tipova:

- funkcionalne zavisnosti,
- nefunkcionalni odnosi,
- ugrađene zavisnosti sadržavanja i
- specijalne funkcionalne zavisnosti sa ugrađenim ograničenjima jedinstvene vrednosti.

Detaljna specifikacija ovih ograničenja data je u [Pavić05].

Nakon zadavanja tipova formi, sledeći korak *forward* inženjeringa je generisanje šeme baze podataka.

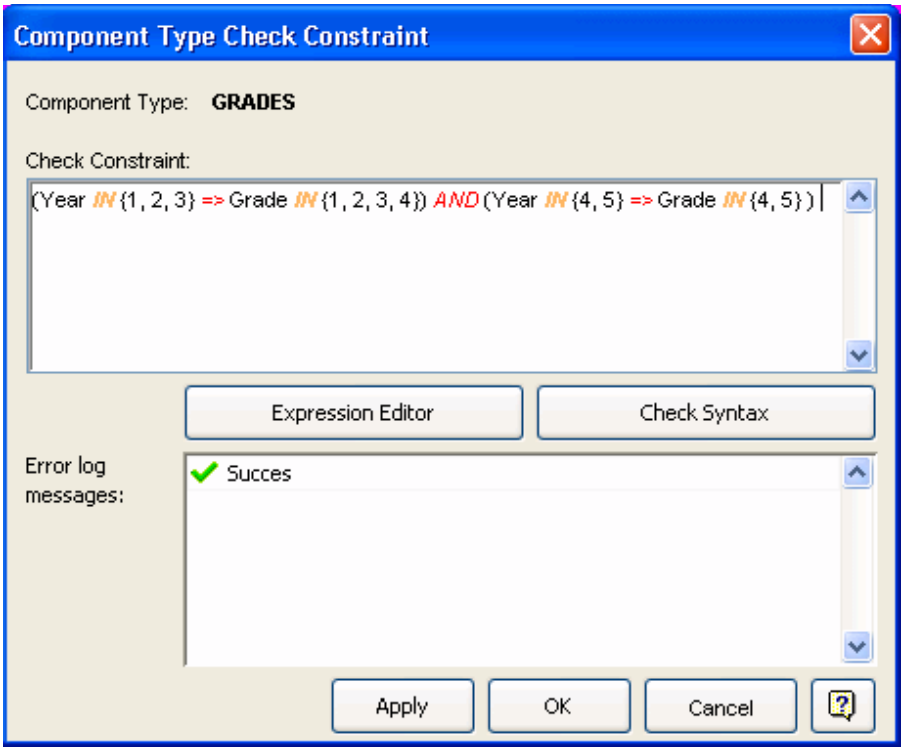
U alat IIS*Case takođe su ugrađeni koncepti i alati koji omogućavaju projektantu da definiše izraze ograničenja (*check* ograničenja) prilikom modelovanja konceptualne šeme baze podataka što, na primer, kod ER modelovanja nije slučaj. Za tu namenu uveden je jezik namenjen za domen putem kojeg je moguće definisati ova ograničenaja [Mosti09] i [Lukov10a]. Glavna svrha izraza ograničenja domena i atributa je da ograniči dozvoljene vrednosti pojedinačnih atributa. Suprotno ovome, izraz ograničenja tipa komponente koristi se da specificira logički uslov koji se odnosi na celu torku vrednosti, za svaku moguću instancu tipa komponente. Na slici 4.5 prikazana je forma za specifikaciju izraza ograničenja tipa komponente. Pored toga, razvijeni su i algoritmi čiji je zadatak da na osnovu specifikacije *check* ograničenja izvrše generisanje programskog koda. Više o ovim transformacijama može se naći u [Obren12].

Alat *Expession Editor* namenjen je za definisanje *check* ograničenja. Putem ovoga alata izrazi se mogu unositi na dva načina:

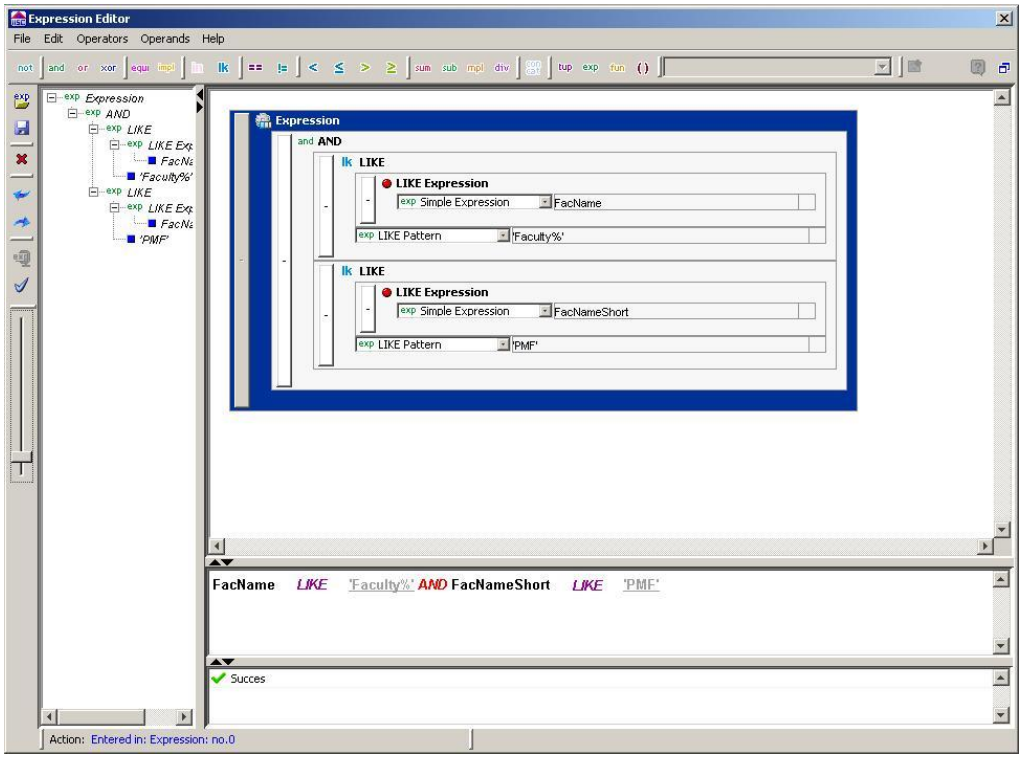
- u tekstualnoj formi i
- putem vizuelnog editora.

Pored sintaksne analize, alat vrši i deo semantičke analize. Na primer, za izraze zadate na nivou domena koji je tipa sloga, kada je referencirano polje sloga, proverava se da li je polje

validno. Alat ne podržava proveru tipova, jer model ne sadrži informaciju o tome da li je neki operator definisan za zadate domene. Na slici 4.6 prikazana je glavna forma ovog alata.



Slika 4.5. Forma za specificiranje izraza ograničenja

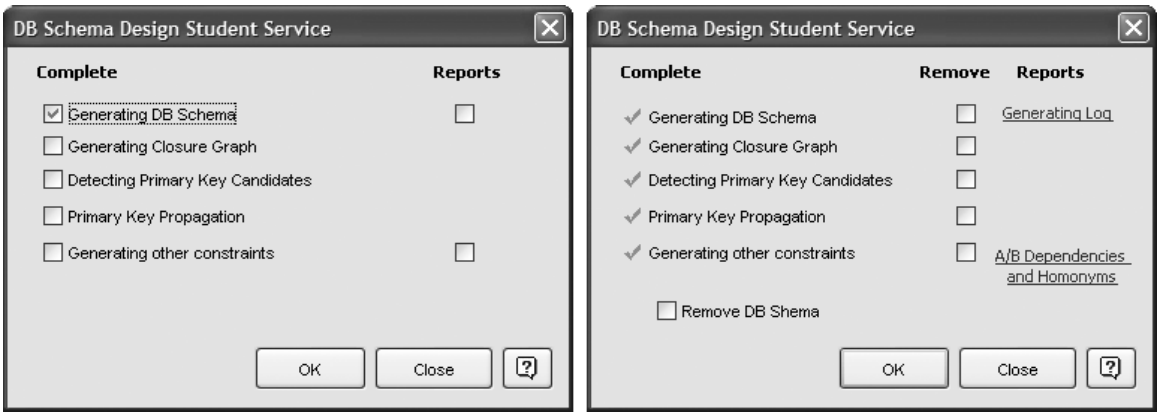


Slika 4.6 Alat Expession Editor

4.4.2. Generisanje šeme baze podataka

Proces generisanja šeme baze podataka aplikativnog sistema na osnovu polaznog skupa ograničenja obuhvata izvođenje sledećih 5 koraka:

1. generisanje skupa šema relacija pomoću algoritma sinteze,
2. generisanje grafa zatvaranja,
3. određivanje kandidata za primarne ključeve,
4. prostiranje primarnih ključeva u skupu šema relacija generisane šeme i
5. generisanje skupa ograničenja date šeme baze podataka.



Slika 4.7. Forma za kontrolu procesa generisanja šeme baze podataka

Svi koraci izvode se automatski, u određenom redosledu, na zahtev projektanta. Ulazna specifikacija je unija skupova tipova formi selektovanog aplikativnog sistema i svih njegovih aplikativnih podsistema. Na slici 4.7 prikazana je IIS*Case forma koju korisnik koristi za kontrolu procesa generisanje šeme baze podataka. Forma na levoj strani slike prikazuje slučaj kada nisu svi koraci kompletirani, dok ista forma, prikazana na slici sa desne trane, prikazuje slučaj kada je čitav postupak kompletiran.

U prvom koraku primenjuje se algoritam sinteze ([Beeri79] i [Diedr88]), značajno unapređen, specijalno za potrebe praktične primene u pristupu i alatu IIS*Case [Mogin04 i Lukov09]. Tokom ovog koraka, konceptualni model transformiše se u relacioni model podataka. Transformacija počinje izvođenjem skupa funkcionalnih, nefunkcionalnih i specijalnih funkcionalnih zavisnosti iz svih tipova formi uključenih u ulaznu specifikaciju, a kao rezultat dobija se skup šema relacija sa definisanim skupovima atributa, sintetizovanim ključevima i ograničenjima jedinstvene vrednosti. Algoritam sinteze predstavljen u [Beeri79] je proširen u IIS*Case-u novim koracima koji, između ostalog, omogućavaju generisanje i sintetiziovanih i nesintetizovanih ključeva [Mogin04].

Drugi korak generiše graf zatvaranja kao grafičku reprezentaciju generisane šeme, koji je posebno važan za vizuelizaciju međurelacionih ograničenja. Neformalno, svaki čvor grafa zatvaranja predstavlja šemu relacije, koja se generiše algoritmom sinteze i svaka direktna veza između čvorova reprezentuje činjenicu da je pravi ili nepravi podskup ključa podređenog čvora propagiran kao strani ključ u nadređeni čvor.

U trećem koraku iz skupa ekvivalentnih ključeva svake šeme relacije, automatski se identifikuju kandidati za primarni ključ.

U četvrtom koraku, projektant proglašava jedan od kandidata ključeva (u slučaju da ih ima više od jednog) za primarni ključ šeme relacije. IIS*Case propagira automatski taj ključ kao strani ključ u sve direktno podređene šeme ralacija. Propagacija takođe, automatski uklanja ključeve koji su ekvivalentni propagiranom iz šeme relacije u nadređenom čvoru.

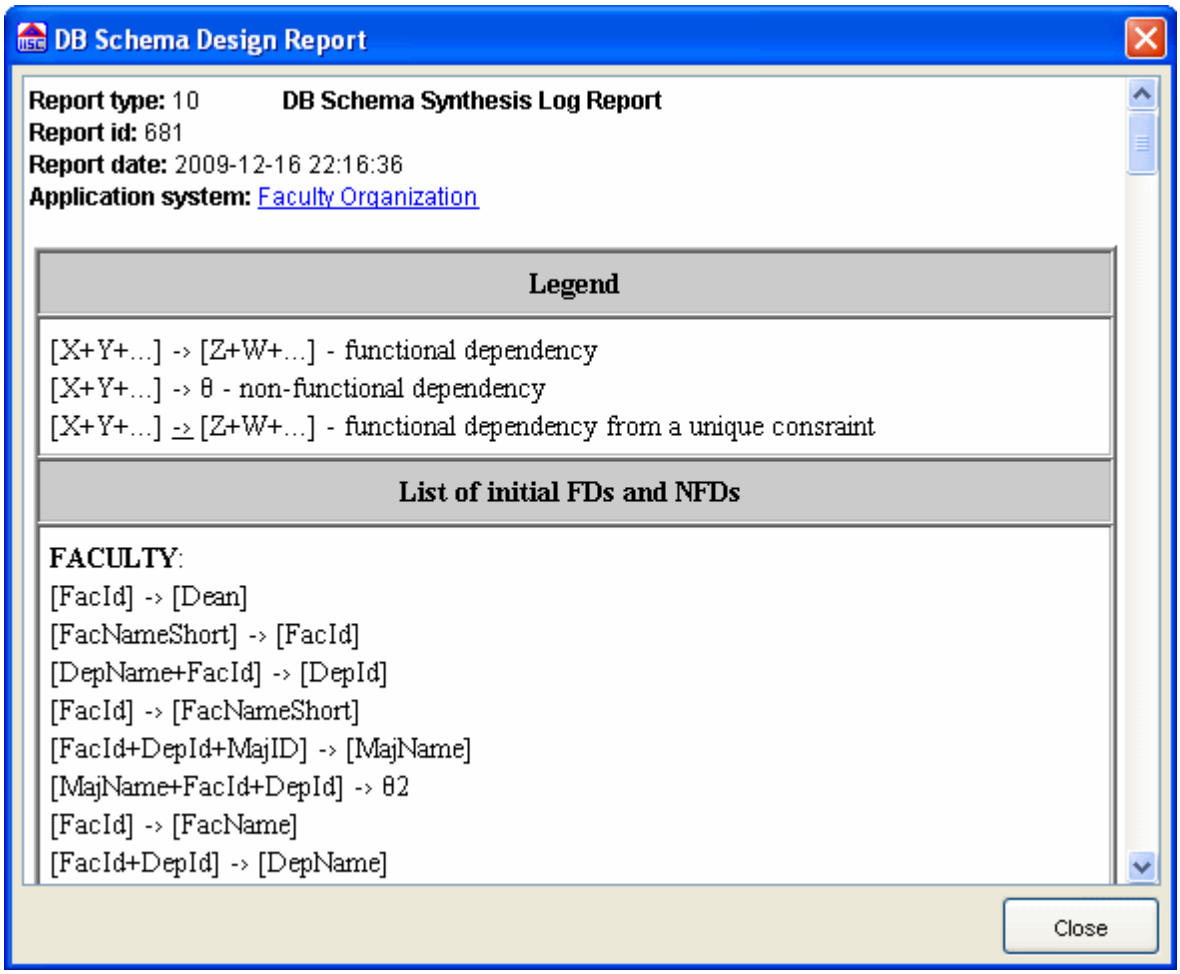
U petom koraku generišu se međurelaciona ograničenja sledećih tipova:

- osnovna ograničenja referencijalnih integriteta,
- proširena ograničenja referencijalnih integriteta,
- ograničenja referencijalnih integriteta, zasnovanih na netrivijalnim zavisnostima sadržavanja i
- ograničenja inverznih referencijalnih integriteta, osnovnih i onih zasnovanih na netrivijalnim zavisnostima sadržavanja.

Osnovna i proširena ograničenja referencijalnog integriteta izvode se na osnovu grafa zatvaranja. Njihovo postojanje je posledica propagacije primarnog ključa. Ako se ceo primarni ključ prostire iz podređene (referencirane) šeme relacije u nadređenu (referencirajuću), IIS*Case generiše ograničenje osnovnog referencijalnog integirteta. U suprotnom se detektuje kao ograničenje proširenog referencijalnog integirteta.

Ograničenja referencijalnog integriteta zasnovana na netrivijalnim zavisnostima sadržavanja nastaju od netrivijalnih zavisnosti sadržavanja koje projektant može da definiše na nivou skupa svih atributa informacionog sistema. Ograničenje inverznog referencijalnog integriteta nastaje od komponenti tipova forme. Preciznije, ako je tip komponente N_i direktno podređen tipu komponente N_j, tada projektant može da specificira da svaka instanca N_i mora biti povezana sa najmanjem jednom instancom od N_j. U tom slučaju IIS*Case zaključuje da se radi o ograničenju inverznog referencijalnog integriteta.

IIS*Case takođe detektuje homonime i A i B-zavisnosti šema relacija u istom koraku. Ovi termini su diskutovani detaljnije u [Honey83] i [Mogin00].



Slika 4.8. Izveštaj o rezultatima i međurezultatima procesa sinteze

IIS*Case nudi i automatsko generisanje odgovarajućih izveštaja nakon prvog i petog koraka procesa generisanja. Na slici 4.8 prikazan je izveštaj koji se generiše tokom sinteze (prvog koraka), a koji uključuje informacije o svim transformacijama koje su urađene nad polaznim skupom ograničenja.

Prethodnim koracima obezbeđeno je generisanje tzv. potencijalne šema baze podataka selektovanog aplikativnog sistema. Ako je aplikativni sistem istovremeno i podsistem nekog drugog aplikativnog sistema, tada generisana šema baze podataka postaje podšema.

Hijerarhijska struktura u okviru koje jedan aplikativni sistem referencira druge aplikativne sisteme kao svoje podsisteme, implicira da u implementacionoj šemi aplikativnog sistema budu ugrađene i podšeme svih njegovih aplikativnih podsistema. To ne znači da je šema baze podataka aplikativnog sistema rezultat proste unije podšema njegovih podsistema. Podšeme će biti integrisane jednim složenim procesom, jer bi prosto uniranje moglo izazvati kolizije i redudantnost podataka. Skup tipova formi jednog aplikativnog sistema, odnosno konceptualna šema baze podataka datog sistema, tokom faza generisanja i konsolidacije, biće transformisana u implementacionu relacionu šemu baze podataka.

4.4.3. Konsolidacija (usaglašavanje) šema baze podataka

Svaka podšema koja odgovara podsistemu aplikativnog sistema mora biti formalno konzistentna sa šemom baze podataka celog sistema, tj. mora obezbediti usaglašenost svih relacionih i međurelacionih ograničenja. Ovo znači da ukoliko se podšeme integrišu u neku šemu baze podataka, neophodno je nakon integracije proveriti konzistentnost na višem nivou. Da bi se moglo smatrati da su u implementacionoj šemi pravilno integrisane sve njene podšeme, mora biti zadovoljen uslov usaglašenosti koncepata podšeme sa konceptima potencijalne implementacione šeme. To znači da mora važiti sledeće:

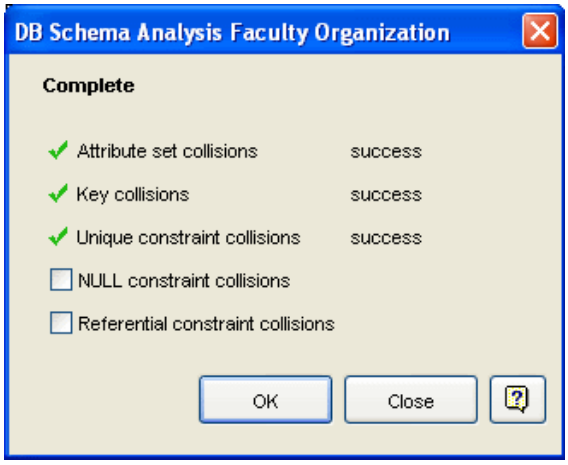
- za svaku šemu relacije podšeme, postoji šema relacije integrisane šeme, takva da je skup obeležja šeme relacije podšeme podskup skupa obeležja šeme relacije implementacione šeme i primarni ključ šeme relacije podšeme je jednak primarnom ključu korespondentne šeme relacije integrisane šeme;
- sva relevantna ograničenja iz skupa ograničenja integrisane šeme su ugrađena u skup ograničenja podšeme.

Nakon bilo kog koraka generisanja šeme buduće baze podataka aplikativnog sistema koji referencira aplikativni podsistem ili tip forme iz drugog sistema, projektant je u mogućnosti da proveri usaglašenost integrisane šeme sa podšemama da bi se utvrdilo da li su uslovi usaglašenosti zadovoljeni, tj. da li ona može da predstavlja implementacionu šemu baze podataka. Algoritam za konsolidaciju kreiran je na osnovu postupaka i algoritama predloženih u [Ristic03]. U [Lukov03], [Ristic06] i [Ristic03a] detaljno je opisno kako je dati algoritam implementiran u IIS*Case-u.

Proces konsolidacije šeme baze podataka sa podšemama podeljen je na segmente koji usaglašavaju pojedinačne grupe ograničenja istog tipa. Proces se sastoji od sledećih koraka:

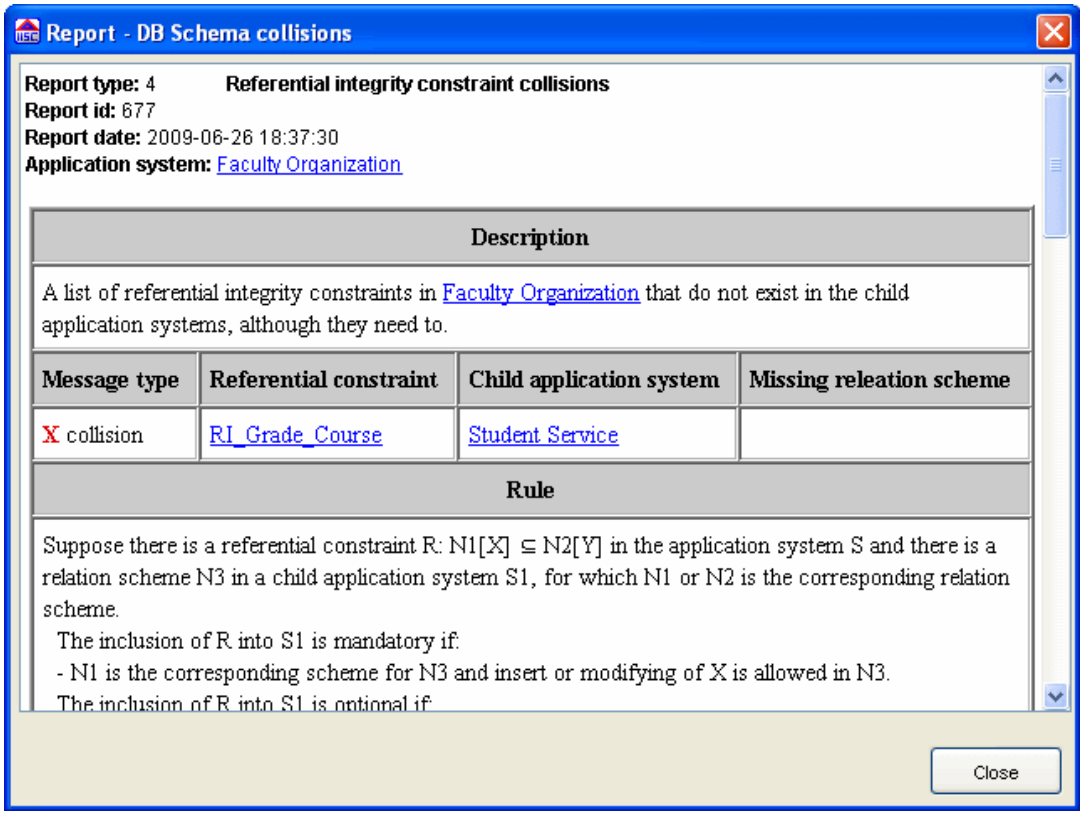
1. usaglašenost skupova atributa podšema i potencijalne implementacione šeme baze podataka,
2. usaglašenost skupova ograničenja jedinstvene vrednosti atributa,
3. usaglašenost skupova ograničenja ključa,
4. usaglašenost skupova ograničenja NULL vrednosti atributa i
5. usaglašenost skupova ograničenja referencijalnih integriteta.

Redosled izvršavanja ovih koraka je bitan, jer uspešnost provere jedne grupe ograničenja može usloviti mogućnost provere druge grupe. Zato korake nije moguće preskakati ili izvršavati u proizvoljnom redosledu. Na slici 4.9 prikazana je ekranska forma preko koje projektant zadaje naredbe za izvršavanje pojedinačnih koraka algoritma konsolidacije.



Slika 4.9. Forma za analizu usaglašenosti šeme baze podataka

I ovaj proces je detaljno dokumentovan kroz generisanje izveštaja u kojima se detaljno opisuju neusaglašenosti podšema, koje se odnose na skup ograničenja koji se u datom koraku usaglašava. Svi generisani izveštaji čuvaju se, tako da projektant može da pristupi istoriji promena. Direktan pristup izveštaju moguć je kroz podstablo stabla projekta koje odgovara datom aplikativnom sistemu. Pretraga po tipu i sadržaju izveštaja olakšava njihovo pregledanje, u slučaju velikog broja izgenerisanih izveštaja. Na slici 4.10 dat je jedan primer izveštaja neusaglašenosti skupova ograničenja referencijalnih integriteta.



Slika 4.10. Izveštaj o kolizijama ograničenja referencijalnog integriteta

Proces projektovanja šeme baze podataka je iterativan. Svaka iteracija podrazumeva kreiranje novog ili promenu postojećeg skupa polaznih ograničenja kroz specifikaciju tipova formi, generisanje šeme baze podataka i primenu algoritma konsolidacije. Ovaj proces

ponavlja se u onoliko iteracija koliko je potrebno, dok se ne dobije usaglašena šema baze podataka.

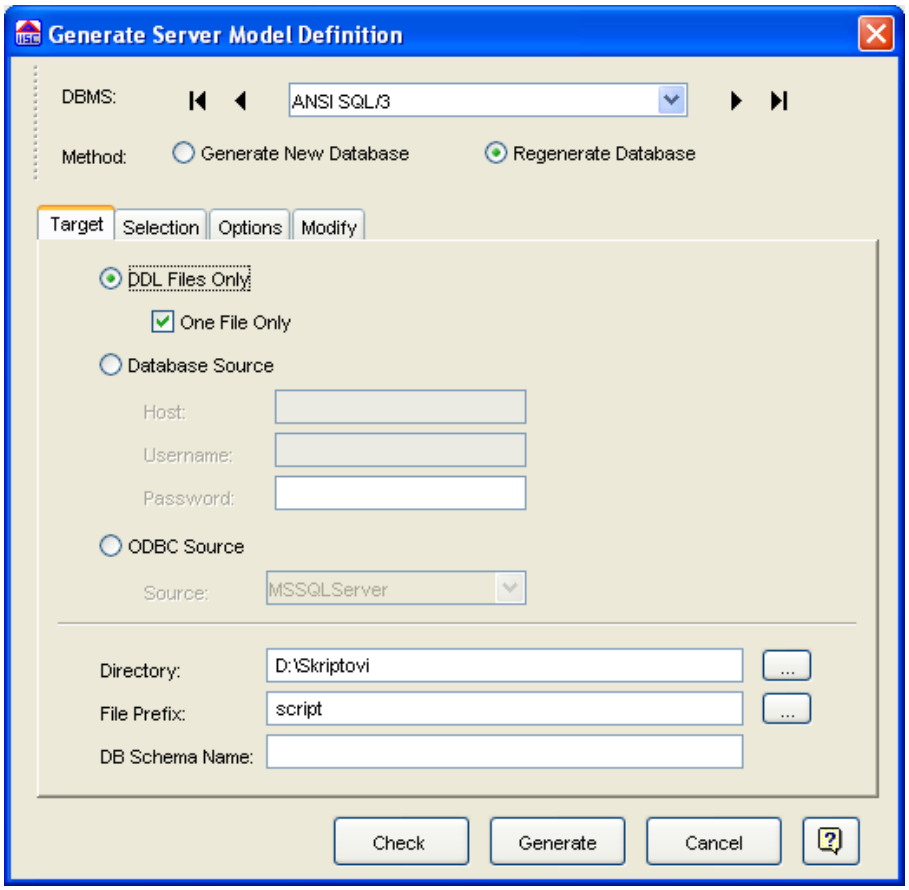
4.4.4. Generisanje implementacionog opisa šeme baze podataka

Sledeći korak u postupku *forward* inženjeringa je generisanje implementacionog opisa šeme baze podataka. U alat IIS*Case je ugrađen SQL generator (slika 4.11) putem kojeg je moguće implementirati internu šemu baze podataka u okviru izabranog sistema za upravljanje bazama podataka podržanog od strane alata. SQL generator koristi specifikaciju nastalu projektovanjem i obezbeđuje automatsko generisanje SQL koda. Ovaj alat povezuje precizne notacije za izražavanje uslova integriteta sa mehanizmima putem kojih relacioni sistemi za upravljanje bazama podataka mogu da održavaju konzistentnost baze podataka.

Projektantu je omogućeno da i bez poznavanja sintakse SQL-a kao i mehanizama za implementciju ograničenja, dobije SQL kod za kreiranje tabela, pogleda, indeksa, procedura, funkcija i okidača.

U ovoj realizaciji, SQL generator obezbeđuje generisanje implemetacionog opisa šeme baze podataka po standardu ANSI SQL [SQL:2003, SQL:2011], kao i za konkretne SUBP MS SQL Server i Oracle, po sintaksi jezika *Microsoft* T-SQL i Oracle PL/SQL.

Na ovaj način zaokružuje se proces projektovanja šeme baze podataka. Principi rada SQL generatora opisani su u [Aleks06].



Slika 4.11. Forma SQL generatora

Primer 4.2. Na osnovu tipa forme prikazanog na slici 4.4 biće generisane dve šeme relacija. Tip komponente *Faculty* je osnova za generisanje istoimene šeme relacije koja sadrži atribute *FacId*, *FacName*, *FacShortName* i *Dean*. Primarni ključ je skup koji sadrži samo atribut *FacId*. Slično, tip komponente *Department* je osnova za generisanje istoimene šeme relacije koja sadrži atribute *FacId*, *DepId* i *DepName*. Primarni ključ ove šeme je dvočlani skup koji sadrži atribute *FacId* i *DepId*. Ove specifikacije moguće je transformisati u implementacioni opis šeme baze podataka prikazan na listingu 4.1.

```
DROP TABLE Faculty CASCADE

DROP TABLE Department CASCADE

CREATE DOMAIN String AS VARCHAR(25);

CREATE DOMAIN NTekst AS LONGTEXT;

CREATE DOMAIN Broj AS DECIMAL;

CREATE DOMAIN DecimalniBroj AS DECIMAL;

CREATE DOMAIN Tekst AS VARCHAR(250);

CREATE DOMAIN LogVrijednost AS BOOLEAN;

CREATE DOMAIN Datum AS DATE;

CREATE DOMAIN Vrijeme AS TIME;

CREATE TABLE Faculty (
FacName Tekst NOT NULL,
Dean Tekst,
FacId Broj NOT NULL,
FacNameShort Tekst NOT NULL
)
CREATE TABLE Department (
FacId Broj NOT NULL,
DepId Broj NOT NULL,
DepName Tekst NOT NULL
)
ALTER TABLE Faculty ADD
   CONSTRAINT PK_Faculty PRIMARY KEY
   (   FacNameShort  ),   CONSTRAINT AK_Faculty_0 UNIQUE
   (   FacId  ),     CONSTRAINT AK_Faculty_1 UNIQUE
   (   FacName   ),   CONSTRAINT AK_Faculty_2 UNIQUE
   (   Dean    )

ALTER TABLE Department ADD
     CONSTRAINT AK_Department_0 UNIQUE
   (   FacId,DepId   ),    CONSTRAINT AK_Department_1 UNIQUE
   (   DepName,FacId  )
```

Listing 4.1. Primer generisanog SQL koda

Prethodno opisani koraci *forward* inženjeringa moraju se izvršavati po navedenom redosledu, a projektant posle svakog koraka ima uvid u generisane relacione šeme i ograničenja. Putem odgovarajućih formi, projektant može proveriti korektnost svojih specifikacija, a u određenom delu i dopuniti generisane specifikacije. Posle svakog koraka,

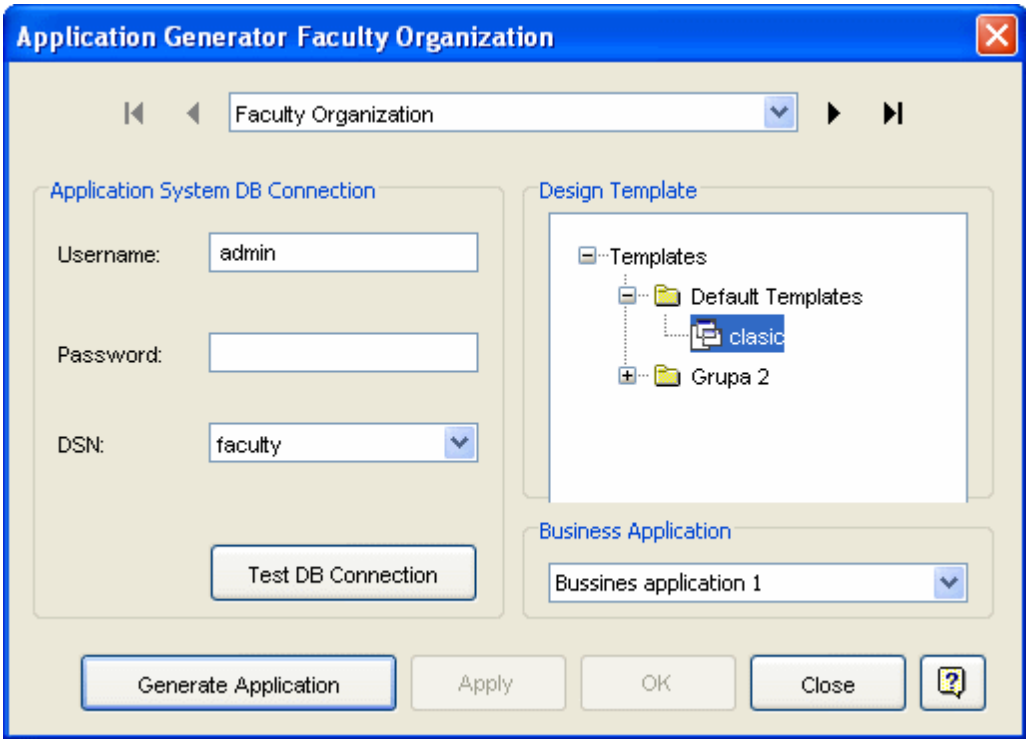
ukoliko projektant nije zadovoljan dobijenim rezultatima, moguće je vratiti se korak unazad, promeniti specifikacije vezane za tipove formi i ponoviti postupak.

Takođe, projektantu su dostupne informacije o atributima, primarnim ključevima, ograničenjima jedinstvene vrednosti kao i drugim relevantnim ograničenjima. Moguće je promeniti redosed atributa, vršiti modifikacije nad ograničenjima primarnog ključa i ograničenjima jedinstvene vrednosti.

4.4.5. Generisanje prototipa aplikacije

Nakon završenog generisanja i konsolidacije šeme baze podataka, kao i njene implementacije u okviru konkretnog sistema za upravljanje bazama podataka projektant može generisati prototip aplikacije. Na slici 4.12 prikazana je slika ekranske forme putem koje projektant zadaje parametre koji su neophodni da bi otpočeo proces generisanja. Neophodno je da projektant zada sledeće:

- pristupne parametre za komunikaciju sa bazom podataka,
- izabrani šablon korisničkog interfejsa i
- izabranu poslovnu aplikaciju.



Slika 4.12. Generator aplikacija

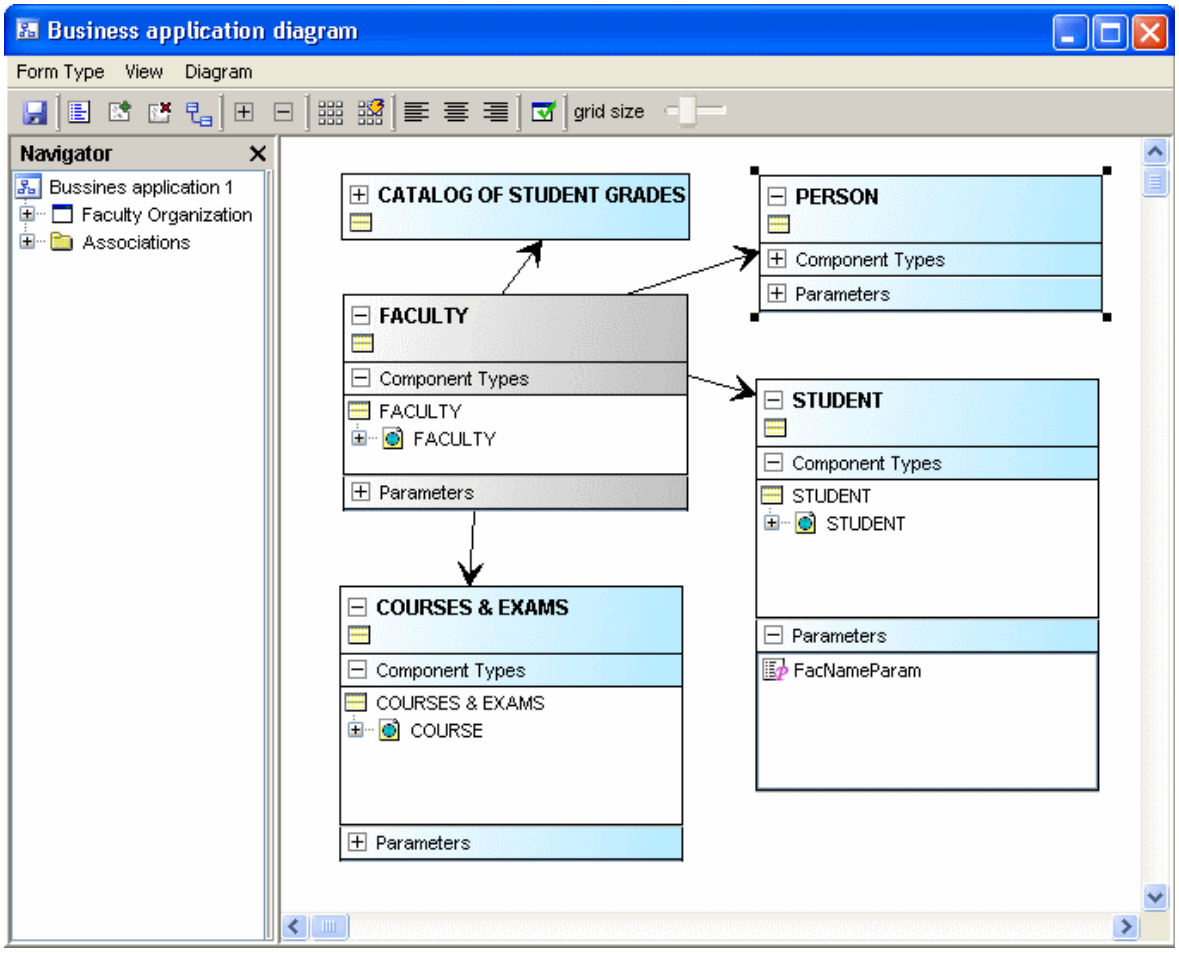
Specifikacija poslovne aplikacije

Koncept poslovne aplikacije uveden je i detaljno opisan u [Popov06]. Specificiranjem poslovnih aplikacija, projektant formalno opisuje funkcionalnosti informacionog sistema, koje se odnose na uzajamne pozive generisanih ekranskih formi koje odgovaraju različitim tipovima formi. Ovaj koncept definiše se na nivou aplikativnog sistema. Pored skupa tipova formi koji učestvuje u poslovnoj aplikaciji, specificira se i kako generisane ekranske forme međusobno komuniciraju. Zato se, na nivou poslovne aplikacije, definiše koncept koji se naziva pozivajuća struktura. Jedna pozivajuća struktura predstavlja uređeni par nad skupom

tipova formi koji učestvuju u poslovnoj aplikaciji. Osim ove informacije, konceptu pozivajuće strukture pridružene su sledeće osobine:

- prosleđene vrednosti,
- način poziva,
- metod poziva i
- UI pozicioniranje.

Na slici 4.13 dat je primer jedne poslovne aplikacije prikazan u okviru modula za grafičko modelovanje poslovnih aplikacija, koji se naziva *Business Aplication Designer*.

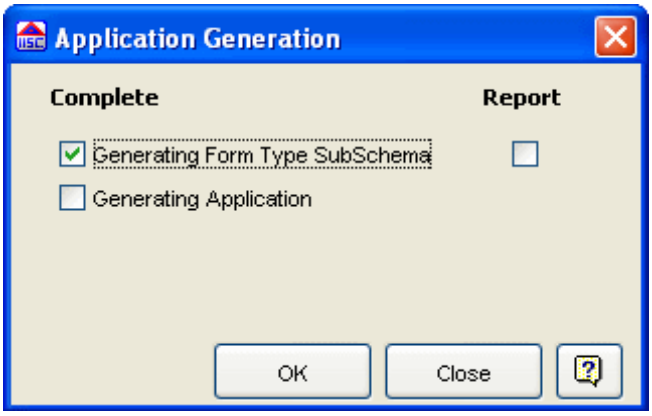


Slika 4.13. *Business Aplication Designer*

Generisanje transakcionog programa

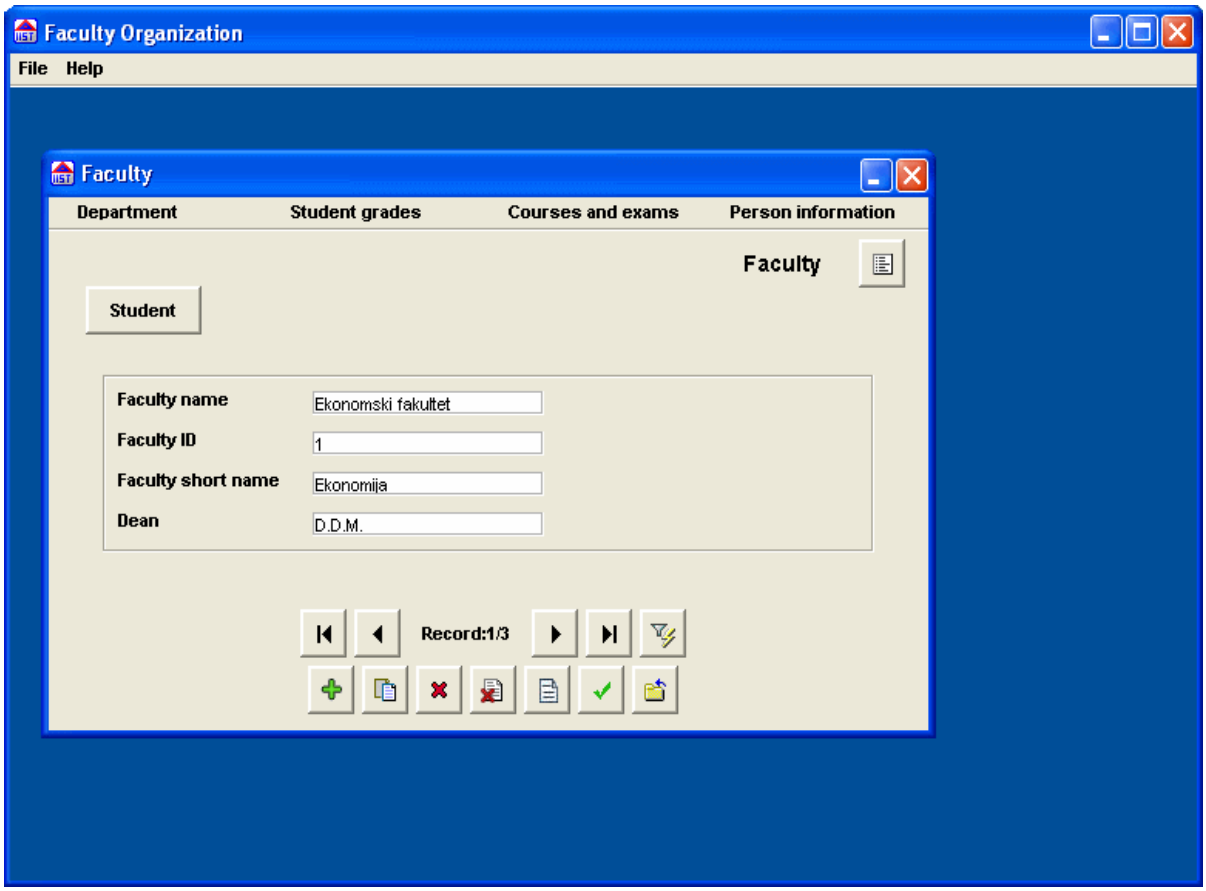
Proces generisanja transakcionog programa sastoji se iz dva koraka (kao što je prikazano na slici 4.14):

- generisanje podšema tipova formi, i
- generisanje aplikacije.



Slika 4.14. Generisanje prototipa aplikacije

Na slici 4.15 prikazan je primer automatski generisane aplikacije.



Slika 4.15. Primer prototipa aplikacije

U ovom poglavlju ukratko je opisan proces *forward* inženjeringa koji omogućava pristup i alat IIS*Case. Jedan od ciljeva istraživanja u ovoj doktorskoj disertaciji je da se okruženje IIS*Studio proširi novim alatom koji će se oslanjati i na mogućnosti alata IIS*Case i njegov proces *forward* inženjeringa. Istraživanjem je predviđeno da se u takav alat ugrade algoritmi i metode koje će omogućiti reverzni inženjering šeme baze podataka. Time bi bio podržan kompletan proces reinženjeringa IS. U narednom poglavlju opisan je čitav proces reinženjeringa i predložen alat IIS*Ree u kojem se taj proces odvija.

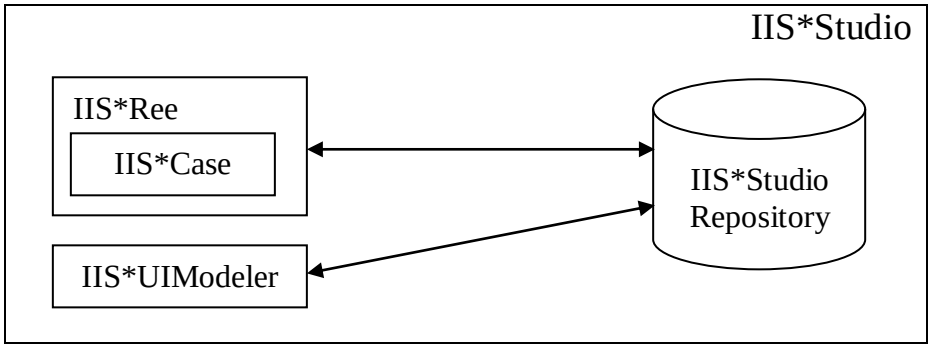
5. IIS*Ree: pristup i softversko rešenje za MD reinženjering IS

U prethodnom poglavlju predstavljen je tradicionalni proces projektovanja informacionih sistema, odnosno *forward* inženjering, u kojem se nakon zadavanja projektnih specifikacija generiše implementaciona šema baze podataka. U ovom poglavlju biće predstavljen kompletan proces reinženjeringa koji uključuje reverzni inženjering i *forward* inženjering. Posebna pažnja će biti posvećena postupcima reverznog inženjeringa u kojima se pomoću model-u-model transformacija transformiše relaciona šema baze podataka, koja je tehnološki specifičan model, zavisan od relacione tehnologije, u tehnološki nezavisan model zasnovan na konceptima tipa forme. Nakon dobijenog konceptualnog tj. domenski orijentisanog modela projektant ga može restruktuirati, a zatim postupcima *forward* inženjeringa ponovo generisati implementacioni opis šeme baze podataka za izabranu platformu. U finalnom koraku, projektantu se pruža mogućnost generisanja transakcionih programa, odnosno izvršnih softverskih specifikacija aplikativnih sistema nad novokreiranom bazom podataka.

Kao rezultat istraživanja ove doktorske teze nastalo je softversko rešenje za polu-automatski MD reinženjering, nazvano IIS*Ree. Neke od glavnih osobina ovog softverskog rešenja su zasnovanost na korišćenju postojećih tehnologija u polju MDA, kao što su EMF i ATL jezik za transformacije modela.

Alat IIS*Ree, namenjen da obezbedi transformacije šema baza podataka u procesu reinženjeringa informacionih sistema, kao i alati IIS*Case i IIS*UIModeler, razvijen je u okviru okruženja IIS*Studio. Sva tri alata koriste isti repozitorijum, koji je proširen za potrebe IIS*Ree. Specifikacija dela repozitorijuma koja se odnosi na ovo proširenje data je u prilogu B. Ostatak rečnika podataka IIS*Studio okruženja može se naći u [Pavić05] i [Banov10]. Konceptualna arhitektura okruženja IIS*Studio prikazana je na slici 5.1.

U alat IIS*Ree ugrađeni su postupci koji obezbeđuju automatizovano prevođenje opisa baza podataka u izabrani ciljni model. Kao ulazni model, za sada je podržan relacioni model, kao veoma često korišćen model u praksi. Trenutno se pomoću IIS*Ree može vršiti reinženjering baza podataka implementiranih na sledećim SUBP: Oracle (verzije 9i, 10g i 11g) i MS SQL Server (verzije 2000, 2005, 2008 i 2012).



Slika 5.1. Okruženje IIS*Studio

U ovoj verziji, kao izlaz iz IIS*Ree alata može se dobiti sledeće:
- konceptualni model podataka opisan PIM konceptima IISCase-a,
- implementacioni opis šeme baze podataka za različite relacione sisteme za upravljanje bazama podataka i
- XML specifikacija informacionog sistema (XML specifikacija tipova formi i XML specifikacija šeme baze podataka).

Alat može da generiše implementacioni opis relacionih baza podataka za konkretne proizvođače Oracle (verzije 9i, 10g i 11g) i MS SQL Server (verzije 2000, 2005, 2008 i 2012), kao i ANSI SQL standard.

U procesu reinženjeringa projektant može raditi i integraciju više izvora od različitih proizvođača. Pretpostavka je da se relaciona šema baze podataka nalazi u trećoj normalnoj formi.

U nastavku teksta ovog poglavlja opisani su postupci i algoritmi koji se primenjuju u ekstrakciji modela iz postojeće baze podataka i transformaciji modela u cilju reinženjeringa informacionih sistema. Takođe, biće prikazana arhitektura alata IIS*Ree kao i njegovi praktični aspekti primene.

5.1. Arhitektura alata IIS*Ree

U skladu sa MDA, kompletan proces reinženjeringa može se posmatrati kao niz transformacija koje se izvršavaju na različitim modelima koji učestvuju u čitavom procesu. Ulazna specifikacija za proces reinženjeringa je fizička šema baze podataka, smeštena u rečniku podataka RSUBP-a. Ova šema, specifična za implementacionu platformu, transformiše se u logičku šemu baze podataka koja je u skladu sa delom meta-modela SQL standarda koji podržava većina RSUBP. Nastala logička šema baze podataka je zavisna od platforme (PSM), ali nezavisana od konkretnog proizvođača. Zatim se ova logička šema transformiše u šemu baze podataka koja je u skladu sa generičkim meta-modelom relacione šeme baze podataka. Ona je, takođe, zavisna od platforme (PSM), ali semantički bogatija od prethodne. U sledećem koraku, dobijena šema baze podataka transformiše se u model tipova formi, koji predstavlja konceptualnu šemu baze podataka. Ovaj model, nezavisan od platforme (PIM), uzima se, zatim, kao polazna tačka u procesu *forward* inženjeringa u kojem se opet sprovodi niz složenih transformacija koje rezultuju programskim kodom transakcionih programa i *SQL* kodom za kreiranje opisa baze podataka.

Arhitektura IIS*Ree sistema, koji je namenjen da podrži prethodno opisan proces, prikazana je na slici 5.2. IIS*Ree se sastoji iz četiri glavna modula i jednog podsistema. U postupku reverznog inženjeringa koriste se sledeći moduli, koji su originalno nastali kao rezultat istraživanja u okviru ove doktorske disertacije:
- *Legacy System Interface*,
- *MetaData Validator & Constraints Discoverer*,
- *XML Transformer* i
- *M2M Transformer*.

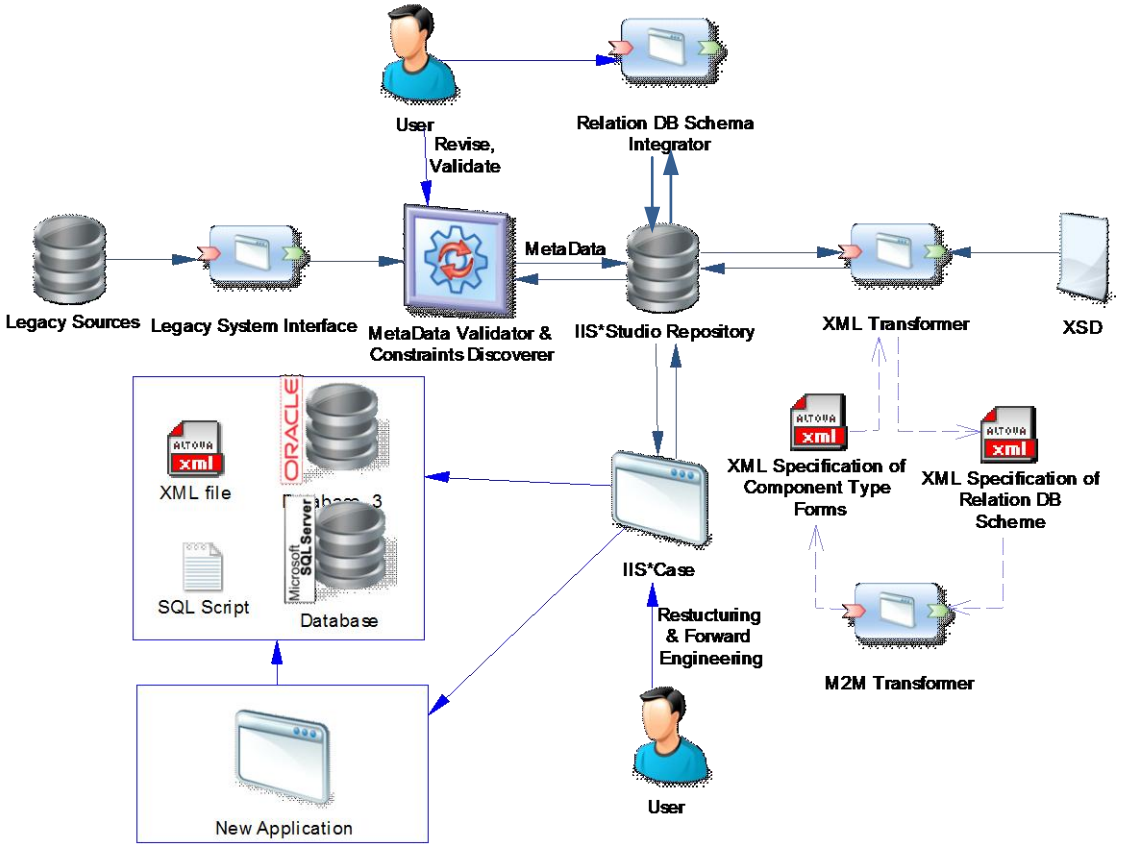
U cilju kompletne podrške postupka *forward* inženjeringa, IIS*Ree uključuje, kao podsistem, alat IIS*Case zajedno sa modulom *Relation DB Schema Integrator*.

Legacy System Interface je modul u kojem je implementiran skup algoritama za ekstrahovanje svih mogućih informacija iz date nasleđene relacione baze podataka. Čita rečnik podataka postojeće baze podataka kao i same podatke, čime se obezbeđuje unapređenje semantike modela. Na osnovu nekih meta-podataka pročitanih iz rečnika podataka i samih podataka nastaje model, koji predstavlja reprezentaciju logičke šeme baze podataka, koji je u skladu sa RSUBP meta-modelom. Model se smešta u repozitorijum okruženja IIS*Studio.

MetaData Validator & Constraints Discoverer je modul koji uz pomoć korisnika-projektanta razrešava kolizije imena atributa i pronalazi kandidate za postojanje ograničenja inverznog referencijalnog integriteta.

Pošto je XML prihvaćen kao standardni format za smeštanje podataka i upravljanje podacima, ekstrahovani modeli se serijalizuju kao XML specifikacije. Ovo radi *XML Transformer* modul. On uzima podatke iz repozitorijuma okruženja IIS*Studio, koji predstavljaju formalni opis logičke šeme baze podataka, nastale kao rezultat rada prethodna dva modula, i primenjuje algoritam za serijalizaciju podataka u XML specifikaciju na osnovu predefinisane XML šeme ugrađene u alat. *XML Transformer*, takođe, ima ulogu da XML specifikaciju, nastalu kao rezultat procesa transformacije u okviru modula *M2M Transformer*, parsira na osnovu druge predefinisane XML šeme. Rezultat parsiranja smešta u repozitorijum okruženja IIS*Studio.

M2M Transformer je modul koji preuzima XML specifikaciju, nad kojom primenjuje niz algoritama za transformacije koje generišu opise šema baza podataka na različitim nivoima apstrakcije. Kao krajnji rezultat ovog modula dobija se konceptulani model tj. instanca meta-modela koji je zasnovan na PIM konceptima alata IIS*Case.



Slika 5.2. Arhitektura IIS*Ree alata

IIS*Case i *Relation DB Schema Integrator* već su prikazani u poglavlju 4.

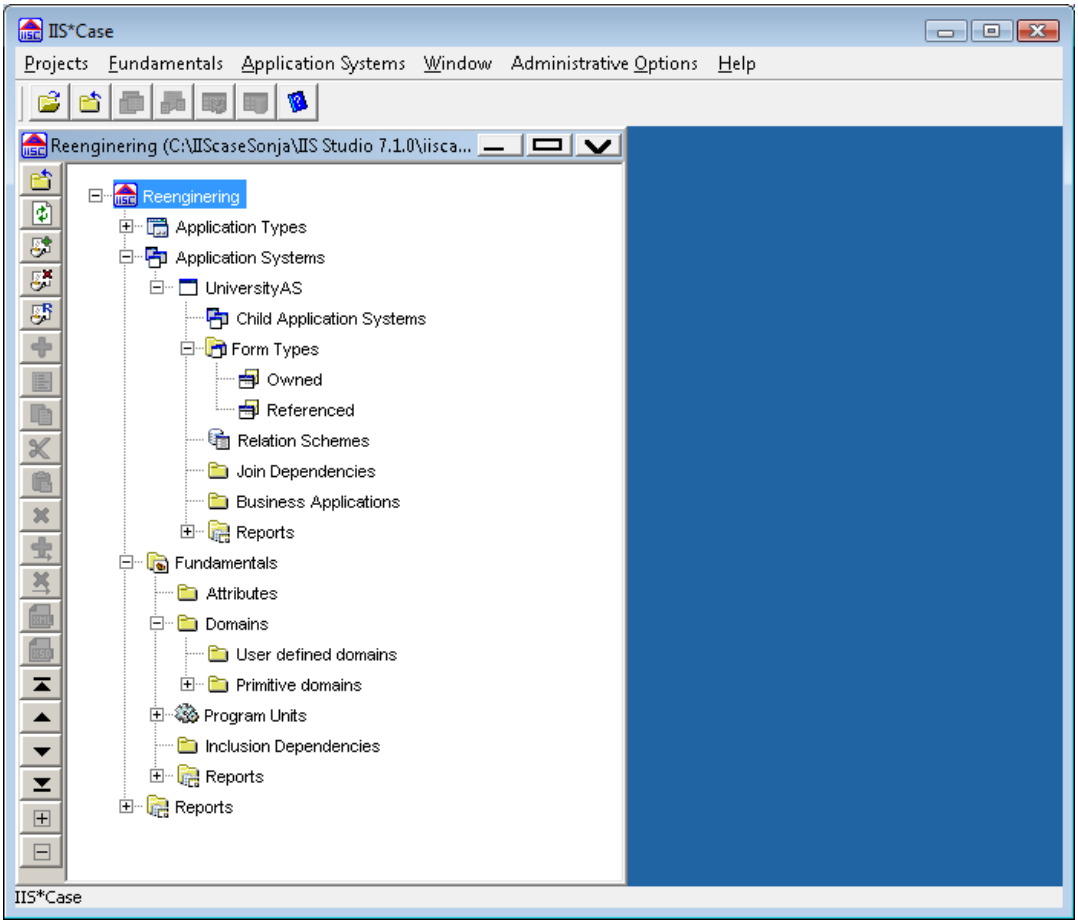
U tekstu koji sledi, opisan je kompletan proces reinženjeringa implementiran u ovom alatu. Prvo je dat uopšteni pregled glavnih koraka procesa, a zatim je svaki korak detaljno objašnjen u narednim odeljcima, uključujući algoritme koji se primenjuju u modulima.

5.2. Opis procesa reinženjeringa

U ovom odeljku ukratko je opisan kompletan proces reinženjeringa implementiran u ovom alatu. Na početku, potrebno je da korisnik kreira novi projekat tj. model aplikacije informacionog sistema putem korisničkog interfejsa alata IIS*Case. Takođe, u okviru projekta treba da kreira po jedan aplikativni sistem za svaku pojedinačnu bazu podataka koja će biti predmet reinženjeringa u okviru iste aplikacije.

Na slici 5.3 prikazana je ekranska forma sa kreiranim projektom pod nazivom *Reenginering*, koji sadrži jedan aplikativni sistem *UniversityAS*, koji je u ovom trenutku potpuno prazan. Takođe, nema podataka ni u osnovnim konceptima (*Fundamentals*), osim nekih predefinisanih primitivnih podataka koje alat nudi.

Za svaki aplikativni sistem pokreće se postupak reverznog inženjeringa čiji rezultat će biti instanca meta-modela PIM koncepata IIS*Case-a, koja reprezentuje konceptualnu šemu odgovarajuće relacione šeme baze podataka na ulazu.



Slika 5.3. Korisnički interfejs okruženja IIS*Studio

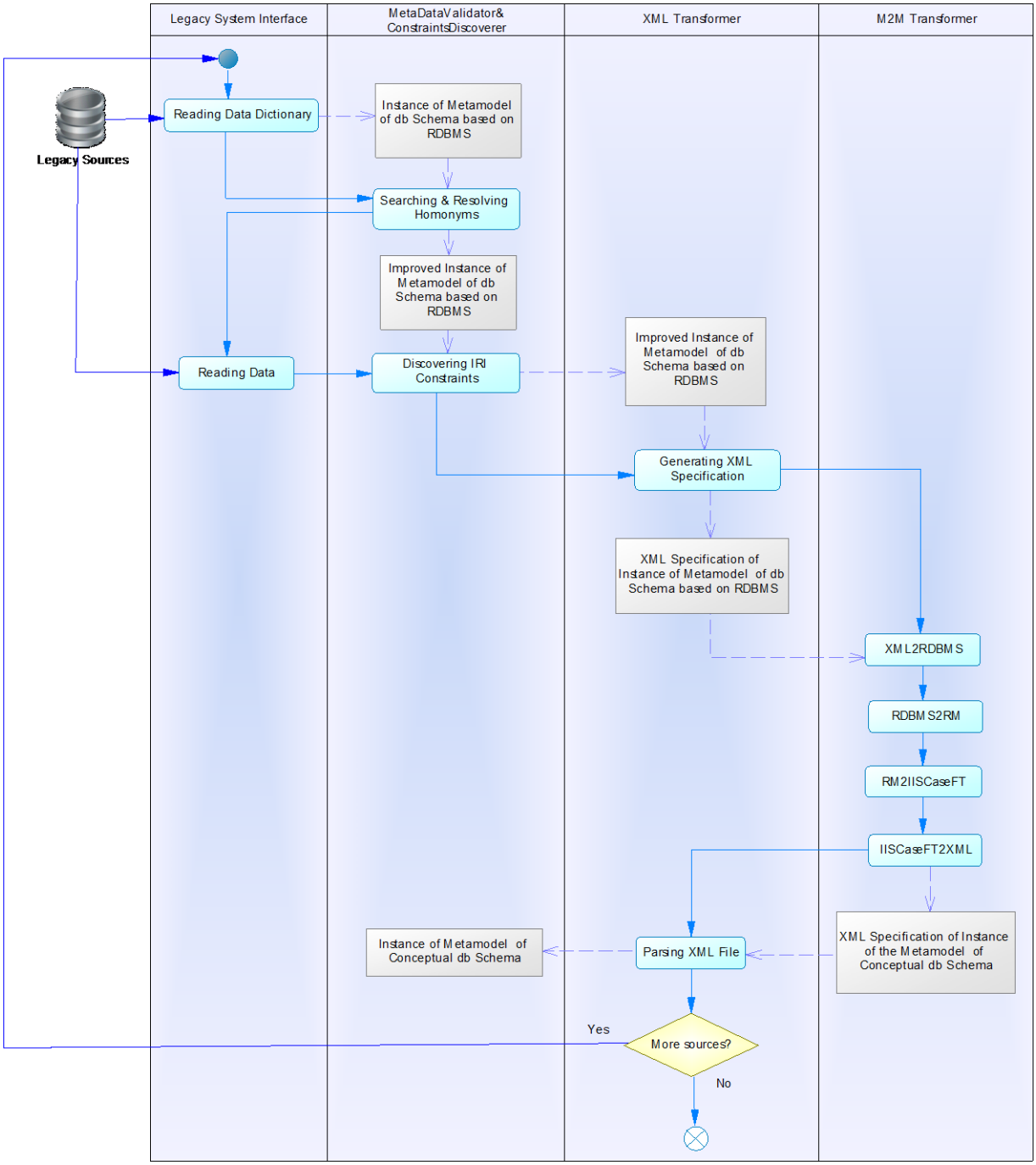
Na slici 5.4 prikazan je dijagram aktivnosti dela procesa reinženjeringa koji se odnosi na reverzni inženjering, dok je na slici 5.5 prikazan dijagram aktivnosti dela reinženjering procesa koji se odnosi na proces *forward* inženjeringa.

Prvi korak u reverznom inženjeringu je ekstrakcija modela. Za ekstrakciju modela neophodno je definisati kako se elementi modela mogu dobiti iz postojeće baze podataka. Ekstrakcija modela uključuje dva glavna zadatka:

- pristupanje bazi podataka zbog dobijanja neophodnih podataka i
- korišćenje tih podataka za kreiranje i inicijalizaciju modela.

Unapređenje semantike (*semantic enrichment*) je proces analize baze podataka radi razumevanja njene strukture i značenja i pronalaženja sakrivene semantike tj. izvlačenja informacija koje se ne mogu dobiti jednostavnim čitanjem iz rečnika podataka SUBP-a.

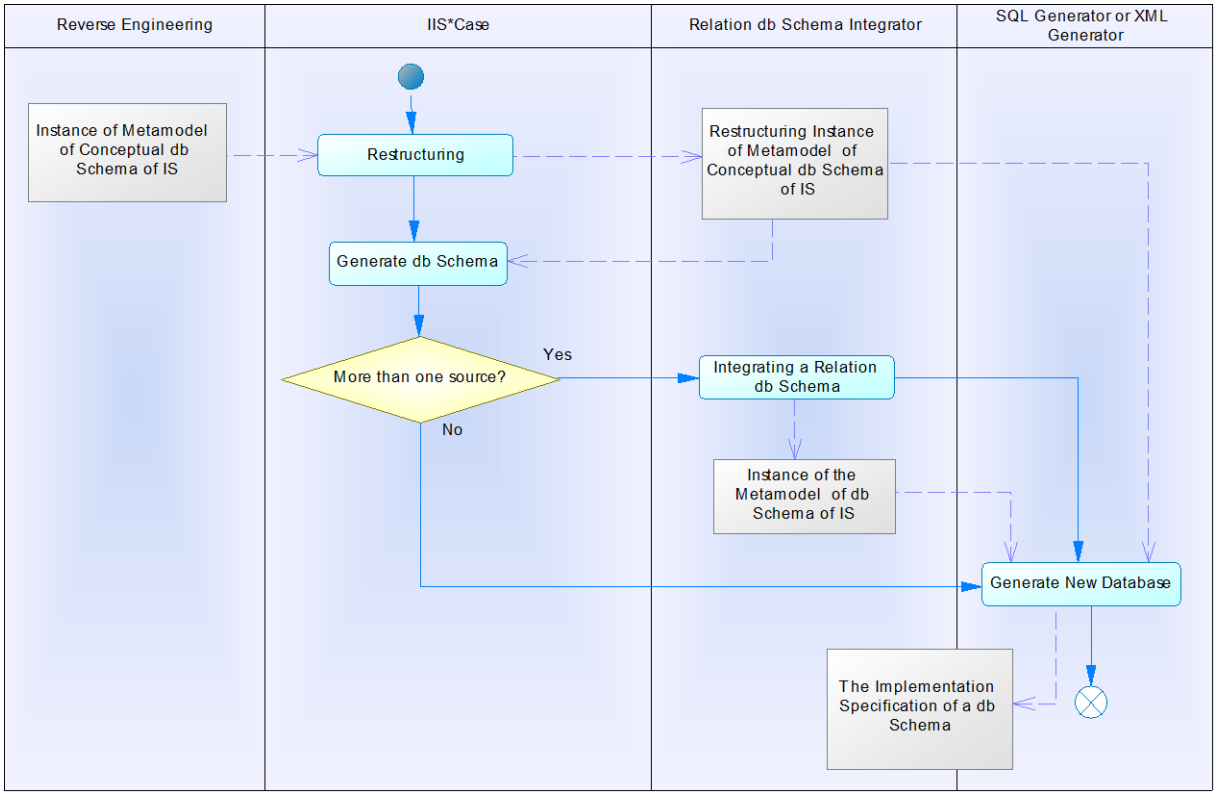
Proces dobijanja semantike počinje ekstrakcijom osnovnih meta-podataka postojeće relacione baze podataka. Projektant interaktivno kontroliše proces ekstrakcije modela, tj. relacione šeme baze podataka, dajući neophodne dodatne informacije. Te informacije se kombinuju sa podacima pročitanim iz rečnika podataka baze podataka i omogućavaju da se sledeći korak, tj. konceptualizacija modela, izvrši automatski.



Slika 5.4. Dijagram aktivnosti procesa reverznog inženjeringa

Na dijagramu aktivnosti sa slike 5.4 prikazani su sledeći osnovni koraci u procesu reverznog inženjeringa, u kojem se kao izvor može naći jedna ili više relacionih baza podataka:

1) *Reading Data Dictionary* - čitanje meta-podataka iz rečnika podataka izabranog SUBP-a i smeštanje u repozitorijum okruženja IIS*Studio,
2) *Searching & Resolving Homonyms* - traženje atributa homonima,
3) *Reading Data* - čitanje podataka iz postojeće baze podataka u cilju unapređivanja semantike modela,
4) *Discovering IRI Constraints* - prepoznavanje ograničenja inverznog referencijalnog integriteta,
5) *Generating XML Specification* - serijalizacija ekstrahovanog unapređenog modela u XML specifikaciju,
6) XML2RDBMS, RDBMS2RM, RM2IISCase i IISCaseFT2XML – niz transformacija modela koje logički model relacione baze podataka zavisan od platforme transformišu u konceptulani model opisan PIM konceptima alata IIS*Case,
7) *Parsing XML File* - parsiranje XML specifikacije, nastale kao rezultat prethodnog koraka i smeštanje u repozitorijum okruženja IIS*Studio, za dalje korišćenje,
8) ukoliko ima više izvora koji predstavljaju ulaz u proces reinženjeringa, za svaki izvor ponavlja se postupak iskazan koracima od 1) do 7).



Slika 5.5. Dijagram aktivnosti procesa *forward* inženjeringa

Na dijagramu aktivnosti sa slike 5.5 prikazane su sledeće aktivnosti u procesu *forward* inženjeringa:

1) *Restructuring* - restrukturiranje, odnosno modifikacija modela zasnovanog na konceptima tipa forme koji je nastao u procesu reverznog inženjeringa,
2) *Generate DB Schema* - generisanje formalnog opisa nove relacione šeme baze podataka,

3) *Integrating a Relation DB Schema* – nakon konsolidacije i analize usaglašenosti podšema sa integrisanom šemom, vrši se integracija podšema ukoliko je bilo više izvora na ulazu u proces reinženjeringa i
4) *Generate New Database* - generisanje novog implementacionog opisa baze podataka za izabrani ciljni model.

Svaki od ovih koraka biće detaljno objašnjen u tekstu koji sledi.

5.3. Ekstrakcija modela iz postojeće baze podataka

Legacy System Interface je modul koji sadrži mehanizme za čitanje rečnika podataka i samih podataka postojeće relacione baze podataka koja je predmet reinženjeringa, u cilju dobijanja reprezentacije implementacione šeme baze podataka nezavisne od konkretnog proizvođača.

Putem ovog modula obezbeđuje se konekcija na odgovarajuću bazu podataka. Ova komponenta je jedina koja je specifična za izvor podataka tj. zavisi od konkretnog proizvođača. Pri vršenju ekstakcije znanja iz različitih izvora ili pri dodavanju novih izvora jedino ona zahteva izmenu.

Pročitani meta-podaci iz rečnika podataka postojeće, nasleđene relacione baze podataka, putem modula *Legacy System Interface* smeštaju se u repozitorijum IIS*Studio okruženja. Podaci koji se ekstrahuju iz rečnika podataka odnose se na: tabele i njihove kolone, primarne ključeve, *check* ograničenja, ograničenja jedinstvenosti vrednosti obeležja, strane ključeve i podržane tipove podatataka izabranog SUBP-a. Kada se meta-podaci pročitaju iz rečnika podataka postojeće baze podataka i smeste u repozitorijumu okruženja IIS*Studio, oni više ne reprezentuju fizičku šemu baze podataka, već logičku, koja ne zavisi od konkretnog proizvođača.

Na slici 5.6. prikazana je ekranska forma kroz koju korisnik vrši specifikaciju parametara potrebnih za konekciju na odgovarajuću bazu podataka i proces reverznog inženjeringa.

Polje *Source Type* određuje tip, dok *Using a Data Source* određuje verziju izvornog servera baze podataka. *Source Type* mogu biti: relacioni model, objektni, objektno-relacioni, XML ili datoteke. U ovoj verziji IIS*Ree nudi korisniku da bira neki od relacionih sistema za upravljanje bazama podataka:

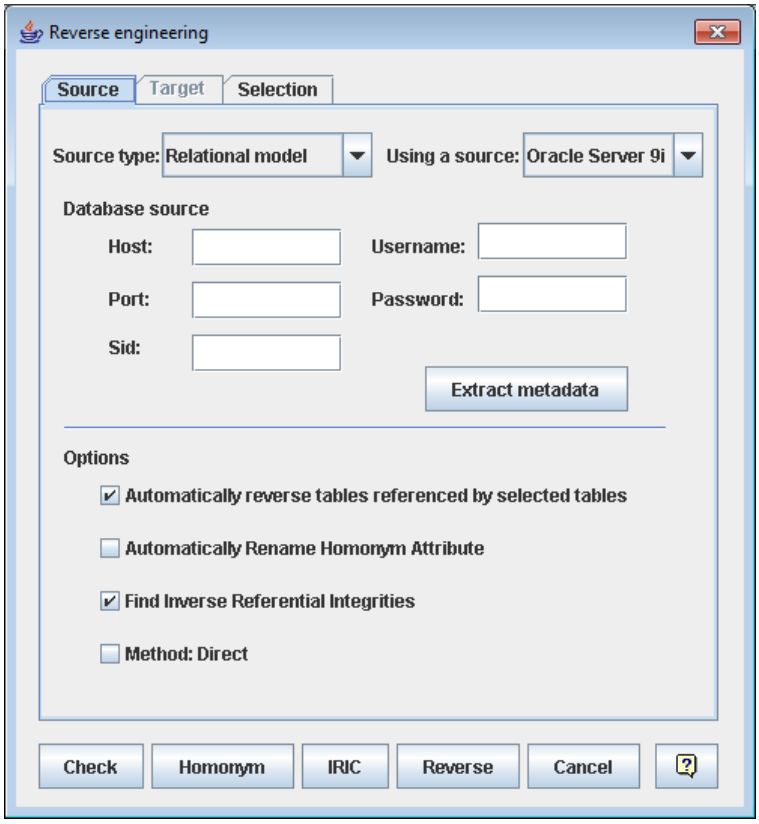
- Oracle Server: verzije 9i, 10g i 11g, kao i
- MS SQL Server: verzije 2000, 2005, 2008 i 2012.

Ukoliko korisnik želi samo da formira novu verziju baze podataka, ili da promeni proizvođača sistema za upravljanje bazama podataka, može odlučiti da to uradi direktno izborom opcije *Method*: *Direct*, bez transformacije u konceptualni model. Ova opcija je omogućena samo kada su izvorni i ciljni tip isti. Za sada je opcija podržana samo za relacioni model.

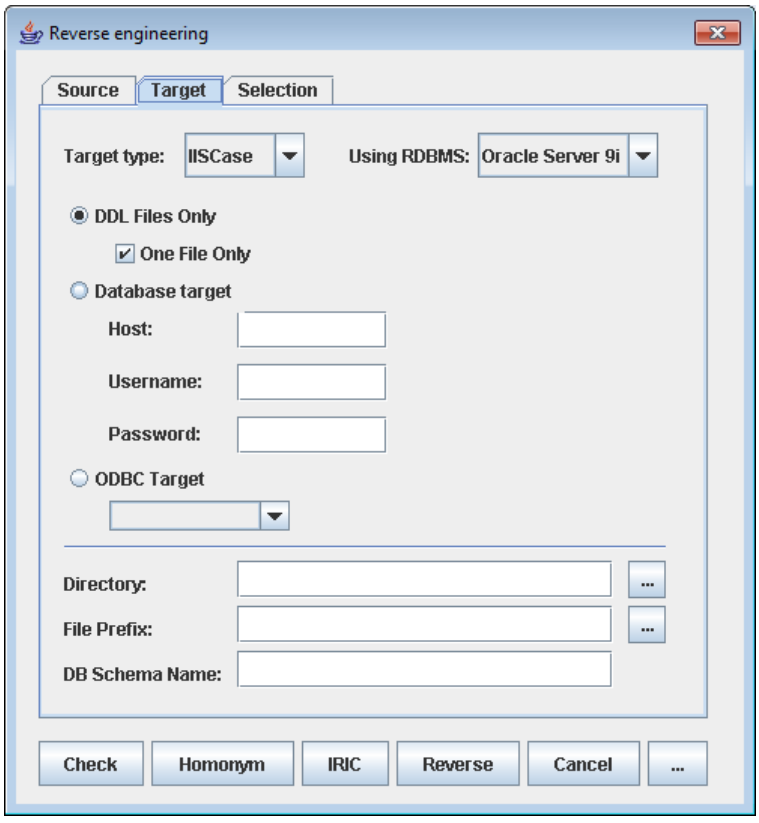
Ukoliko korisnik izabere relacioni model, može da bira sledeće verzije ciljnog servera baze podataka:

- Oracle Server: verzije 9i, 10g i 11g, i
- MS SQL Server: verzije 2000, 2005, 2008 i 2012 i
- ANSI SQL-2003 [145].

Na osnovu izabranog SUBP-a, generiše se nova baza podataka ili odgovarajući SQL opis šeme baze podataka. Ove opcije su korisniku dostupne putem stranice *Target* prikazane na slici 5.7.

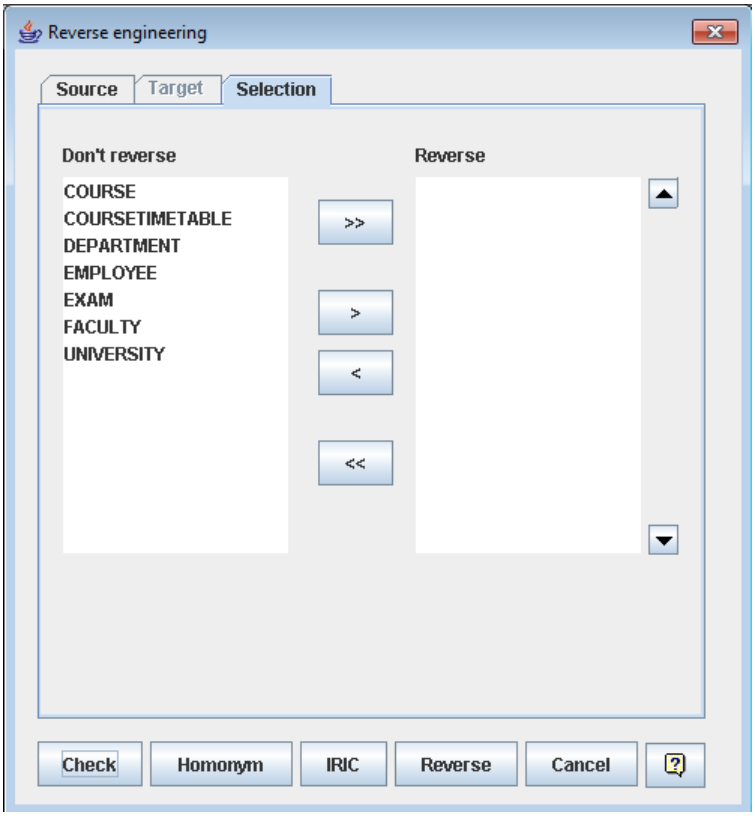


Slika 5.6. Početna ekranska forma alata IIS*Ree, stranica *Source*



Slika 5.7. Prikaz stranice *Target*

Nakon ekstrakcije meta-podataka, može se pregledati skup svih pronađenih šema relacija kao što je prikazano na slici 5.8. Iz skupa svih šema relacija koje pripadaju aplikativnom sistemu, korisnik može da bira one koje želi da budu obuhvaćene procesom reinženjeringa. Omogućena mu je opcija da bira jednu po jednu, po nekoliko selektovanih ili sve odjednom. Lista šema relacija ne sme da bude prazna. U suprotnom, korisnik dobija upozorenje i automatski mu se otvara ova stranica za izbor.

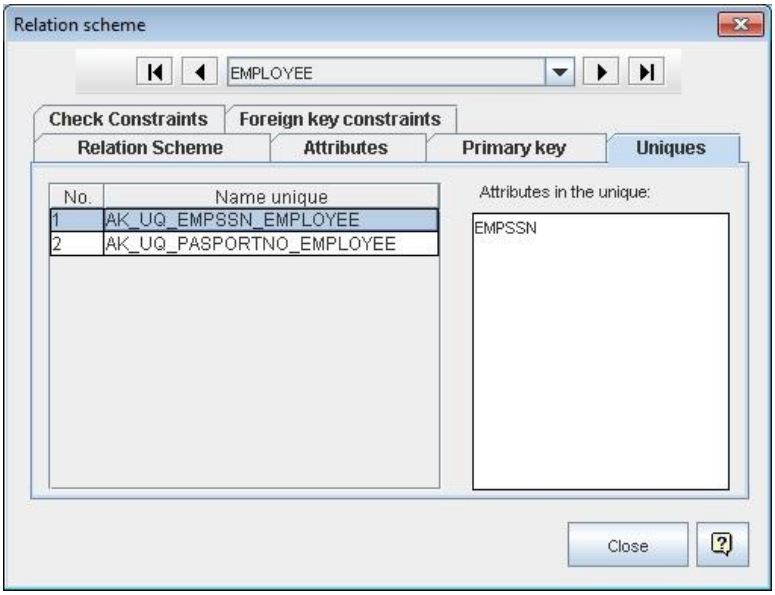


Slika 5.8. Prikaz stranice *Selection*

Ukoliko korisnik izabere neke šeme ralacija, ali ne i sve sa kojima su povezane, alat proverava da li postoje povezene šeme ralacija i nudi mogućnost uključivanja svih povezanih šema relacija ili ih automatski uključuje ukoliko je korisnik izabrao opciju *Automatically reverse tables referenced by selected tables* (slika 5.6).

Za svaku od izabranih šema relacija korisnik može otvoriti dijalog u kojem su prikazani njeni atributi, ključevi, ograničenja jedinstvenosti, *check* ograničenja i ograničenja stranog ključa. Ovaj dijalog je prikazan na slici 5.9.

Radi unapređenja ekstrahovanog modela, nastalog na osnovu podataka pročitanih iz rečnika podataka, *Legacy System Interface* takođe pristupa i podacima koji se nalaze u relacijama, radi dobijanja dodatnih informacija potrebnih za dopunu semantike. Ovi podaci se koriste u narednim koracima procesa reverznog inženjeringa koji se izvršavaju pomoću modula *MetaData Validator & Constraints Discoverer*.



Slika 5.9. Dijalog sa detaljima izabrane šeme relacije

5.4. Traženje atributa homonima

Sledeći korak u procesu reverznog inženjeringa je identifikacija atributa homonima. Na slici 5.4 ova aktivnost je predstavljena kao *Searching & Resolving Homonyms*. Specifikacija šeme baze podataka opisana PIM konceptima alaza IIS*Case, koja predstavlja rezultat procesa reverznog inženjeringa, zasnovana je na pretpostavci o postojanju univerzalne šeme relacije, gde su obeležja, tj. atributi, jedinstveno identifikovani svojim nazivom. Kako ova pretpostavka ne važi u relacionim sistemima za upravljanje bazama podataka, gde se atribut identifikuje unutar šeme relacije kojoj pripada, može se dogoditi da u šemi baze podataka koja je predmet reinženjeringa, postoji više atributa sa istim nazivom, a različitom semantikom, tzv. *homonimi*.

Homonimima se smatraju svi neprimarni atributi koji pripadaju različitim šemama relacija jedne šeme baze podataka. Neka je *Hom* skup homonima šeme baze podataka *S*, tada je:

$$Hom = \{A \mid (\exists N_i(R_i, K_i), N_j(R_j, K_j) \in S\ )\ (A \in R_j \cap R_i\ \wedge\ A \notin K_i \cup K_j)\}.$$

Da bi se mogli primeniti algoritmi ugrađeni u modul *M2M Transformer*, u kojima se polazi od pretpostavke da nema atributa homonima, konflikt u imenovanju mora biti rešen pre njihove primene. U modul *MetaData Validator & Constraints Discoverer* ugrađen je algoritam za pronalaženje homonima. U slučaju da više atributa ima isti naziv, na korisniku je da odabere koje će atribute da preimenuje.

Korisniku se prikazuje skup *Hom* i nude sledeće dve mogućnosti:

- da korisnik sam preimenuje kritična obeležja ili
- da izabere opciju kojom bi se automatski preimenovala sva obeležja iz skupa *Hom*.

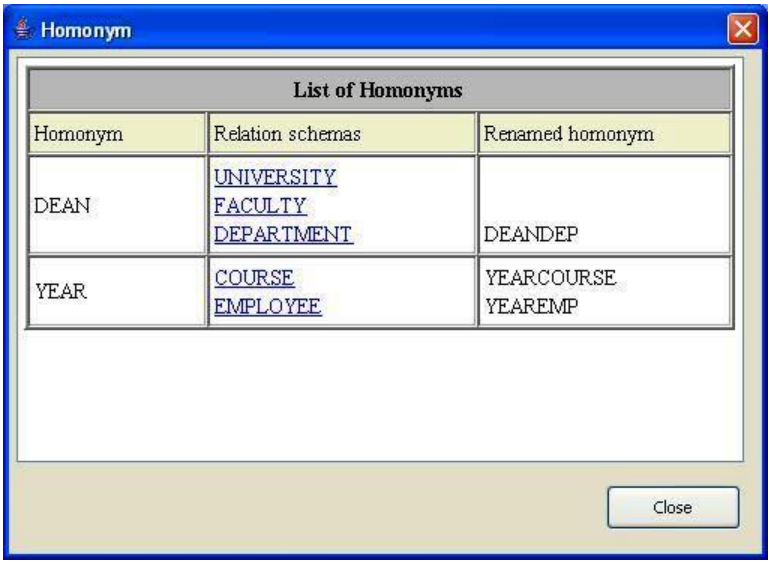
Pri automatskom preimenovanju modul *MetaData Validator & Constraints Discoverer* na naziv obeležja dodaje naziv šeme relacije. Ukoliko korisnik želi automatsko preimenovanje atributa homonima može da uključi opciju *Automatically Rename Homonym Attribute*, koja se nalazi u *Options* (slika 5.6).

Primer 5.1. Šema relacije *Student* sadrži atribut *Year* koji označava godinu studija studenta, dok se u šemi relacije *Employee* atribut *Year* pojavljuje kao godina zaposlenja.

- *Student*({*UniId, FacId, DepId, StudentId, StudFName, StudLName, StudBirth, Year*}, {PrimaryKey(*UniId+FacId+DepId+StudentId*)}),
- *Employee*({*EmpId, UniId, FacId, EmpFName, EmpLName, EmpBirthD, EmpSSN, Year, EmpPosition, PasportNo, WSId*}, {PrimaryKey(*UniId + FacId + EmpId*)}).

Kako *Year* pripada skupovima atributa šema relacija *Student* i *Employee* biće detektovan kao homonim i ponuđen korisniku za preimenovanje. Ukoliko korisnik izabere opciju za automatsko preimenovanje, umesto jednog atributa *Year* bila bi kreirana dva atributa, *YearStudent* i *YearEmployee.*

Na slici 5.10 prikazana je ekranska forma za preimenovanje atributa homonima.



Slika 5.10. Ekranska forma za preimenovanje atributa homonima

5.5. Prepoznavanje ograničenja inverznog referencijalnog integriteta

Pored algoritma za traženje atributa homonima u modul *MetaData Validator & Constraints Discoverer* ugrađen je i algoritam za identifikaciju potencijalnih ograničenja inverznih referencijalnih integriteta, čime se, takođe, semantički unapređuju meta-podaci pročitani iz rečnika podataka postojeće baze podataka.

Traženje mogućih ograničenja inverznih referencijalnih integriteta predstavlja sledeći korak u procesu reverznog inženjeringa koji je na dijagramu aktivnosti sa slike 5.4 prikazan kao aktivnost *Discovering IRI Constraints*. Najvažnije potrebne informacije u ovom koraku su o ograničenjima referencijalnih integriteta, $N_l[LHS] \subseteq N_r[RHS]$ tj. o skupovima atributa levih (*LHS*) i desnih strana ograničenja (*RHS*). Pored toga, ova aktivnost zahteva dodatne informacije, zbog čega je potrebno pristupiti podacima postojeće baze podataka. Ova aktivnost pristupanja podacima postojeće baze podataka, koja se izvršava putem modula *Legacy System Interface*, na dijagramu je prikazana kao *Reading Data*. U ovom koraku je, takođe, neophodna interakcija sa korisnikom zbog obezbedjivanja nedostajuće semamtike.

Za postojanje ograničenja inverznog referencijalnog integriteta potrebno je da postoji odgovarajuće ograničenje referencijalnog integriteta. U RSUBP-ovima skupu ograničenja

referencijalnog integriteta ne pripada svako ograničenje stranog ključa. U RSUBP-ovima je u ograničenjima stranog ključa dozvoljeno da sa desne strane ograničenja ne bude uopšte zastupljen ključ šeme relacije, već može da se nađe i skup atributa nad kojim je definisano ograničenje jedinstvenosti. U opštem slučaju, ti atributi smeju da imaju nula vrednost, zbog čega ne pripadaju skupu ključeva. Iz ovog razloga takva ograničenja stranog ključa ne pripadaju skupu ograničenja referencijalnog integriteta.

Pseudokod algoritma za traženje potencijalnih ograničenja inverznog referencijalnog integriteta prikazan je na slici 5.11. Za svako ograničenje stranog ključa, iz skupa ograničenja koja su se nalazila u rečniku podataka postojeće baze podataka, proverava se da li se radi o ograničenju referencijalnog integriteta. Ova provera se vrši pomoću procedure *ProveravanjePostojanjaRIC* čiji pseudokod je prikazan na slici 5.12. Ulazni parametar procedure je ograničenje stranog ključa, dok je izlazni parametar indikator *ind* koji se postavlja na jedinicu ukoliko se sa desne strane ograničenja nalazi ključ. U suprotnom inikator *ind* se postavlja na nulu.

Za svako ograničenje stranog ključa koje predstavlja referencijalni integritet radi se sledeće:

- pristupa se podacima relacije koja se nalazi na desnoj strani ograničenja i traže se vrednosti atributa skupa *RHS*,
- pristupa se podacima relacije koja se nalazi na levoj strani ograničenja i traže se vrednosti atributa skupa *LHS* koji nemaju nula vrednost (traže se samo različite vrednosti),
- ukoliko se skupovi rezultata ova dva traženja poklapaju, može se pretpostaviti da između šema relacija N_l i N_r postoji ograničenje inverznog referencijalnog integriteta, $N_r[RHS] \subseteq N_l[LHS]$.

Algoritam:	ZA TRAŽENJE IRIC-a
Ulaz:	*DB* $S = \{N_i(R_i, O_i) \mid i = 1,.., n\}$ $FK = \{fk_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$
Izlaz:	$IRIC = \{iric_i: N_r[RHS] \subseteq N_l[LHS] \mid i = 1,.., n\}$
Lokalne deklaracije: *ind* - izlazni parametar iz procedure *ProveravanjePostojanjaRIC* (*ind* = 1 – na desnoj strani je ključ, *ind* = 0 – na desnoj strani nije ključ)	
Pseudokod: **POČETAK PROCESA** TražiIRIC **RADI** traženje ($\forall fk \in FK$) **POZOVI** *ProveravanjePostojanjaRIC*(*fk*, *ind*) ***AKO JE*** *ind* = 1 **TADA** **POSTAVI** $r \leftarrow \pi_{RHS}(N_r)$ **POSTAVI** $s \leftarrow \sigma_{LHS \text{ is not null}}(\text{distinct } \pi_{LHS}(N_l))$ ***AKO JE*** *size*(*r*) = *size*(*s*) **TADA** **POSTAVI** $IRIC \leftarrow IRIC \cup (N_r[RHS] \subseteq N_l[LHS])$ ***KRAJ AKO*** **KRAJ RADI** traženje **KRAJ PROCESA** TražiIRIC	

Slika 5.11. Pseudokod algoritma za traženje ograničenja inverznog referencijalnog integriteta

Treba naglasiti da poslednju reč ima projektant koji odlučuje da li se radi o ograničenju inverznog referencijalnog integriteta ili je možda vremenom došlo do toga da za svaku različitu vrednost atributa iz *RHS* referencirane šeme relacije postoji bar jedna torka iz referencirajuće šeme relacije koja je referencira.

U pseudokodu algoritma za traženje ograničenja inverznog referencijalnog integriteta, prikazanom na slici 5.11 koriste se operatori relacione algebre:

- σ za selekciju, po selekcionoj formuli F= *LHS* is not null i
- π za projekciju na skup obeležja *RHS* ili *LHS*.

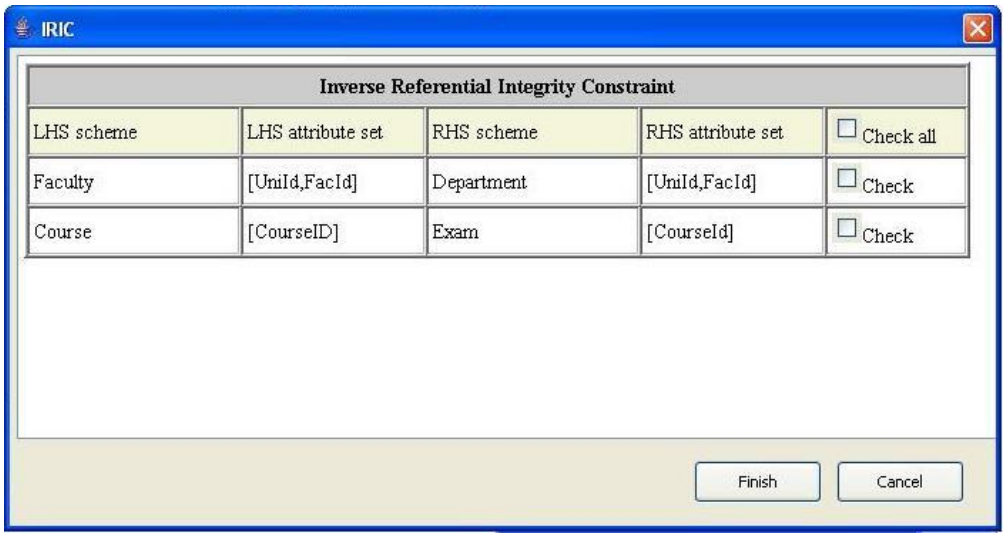
Procedura:	*ProveravanjePostojanjaRIC*
Formalni parametri:	
Ulazni:	*fk*
Izlazni:	*ind*

Pseudokod:

POČETAK PROCESA ProveravanjePostojanjaRIC
  POSTAVI *ind* ← *1*
  RADI proveru (∀*A* ∈ *RHS* **DOK JE** *ind* = 1)
    AKO JE *A* = ω **TADA**
      POSTAVI *ind* ← 0
    KRAJ AKO
  KRAJ RADI proveru
KRAJ PROCESA ProveravanjePostojanjaRIC

Slika 5.12. Pseudokod procedure *ProveravanjePostojanjaRIC*

Ekranska forma za otkrivanje i validaciju ograničenja inverznog referencijalnog integriteta prikazana je na slici 5.13. Iz skupa ponuđeninih ograničenja inverznog referencijalnog integriteta korisnik označava ona za koja se slaže da pripadaju ograničenjima tog tipa. Ukoliko korisnik ne želi da razmatra ograničenja inverznog referencijalnog integriteta može da isključi opciju *Find Inverse Referential Integrities*, koja se nalazi u sekciji *Options*, prikazanoj na slici 5.6.



Slika 5.13. Ekranska forma za izbor ograničenja inverznog referencijalnog integriteta

5.6. XML specifikacija šema baza podataka

Sledeći korak u procesu reverznog inženjeringa je generisanje XML specifikacije logičke šeme baze podataka smeštene u repozitorijumu okruženja IIS*Studio. Ovaj korak je neophodan zbog povezivaja sa modulom *M2M Transformer* koji je, za razliku od prethodno prikazanih modula implementiranih u okviru okruženja Oracle JDeveloper, implementiran u Eclipse okruženju.

Pošto je XML prihvaćen kao format za smeštanje podataka i upravljanje podacima, ekstrahovani modeli se serijalizuju kao XML specifikacije. Ovo radi *XML Transformer* modul. On uzima podatke iz repozitorijuma okruženja IIS*Studio, koji predstavljaju formalni opis logičke šeme baze podataka, nastale kao rezultat iz prethodna dva modula, i primenjuje algoritam za kreiranje XML specifikacije na osnovu predefinisane XML šeme čiji deo je predstavljen u listingu 5.1. Na dijagramu aktivnosti sa slike 5.4 ovaj korak je predstavljen kao aktivnost *Generating XML Specification*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.uns.ac.rs/dbXMLSchema"
xmlns="http://www.uns.ac.rs/dbXMLSchema" elementFormDefault="qualified">
  <xsd:element name="RSUBP">
    <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
         <xsd:element ref="Database" minOccurs="1" maxOccurs="unbounded"/>
         <xsd:element ref="SystemDataType" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="Database">
    <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
         <xsd:element ref="Table" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="UserDefinedDataType" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Table">
    <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
         <xsd:element ref="Column" minOccurs="1" maxOccurs="unbounded"/>
         <xsd:element ref="PrimaryKeyCon" minOccurs="1" maxOccurs="1"/>
         <xsd:element ref="UniqueCon" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="CheckCon" minOccurs="0" maxOccurs="unbounded"/>
         <xsd:element ref="ForeignKey" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
...
</xsd:schema>
```

Listing 5.1. Deo XML šeme za serijalizaciju logičke šeme relacione baze podataka

Na listingu 5.2 prikazan je primer XML specifikacije, kreiran na osnovu XML šeme prikazane u listingu 5.1. U primeru je predstavljen deo šeme baze podataka informacionog sistema nekog univerziteta.

```
<?xml version="1.0" encoding="UTF-8"?>
<db:RSUBP xmlns:db="http://www.uns.ac.rs/dbXMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.uns.ac.rs/dbXMLSchema dbXMLSchema.xsd ">
  <db:Name>Oracle10g</db:Name>
  <db:Database>
    <db:Name>UniversityDB</db:Name>
    <db:Table>
      <db:Name>University</db:Name>
      <db:Column>
        <db:Name>UniId</db:Name>
        <db:Nullable>false</db:Nullable>
        <db:DataType>integer</db:DataType>
      </db:Column>
      <db:Column>
        <db:Name>UniName</db:Name>
        <db:Nullable>false</db:Nullable>
        <db:DataType>String</db:DataType>
        <db:Length>20</db:Length>
      </db:Column>
      <db:Column>
        <db:Name>UniCity</db:Name>
        <db:Nullable>false</db:Nullable>
        <db:DataType>String</db:DataType>
        <db:Length>20</db:Length>
      </db:Column>
      <db:Column>
        <db:Name>RectorId</db:Name>
        <db:Nullable>true</db:Nullable>
        <db:DataType>integer</db:DataType>
      </db:Column>
      <db:PrimaryKeyCon>
        <db:Name>PK_University</db:Name>
        <db:PKColumn>UniId</db:PKColumn>
      </db:PrimaryKeyCon>
    </db:Table>
    <db:Table>
    <db:Name>Faculty</db:Name>
...
```

Lisitnig 5.2. Primer XML specifikacije koja je u skladu sa XML šemom iz listinga 5.1

Modul *XML Transformer* omogućava, takođe, da se konceptualna šema baze podataka koja se dobija kao rezultat transformacije, realizovane pomoću *M2M Transformer*-a i opisana sintaksom XML jezika, smesti u repozitorijum IIS*Studio okruženja. XML specifikacija konceptualne šeme baze podataka generiše se na osnovu unapred definisane XML šeme čiji je deo prikazan na listingu 5.3. Primer XML specifikacije koja je u skladu sa ovom XML šemom prikazan je na listingu 5.4. Aktivnost koja opisuje ovaj postupak na dijagramu aktivnosti prikazanom na slici 5.4 nazvana je *Parsing XML File*.

Kompletne XML šeme, za serijalizaciju logičke šeme relacione baze podataka i konceptualne šeme baze podataka, nalaze se u prilogu A.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.uns.ac.rs/dbXMLSchema"
xmlns="http://www.uns.ac.rs/dbXMLSchema" elementFormDefault="qualified">
  <xsd:element name="ISApplicationModel">
    <xsd:complexType>
      <xsd:sequence>
```

```
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element ref="Fundamentals" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="ApplicationSystem" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Fundamentals">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="PrimitiveDomain" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="UserDefinedDomainFromPrimitiveDomain" minOccurs="0"
                                                          maxOccurs="unbounded"/>
        <xsd:element ref="UserDefinedDomainFromUserDefinedDomain" minOccurs="0"
                                                          maxOccurs="unbounded"/>
        <xsd:element ref="Attribute" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="ApplicationSystem">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element ref="FormTypeProgram" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Listing 5.3. Deo XML šeme za serijalizaciju konceptualne šeme baze podataka

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<ISAplicationModel xmlns="http://www.uns.ac.rs/dbXMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.uns.ac.rs/dbXMLSchema dbXMLSchema.xsd">
  <Fundamentals>
    <Attribute>
      <AttributeID>aidb0</AttributeID>
      <Name>FacShortName</Name>
      <Description>FacShortName</Description>
      <UserDefinedDomain>udpd7</UserDefinedDomain>
    </Attribute>
    <Attribute>
      <AttributeID>aidb1</AttributeID>
      <Name>FacName</Name>
      <Description>FacName</Description>
      <UserDefinedDomain>udpd8</UserDefinedDomain>
    </Attribute>
    <PrimitiveDomain>
      <PrimitiveDomainID>pd0</PrimitiveDomainID>
      <Name>Integer</Name>
      <DefaultLength>0</DefaultLength>
      <DefaultDecimalPlaces>0</DefaultDecimalPlaces>
    </PrimitiveDomain>
    <PrimitiveDomain>
      <PrimitiveDomainID>pd1</PrimitiveDomainID>
      <Name>String</Name>
      <DefaultLength>30</DefaultLength>
      <DefaultDecimalPlaces>0</DefaultDecimalPlaces>
    </PrimitiveDomain>
    <UserDefinedDomainFromPrimitiveDomain>
      ...
  </Fundamentals>
  <ApplicationSystem>
    <FormTypeProgram>
      <Name>FormType_University</Name>
```

```
      <Title>FormType_University</Title>
      <Frequency>1</Frequency>
      <ResponseTime>1</ResponseTime>
      <ConsideredINDBSchDesign>true</ConsideredINDBSchDesign>
      <ComponentTypeRoot>
        <Name>ComponentType_University</Name>
        <Title>ComponentType_University</Title>
        <Query>true</Query>
        <Insert>false</Insert>
        <Delete>false</Delete>
        <Update>false</Update>
      </ComponentTypeRoot>
    </FormTypeProgram>
  </ApplicationSystem>
  <Name>IISCase_Project_Oracle10g</Name>
</ISAplicationModel>
```

Listing 5.4. Primer XML specifikacije koja je u skladu sa XML šemom iz listinga 5.3

5.7. Transformacija modela

U MDSE generalno, a naročito u MDA, modeli nisu samo artefakti projektanata, tj. ne koriste se samo u prvim fazama razvoja nekog informacionog sistema. Modeli su uključeni u čitav proces razvoja, pri čemu se na kraju može generisati programski kod za izabranu ciljnu platformu. Ovi modeli razlikuju se po meri sadržavanja onih informacija koje su specifične za platformu. Projektant počinje sa modelom koji je nazavisan od platforme, i kroz niz model-u-model transformacija, transformiše seriju modela specifičnih za platformu sa sve manjim stepenom nezavisnosti, tj. sa sve većim i većim specifičnostima vezanim za platformu. U procesu reverznog inženjeringa postupak se kreće u obrnutom smeru. Kreće se od modela sa velikim stepenom zavisnosti od platforme, a zatim se nizom model-u-model transformacija dolazi do konceptualnog modela.

U ovoj doktorskoj disertaciji posebno su obrađene transformacije šema baza podataka, namenskih za relacione sisteme za upravljanje bazama podataka, koji su veoma zavisni od platforme, u opis šeme baze podataka pomoću koncepata alata IIS*Case, koji se smatra nezavisnim od platforme.

Transformacija modela je centralni korak u procesu reverznog inženjeringa, i predstavlja konceptualizaciju modela postojeće baze podataka, ekstrahovanog u prethodnim koracima. Ovaj korak izvršava se unutar modula *M2M Transformer*. Ovaj modul implementiran je potpuno u skladu sa MDA pristupom, u Eclipse okruženju, pomoću ATL jezika za transformaciju modela. U njega je ugrađen skup algoritama pomoću kojih se automatski vrši niz transformacija modela, počev od opisa relacione šeme baze podataka, orijentisanog ka implementacji (PSM), u opis šeme baze podataka na višem apstraktnom nivou (PIM), koji predstavlja konceptulani model opisan PIM konceptima alata IIS*Case.

Putem modula *M2M Transformer,* model, nastao iz postojeće baze podataka, translira se nizom uzastopnih transformacija u novi model, svaki put bliži ciljnom konceptualnom modelu.

Ukupnu transformaciju čini kompozicija pojedinačnih transformacija u kojima model na izlazu iz jedne transformacije predstavlja model na ulazu u drugu. Na dijagramu aktivnosti sa slike 5.4 transformacije su predstavljene kao sledeće aktivnosti:

- XML2RDBMS,
- RDBMS2RM,
- RM2IISCase i
- IISCase2XML.

Sve četiri transformacije izvršavaju se u MOF tehnološkom prostoru. Svi modeli, ulazni i izlazni, u skladu su sa odgovarajućim meta-modelima nad kojima se definišu metode transformacije. Meta-modeli će detaljno biti prikazani u šestom poglavlju. Prva i poslednja transformacija potrebne su zbog prilagođavanja modela, smeštenog u repozitorijum okruženja IIS*Studio, MOF tehnološkom prostoru, i obrnuto.

Logička šema baze podataka opisana XML jezikom se pomoću transformacije XML2RDBMS transformiše u model opisan pomoću koncepata relacionih sistema za upravljanje bazama podataka. Ovaj model se pomoću transformacije RDBMS2RM transformiše u model koji je u skladu sa generičkim meta-modelom koji reprezentuje koncepte relacione šeme baze podataka zasnovane na teoretskim principima. Novonastali model se potom, RM2IISCase transformacijom, transformiše u model koji predstavlja konceptualnu šemu baze podataka zasnovanu na PIM konceptima IIS*Case alata. Rezultat ove transformacije se transformiše u XML model pomoću transformacije IISCase2XML i serijalizuje u XML format. Podaci iz dobijene XML specifikacije, koja predstavlja izlaz iz modula *M2M Transformer*, već opisanim postupkom u prethodnom odeljku, smeštaju se u repozitorijum okruženja IIS*Studio.

Kompletan proces transformacije, svi algoritmi ugrađeni u *M2M Transformer* modul, kao i njihova implementacija biće detaljno prikazani u sedmom poglavlju.

Ukoliko ima više baza podataka koje pripadaju informacionom sistemu nad kojim se vrši reinženjering, za svaku se ponavlja prethodno opisani postupak. On započinje kreiranjem novog aplikativnog sitema u okviru modela aplikacije informacionog sistema, zatim se nastavlja čitanjem meta-podataka iz rečnika podataka odgovarajuće baze podataka, traženjem homonima i potencijalnih ograničenja inverznih referencijalnih integriteta, do transformacije modela zavisnog od platforme, zasnovanog na relacionom modelu, u konceptulani model opisan PIM konceptima alata IIS*Case.

Nakon smeštanja u repozitorijum okruženja IIS*Studio, sve tipove formi, kao i osnovne koncepte modela aplikacije nastale transformacijom, korisnik može pregledati putem alata IIS*Case, odakle može nastaviti sa reinženjeringom: sa izmenama ili restrukturiranjem ili započeti postupak *forward* inženjeringa.

5.8. Forward *inženjering*

Nakon postupka reverznog inženjeringa, projektant može modifikovati model predložen od strane alata, na konceptualnom nivou, kroz korisnički interfejs alata IIS*Case. Na dijagramu aktivnosti prikazanom na slici 5.5 ova aktivnost je predstavljena kao *Restructuring*. Nakon toga, korisnik-projektant može generisati novu šemu baze podataka datog aplikativnog sistema. Ova aktivnost je na dijagramu sa slike 5.5 nazvana *Generate DB Schema*. Ukoliko postoji više aplikativnih sistema datog informacionog sistema, kao posledica postojanja više baza podataka na ulazu u proces reinženjeringa, vrši se analiza i integracija podšema, putem aktivnosti, nazvane *Integrating a Relation DB Schema*. Na kraju svakog međukoraka, projektant može proveriti i validirati dobijene modele.

Poslednji korak u reinženjeringu baze podataka je generisanje nove baze podataka ili implementacionog opisa za izabrani ciljni SUBP. Arhitektura alata IIS*Ree omogućava dodavanje novih generatora implementacionog opisa šeme baze podataka za druge proizvođače relacionih sistema za upravljanje bazama podataka ili sisteme za upravljanje bazama podataka zasnovane na nekim drugim modelima podataka, kao što su objektni, objektno-relacioni ili XML modeli.

Putem okruženja IIS*Studio, na osnovu novokreirane baze podataka može se generisati i prototip poslovne aplikacije. Kompletan postupak *forward* inženjeringa već je opisan u četvrtom poglavlju.

6. Meta-modeli šema baza podataka

U prethodnom poglavlju ukratko su prikazani osnovni koraci procesa transformacije koji se obavlja u okviru modula *M2M Transformer*. Proces transformacije sastoji se od kompozicije transformacija koje su zasnovane na meta-modelima.

U ovom poglavlju biće detaljno prikazani koncepti meta-modela, odnosno elementi apstraktnih sintaksi svih jezika za modelovanje šema baza podataka, koji učestvuju u procesu transformacije. Za specifikaciju meta-modela korišćen je EMF [EMF13] i njegov jezik za meta-modelovanje Ecore, koji je zasnovan na EMOF specifikaciji [MOF13].

U odeljcima od 6.1 do 6.4 biće prikazani sledeći meta-modeli:

- XML meta-model,
- RSUBP meta-model,
- generički meta-model relacione šeme baze podataka i
- meta-model PIM koncepata alata IIS*Case.

Meta-modelom se definiše apstraktna sintaksa jezika za modelovanje, odnosno konstrukcije koje se mogu koristiti za definisanje validnih modela [Assma06]. U cilju unapređenja semantike meta-modela korišćen je *Object Constraint Language* (OCL) koji omogućava definisanje dodatnih ograničenja između elemenata jezika.

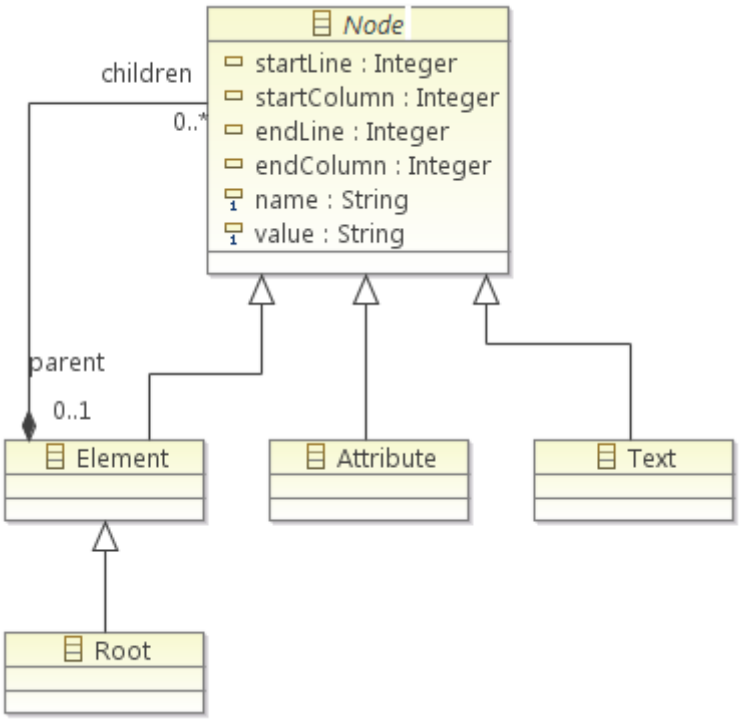
6.1. XML meta-model

XML meta-model koristi se u svrhu prilagođavanja XML specifikacije, koja predstavlja ulaz u proces transformacije, MOF tehnološkom prostoru. U XML dokumentu se nalazi instanca meta-modela relacione šeme baze podataka karakteristične za RSUBP-ove. Kako je instanca definisana u XML tehnološkom prostoru potrebno je da se njen format prilagodi formatu instance istog meta-modela u MOF tehnološkom prostoru.

U ovoj doktorskoj disertaciji korišćen je meta-model predložen od strane NetBeans [XML06]. Meta-model je prikazan na slici 6.1. NetBeans je integrisano razvojno okruženje za razvoj, primarno, pomoću Java programskog jezika. Okruženje je nastalo 1996. godine na Fakultetu za matematiku i fiziku iz Praga kao studentski projekat. 1999. godine NetBeans kupio je Sun Microsistem koji je 2010. godine prešao u vlasništvo Oracle korporacije.

XML meta-model opisuje XML dokument koji se sastoji od jednog korenog čvora, predstavljenog apstraktnom klasom ***Root***. Apstraktne klase će u tekstu, kao i u meta-modelima biti prikazane u *italic* formatu. Sve klase, atributi klasa i veze biće predstavljene u **bold** formatu.

Node je apstraktna klasa koju nasleđuju sledeće tri klase: **Element**, **Attribute** i **Text**. Klasa **Element** reprezentuje *tag*-ove i može da sadrži više čvorova (***Node***-ova). Klasa **Attribute** reprezentuje atribute, koji se mogu definisani na nekom *tag*-u. Klasa **Text** predstavlja poseban čvor, koji ne izgleda kao *tag*, već kao niz karaktera.



Slika 6.1. XML meta-model

6.2. RSUBP meta-model

U ovom odeljku biče prikazan meta-model šeme baze podataka specifičan za relacione sisteme za upravljanje bazama podataka. Meta-model je zasnovan na standardima SQL:1999 [SQL:1999] do SQL:2011 [SQL:2011] koje podržava većina komercijalnih i *open source* RSUBP-ova u koje spadaju Oracle i MSSQL Server. Treba napomenuti da je Oracle objektno-relacioni SUBP i da su u ovom meta-modelu izostavljeni objektni koncepti.

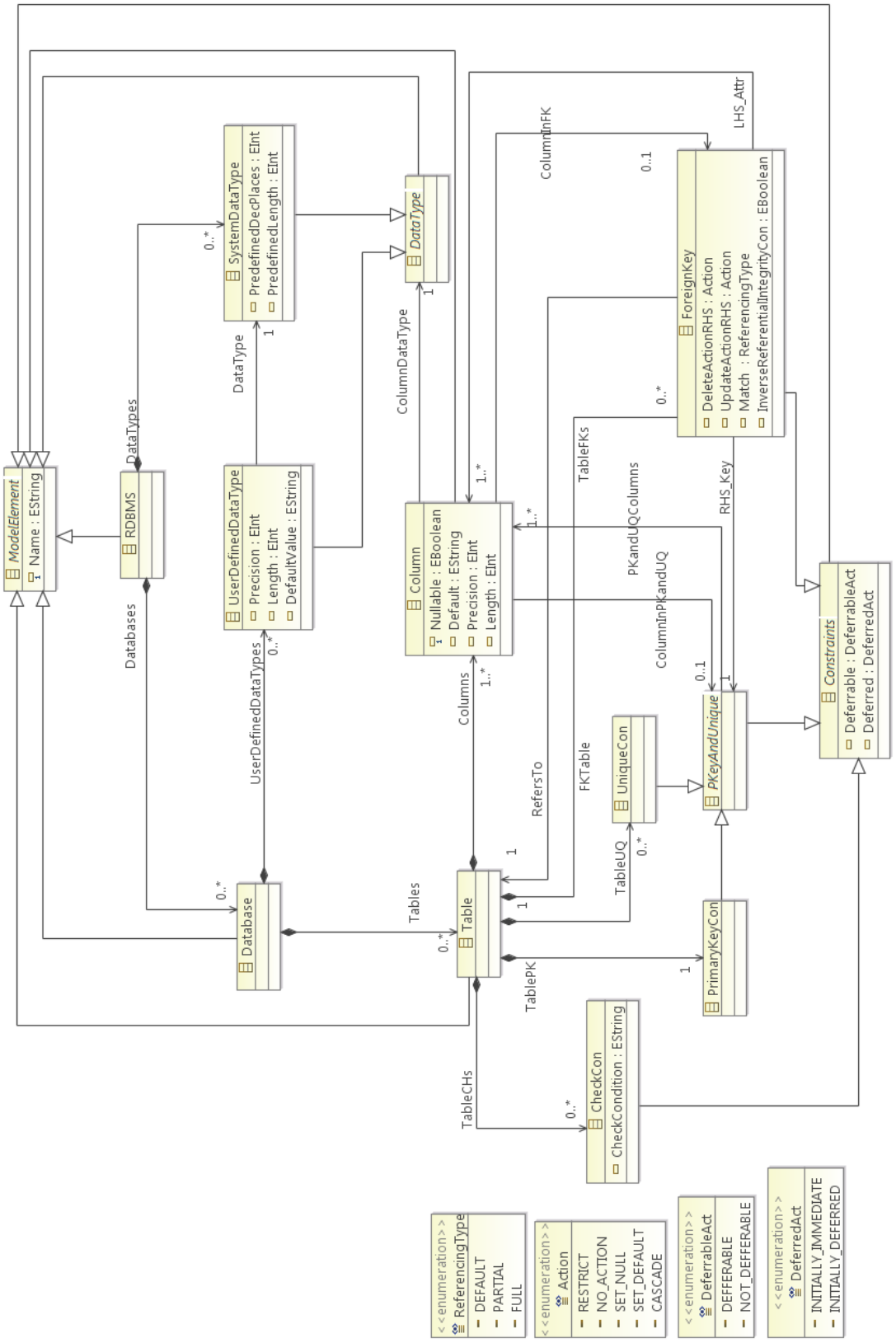
Na slici 6.2 prikazan je meta-model relacione šeme baze podataka kojim su obuhvaćeni osnovni koncepti potrebni u procesu M2M transformacije u cilju reverznog inženjeringa šeme baze podataka. Osnovni koncepti od kojih se sastoji ovaj meta-model: **RDBMS**, **SystemDataType**, **UserDefinedDataType**, **Database**, **Table**, **Column** i ***Constraints***. Korišćena terminologija u ovom meta-modelu karakteristična je za RSUBP-ove. Osnovni koncepti za modelovanje u meta-modelu generalizovani su apstraktnom klasom ***ModelElement*** u kojoj se nalazi atribut **Name**, koji nasleđuju sve ostale klase. Definicije osnovnih koncepata meta-modela relacione šeme baze podataka biće date u odeljku 6.3.

Kako Ecore zahteva XML strukturu, svaki meta-model mora da ima korenski element. U ovom slučaju, korenski element je klasa **RDBMS** koja predstavlja odgovarajući relacioni sistem za upravljanje bazama podataka, za koji se mora zadati naziv (**Name**).

Svaki RSUBP ima svoj skup sistemskih tipova podataka, koji su u meta-modelu predstavljeni klasom **SystemDataType,** npr. integer, real, string itd. Pri specifikaciji sistemskog tipa podatka mora se zadati naziv (atribut **Name**), dok su dužina i broj cifara iza decimalne tačke, predstavljeni u meta-modelu atributima **PredefinedLength** i **PredefinedDecPlaces**, osobine primenljive samo na određene tipove podataka i njihovo zadavanje nije obavezno.

RSUBP može da sadrži više šema baza podataka koje su predstavljene u meta-modelu klasom **Database**. Za svaku šemu baze podataka, takođe, mora biti zadat njen naziv (**Name**). U okviru svake šeme baze podataka, korisnik može definisati i svoje tipove podataka koje će

koristiti (korisnički tipovi podataka). Korisnički definisani tip podatka je u meta-modelu predstavljen klasom **UserDefinedDataType**. Za korisnički tip podatka mora se definisati njegov naziv (**Name**), dok se dužina, preciznost i podrazumevana vrednost zadaju po potrebi. Ove osobine predstavljene su u meta-modelu atributima **Length**, **Precision** i **DefaultValue**.



Slika 6.2. RSUBP meta-model

Osnovni koncept šeme baze podataka je tabela. Tabela, u ovom slučaju reprezentuje pojam šeme relacije. U meta-modelu je predstavljena klasom **Table.** Svaka tabela poseduje svoj skup obeležja i skup ograničenja. Za svaku tabelu mora se definisati njen naziv (**Name**).

Obeležja se, u ovoj terminologiji, nazivaju kolonama koje su u meta-modelu predstavljene klasom **Column**. Za svaku kolonu se pored naziva, definiše tip podatka, maksimalna dužina i preciznost podatka, inicijalna vrednost i dozvola ili zabrana nula vrednosti. Pridruživanje tipa podatka koloni tabele reprezentovano je u meta-modelu vezom **ColumnDataType**, dok su ostale osobine predstavljene sledećim atributima klase **Column**: **Length**, **Precision**, **Default** i **Nullable**, respektivno.

Sva ograničenja relacione šeme baze podataka, predstavljena apstraktnom klasom ***Constraints***, u RSUBP-ovima tehnički se vezuju za odgovarajuću tabelu, bez obzira da li je reč o unutarrelacionom ili međurelacionom ograničenju. Za svaku tabelu je moguće definisati sledeće tipove ograničenja:

- ograničenje primarnog ključa, pri čemu se specificira niz kolona koje primarni ključ sadrži,
- ograničenje jedinstvene vrednosti, pri čemu se specificira niz kolona za koje se zahteva da, ako su sve vrednosti datih kolona različite od nula vrednosti, tada kombinacija vrednosti bude jedinstvena,
- ograničenje stranog ključa, pri čemu se specificira:
  - niz kolona koje strani ključ sadrži,
  - referencirana tabela u kojoj se nalazi odgovarajući primarni ključ ili jedinstvena vrednost,
  - ključ ili jedinstvena vrednost referencirane tabele,
  - aktivnost, koja se sprovodi u slučaju narušavanja referencijalnog integriteta pri brisanju, ili modifikaciji podataka,
  - tip referenciranja,
  - specifikacija trenutka provere ograničenja i
- ograničenje torke, pri čemu se navodi logički izraz koji mora biti zadovoljen.

U meta-modelu ograničenja primarnog ključa i jedinstvene vrednosti predstavljena su klasama: **PrimaryKeyCon** i **UniqueCon**. Vezom **PKandUQColumns** specificiraju se kolone koje pripadaju odgovarajućem ograničenju. Ukoliko je za sve kolone ograničenja jedinstvene vrednosti specificirana zabrana nula vrednosti, tada se ovo ograničenje može kvalifikovati kao ograničenje ekvivalentnog ključa.

Ograničenje stranog ključa predstavljeno je klasom **ForeignKey**. Veza **TableFKs** reprezentuje skup svih ograničenja stranog ključa koje jedna tabela ima, dok veza **FKTable** referencira tabelu koja se nalazi na levoj strani ograničenja stranog ključa tj. referencirajuću tabelu. Veza **RefersTo** upućuje na tebelu koja se nalazi sa desne strane ograničenja stranog ključa, referenciranu tabelu, dok veza **RHS_Key** specificira njen ključ. U RSUBP-ovima moguće je da sa desne strane ograničenja stranog ključa bude primarni ključ referencirane tabele, ali i skup kolona koje pripadaju ograničenju jedinstvene vrednosti. Vezom **LHS_Attr** definiše se niz kolona koje se nalaze na levoj strani ograničenja stranog ključa, odnosno niz kolona koje sadrži strani ključ.

Aktivnost koja se sprovodi u slučaju narušavanja ograničenja stranog ključa, pri brisanju i modifikaciji vrednosti postojećih torki u referenciranoj tabeli, specificira se tako što se atributima **DeleteActionRHS** i **UpdateActionRHS** dodeljuje jedna vrednost, iz skupa vrednosti enumeracije **Action**, Moguće vrednosti su:

- sprečavanje (*Restrict* ili *No Action*),
- svođenje na nula vrednost (*Set Null*),
- svođenje na predefinisanu vrednost (*Set Default*) i
- kaskadna aktivnost (*Cascade*).

Pri izvođenju operacije brisanja, oba SUBP-a korišćena za verifikaciju rezultata nastalih u okviru ove doktorske teze, Oracle Server i MS SQL Server, podržavaju pri deklarativnoj specifikaciji ograničenja aktivnosti: *No Action*, *Set Null* i *Cascade* u najnovijim verzijama. MS SQL Server podržava i aktivnost *Set Default*. U starijim verzijama, Oracle Server 9i podržavao je isti skup aktivnosti kao i ranije, dok je MS SQL Server 2000 podržavao samo *No Action* i *Cascade*.

Pri operaciji modifikacije MS SQL Server 2000 podržavao je, pri deklarativnoj specifikaciji ograničenja, samo *No Action* i *Cascade*, dok u najnovijoj verziji MS SQL Server 2012 podržava sve navedene aktivnosti: *No Action*, *Set Null*, *Set Default* i *Cascade*. Oracle Server u starijim kao i u najnovijoj verziji podržava samo *No Action*.

Treći atribut klase **ForeignKey** je **Match**, koji se koristi za specifikaciju tipa referenciranja. Tip referenciranja može biti: podrazumevani (DEFAULT), parcijalni (PARTIAL) ili potpuni (FULL). Mogući tipovi referenciranja predstavljeni su u meta-modelu enumeracijom **ReferencingType**. Detaljno objašnjenje tipova referenciranja biće dato u odeljku 6.3.

Klasi **ForeignKey** pridružen je još jedan atribut pod nazivom **InverseReferentialIntegrityCon** kojim se specificira postojanje ograničenja inverznog referencijalnog integriteta za odgovarajuće ograničenje stranog ključa. Ova osobina ne nalazi se u meta-podacima pročitanim iz rečnika podataka SUBP-a, već predstavlja dopunu semantike na osnovu podataka. Postupak dobijanja informacija o postojanju inverznog referencijalnog integriteta opisan je u odeljku 5.5.

Za sve tipove ograničenje može se definisati specifikacija trenutka provere ograničenja. Trenutak provere ograničenja specificira se tako, što se atributima **Deferrable** i **Deferred** (atributi apstraktne klase ***Constraints***) dodeljuje jedna vrednost, iz skupa vrednosti, koje sadrže enumeracije **DeferrableAct** i **DeferredAct**:

- DEFFERABLE (trenutak provere ograničenja se može odložiti za kraj transakcije);
- NOT DEFFERABLE (trenutak provere ograničenja se ne može odložiti, ograničenje se proverava odmah, tokom realizacije zahtevane operacije ažuriranja);
- INITIALLY IMMEDIATE (inicijalno, ograničenje se proverava odmah);
- INITIALLY DEFERRED (inicijalno, trenutak provere ograničenja se odlaže za kraj transakcije).

Validne kombinacije su:

- INITIALLY IMMEDIATE DEFERRABLE,
- INITIALLY IMMEDIATE NOT DEFERRABLE i
- INITIALLY DEFERRED [DEFERRABLE].

Provera dozvoljenih vrednosti za atribute **Deferrable** i **Deferred** obezbeđena je OCL ograničenjem prikazanim na listingu 6.1.

```
invariant DeferrabilityCombination:

if InitiallyDefer = ConInitiallyDefer::INITIALLY_DEFERRED and
        Deferrability = ConDeferrability::NOT_DEFFERABLE
 then false else true endif;
```

Listing 6.1. OCL ograničenje za proveru trenutka provere ograničenja šeme baze podataka

```
invariant UniquenessConName:

Constraint.allInstances()->forAll(e1 : Constraint, e2 : Constraint |
        e1.name = e2.name implies e1 = e2);
```

Listing 6.2. OCL ograničenje za proveru jedinstvenog naziva ograničenja šeme baze podataka

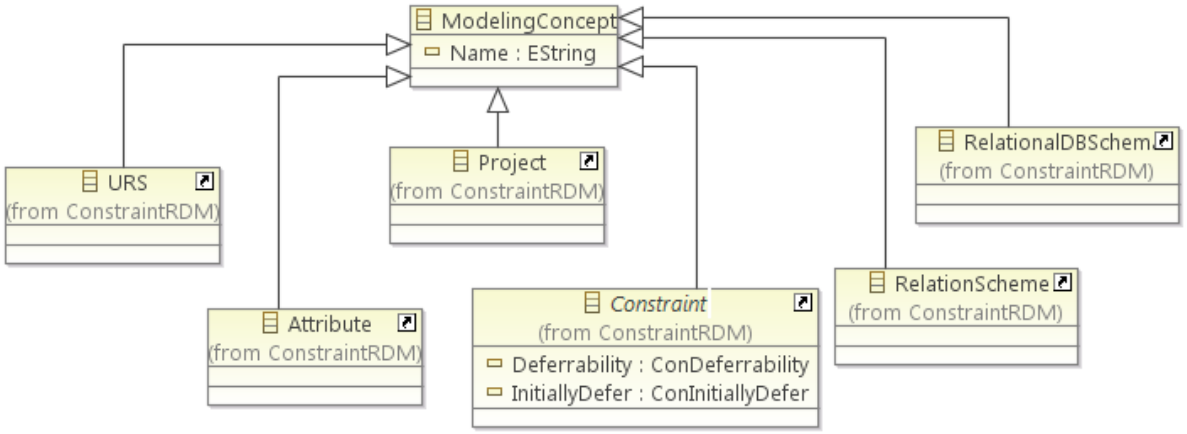
Ograničenje torke je u meta-modelu predstavljeno skupom *Check* ograničenja. Pojedinačno ograničenje predstavljeno je klasom **CheckCon**. Logički izraz svakog *Check* ograničenja definiše se u okviru **CheckCondition** atributa ove klase.

Za svako ograničenje važi da se u skupu svih ograničenja jedinstveno identifikuje svojim nazivom. Na listingu 6.2 prikazano je OCL ograničenje koje obezbeđuje ovo svojstvo.

6.3. Generički meta-model

U ovom odeljku biće prikazan Generički meta-model relacione šeme baze podataka koji je u skladu sa teoretskim definicijama relacionog modela podataka. Osnovni koncepti relacionog modela zasnovani su na konceptima datim, između ostalih, u [Mogin04], [Date06] i [Elmas11].

Predloženi meta-model je kompleksan i veliki, zbog čega su njegove celine organizovane pomoću paketa. Osnovni koncepti za modelovanje, generalizovani apstraktnom klasom ***ModelingConcept***, od kojih se meta-model sastoji su: atribut, ograničenje, relaciona šema, šema univerzalne relacije i relaciona šema baze podataka. Ovi koncepti prikazani su na slici 6.3 klasama: **Attribute**, ***Constraint***, **RelationScheme**, **URS** i **RelationalDBSchema**, respektivno. Apstraktna klasa ***ModelingConcept*** ima atribut **Name**, koji nasleđuju svi ostale klase.

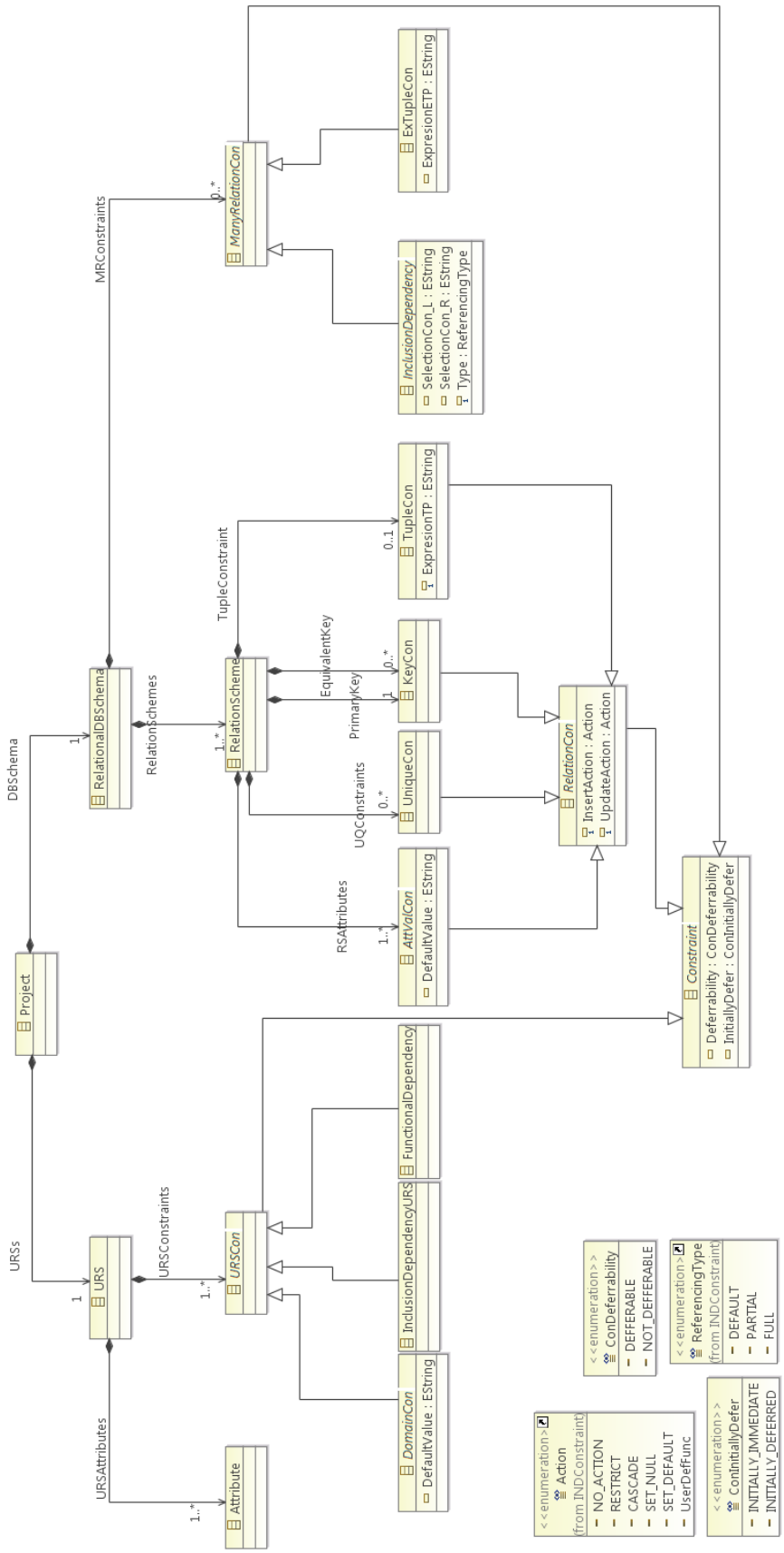


Slika 6.3. Koncepti za modelovanje relacione šeme baze podataka

Klasa **Project** je korenska klasa, koja je uvedena zbog XML struktre, koju Ecore zahteva. Ova klasa reprezentuje koncept projekta koji sadrži sve elemente potrebne za projektovanje relacione šeme baze podataka. Na slici 6.4 prikazan je meta-model koncepta projekta koji se sastoji od šeme univerzalne relacije (URS) i relacione šeme baze podataka. Paket sa meta-modelom URS-e biće prezentovan u tački 6.3.1, a paket sa meta-modelom relacione šeme baze podataka u tački 6.3.2.

Metodologija projektovanja baza podataka zasnovana na pretpostavci o postojanju URS-e ekstrahuje relacione šeme baze podataka iz:

- skupa atributa sa pridruženim domenima,
- skupa funkcionalnih zavisnosti i
- skupa netrivijalnih zavisnosti sadržavanja.



Slika 6.4. Meta-model koncepta projekta

Kao što je prikazano na slici 6.4, ograničenja baze podataka, generalizovana apstraktnom klasom ***Constraint***, mogu se specijalizovati kao:

- ograničenja URS-e (***URSCon***),
- relaciona ograničenja (***RelationCon***) i
- višerelaciona ograničenja (***ManyRealtionCon***).

Apstraktna klasa ***Constraint*** ima dva artibuta **Deferrability** i **InitiallyDefer** koja služe za specifikaciju trenutka provere ograničenja. Validne kombinacije su iste kao kod RSUBP meta-modela, kao i OCL ograničenje kojim se vrši provera ove specifikacije. OCL ograničenje prikazano je na listingu 6.1.

Detaljan opis svih koncepata i njihovi meta-modeli biće prezentovani u tekstu koji sledi.

6.3.1. Meta-model koncepta URS

URS-a je uređeni par (*R*, *UC*), gde je *R* skup koji sadrži sve atribute posmatranog realnog sistema sa pridruženim ograničenjima domena, dok je *UC* skup URS ograničenja koji obuhvata: skup funkcionalnih zavisnosti i netrivijalnih zavisnosti sadržavanja. *R* se naziva univerzalnim skupom atributa (UAS). Univerzalna relacija *u*(UAS) je relacija nad UAS.

Posledica pretpostavke o postojanju URS-e je, da svako obeležje iz UAS mora da ima jedinstvenu ulogu, odnosno da se svako obeležje jedinstveno identifikuje svojim nazivom. Na listingu 6.3 prikazano je OCL ograničenje koje obezbeđuje ovo svojstvo.

```
invariant UniquenessAttrName:

Attribute.allInstances()->forAll(e1 : Attribute, e2 : Attribute |
        e1.Name = e2.Name implies e1 = e2);
```

Listing 6.3. OCL ograničenje za proveru jedinstvenog naziva atributa u okviru UAS

Osnovni koncepti URS-e, prikazani na slici 6.4, predstavljeni su sledećim klasama:

- klasa **Attribute** koja predstavlja koncept atributa i
- klase za tri moguće vrste URS ograničenja (***URSCon***):
  - ***DomainCon*** koja predstavlja ograničenje domena,
  - **FunctionalDependency** koja predstavlja funkcionalne zavisnosti i
  - **InclusionDependencyURS** koja predstavlja netrivijalne zavisnosti sadržavanja.

Na slici 6.5 prikazan je detaljan meta-model URS ograničenja.

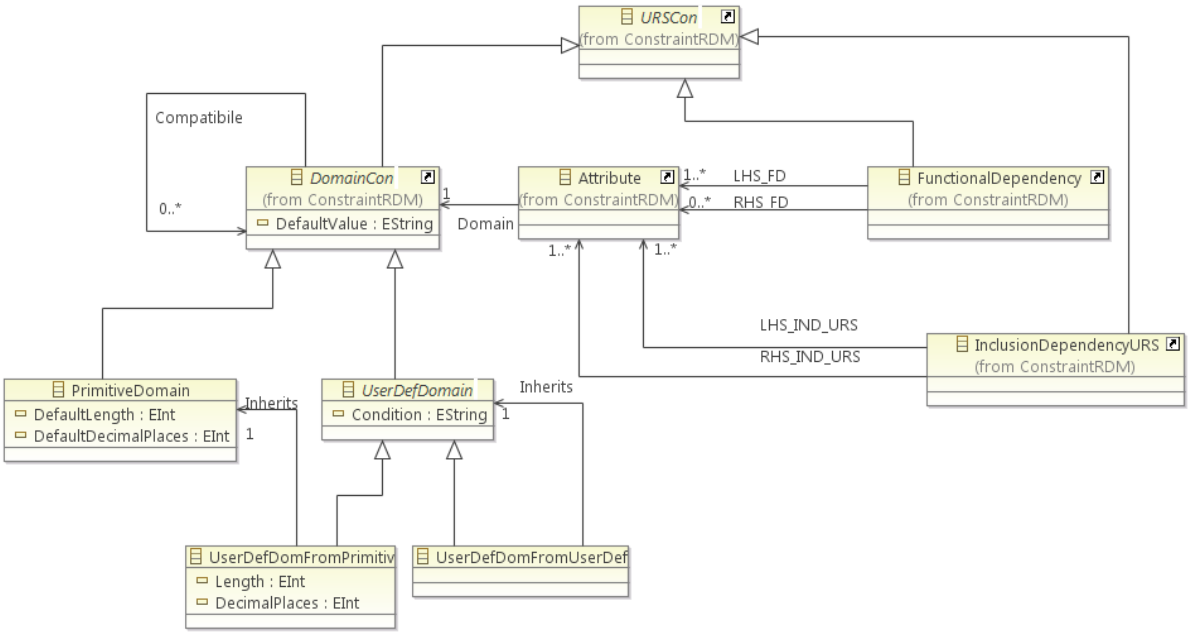
Neka je *R* konačan skup atributa. Za svaki atribut $A \in R$, skup svih mogućih vrednosti predstavljen je domenom za *A*. Domen pridružen atributu *A* se označava sa *Dom*(*A*), a skup mogućih vrednosti atributa *A* (*A*-vrednosti) označen je sa *dom*(*A*). Ograničenje domena ograničva dozvoljene vrednosti unutar nekog domena. Torka *t* nad $R = \{A_1, ..., A_m\}$ je niz vrednosti $(a_1, ..., a_m)$ gde je: $(\forall i \in \{1, \ldots, m\})(a_i \in dom(A_i))$. Relacija nad skupom *R*, označena sa *r*(*R*), je skup torki nad *R*.

Domen (***DomainCon***) može biti:

- primitivni (predefinisani) domen, predstavljen klasom **PrimitiveDomain** ili
- korisnički definisani domen, predstavljen apstraktnom klasom ***UserDefDomain***.

Korisnički definisani domen može da nasledi:

- primitivni domen, predstavljen klasom **UserDefDomainFromPrimitiv** ili
- prethodno definisani korisnički domen, predstavljen klasom **UserDefDomainFromUserDef**.



Slika 6.5. Meta-model koncepta URS ograničenja

Pri definisanju primitivnog domena mogu se zadati osobine dužina i broj cifara iza decimalne tačke, predstavljene atributima **DefaultLength** i **DefaultDecimalPlaces**.

Specifikacija korisnički definisanog domena uključuje i zadavanje uslova (atribut **Condition**). Uslov je regularni izraz koji može dodatno ograničiti skup mogućih vrednosti nekog domena. Za domen koji nasleđuje primitivni domen se, takođe, mogu definisati dužina (atribut **Length**) i broj cifara iza decimalne tačke (atribut **DecimalPlaces**).

Podrazumevana vrednost je osobina koju poseduju svi tipovi domena i predstavljena je atributom **DefaultValue**, apstraktne klase ***DomainCon***. Podrazumevana vrednost pri specifikaciji domena nije obavezna.

Svakom atributu se pridružuje samo jedan domen, što je u meta-modelu reprezentovano vezom **Domain** i njenim kardinalitetom 1.

Funkcionalna zavisnost (FD) je veza koja postoji kada svaka X-vrednost jedinstveno određuje Y-vrednost. Formalno, za dati skup atributa R, funkcionalna zavisnost između skupova atributa X i Y je predstavljena kao $X \rightarrow Y$, čime se specificira da je Y funkcionalno zavisno od X.

Za funkcionalne zavisnosti specificira se skup atributa leve i desne strane funkcionalne zavisnosti, što je predstavljeno u meta-modelu vezama **LHS_FD** i **RHS_FD**, respektivno. Skup atributa na desnoj strani funkcionalne zavisnosti može biti prazan, što se može videti iz minimalnog kardinaliteta nula, veze **RHS_FD**.

Netrivijalna zavisnost sadržavanja je izraz u obliku $[X] \subseteq [Y]$, gde su X i Y neprazni nizovi atributa UAS. Kardinaliteti nizova X i Y moraju biti jednaki (za razliku od kardinaliteta skupova atributa X i Y u FD). Za skupove obeležja X i Y mora da važi da su domen kompatiblni.

Kao i za funkcionalne zavisnosti, za netrivijalne zavisnosti sadržavanja specificira se skup atributa leve i desne strane, što je predstavljeno u meta-modelu vezama **LHS_IND_URS** i **RHS_IND_URS**, respektivno. Za razliku od funkcionalnih zavisnosti, skupovi atributa i leve i desne strane zavisnosti sadržavanja (**InclusionDependencyURS)** moraju biti neprazani.

Kaže se da univerzalna relacija u zadovoljava netrivijalnu zavisnost sadržavanja ako za svaku torku $t \in u$ postoji najmanje jedna torka $s \in u$ takva da $t[X]=s[Y]$, gde $t[X]$ predstavlja projekciju torke t na X.

Postoje različiti pristupi projektovanju šeme baze podataka. Jedan od njih, ovde prihvaćen, zasnovan je na pretpostavci o postojanju URS-e i ugrađen u alat IIS*Case. Korišćenjem skupa FD, URS-a se dekomponuje na skup šema relacija, što rezultuje relacionom šemom baze podataka.

6.3.2. Meta-model koncepta relacione šeme baze podataka

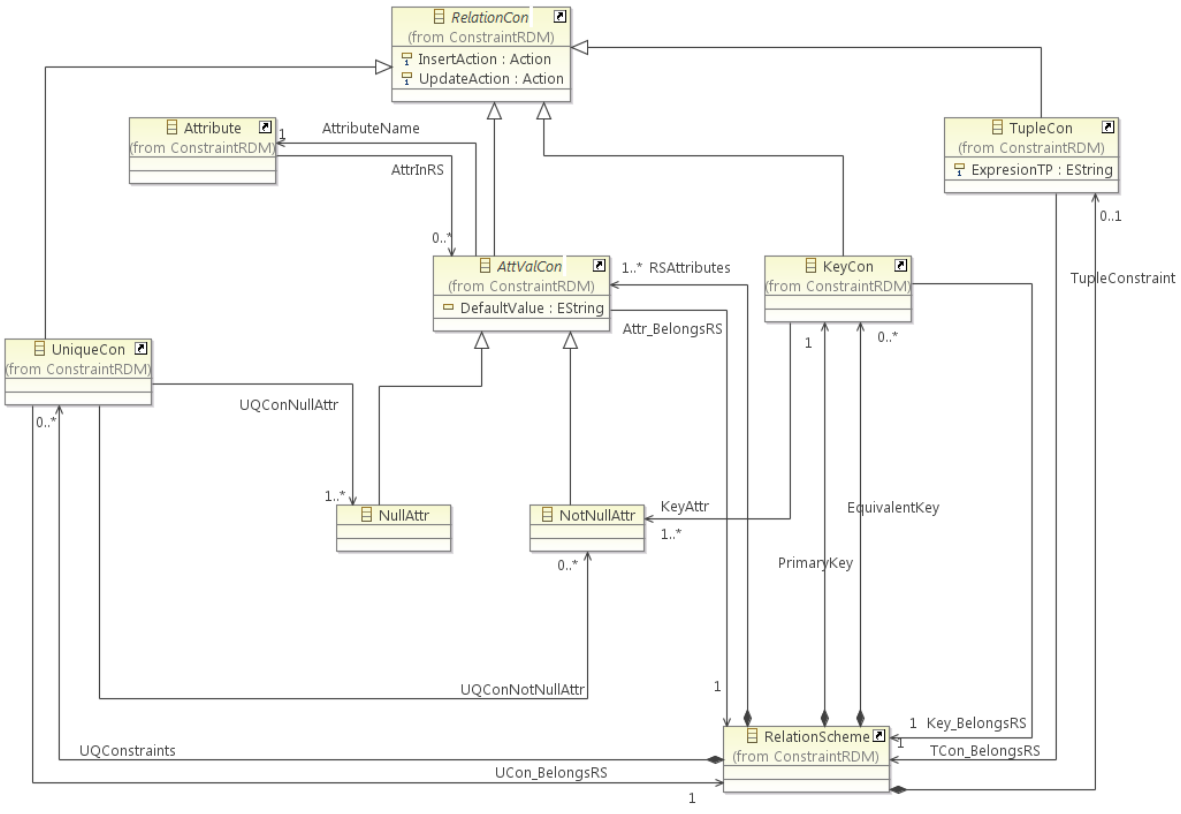
Formalno, relaciona šema baze podatka je par (S, I), gde je S konačan skup šema relacija, a I konačan skup međurelacionih ograničenja. Relaciona šema baze podatka je u meta-modelu (slika 6.4) predstavljena klasom **RelationalDBSchema**, šema relacije klasom **RelationScheme**, dok su međurelaciona ograničenja predstavljena apstraktnom klasom ***ManyRelationCon***. Za relacionu šemu baze podataka može se zadati i naziv, zadavanjem vrednosti atributa **Name**.

Šema relacije je imenovani par $N(R, C)$ gde je N naziv šeme relacije (atribut **Name**), R konačan skup atributa iz UAS, a C konačan skup relacionih ograničenja. R je u meta-modelu predstavljen vezom **RSAttributes**, a C apstraktnom klasom ***RelationCon***. Skup relacionih ograničenja C sadrži:

- ograničenje vrednosti atributa sa ograničenjem nula vrednosti,
- ograničenje ključa,
- ograničenje jedinstvenosti i
- ograničenje torke,

zbog čega se apstraktna klasa ***RelationCon*** specijalizuje u klase: ***AttValCon***, **KeyCon, UniqueCon** i **TupleCon**, respektivno.

Na slici 6.6 prikazan je detaljan prikaz meta-modela relacionih ograničenja.



Slika 6.6. Meta-model koncepata relacionih ograničenja

Za razliku od RSUBP meta-modela, opisanog u poglavlju 6.2, u ovom meta-modelu su razdvojeni koncepti primarnog ključa, ekvivalentnog ključa i ograničenja jedinstvenosti. Klasa **KeyCon** predstavlja skup ekvivalentnih ključeva od kojih se jedan bira za primarni. Svaka šema relacije mora da poseduje tačno jedan primarni ključ. Primarni ključ je predstavljen vezom **PrimaryKey**.

Za svako obeležje šeme relacije zadaje se ograničenje vrednosti obeležja, koje podrazumeva pridruživanje domena i specificiranje zabrane ili dozvole nula vrednosti, i njegova predefinisana vrednost. Ograničenje vrednosti obeležja je obavezna komponenta, dok predefinisana vrednost ne mora biti zadata.

Ograničenje vrednosti obeležja predstavljeno je apstraktnom klasom ***AttValCon***, kojom se specificira da neki atribut iz UAS, pripada nekoj šemi relacije. Svakom atributu je prethodno definisan domen. Apstraktna klasa ***AttValCon*** se specijalizuje u klasu **NullAttr**, koja predstavlja koncept atributa koji smeju da imaju nula vrednost, i klasu **NotNullAttr** koja predstavlja atribute koji ne smeju da imaju nula vrednost. Na ovaj način, obezbeđena je specifikacija ograničenja nula vrednosti obeležja.

Osim toga, kako su, po definiciji, za sva obeležja ključa zabranjene nula vrednosti, razdvajanje atributa u dve klase **NullAttr** i **NotNullAttr**, takođe, obezbeđuje da se diferenciraju atributi koji mogu ulaziti u neki od ekvivalentnih ključeva. Specifikacija obeležja koja ulaze u neki od ključeva šeme relacije kreira se korišćenjem **KeyAttr** veze, koja povezuje ključ samo sa obeležjima sa zabranjenom nula vrednošću.

U ograničenje jedinstvenosti ulaze atributi i sa zabranjenom i dozvoljenom nula vrednošću. Ovo ograničenje mora imati barem jedan atribut iz skupa obeležja sa dozvoljenom nula vrednošću. Ukoliko to nije slučaj, radi se o ekvivalentnom ključu. Specifikacija atributa ograničenja jedinstvenosti u meta-modelu vrši se putem veza ka obe grupe atributa (**NullAttr** i **NotNullAttr**) **UQConNullAttr** i **UQConNotNullAttr**, respektivno.

Podrazumevana vrednost atributa u okviru šeme relacije kojoj pripada, predstavljena je atributom **DefaultValue**.

Ograničenje torke izražava ograničenja na moguće vrednosti unutar jedne torke. Specifikacija ograničenja vrši se tako što se zadaje logički uslov koji svaka torka šeme relacije mora da zadovolji. Ovo ograničenje u meta-modelu predstavljeno je klasom **TupleCon**. Logički izraz za proveru ograničenja torke, definiše se u okviru **ExpresionTP** atributa ove klase. Šema relacije može imati jedno ograničenje torke.

Za sva relaciona ograničenja kritične operacije koje mogu dovesti do narušavanja ograničenja su: upis nove torke u relaciju i modifikacija vrednosti postojeće torke u relaciji. Za svaku kritičnu opraciju mora biti definisana aktivnost očuvanja konzistentnosti baze podataka, koja se sprovodi u slučaju pokušaja narušavanja ograničenja. Specifikacija mogućih aktivnosti u meta-modelu vrši se zadavanjem vrednosti atributa **InsertAction** i **UpdateAction** apstraktne klase ***RelationCon***. Vrednost koja se zadaje mora biti iz skupa dozvoljenih vrednosti za ovu vrstu ograničenja. Moguće aktivnosti su: *NoAction* ili *Restrict*, *SetNull*, *SetDefault* i *UserDef*. *UserDef* je aktivnost koju definiše korisnik. Akcija *Cascade* u unutarrelacionim ograničenjima nije dozvoljena. Ova zabrana predstavljena je OCL ograničenjem prikazanim na listingu 6.4.

```
invariant NoCascade:
InsertAction <> Action::CASCADE and UpdateAction <> Action::CASCADE;
```

Listing 6.4. OCL ograničenje za kontrolu dozvoljenih akcija relacionih ograničenja

6.3.3. Meta-model koncepta višerelacionih ograničanja

Skup višerelacionih ograničenja I, u meta-modelu predstavljen apstraktnom klasom ***ManyRelationCon*** (slika 6.4), sadrži:

- ograničenja zavisnosti sadržavanja, predstavljena apstraktnom klasom ***InclusionDependency*** i
- prošireno ograničenje torke, predstavljeno u meta-modelu klasom **ExTupleCon**.

Na slici 6.7 prikazan je meta-model koncepta zavisnosti sadržavanja.

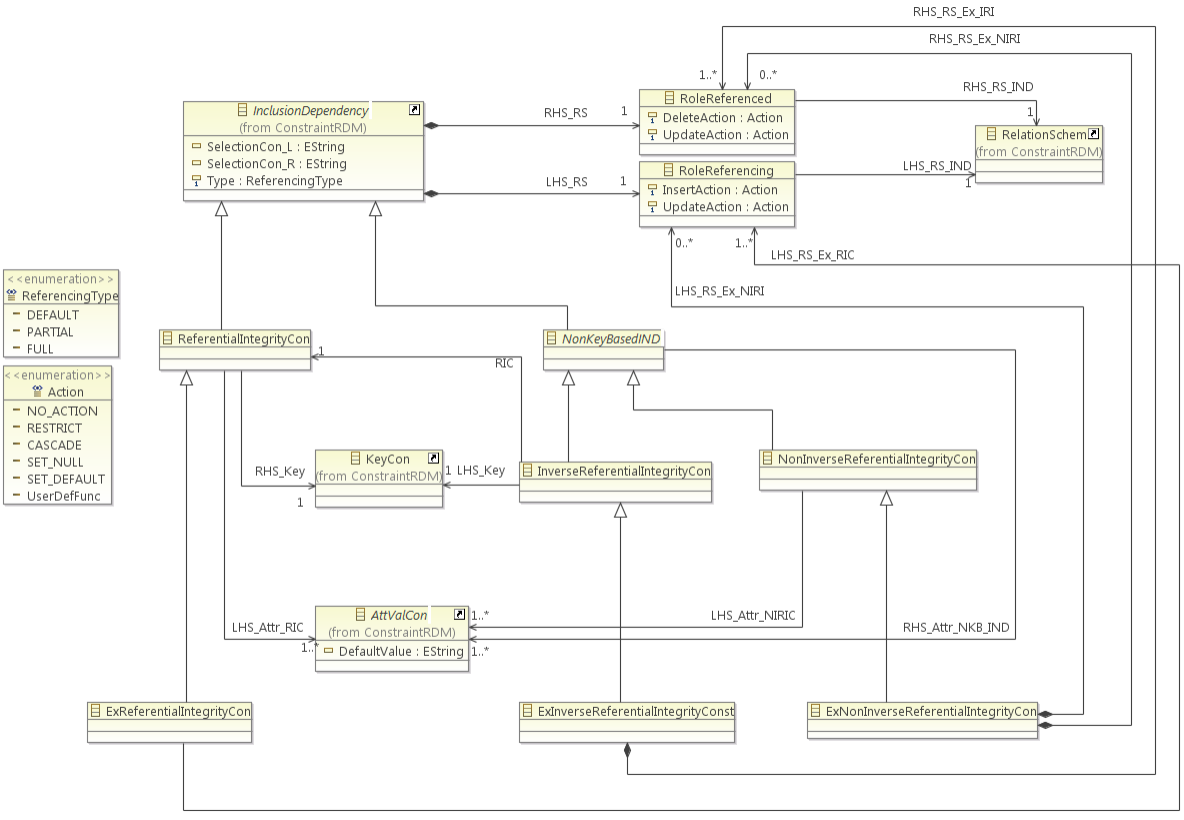
Neka su $N_l(R_l, C_l)$ i $N_r(R_r, C_r)$ dve šeme relacije, gde su N_l i N_r njihovi nazivi, R_l i R_r, odgovarajući skupovi atributa, a C_l i C_r odgovarajući skupovi ograničenja šema relacija. Zavisnost sadržavanja je izraz u obliku $N_l[LHS] \subseteq N_r[RHS]$, gde su LHS i RHS neprazni nizovi obeležja iz R_l i R_r respektivno. LHS i RHS moraju biti domenski kompatibilni. U odnosu na operator sadržavanja ($\subseteq$) $N_l[LHS]$ naziva se levom, a $N_r[RHS]$ desnom stranom zavisnosti sadržavanja. Korišćeni su indeksi l i r, i nazivi nizova atributa LHS i RHS, u cilju označavanja leve i desne strane zavisnosti sadržavanja, respektivno. Za definisanje pravila za validaciju zavisnosti sadržavanja korišćena je sledeća notacija:

(i) relacija $r(N_l)$ je skup torki $u(R_l)$ (ili samo u) koje zadovoljavaju sva ograničenja iz skupa ograničenja C_l;

(ii) relacija $r(N_r)$ je skup torki $v(R_r)$ (ili samo v) koje zadovoljavaju sva ograničenja iz skupa ograničenja C_r;

(iii) $u[LHS]$ i $v[RHS]$ predstavljaju projekcije torki u i v na skup atributa LHS i RHS, respektivno; i

(iv) $r(N_l)$ se naziva referencirajućom relacijom, dok se $r(N_r)$ naziva referenciranom relacijom.



Slika 6.7. Meta-model koncepta zavisnosti sadržavanja

Zavisnost sadržavanja je zadovoljena u pojavi nad šemom baze podataka, ako važi

$$(\forall u \in r(N_l))(\exists v \in r(N_r))(u[LHS] = v[RHS]).$$

Neformalno, baza podataka zadovoljava zavisnost sadržavanja ako je skup vrednosti iz *LHS* u referencirajućoj relaciji $r(N_l)$ podskup skupa vrednosti iz *RHS* u referenciranoj relaciji $r(N_r)$.

Postoje dve osnovne vrste zavisnosti sadržavanja:

- zavisnosti sadržavanja zasnovane na ključu (*key-based INDs*) i
- zavisnosti sadržavanja koje nisu zasnovane na ključu (*non-key-based INDs*).

Zavisnost sadržavanja je *key-based* ako je *RHS* ključ šeme relacije N_r. U suprotnom, zavisnosti sadržavanja je *non-key-based*. Veoma često se *key-based IND* naziva ograničenjem referencijalng integriteta (RIC). Zbog ove podele apstraktna klasa ***InclusionDependency*** prvo se specijalizuje na klase **ReferentialIntegrityCon** i ***NonKeyBased IND***.

Apstraktna klasa ***NonKeyBased IND*** se dalje specijalizuje na klase **InverseReferentialIntegrityCon** i **NonInverseReferentialIntegrityCon** koje u meta-modelu predstavljaju ograničenje inverznog referencijalnog integritet (IRIC) i druge zavisnosti sadržavanja (koje nisu ni RIC ni IRIC), respektivno.

Ograničenje inverznog referencijalnog integriteta je *Non-key-based IND* gde je *LHS* ključ šeme relacije N_l, i gde je istovremeno specificirano i ograničenje referencijalnog integriteta, $N_r[RHS] \subseteq N_l[LHS]$. Detaljno objašnjenje ovog ograničenja može se naći u [Aleks13]. Postojanje ograničenja inverznog referencijalnog integriteta šeme baze podataka uslovljeno je postojanjem odgovarajućeg ograničenja referencijalnog integriteta, što je obezbeđeno OCL ograničenjem prikazanim na listingu 6.5.

```
invariant CheckRIC:
LHS_Key = RIC.RHS_Key and RHS_Attr_NKB_IND = RIC.LHS_Attr_RIC;
```

Listing 6.5. OCL ograničenje za proveru postojanja odgovarajućeg RIC za IRIC

Zavisnost sadržavanja može biti osnovna ili proširena. Tip zavisnosti sadržavanja se naziva proširenim ukoliko najmanje na jednoj strani zavisnosti sadržavanja postoji obaveza prirodnog spajanja relacija nad dve ili više šema relacija. Prethodno opisani tipovi zavisnosti sadržavanja spadaju u osnovne, dok su prošireni predstavljeni klasama:

- **ExReferentialIntegrityCon**, za prošireni RIC,
- **ExInverseReferentialIntegrityConst**, za prošireni IRIC i
- **ExNonInverseReferentialIntegrityCon**, za ostale proširene zavisnosti sadržavanja.

Zavisnost sadržavanja može biti i selektivna. Za zavisnost sadržavanja kaže se da je selektivna ako postoji uslov selekcije najmanje na jednoj strani. Selektivna zavisnost sadržavanja definiše se pomoću atributa **SelectionCon_L**, **SelectionCon_R** klase ***InclusionDependency***. Prvi se koristi za specifikaciju uslova selekcije na levoj, a drugi na desnoj strani zavisnost sadržavanja. Ako je barem jedan od uslova speficiran znači da se radi o selektivnom ograničenju zavisnosti sadržavanja. Selektivna zavisnost sadržavanja, takođe, može biti osnovna i proširena.

Svim prethodno opisanim konceptima mogu se modelovati sledeći tipovi zavisnosti sadržavanja:

- ograničenje referencijalnog integriteta:
  proširenо, selektivno, selektivno i proširenо,
- ograničenje inverznog referencijalnog integriteta:
  proširenо, selektivno, selektivno i proširenо i
- zavisnost sadržavanja koja nije ni RIC ni IRIC:
  proširena, selektivna, selektivna i proširena.

Apstraktna klasa ***InclusionDependency*** ima još jedan atribut **Type**, koji se koristi za specifikaciju tipa referenciranja koji može biti: podrazumevani, parcijalni ili potpuni. Mogući tipovi referenciranja ograničenja zavisnosti sadržvanja predstavljeni su u meta-modelu enumeracijom **ReferencingType**.

Podrazumevani tip referenciranja je zadovoljen ako za svaku torku *u* iz $r(N_l)$, važi

- svako obeležje iz *LHS* ima nenula vrednost i postoji jedna torka *v* u relaciji $r(N_r)$ sa istom *RHS* vrednošću, ili
- torka *u* ima, bar za jedno obeležje iz *LHS*, nula vrednost.

Parcijalni tip referenciranja je zadovoljen ako, za svaku torku *u* iz $r(N_l)$, važi da postoji bar jedna torka *v* u relaciji $r(N_r)$ takva, da je svaka nenula vrednost u *LHS* jednaka vrednosti odgovarajućeg, domen kompatibilnog, obeležja u *RHS*.

Potpuni tip referenciranja, u bazi podataka, zadovoljen je, ako, za svaku torku *u* iz $r(N_l)$, važi:

- svako obeležje iz *LHS* ima nenula vrednost i postoji jedna torka *v* u relaciji $r(N_r)$ sa istom *RHS* vrednošću, ili
- svako obeležje iz *LHS* ima nula vrednost. [Mogin04]

Ukoliko je skup atributa *LHS* jednočlan, jedini mogući tip je podrazumevani. Isto važi i za slučaj kada je *LHS* višečlan, a svi atributi imaju zabranjene nula vrednosti. U slučaju da barem jedan atribut iz skupa *LHS* ima dozvoljenu nula vrednost moguće je osim podrazumevanog izabrati i parcijalni tip. Ako svi atributi iz *LHS* imaju dozvoljene nula vrednosti, ograničenje zavisnosti sadržavanja može se proglasiti i potpunim.

Šeme relacija specificirane u ograničenju zavisnosti sadržavanja mogu imati dve uloge: referencirajuću, ako su na levoj strani ili referenciranu, ako su na desnoj strani ograničenja zavisnosti sadržavanja. Uloge šeme relacije u ograničenju predstavljene su u meta-modelu klasama **RoleReferencing** za referencirajuću i **RoleReferenced** za referenciranu šemu relacije. Za ulogu referencirane šeme relacije, kritične operacije koje mogu dovesti do narušavanja ograničenja su: brisanje i modifikacija vrednosti postojeće torke u relaciji, dok su za referencirajuću, kritične operacije: upis nove torke u relaciju i modifikacija vrednosti postojeće torke u relaciji. Klase **RoleReferenced** i **RoleReferencing** sadrže atribute **DeleteAction**, **UpdateAction**, i **InsertAction, UpdateAction**, respektivno, namenjene za specifikaciju mogućih aktivnosti očuvanja konzistentnosti baze podatka pri pokušaju narušavanja ograničenja putem kritičnih operacija. Na listingu 6.6. prikazano je OCL ograničenje kojim se zabranjuje specifikacija kaskadne aktivnosti za upis nove torke u relaciju i modifikaciju vrednosti postojeće torke u relaciji, za referencirajuću relaciju, za sve tipove ograničenja zavisnosti sadržavanja.

```
invariant NoCascade:
InsertAction <> Action::CASCADE and
        UpdateAction <> Action::CASCADE;
```

Listing 6.6. OCL ograničenje za zabranu kaskadne aktivnosti u referencirajućoj šemi relacije

Za sve tipove ograničenje može se definisati specifikacija trenutka provere ograničenja. Trenutak provere ograničenja specificira se tako, što se atributima **Deferrability** i **InitiallyDefer** (atributi apstraktne klase ***Constraint***) dodeljuje jedna vrednost, iz skupa vrednosti, koje sadrže enumeracije **ConDeferrability** i **ConInitiallyDefer**. Pravila za validne kombinacije ista su kao kod prethodno opsanog meta-modela iz odeljka 6.2. Provera dozvoljenih vrednosti za atribute **Deferrability** i **InitiallyDefer** obezbeđena je OCL ograničenjem koje je identično ograničenju prikazanom na listingu 6.1.

Na listingu 6.7. prikazano je OCL ograničenje kojim se vrši kontrola pri specifikaciji aktivnosti za kritične operacije u IRIC. U referencirajućoj relaciji dozvoljena aktivnost za obe kritične operacije, upisa nove torke u relaciju i modifikaciju vrednosti postojeće torke u relaciji, dozvoljena je samo aktivnot *No action* ili *Restrict*, dok je u referenciranoj relaciji za obe kritične operacije, brisanje i modifikaciju vrednosti postojeće torke u relaciji, dozvoljena još i kaskadna aktivnost.

```
invariant IRIC_LHS_ActionInsert:
LHS_RS.InsertAction = Action::NO_ACTION or LHS_RS.InsertAction =
        Action::RESTRICT;
invariant IRIC_LHS_ActionUpdate:
LHS_RS.UpdateAction = Action::NO_ACTION or LHS_RS.UpdateAction =
        Action::RESTRICT;
invariant IRIC_RHS_ActionDelete:
RHS_RS.DeleteAction = Action::NO_ACTION or RHS_RS.DeleteAction =
        Action::RESTRICT or RHS_RS.DeleteAction = Action::CASCADE;
invariant IRIC_RHS_ActionUpdate:
RHS_RS.UpdateAction = Action::NO_ACTION or RHS_RS.UpdateAction =
        Action::RESTRICT or RHS_RS.UpdateAction = Action::CASCADE;
```

Listing 6.7. OCL ograničenje za kontorlu mogućih aktivnosti u IRIC

Za zavisnost sadržavanja mora se specificirati najmanje jedna šema relacije u svakoj od ove dve uloge. Ako je zavisnost sadržavanja ograničenje proširenog referencijalnog integriteta, onda najmanje još jedna šema relacije mora biti specificirana kao referencirajuća. U slučaju ograničenja proširenog inverznog referencijalnog integriteta najmanje još jedna šema relacije mora biti specificirana sa ulogom referencirane. Za ostale proširene zavisnosti sadržavanja najmanje još jedna šema relacije mora biti specificirana sa ulogom referencirajuće ili referencirane. Na listingu 6.8 prikazano je OCL ograničenje koje obezbeđuje proveru ovog uslova.

```
invariant MandatoryOneExRS:
not (LHS_RS_Ex_NIRI->isEmpty() and self.RHS_RS_Ex_NIRI->isEmpty());
```

Listing 6.8. OCL ograničenje za proveru broja šema relacija u ograničenju proširene zavisnosti sadržavanja koje nije RIC ni IRIC

Za svaki tip ograničenja zavisnosti sadržavanja, predstavljen na slici 6.7, mora biti specificiran niz atributa na levoj i desnoj strani ograničenja. Za RIC se na desnoj strani ograničenja specificira ključ referencirane šeme relacije, dok se za IRIC ključ specificira na levoj strani ograničenja. U meta-modelu ove specifikacije predstavljene su vezama **RHS_Key** i **LHS_Key** od klasa **ReferentialIntegrityCon** i **InverseReferentialIntegrityCon** ka klasi **KeyCon**, respektivno.

Sa leve strane RIC-a specificira se niz obeležja koja sadrži strani ključ, što je predstavljeno vezom **LHS_Attr_RIC**.

Za sva ograničenja zavisnosti sadržavanja koja spadaju u *non-key-based*, specifikacija obeležja na desnoj strani predstavljena je vezom **RHS_Attr_NKB_IND**, dok je specifikacija

obeležja na levoj strani za zavisnosti sadržavanja koje nisu RIC ni IRIC predstavljena vezom **LHS_Attr_NIRIC**.

Pri zadavanju atributa leve i desne strane ograničenja zavisnosti sadržavanja potrebno je proveravati da li svi atributi iz nizova atributa pripadaju skupovima atributa odgovarajućih šema relacija koje se nalaze na levoj i desnoj strani ograničenja.

Na listingu 6.9 prikazano je OCL ograničenje koje obezbeđuje proveru da li sva obeležja iz niza obeležja na levoj strani RIC-a pripadaju skupu obeležja šeme relacije koja je specificirana kao referencirajuća u RIC-u. Pored toga, obezbeđuje i proveru da li je ključ specificiran na desnoj strani RIC-a takođe ključ šeme relacije specificirane kao referencirana u RIC.

```
invariant CheckLHSAttributeRIC:

LHS_Attr_RIC->forAll(a: ConstraintRDM::AttValCon |
        a.Attr_BelongsRS = self.LHS_RS.LHS_RS_IND);

invariant CheckRHSAttributeRIC:

RHS_Key->forAll(a: ConstraintRDM::KeyCon |
        a.Key_BelongsRS = self.RHS_RS.RHS_RS_IND);
```

Listing 6.9. OCL ograničenje za proveru atributa u RIC

OCL ograničenje za prošireni RIC prikazano je na listingu 6.10. Razlika u odnosu na prethodno opisano OCL ograničenje je u činjenici da obeležja iz niza obeležja specificiranih na levoj strani proširenog RIC-a moraju pripadati najmanje jednoj šemi relacije koja je specificirana kao referencirajuća u tom ograničenju.

```
invariant CheckLHSAttributeExRIC:

LHS_Attr_RIC->forAll(a: ConstraintRDM::AttValCon |
        a.Attr_BelongsRS = self.LHS_RS.LHS_RS_IND or
        a.Attr_BelongsRS = LHS_RS_Ex_RIC.LHS_RS_IND);

invariant CheckRHSAttributeExRIC:

RHS_Key->forAll(a: ConstraintRDM::KeyCon |
        a.Key_BelongsRS = self.RHS_RS.RHS_RS_IND);
```

Listing 6.10. OCL ograničenje za proveru atributa proširenog RIC

Na sličan način definisana su ograničenja za sve tipove zavisnosti sadržavanja. OCL specifikacije tih ograničenje prikazane su na listinzima od 6.11 do 6.14.

```
invariant CheckRHSAttributeIRIC:

RHS_Attr_NKB_IND->forAll(a: ConstraintRDM::AttValCon |
        a.Attr_BelongsRS = self.RHS_RS.RHS_RS_IND);

invariant CheckLHSAttributeIRIC:

LHS_Key->forAll(a: ConstraintRDM::KeyCon |
        a.Key_BelongsRS = self.LHS_RS.LHS_RS_IND);
```

Listing 6.11. OCL ograničenje za proveru atributa u IRIC

```
invariant CheckRHS_RS_Ex_IRIC:

RHS_RS_Ex_IRI -> forAll(a : ConstraintRDM::AttValCon |
a.Attr_BelongsRS =
        self.RHS_RS_Ex_IRI.RHS_RS_IND or
        a.Attr_BelongsRS = self.RHS_RS.RHS_RS_IND);
```

Listing 6.12. OCL ograničenje za proveru atributa proširenog IRIC

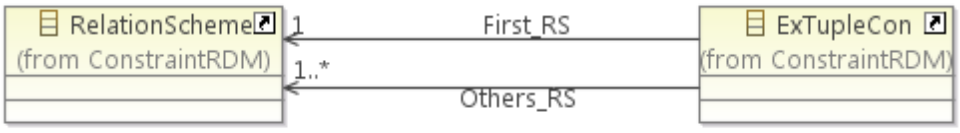
```
invariant CheckRHSAttributeNIRIC:

RHS_Attr_NKB_IND->forAll(a : ConstraintRDM::AttValCon |
        a.Attr_BelongsRS = self.RHS_RS.RHS_RS_IND);

invariant CheckLHSAttributeNIRIC:

LHS_Attr_NIRIC->forAll(a : ConstraintRDM::AttValCon |
        a.Attr_BelongsRS = self.LHS_RS.LHS_RS_IND);
```

Listing 6.13. OCL ograničenje za proveru atributa u ostalim osnovnim zavisnostima sadržavanja

```
invariant CheckLHS_RS_Ex_NIRI:

LHS_RS_Ex_NIRI->forAll(a : ConstraintRDM::AttValCon |
        a.Attr_BelongsRS = self.LHS_RS.LHS_RS_IND or
        a.Attr_BelongsRS = self.LHS_RS_Ex_NIRI.LHS_RS_IND);

invariant CheckRHS_RS_Ex_NIRI:

RHS_RS_Ex_NIRI ->forAll(a : ConstraintRDM::AttValCon |
        a.Attr_BelongsRS = self.RHS_RS.RHS_RS_IND or
        a.Attr_BelongsRS = self.RHS_RS_Ex_NIRI.RHS_RS_IND);
```

Listing 6.14. OCL ograničenje za proveru atributa u ostalim proširenim zavisnostima sadržavanja

Na kraju, na slici 6.8 prikazan je meta-model proširenog ograničenja torke. Prošireno ograničenje torke je višerelaciono ograničenje, predstavljeno u meta-modelu klasom **ExTupleCon**. Ograničenje torke se interpretira za svaku pojedinačnu torku t koja pripada spoju relacija $r(N_1) \triangleright\triangleleft \ldots \triangleright\triangleleft r(N_m)$. Logički uslov proširenog ograničenja torke definiše se nad skupom obeležja koja pripadaju uniji $\cup_{i=1}^{m}(R_i)$, gde su R_i skupovi obeležja šema relacija N_i.



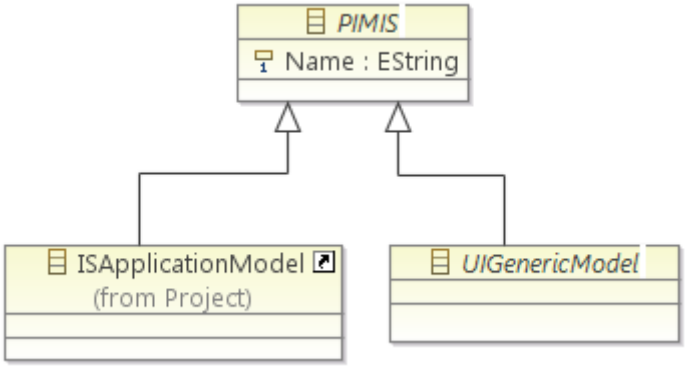
Slika 6.8. Meta-model proširenog ograničenja torke

6.4. Meta-model PIM koncepata alata IIS*Case

Meta-model okruženja IIS*Studio sadrži veliki broj koncepata, njihovih osobina, veza i pravila. Kompletan meta-model obezbeđuje formalne specifikacije svih koncepata ugrađenih u okruženje. U zavisnosti od namene alata kojem pripadaju, IIS*Studio PIM koncepti, generalizovani apstraktnom klasom ***PIMIS***, podeljeni su na:

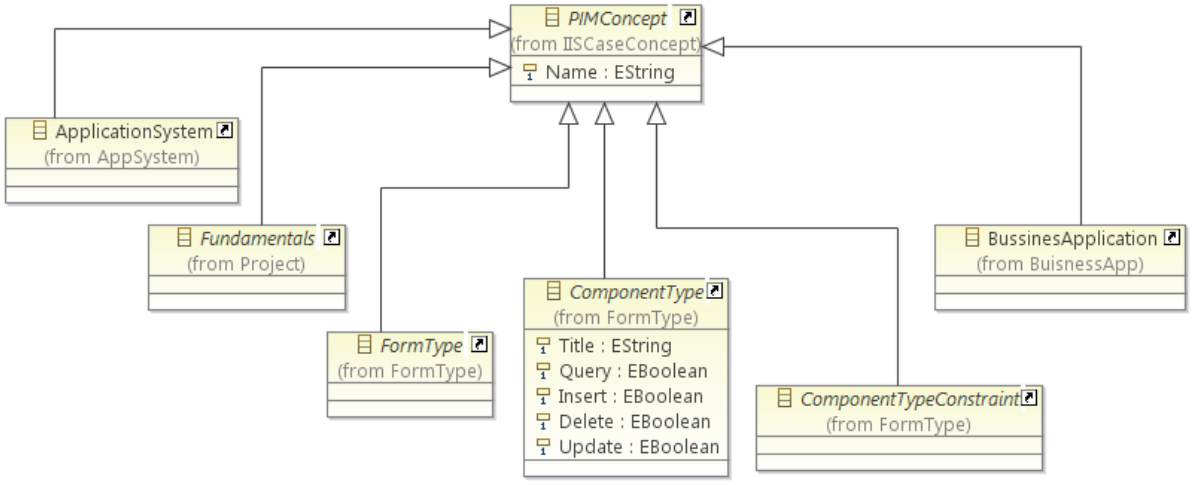
- koncepte za modelovanje aplikacije informacionog sistema i
- koncepte za modelovanje grafičkog interfejsa aplikacije.

Prva grupa koncepata predstavljena je u meta-modelu klasom **ISApplicationModel** i pripada IIS*Case alatu, namenjenom za konceptualno modelovanje šema baza podataka. Druga grupa koncepata, predstavljena apstraktnom klasom ***UIGenericModel***, pripada alatu UIModeler, koji je namenjen konceptualnom modelovanju korisničkih interfejsa. Na slici 6.9 prikazan je meta-model glavnih IIS*Studio PIM koncepata.



Slika 6.9. Meta-model glavnih IIS*Studio PIM koncepata

U narednom tekstu biće opisani samo odabrani koncepti IIS*Case-a koji su potrebni za opravdavanje hipoteza ove doktorske disertacije. Detaljan opis svih IIS*Case PIM koncepata može se naći u [Čelik11] i [Lukov13], dok se koncepti UIModeler-a predmet druge doktorske disertacije.



Slika 6.10. Meta-model glavnih IIS*Case PIM koncepata

Na slici 6.10 prikazani su glavni IIS*Case PIM koncepti, generalizovani apstraktnom klasom ***PIMConcept***: aplikativni sistem (**ApplicationSystem**), tip forme (***FormType***), tip komponente (***ComponentType***), ograničenje tipa komponente (***ComponentTypeConstraint***), poslovna aplikacija (**BussinesApplication**), kao i koncepti koji su svrstani u grupu osnovnih

(***Fundamentals***). U zagradama su navedeni nazivi klasa koje reprezentuju odgovarajući koncept u meta-modelu. Apstraktna klasa ***PIMConcept*** ima atribut **Name**, koji nasleđuju sve ostale klase.

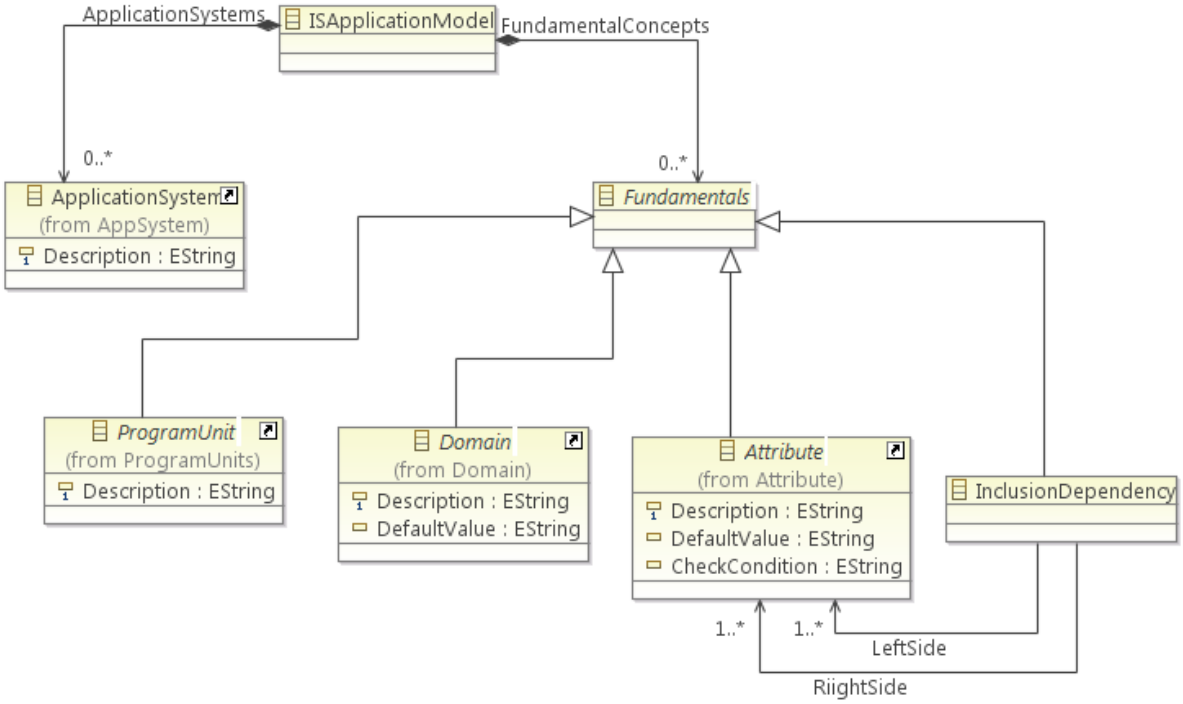
Svaki od ovih koncepata detaljno je opisan u narednim tačkama.

6.4.1. Meta-model koncepta modela aplikacije IS-a

Na vrhu hijerarhije IISCase meta-modela je model aplikacije informacionog sistema predstavljen klasom **ISApplicationModel**. U starijim verzijama alata ovo je bio koncept projekta. Svaki model aplikacije može uključivati više aplikativnih sistema i osnovnih koncepata. Koncept aplikativnog sistema predstavljen je **ApplicationSystem** klasom, dok su osnovni koncepti predstavljeni apstraktnom klasom ***Fundamentals***. Za svaki model aplikacije mora biti definisan naziv, što je u meta-modelu predstavljeno atributom **Name** koji se nasleđuje iz klase ***PIMConcept***. Meta-model koncepta modela aplikacije informacionog sistema prikazan je na slici 6.11.

Osnovni koncepti (***Fundamentals***) formalno su nezavisni od bilo kog aplikativnog sistema. Oni se kreiraju na nivou modela aplikacije informacionog sistema i mogu biti korišćeni u različitim aplikativnim sistemima. Osnovni koncepti su: domeni, atributi, zavisnosti sadržavanja i programske jedinice, predstavljeni na meta-modelu klasama ***Domain***, ***Attribute***, **InclusionDependency** i ***ProgramUnit***, respektivno. Poslednji pomenut koncept nije razmatran u procesu transformacije šema baza podataka, zbog čega će detalji o njemu biti izostavljeni. Detalji o ovom konceptu se mogu naći u [Čelik11].

Aplikativni sistem je organizacioni deo koji može imati podsisteme. Motiv za uvođenje ovog koncepta bio je obezbeđenje mehanizma za dekompoziciju velikih aplikacija na segmente kojima se može lakše upravljati.



Slika 6.11. Meta-model koncepta modela aplikacije IS-a

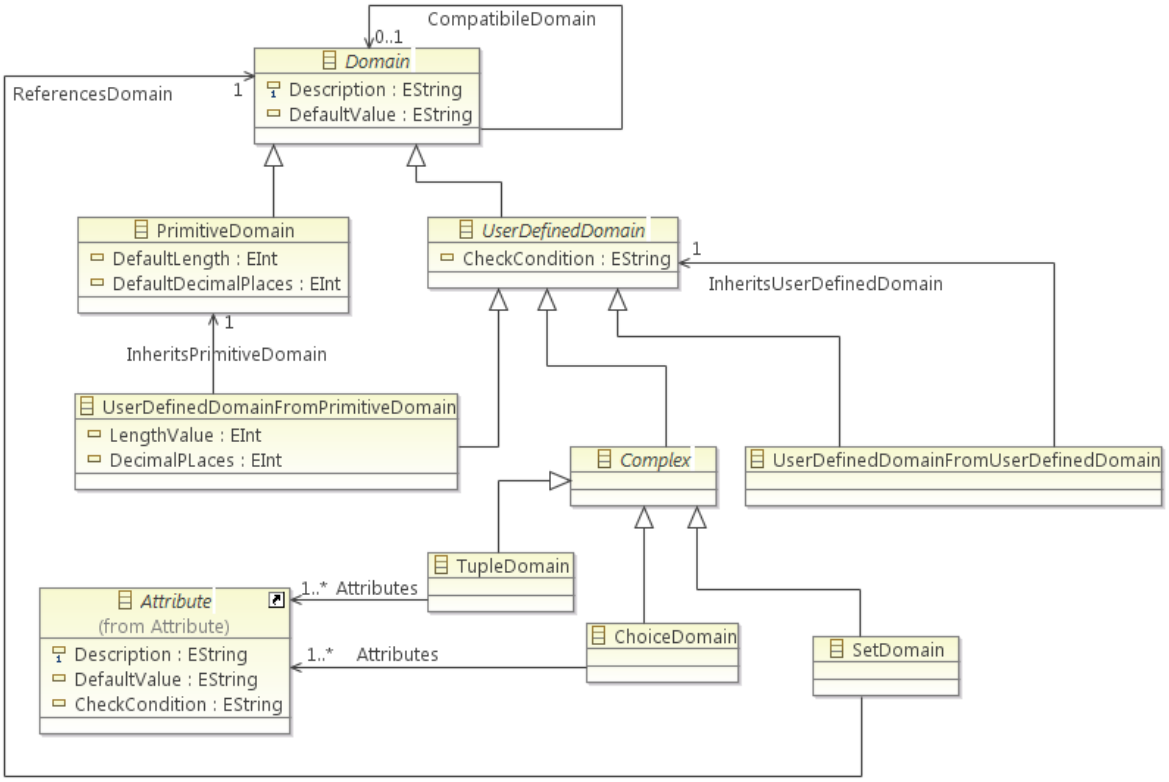
6.4.2. Mata-model koncepta domena

Domen, predstavljen u meta-modelu apstrkatnom klasom ***Domain***, pripada osnovnim konceptima (***Fundamentals***) i označava skup dozvoljenih vrednosti nekog atributa. Pojam domena koristi se sa značenjem uobičajenim za baze podataka. Na slici 6.12 prikazan je meta-model IIS*Case domen koncepta.

Za svaki domen moraju se definisati naziv i opis, dok je podrazumevana vrednost opciona. Naziv je u meta-modelu predstavljen atributom **Name**, nasleđenim iz klase ***PIMConcept***, dok su osobine opis i podrazumevana vrednost predstavljene atributima **Description** i **DefaultValue**, apstraktne klase ***Domain***. Naziv domena mora biti jedinstven u okviru čitavog modela aplikacije informacionog sistema.

Kao što se vidi na slici 6.12 domeni se dele na:

- primitivne domene predstavljene klasom **PrimitiveDomain** i
- korisnički definisane domene predstavljene apstraktnom klasom ***UserDefinedDomain***.



Slika 6.12. Meta-model *Domain* koncepta

Primitivni domeni predstavljaju primitivne tipove podataka različitih formalnih jezika npr. string, integer, float itd. Uvođenjem koncepta primitivnog domena, projektantima se dozvoljava da specificiraju potrebne primitivne domene prema potrebama informacionog sistema koji modeluju. Sa ovim konceptom povećava se izražajna moć modela koje projektant može da kreira. Pri definisanju primitivnog domena mogu se zadati osobine dužina i broj cifara iza decimalne tačke, predstavljene atributima **DefaultLength** i **DefaultDecimalPlaces**.

Specifikacija korisnički definisanog domena uključuje i zadavanje uslova (atribut **CheckCondition**). Uslov je regularni izraz koji može dodatno ograničiti skup mogućih vrednosti nekog domena. Korisnički definisani domeni mogu biti:

- domeni kreirani pravilom nasleđivanja i
- složeni domeni.

Domen koji je kreiran putem pravila nasleđivanja, nasleđuje specifikaciju prethodno definisanog primitivnog domena, predstavljenog klasom **UserDefinedDomainFrom PrimitiveDomain** ili korisnički definisanog domena, predstavljenog klasom **UserDefined DomainFromUserDefinedDomain**. Za domen koji nasleđuje primitivni domen, takođe, mogu se definisati dužina (atribut **LengthValue**) i broj cifara iza decimalne tačke (atribut **DecimalPlaces**).

Složeni domeni mogu biti kreirani putem tri pravila:

- pravila torke,
- pravila skupa i
- pravila izbora.

Domen kreiran upotrebom pravila torke naziva se domen tipa torke, predstavljen klasom **TupleDomain**. On predstavlja torku vrednosti. Ovakva vrsta složenog domena referencira već postojeće atribute kao elemente domena torke. Skupovni domen ili domen tipa skupa predstavljen klasom **SetDomain**, predstavlja skupove vrednosti nad definisanim domenom. Svaka vrednost iz skupovnog domena predstavlja skup vrednosti iz povezanog domena. Domen tipa izbora predstavljen klasom **ChoiceDomain**, definiše se na isti način kao i domen tipa torke. Svaka vrednost iz takvog domena mora odgovarati tačno jednom atributu koji je element izbora za dati domen izbora. IIS*Case domen izbora je identičan domenu tipa izbora koji je definisan u XML jeziku.

Specificirani domeni se referenciraju pri specifikaciji atributa. Atributi se koriste u raznim specifikacijama tipa forme aplikativnih sistema.

6.4.3. Meta-model koncepta atributa

Na slici 6.13 prikazan je meta-model IIS*Case ***Attribute*** koncepta. U okviru IIS*Case modela aplikacije, svaki atribut jedinstveno se identifikuje isključivo svojim nazivom. OCL specifikacija ograničenja za proveru jedinstvenog naziva atributa ista je kao na listingu 6.3. Ova osobina je u skladu sa pretpostavkom o postojanju šeme univerzalne relacije (URS). Osim naziva i opisa, koji su obavezani pri specifikaciji, za svaki atribut se može definisati podrazumevana vrednost i zadati uslov. Naziv je u meta-modelu predstavljen atributom **Name**, dok su osobine opis, podrazumevana vrednost i zadati uslov predstavljene atributima **Description**, **DefaultValue** i **CheckCondition**, respektivno, u apstraktnoj klasi ***Attribute***.

Uslov je regularni izraz koji može dodatno ograničiti skup mogućih vrednosti atributa. Ovaj uslov se definiše na sličan način kao i uslov prilikom kreiranja domena. Ukoliko je definisan uslov pri specifikaciji atributa, a postoji uslov i kod njemu pridruženog domena, koristi se logička operacija I (AND) za njihovo spajanje. Ukoliko se prilikom kreiranja atributa ne definiše njegova vrednost, vrednost atributa će biti inicijalizovana na podrazumevanu vrednost definisanu na nivou atributa. Ukoliko podrazumevana vrednost atributa nije definisana, vrednost atributa će biti inicijalizovana na podrazumevanu vrednost definisanu na nivou domena.

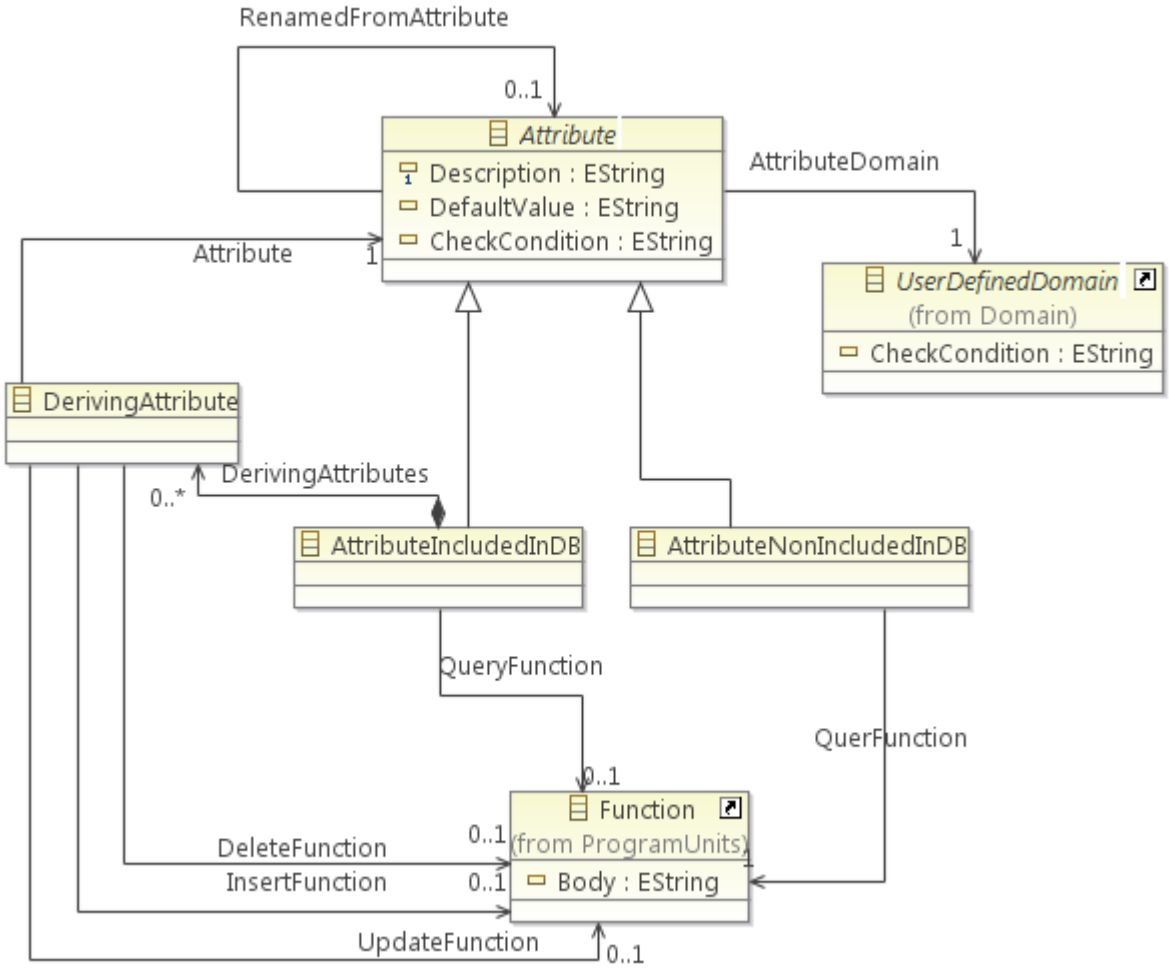
Svakom atributu, takođe, treba pridružiti domen što je na meta-modelu predstavljeno vezom **AttributeDomain** ka korisnički definisanom domenu.

Većina atributa nekog modela aplikacije je deo šeme baze podataka koja se projektuje u okviru informacionog sistema. Takođe, mogu postojati atributi koji predstavljaju neke izračunate vrednosti u okviru izveštaja ili ekranskih formi, a nisu uključeni u šemu baze podataka. Na osnovu toga, IIS*Case atributi se klasifikuju na:

- atribute uključene u šemu baze podataka, predstavljene u meta-modelu klasom **AttributeIncludedInDB** i
- atribute koji nisu uključeni u šemu baze podataka, predstavljene u meta-modelu klasom **AttributeNonIncludedInDB**.

Prema kriterijumu načina zadavanja vrednosti atributa, atributi se dele na:

- elementarne ili neizvedene, predstavljene u meta-modelu apstraktnom klasom ***Attribute*** i
- izvedene, predstavljene u meta-modelu klasom **DerivingAttribute**.



Slika 6.13. Meta-model *Attribute* koncepta

Vrednost neizvedenih atributa specificira najčešće krajnji korisnik, ili će njihova vrednost biti preuzeta iz nekog drugog spoljnjeg konteksta, dok se vrednost izvedenog atributa izračunava korišćenjem funkcije koja može predstavljati neku formulu ili algoritam. Postoji pravilo da svaki atribut koji nije uključen u šemu baze podataka mora biti definisan kao izveden. Izvedeni atributi mogu referencirati neku od funkcija: funkciju unosa, funkciju izmene ili funkciju brisanja, što je u meta-modelu predstavljeno vezama: **InsertFunction**, **UpdateFunction** i **DeleteFunction**, respektivno. Funkcije nisu razmatrane u okviru ove doktorske disertacije. Detalji o funkcijama mogu se naći u [Popov13].

Pored elementarnih, mogu se specificirati i preimenovani atributi, što je reprezentovano rekurzivnom vezom **RenamedFromAttribute**. Preimenovani atribut referencira prethodno definisani atribut i mora biti uključen u šemu baze podataka. Osnova nastajanja takvog atributa je referencirani atribut, ali sa drugačijom semantikom. Mehanizam preimenovanja potreban je kako bi se razlikovala semantika atributa. Ovaj mehanizam je sličan mehanizmu preimenovanja koji se koristi u mapiranju ER modela šeme baze podataka u relacioni model. Ukoliko projektant definiše atribut A1 koji je dobijen preimenovanjem atributa A, on u stvari

definiše zavisnost sadržavanja oblika $[A1] \subseteq [A]$ na nivou univerzalne šeme relacije. Zavisnost sadržavanja modelovana je kao osnovni koncept ***InclusionDependency*** prikazan na slici 6.11.

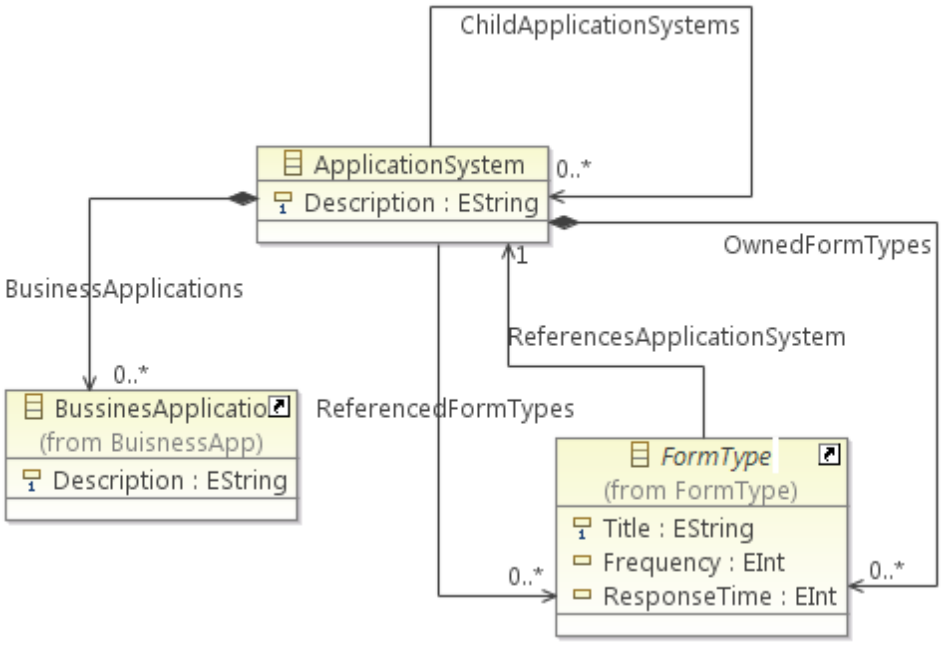
6.4.4. Meta-model koncepta aplikativnog sistema

Aplikativni sistem je koncept koji omogućava organizaciju projekta po celinama. **ApplicationSysteChildren** je rekurzivna veza koja predstavlja hijerarhijsku strukturu aplikativnih sistema. Aplikativni sistem može biti roditelj više aplikativnih sistema, dok aplikativni sistem može imati samo jednog direktno nadređenog.

Koncept **ApplicationSystem** prikazan je na slici 6.14. Aplikativni sistem može da sadrži više tipova formi, dok tip forme pripada isključivo jednom aplikativnom sistemu. Koncept tipa forme u meta-modelu predstavljen je apstraktnom klasom ***FormType***.

U okviru aplikativnih sistema definišu se i poslovne aplikacije (***BusinessApplications***) koje nisu predmet ove doktorske disertacije. Detalji o meta-modelu poslovnih aplikacija mogu se naći u [Čelik11].

Pri specifikaciji aplikativnog sistema mora se definisati njegov naziv i opis. Naziv je u meta-modelu predstavljen atributom **Name** nasleđene klase ***PIMConcept***, dok je opis predstavljen atributom **Description**, klase **ApplicationSystem.**



Slika 6.14. Meta-model AplicationSystem koncepta

6.4.5. Meta-model koncepta tipa forme

Koncept tipa forme predstavlja glavni meta-koncept u IIS*Case alatu. On predstavlja apstrakciju tipova dokumenata, ekranskih formi ili izveštaja koje krajnji korisnici informacionog sistema mogu koristiti u obavljanju svakodnevnih poslova. Putem koncepta tipa forme, projektant na nivou PIM-a, indirektno vrši specifikaciju šeme baze podataka sa uključenim atributima i ograničenjima, kao i sa modelima transakcionih programa i aplikacija informacionog sistema koji je predmet modelovanja. Osim kreiranja različitih tipova formi,

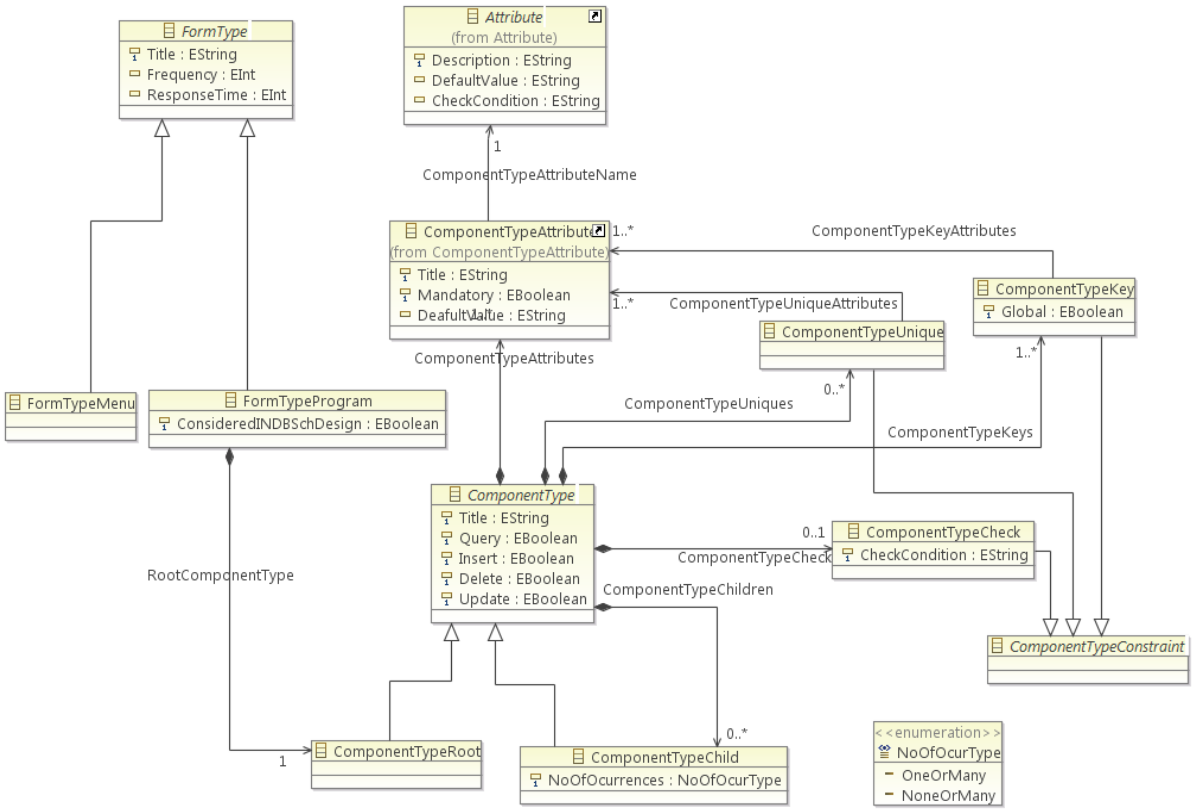
projektant može, u projektovani aplikativni sistem, uključiti i tipove formi kreirane za potrebe drugog aplikativnog sistema. Na osnovu toga, tipovi formi se mogu podeliti na:

- tipove formi koje aplikativni sistem poseduje i
- referencirane tipove formi.

Aplikativni sistem poseduje tip forme ukoliko je tip kreiran u okviru tog istog sistema. Ovaj tip forme može se naknadno modifikovati u okviru aplikativnog sistema koji ga poseduje bez bilo kakvih ograničenja. Na slici 6.14 prikazan je kompozitnom vezom **OwnedFormTypes**.

Referencirani tip forme kreiran je u okviru jednog aplikativnog sistema, a zatim uključen u drugi aplikativni sistem koji je predmet projektovanja. Svi referencirani tipovi formi se, u aplikativnom sistemu u koji su uključeni, mogu samo koristiti bez bilo kakve mogućnosti izmene strukture tipa. Na slici 6.14 prikazan je vezom **ReferencedFormTypes**.

Meta-model koncepta tipa forme prikazan je na slici 6.15.



Slika 6.15. Meta-model koncepta tipa forme

Svaki tip forme (***FormType***) poseduje sledeće osobine: naziv, koji ga jedinstveno identifikuje u okviru modela aplikacije, naslov, učestalost upotrebe i vreme odgovora. Osobine naziv i naslov, obavezne pri specifikaciji, predstavljene su u meta-modelu atributima **Name** i **Title**, respektivno. Učestalost upotrebe i vreme odgovora su opcione osobine, predstavljene u meta-modelu atributima **Frequency** i **ResponseTime**. Vreme odgovora je osobina koja specificira očekivano vreme odgovora nakon izvršenja odgovarajućeg programa.

Tipovi formi klasifikuju se na menije i programske tipovi forme. Meni tipovi formi se koristite za modelovanje menija aplikacije bez elemenata koji manipulišu podacima. Koncept meni tip forme predstavljen je u meta-modelu klasom **FormTypeMenu.** Programski tipovi formi koristite se za specifikaciju transakcionih programa sa korisničkim interfejsom. Ovi tipovi formi mogu predstavljati ekranske forme za preuzimanje i ažuriranje podataka, ili izveštaje za prikaz preuzetih podataka. Koncept programski tip forme predstavljen je u meta-modelu klasom **FormTypeProgram**.

Programski tip forme ima kompleksnu strukturu i može biti uzet ili izostavljen iz razmatranja prilikom projektovanja šeme baze podataka. Ovo ponašanje modeluje se atributom **ConsideredInDBSchDesign** klase **FormTypeProgram**. Tipovi formi, koji su označeni da se razmatraju prilikom projektovanja šeme baze podataka (atribut **ConsideredInDBSchDesign** ima vrednost *true*), kasnije predstavljaju ulazne podatke za proces generisanja šeme baze podataka. Tipovi formi koji su označeni da se ne uzimaju u razmatranje prilikom projektovanja šeme baze podataka (atribut **ConsideredInDBSchDesign** ima vrednost *false*), ne koriste se u procesu njenog generisanja. Ovi tipovi formi predstavljaju samo izveštaje ili tipove formi za preuzimanje podataka.

Za proces transformacije šema baza podataka interesantni su samo programski tipovi formi koji učestvuju u projektovanju baze podataka i u daljem tekstu će se za naziv programski tip forme koristiti samo tip forme.

Svaki tip forme predstavlja strukturu stabla tipova komponenti, koji su u meta-modelu predstavljeni apstraktnom klasom ***ComponentType***. Tip forme mora imati makar jedan tip komponente označen kao korenski, a može imati i više podređenih tipova. Korenski tip komponente je predstavljen klasom **ComponentTypeRoot**, dok su ostali predstavljeni klasom **ComponentTypeChild**.

Za svaki tip komponente definišu se osobine naziv (**Name**), naslov (**Title**) i dozvoljene operacije nad pojavama tipa komponente upit (**Query**), unos (**Insert**), izmena (**Update**) i brisanje (**Delete**). U zagradama su navedeni nazivi atributa klase ***ComponentType*** koji reprezentuju odgovarajuće osobine koncepta tipa komponente. Sve osobine pri specifikaciji su obavezne.

Naziv tipa komponente je ujedno i njegov jedinstveni identifikator. Svaka instanca nadređenog tipa komponente u stablu može imati više od jednog podređenog tipa komponente. Osobina broj pojavljivanja, predstavljena u meta-modelu atributom **NoOfOccurrences,** ograničava minimalan broj instanci podređenog tipa komponente koje mogu biti povezane sa instancom istog nadređenog tipa u okviru stabla. Broj pojavljivanja može biti specificiran kao: 0-N ili 1-N. Ove dve vrednosti u meta-modelu predstavljene su kao enumeracija sa nazivom **NoOfOcurType.** Vrednost 0-N označava da za svaku instancu nadređenog tipa komponente može postojati nula ili više instanci podređenog tipa. Vrednost 1- N označava da se za svaku instancu nadređenog tipa komponente zahteva da postoji bar jedna instanca podređenog tipa komponente.

Projektant, takođe, može definisati i osobine načina prikaza datog tipa komponente. Te osobine mogu se koristiti od strane generatora programa kako bi se definisao izgled komponente u okviru forme. Kako generator programa nije predmet razmatranja ove doktorske disertacije detalji o konceptu izgleda forme će biti izostavljeni.

Svaki tip komponente uključuje jedan ili više atributa. Atributi tipa komponente predstavljeni su klasom **ComponentTypeAttribute**. Ovaj koncept povezuje atribut (***Attribute***), iz skupa atributa specificiranih na nivou čitavog modela aplikacije, sa tipom komponente na kojoj će bit prikazan (veza u meta-modelu **ComponentTypeAttributeName**). Za svaki atribut tipa komponente se mora specificirati naslov (**Title**) kojim će atribut biti prezentovan na generisanoj ekranskoj formi kojoj taj tip komponente pripada, i da li je vrednost atributa obavezna ili opciona (**Mandatory**). Podrazumevana vrednost (**DefaultValue**) atributa na tipu komponente se može, a ne mora zadati. Atribut može biti uključen najviše jednom na nivou čitavog tipa forme.

Svaki tip komponente ima skup ograničenja, predstavljen apstraktnom klasom **ComponentTypeConstraint**, koji sadrži:

- ograničenje ključa,
- ograničenje jedinstvenosti i
- *check* ograničenje.

Svaki tip komponente mora imati najmanje jedan ključ koji sadrži najmanje jedan atribut tipa komponente. Ključ tipa komponente je predstavljen u meta-modelu klasom **ComponentTypeKey**. Ključ tipa komponente jedinstveno identifikuje svaku instancu tipa komponente, ali samo u okviru instance nadređenog tipa komponente.

Za svaki tip komponente mogu se definisati i ograničenja jedintvenosti, predstavljena u meta-modelu klasom **ComponentTypeUniqueness**. Kao i ograničenje ključa i ograničenje jedinstvenosti mora da sadrži najmanje jedan atribut tipa komponente, ali za razliku od ključeva, atributi u ograničenju jedinstvenosti mogi biti opcioni tj. mogu imati nula vrednost. Ograničenje jedintvenosti tipa komponente obezbeđuje da se svaka instanca tipa komponente jedinstveno identifikuje u okviru instance nadređenog tipa komponente, kada atributi koje sadrži nemaju nula vrednost.

Za tip komponente može se definisati i jedno *check* ograničenje koje izražava ograničenja na moguće vrednosti unutar jedne instance tipa komponente. Ovo ograničenje u meta-modelu predstavljeno je klasom **ComponentTypeCheck**. **CheckCondition** je atribut ove klase kojim se zadaje logički izraz kojim se proveravaju vrednosti svake instance tipa komponente.

6.5. Zaključak

Jedan od osnovnih ciljeva ove doktorske teze je da se u IIS*Studio ugrade transformacije između različitih modela podataka. Transformacija modela podataka ima važnu ulogu u procesu razvoja informacionih sistema kao i u njihovom reinženjeringu. U skladu sa MDA, pravila transformacije modela definišu se na nivou meta-modela. U ovom poglavlju su predstavljeni meta-modeli šema baza podataka potrebni za specifikaciju metoda transformacije modela u cilju reinženjeringa IS, koje će biti opisane u narednom poglavlju.

RSUBP meta-model i generički meta-model originalno su nastali u okviru ovog istraživanja, dok je IISCase meta-model nastajao postepeno, kao rezultat dugogodišnjeg istraživanja, a njegova kompletna MOF specifikacija je i predmet druge doktorske disertacije, čija realizacija je u toku. Svi meta-modeli su izraženi na jedinstven način, pomoću MOF meta-metamodela.

7. Proces transformacije šema baza podataka podržan modulom *M2M Transformer*

M2M Transformer je sastavni deo IIS*Ree alata, namenjen automatskoj transformaciji opisa šeme baze podataka, orijentisanog ka implementacji (PSM), u opis na višem apstraktnom nivou (PIM). Model, ekstrahovan iz nasleđene relacione baze podataka pomoću *M2M Transformer-a*, translira se nizom uzastopnih transformacija u novi, konceptualni model tipova formi, podržan alatom IIS*Case. Transformacije su zasnovane na meta-modelima prezentovanim u prethodnom poglavlju. Sve transformacije originalno su nastale kao rezultat istraživanja u okviru ove doktorske disertacije.

Transformaciju čine mapiranja kojima se definiše konverzija elemenata jednog modela, koji je u skladu sa odgovarajućim meta-modelom, u elemente drugog modela koji je, takođe, u skladu sa nekim meta-modelom. Konverzija elemenata definiše se tako što se uspostavlja relacija između odgovarajućih koncepata meta-modela.

Po preporuci MDA u [MDA03] mapiranje može biti opisano prirodnim jezikom, algoritmom nekog jezika za opis aktivnosti ili modelom, pomoću jezika za mapiranje. U skladu sa ovim preporukama, u ovoj doktorskoj disertaciji definicije transformacija opisane su na sledeći način:

- prvo, mapiranje između koncepata meta-modela definiše se korišćenjem prirodnog jezika,
- zatim se mapiranja strukturiraju u skup pravila koja se opisuju algoritmima u meta-jeziku ili opet prirodnim jezikom, u zavisnosti od složenosti mapiranja i
- na kraju, pravila transformacije su implementirana korišćenjem ATL jezika.

Poglavlje je organizovano na sledeći način. U odeljku 7.1 biće prezentovan scenario procesa transformacije, koji čini kompozicija transformacija, relacione šeme baze podataka u konceptualnu šemu alata IIS*Case, dok će u odeljcima od 7.2 do 7.5 biti dati detaljni opisi svih transformacija koje učestvuju u lancu.

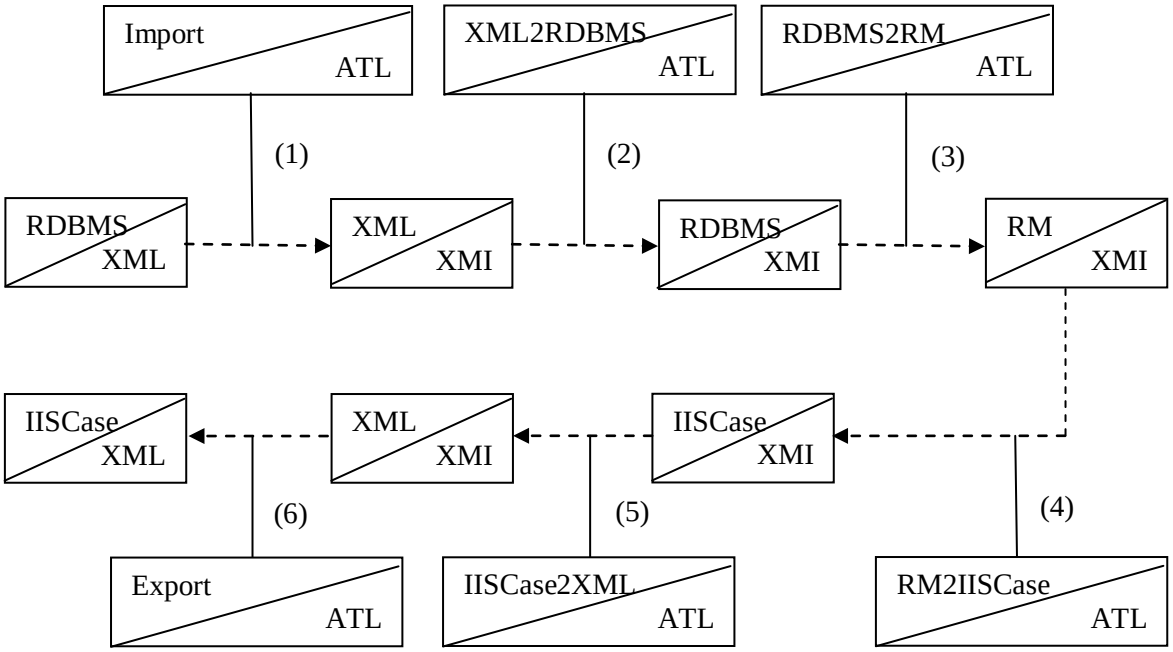
Transformacije su implementirane u ATL SDK, koji je deo Eclipse okruženja, korišćenjem ATL transformacionog jezika, verzija 3.3.1 [ATL10].

Primeri, na kojima će biti prezentovana primena algoritama transformacije, izabrani su iz veoma pojednostavljenog i hipotetičkog modela informacionog sistema univerziteta ***UniversityDB***.

7.1. Proces transformacije

U ovom odeljku ukratko će biti opisan metod, koji se sastoji od niza transformacija, primenjen u okviru modula *M2M Transformer*. Scenario procesa transformacije prikazan je na slici 7.1. Ukupnu transformaciju čini kompozicija pojedinačnih transformacija u kojima model na izlazu iz jedne transformacije predstavlja model na ulazu u drugu.

Instanca meta-modela relacione šeme baze podataka dobija se ekstrakcijom meta-podataka iz rečnika podataka nekog RSUBP-a, koje modul *XMLTransformer* serijalizuje u XML specifikaciju na osnovu predefinisane XML šeme. Algoritam za kreiranje XML specifikacija implementiran je pomoću GPL Java. Detalji ovog postupka prikazani su poglavlju 5.



Slika 7.1. Koraci procesa transformacije

XML specifikacija predstavlja ulaznu instancu niza transformacija koje se izvršavaju u modulu *M2M Transformer*. Kako ova XML specifikacija, pripada XML tehnološkom prostoru (XML TS) neophodno je dobiti odgovarajući model relacione baze podataka koji će pripadati MDA MOF tehnološkom prostoru (MOF TS).

U prvom koraku, označenom na slici 7.1 rednim brojem (1), konvertuje se XML specifikacija sa meta-podacima u MOF TS. Ovaj korak je neophodan da bi se prevazišao problem različitih tehnoloških prostora. Konverzija iz XML TS u MOF TS, vrši se XML *injector*-om, dok se za obrnuti smer koristi XML *extractor*. XML *injector* i *extractor* se ugrađeni u ATL SDK. XML *injector* od ulazne XML specifikacije (u XML TS) kreira instancu XML meta-modela (u MOF tehnološkom prostoru), dok XML *extractor* obezbeđuje obratno, od instance XML meta-modela kreira XML specifikaciju.

U drugom koraku, označenom na slici 7.1 rednim brojem (2), dobijeni XML model, koji je u skladu sa meta-modelom prezentovanim u odeljku 6.1, pomoću transformacije koja je na slici 7.1 predstavljena kao XML2RDBMS, transformiše se u model nazvan RDBMSM, koji je u skladu sa RSUBP meta-modelom, predstavljenim u odeljku 6.2.

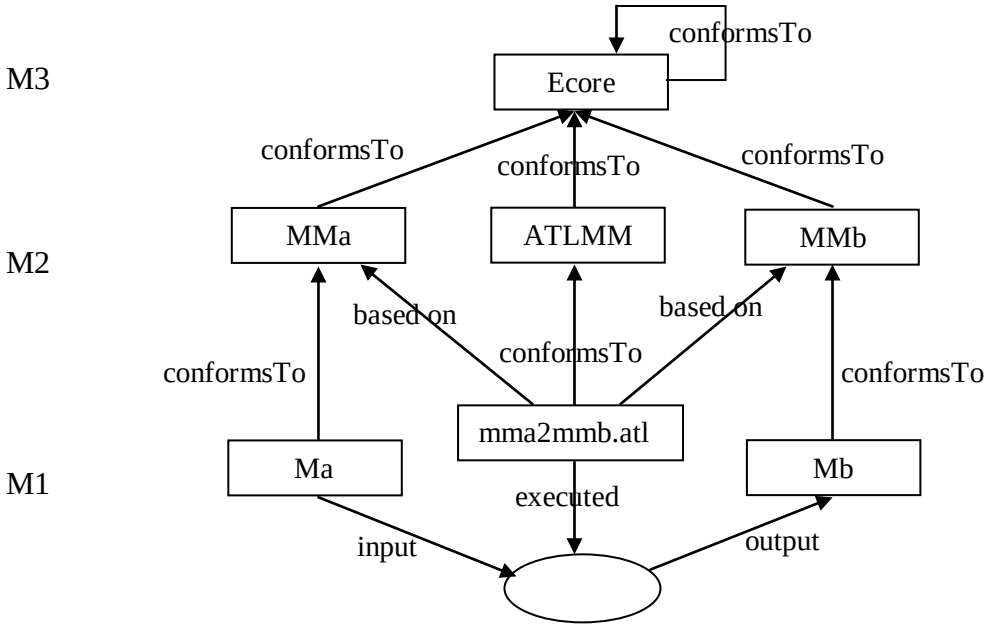
U trećem koraku, označenom na slici 7.1 rednim brojem (3), se RDBMSM, pomoću transformacije koja je na slici 7.1 predstavljena kao RDBMS2RM, transformiše u RM model koji je u skladu sa Generičkim meta-modelom prezentovanim u odeljku 6.3, a koji reprezentuje koncepte relacione šeme baze podataka zasnovane na teoretskim principima relacionih modela.

RM model se potom, u četvrtom koraku označenom na slici 7.1 rednim brojem (4), RM2IISCase transformacijom, transformiše u model IISCaseM koji je u skladu sa IISCase meta-modelom prezentovanim u odeljku 6.4 i predstavlja konceptualnu šemu baze podataka zasnovanu na PIM konceptima IIS*Case alata.

Početna ideja u okviru ove doktorske disertacije bila je da se postupak konceptualizacije izvrši u jednom koraku. Na ulazu bi bio ekstrahovani model relacione baze podataka, koji je u skladu sa RSUBP meta-modelom, a na izlazu konceptualni model tipova formi koji je u skladu sa IISCase meta-modelom. Međutim, ograničenje ATL-a, koji pri transformaciji modela ne obezbeđuje mogućnost navigacije kroz ciljni model, uticalo je na način implementacije transformacije. Transformacija je morala da se podeli na dve, kako bi se u međukoraku obezbedili nedostajući koncepti potrebni za formiranje konceptualnog modela tipova formi. Jedan od nedostajućih koncepata je šema univerzalne relacije, odnosno njeni atributi i ograničenja. Model tipova formi se zasniva na pretpostavci o postojanju URS-e, dok relaciona šema baze podataka karakteristična za neki RSUBP ne prepoznaje taj koncept. Takođe, od skupa ograničenja stranog ključa, koji je po pravilu RSUBP-ova vezan za odgovarajuću referenciranu tabelu, u međukoraku se generiše skup sledećih međurelacionih ograničenja: ograničenja referencijalnog integriteta, ograničenja inverznog referencijalnog integriteta i ograničenja koja sa desne strane nemaju ključ a nisu ograničenja inverznog referencijalnog integriteta. Ograničenja referencijalnog integriteta su ključni izvor informacija za kreiranje tipova formi.

U petom koraku, označenom na slici 7.1 rednim brojem (5), model IISCaseM u MOF tehnološkom prostoru tranformiše se u XML model pomoću transformacije IISCase2XML. U poslednjem, šestom koraku, označenom na slici 7.1 rednim brojem (6), šema baze podataka, predstavljena kao instanca XML meta-modela (MOF TS) serijalizuje se u XML format (XML TS) korišćenjem XML *extractor*-a.

Svi predstavljeni koraci transformacije biće detaljno opisani u narednim odeljcima. Prvo će biti prezentovana specifikacija pravila transformacije trećeg i četvrtog koraka niza transformacija jer su oni glavni predmet istraživanja ove doktorske disertacije. Prvi i drugi korak, kao i peti i šesti korak, koji su neophodni za prilagođavanje modela određenim tehnološkim prostorima, biće prezentovani u drugom delu ovog poglavlja.



Slika 7.2. Obrazac transformacije modela

Kao scenario transformacije modela biće korišćen MDE opšti obrazac prezentovan u tački 2.1.3. U skladu sa slikom 2.4, na slici 7.2 predstavljen je obrazac transformacije modela primenjen u ovoj doktorskoj disertaciji. Pošto su specifikacije svih transformacija napisane u ATL jeziku, transformacije (*Tab*) će biti predstavljene nazivima u obliku *mma2mmb.atl*.

Specifikacije transformacija su u skladu sa ATL meta-modelom koji je na slici 7.2 predstavljen sa ATLMM. Kao meta-metamodel korišćen je Ecore. Svi modeli i meta-modeli dobiće skraćene nazive, u cilju postizanja bolje preglednosti slika i algoritama.

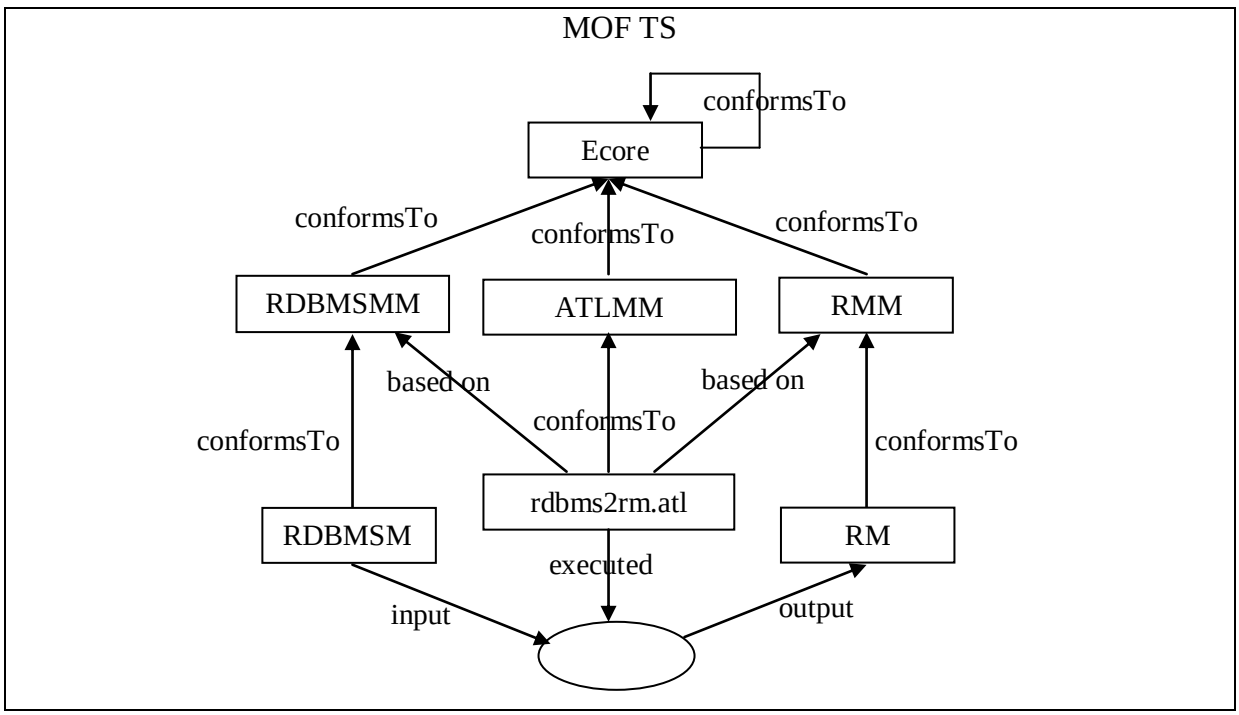
7.2. Definicija mapiranja koncepata RSUBP meta-modela u koncepte Generičkog meta-modela

U literaturi ravnopravno su zastupljeni izrazi specifikacija transformacije i definicija transformacije, kao i specifikacija mapiranja i definicija mapiranja. U ovoj doktorskoj disertaciji, kada se govori o celokupnoj transformaciji biće korišćen izraz specifikacija transformacije, dok će za pojedinačna mapiranja biti korišćen izraz definicija mapiranja, da bi ih razlikovali. Reč definicija, u ovom kontekstu, koristi se u svom širem značenju od onog koje je uobičajeno u matematici.

U ovom odeljku biće prezentovana specifikacija transformacije koja koncepate meta-modela šeme baze podataka, koji reprezentuje deo SQL standarda sa kojim je kompatibilna većina RSUBP mapira u koncepte Generičkog meta-modela relacione šeme baze podataka zasnovanog na teoriji relacionih modela. Transformacija je na slici 7.1 označena kao četvrti korak u nizu transformacija. Ova transformacija, nazvana RDBMS2RM, je tipa PSM-u-PSM i predstavlja međukorak u cilju dobijanja konceptualne šeme baze podataka opisane IIS*Case PIM konceptima. Transformacija RDBMS2RM relacionu šemu baze podataka postojećeg sistema transformiše u semantički bogatiji model na istom MDA nivou apstrakcije.

Scenario ovog koraka transformacije prikazan je na slici 7.3. Izvorni model je nazvan RDBMSM. On je u skladu sa RSUBP meta-modelom, koji je nazvan RDBMSMM. Izvorni model se pomoću transformacije *rdbms2rm.atl* transformiše u ciljni model, nazvan RM. Ovaje model je u skladu sa Generičkim meta-modelom, koji je nazvan RMM. Izvorni i ciljni meta-modeli (RDBMSMM i RMM) prezentovani su u poglavljima 6.2 i 6.3, respektivno.

Nakon transformacije, na izlazu se dobija semantički ekvivalentan model koji predstavlja ulazni model za transformaciju u sledećem, petom koraku niza transformacija.



Slika 7.3. Scenario trećeg koraka transformacije

U tabeli 7.1 prikazani su svi koncepti koji učestvuju u mapiranjima pri transformaciji. Sva pravila koja se primenjuju za mapiranje koncepata RSUBP meta-modela (RDBMSMM) u koncepte Generičkog meta-modela (RMM) biće detaljno opisana u tekstu koji sledi.

Ostatak odeljka organizovan je na sledeći način. U tački 7.2.1 predstavljen je algoritam za mapiranje korenskog elementa koji predstavlja RSUBP, njegovih tipova podataka i baze podataka, u projekat koji sadrži relacionu šemu baze podataka i šemu univerzalne relacije. U tački 7.2.2 biće prezentovan algoritam za mapiranje koncepta baze podataka na koncept relacione šeme baze podataka. Algoritam za mapiranje koncepta tabele sa atributima i relacionim ograničenjima u koncept šeme relacije sa relacionim ograničenjima biće opisan u tački 7.2.3, dok će u tački 7.2.4 biti prikazan algoritam mapiranja koncepta ograničenja stranog ključa u koncepte međurelacionih ograničenja. Pošto se u oba meta-modela specificiraju koncepti jezika za modelovanje relacionih šema baza podataka, veliki broj koncepta jednog se mapira na isti koncept drugog meta-modela. Razlika može biti samo u terminologiji, jer je u RSUBP meta-modelu korišćena terminologija karakteristična za RSUBP-ove, dok je u Generičkom meta-modelu korišćena terminologija iz [Mogin04].

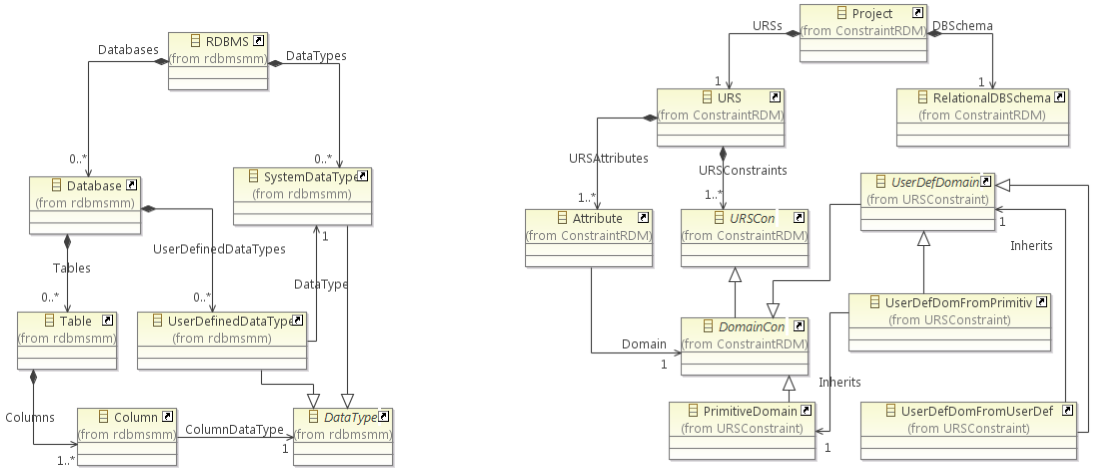
Tabela 7.1. Koncepti za mapiranje transformacije RDBMSM2RM

Koncepti RSUBP meta-modela (RDBMSMM)	Koncepti Generičkog meta-modela (RMM)
RDBMS	Project URS
Database	RelationalDBSchema
SystemDataType	PrimitiveDomain
UserDefinedDataType	UserDefDomFromPrimitiv
Table	RelationScheme
Column	Attribute NullAttr NotNullAttr UserDefDomFromPrimitiv UserDefDomFromUserDef
PrimaryKeyCon	KeyCon
UniqueCon	KeyCon UniqueCon
CheckCon	TupleCon
ForeignKey	ReferentialIntegrityCon RoleReferenced RoleReferencing
	NonInverseReferentialIntegrityCon RoleReferenced RoleReferencing
	InverseReferentialIntegrityCon RoleReferenced RoleReferencing

Radi lakšeg praćenja postupka transformacije, u svakom odeljku biće prikazani delovi meta-modela, sa konceptima koji učestvuju u algoritmima za mapiranje. U meta-modelima će biti smanjen nivo detaljnosti u odnosu na kompletne dijagrame, koji su prezentovani u šestom poglavlju.

7.2.1. Mapiranje koncepta RDBMS u koncept Project

Zbog XML strukture koju zahteva Ecore, oba meta-modela imaju korenski element koji predstavlja koncept koji sadrži sve ostale. Korenski element **RDBMS,** meta-modela RDBMSMM, koji predstavlja koncept relacionog sistema za upravljanje bazama podataka, mapira se na korenski element RMM meta-modela, koji predstavlja koncept projekta, pod nazivom **Project**. Na slici 7.4 prikazani su delovi meta-modela sa ovim konceptima, kao i sa ostalim konceptima koji su obuhvaćeni algoritmom za mapiranje, čiji je pseudokod prikazan na slici 7.5.



a) RDBMSMM b) RMM

Slika 7.4. Delovi meta-modela za mapiranje koncepta **RDBMS** u koncept **Project**

Naziv projekta dobija naziv relacionog sistema za upravljanje bazama podataka sa prefiksom Project_. Od šeme baze podataka koju je sadržao **RDBMS**, *DB*(*TB, FK*), gde je *TB* skup tabela, a *FK* skup ograničenja stranog ključa (u meta-modelu klasa **Database**), biće kreirana relaciona šema baze podataka, *DBSchema*, koju će sadržati projekat (u meta-modelu klasa **RelationalDBSchema**). Algoritam procedure *KreiranjeDBSchema*, za mapiranje ova dva koncepta biće prikazan i objašnjen u narednoj tački odeljka.

U Generičkom meta-modelu relacione šeme baze podataka podržan je koncept šeme univerzalne relacije (URS), pri čemu se u skladu sa pretpostavkom o postojanju URS-e, atributi definišu nezavisno od šeme relacije kojoj će pripadati. U okviru URS-e svaki atribut se jedinstveno identifikuje svojim nazivom, za razliku od RSUBP-ova kod kojih se svaki atribut jednistveno identifikuje unutar šeme relacije kojoj pripada. Usled ove razlike može se pojaviti problem homonima. Identifikacija homonima i način razrešavanja kolizije naziva obeležja detaljno je opisan u petom poglavlju. Smatra se da u ovom koraku transformacije u instanci koja predstavlja ulazni model, nema atributa sa istim nazivom, a različitom semantikom. Ova pretpostavka omogućava da se od skupova atributa koji pripadaju šemama relacija, njihovim uniranjem dobije univerzalni skup obeležja *U*:

$$\bigcup_{i=1}^{m} R_i = U,$$

$$(\forall A_i, A_j \in U)(i \neq j \Leftrightarrow A_i \neq A_j).$$

Algoritam:	ZA MAPIRANJE KORENSKIH ELEMENATA
Ulaz:	*RDBMS* *DB*(*TB, FK*) *PDT* – skup sistemskih tipova podataka *UDT* – skup korisnički definisanih tipova podataka
Izlaz:	*Projekat*
Lokalne deklaracije: *urs* - izlazni parametar iz procedure *KreiranjeURS* (*urs* – šema univerzalne relacije *URS*(*U, O*)) *dbsh* – izlazni parametar iz procedure *KreiranjeDBSchema* (*dbsh* – šema baze podataka)	
Pseudokod: **POČETAK PROCESA** KreirajProjekat **RADI** kreiranjeP **POSTAVI** *Naziv* ←'Project_' + *Naziv*(*RDBMS*) **POZOVI** *KreiranjeURS*(*DB, PDT, UDT, urs*) **POSTAVI** *URSs* ← *urs* **POZOVI** *KreiranjeDBSchema*(*DB, dbsh*) **POSTAVI** *DBSchema* ← *dbsh* **KRAJ RADI** kreiranjeP **KRAJ PROCESA** KreirajProjekat	

Slika 7.5. Algoritam za mapiranje koncepta **RDBMS** u koncept **Project**

Pored skupa stributa, URS-a sadrži i skup ograničenja koja su dataljno objašnjena u tački 6.3.1. Ograničenja URS-e koja se mogu dobiti transformacijom iz ekstrahovanih meta-podataka postojeće relacione baze podataka su:

- ograničenje primitivnih domena (**PrimitiveDomain**),
- ograničenje korisnički definisanih domena koji nasleđuju primitivni domen (**UserDefDomFromPrimitiv**) i
- ograničenje korisnički definisanih domena koji nasleđuju korisnički definisani domen (**UserDefDomFromUserDef**).

Skup primitivnih domena, *PD*, nastaće od skupa sistemskih tipova podataka, *SDT* (u meta-modelu **SystemDataType**). Skup korisnički definisanih domena, *UDP*, koji nasleđuju primitivni domen nastaće od skupa korisnički definisanih tipova podataka, *UDT* (u meta-modelu **UserDefinedDataType**) i skupa domena koji nastaju iz specifikacije kolona čiji tip podatka je iz skupa sistemskih tipova podataka. Skup korisnički definisanih domena koji nasleđuju korisnički definisani domen, *UDU*, nastaće iz specifikacije kolona čiji tip podatka je iz skupa korisnički definisanih tipova podataka. Korisnički definisani domen kreira se za sve kolone bez obzira da li je u kolonama eksplicitno zadata preciznost i dužina tipa podatka ili je korišćena definicija i podrazmevane vrednosti sistemskog tj. primitivnog tipa podatka. Ovaj način implementacije izabran je iz razloga što IIS*Case, čiji model tipova formi je krajnji cilj čitavog niza transformacija, dozvoljava pridruživanje samo korisnički definisanih domena atributu, ali ne i primitivnih.

Pseudokod algoritma procedure koja, na osnovu skupa tabela sa kolonama, sistemskih tipova podataka i korisnički definisanih tipova podataka, kreira instancu koncepta URS-e, prikazan je na slici 7.6.

Procedura:	*KreiranjeURS*
Formalni parametri:	
Ulazni:	*DB*(*TB*,*FK*) *TB* = { $T_i(C_i, O_i)$ | *i* = 1,.., *n*} *SDT* – skup sistemskih tipova podataka *UDT* – skup korisnički definisanih tipova podataka
Izlazni:	*urs* – *URS*(*U*, *O*) *O* = *PD* ∪ *UDP* ∪ *UDU*
Lokalne deklaracije: *o* - izlazni parametar iz procedure *KreiranjeKorisničkiDefDomena* (*o* –ograničenje korisnički definisanog domena)	

Pseudokod:

POČETAK PROCESA KreiranjeURS
  POSTAVI *Naziv* ←'URS'
  RADI kreirajU (∀ $T_i \in TB$)
    POSTAVI $U \leftarrow U \cup C_i$
  KRAJ RADI kreirajU
  RADI kreirajPD (∀ $DT_i \in SDT$)
    POSTAVI $PD \leftarrow PD \cup DT_i$
  KRAJ RADI kreirajPD
  RADI kreirajUDP (∀ $UT_i \in UDT$)
    POSTAVI $UDP \leftarrow UDP \cup UT_i$
  KRAJ RADI kreirajUDP
  RADI kreiraj (∀ $T_i \in TB$, ∀ $C_i \in T_i$)
    AKO JE $DT(C_i) \in SDT$ **TADA**
      POZOVI *KreiranjeUDPodSDT*(C_i, *o*)
      POSTAVI $UDP \leftarrow UDP \cup o$
    INAČE
      POZOVI *KreiranjeUDUodUDT*(C_i, *o*)
      POSTAVI $UDU \leftarrow UDU \cup o$
    KRAJ AKO
    POSTAVI $O \leftarrow PD \cup UDP \cup UDU$
  KRAJ RADI kreiraj
KRAJ PROCESA KreiranjeURS

Slika 7.6. Algoritam procedure *KreiranjeURS*

Radi jednostavnosti prikaza algoritma izostavljen je pseudokod za procedure *KreiranjeUDPodSDT*, kojom se kreira korisnički definisani domen koji nasleđuje primitivni domen i *KreiranjeUDUodUDT*, kojom se kreira korisnički definisani domen koji nasleđuje korisnički definisani domen iz specifikacije kolone neke tabele. Definicija procedura data je pomoću ATL specifikacije koja će biti objašnjena kasnije.

Definicija opisanog mapiranja sa slike 7.6 implementirana je pomoću ATL pravila *RDBMS2Project* prikazanog na listingu 7.1. Primenom ovog pravila na ulazni model dobija se projekat (*p*) i šema univerzalne relacije (*urs*) sa skupom atributa (*attr*) i skupom ograničenja (*URSConstraints*), izlaznog modela. Skup atributa se kreira na osnovu skupa jedinstvenih imena (*names*), pri čemu se za svaki naziv kreira po jedan atrbut. Ime atributa postaje upravo ovaj naziv. Svakom atributu se pridružuje domen. ATL pomoćna metoda (*helper*) za pronalaženje odgovarajućeg domena za zadati atribut prikazana je na listingu 7.2. U ATL

pravilu se primenjuju i pomoćni atributi za specifikaciju skupa ograničenja koja pripadaju URS-i, prikazani na listingu 7.3.

```
rule RDBMS2Project {
from
   db : RDBMS!RDBMS
   using {
      names: Set(String) = RDBMS!Column.allInstances() -> collect(e |
                                                   e.refGetValue('Name'))->asSet();}
to
   p : RM!Project (
      Name <- 'Project_' + db.Name,
      DBSchema <-db.Databases -> first(),
      URSs <- urs
   ),
   urs: RM!URS  (
      Name <- 'URS',
      URSConstraints <- db.DataTypes.union(db.getUserDefDataType).
      union(thisModule.DataTypeForColumn -> collect(e |
      thisModule.Column2DataType(e))).union(thisModule.UdataTypeForColumn
      ->collect(e | thisModule.Column2UDataType(e))),
      URSAttributes <- attr
   ),
   attr: distinct RM!Attribute foreach(e in names)(
      Name <- e,
      Domain <- thisModule.getDataTypeOfAttribute(e)
   )
}
```

Listing 7.1. ATL pravilo za mapiranje koncepta **RDBMS** u koncept **Project**

```
helper def: getDataTypeOfAttribute (n:String): RM!DomainCon =
RM!DomainCon.allInstances() -> select(e |
            e.refGetValue('Name').toString().startsWith(n)).first();
```

Listing 7.2. ATL *helper* za određivanje ograničenja domena određenog atributa

Pomoću prvog atributa sa listinga 7.3, *getUserDefDataType* dobija se skup korisnički definisanih domena iz skupa korisnički definisanih tipova podataka koje je korisnik specificirao u okviru RSUBP-a. Pomoću drugog atributa, *DataTypeForColumn*, dobija se skup kolona, čiji je tip podatka (*ColumnDataType*) iz skupa sistemskih tipova podataka (*SystemDataType*), na osnovu čije specifikacije će se dobiti skup korisnički definisanih domena koji nasleđuju primitivni domen, dok se pomoću trećeg atributa, *UdataTypeForColumn*, dobija skup kolona čiji je tip podatka iz skupa korisnički definisanih domena (*UserDefinedDataType*), na osnovu čije specifikacije će se dobiti skup korisnički definisanih domena koji nasleđuju korisnički definisani domen. Iz pravila prikazanog na listingu 7.1, za svaku kolonu iz poslednja dva skupa kolona, pozivaju se ATL 'lenja' (*lazy*) pravila prikazana na listinzima 7.4 i 7.5, respektivno, koja će kreirati odgovarajući domen.

```
helper context RDBMS!RDBMS def: getUserDefDataType:
Set(RDBMS!UserDefinedDataType) =
            self.Databases -> collect(e | e.UserDefinedDataTypes);

helper def: DataTypeForColumn:Set(RDBMS!Column) =
let col:Set(RDBMS!Column) = RDBMS!Column.allInstances() in
   col->iterate(e; rez: Set(RDBMS!Column) = Set{} |
```

```
   if e.ColumnDataType.oclIsTypeOf(RDBMS!SystemDataType) then rez.including(e)
   else rez endif);

helper def: UDataTypeForColumn:Set(RDBMS!Column) =
let col:Set(RDBMS!Column) = RDBMS!Column.allInstances() in
   col->iterate(e; rez: Set(RDBMS!Column) = Set{} |
   if e.ColumnDataType.oclIsTypeOf(RDBMS!UserDefinedDataType) then
   rez.including(e) else rez endif);
```

Listing 7.3. ATL atributi sa skupovima korisnički definisanih tipova podaka i kolona za kreiranje korisničkih domena

Naziv (*Name*) i jednog i drugog domena sadržaće naziv tipa podatka kolone sa prefiksom naziva kolone. Prefiks naziva kolone pomaže u pronalaženju odgovarajućeg ograničenja domena svakog atributa koji pripada URS-i, što je prikazano na listingu 7.2. Ovaj mehanizam je upotrebljen zbog ograničenja ATL jezika, pomoću kojeg nije dozvoljeno da se deklarativnim putem referenciraju novokreirane komponente ciljnog modela, kao što je u ovom slučaju URS-a tj. njeni atributi i ograničenja.

```
lazy rule Column2DataType{
from
   c : RDBMS!Column
to
   dt: RM!UserDefDomFromPrimitiv(
      Name <- c.Name + '_' + c.ColumnDataType.Name,
      DecimalPlaces <- c.Precision,
      Length <- c.Length,
      DefaultValue <- c.Default,
      Inherits <- c.ColumnDataType
   )
}
```

Listing 7.4. ATL *lazy* pravilo za mapiranje u korisničkih definisani domen koji nasleđuje primitivni domen

Pravilom *Column2DataType* prikazanim na listingu 7.4 podrazumevana vrednost, broj decimalnih mesta i dužina koje su zadate pri specifikaciji kolone neke tabele, postaju podrazumevana vrednost (*DefaultValue*), broj decimalnih mesta (*DecimalPlaces*) i dužina (*Length*) korisnički definisanog domena koji nasleđuje primitivni domen. Tip podatka kolone (*ColumnDataType*), koji je iz skupa sistemskih tipova podataka i od kog će nastati primitivni domen pomoću pravila prikazanog na listingu 7.6, postaje nasleđeni primitivni domen ovog domena (*Inherits*).

Pravilom *Column2UDataType* prikazanim na listingu 7.5 podrazumevana vrednost koja je zadata pri specifikaciji kolone neke tabele, postaje podrazumevana vrednost (*DefaultValue*) korisnički definisanog domena koji nasleđuje korisnički definisani domen. Pošto je ovo pravilo definisano nad skupom atributa čiji tip podatka je iz skupa korisnički definisanih tipova podatka, nasleđeni domen domena koji nastaje ovim pravilom biće tipa korisnički definisanog domena koji nasleđuje primitivni domen. Pravilo za kreiranje ovog domena koji se nasleđuje, prikazano je na listingu 7.7.

```
lazy rule Column2UDataType{
from
   c : RDBMS!Column
to
```

```
   dt: RM!UserDefDomFromUserDef(
      Name <- c.Name + '_' + c.ColumnDataType.Name,
      DefaultValue <- c.Default,
      Inherits <- c.ColumnDataType
   )
}
```

Listing 7.5. ATL *lazy* pravilo za mapiranje u korisničkih definisani domen koji nasleđuje korisnički definisani domen

Koncept sistemskih tipova podataka RSUBP-a mapira se direktno na koncept primitivnog domena, tako da je zbog jednostavnosti izostavljen pseudokod algoritma ovog pravila. Definicija mapiranja biće prezentovana samo pomoću ATL specifikacije. Za svaki primitivni tip podatka koji RSUBP ima, kreira se instanca primitivnog domena. Naziv, podrazumevana dužina i broj decimalnih mesta sistemskog tipa podatka postaju naziv (*Name*), podrazumevana dužina (*DefaultLength*) i broj decimalnih mesta (*DefaultDecimalPlaces*) primitivnog domena. ATL specifikacija mapiranja ovih koncepata predstavljena je pravilom *SystemDataType2PrimitiveDomain* prikazanim na listingu 7.6.

```
rule SystemDataType2PrimitiveDomain {
from
   sdt : RDBMS!SystemDataType
to
   out : RM!PrimitiveDomain (
      Name <- sdt.Name,
      DefaultLength <- sdt.PredefinedLength,
      DefaultDecimalPlaces <- sdt.PredefinedDecPlaces
   )
}
```

Listing 7.6. ATL pravilo za transformaciju sistemskih tipova podataka

Na listingu 7.7. prikazano je ATL pravilo za mapiranje korisnički definisanih tipova podataka u korisnički definisani domen koji nasleđuje primitivni domen. Naziv, podrazumevana vrednost, dužina i broj decimalnih mesta korisnički definisanog tipa podatka postaju naziv (*Name*), podrazumevana vrednost (*DefaultValue*), dužina (*Length*) i broj decimalnih mesta (*DecimalPlaces*) korisnički definisanog domena. Nasleđeni primitivni tip podatka postaje primitivni domen koji se nasleđuje.

```
rule UserDefDataType2UserDefFromPrimitiveD {
from
   udt : RDBMS!UserDefinedDataType
to
   out : RM!UserDefDomFromPrimitiv (
      Name <- udt.Name,
      Length <- udt.Length,
      DecimalPlaces <- udt.Precision,
      DefaultValue <- udt.DefaultValue,
      Inherits <- udt.DataType
   )
}
```

Listing 7.7. ATL pravilo za transformaciju korisničkih tipova podataka

Primer 7.1. Dat je deo šeme baze podataka IS univerziteta pod nazivom *UniversityDb*, implementiran pomoću SUBP Oracle10g. Na slici 7.7 prikazan je, u vidu stabla, jedan mali segment šeme baze podataka koji će ilustrovati primenu prethodno navedenih pravila. Prikazani segment šeme baze sadrži:

- dva sistemska tipa podatka: *Integer* i *Varchar*,
- jedan korisnički definisani tip podatka: *D_Name* i
- dve šeme relacije samo sa skupovima atributa:
  - *University*({*UniId*, *UniName*, *UniCity*})
  - *Faculty*({*UniId*, *FacId*, *FacShortName*, *FacName*, *Dean*})

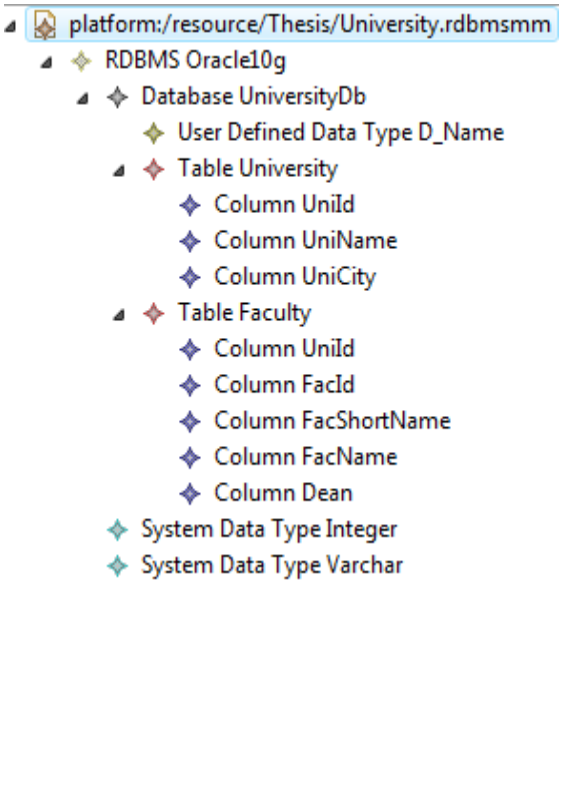
Ograničenja šema relacija su, u ovom trenutku, izostavljena, jer će kasnije biti kreirana primenom drugih pravila.

Za predstavljeni deo šeme baze podataka na slici 7.7 a) primenom ATL pravila *RDBMS2Project* dobija se deo relacione šeme baze podataka koja je u skladu sa Generičkim meta-modelom, prikazan na slici 7.7 b). Na slici se može videti da je od unije skupova kolona tabela *University* i *Faculty* nastao skup atributa URS-e.

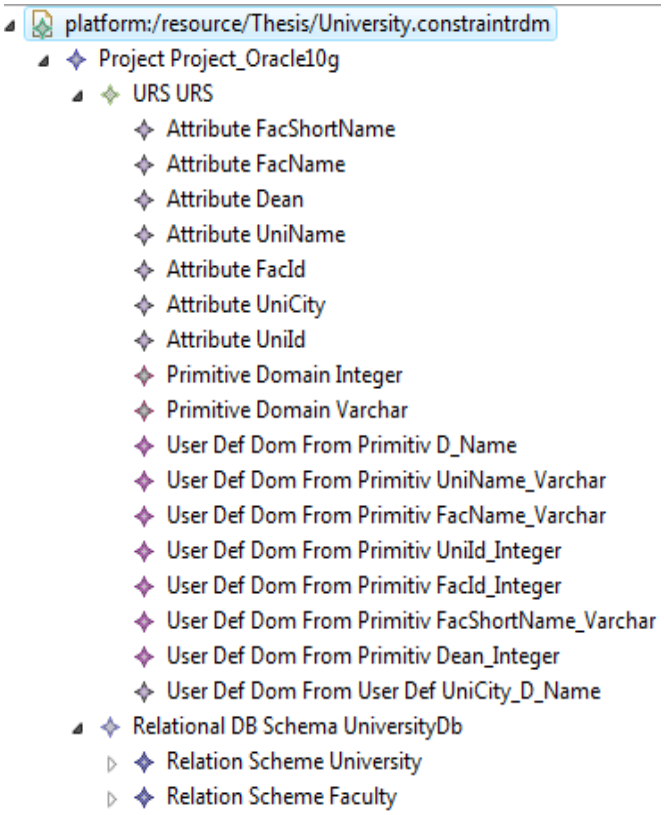
Sistemski tipovi podataka *Integer* i *Varchar* postaju primitivni domeni sa istim nazivom. Korisnički definisani tip podatka *D_Name* postaje korisnički definisani domen koji nasleđuje primitivni.

Za svaku kolonu obe tabele nastao je korisnički definisani domen koji nasleđuje primitivni domen, osim za kolonu *UniCity* čiji tip podatka nije bio iz skupa sistemskih tipova, već je bio korisnički definisan. Od ove kolone nastao je korisnički definisani domen koji nasleđuje korisnički definisani domen sa nazivom *UniCity_D_Name*.

Svi kreirani domeni su, kao i atributi, uključeni u URS-u. Za svaku tabelu kreirana je šema relacije.



a) Model dela *UniversityDb* u skladu sa RDBMSMM



b) Model dela *UniversityDb* u skladu sa RMM

Slika 7.7. Deo *UniversityDb* relacione šeme baze podataka u Eclipse editoru

Na listinzima 7.8 i 7.9 prikazani su primeri sa slike 7.7 a) i b) u XML XMI formatu, respektivno.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdbmsmm:RDBMS xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:rdbmsmm="http://rdbmsmm/1.0" Name="Oracle10g">
  <Databases Name="UniversityDb">
    <UserDefinedDataTypes Name="D_Name" DataType="//@DataTypes.1" Length="15"
                                                              DefaultValue=""/>
    <Tables Name="University">
      <Columns Name="UniId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="UniName" Length="20" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="UniCity" Nullable="true" Length="15"
                                               ColumnDataType="//@DataTypes.1"/>
    </Tables>
    <Tables Name="Faculty">
      <Columns Name="UniId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="FacId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="FacShortName" Length="5" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="FacName" Length="20" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="Dean" Nullable="true" ColumnDataType="//@DataTypes.0"/>
    </Tables>
  </Databases>
  <DataTypes Name="Integer"/>
  <DataTypes Name="Varchar" PredefinedLength="256"/>
</rdbmsmm:RDBMS>
```

Listing 7.8. Model dela *UniversityDb* u skladu sa RDBMSMM u XML XMI formatu

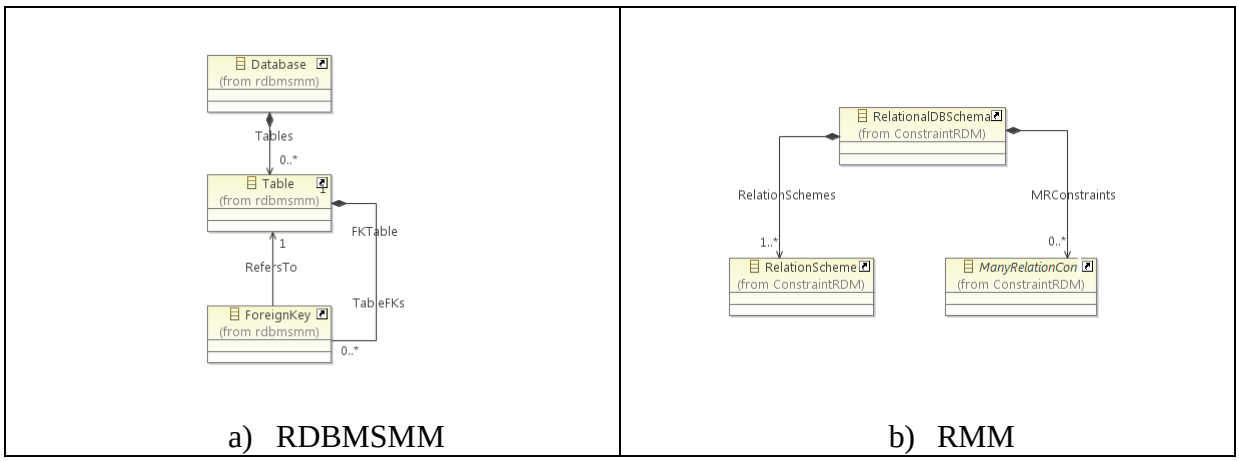
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConstraintRDM:Project xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ConstraintRDM="http://relacionimm/1.0/ConstraintRDM"
xmlns:RelationConstraint="http://relacionimm/1.0/RelationConstraint"
xmlns:URSConstraint="http://relacionimm/1.0/URSConstraint" Name="Project_Oracle10g">
  <URSs Name="URS">
    <URSAttributes Name="FacShortName" Domain="//@URSs/@URSConstraints.1"/>
    <URSAttributes Name="FacName" Domain="//@URSs/@URSConstraints.1"/>
    <URSAttributes Name="Dean" Domain="//@URSs/@URSConstraints.0"/>
    <URSAttributes Name="UniName" Domain="//@URSs/@URSConstraints.1"/>
    <URSAttributes Name="FacId" Domain="//@URSs/@URSConstraints.0"/>
    <URSAttributes Name="UniCity" Domain="//@URSs/@URSConstraints.1"/>
    <URSAttributes Name="UniId" Domain="//@URSs/@URSConstraints.0"/>
    <URSConstraints xsi:type="URSConstraint:PrimitiveDomain" Name="Integer"/>
    <URSConstraints xsi:type="URSConstraint:PrimitiveDomain" Name="Varchar"
                                                              DefaultLength="256"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromPrimitiv" Name="D_Name"
            DefaultValue="" Inherits="//@URSs/@URSConstraints.1" Length="15"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromPrimitiv"
        Name="UniName_Varchar" Inherits="//@URSs/@URSConstraints.1" Length="20"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromPrimitiv"
        Name="FacName_Varchar" Inherits="//@URSs/@URSConstraints.1" Length="20"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromPrimitiv"
        Name="UniId_Integer" Inherits="//@URSs/@URSConstraints.0"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromPrimitiv"
        Name="FacId_Integer" Inherits="//@URSs/@URSConstraints.0"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromPrimitiv"
        Name="FacShortName_Varchar" Inherits="//@URSs/@URSConstraints.1" Length="5"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromPrimitiv"
        Name="Dean_Integer" Inherits="//@URSs/@URSConstraints.0"/>
    <URSConstraints xsi:type="URSConstraint:UserDefDomFromUserDef"
        Name="UniCity_D_Name" Inherits="//@URSs/@URSConstraints.2"/>
```

```
  </URSs>
  <DBSchema Name="University">
    <RelationSchemes Name="University">
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                                AttributeName="//@URSs/@URSAttributes.6"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniName"
                                                AttributeName="//@URSs/@URSAttributes.3"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="UniCity"
                                                AttributeName="//@URSs/@URSAttributes.5"/>
    </RelationSchemes>
    <RelationSchemes Name="Faculty">
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                                AttributeName="//@URSs/@URSAttributes.6"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacId"
                                                AttributeName="//@URSs/@URSAttributes.4"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacShortName"
                                                AttributeName="//@URSs/@URSAttributes.0"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacName"
                                                AttributeName="//@URSs/@URSAttributes.1"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="Dean"
                                                AttributeName="//@URSs/@URSAttributes.2"/>
    </RelationSchemes>
  </DBSchema>
</ConstraintRDM:Project>
```

Listing 7.9. Model dela *UniversityDb* u skladu sa RMM u XML XMI formatu

7.2.2. Mapiranje koncepta Database u koncept RelationalDBSchema

U ovoj tački biće opisan algoritam procedure *KreiranjeDBSchema* koja će biti pozivana iz algoritma za mapiranje korenskih elemenata sa slike 7.5. Takođe, biće data i njena definicija pomoću ATL pravila. Ovom procedurom se mapira koncept baze podataka (**Database**) iz RSUBP meta-modela, na koncept relacione šeme baze podataka (**RelationalDBSchema**) iz Generičkog meta-modela. Na slici 7.8 prikazani su delovi meta-modela sa konceptima koji su obuhvaćeni procedurom, čiji pseudokod je prikazan na slici 7.9.



a) RDBMSMM b) RMM

Slika 7.8. Delovi meta-modela za mapiranje koncepta **Database** u koncept **RelationalDBSchema**

Procedura:	*KreiranjeDBSchema*
Formalni parametri:	
Ulazni:	*DB*(*TB, FK*)
Izlazni:	(*S, I*)
Lokalne deklaracije: *i* - izlazni parametar iz procedure *KreiranjeMeđurelacionihOgraničenja* (*i* – skup međurelacionih ograničenja *I*) *rs* – izlazni parametar iz procedure *KreiranjeŠemaRelacija* (*rs* – skup šema relacija)	
Pseudokod: **POČETAK PROCESA** KreirajRDBS **RADI** kreiranjeDBS **POSTAVI** *Naziv* ←*Naziv*(*DB*) **POZOVI** *KreiranjeŠemaRelacija*(*TB, rs*) **POSTAVI** *RelationSchemes* ← *rs* **POZOVI** *KreiranjeMeđurelacionihOgraničenja*(*DB, i*) **POSTAVI** *I* ← *i* **KRAJ RADI** kreiranjeDBS **KRAJ PROCESA** KreirajRDBS	

Slika 7.9. Algoritam procedure *KreiranjeDBSchema*

Naziv baze podatka postaje naziv relacione šeme baze podataka. Skup tabela baze podataka (*TB*) postaje skup šema relacija relacione šeme baze podataka. Algoritam procedure *KreiranjeŠemaRelacija* biće detaljno objašnjen u tački 7.2.3.

Pri specifikaciji meta-modela, u relacionim sistemima za upravljanje bazama podataka ograničenja stranog ključa (*FK*), koja pripadaju međurelacionim ograničenjima, tehnički se vezuju za odgovarajuću tabelu, dok međurelaciona ograničenja u RM meta-modelu pripadaju relacionoj šemi baze podataka. Procedura *KreiranjeMeđurelacionihOgraničenja* biće prikazana i objašnjena u tački 7.2.4.

Na listingu 7.10. prikazano je ATL pravilo za mapiranje koncepta baze podataka na koncept relacione šeme baze podataka. Ovo pravilo koristi pomoćne metode za pronalaženje svih ograničenja stranog ključa u šemi baze podataka, kao i moguća ograničenja inverznog referencijalnog integriteta, prikazane na listinzima 7.11 i 7.12, respektivno. Pronalaženje kandidata za ograničenje inverznog referencijalnog integriteta omogućava atribut *InverseReferentialIntegrityCon*. Detaljno objašnjenje ovog postupka biće dato u tački 7.2.4.

```
rule Database2RDBSchema {
from
   db : RDBMS!Database
to
   out : RM!RelationalDBSchema (
      Name <- db.Name,
      RelationSchemes<-db.Tables,
      MRConstraints <-
      db.getAllFKsForDB().union(thisModule.getCandidateForIRIC()->
                                             collect(e | thisModule.FK2IRIC(e)))
      )}
```

Listing 7.10. ATL pravilo za mapiranje koncepta **Database** u koncept **RelationalDBSchema**

```
helper context RDBMS!Database def: getAllFKsForDB() : Set(RDBMS!ForeignKey) =
   self.Tables -> collect(e | e.TableFKs);
```

Listing 7.11. ATL *heleper* za pronalaženje svih ograničenja stranog ključa u šemi baze podataka

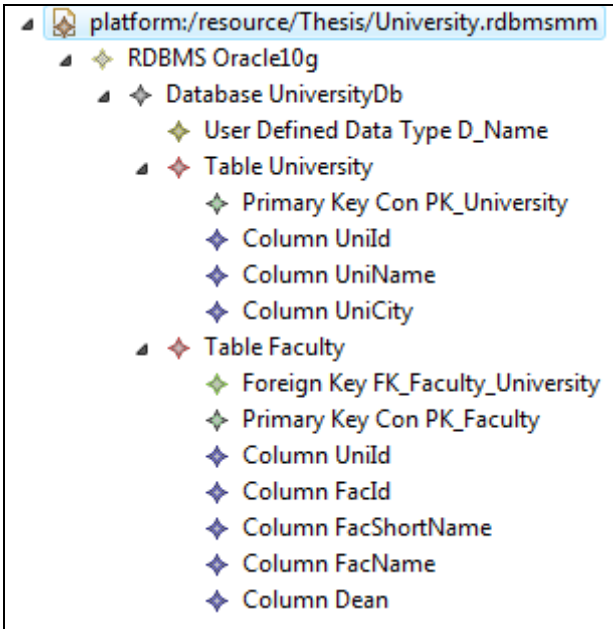
```
helper def: getCandidateForIRIC(): Set(RDBMS!ForeignKey) =
   RDBMS!ForeignKey.allInstances()->select(e| e.InverseReferentialIntegrityCon);
```

Listing 7.12. ATL *heleper* za pronalaženje ograničenja inverznog referencijalnog integriteta

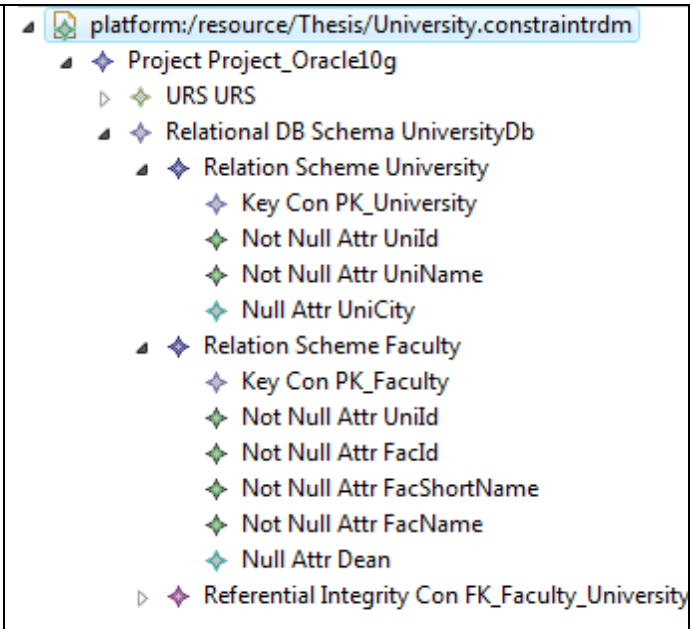
Primer 7.2. Šemama relacija datim u primeru 7.1 dodata su ograničenja primarnog ključa i ograničenje referencijalnog integriteta između šema relacija *Faculty* i *University*:

- *University*({*UniId*, *UniName*, *UniCity*}, {PrimaryKey(*UniId*)}),
- *Faculty*({*UniId*, *FacId*, *FacShortName*, *FacName*, *Dean*}, {PrimaryKey(*UniId*+*FacId*)}),
- *FK_ Faculty_University*: *Faculty*[*UniId*] ⊆ *University*[*UniId*].

Na slici 7.10 prikazan je segment šeme baze podataka koji ilustruje primenu prethodno navedenog pravila sa listinga 7.10. Ograničenje stranog ključa koje je pripadalo tabeli *Faculty* izvornog modela, prikazanog na slici 7.10 a), postaje međurelaciono ograničenje na nivou šeme baze podataka u ciljnom modelu, prikazanom na slici 7.10 b).



a) Model dela *UniversityDb* u skladu sa RDBMSMM



b) Model dela *UniversityDb* u skladu sa RMM

Slika 7.10. Deo *UniversityDb* relacione šeme baze podataka u Eclipse editoru

Na listinzima 7.13 i 7.14 prikazani su primeri sa slike 7.10 a) i b) u XML XMI formatu, respektivno.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdbmsmm:RDBMS xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:rdbmsmm="http://rdbmsmm/1.0" Name="Oracle10g">
  <Databases Name="UniversityDb">
    <UserDefinedDataTypes Name="D_Name" DataType="//@DataTypes.1" Length="15"
                                                                    DefaultValue=""/>
    <Tables Name="University">
      <TablePK Name="PK_University" PKandUQColumns="//@Databases.0/@Tables.0/@Columns.0"/>
      <Columns Name="UniId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="UniName" Length="20" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="UniCity" Nullable="true" Length="15"
                                                    ColumnDataType="//@DataTypes.1"/>
    </Tables>
    <Tables Name="Faculty">
      <TableFKs Name="FK_Faculty_University" RHS_Key=//@Databases.0/@Tables.0/@TablePK
      LHS_Attr="//@Databases.0/@Tables.1/@Columns.0" RefersTo="//@Databases.0/@Tables.0"/>
      <TablePK Name="PK_Faculty" PKandUQColumns="//@Databases.0/@Tables.1/@Columns.0
                                                  //@Databases.0/@Tables.1/@Columns.1"/>
      <Columns Name="UniId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="FacId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="FacShortName" Length="5" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="FacName" Length="20" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="Dean" Nullable="true" ColumnDataType="//@DataTypes.0"/>
    </Tables>
  </Databases>
  <DataTypes Name="Integer"/>
  <DataTypes Name="Varchar" PredefinedLength="256"/>
</rdbmsmm:RDBMS>
```

Listing 7.13. Model dela *UniversityDb* u skladu sa RDBMSMM u XML XMI formatu

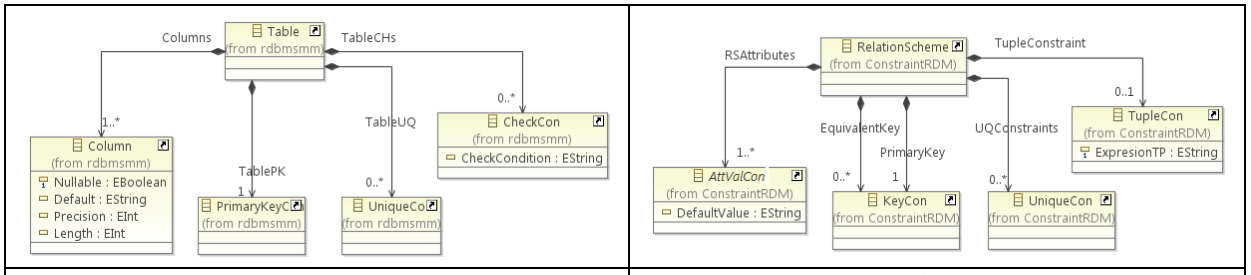
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConstraintRDM:Project xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ConstraintRDM="http://relacionimm/1.0/ConstraintRDM"
xmlns:INDConstraint="http://relacionimm/1.0/INDConstraint"
xmlns:RelationConstraint="http://relacionimm/1.0/RelationConstraint"
xmlns:URSConstraint="http://relacionimm/1.0/URSConstraint" Name="Project_Oracle10g">
  <URSs Name="URS">
    ...
  </URSs>
  <DBSchema Name="UniversityDb">
    <RelationSchemes Name="University">
      <PrimaryKey Name="PK_University"
                                KeyAttr="//@DBSchema/@RelationSchemes.0/@RSAttributes.0"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                                AttributeName="//@URSs/@URSAttributes.6"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniName"
                                                AttributeName="//@URSs/@URSAttributes.3"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="UniCity"
                                                AttributeName="//@URSs/@URSAttributes.5"/>
    </RelationSchemes>
    <RelationSchemes Name="Faculty">
      <PrimaryKey Name="PK_Faculty"
                        KeyAttr="//@DBSchema/@RelationSchemes.1/@RSAttributes.0
                                          //@DBSchema/@RelationSchemes.1/@RSAttributes.1"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                                AttributeName="//@URSs/@URSAttributes.6"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacId"
                                                AttributeName="//@URSs/@URSAttributes.4"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacShortName"
                                                AttributeName="//@URSs/@URSAttributes.0"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacName"
```

```
                                                AttributeName="//@URSs/@URSAttributes.1"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="Dean"
                                                AttributeName="//@URSs/@URSAttributes.2"/>
    </RelationSchemes>
    <MRConstraints xsi:type="INDConstraint:ReferentialIntegrityCon"
     Name="FK_Faculty_University" RHS_Key="//@DBSchema/@RelationSchemes.0/@PrimaryKey"
                        LHS_Attr_RIC="//@DBSchema/@RelationSchemes.1/@RSAttributes.0">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT"
                                          RHS_RS_IND="//@DBSchema/@RelationSchemes.0"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.1"/>
    </MRConstraints>
  </DBSchema>
</ConstraintRDM:Project>
```

Listing 7.14. Model dela *UniversityDb* u skladu sa RMM u XML XMI formatu

7.2.3. Mapiranje koncepta Table u koncept RelationScheme

U ovoj tački odeljka biće predstavljen algoritam procedure *KreiranjeŠemaRelacija*, koja se poziva iz procedure sa slike 7.9, kao i njegova implementacija. Procedura *KreiranjeŠemaRelacija* namenjena je za mapiranje koncepta **Table**, RSUBP meta-modela, u koncept **RelationScheme**, Generičkog meta-modela. Na slici 7.11 prikazani su delovi meta-modela sa konceptima koji su obuhvaćeni ovim algoritmom, čiji pseudokod je prikazan na slici 7.12.



a) RDBMSMM | b) RMM

Slika 7.11. Delovi meta-modela za mapiranje koncepta **Table** u koncept **RelationScheme**

Za svaku tabelu (T) baze podataka pri transformaciji kreira se šema relacije sa istim nazivom (N). Kolone tabele (C) postaju obeležja šeme relacije (R), a ograničenje primarnog ključa (PK^T) postaje ograničenje primarnog ključa šeme relacije (PK^{RS}).

Relacioni sistemi za upravljanje bazama podataka ne pružaju mehanizam eksplicitne specifikacije ograničenja ekvivalentnih ključeva, već samo ograničenja jedinstvenosti (UQ^T) pri čemu važi:

$$UQ^T = EK^T \cup UQ^{Tuq}.$$

Skup ograničenja ekvivalentnih ključeva (EK) je podskup skupa ograničenja jedinstvenosti koji se pri transformaciji mora izdvojiti kao poseban skup. Ukoliko su u ograničenju jedinstvenosti neke tabele, sve kolone koje pripadaju ograničenju, specificirane sa zabranom nula vrednosti, pri transformaciji za njega se kreira ograničenje ekvivalentnog ključa odgovarajuće šeme relacije (EK^{RS}).

UQ^{Tuq} predstavlja skup ograničenja jedinstvenosti u kojem je barem jedno obeležje specificirano sa dozvoljenom nula vrednošću.

Procedura:	*KreiranjeŠemaRelacija*
Formalni parametri:	
Ulazni:	$TB = \{ T_i(C_i, O^T_i) \mid i = 1,.., n\}$ $O^T = PK^T \cup UQ^T \cup CH^T$
Izlazni:	$S = \{ N_i(R_i, O^{RS}_i) \mid i = 1,.., n\}$ $O^{RS} = PK^{RS} \cup EK^{RS} \cup UQ^{RS} \cup CH^{RS}$

Lokalne deklaracije:
ind - izlazni parametar iz procedure *ProveravanjeAtributaUQ*
(*ind* = 1 – UQ^T pripada ekvivalentnim ključevima,
ind = 0 – UQ^T pripada ograničenjima jedinstvenosti)
ch - izlazni parametar iz procedure *KreiranjeOgraničenjaTorke*
(*ch* – ograničenje torke)

Pseudokod:

```
POČETAK PROCESA KreirajRS
   RADI kreiranjeR (∀T ∈ TB)
      POSTAVI N ← T
      POSTAVI R ← C
      POSTAVI PK^RS ← PK^T
      RADI kreiranjeEK (∀ uq ∈ UQ^T)
         POZOVI ProveravanjeAtributaUQ(C^uq, ind)
         AKO JE ind = 0 TADA
            POSTAVI UQ^RS ← UQ^RS ∪ uq
         INAČE
            POSTAVI EK^RS ← EK^RS ∪ uq
         KRAJ AKO
      KRAJ RADI kreiranjeEK
      POZOVI KreiranjeOgraničenjaTorke(CH^T, ch)
      POSTAVI CH^RS ← ch
   KRAJ RADI kreiranjeR
KRAJ PROCESA KreirajRS
```

Slika 7.12. Algoritam za mapiranje koncepta **Table** u koncept **RelationScheme**

Za svako ograničenje jedinstvenosti ($uq \in UQ^T$) u kojem je barem jedno obeležje specificirano sa dozvoljenom nula vrednošću kreira se ograničenje jedinstvenosti odgovarajuće šeme relacije (UQ^{RS}). Proveru da li ograničenje jedinstvenosti zadovoljava uslov da svi njegovi atributi moraju biti različiti od nula vrednosti vrši procedura *ProveravanjeAtributaUQ*, čiji pseudokod je prikazan na slici 7.13. Ulazni parametar procedure je skup kolona tabele nad kojim je specificirano ograničenje jedinstvenosti (C^{uq}).

Za sva ograničenja važi da ograničenje šeme relacije dobija naziv ograničenja tabele.

Za razliku od relacionih sistema za upravljanje bazama podataka koji omogućavaju korisniku da definiše više *check* ograničenja nad jednom tabelom u Generičkom meta-modelu je ograničeno da šema relacije može imati samo jedno ograničenje torke, ali koje može imati više uslova. Uslovi se spajaju logičkim operatorom AND. U procesu transformacije skup uslova svih *check* ograničenja jedne tabele se povezuje u jedan uslov koji se pridružuje odgovarajućoj šemi relacije. Pseudokod procedure *KreiranjeOgraničenjaTorke*, koja kreira ograničenje torke prikazan je na slici 7.14.

Ukoliko je nad tabelom definisano samo jedno *check* ograničenje, pri transformaciji naziv i uslov tog ograničenja postaje naziv i uslov ograničenja torke odgovarajuće šeme relacije.

Ukoliko je nad tabelom definisano više *check* ograničenja, naziv prvog postaje naziv ograničenja torke, a uslov se formira na prethodno opisan način.

Procedura:	*ProveravanjeAtributaUQ*
Formalni parametri:	
Ulazni:	$C^{uq} = \{C_1, .., C_k\}$, $i = 1..k$
Izlazni:	*ind*
Pseudokod:	

```
POČETAK PROCESA ProveravanjeAtributaUQ
    POSTAVI ind ← 1
    RADI proveru (∀C ∈ C^uq DOK JE ind = 1)
      AKO JE C = ω TADA
        POSTAVI ind ← 0
      KRAJ AKO
    KRAJ RADI proveru
KRAJ PROCESA ProveravanjeAtributaUQ
```

Slika 7.13. Algoritam procedure *ProveravanjeAtributaUQ*

Procedura:	*KreiranjeOgraničenjaTorke*
Formalni parametri:	
Ulazni:	*CH*
Izlazni:	*ch*
Pseudokod:	

```
POČETAK PROCESA KreiranjeOgraničenjaTorke
    POSTAVI ch.ime ← CH1.ime
    AKO JE CH.size = 1 TADA
      POSTAVI ch.uslov ← CH1. uslov
    INAČE
      RADI kreirajUslov (∀c ∈ CH)
        POSTAVI ch.uslov ← ch.uslov + 'AND' + c.uslov
      KRAJ RADI kreirajUslov
    KRAJ AKO
KRAJ PROCESA KreiranjeOgraničenjaTorke
```

Slika 7.14. Algoritam procedure *KreiranjeOgraničenjaTorke*

Na listingu 7.14 prikazana je ATL specifikacija pravila *Table2RScheme*, za mapiranje koncepta **Table** u koncept **RelationScheme**, opisana prethodnim algoritmima. Ovo pravilo pokreće čitav niz ATL pravila kojima se implementira mapiranje svakog pojedinačnog koncepta koji pripada šemi relacije: koncepta atributa, ograničenja primarnog ključa, ograničenja ekvivalentnog ključa, ograničenja jedinstvenosti i ograničenja torke.

```
rule Table2RScheme {
from
   tb : RDBMS!Table
```

```
to
   out : RM!RelationScheme (
      Name <- tb.Name,
      RSAttributes <- tb.Columns,
      PrimaryKey <- tb.TablePK,
      EquivalentKey <- tb.TableUQ -> select(e|not e.PKandUQColumns ->
                                                        exists(c | c.Nullable)),
      UQConstraints <- tb.TableUQ -> select(e|e.PKandUQColumns ->
                                                        exists(c | c.Nullable)),
      TupleConstraint <-if not tb.TableCHs.oclIsUndefined() then
          tb.TableCHs-> collect(e| thisModule.getTableCHs(tb,e)).first()
          else OclUndefined endif
      )
}
```

Listing 7.14. ATL pravilo za mapiranje koncepta **Table** u koncept **RelationScheme**

Na listinzima 7.15 i 7.16 prikazana su ATL pravila za mapiranje kolone u atribut koji pripada odgovarajućoj šemi relacije. U Generičkom meta-modelu ograničenje nula vrednosti je implementirano tako što je skup atributa neke šeme relacije podeljen na skup atributa kojima je dozvoljena nula vrednost i skup atributa kojima je nula vrednost zabranjena (detaljno je objašnjeno u 6.3.2). Iz tog razloga postoje dva pravila za transformaciju kolona tabele: *Column2NullAttr* i *Column2NotNullAtt*. Oba pravila pozivaju ATL pomoćnu metodu *isNullable*, za proveru ograničenja nula vrednosti atributa. Ova metoda je prikazana na listingu 7.17.

```
rule Column2NullAttr {
from
   c : RDBMS!Column (c.isNullable())
to
   out : RM!NullAttr (
      Name <- c.Name,
      DefaultValue <- c.Default,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION'
   )
   do{out.AttributeName <- thisModule.getAttributeName(c.Name);}
}
```

Listing 7.15. ATL pravilo za mapiranje koncepta **Column** u koncept **NullAttr**

```
rule Column2NotNullAtt {
from
   c : RDBMS!Column (not c.isNullable())
to
   out : RM!NotNullAttr (
      Name <- c.Name,
      DefaultValue <- c.Default,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION'
   )
   do{out.AttributeName <- thisModule.getAttributeName(c.Name);}
}
```

Listing 7.16. ATL pravilo za mapiranje koncepta **Column** u koncept **NotNullAttr**

```
helper context RDBMS!Column def: isNullable() : Boolean =
   if self.Nullable then true else false endif;
```

Listing 7.17. ATL *helper* za proveru ograničenja nula vrednosti atributa

Zbog postojanja šeme univezalne relacije u Generičkom meta-modelu, šemi relacije se dodeljuje atribut iz skupa atributa *U*. Kako koncept URS-e nije postojao u izvornom meta-modelu već je dobijen transformacijom, zbog mogućnosti ATL jezika, ovo dodeljivanje atributa implementirano je pomoćnom metodom koja se poziva u proceduralnom bloku. ATL pomoćna metoda za povezivanje atributa iz URS-e sa odgovarajućom šemom relacije prikazana je na listingu 7.18.

```
helper def: getAttributeName(n:String): RM!Attribute =
   RM!Attribute.allInstances() -> select(e | e.refGetValue('Name')=n).first();
```

Listing 7.18. ATL *helper* za povezivanje atributa iz URS-e sa odgovarajućom šemom relacije

Na listinzima od 7.19 do 7.24 prikazana su ATL pravila za mapiranje svakog pojedninačnog relacionog ograničenja. Za ograničenja primarnog ključa i ograničenja jedinstvenosti mapiranje je 1-1, pošto se radi o potpuno istim konceptima u oba meta-modela. Kod ograničenja torke postoji već objašnjena razlika, zbog čega se logički uslov formira od skupa logičkih uslova *check* ograničenja.

Za sva relaciona ograničenja važi sledeće:

- naziv ograničenja tabele postaje naziv ograničenja šeme relacije,
- skup kolona ograničenja tabele postaje skup atributa ograničenja šeme relacije,
- podrazumevana vrednost za aktivnost koja se sprovodi pri kritičnim operacijama upisa i izmene torke u šemi relacije, inicijalizuje se na NO ACTION i
- specifikacija trenutka provere ograničenja tabele postaje specifikacija trenutka provere ograničenja šeme relacije.

NO ACTION je podrazumevana vrednost za RSUBP-ove, ali se ne može pročitati direktno iz rečnika podataka, zbog čega je u ATL specifikaciji dodeljena kao konstanta.

```
rule PK2KeyCon{
from
   kc : RDBMS!PrimaryKeyCon
to
   out : RM!KeyCon (
      Name <- kc.Name,
      KeyAttr <- kc.PKandUQColumns,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION',
      Deferrability <- kc.Deferrable,
      InitiallyDefer <- kc.Deferred
   )
}
```

Listing 7.19. ATL pravilo za mapiranje koncepta **PrimaryKeyCon** u **KeyCon**

```
rule UQ2KeyCon{
from
   uc : RDBMS!UniqueCon (not uc.hasNull())
to
```

```
   out : RM!KeyCon (
      Name <- uc.Name,
      KeyAttr <- uc.PKandUQColumns,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION',
      Deferrability <- uc.Deferrable,
      InitiallyDefer <- uc.Deferred
   )
}
```

Listing 7.20. ATL pravilo za mapiranje koncepta **UniqueCon** u **KeyCon**

```
rule UQ2UniqueCon{
from
   uc : RDBMS!UniqueCon (uc.hasNull())
to
   out : RM!UniqueCon (
      Name <- uc.Name,
      UQConNotNullAttr <- uc.PKandUQColumns -> select(e |not e.Nullable),
      UQConNullAttr <- uc.PKandUQColumns -> select(e | e.Nullable),
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION',
      Deferrability <- uc.Deferrable,
      InitiallyDefer <- uc.Deferred
   )}
```

Listing 7.21. ATL pravilo za mapiranje koncepta **UniqueCon** u **UniqueCon**

```
helper context RDBMS!UniqueCon def: hasNull() : Boolean =
            self.PKandUQColumns -> exists(e|e.Nullable);
```

Listing 7.22. ATL *helper* za proveru ograničenja nula vrednosti obeležja koja pripadaju ograničenju jedinstvenosti

```
lazy rule getTableCHs {
from
   tb : RDBMS!Table,
   ch : RDBMS!CheckCon
to
   rel: RM!TupleCon(
      Name <- tb.TableCHs-> collect(e |
                                    e.refGetValue('Name').toString()).first(),
      ExpresionTP <- tb.TupleConstraintExpresion,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION',
      Deferrability <- ch.Deferrable,
      InitiallyDefer <- ch.Deferred
   )
}
```

Listing 7.23. ATL *lazy* pravilo za mapiranje koncepta **CheckCon** u **TupleCon**

```
helper context RDBMS!Table def:TupleConstraintExpresion:String=
let allCheckExpresion:Sequence(String)= self.TableCHs-> collect(e |
                                    e.refGetValue('CheckCondition').toString())
in allCheckExpresion->iterate(e; res:String=allCheckExpresion.first().toString()|
    if (allCheckExpresion.size()=1) then res else
            if res=allCheckExpresion.first().toString() then '(' + e + ')' else
```

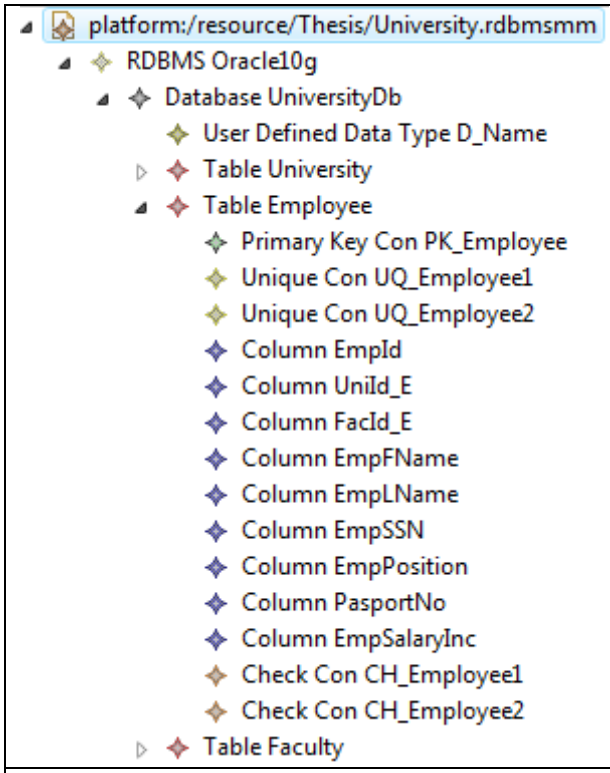
```
                res + ' and (' + e + ')' endif
    endif);
```

Listing 7.24. ATL *helper* za kreiranje logičkog uslova ograničenja torke

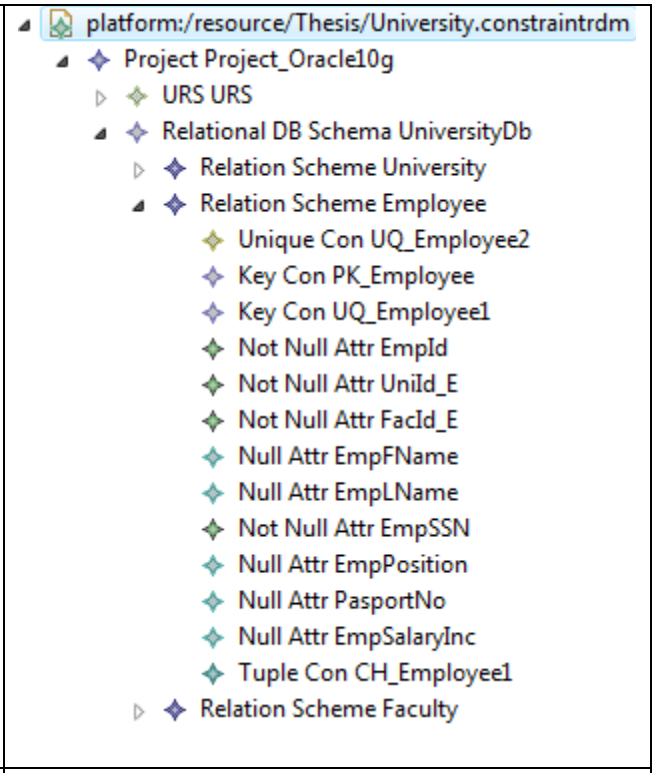
Primer 7.3. Data je šema relacije *Employee* sa skupom relacionih ograničenja koji čine: primarni ključ, ekvivalentni ključ, ograničenje jedinstvenosti i dva *check* ograničenja.

Employee({*EmpId*, *UniId*, *FacId*, *EmpFName*, *EmpLName*, *EmpSSN*, *EmpPosition*, *PasportNo*, *EmpSalaryInc*}, {PrimaryKey(*UniId* + *FacId* + *EmpId*), EquivalentKey(*UniId* + *FacId* + *EmpSSN*), Unique(*PasportNo*), CheckCon(*EmpSalaryInc* > 0), CheckCon(*EmpPosition* = 'Prof' ∨ *EmpPosition* = 'Assistent')}).

Na slici 7.15 prikazan je segment šeme baze podataka koji ilustruje primenu prethodno opisanih pravila transformacije. U izvornom modelu prikazanom na slici 7.15 a) za tabelu *Employee* postoje dva ograničenja jedinstvenosti: *UQ_Employee1* i *UQ_Employee2* od kojih transformacijom prvo postaje ograničenje ekvivalentnog ključa, a drugo ograničenje jedinstvenosti. Od dva *check* ograničenja koja tabela *Employee* ima: *CH_Employee1* i *CH_Employee2*, prikazana na slici 7.16 a), nakon transformacije nastaje jedno ograničenje torke *CH_Employee1* sa spojenim logičkim uslovima, prikazano na slici 7.16 b).

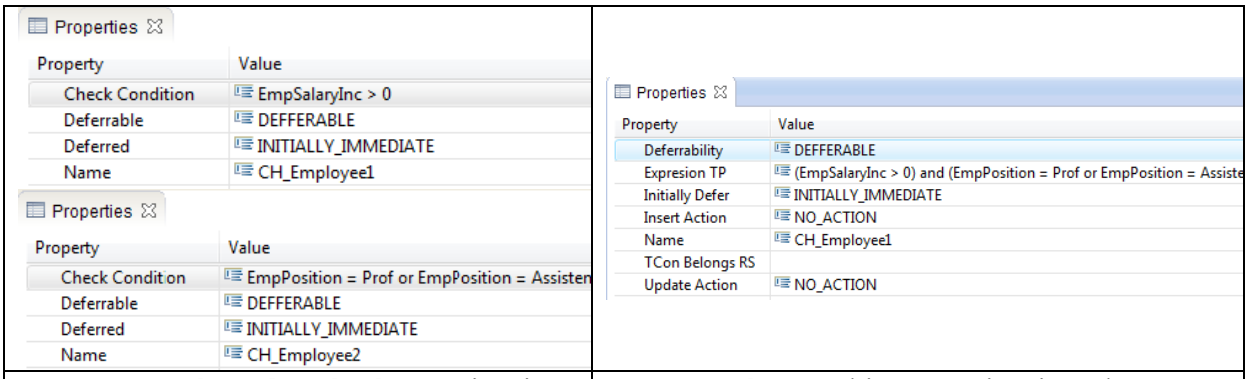


a) Model dela *UniversityDb* u skladu sa RDBMSMM



b) Model dela *UniversityDb* u skladu sa RMM

Slika 7.15. Deo *UniversityDb* relacione šeme baze podataka u Eclipse editoru



a) Osobine dva *check* ograničenja izvornog modela

b) Osobine ograničenja torke ciljnog modela

Slika 7.16. Deo *UniversityDb* relacione šeme baze podataka u Eclipse editoru

Na listinzima 7.25 i 7.26 prikazani su primeri sa slike 7.15 a) i b) u XML XMI formatu, respektivno.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdbmsmm:RDBMS xmi:version="2.0" xmlns:xmi=http://www.omg.org/XMI
                                  xmlns:rdbmsmm="http://rdbmsmm/1.0" Name="Oracle10g">
  <Databases Name="UniversityDb">
    <UserDefinedDataTypes Name="D_Name" DataType="//@DataTypes.2" Length="15"
                                                                  DefaultValue=""/>
    <Tables Name="University">
      ...
    </Tables>
    <Tables Name="Employee">
      <TablePK Name="PK_Employee" PKandUQColumns="//@Databases.0/@Tables.1/@Columns.1
            //@Databases.0/@Tables.1/@Columns.2 //@Databases.0/@Tables.1/@Columns.0"/>
      <TableUQ Name="UQ_Employee1" PKandUQColumns="//@Databases.0/@Tables.1/@Columns.1
            //@Databases.0/@Tables.1/@Columns.2 //@Databases.0/@Tables.1/@Columns.5"/>
      <TableUQ Name="UQ_Employee2" PKandUQColumns="//@Databases.0/@Tables.1/@Columns.7"/>
      <Columns Name="EmpId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="UniId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="FacId" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="EmpFName" Nullable="true" Length="20"
                                          ColumnDataType="//@DataTypes.2"/>
      <Columns Name="EmpLName" Nullable="true" Length="25"
                                          ColumnDataType="//@DataTypes.2"/>
      <Columns Name="EmpSSN" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="EmpPosition" Nullable="true" Length="10"
                                          ColumnDataType="//@DataTypes.2"/>
      <Columns Name="PasportNo" Nullable="true" Length="10"
                                          ColumnDataType="//@DataTypes.0"/>
      <Columns Name="EmpSalaryInc" Nullable="true" ColumnDataType="//@DataTypes.1"/>
      <TableCHs Name="CH_Employee1" CheckCondition="EmpSalaryInc > 0"/>
      <TableCHs Name="CH_Employee2" CheckCondition="EmpPosition = 'Prof' or
                                                    EmpPosition = 'Assistent'"/>
    </Tables>
    <Tables Name="Faculty">
    ...
    </Tables>
  </Databases>
  <DataTypes Name="Integer"/>
  <DataTypes Name="Decimal"/>
  <DataTypes Name="Varchar" PredefinedLength="256"/>
</rdbmsmm:RDBMS>
```

Listing 7.25. Model dela *UniversityDb* u skladu sa RDBMSMM u XML XMI formatu

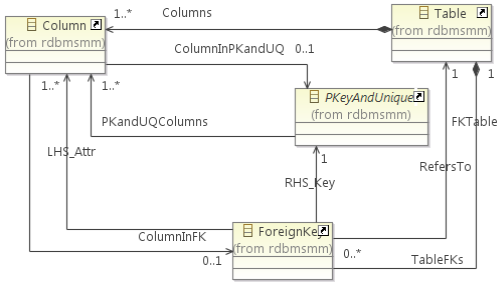
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConstraintRDM:Project xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ConstraintRDM="http://relacionimm/1.0/ConstraintRDM"
xmlns:INDConstraint="http://relacionimm/1.0/INDConstraint"
xmlns:RelationConstraint="http://relacionimm/1.0/RelationConstraint"
xmlns:URSConstraint="http://relacionimm/1.0/URSConstraint" Name="Project_Oracle10g">
  <URSs Name="URS">
    ...
  </URSs>
  <DBSchema Name="UniversityDb">
    <RelationSchemes Name="University">
    ...
    </RelationSchemes>
    <RelationSchemes Name="Employee">
      <UQConstraints Name="UQ_Employee2" UQConNullAttr=
                                  "//@DBSchema/@RelationSchemes.1/@RSAttributes.7"/>
      <PrimaryKey Name="PK_Employee" KeyAttr=
            "//@DBSchema/@RelationSchemes.1/@RSAttributes.1
            //@DBSchema/@RelationSchemes.1/@RSAttributes.2
            //@DBSchema/@RelationSchemes.1/@RSAttributes.0"/>
      <EquivalentKey Name="UQ_Employee1" KeyAttr=
            "//@DBSchema/@RelationSchemes.1/@RSAttributes.1
            //@DBSchema/@RelationSchemes.1/@RSAttributes.2
            //@DBSchema/@RelationSchemes.1/@RSAttributes.5"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="EmpId"
                                   AttributeName="//@URSs/@URSAttributes.5"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                  AttributeName="//@URSs/@URSAttributes.13"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacId"
                                  AttributeName="//@URSs/@URSAttributes.9"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="EmpFName"
                                  AttributeName="//@URSs/@URSAttributes.1"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="EmpLName"
                                  AttributeName="//@URSs/@URSAttributes.12"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="EmpSSN"
                                  AttributeName="//@URSs/@URSAttributes.8"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="EmpPosition"
                                  AttributeName="//@URSs/@URSAttributes.3"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="PasportNo"
                                  AttributeName="//@URSs/@URSAttributes.14"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="EmpSalaryInc"
                                  AttributeName="//@URSs/@URSAttributes.2"/>
      <TupleConstraint Name="CH_Employee1" ExpresionTP="(EmpSalaryInc > 0) and
                                  (EmpPosition = 'Prof' or EmpPosition = 'Assistent')"/>
    </RelationSchemes>
    <RelationSchemes Name="Faculty">
    ...
    </RelationSchemes>
  </DBSchema>
</ConstraintRDM:Project>
```

Listing 7.26. Model dela *UniversityDb* u skladu sa RMM u XML XMI formatu

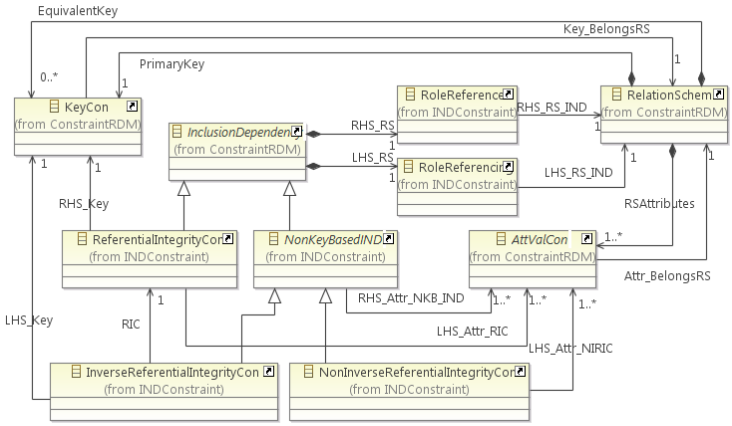
7.2.4. Mapiranje koncepta ForeignKey u koncepte međurelacionih ograničenja

U ovoj tački odeljka biće predstavljen algoritam za mapiranje koncepta **ForeignKey**, RSUBP meta-modela, u sledeće koncepte međurelacionih ograničenja: ograničenje referencijalnog integriteta (**ReferentialIntegrityCon**), ograničenje inverznog referencijalnog

integriteta (**InverseReferentialIntegrityCon**) i ostala *non-key based* ograničenja (**NonInverseReferential IntegrityCon)**, Generičkog meta-modela i njihova implementacija. Na slici 7.17 prikazani su delovi meta-modela koji obuhvataju ove koncepte, dok je na slici 7.18 prikazan pseudokod algoritma *KreiranjeMeđurelacionihOgraničenja*, koji se poziva iz procedure prikazane na slici 7.9.



a) RDBMS meta-model



b) RM meta-model

Slika 7.17. Delovi meta-modela za mapiranje koncepta **ForeignKey** u koncepte međurelacionih ograničenja

Pri implementaciji na konkretnim RSUBP-ovima, međurelaciona ograničenja tipa zavisnosti sadržavanja (ograničenje stranog ključa, *fk*: $T_l[LHS] \subseteq T_r[RHS]$), vezuju se za tabelu koja se nalazi sa leve strane ograničenja, odnosno referencirajuću tabelu. Takođe, RSUBP-ovi omogućavaju da se na desnoj strani ograničenja stranog ključa može naći:

- primarni ključ tabele (*RHS* je primarni ključ tabele T_r),
- skup atributa nad kojim je definisano ograničenje jedinstvenosti i u kojem su sva obeležja sa specificiranom zabranom nula vrednosti, (*RHS* je ekvivalnetni ključ tabele T_r) ili
- skup atributa nad kojim je definisano ograničenje jedinstvenosti i u kojem je barem jedno obeležje specificirano sa dozvoljenom nula vrednošću.

U zavisnosti od toga šta se od prethodno navedenog nalazi na desnoj strani ograničenja, nakon transformacije se dobija sledeće:

- ograničenje referencijalnog integriteta kada je na desnoj strani ključ (primarni ili ekvivalentni) ili
- *non-key based* ograničenje koje nije ograničenje inverznog referencijalnog integriteta.

Da bi se napravila diferencijacija ograničenja potrebno je izdvojiti ograničenja stranog ključa koja sa desne strane imaju ključ. Pseudokod procedure *ProveravanjeDesneStrane*, koja radi proveru da li je na desnoj strani ograničenja ključ, prikazan je na slici 7.19.

Procedura:	*KreiranjeMeđurelacionihOgraničenja*
Formalni parametri:	
Ulazni:	*DB*(*TB*,*FK*)
Izlazni:	*I*
Lokalne deklaracije: *ind* - izlazni parametar iz procedure *ProveravanjeDesneStrane* (*ind* = 1 – na desnoj strani je ključ, *ind* = 0 – na desnoj strani nije ključ)	
Pseudokod: **POČETAK PROCESA** KreiranjeMeđurelacionihOgraničenja **RADI** kreiranje (∀*fk* ∈ *FK*) **POZOVI** *ProveravanjeDesneStrane*(*fk*, *ind*) ***AKO JE*** *ind* = 1 **TADA** **POZOVI** *KreiranjeRIC*(*fk*, *i*) **POSTAVI** *I* ← *I* ∪ *i* ***AKO JE*** *InverseReferentialIntegrityCon* = *true* **TADA** **POZOVI** *KreiranjeIRIC*(*fk*, *i*) **POSTAVI** *I* ← *I* ∪ *i* ***KRAJ AKO*** **INAČE** **POZOVI** *KreiranjeNonIRIC*(*fk*, *i*) **POSTAVI** *I* ← *I* ∪ *i* ***KRAJ AKO*** **KRAJ RADI** kreiranje **KRAJ PROCESA** KreiranjeMeđurelacionihOgraničenja	

Slika 7.18. Algoritam procedure *KreiranjeMeđurelacionihOgraničenja*

Procedura:	*ProveravanjeDesneStrane*
Formalni parametri:	
Ulazni:	*fk*
Izlazni:	*ind*
Pseudokod: **POČETAK PROCESA** ProveravanjeDesneStrane **POSTAVI** *ind* ← *1* **RADI** proveru (∀*A* ∈ *RHS* **DOK JE** *ind* = 1) ***AKO JE*** *A* = ω **TADA** **POSTAVI** *ind* ← 0 ***KRAJ AKO*** **KRAJ RADI** proveru **KRAJ PROCESA** ProveravanjeDesneStrane	

Slika 7.19. Algoritam procedure *ProveravanjeDesneStrane*

Ulazni parametar procedure *ProveravanjeDesneStrane* je ograničenje stranog ključa za koje se proverava da li skup atributa *RHS* koji se nalazi na desnoj strani ograničenja sadrži barem jedno obeležje za koje je dozvoljena nula vrednost. Ako sadrži, procedura vraća

indikator *ind* sa vrednošću nula što označava da se na desnoj strani ograničenja ne nalazi ključ. U suprotnom, ako su sva obeležja iz *RHS* sa specificiranom zabranom nula vrednosti procedura vraća indikator *ind* sa vrednošću jedan, što znači da je na desnoj strani ograničenja ključ.

Ukoliko se sa desne strane ograničenja stranog ključa nalazi ključ tabele (primarni ili ekvivalentni), od ovog ograničenja putem transformacije nastaje ograničenje referencijalnog integriteta koje se kreira pozivom procedure *KreiranjeRIC*.

Takođe, za svako ograničenje stranog ključa proverava se postojanje ograničenja inverznog referencijalnog integriteta. Provera mogućeg prisustva ograničenja inverznog referencijalnog integrtiteta omogućena je pomoću atributa *InverseReferentialIntegrityCon*, kojim je proširen RSUBP meta-model i predstavlja dopunu meta-podacima pročitanim iz rečnika podataka SUBP-a. Vrednost ovog atributa (*true* ili *false*) se dobija iz samih podataka koji se nalaze u bazi, pošto RSUBP-ovi ne omogućavaju deklarativnu specifikaciju ograničenja inverznog referencijalnog integriteta. Takođe, treba napomenuti da je postojanje ograničenja inverznog referencijalnog integriteta samo pretpostavka. Detaljno objašnjenje kako se dolazi do predloga za postojanje ograničenja inverznog referencijalnog integriteta dato je u odeljku 5.5. Ukoliko je vrednost atributa *InverseReferentialIntegrityCon* jednako *true*, od ograničenja stranog ključa, pored ograničenja referencijalnog integriteta, nastaje i ograničenje inverznog referencijalnog integriteta. Za kreiranje ograničenja inverznog referencijalnog integriteta poziva se procedura *KreiranjeIRIC*.

Ukoliko se sa desne strane ograničenja ne nalazi ključ referencirane tabele, kreiraju se ostala *non-key based* ograničenja pozivom procedure *KreiranjeNonIRIC*.

Mapiranje koncepta stranog ključa na neko od pomenuta tri ograničenja zavisnosti sadržavanja je 1-1 zbog čega je za sve tri procedure *KreiranjeRIC*, *KreiranjeNonIRIC* i *KreiranjeIRIC* izostavljen pseudokod i date su samo ATL specifikacije pravila mapiranja prikazane na listinzima 7.27, 7.28 i 7.29 , respektivno. ATL pomoćna metoda za proveru da li se na desnoj strani ograničenja stranog ključa nalazi ključ referencirane tabele prikazana je na listingu 7.30.

```
rule FK2RIC{
from
   fk: RDBMS!ForeignKey (fk.RHS_Key.isKey())
to
   out: RM!ReferentialIntegrityCon (
      Name <- fk.Name,
      LHS_Attr_RIC <- fk.LHS_Attr,
      RHS_Key <- fk.RHS_Key,
      RHS_RS <- roleRHS,
      LHS_RS <- roleLHS,
      Type <- fk.Match,
      Deferrability <- fk.Deferrable,
      InitiallyDefer <- fk.Deferred
   ),
   roleRHS: RM!RoleReferenced (
      RHS_RS_IND <- fk.RefersTo,
      DeleteAction <- fk.DeleteActionRHS,
      UpdateAction <- fk.UpdateActionRHS
   ),
   roleLHS: RM!RoleReferencing (
      LHS_RS_IND <- fk.FKTable,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION'   )}
```

Listing 7.27. ATL pravilo za mapiranje koncepta **ForeignKey** u koncept **ReferentialIntegrityCon**

Za mapiranje ograničenja stranog ključa u ograničenja zavisnosti sadržavanja tipa ograničenja referencijalnog integriteta i *non-key based* ograničenja koja nisu ograničenja inverznog referencijalnog integriteta, važi sledeće:

- naziv ograničenja stranog ključa postaje naziv ograničenja zavisnosti sadržavnja (*Name*),
- tabela vlasnik ograničenja postaje šema relacije na levoj strani ograničenja (*LHS_RS*),
- referencirana tabela postaje šema relacije na desnoj strani ograničenja (*RHS_RS*),
- niz kolona koje sadrži strani ključ postaje niz atributa koji se nalaze na levoj strani ograničenja (*LHS_Attr_RIC* ili *LHS_Attr_NIRIC*),
- ključ ili jedinstvena vrednost referencirane tabele postaje ključ ili skup atributa sa desne strane ograničenja (*RHS_Key za RIC* ili *RHS_Attr_NKB_IND*),
- aktivnost, koja se sprovodi u slučaju narušavanja ograničenja stranog ključa referencijalnog integriteta pri brisanju, ili modifikaciji podataka referencirane tabele postaje aktivnost koja se sprovodi u slučaju narušavanja nekog od pomenuta dva ograničenja za iste operacije (*DeleteAction*, *UpdateAction*),
- tip referenciranja se mapira na isti koncept ciljnog meta-modela (*Type*),
- specifikacija trenutka provere ograničenja se mapira na isti koncept ciljnog meta-modela (*Deferrability*, *InitiallyDefer*).

Specifikacija šema relacija sa leve i desne strane ograničenja u ciljnom meta-modelu vrši se preko koncepta uloge (*roleRHS* i *roleLHS*). Na desnoj strani je šema relacije referencirana, a na levoj referencirajuća. RSUBP-ovi ne podržavaju specifikaciju različitih aktivnosti koje se sprovode u slučaju narušavanja ograničenja pri upisu, ili modifikaciji podataka u referencirajućoj tabeli. Podrazumevana vrednost je sprečavanje zbog čega se pri transformaciji za kritične operacije *insert* i *update* aktivnost specificira na vrednost NO ACTION.

```
rule FK2NonIRIC{
from
   fk: RDBMS!ForeignKey(not fk.RHS_Key.isKey())
to
   out: RM!NonInverseReferentialIntegrityCon (
      Name <- fk.Name,
      LHS_Attr_NIRIC <- fk.LHS_Attr,
      RHS_Attr_NKB_IND <- fk.RHS_Key.PKandUQColumns,
      RHS_RS <- roleRHS,
      LHS_RS <- roleLHS,
      Type <- fk.Match,
      Deferrability <- fk.Deferrable,
      InitiallyDefer <- fk.Deferred
   ),
   roleRHS: RM!RoleReferenced (
      RHS_RS_IND <- fk.RefersTo,
      DeleteAction <- fk.DeleteActionRHS,
      UpdateAction <- fk.UpdateActionRHS
   ),
   roleLHS: RM!RoleReferencing (
      LHS_RS_IND <- fk.FKTable,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION'
   )
}
```

Listing 7.28. ATL pravilo za mapiranje koncepta **ForeignKey** u koncept **NonInverseReferentialIntegrityCon**

Za mapiranje ograničenja stranog ključa u ograničenje zavisnosti sadržavanja tipa ograničenja inverznog referencijalnog integriteta važi sledeće:

- naziv ograničenja (*Name*) dobija naziv ograničenja stranog ključa sa prefiksom IRI_,
- tabela vlasnik ograničenja stranog ključa postaje šema relacije na desnoj strani ograničenja (*RHS_RS*),
- referencirana tabela postaje šema relacije na levoj strani ograničenja (*LHS_RS*),
- niz kolona koje sadrži strani ključ postaje niz atributa koji se nalaze na desnoj strani ograničenja (*RHS_Attr_NKB_IND*),
- ključ ili jedinstvena vrednost referencirane tabele postaje ključ ili skup atributa sa leve strane ograničenja (*LHS_Key*),
- aktivnost koja se sprovodi u slučaju narušavanja ograničenja, tip referenciranja i specifikacija trenutka provere ograničenja se mapira na isti način objašnjen za prethodna dva ograničenja.

Zahtev za postojanjem odgovarajućeg ograničenja referencijalnog integriteta (*RIC*) u ovom slučaju je automatski obezbeđeno, jer ograničenje inverznog referencijalnog integriteta upravo i nastaje od ograničenja stranog ključa koje predstavlja ograničenje referencijalnog integriteta.

```
unique lazy rule FK2IRIC{
from
   fk: RDBMS!ForeignKey
to
   iric: RM!InverseReferentialIntegrityCon(
      Name <- 'IRI_' + fk.Name,
      LHS_Key <- fk.RHS_Key,
      RHS_Attr_NKB_IND <- fk.LHS_Attr,
      RHS_RS <- roleRHS,
      LHS_RS <- roleLHS,
      Type <- fk.Match,
      Deferrability <- fk.Deferrable,
      InitiallyDefer <- fk.Deferred,
      RIC <- fk
   ),
   roleRHS: RM!RoleReferenced (
      RHS_RS_IND <- fk.FKTable,
      DeleteAction <- fk.DeleteActionRHS,
      UpdateAction <- fk.UpdateActionRHS
   ),
   roleLHS: RM!RoleReferencing (
      LHS_RS_IND <- fk.RefersTo,
      InsertAction <- 'NO_ACTION',
      UpdateAction <- 'NO_ACTION'
   )
}
```

Listing 7.29. ATL pravilo za mapiranje koncepta **ForeignKey** u koncept **InverseReferentialIntegrityCon**

```
helper context RDBMS!PKeyAndUnique def: isKey() : Boolean =
            self.PKandUQColumns -> exists(e|e.Nullable=false);
```

Listing 7.30. ATL *helper* za proveru da li se na desnoj strani ograničenja stranog ključa nalazi ključ referencirane tabele

Primer 7.4. Skup šema relacija iz prethodnih primera proširen je šemom relacije *Department* i njenim ograničenjima, kao i ograničenjem ekvivalentnog ključa šeme relacije *Faculty* koje se u izvornom modelu zadaje kao ograničenje jedinstvenosti u kojem su sve kolone sa zabranjenom nula vrednošću. Takođe je skup ograničenja proširen sa ograničenjem inverznog referencijalnog integriteta između šema relacija *University* i *Faculty*:

- *Faculty*({*UniId*, *FacId*, *FacShortName*, *FacName*, *Dean*},

  {PrimaryKey(*UniId*+*FacId*), EquivalentKey(*UniId*+*FacShortName*)}),
- *Department*({*UniId*, *FacShortName*, *DepId*, *DepName*, *Director*},

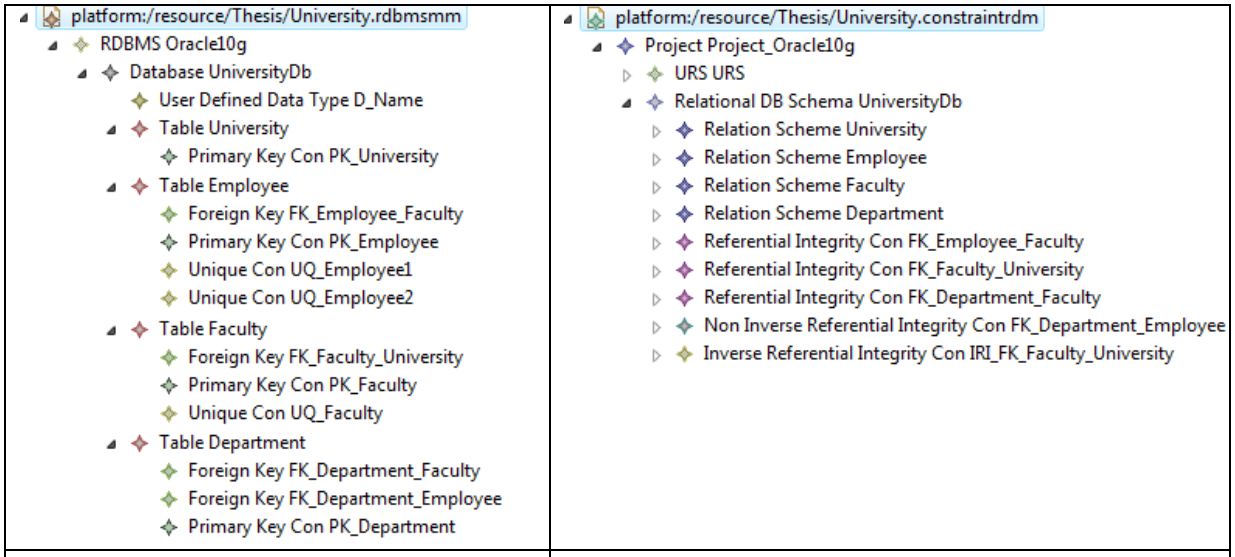
  {PrimaryKey(*UniId* + *FacShortName* + *DepId*)}),
- *FK_Department_Faculty*:

  Department[*UniId*, *FacShortName*] ⊆ *Faculty*[*UniId*, *FacShortName*],
- *FK_Department_Employee*:

  Department[*Director*] ⊆ *Employee* [*PasportNo*],
- *IRI_FK_ Faculty_University*:

  University[*UniId*] ⊆ *Faculty*[*UniId*].

Na slici 7.20 prikazan je segment šeme baze podataka koji ilustruje primenu prethodno opisanih pravila transformacije. U izvornom modelu prikazane su tabele sa relacionim ograničenjima i ograničenjima stranog ključa, slika 7.20 a). Atributi su izostavljeni zbog preglednosti.



a) Model dela *UniversityDb* u skladu sa RDBMSMM

b) Model dela *UniversityDb* u skladu sa RMM

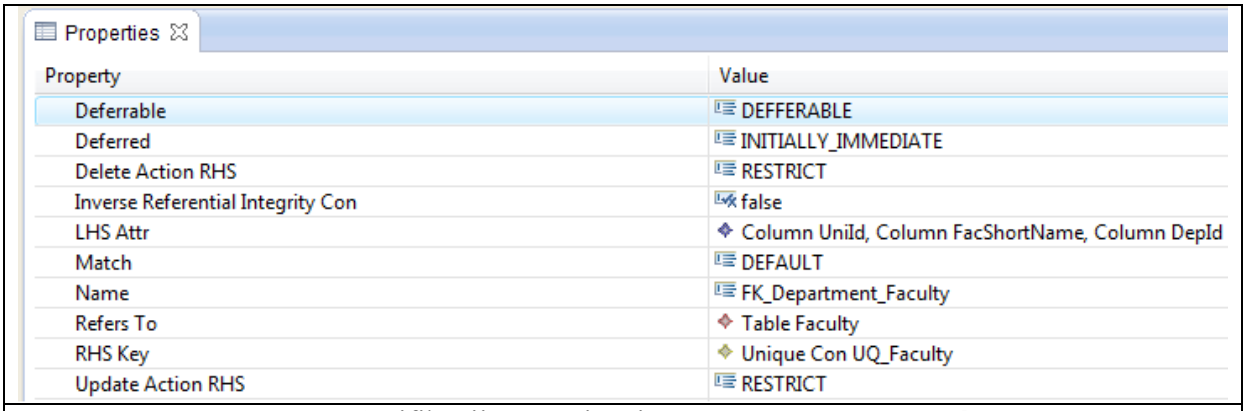
Slika 7.20. Deo *UniversityDb* relacione šeme baze podataka u Eclipse editoru

Od ograničenja stranog ključa *FK_Employee_Faculty* i *FK_Faculty_University*, koja sa desne strane ograničenja imaju primarni ključ referencirane tabele, nakon transformacije nastaju dva međurelaciona ograničenja tipa referencijalnog integriteta sa istim nazivima, što je prikazano na slici 7.20 b).

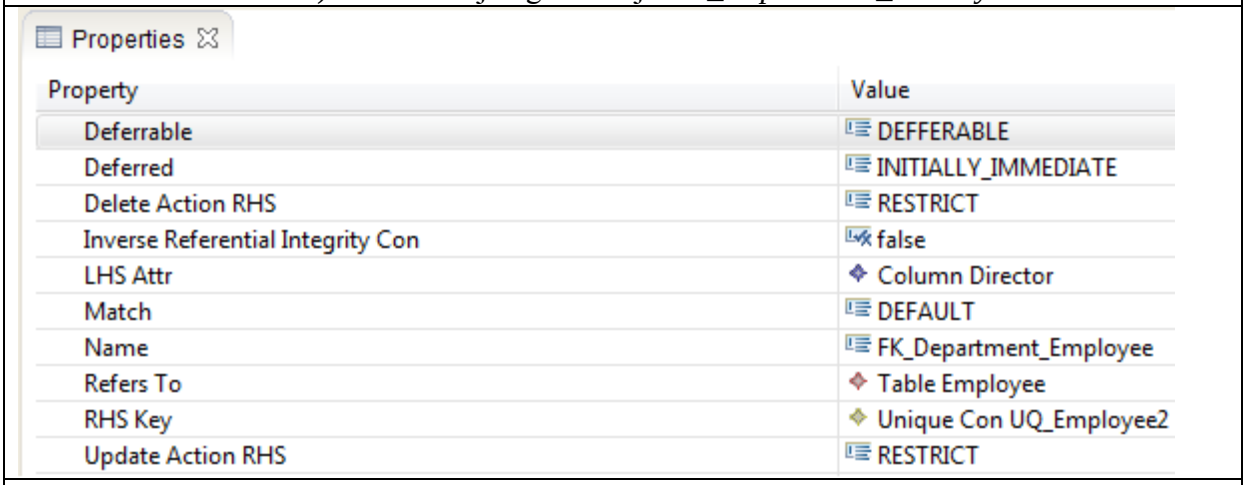
Od ograničenja stranog ključa *FK_Department_Faculty* koje na desnoj strani ograničenja ima skup kolona nad kojim je definisano ograničenje jedinstvenosti i gde je za sve kolone

specificirana zabrana nula vrednosti, takođe je nakon transformacije nastalo ograničenje referencijalnog integriteta sa istim nazivom. Osobine ovog ograničenja prikazane su na slici 7.21 a).

Ograničenje stranog ključa *FK_Department_Employee* sa desne strane ima ograničenje jedinstvenosti gde je atributu ograničenja dozvoljena nula vrednost, zbog čega je nakon transformacije od ovog ograničenja nastalo ograničenje tipa *non-key based* ograničenja koja nisu ograničenje inverznog referencijalnog integriteta. Osobine ovog ograničenja prikazane su na slici 7.21 b).

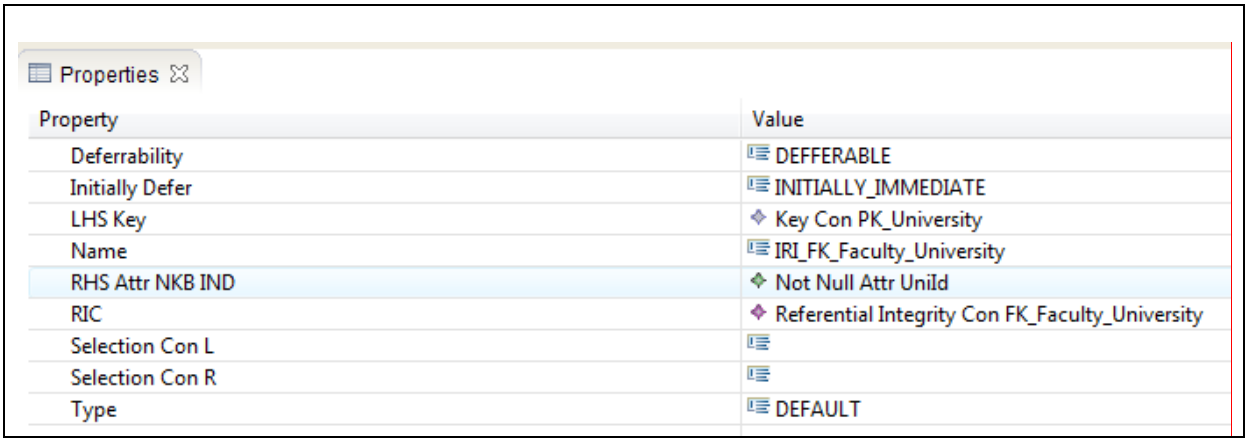


a) Secifikacija ograničenja *FK_Department_Faculty*



b) Secifikacija ograničenja *FK_Department_Employee*

Slika 7.21. Specifikacija ograničenja stranog ključa tabele *Department*



Slika 7.22. Specifikacija ograničenja stranog ključa tabele *Faculty*

Za ograničenje stranog ključa *FK_Faculty_University* vrednost atributa *InverseReferentialIntegrityCon* je *true*, što znači da postoji ograničenje inverznog referencijalnog integriteta, što se može videti na slici 7.22. Za ovo ograničenje stranog ključa nakon transformacije kreira se i ograničenje inverznog referencijalnog integriteta *IRI_FK_Faculty_University*.

Na listinzima 7.31 i 7.32 prikazani su primeri sa slike 7.20 a) i b) u XML XMI formatu, respektivno.

```
<?xml version="1.0" encoding="UTF-8"?> <rdbmsmm:RDBMS xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI" xmlns:rdbmsmm="http://rdbmsmm/1.0" Name="Oracle10g">
  <Databases Name="UniversityDb">
    <Tables Name="University">
      <TablePK Name="PK_University" PKandUQColumns="//@Databases.0/@Tables.0/
            @Columns.0"/>
     ...
    </Tables>
    <Tables Name="Employee">
      <TableFKs Name="FK_Employee_Faculty"
            RHS_Key="//@Databases.0/@Tables.2/@TablePK"
            LHS_Attr="//@Databases.0/@Tables.1/@Columns.2
            //@Databases.0/@Tables.1/@Columns.1"
            RefersTo="//@Databases.0/@Tables.2"/>
      <TablePK Name="PK_Employee"
            PKandUQColumns="//@Databases.0/@Tables.1/@Columns.1
            //@Databases.0/@Tables.1/@Columns.2
            //@Databases.0/@Tables.1/@Columns.0"/>
      <TableUQ Name="UQ_Employee1"
            PKandUQColumns="//@Databases.0/@Tables.1/@Columns.1
            //@Databases.0/@Tables.1/@Columns.2
            //@Databases.0/@Tables.1/@Columns.5"/>
      <TableUQ Name="UQ_Employee2"
            PKandUQColumns="//@Databases.0/@Tables.1/@Columns.7"/>
      ...
    </Tables>
    <Tables Name="Faculty">
      <TableFKs Name="FK_Faculty_University"
            RHS_Key="//@Databases.0/@Tables.0/@TablePK"
            LHS_Attr=//@Databases.0/@Tables.2/@Columns.0
            RefersTo="//@Databases.0/@Tables.0"
            InverseReferentialIntegrityCon="true"/>
      <TablePK Name="PK_Faculty"
            PKandUQColumns="//@Databases.0/@Tables.2/@Columns.0
            //@Databases.0/@Tables.2/@Columns.1"/>
      <TableUQ Name="UQ_Faculty"
            PKandUQColumns="//@Databases.0/@Tables.2/@Columns.0
            //@Databases.0/@Tables.2/@Columns.1
            //@Databases.0/@Tables.2/@Columns.2"/>
    </Tables>
    <Tables Name="Department">
      <TableFKs Name="FK_Department_Faculty"
            RHS_Key="//@Databases.0/@Tables.2/@TableUQ.0"
            LHS_Attr="//@Databases.0/@Tables.3/@Columns.0
            //@Databases.0/@Tables.3/@Columns.1
            //@Databases.0/@Tables.3/@Columns.2"
            RefersTo="//@Databases.0/@Tables.2"/>
      <TableFKs Name="FK_Department_Employee"
            RHS_Key="//@Databases.0/@Tables.1/@TableUQ.1"
            LHS_Attr="//@Databases.0/@Tables.3/@Columns.4"
            RefersTo="//@Databases.0/@Tables.1"/>
      <TablePK Name="PK_Department"
            PKandUQColumns="//@Databases.0/@Tables.3/@Columns.0
            //@Databases.0/@Tables.3/@Columns.1
            //@Databases.0/@Tables.3/@Columns.2"/>
```

```
      ...
    </Tables>
  </Databases>
</rdbmsmm:RDBMS>
```

Listing 7.31. Model dela *UniversityDb* u skladu sa RDBMSMM u XML XMI formatu

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConstraintRDM:Project xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ConstraintRDM="http://relacionimm/1.0/ConstraintRDM"
xmlns:INDConstraint="http://relacionimm/1.0/INDConstraint"
xmlns:RelationConstraint="http://relacionimm/1.0/RelationConstraint"
xmlns:URSConstraint="http://relacionimm/1.0/URSConstraint" Name="Project_Oracle10g">
  <URSs Name="URS">
    ...
  </URSs>
  <DBSchema Name="UniversityDb">
    <RelationSchemes Name="University">
      <PrimaryKey Name="PK_University"
                        KeyAttr="//@DBSchema/@RelationSchemes.0/@RSAttributes.0"/>
      ...
    </RelationSchemes>
    <RelationSchemes Name="Employee">
      <UQConstraints Name="UQ_Employee2"
            UQConNullAttr="//@DBSchema/@RelationSchemes.1/@RSAttributes.7"/>
      <PrimaryKey Name="PK_Employee"
            KeyAttr="//@DBSchema/@RelationSchemes.1/@RSAttributes.1
            //@DBSchema/@RelationSchemes.1/@RSAttributes.2
            //@DBSchema/@RelationSchemes.1/@RSAttributes.0"/>
      <EquivalentKey Name="UQ_Employee1"
            KeyAttr="//@DBSchema/@RelationSchemes.1/@RSAttributes.1
            //@DBSchema/@RelationSchemes.1/@RSAttributes.2
            //@DBSchema/@RelationSchemes.1/@RSAttributes.5"/>
      ...
    </RelationSchemes>
    <RelationSchemes Name="Faculty">
      <PrimaryKey Name="PK_Faculty"
            KeyAttr="//@DBSchema/@RelationSchemes.2/@RSAttributes.0
            //@DBSchema/@RelationSchemes.2/@RSAttributes.1"/>
      <EquivalentKey Name="UQ_Faculty"
            KeyAttr="//@DBSchema/@RelationSchemes.2/@RSAttributes.0
            //@DBSchema/@RelationSchemes.2/@RSAttributes.1
            //@DBSchema/@RelationSchemes.2/@RSAttributes.2"/>
      ...
    </RelationSchemes>
    <RelationSchemes Name="Department">
      <PrimaryKey Name="PK_Department"
            KeyAttr="//@DBSchema/@RelationSchemes.3/@RSAttributes.0
            //@DBSchema/@RelationSchemes.3/@RSAttributes.1
            //@DBSchema/@RelationSchemes.3/@RSAttributes.2"/>
      ...
    </RelationSchemes>
    <MRConstraints xsi:type="INDConstraint:ReferentialIntegrityCon"
            Name="FK_Employee_Faculty"
            RHS_Key="//@DBSchema/@RelationSchemes.2/@PrimaryKey"
            LHS_Attr_RIC="//@DBSchema/@RelationSchemes.1/@RSAttributes.2
            //@DBSchema/@RelationSchemes.1/@RSAttributes.1">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT"
            RHS_RS_IND="//@DBSchema/@RelationSchemes.2"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.1"/>
    </MRConstraints>
    <MRConstraints xsi:type="INDConstraint:ReferentialIntegrityCon"
            Name="FK_Faculty_University"
            RHS_Key="//@DBSchema/@RelationSchemes.0/@PrimaryKey"
```

```
            LHS_Attr_RIC="//@DBSchema/@RelationSchemes.2/@RSAttributes.0">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT"
            RHS_RS_IND="//@DBSchema/@RelationSchemes.0"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.2"/>
    </MRConstraints>
    <MRConstraints xsi:type="INDConstraint:ReferentialIntegrityCon"
            Name="FK_Department_Faculty"
            RHS_Key="//@DBSchema/@RelationSchemes.2/@EquivalentKey.0"
            LHS_Attr_RIC="//@DBSchema/@RelationSchemes.3/@RSAttributes.0
            //@DBSchema/@RelationSchemes.3/@RSAttributes.1
            //@DBSchema/@RelationSchemes.3/@RSAttributes.2">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT"
            RHS_RS_IND="//@DBSchema/@RelationSchemes.2"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.3"/>
    </MRConstraints>
    <MRConstraints xsi:type="INDConstraint:NonInverseReferentialIntegrityCon"
            Name="FK_Department_Employee"
            RHS_Attr_NKB_IND="//@DBSchema/@RelationSchemes.1/@RSAttributes.7"
            LHS_Attr_NIRIC="//@DBSchema/@RelationSchemes.3/@RSAttributes.4">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT"
            RHS_RS_IND="//@DBSchema/@RelationSchemes.1"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.3"/>
    </MRConstraints>
    <MRConstraints xsi:type="INDConstraint:InverseReferentialIntegrityCon"
            Name="IRI_FK_Faculty_University"
            RHS_Attr_NKB_IND="//@DBSchema/@RelationSchemes.2/@RSAttributes.0"
            LHS_Key="//@DBSchema/@RelationSchemes.0/@PrimaryKey"
            RIC="//@DBSchema/@MRConstraints.1">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT"
            RHS_RS_IND="//@DBSchema/@RelationSchemes.2"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.0"/>
    </MRConstraints>
  </DBSchema>
</ConstraintRDM:Project>
```

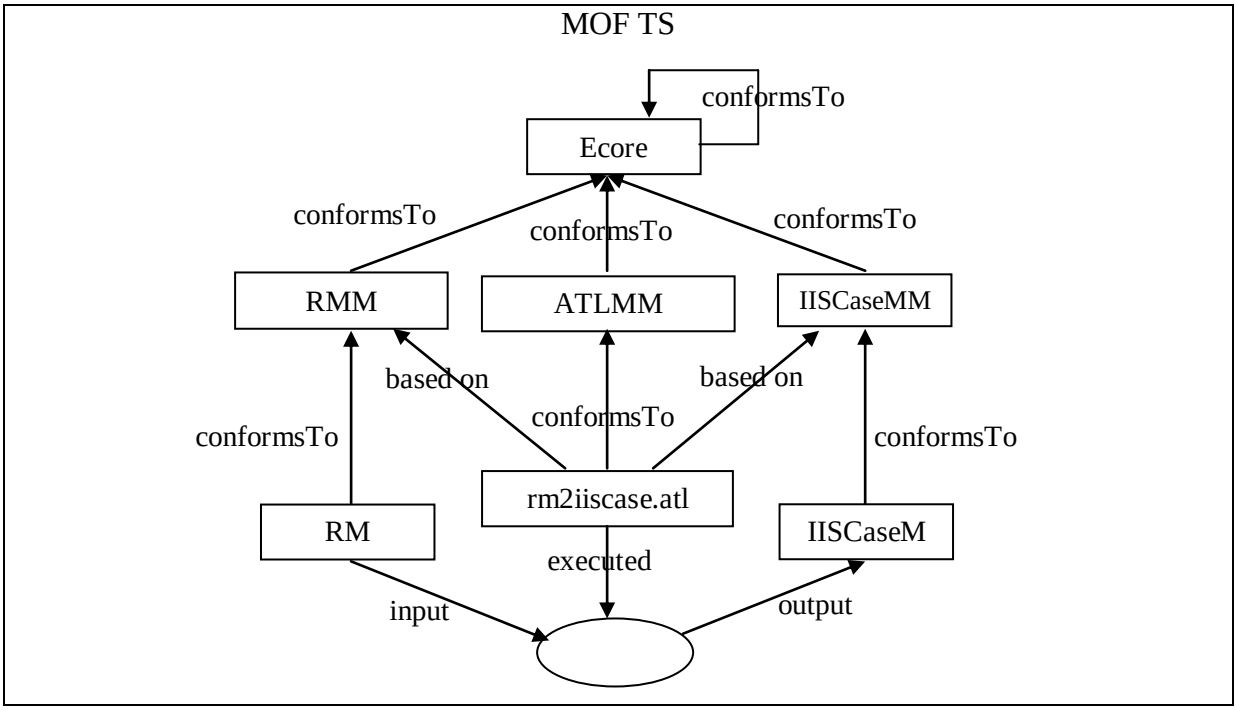
Listing 7.32. Model dela *UniversityDb* u skladu sa RMM u XML XMI formatu

7.3. Definicije mapiranja koncepata Generičkog meta-modela u koncepte IISCase meta-modela

U prethodnom odeljku prezentovan je postupak transformacije tipa PSM-u-PSM, čiji rezultat je instanca Generičkog meta-modela. Rezultat ove transformacije predstavlja ulazni model u transformaciju RM2IISCase, koja je predstavljena kao četvrti korak u nizu transformacija prikazanih na slici 7.1. Rezultat četvrtog koraka transformacije, čiji scenario je prikazan na slici 7.23, je konceptualna šema baze podataka zasnovana na IIS*Case konceptima tipa forme.

Tip ove transforamcije je PSM-u-PIM jer su IIS*Case koncepti tipa forme nezavisni od platformi za implementaciju.

U ovom odeljku će biti prezentovani algoritmi pravila i njihova imlementacija ATL jezikom. Pravila se primenjuju na koncepte ulaznog modela, koji je u skladu sa Generičkim meta-modelom (RMM), u cilju dobijanja semantički ekvivalentog modela koji je u skladu sa IISCase meta-modelom (IISCaseMM). Izvorni i ciljni meta-modeli su prezentovani u poglavljima 6.3 i 6.4, respektivno.



Slika 7.23. Scenario četvrtog koraka transformacije

U tabeli 7.2. prikazani su koncepti oba meta-modela koji učestvuju u mapiranjima pri izvršavanju transformacije RM2IISCase. Svako od ovih mapiranja biće detaljno opisano u tekstu koji sledi.

Tabela 7.2. Koncepti za mapiranje transformacije RM2IISCase

Koncepti generičkog meta-modela (RMM)	Koncepti IISCase meta-modela (IISCaseMM)
Project	ISApplicationModel
	ApplicationSystem
RelationScheme	FormTypeProgram
	ComponentTypeRoot
ReferentialIntegrityCon InverseReferentialIntegrityCon	FormTypeProgram
	ComponentTypeRoot
	ComponentTypeChild
Attribute	AttributeIncludedInDB
PrimitiveDomain	PrimitiveDomain
UserDefDomFromPrimitiv	UserDefinedDomainFromPrimitiveDomain
UserDefDomFromUserDef	UserDefinedDomainFromUserDefinedDomain
InclusionDependencyURS	InclusionDependency
NullAttr	ComponentTypeAttribute
NotNullAttr	
KeyCon	ComponentTypeKey
UniqueCon	ComponentTypeUnique
TupleCon	ComponentTypeCheck

Ostatak odeljka je organizovan na sledeći način. U tački 7.3.1. biće data definicija koncepta tipa forme, kao i klasifikacija tipova formi, dok će u tački 7.3.2 biti prikazana klasifikacija šema relacija. Obe klasifikacije su napravljene za potrebe transformacije. U tačkama od 7.3.3 do 7.3.6 biće prezentovani algoritmi mapiranja koncepata iz tabele 7.2 u

postupku transformacije koja je razvijena kao jedan od osnovnih rezultata istraživanja ove doktorske disertacije.

Kao i u prethodnom odeljku, rezultati transformacije biće prikazani na primerima šema relacija koje pripadaju segmentu realcione baze podataka jednog univerziteta (*UnivesityDb*).

7.3.1. Koncept tipa forme i klasifikacija tipova formi

U ovoj tački odeljka ukratko će biti ponovljeni neki osnovni uslovi kojima se uvodi koncept tipa forme. Uvodi se i notacija koja će biti korišćena u opisu transformacija, radi lakšeg praćenja algoritama koji će biti prezentovani.

Tip forme $\mathcal{F}$ je imenovana struktura tipa stabla, čije čvorove predstavljaju *tipovi komponenti C*. Neka je $W(\mathcal{F})$ skup obeležja tipa forme sa nazivom $\mathcal{F}$, pri čemu svakom obeležju iz skupa $W(\mathcal{F})$ odgovara jedno polje za unos ili prikaz podataka na ekranskoj formi.

Svaki tip komponente tipa forme $\mathcal{F}$ je imenovana dvojka $N(Q, O)$, gde je N naziv, Q skup atributa $Q=\{A_1, .., A_n\} \subseteq W(\mathcal{F})$, a O skup ograničenja tipa komponente. Svaki tip komponente je jedinstveno identifikovan svojim nazivom u okviru tipa forme $\mathcal{F}$. Skup ograničenja tipa komponente O čini unija skupova ključeva i ograničenja jedinstvenosti i ograničenje torke:

$$O = K \cup UQ \cup CH.$$

Svaki atribut tipa komponente jednoznačno je određen unijom po jednog ključa tipova komponenti koji su hijerarhijski iznad datog tipa komponente, gdje je N tip komponente tipa forme $\mathcal{F}$, $\chi(N)$ je skup unija po jednog ključa od korenskog do tipa komponente N. Takođe, za skupove atributa važi:

$$\bigcup_{i=1}^{m} Q_i = W(\mathcal{F}) \text{ i}$$

$$(\forall N_i, N_j \in C)(i \neq j \Leftrightarrow Q_i \cap Q_j = \varnothing).$$

Jedan atribut može biti, u okviru tipova komponenti, uključen u različite tipove formi, ali ne sme pripadati različitim tipovima komponenti istog tipa forme.

Svakom tipu komponente mora biti pridružen skup dozvoljenih operacija nad bazom podataka. Skup dozvoljenih operacija je podskup skupa "standardnih" operacija nad bazom podataka {*retrieve, insert, update, delete*}. Ako je *update* operacija pridružena tipu komponente, skup atributa tipa komponente čije vrednosti se mogu menjati, takođe, moraju biti specificirani. Dodatno, svaki atribut tipa komponente može biti obavezan ili opcion. Ako atribut tipa komponente ne sme da ima nula vrednost, označava se kao obavezan. U suprotnom je opcioni.

Kako je već objašnjeno u odeljku 6.4.5, samo programski tipovi formi učestvuju u projektovanju baze podataka u *forward* inženjeringu. Iz ovog razloga u reverznom inženjeringu baze podataka interesantna je samo ova vrsta tipova formi, tako da se u narednom tekstu pod terminom tipa forme smatra programski tip forme. Specifikacija domena, atributa i atributa tipova komponenti detaljno je opisana u odeljku 6.4.

Pošto se radi o automatizovanom procesu u kojem nema interakcije sa korisnikom, pri izboru načina implementacije RM2IISCase transformacije razmatrana su dva pristupa:

- pristup u kojem bi korisnik dobio samo jednostavne tipove formi, a zatim dodavao tipove komponenti i sve ostale dozvoljene koncepte koje želi (atribute, ograničenja i njihove razne osobine) i

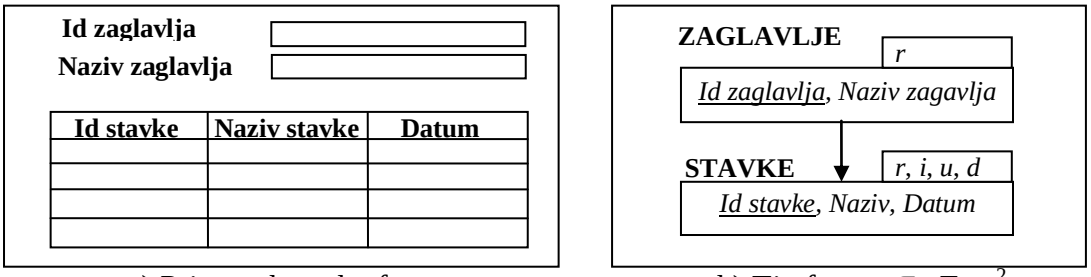
- pristup u kojem bi se generisale sve relevantne kombinacije tipova formi, gde bi korisnik nakon transformacije, ukoliko bi smatrao da mu nisu sve potrebne, mogao da briše one koje mu ne trebaju.

Izabran je ovaj drugi pristup jer je smatrano da će korisnik uložiti mnogo manji napor da, eventualno, obriše ono što mu je nepotrebno, nego da nakon transformacije mora da dodaje ono što mu je potrebno. I u ovom drugom pristupu korisniku je prirodno omogućeno, ako želi, da dodaje nove tipove formi, ili sprovodi izmene nad ponuđenim tipovima formi.

U skladu sa izabranim pristupom, prema načinu implementacije transformacije, tipovi formi su podeljeni na tri vrste:

- $\mathcal{F}$ *_Basic* – elementarni tip forme čije stablo ima samo jedan korenski tip komponente,
- $\mathcal{F}$ *_Tree*^{2} – tip forme koji u stablu osim korenskog tipa komponenete ima jednog potomka i
- $\mathcal{F}$ *_Tree*^{n} – tip forme koji u stablu osim korenskog tipa komponenete ima proizvoljan broj potomaka.

$\mathcal{F}$ *_Tree*^{2} je specijalni slučaj $\mathcal{F}$ *_Tree*^{n} tipa forme. Razlog zašto je $\mathcal{F}$ *_Tree*^{2} tip forme izdvojen kao posebna vrsta proizilazi iz moći prosečne ljudske percepcije i odnosi se na sledeće. Smatra se da će, u najvećem broju slučajeva, ekranske forme, pomoću kojih korisnik manipuliše podacima, biti relativno jednostavne, odnosno svedene samo na zaglavlje i skup stavki, što je reprezentovano $\mathcal{F}$ *_Tree*^{2} vrstom tipova formi. Primer takvog tipa forme prikazan je na slici 7.24 gde struktura tipa forme $\mathcal{F}$, prikazana na slici 7.24 b) generalizuje ekransku formu prikazanu na slici 7.24. a). Izvorni koncepti koji odgovaraju ulaznom meta-modela za ovu vrstu tipova formi biće ograničenja referencijalnog integriteta i šeme relacija koje se nalaze na levoj i desnoj strani ograničenja. Ograničenje ATL jezika, koji je izabran za specifikaciju i implementaciju transformacije, je još jedan dodatni razlog zašto je $\mathcal{F}$ *_Tree*^{2} tip forme izdvojen kao posebna vrsta. U ATL jeziku više različitih pravila uparivanja (*Matched Rules*) ne mogu imati isti ulazni koncept. Pošto izabrani pristup generisanju svih relevantnih kombinacija tipova formi zahteva da se jedna šema relacije pojavi više puta u različitim tipovima formi, odlučeno je da se implementiraju različita pravila za $\mathcal{F}$ *_Tree*^{2} i $\mathcal{F}$ *_Tree*^{n}, gde je za pravila kreiranja $\mathcal{F}$ *_Tree*^{n} tipova formi korišćen složeniji ulazni koncept. Detalji pravila transformacije za sve vrste tipova formi biće prikazani u narednim tačkama odeljka.



a) Primer ekranske forme b) Tip forme $\mathcal{F}$ *_Tree*^{2}

Slika 7.24. Primer ekranske forme sa zaglavljem i stavkama

Ceo skup tipova formi informacionog sistema, koji se može dobiti reverznim inženjeringom iz nasleđene relacione baze podataka, pomoću RM2IISCase transformacije, čini unija skupova tipova formi navedene tri vrste:

$$\mathcal{F}\,T = \mathcal{F}\,T_B \cup \mathcal{F}\,T_{T2} \cup \mathcal{F}\,T_{Tn},$$

gde je $\mathcal{F}\,T_B$ skup *F_Basic* tipova formi, $\mathcal{F}\,T_{T2}$ skup $\mathcal{F}$ *_Tree*^{2} tipova formi i $\mathcal{F}\,T_{Tn}$ skup $\mathcal{F}$ *_Tree*^{n} tipova formi.

Primer 7.5. Za sledeće šeme relacija:

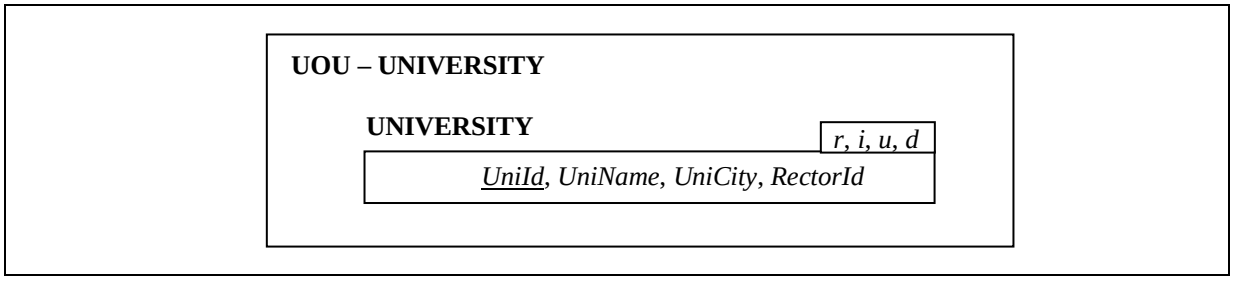
- *University*({*UniId, UniName, UniCity, RectorId*}, {PrimaryKey(*UniId*)})
- *Faculty*({*UniId, FacId, FacShortName, FacName, Dean*}, {PrimaryKey(*UniId*+*FacId*)})
- *Department*({*UniId, FacId, DepId, DepName, DeanId*}, {PrimaryKey(*UniId + FacId + DepId*)})
- *Course*({*CourseId, CourseShortName, CourseName, Lecturer, Prerequisite, Year, Semester*}, {PrimaryKey(*CourseId*)}),
- *Exam*({*CourseId, ExamId, Examiner, DateTime*}, {PrimaryKey(*CourseId + ExamId*)}) i
- *CourseTimetable*({*CourseId, Day, Time, Classroom*}, {PrimaryKey(*CourseId + Day + Time*)}) ,

u skladu sa izabranim pristupom generisanja svih relevantnih kombinacija tipova formi, na izlazu iz RM2IISCase transformacije biće dobijeno sledeće:

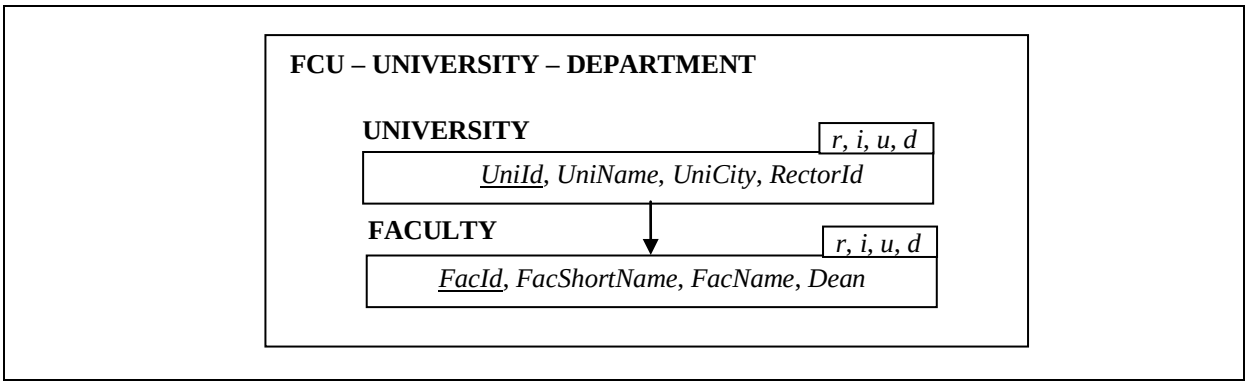
- tipovi formi sa samo jednim korenkim tipom komponente za svaku od ovih pet šema relacija ($\mathcal{F}$ *_Basic*),
- tipovi formi sa jednim korenskim tipom komponente i jednim potomkom za parove šema relacija: (***University***, ***Faculty***), (***Faculty, Department***), (***Course, Exam***) i (***Course, CourseTimetable***), gde je prvi element uređene dvojke šema relacije od koje nastaje korenski tip komponente ($\mathcal{F}$ *_Tree*^{2}),
- tip forme u kojem ima više nivoa hijerarhije nastao od šema relacija: ***University***, ***Faculty*** i ***Department***, gde je ***University*** šema relacije od koje nastaje korenski tip komponenete, ***Faculty*** šema relacije od koje nastaje tip komponente direktni potomak od ***University,*** dok je ***Department*** šema relacije od koje nastaje tip komponente potomak od ***Faculty*** ($\mathcal{F}$ *_Tree*^{n}) i
- tip forme u kojem, osim korenskog tipa komponente, postoje i dva tipa komponente potomka na istom nivou hijerarhije. On nastaje od šema relacija: ***Course***, ***CourseTimetable*** i ***Exam***, gde je ***Course*** šema relacije od koje nastaje korenski tip komponenete, dok su ***CourseTimetable*** i ***Exam*** šeme relacije od kojih nastaju tipovi komponenti potomci.

Neki od ovih tipova formi, koji će nastati transformacijom, prikazani su na slikama 7.25, 7.26 i 7.27, respektivno.

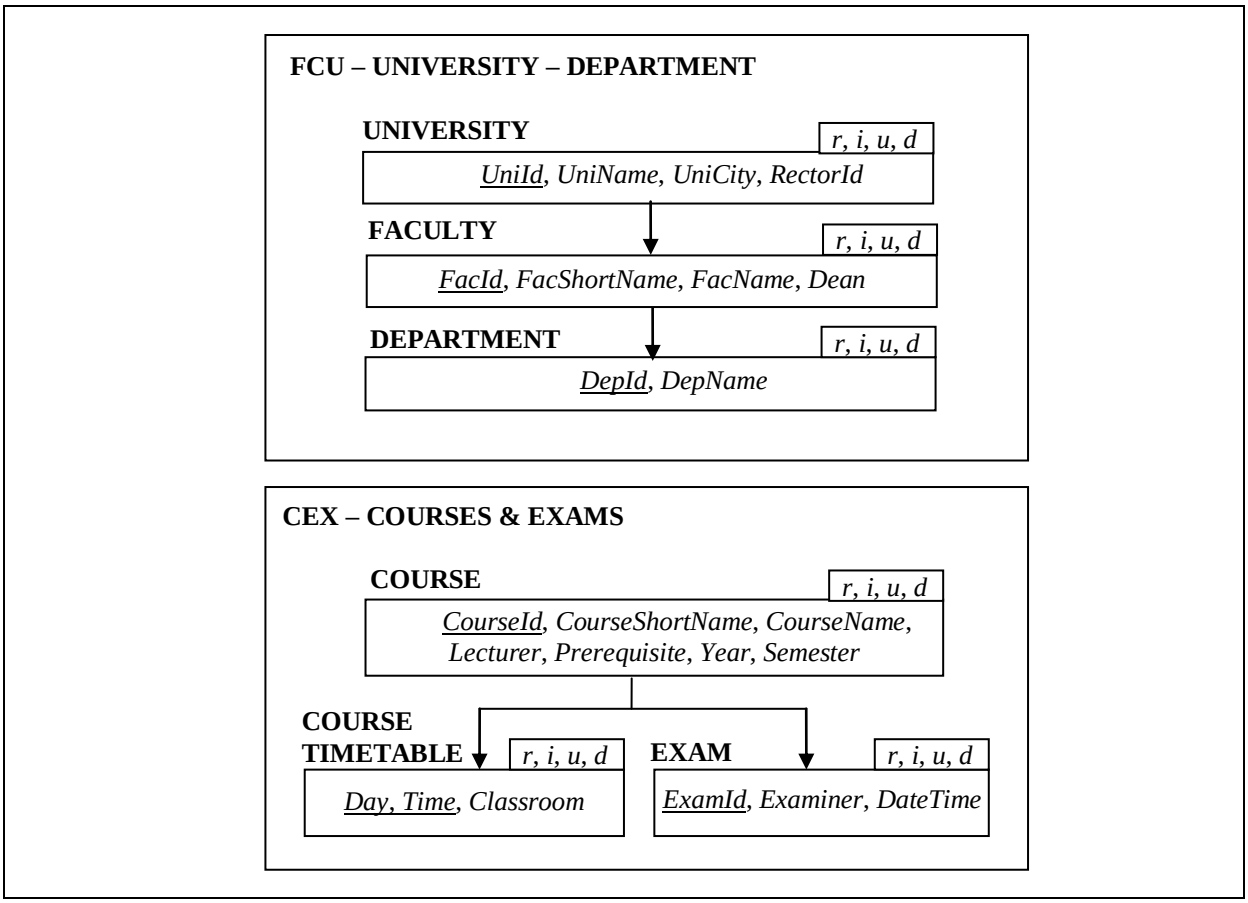
U tački 7.3.3 biće detaljno objašnjeni postupci dobijanja svih tipova formi koji nastaju kao rezultat transformacije RM2IISCase, ugrađene u *M2M Transfromer* alata IIS*Ree.



Slika 7.25. Primer $\mathcal{F}$ *_Basic* tipa forme



Slika 7.26. Primer $\mathcal{F}$ *_Tree*^{2} tipa forme



Slika 7.27. Primeri $\mathcal{F}$ *_Tree*^{n} tipova formi

7.3.2. Klasifikacija šema relacija

U reverznom inženjeringu relacione baze podataka, tj. pri transformaciji relacione šeme baze podataka u konceptualnu šemu, potrebno je identifikovati različite tipove šema relacija, jer u zavisnosti od tipa šeme relacije primenjuju se odgovarajući algoritmi za mapiranje.

Identifikacija različitih šema relacija vrši se na osnovu primarnih ključeva šema relacija i zavisnosti sadržavanja.

Autori rada [Hamm02], referencirajući sa na radove [Chiang95] i [Fong06], dali su klasifikacije šema relacija, pri čemu je u njihovim radovima cilj reverznog inženjeringa bio transformacija u ER model podataka. U ovoj doktorskoj disertaciji napravljena je slična

klasifikacija, ali je prilagođena transformaciji koja je predmet istraživanja i čiji je krajnji cilj konceptualna šema zasnovana na IIS*Case konceptu tipa forme.

Algoritmi za transformaciju primenjeni u ovoj doktorskoj disertaciji prepoznaju sledeće tipove šema relacija:

- *osnovna šema relacije* (*Basic Relation Scheme, BR*),
- *slaba šema relacije* (*Weak Relation Scheme, WR*) i
- *šema relacije tipa "samo ključevi"* (*All Keys, AK*).

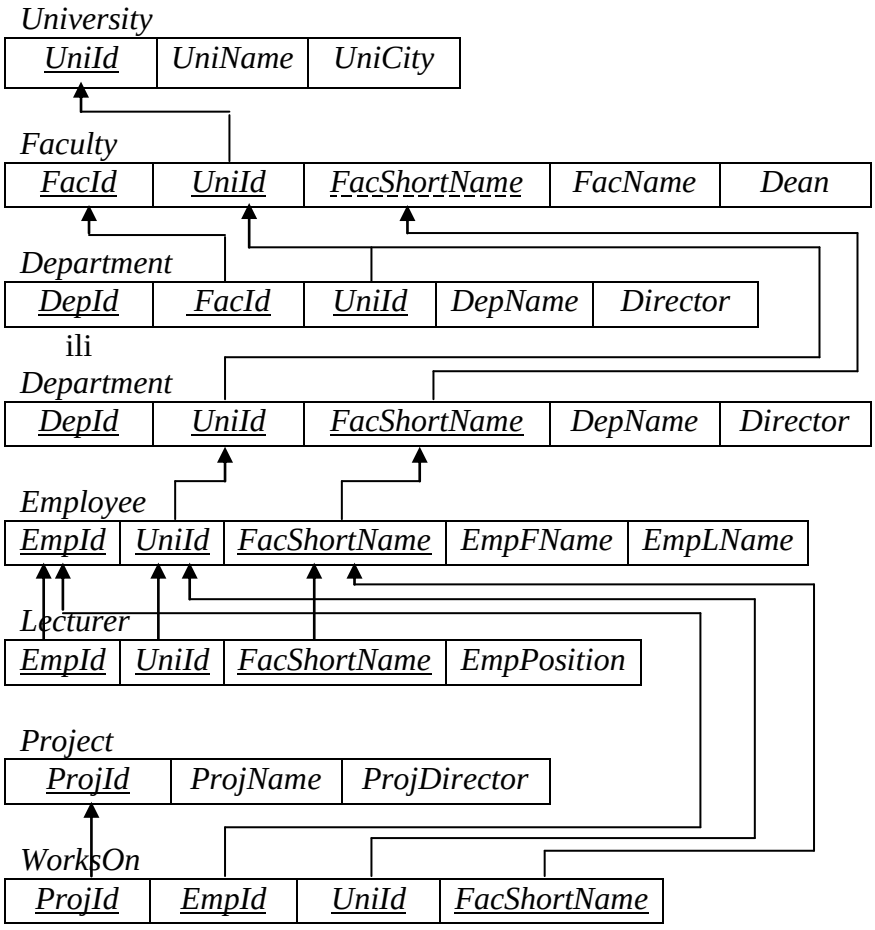
BR je šema relacije čiji primarni ili ekvivalentni ključ ne sadrži ključna obeležja neke druge šeme relacije.

WR je šema relacije koja zadovoljava sledeće uslove:

- podskup od skupa atributa koji pripadaju primarnom ključu sadrži u potpunosti obeležja ključa (primarnog ili ekvivalnetnog) drugih šema relacija,
- ostatak atributa ključa može, a ne mora da sadrži sopstvena identifikaciona obeležja i
- ostatak atributa ključa može, a ne mora da sadrži ključna obeležja neke druge šeme relacije.

AK je šema relacije čiji se primarni ključ sastoji samo od unije ključeva drugih šema relacija i nema drugih obeležja.

Ova poslednja vrsta je specijalni slučaj *slabe šeme relacije* i u najvećem broju slučajeva predstavlja šemu relacije koja povezuje dve ili više drugih šema relacija. Izdvojena je kao poseban tip, jer se smatralo da skupu atributa tipa komponente koji nastaje mapiranjem njenog skupa atributa, koji čine samo ključna obeležja, treba dodati i skupove atributa povezanih šema relacija. Detalji mapiranja biće prikazani u tekstu koji sledi.

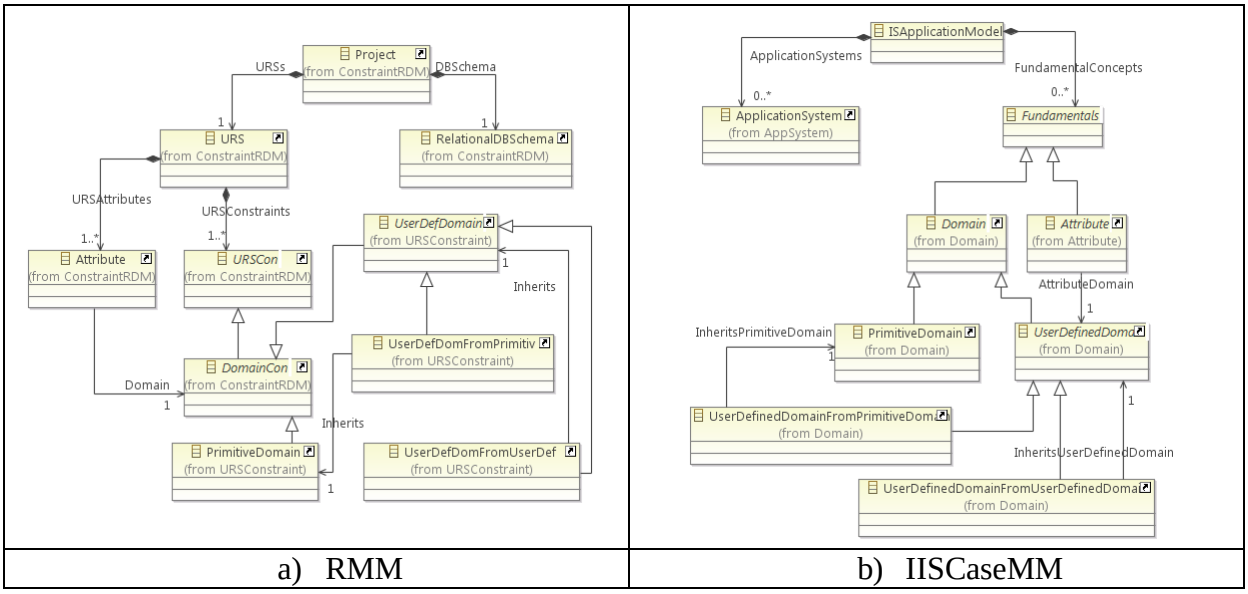


Slika 7.28. Primeri različitih tipova šema relacija

Primer 7.6. Na slici 7.28 prikazani su primeri različitih tipova šema relacija. Šema relacije *University* je tipa *BR*. Šeme relacija *Faculty*, *Department* i *Employee* su tipa *WR*, sa sopstvenim identifikacionim obeležjima, dok je šema relacije *Lecturer* primer šeme relacije tipa *WR* koja nema sopstvena identifikaciona obeležja. Druga varijanta šeme relacije *Department* je primer šeme relacije tipa *WR* koja sadrži u primarnom ključu ekvivalnetni ključ šeme relacije *Faculty*. Šema relacije *WorksOn* sadrži samo ključna obeležja i predstavlja šemu relacije tipa *AK*.

7.3.3. Mapiranje koncepta Project u koncept ISApplicationModel

Kao što je već prikazano u tački 7.2.1, korenski element Generičkog meta-modela, predstavlja koncept projekta, **Project**, dok je na vrhu hijerarhije IISCase meta-modela model aplikacije informacionog sistema, **ISApplicationModel**. Na slici 7.29 prikazani su delovi meta-modela sa ovim konceptima, kao i sa ostalim konceptima koji su obuhvaćeni algoritmom za mapiranje, čiji pseudokod je prikazan na slici 7.30.



a) RMM | b) IISCaseMM

Slika 7.29. Delovi meta-modela za mapiranje koncepta **Project** u koncept **ISApplicationModel**

Model aplikacije može uključivati više aplikativnih sistema, **ApplicationSystem** i osnovnih koncepata, ***Fundamentals***, pri projektovanju informacionog sistema u *forward* inženjeringu. U procesu reverznog inženjeringa, međutim, za postojeću šemu baze podataka kreira se samo jedan aplikativni sistem.

Naziv modela aplikacije dobija naziv projekta sa prefiksom IISCase_, dok aplikativni sistem dobija naziv šeme baze podataka sa prefiksom AppSystem_.

Osnovni koncepti su formalno nezavisni od bilo kog aplikativnog sistema i kreiraju se na nivou modela aplikacije informacionog sistema. Osnovni koncepti koje razmatramo u reverznom inženjeringu su: domeni (***Domain***), atributi (***Attribute***) i zavisnosti sadržavanja (**InclusionDependency**). IIS*Case je alat zasnovan na pretpostavci o postojanju šeme univerzalne relacije i ovi koncepti su identični konceptima domena, atributa i zavisnosti sadržavanja iz Generičkog meta-modela. U procesu transformacije skup osnovnih kocepata IISCase meta-modela će nastati od unije skupova atributa *U* i ograničenja *O* šeme univerzalne relacije.

U *forward* inženjeringu, putem koncepta tipa forme projektant na nivou apstrakcije koji je nezavisan od platforme, indirektno kreira specifikaciju šeme baze podataka sa uključenim atributima i ograničenjima. Ovo omogućava da se u postupku reverznog inženjeringa iz postojeće šeme baze podataka rekonstruišu tipovi formi. Pošto se za postojeću šemu baze podataka kreira jedan aplikativni sistem, od interesa su samo tipovi forme koje aplikativni sistem poseduje, **OwnedFormTypes**.

Tipovi formi se u postupku transformacije kreiraju pomoću procedura: *KreiranjeF_Basic*, *KreiranjeF_Tree*[2] i *Kreiranje F_Tree*[n]. Svaka od ovih procedura poziva se za odgovarajuću vrstu tipova formi, čija klasifikacija je data u tački 7.3.1. Algoritmi ovih procedura biće prikazani i objašenjeni u narednim tačkama odeljka.

Algoritam:	ZA MAPIRANJE KORENSKIH ELEMENATA
Ulaz:	*Project* $S = \{N_i(R_i, O^{RS}_i) \mid i = 1,.., n\}$ $RIC = \{ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$
Izlaz:	*ISApplicationModel*
Lokalne deklaracije: *ft* - izlazni parametar iz procedura (*ft* –skup tipova formi)	
Pseudokod: **POČETAK PROCESA** KreirajISApplicationModel **RADI** kreiranjeISAM **POSTAVI** *Name* ← 'IISCase_' + *NazivProjekta* **POSTAVI** *Fundamentals* ← $U \cup O$ **RADI** kreiranjeAppSystem **POSTAVI** *Name* ← 'AppSystem_' + *NazivDBSchema* **POZOVI** *KreiranjeF_Basic*(*S*, *ft*) **POSTAVI** $FT_B \leftarrow FT_B \cup ft$ **POZOVI** *KreiranjeF_Tree*[2](*S*, *RIC*, *ft*) **POSTAVI** $FT_{T2} \leftarrow FT_{T2} \cup ft$ **POZOVI** *KreiranjeF_Tree*[n](*S*, *RIC*, *ft*) **POSTAVI** $FT_{Tn} \leftarrow FT_{Tn} \cup ft$ **POSTAVI** $FT = FT_B \cup FT_{T2} \cup FT_{Tn}$ **KRAJ RADI** kreiranjeAppSystem **KRAJ RADI** kreiranjeISAM **KRAJ PROCESA** KreirajISApplicationModel	

Slika 7.30. Algoritam za mapiranje koncepta **Project** u koncepte **ISApplicationModel** i **AppSystem**

Definicija opisanog mapiranja implementirana je pomoću ATL pravila *Project2ISApplicationModel* prikazanog na listingu 7.33. Primenom ovog pravila na ulazni model dobija se model aplikacije informacionog sistema (*appModel*) sa skupom osnovnih koncepata (*FundamentalConcepts*) i aplikativnim sistemom (*appSystem*). Skupovi tipova formi koje će pripadati novokreiranom aplikativnom sistemu kreiraju se pomoću *lazy* ATL pravila, koja će biti prikazana kasnije.

```
rule Project2ISApplicationModel {
from
```

```
   p: RM!Project
using {
   rct: Set(RM!RelationScheme) = thisModule.CandidateForRootCT.asSet();
   ric:Sequence(RM!ReferentialIntegrityCon)=
            RM!ReferentialIntegrityCon.allInstancesFrom('IN');
   }
to
   appModel:IISCase!ISApplicationModel (
      Name <- 'IISCase_' + p.Name,
      FundamentalConcepts <- p.URSs.URSAttributes.union(p.URSs.URSConstraints),
      ApplicationSystems <- AppSystem
   ),
   appSystem: IISCase!ApplicationSystem(
      Name <- 'AppSystem_' + p.DBSchema.Name,
      OwnedFormTypes <- Sequence {p.DBSchema.RelationSchemes,
                        ric-> select(a| thisModule.isRSforFormType(a)) ->
                        collect(e | thisModule.RIC2FormTypes(e)),
                        rct->collect(ft | thisModule.getFormType(ft))}
   )
}
```

Listing 7.33. ATL pravilo za mapiranje koncepta **Project** u koncepte **ISApplicationModel** i **AppSystem**

U pravilu *Project2ISApplicationModel* pozivaju se ATL pomoćne metode kojima se izdvajaju odgovarajući koncepti od kojih mogu nastati tipovi formi. ATL pomoćna metoda koja pronalazi sve šeme relacija koje mogu biti korenski tip komponente u nekom tipu forme, prikazana je na listingu 7.34, dok je na listingu 7.35 prikazana ATL pomoćna metoda koja izdvaja odgovarajuća ograničenja referencijalnog integriteta, od čijih šema relacija sa desne i leve strane ograničenja postaju korenski tipovi komponenti i njihovi potomci, respektivno.

Na listinzima od 7.36 do 7.40, pomoću ATL pravila prikazane su definicije mapiranja za osnovne koncepte modela aplikacije: atrubute, primitivne domene, korisnički definisane domene koji nasleđuju primitivni domen, korisnički definisane domene koji nasleđuju druge korisnički definisane domene i zavisnosti sadržavanja. Pošto su u pitanju isti koncepti, osobine koncepta iz Generičkog meta-modela postaju i osobine koncepta iz IISCase meta-modela.

```
helper def: CandidateForRootCT: Sequence(RM!RelationScheme) =
   let allRIC:Sequence(RM!ReferentialIntegrityCon)=
            RM!ReferentialIntegrityCon.allInstances() in
   allRIC->iterate(ric; rez: Sequence(RM!RelationScheme) = Sequence{} |
   if (thisModule.isRSforFormType(ric)) then
      rez.append(ric.RHS_RS.RHS_RS_IND) else rez endif
);
```

Listing 7.34. ATL *helper* za pronalaženje kandidata za korenski tip komponente

```
helper def: isRSforFormType(ric: RM!ReferentialIntegrityCon): Boolean =
(ric.LHS_Attr_RIC.asSet()-
            ric.LHS_RS.LHS_RS_IND.PrimaryKey.KeyAttr.asSet()).isEmpty() ;
```

Listing 7.35. ATL *helper* za proveravanje da li je šema relacije na desnoj strani ograničenja referencijalnog integriteta kandidat za korenski tip komponente

```
rule Attribute2Attribute{
```

```
from
   a: RM!Attribute
to
   out: IISCase!AttributeIncludedInDB(
      Name <- a.Name,
      Description <-a.Name,
      AttributeDomain <- a.Domain
   )}
```

Listing 7.36. ATL pravilo za mapiranje koncepta **Attribute** u koncept **Attribute**

```
rule PrimitiveDomain2PrimitiveDomain{
from
   pd: RM!PrimitiveDomain
to
   out: IISCase!PrimitiveDomain(
      Name <- pd.Name,
      DefaultValue <- pd.DefaultValue,
      DefaultLength <- pd.DefaultLength,
      DefaultDecimalPlaces <- pd.DefaultDecimalPlaces
   )}
```

Listing 7.37. ATL pravilo za mapiranje koncepta **PrimitiveDomain** u koncept **PrimitiveDomain**

```
rule UserDefDomFromPrim2UserDefDomFromPrim{
from
   upd: RM!UserDefDomFromPrimitiv
to
   out: IISCase!UserDefinedDomainFromPrimitiveDomain(
      Name <- upd.Name,
      DefaultValue <- upd.DefaultValue,
      LengthValue <- upd.Length,
      DecimalPLaces <- upd.DecimalPlaces,
      CheckCondition <- upd.Condition,
      InheritsPrimitiveDomain <- upd.Inherits
   )}
```

Listing 7.38. ATL pravilo za mapiranje koncepta **UserDefDomFromPrim** u koncept **UserDefinedDomainFromPrimitiveDomain**

```
rule UserDefDomFromUserDef2UserDefDomFromUserDef{
from
   ud: RM!UserDefDomFromUserDef
to
   out: IISCase!UserDefinedDomainFromUserDefinedDomain(
      Name <-ud.Name,
      DefaultValue <- ud.DefaultValue,
      CheckCondition <- ud.Condition,
      InheritsUserDefinedDomain <- ud.Inherits
   )}
```

Listing 7.39. ATL pravilo za mapiranje koncepta **UserDefDomFromUserDef** u koncept **UserDefinedDomainFromUserDefinedDomain**

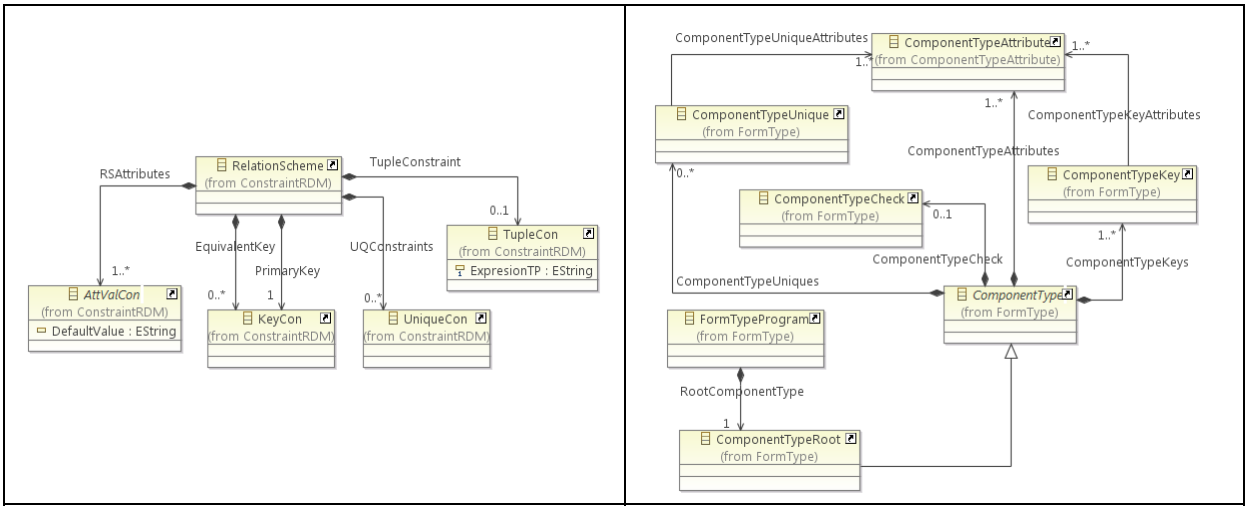
```
rule InclusionDependency2InclusionDependency{
from
```

```
   id: RM!InclusionDependencyURS
to
   out:IISCase!InclusionDependency(
      Name <- id.Name,
      LeftSide <- id.LHS_IND_URS,
      RiightSide <- id.RHS_IND_URS
   )}
```

Listing 7.40. ATL pravilo za mapiranje koncepta I**nclusionDependencyURS** u koncept **InclusionDependency**

*7.3.4. $\mathcal{F}$ _***Basic** *tip forme*

U procesu transformacije, za sve šeme relacija koje postoje u šemi baze podataka nad kojom se vrši reverzni inženjering, kreira se tip forme $\mathcal{F}$, vrste $\mathcal{F}$ *_Basic*. Na slici 7.31 prikazani su delovi meta-modela sa konceptima koji su obuhvaćeni algoritmom za mapiranje šeme relacije u $\mathcal{F}$ *_Basic* tip forme. Mapiranje se vrši pomoću procedure *KreiranjeF_Basic*, čiji pseudokod je prikazan na slici 7.32.



a) RMM | b) IISCaseMM

Slika 7.31. Delovi meta-modela za mapiranje koncepta *RelationScheme* u $\mathcal{F}$ *_Basic* tip forme

Skup šema relacija *S* predstavlja ulazni parametar procedure *KreiranjeF_Basic*, dok je rezultat transformacije skup tipova formi tipa $\mathcal{F}$ *_Basic*. Nazivu šeme relacije dodaje se prefiks *FormType_* čime se formira naziv tipa forme (*Name*), dok naslov tipa forme (*Title*) dobija istu vrednost kao i naziv.

Pri specifikaciji tipa forme u *forward* inženjeringu zadaje se da li se tip forme koristi u postupku projektovanja baze podataka, što je reprezentovano osobinom *ConsideredINDBSchDesign*. Vrednost *ConsideredINDBSchDesign* može biti postavljena na *true* ili *false*. U obrnutom smeru, kada se sprovodi reverzni inženjering šeme baze podataka, ova osobina će uvek imati vrednost *true* (slika 7.32).

Učestalost upotrebe (*Frequency*) i vreme odgovora (*ResponseTime*) inicijalizovane su na vrednost jedan. Ovu vrednost, kao i ostale, nakon transformacije, putem korisničkog interfejsa alata IIS*Case, korisnik može da promeni.

Korenski tip komponente, koji je u ovom tipu forme i jedini, kreira se pomoću procedure *KreiranjeKorenskogC_bezPotomaka*. Pseudokod procedure prikazan je na slici 7.33.

Procedura:	*KreiranjeF_Basic*
Formalni parametri:	
Ulazni:	$S = \{N_i(R_i, O^{RS}_i) \mid i = 1,.., n\}$ $O^{RS} = K^{RS} \cup UQ^{RS} \cup CH^{RS}$
Izlazni:	$FT_{\mathcal{B}} = \{\mathcal{F}_i \mid i = 1,.., n\}$
Lokalne deklaracije: *ct* – izlazni parametar iz procedure *KreiranjeKorenskogC_bezPotomaka* (*ct* –korenski tip komponente)	
Pseudokod: **POČETAK PROCESA** KreirajTipoveFormi_Basic **RADI** kreiranjeF ($\forall N \in S$) **POSTAVI** *Name* ←'FormType_' + *N* **POSTAVI** *Title* ←'FormType_' + *N* **POSTAVI** *ConsideredINDBSchDesign* ←*true* **POSTAVI** *Frequency* ← 1 **POSTAVI** *ResponseTime* ← 1 **POZOVI** *KreiranjeKorenskogC_bezPotomaka*(*N*, *ct*) **POSTAVI** *RootComponentType* ← *ct* **KRAJ RADI** kreiranjeF **KRAJ PROCESA** KreirajTipoveFormi_Basic	

Slika 7.32. Algoritam procedure za mapiranje šema relacija u $\mathcal{F}$ *_Basic* tipove formi

Ulazni parametar procedure *KreiranjeKorenskogC_bezPotomaka* je šema relacije, koja se prosleđuje pri pozivu procedure. Šema relacije je definisana svojim nazivom *N*, skupovima atributa (*R*) sa ograničenjem domena i nula vrednosti i skupom relacionih ograničenja (O^{RS}). Skup relacionih ograničenja čine: skup ograničenja ključeva (K^{RS}), skup ograničenja jedinstvenosti (UQ^{RS}) i ograničenje torke (CH^{RS}). Izlazni parametar procedure je korenski tip komponente (*C*), koji nema potomaka.

Nazivu šeme relacije dodaje se prefiks *ComponentType_* čime se formira naziv tipa komponente (*Name*), dok naslov tipa komponente (*Title*) dobija istu vrednost kao i naziv. Skup ključeva, skup ograničenja jedinstvenosti i ograničenje torke šeme relacije, postaju, redom, skup ključeva, skup ograničenja jedinstvenosti i ograničenje torke tipa komponente. Dozvoljene operacije nad bazom podataka (*delete, insert, update*) moraju biti pridružene svakom tipu komponente. Inicijalizuju se na nedozvoljenu vrednost (*false*), osim operacije čitanja (*retrieve*) iz baze podataka koja je inicijalno jedina dozvoljena (*true*).

Procedura:	*KreiranjeKorenskogC_bezPotomaka*
Formalni parametri:	
Ulazni:	$N(R, O^{RS})$ $O^{RS} = K^{RS} \cup UQ^{RS} \cup CH^{RS}$
Izlazni:	$C(Q, O)$ $O = K \cup UQ \cup CH$

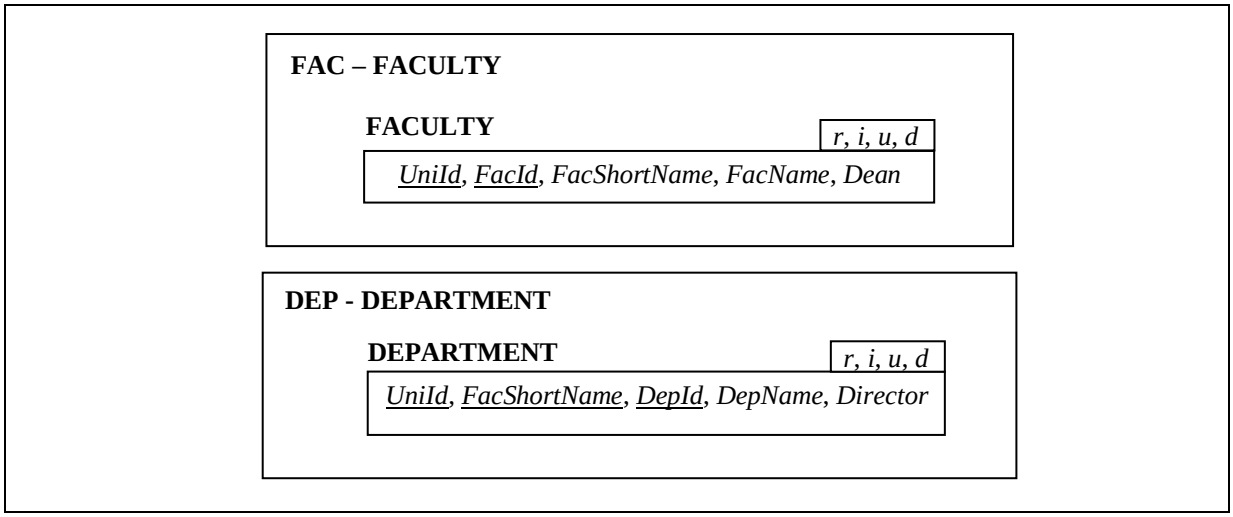
Pseudokod:

POČETAK PROCESA KreiranjeC_bezPotomaka
  RADI kreiranjeC
    POSTAVI *Name* ←'ComponentType_' + *N*
    POSTAVI *Title* ←'ComponentType_' + *N*
    POSTAVI $Q \leftarrow R$
    POSTAVI $K \leftarrow K^{RS}$
    POSTAVI $UQ \leftarrow UQ^{RS}$
    POSTAVI $CH \leftarrow CH^{RS}$
    POSTAVI *Retrieve* ← true
    POSTAVI *Delete* ← false
    POSTAVI *Insert* ← false
    POSTAVI *Update* ← false
  KRAJ RADI kreiranjeC
KRAJ PROCESA KreiranjeC_bezPotomaka

Slika 7.33. Algoritam procedure *KreiranjeKorenskogC_bezPotomaka*

Primer 7.7. Date su šeme relacija baze podataka *UniversityDb*:

- *Faculty*({*UniId, FacId, FacShortName, FacName, Dean*},
  {PrimaryKey(*UniId+FacId*), EquivalentKey(*UniId+FacShortName*)}) i
- *Department*({*UniId, FacShortName, DepId, DepName, Director*},
  {PrimaryKey(*UniId + FacShortName + DepId*)}).



Slika 7.34. $\mathcal{F}$ *_Basic* tipovi formi nastali od šema relacija *Faculty* i *Department*

Na osnovu ovih šema relacija i prethodno opisanih alagoritama, biće kreirani tipovi formi: **FAC – FACULTY** i **DEP – DEPARTMENT,** prikazani na slici 7.34.

Definicija opisanog mapiranja implementirana je pomoću ATL pravila *RelationScheme2ComponentType* prikazanog na listingu 7.41. Primenom ovog pravila, od šeme relacije (*rs*) nastaje tip forme (*ft*) sa korenskim tipom komponente (*ctr*).

U uvodu odeljka objašnjeno je da izabrani pristup reverznom inženjeringu obuhvata kreiranje relevantnih kombinacija tipova formi, što prouzrokuje da se jedna šema relacije može pojaviti više puta kao izvor za kreiranje tipova komponenti koji će pripadati različitim tipovima formi.

Svaki tip komponente ima svoj skup atributa koji pripadaju skupu atributa na nivou čitavog modela aplikacije. Takođe, poseduje i skup ograničenja, zbog čega bi trebalo na istim instancama atributa i ograničenja šeme relacije više puta primeniti pravilo za mapiranje. Zbog mogućnosti ATL jezika, koji dozvoljava da se jedno pravilo za uparivanje može primeniti samo jednom na jednu instancu elementa ulaznog modela, u implementaciji transformacije nisu mogla biti korišćena *matched* pravila, već samo *lazy* pravila za mapiranje koja se mogu pozivati više puta i koja će pri svakom pozivu kreirati novu instancu izlaznog modela.

Iz istog razloga su *lazy* pravila korišćena za mapiranje skupa ograničenja šeme relacije na skup ograničenja tipa komponente. U ograničenjima se mora specificirati skup atributa nad kojim je ograničenje definisano. Skup atributa ograničenja pripada skupu atributa tipa komponente koji je prethodno kreiran. ATL ne omogućava referenciranje novokreiranih elemenata ciljnog modela pomoću deklarativnih mehanizama. Zbog toga se uvodi ATL pomoćni atribut *attributes* koji će čuvati skup atributa tipa komponente, prikazan na listingu 7.42. Ovaj atribut se inicijalizuje na prazan skup u imperativnom bloku pravila *RelationScheme2ComponentType* nakon svakog kreiranja jedne instance tipa forme, dok mu se u imperativnom bloku pravila za kreiranje atributa tipa komponente, nakon svakog kreiranja instance atributa, dodeljuje referenca na taj atribut.

```
rule RelationScheme2ComponentType{
from
   rs:RM!RelationScheme
to
   ft: IISCase!FormTypeProgram(
      Name <- 'FormType_' + rs.Name,
      Title <- 'FormType_' + rs.Name,
      ConsideredINDBSchDesign <- true,
      Frequency <- 1,
      ResponseTime <- 1,
      RootComponentType <- ct
   ),
   ctr: IISCase!ComponentTypeRoot(
      Name <- 'ComponentType_' + rs.Name,
      Title <- 'ComponentType_' + rs.Name,
      ComponentTypeAttributes <- rs.RSAttributes->collect(e|
         if (e.oclIsTypeOf(RM!NotNullAttr)) then
            thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
            thisModule.RSAttributes2NullCompTypeAttribute(e) endif),
      ComponentTypeKeys <- (rs.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(rs.PrimaryKey)),
      ComponentTypeUniques <- rs.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e,rs)),
      ComponentTypeCheck <-
         thisModule.TupleConstraints2ComponentTypeCheckCon(rs.TupleConstraint),
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false
```

```
    )
    do{
        thisModule.attributes <- Sequence{};
    }
}
```

Listing 7.41. ATL pravilo za mapiranje koncepta **RelationScheme** u koncept **ComponentTypeRoot**

```
helper def:attributes:Sequence(IISCase!ComponentTypeAttribute) = Sequence{};
```

Listing 7.42. ATL pomoćni atribut *attributes* koji predstavlja skup atributa tipa komponente

Skup atributa šeme relacije podeljen je na skup atributa koji imaju zabranjenu nula vrednost i skup atributa kojima je nula vrednost dozvoljena. Od ova dva skupa generiše se skup atributa tipa komponente. *Lazy* pravila za mapiranje atributa šeme relacije na atribut tipa komponente prikazana su na listinzima 7.43 i 7.44.

```
lazy rule RSAttributes2NullCompTypeAttribute{
from
    a:RM!NullAttr
to
    cta:IISCase!ComponentTypeAttribute(
        Title <- a.AttributeName.Name,
        ComponentTypeAttributeName <- a.AttributeName,
        Mandatory <- false,
        DeafultValue <- a.DefaultValue
    )
    do{
        thisModule.attributes <- thisModule.attributes.append(cta);
    }
}
```

Listing 7.43. ATL *lazy* pravilo za mapiranje koncepta **NullAttr** u koncept **ComponentTypeAttribute**

```
lazy rule RSAttributes2NotNullCompTypeAttribute{
from
    a:RM!NotNullAttr
to
        cta:IISCase!ComponentTypeAttribute(
        Title <- a.AttributeName.Name,
        ComponentTypeAttributeName <- a.AttributeName,
        Mandatory <- true,
        DeafultValue <- a.DefaultValue
    )
    do{
        thisModule.attributes <- thisModule.attributes.append(cta);
    }
}
```

Listing 7.44. ATL *lazy* pravilo za mapiranje koncepta **NotNullAttr** u koncept **ComponentTypeAttribute**

Lazy pravila za mapiranje relacionih ograničenja na ograničenja tipa komponente prikazana su na listinzima 7.45, 7.46 i 7.48.

Nazivi ograničenja šeme relacije postaju nazivi ograničenja tipa komponente. Ključevi mogu biti označeni kao *lokalni* ili *globalni*. To neće praviti razliku u implementacionoj šemi matičnog aplikativnog sistema. Međutim, ukoliko je tip forme referenciran nekim drugim aplikativnim sistemom ili je dati aplikativni sistem referenciran nekim drugim aplikativnim sistemom, situacija će biti drugačija. Lokalni ključ koji pripada tipu forme van matičnog aplikativnog sistema biće prepoznat kao ograničenje jedinstvene vrednosti. U procesu reverznog inženjeringa ključ se označava kao globalni, odnosno atribut *Global* se postavlja na vrednost *true*.

Skup atributa ograničenja ključa i ograničenja jedinstvenosti inicijalizuje se u imperativnom bloku pravila pomoću *getKeyAttributes* i *getUniqueAttributes* metoda koje su prikazane na listingu 7.47. Pomoću ovih metoda, iz skupa atributa koji pripadaju tipu komponenete, a koji odgovaraju skupu atributa ograničenja šeme relacije, izdvaja se skup atributa koji pripadaju odgovarajućem ograničenju tipa komponente.

```
lazy rule EquivalentKey2CompTypeKey{
from
   ek:RM!KeyCon
to
   out:IISCase!ComponentTypeKey(
      Name <- ek.Name,
      Global <- true
   )
   do{
      out.ComponentTypeKeyAttributes <- thisModule.getKeyAttributes(ek);
   }
}
```

Listing 7.45. ATL *lazy* pravilo za mapiranje koncepta **KeyCon** u koncept **ComponentTypeKey**

```
lazy rule UniqueCon2ComponentTypeUnique{
from
   uq:RM!UniqueCon
to
   out:IISCase!ComponentTypeUnique(
      Name <- uq.Name
   )
   do{
      out.ComponentTypeUniqueAttributes <- thisModule.getUniqueAttributes(uq);
   }
}
```

Listing 7.46. ATL *lazy* pravilo za mapiranje koncepta **UniqueCon** u koncept **ComponentTypeUnique**

```
helper def: getKeyAttributes(kc:RM!KeyCon): Set(IISCase!ComponentTypeAttribute) =
let kAttrName:Set(String) = kc.KeyAttr ->
collect(e | e.AttributeName.refGetValue('Name'))->asSet() in
   let cAttr: Set(IISCase!ComponentTypeAttribute)= thisModule.attributes in
      cAttr -> iterate(c; rez:Set(IISCase!ComponentTypeAttribute) = Set{} |
         if kAttrName.includes(c.ComponentTypeAttributeName.refGetValue('Name'))
         then rez.including(c) else rez endif );

helper def: getUniqueAttributes(kc:RM!UniqueCon):
Set(IISCase!ComponentTypeAttribute) = let kAttrName:Set(String) =  kc.UQConNotNullAttr ->
collect(e | e.AttributeName.refGetValue('Name'))
```

```
->asSet().union(kc.UQConNullAttr -> collect(e | e.refGetValue('Name'))
->asSet()) in
   let cAttr: Set(IISCase!ComponentTypeAttribute)= thisModule.attributes in
      cAttr -> iterate(c; rez:Set(IISCase!ComponentTypeAttribute) = Set{} |
         if kAttrName.includes(c.ComponentTypeAttributeName.refGetValue('Name'))
         then rez.including(c) else rez endif);
```

Listing 7.47. ATL *helper*-i za pronalaženje atributa ograničenja

Pri mapiranju ograničenja torke, logički izraz ograničenja torke iz Generičkog meta-modela (*ExpresionTP*) postaje logički izraz ograničenja torke IISCase meta-modela (*CheckCondition*). ATL pravilo za mapiranje ograničenja torke prikazano je na listingu 6.48.

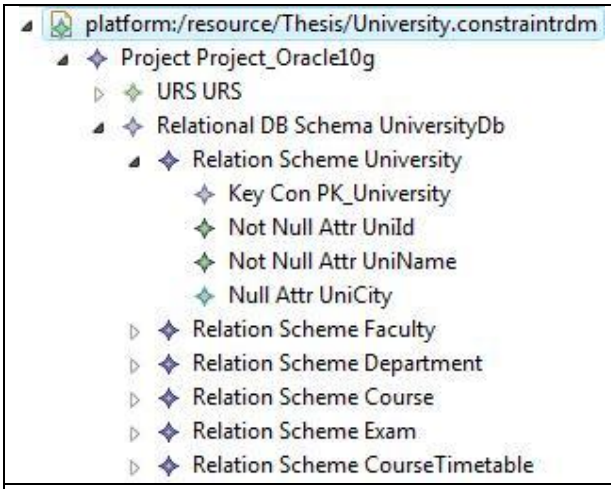
```
lazy rule TupleConstraints2ComponentTypeCheckCon{
from
   ch: RM!TupleCon
to
   out: IISCase!ComponentTypeCheck(
      Name <- ch.Name,
      CheckCondition <- ch.ExpresionTP
   )
}
```

Listing 7.48. ATL *lazy* pravilo za mapiranje koncepta **TupleCon** u koncept **ComponentTypeCheck**

Primer 7.8. Dat je skup sledećih šema relacija i relacionih ograničenja:

- *University*({*UniId, UniName, UniCity, RectorId*}, {PrimaryKey(*UniId*)})
- *Faculty*({*UniId, FacId, FacShortName, FacName, Dean*}, {PrimaryKey(*UniId, FacId*)})
- *Department*({*UniId, FacId, DepId, DepName, DeanId*}, {PrimaryKey(*UniId, FacId, DepId*)})
- *Course*({*CourseId, CourseShortName, CourseName, Lecturer, Prerequisite, Year, Semester*}, {PrimaryKey(*CourseId*)}),
- *Exam*({*CourseId, ExamId, Examiner, DateTime*}, {PrimaryKey(*CourseId, ExamId*)})
- *CourseTimetable*({*CourseId, Day, Time, Classroom*}, {PrimaryKey(*CourseId, Day, Time*)}).

Na slici 7.35 prikazan je segment šeme baze podataka koji ilustruje primenu prethodno opisanih pravila transformacije. U izvornom modelu prikazane su navedene šeme relacija sa relacionim ograničenjima, slika 7.35 a). Nakon transformacije dobija se model prikazan na slici 7.35 b).



a) Model dela *UniversityDb* u skladu sa RMM

b) Model dela *UniversityDb* u skladu sa IISCaseMM

Slika 7.35. Deo *UniversityDb* relacione šeme baze podataka u Eclipse editoru

Na slici 7.35 a) može se videti da je od šeme relacije *University* nastao tip forme *FormType_University* koji ima samo jedan korenski tip komponente *ComponentType_University*. Na isti način, i od ostalih šema relacija nastao je po jedan tip forme sa jednim korenskim tipom komponente.

Na listinzima 7.49 i 7.50 prikazani su primeri sa slike 7.35 a) i b) u XML XMI formatu, respektivno.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConstraintRDM:Project xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ConstraintRDM="http://relacionimm/1.0/ConstraintRDM"
xmlns:INDConstraint="http://relacionimm/1.0/INDConstraint"
xmlns:RelationConstraint="http://relacionimm/1.0/RelationConstraint"
xmlns:URSConstraint="http://relacionimm/1.0/URSConstraint" Name="Project_Oracle10g">
  <URSs Name="URS">
    ...
  </URSs>
  <DBSchema Name="UniversityDb">
    <RelationSchemes Name="University">
      <PrimaryKey Name="PK_University"
                      KeyAttr="//@DBSchema/@RelationSchemes.0/@RSAttributes.0"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                  AttributeName="//@URSs/@URSAttributes.11"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniName"
                                  AttributeName="//@URSs/@URSAttributes.24"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="UniCity"
                                  AttributeName="//@URSs/@URSAttributes.7"/>
    </RelationSchemes>
    <RelationSchemes Name="Employee">
      ...
    </RelationSchemes>
    <RelationSchemes Name="Faculty">
      ...
    </RelationSchemes>
    <RelationSchemes Name="Department">
      ...
    </RelationSchemes>
    <RelationSchemes Name="Course">
      ...
    </RelationSchemes>
```

```
    <RelationSchemes Name="Exam">
      ...
    </RelationSchemes>
    <RelationSchemes Name="CourseTimetable">
      ...
    </RelationSchemes>
   ...
  </DBSchema>
</ConstraintRDM:Project>
```

Listing 7.49. Model dela *UniversityDb* u skladu sa RMM u XML XMI formatu

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project:ISApplicationModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:FormType="http://model/1.0/FormType" xmlns:atribute="http://model/1.0/attribute"
xmlns:domain="http://model/1.0/domain" xmlns:project="http://model/1.0/project"
Name="IISCase_Project_Oracle10g">
  <FundamentalConcepts>
   ...
  </FundamentalConcepts>
  <ApplicationSystems Name="AppSystem_UniversityDb">
    <OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name="FormType_University"
       Title="FormType_University" Frequency="1" ResponseTime="1"
                                              ConsideredINDBSchDesign="true">
      <RootComponentType Name="ComponentType_University" Title="ComponentType_University"
                                                                  Query="true">
       <ComponentTypeAttributes ComponentTypeAttributeName=
      "//@FundamentalConcepts.0/@FundamentalConcept.11" Title="UniId" Mandatory="true"/>
       <ComponentTypeAttributes ComponentTypeAttributeName=
      "//@FundamentalConcepts.0/@FundamentalConcept.24" Title="UniName" Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
      "//@FundamentalConcepts.0/@FundamentalConcept.7" Title="UniCity"/>
      <ComponentTypeKeys Name="PK_University"
        ComponentTypeKeyAttributes="//@ApplicationSystems.0/@OwnedFormTypes.0/
          @RootComponentType/@ComponentTypeAttributes.0" Global="true"/>
      </RootComponentType>
    </OwnedFormTypes>
    <OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name="FormType_Employee"
     Title="FormType_Employee" Frequency="1" ResponseTime="1"
                                              ConsideredINDBSchDesign="true">
    </OwnedFormTypes>
    <OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name="FormType_Faculty"
       Title="FormType_Faculty" Frequency="1" ResponseTime="1"
                                              ConsideredINDBSchDesign="true">
     ...
    </OwnedFormTypes>
     ...
  </ApplicationSystems>
</project:ISApplicationModel>
```

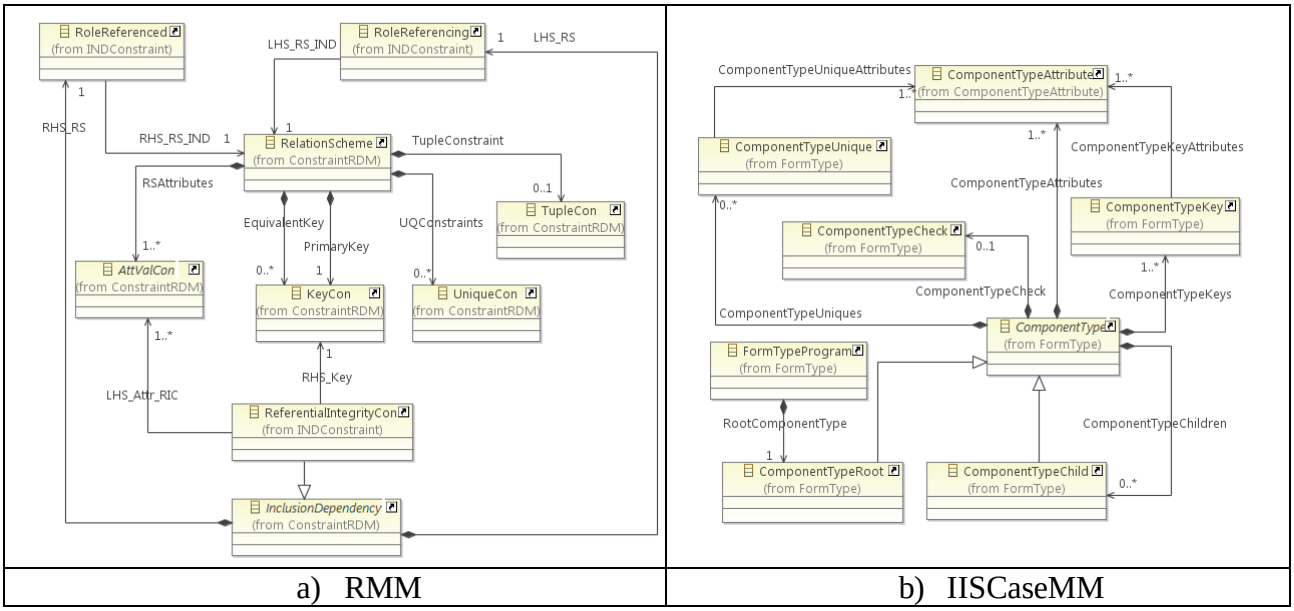
Listing 7.50. Model dela *UniversityDb* u skladu sa IISCaseMM u XML XMI formatu

7.3.5. $\mathcal{F}$ _Tree² tip forme

U ovoj tački biće prezentovani algoritmi za generisanje $\mathcal{F}$ *_Tree²* tipova formi. Ova vrsta tipova formi nastaje na osnovu skupa ograničenja referencijalnog integriteta u kojem je šema relacije sa leve strane ograničenja tipa *WR*, tj. slaba šema relacije.

Pseudokod algoritma za mapiranje šema relacija, koje pripadaju ograničenju referencijalnog integriteta, u $\mathcal{F}$ *_Tree²* tip fome prikazan je na slici 7.37. Na slici 7.36

prikazani su delovi meta-modela sa konceptima koji su obuhvaćeni ovim algoritmom za mapiranje.



a) RMM | b) IISCaseMM

Slika 7.36. Delovi meta-modela za mapiranje odgovarajućih koncepata u koncept tipa forme

Skup ograničenja referencijalnog integriteta, kao i skup šema relacija koje se nalaze na levoj i desnoj strani ograničenja, predstavljaju ulazne podatke u algoritam, dok su rezultat transformacije tipovi formi tipa $\mathcal{F}\_Tree^2$.

Za svako ograničenje iz skupa ograničenja referencijalnog integriteta:

$$RIC = \{ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$$

proverava se da li je šema relacije N_l sa leve strane datog ograničenja referencijalnog integriteta tipa *WR*, tj. slaba šema relacije (definicija slabe šeme relacije data je u tački 7.3.1). Ukoliko šema relacije N_l jeste slaba (ind = 1), tada od para šema relacija (N_l, N_r) putem transformacije nastaje tip forme, gde se od šeme relacije N_r dobija korenski tip komponente, a od šeme relacije N_l tip komponente potomak.

Procedura:	*KreiranjeF_Tree*^{2}
Formalni parametri:	
Ulazni:	$RIC = \{ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $S = \{N_i(R_i, O^{RS}{}_i) \mid i = 1,.., n\}$ $O^{RS} = K^{RS} \cup UQ^{RS} \cup CH^{RS}$
Izlazni:	$FT_{T2} = \{ \mathcal{F}_i \mid i = 1,.., l\}$

Lokalne deklaracije:
ind – izlazni parametar iz procedure *ProveravanjeKandidataZaF*
(*ind* = 1 – jeste kandidat za kreiranje $\mathcal{F}$,
ind = 0 – nije kandidat za kreiranje $\mathcal{F}$)
ct – izlazni parametar iz procedure *KreiranjeKorenskogC_zaF_Tree*^{2}
(*ct* – korenski tip komponente)

Pseudokod:

```
POČETAK PROCESA KreirajTipoveFormi_Tree²
   RADI kreiranjeF (∀ric ∈ RIC)
      POZOVI ProveravanjeKandidataZaF(ric, N_l, ind)
      AKO JE ind = 1 TADA
         POSTAVI Name ←'FormType_' + ric
         POSTAVI Title ←'FormType_' + ric
         POSTAVI ConsideredINDBSchDesign ←true
         POSTAVI Frequency ← 1
         POSTAVI ResponseTime ← 1
         POZOVI KreiranjeKorenskogC_zaF_Tree²(RIC, S, ric, N_r, N_l, ct)
         POSTAVI RootComponentType ← ct
      KRAJ AKO
   KRAJ RADI kreiranjeF
KRAJ PROCESA KreirajTipoveFormi_Tree²
```

Slika 7.37. Algoritam procedure *KreiranjeF_Tree*^{2}

Provera da li je šema relacije N_l slaba, tj. da li je ograničenje referencijalnog integriteta, u kojem se ova šema relacije nalazi sa leve strane, kandidat za kreiranje tipa forme, vrši se pomoću procedure *ProveravanjeKandidataZaF*, čiji algoritam je prikazan na slici 7.38. Proceduri se, kao ulazni parametri, prosleđuju: ograničenje referencijalnog integriteta i šema relacije sa leve strane ograničenja. Ukoliko je skup atributa sa leve strane ograničenja referencijalnog integriteta, *LHS*, pravi podskup skupa atributa koji pripadaju ključu te šeme relacije (PKA_l), šema relacije N_l je tipa *WR* i indikator *ind* se postavlja na vrednost 1. U suprotnom, indikator *ind* se postavlja na vrednost 0.

Nazivu ograničenja referencijalnog integirteta dodaje se prefiks *FormTupe_* čime se formira naziv tipa forme (*Name*), dok naslov tipa forme (*Title*) dobija istu vrednost kao i naziv. Objašnjenje za *ConsideredINDBSchDesign*, *Frequency* i *ResponseTime* je identično kao u prethodno opisanom mapiranju za $\mathcal{F}$ *_Basic* tipove formi.

Procedura:	*ProveravanjeKandidataZaF*
Formalni parametri:	
Ulazni:	*ric*: $N_l[LHS] \subseteq N_r[RHS]$ $N_l(R_l, PK_l)$
Izlazni:	*ind*
Detalji specifikacije: $PKA_l = \{A_1, .., A_k\}, A_i \in PK_l$	
Pseudokod: **POČETAK PROCESA** ProveravanjeKandidataZaF **RADI** proveru ***AKO JE*** $LHS / PKA_l = \varnothing \wedge$ **TADA** **POSTAVI** $ind \leftarrow 1$ **INAČE** **POSTAVI** $ind \leftarrow 0$ ***KRAJ AKO*** **KRAJ RADI** proveru **KRAJ PROCESA** ProveravanjeKandidataZaF	

Slika 7.38. Algoritam procedure *ProveravanjeKandidataZaF*

Mapiranje šeme relacije N_r koja se nalazi na desnoj strani ograničenja referencijalnog integriteta na korenski tip komponente vrši se pomoću procedure *KreiranjeKorenskogC_zaTree*^{2}, čiji je pseudokod algoritma prikazan na slici 6.39. Algoritam procedure za kreiranje korenskog tipa komponente za $\mathcal{F}$ *_Tree*^{2} tip forme veoma je sličan algoritmu za kreiranje korenskog tipa komponente $\mathcal{F}$ *_Basic* tipova formi, *KreiranjeKorenskogC_bezPotomaka*, prikazanom na slici 7.33. Osnovna razlika je u tome što korenski tip komponente za $\mathcal{F}$ *_Tree*^{2} tip forme ima tip komponente koji je potomak.

Prvi ulazni parametar procedure *KreiranjeKorenskogC_zaTree*^{2} je šema relacije N_r koja se nalazi na desnoj strani ograničenja referencijalnog integriteta i od koje nastaje korenski tip komponente. Ostali ulazni parametri procedure, koji su takođe i ulazni paramteri procedure *KreiranjeC_zaF_Tree*^{2} koja se poziva iz procedure *KreiranjeKorenskogC_zaTree*^{2} su sledeći: šema relacije sa leve strane ograničenja N_l, skup svih šema relacija i skup svih ograničenja referencijalnog integriteta RIC. Algoritam procedure *KreiranjeC_zaF_Tree*^{2} biće objašnjen u tekstu koji sledi. Izlazni parametar procedure *KreiranjeKorenskogC_zaTree*^{2} je korenski tip komponente (*C*) sa jednim potomkom.

Nazivu šeme relacije N_r dodaje se prefiks *ComponentType_* čime se formira naziv tipa komponente (*Name*), dok naslov tipa komponente (*Title*) dobija istu vrednost kao i naziv. Skup ključeva, skup ograničenja jedinstvenosti i ograničenje torke šeme relacije N_r, postaje skup ključeva, skup ograničenja jedinstvenosti i ograničenje torke korenskog tipa komponente. Dozvoljene operacije nad bazom podataka (*Delete, Insert, Update*), koje moraju biti pridružene svakom tipu komponente, inicijalizuju se na nedozvoljenu vrednost (*false*), osim operacije čitanja (*Query*) iz baze podataka koja je inicijalno dozvoljena (*true*).

Procedura:	*KreiranjeKorenskogC_zaF_Tree*^{2}
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $S = \{N_i(R_i, K_i) \mid i = 1,.., n\}$ $ric: N_l[LHS] \subseteq N_r[RHS]$ $N_r(R_r, O_r)$,$O_r = K_r \cup UQ_r \cup CH_r$ $N_l(R_l, O_l)$,$O_l = K_l \cup UQ_l \cup CH_l$
Izlazni:	$C(Q, O_C)$ $O_C = K_C \cup UQ_C \cup CH_C$
Lokalne deklaracije: *ct* – izlazni parametar iz procedure *KreiranjeC_zaF_Tree*^{2} (*ct* – tip komponente koji je direktni potomak korenskog tipa komponente)	
Pseudokod: **POČETAK PROCESA** KreiranjeKorenskogC **RADI** kreiranjeC **POSTAVI** *Name* ←'ComponentType_' + N_r **POSTAVI** *Title* ← 'ComponentType_' + N_r **POSTAVI** $Q \leftarrow R_r$ **POSTAVI** $K_C \leftarrow K_r$ **POSTAVI** $UQ_C \leftarrow UQ_r$ **POSTAVI** $CH_C \leftarrow CH_r$ **POSTAVI** *Query* ← true **POSTAVI** *Delete* ← false **POSTAVI** *Insert* ← false **POSTAVI** *Update* ← false **POZOVI** *KreiranjeC_zaF_Tree*^{2}(*RIC, S, ric,* N_r, N_l, *ct*) **POSTAVI** *ComponentTypeChildren* ← *ct* **KRAJ RADI** kreiranjeC **KRAJ PROCESA** KreiranjeKorenskogC	

Slika 7.39. Algoritam procedure *KreiranjeKorenskogC_zaF_Tree*^{2}

Kreiranje tipa komponente potomka vrši se procedurom *KreiranjeC_zaF_Tree*^{2}, čiji algoritam je prikazan na slici 7.40.

Tip komponente potomak nastaje od šeme relacije N_l, koja se nalazi na levoj strani ograničenja referencijalnog integriteta. Naziv i naslov se formiraju kao i kod kreiranja korenskog tipa komponente čiji algoritam je prikazan na slici 7.39, sa razlikom da se sada koristi naziv šeme relacije sa leve strane ograničenja referencijalnog integritera, N_l.

Definisanje skupa atributa tipa komponente potomka vrši se tako što se prvo proverava da li je šema relacije tipa *AK*, odnosno proverava se da li N_l sadrži samo ključeve drugih šema relacija. Ova provera vrši se pomoću procedure *ProveravanajeSamoKljučevi*, čiji algoritam je prikazan na slici 7.41.

U slučaju da šema relacije N_l nije tipa *AK*, zbog uslova kojim je definisano da presek skupova atributa tipova komponenti na jednom tipu forme mora biti prazan skup, skup atributa tipa komponente formira se tako što se iz skupa atributa šeme relacije N_l izbacuje skup atributa stranog ključa propagiranog iz šeme relacije N_r od koje je nastao korenski tip komponente. Ovo je u algoritmu sa slike 7.40 predstavljeno izrazom

$$Q \leftarrow R_l/LHS.$$

Ukoliko šema relacije N_l jeste tipa *AK*, skup atributa koji će postati atributi tipa komponente potomka formira se pomoću procedure *PronalaženjeAtributaC_AK*, čiji algoritam je prikazan na slici 7.42.

Procedura:	*KreiranjeC_zaF_Tree*^{2}
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $S = \{N_i(R_i, K_i) \mid i = 1,.., n\}$ $ric: N_l[LHS] \subseteq N_r[RHS$ $N_r(R_r, O_r)$,$O_r = K_r \cup UQ_r \cup CH_r$ $N_l(R_l, O_l)$,$O_l = K_l \cup UQ_l \cup CH_l$
Izlazni:	$C(Q, O_C)$ $O_C = K_C \cup UQ_C \cup CH_C$
Lokalne deklaracije: *ind* - izlazni parametar iz procedure *ProveravanajeSamoKljučevi* (*ind* = 1 – N_l jeste tipa *AK*, *ind* = 0 – N_l nije tipa *AK*)	

Pseudokod:

POČETAK PROCESA KreiranjeC_zaF_Tree^2
  RADI kreiranjeC
    POSTAVI *Name* ←'ComponentType_' + N_l
    POSTAVI *Title* ←'ComponentType_' + N_l
    POZOVI *ProveravanajeSamoKljučevi*(N_l, *ind*)
    AKO JE *ind* = 0 **TADA**
      POSTAVI $Q \leftarrow R_l/LHS$
    INAČE
      POZOVI *PronalaženjeAtributaC_AK*(*RIC*, *S*, N_l, N_r, *attr*)
      POSTAVI $Q \leftarrow attr$
    KRAJ AKO
    POSTAVI $K_C \leftarrow K_l$
    POSTAVI $UQ_C \leftarrow UQ_l$
    POSTAVI $CH_C \leftarrow CH_l$
    POSTAVI *Query* ← true
    POSTAVI *Delete* ← false
    POSTAVI *Insert* ← false
    POSTAVI *Update* ← false
    AKO ∃ *IRIC* **TADA**
      POSTAVI *NoOfOcurrences* ← (1, M)
    INAČE
      POSTAVI *NoOfOcurrences* ← (0, M)
    KRAJ AKO
  KRAJ RADI kreiranjeC
KRAJ PROCESA KreiranjeC_zaF_Tree^2

Slika 7.40. Algoritam procedure *KreiranjeC_zaF_Tree*^{2}

Ključevi, ograničenja jedinstvenosti, ograničenje torke kao i dozvoljene operacije nad bazom podataka mapirane su na isti način kao i kod korenskog tipa komponente.

Broj pojava tipa komponente (*NoOfOcurrences*) potomka u okviru pojave korenskog tipa komponente, u *forward* inženjeringu definiše se izborom jedne od opcija: 0-N i 1-N. Druga opcija znači da komponenta, koja odgovara datom tipu komponente na odgovarajućoj formi aplikacije, mora biti zadata. Na ovaj način, projektant može zadati ograničenja koja će kroz algoritam integracije biti prepoznata kao ograničenja inverznog referencijalnog integriteta. U obrnutom smeru, pri reverznom inženjeringu, proverava se postojanje ograničenja inveznog referencijalnog integriteta za odgovarajuće ograničenje referencijalnog integriteta. Ukoliko postoji ograničenje inverznog referencijalnog integriteta *NoOfOcurrences* inicijalzuje se na 1-N, a u suprotnom na 0-N.

Procedura:	*ProveravanajeSamoKljučevi*
Formalni parametri:	
Ulazni:	$N_l(R_l, PK_l)$,
Izlazni:	*ind*
Pseudokod:	

```
POČETAK PROCESA ProveravanajeSamoKljučevi
   RADI proveravanje
      AKO JE Nl / PKAl = ∅ TADA
         POSTAVI ind ← 1
      INAČE
         POSTAVI ind ← 0
      KRAJ AKO
   KRAJ RADI proveravanje
KRAJ PROCESA ProveravanajeSamoKljučevi
```

Slika 7.41. Algoritam procedure *ProveravanajeSamoKljučevi*

Procedura:	*PronalaženjeAtributaC_AK*
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $S = \{N_i(R_i, K_i) \mid i = 1,.., n\}$ $N_l(R_l, O_l), N_r(R_r, O_r)$
Izlazni:	$Q=\{A_1, .., A_n\} \subseteq W(F)$
Pseudokod:	

```
POČETAK PROCESA PronalaženjeAtributaC_SamoKljučevi
   POSTAVI Q ← Rl/LHSl
   RADI nalaženjeAtributa (∀ric: Nc[LHSc] ⊆ Np[RHSp] ∈ RIC)
      AKO JE Nl = Nc ∧ Nr ≠ Np TADA
         POSTAVI Q ← Q ∪ (Rp/RHSp)
      KRAJ AKO
   KRAJ RADI nalaženjeAtributa
KRAJ PROCESA PronalaženjeAtributaC_SamoKljučevi
```

Slika 7.42. Algoritam procedure *PronalaženjeAtributaC_AK*

Atributi tipa komponente, koji nastaje od šeme relacije koja sadrži samo ključeve drugih šema relacija, tj. šeme relacije koja povezuje dve ili više drugih šema relacija, formiraju se tako što se pored ključnih atributa koji već pripadaju toj šemi relacije, uključuju i ostali neključni atributi šema relacija koje povezuje (slika 7.42).

Definicija opisanog mapiranja implementirana je pomoću ATL *lazy* pravila *RIC2FormTypes* prikazanog na listingu 7.51. Primenom ovog pravila na instance ograničenja referencijalnog integriteta (*ric*) nastaje tip forme (*ft*) sa korenskim tipom komponente (*ctr*).

Atributi i ograničenja korenskog tipa komponente se kreiraju na isti način kao u ATL pravilu za generisanje $\mathcal{F}$ *_Basic* tipova formi, prikazanom na listingu 7.41, pozivanjem ATL *lazy* pravila prikazanih na listinzima od 7.43 do 7.48.

Za razliku od pravila za generisanje $\mathcal{F}$ *_Basic* tipova formi, u ovom pravilu poziva se ATL *lazy* pravilo *ChildCT* za kreiranje tipa komponente koji je potomak kreiranog korenskog tipa komponente, što je prikazano na listingu 7.52.

```
lazy rule RIC2FormTypes {
from
   ric: RM!ReferentialIntegrityCon
to
   ft: IISCase!FormTypeProgram(
      Name <- 'FormType_' + ric.Name,
      Title <-'FormType_' + ric.Name,
      ConsideredINDBSchDesign <- true,
      Frequency <- 1,
      ResponseTime <- 1,
      RootComponentType <- ctr
   ),
   ctr: IISCase!ComponentTypeRoot(
      Name <- 'ComponentTypeRoot_' + ric.RHS_RS.RHS_RS_IND.Name,
      Title <- 'ComponentTypeRoot_' + ric.RHS_RS.RHS_RS_IND.Name,
      ComponentTypeAttributes <- ric.RHS_RS.RHS_RS_IND.RSAttributes->collect(e|
      if (e.oclIsTypeOf(RM!NotNullAttr)) then
         thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
         thisModule.RSAttributes2NullCompTypeAttribute(e) endif),
      ComponentTypeKeys <- (ric.RHS_RS.RHS_RS_IND.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(ric.RHS_RS.RHS_RS_IND.PrimaryKey))
      ComponentTypeUniques <- ric.RHS_RS.RHS_RS_IND.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e)),
      ComponentTypeCheck <- thisModule.TupleConstraints2ComponentTypeCheckCon(
                                    ric.RHS_RS.RHS_RS_IND.TupleConstraint),
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false
   )
   do{
      thisModule.attributes <- Sequence{};
      ctr.ComponentTypeChildren <- thisModule.ChildCT(ric);
   }
}
```

Listing 7.51. ATL *lazy* pravilo za mapiranje koncepta **ReferentialIntegrityCon** u koncepte **FormTypeProgram** i **ComponentTypeRoot**

```
lazy rule ChildCT{
from
   ric: RM!ReferentialIntegrityCon
to
   ct: IISCase!ComponentTypeChild(
      Name <- 'ComponentType_' + ric.LHS_RS.LHS_RS_IND.Name,
      Title <- 'ComponentType_' + ric.LHS_RS.LHS_RS_IND.Name,
      ComponentTypeAttributes <- if thisModule.RS_AllKeys(ric.LHS_RS.LHS_RS_IND)
      then thisModule.AttributesOfRSs(ric.LHS_RS.LHS_RS_IND,
      ric.RHS_RS.RHS_RS_IND)->collect(e|
      thisModule.RSAttributes2CompTypeAttribute(e))
      else ric.ChildComponentTypeAttribute->collect(e|
         if (e.oclIsTypeOf(RM!NotNullAttr)) then
            thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
            thisModule.RSAttributes2NullCompTypeAttribute(e) endif) endif,
      ComponentTypeKeys <- (ric.LHS_RS.LHS_RS_IND.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(ric.LHS_RS.LHS_RS_IND.PrimaryKey)),
      ComponentTypeUniques <- ric.LHS_RS.LHS_RS_IND.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e)),
      ComponentTypeCheck <- thisModule.TupleConstraints2ComponentTypeCheckCon(
         ric.LHS_RS.LHS_RS_IND.TupleConstraint),
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false,
      NoOfOcurrences <- if thisModule.ExistIRIC(ric) then #OneOrMany else
         #NoneOrMany endif
   )
   do{
      thisModule.attributes <- Sequence{};
   }
}
```

Listing 7.52. ATL *lazy* pravilo za mapiranje koncepta **ReferentialIntegrityCon** u koncept **ComponentTypeChild**

Provera da li je šema relacije od koje nastaje tip komponente potomka tipa *AK* vrši se pomoćnom metodom *RS_AllKeys*, koja je prikazana na listingu 7.53. Ukoliko šema relacije jeste tipa *AK*, skup atributa od kojih će nastati atributi tipa komponente formira se ATL pomoćnom metodom *AttributesOfRSs*, koja je prikazana na listingu 7.54. U suprotnom, skup obeležja činiće sva obeležja šeme relacije, osim atributa koji pripadaju stranom ključu. Za izdvajanje ovih obeležja koristi se ATL pomoćni atribut *ChildComponentTypeAttribute*, prikazan na listingu 7.55.

Nakon izdvajanja atributa iz izvornog modela od kojih će biti kreirani atributi tipa komponente ciljnog modela, pozivaju se *lazy* pravila prikazana na listinzima 7.43 i 7.44. Ograničenja tipa komponente potomka kreiraju se na isti način kao za korenski tip komponenete.

```
helper def: RS_AllKeys(rs:RM!RelationScheme):Boolean=
   let AllRSAttributes:Sequence(RM!AttValCon)=rs.RSAttributes.asSet() in
      if AllRSAttributes-rs.PrimaryKey.KeyAttr.asSet() = Set{} then true
      else false endif;
```

Listing 7.53. ATL *helper* za proveru da li je šema relacije tipa *AK*

```
helper def:
AttributesOfRSs(rs:RM!RelationScheme,parent:RM!RelationScheme):Set(RM!AttValCon)=
let relatedRS:Sequence(RM!RelationScheme)= thisModule.AllRelatedRS(rs,parent) in
   relatedRS->iterate(rsc; rez:Set(RM!AttValCon)= Set{}|
   rez.union(rsc.RSAttributes));

helper def: AllRelatedRS(rs:RM!RelationScheme,parent:RM!RelationScheme):
                                                  Sequence(RM!RelationScheme)=
let allRIC:Sequence(RM!ReferentialIntegrityCon)=
                                    RM!ReferentialIntegrityCon.allInstances() in
   allRIC->iterate(ric; rez: Sequence(RM!RelationScheme) = Sequence{} |
      if ric.LHS_RS.LHS_RS_IND=rs then
         rez.append(ric.RHS_RS.RHS_RS_IND).excluding(parent) else
         rez.excluding(parent) endif);
```

Listing 7.54. ATL *helper* za formiranje skupa atributa od kojih će nastati atributi tipa komponente

```
helper context RM!ReferentialIntegrityCon
def:ChildComponentTypeAttribute:Sequence(RM!AttValCon) =
self.LHS_RS.LHS_RS_IND.RSAttributes ->
   iterate(attr; rez: Sequence (RM!AttValCon) = Sequence{}|
   if self.RHS_Key.KeyAttr->collect(e | e.AttributeName.refGetValue('Name'))
      ->asSet().includes(attr.AttributeName.Name) then rez else
      rez.append(attr) endif);
```

Listing 7.55. ATL pomoćni atribut *ChildComponentTypeAttribute*

Provera postojanja ograničenja inverznog referencijalnog integriteta za odgovarajuće ograničenje referencijalnog integriteta vrši se ATL pomoćnom metodom *ExistIRIC*, koja je prikazana na slici 7.56.

```
helper def:ExistIRIC(ric: RM!ReferentialIntegrityCon):Boolean =
   let allIRIC:Sequence(RM!InverseReferentialIntegrityCon)=
      RM!InverseReferentialIntegrityCon.allInstances() in
      allIRIC->iterate(iric; rez:Boolean = false |
      if iric.RIC = ric then true else false endif);
```

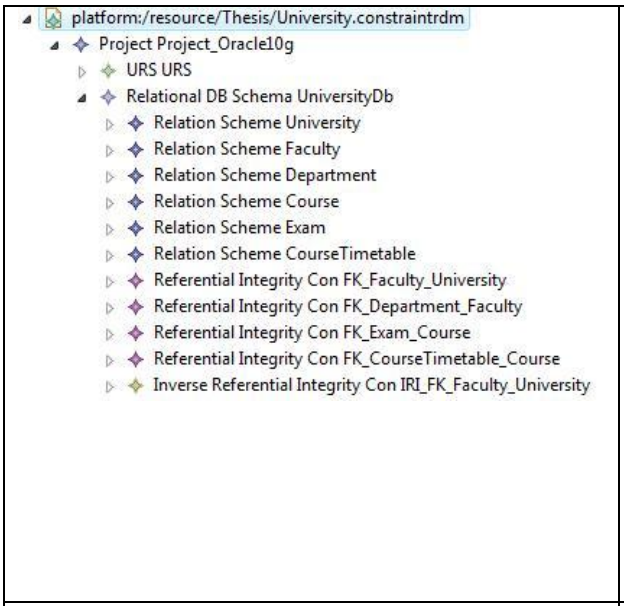
Listing 7.56. ATL *helper* za proveru postojanja ograničenja inverznog referencijalnog integriteta

Primer 7.9. Skup šema relacija iz primera 7.8 proširen je skupom sledećih međurelacionih ograničenja:

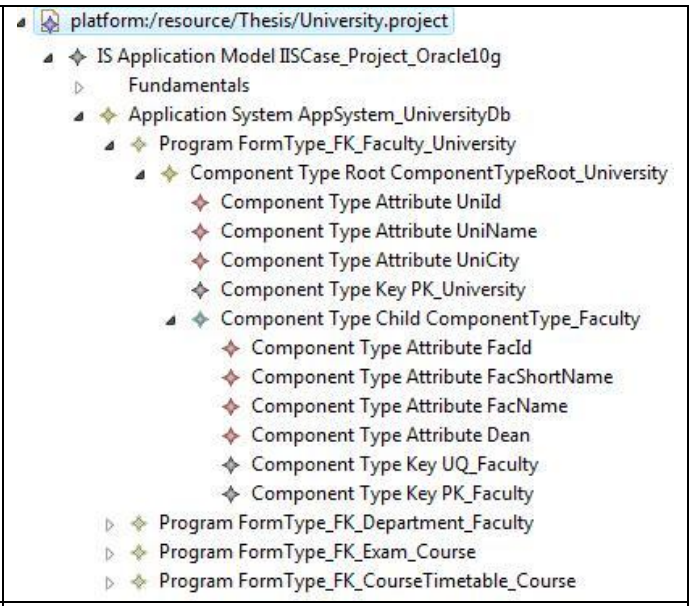
- *FK_ Faculty_University: Faculty*[*FacId*] ⊆ *University*[*UniId*],
- *FK_Department_Faculty*: *Department*[*UniId* + *FacId*] ⊆ *Faculty*[*UniId* + *FacId*],
- *FK_Exam_Course*: *Exam*[*CourseId*] ⊆ *Course*[*CourseId*],
- *FK_ CourseTimetable _ Course*: *CourseTimetable*[*CourseId*] ⊆ *Course*[*CourseId*],
- *IRI_FK_ Faculty_University*: *University*[*UniId*] ⊆ *Faculty*[*UniId*].

Na slici 7.43 prikazan je segment šeme baze podataka koji ilustruje primenu prethodno opisanih pravila transformacije. U izvornom modelu prikazane su navedene šeme relacija sa

relacionim ograničenjima, slika 7.43 a). Nakon transformacije dobija se model prikazan na slici 7.43 b).



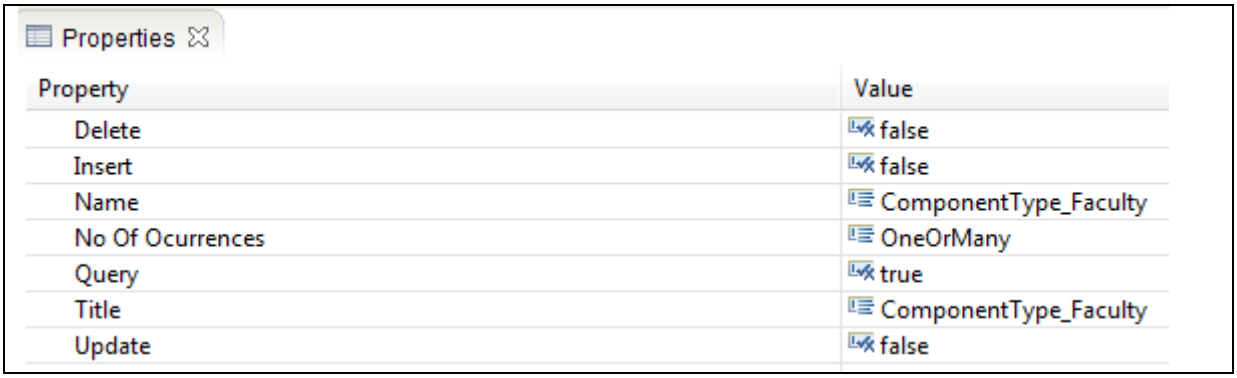
c) Model dela *UniversityDb* u skladu sa RMM



d) Model dela *UniversityDb* u skladu sa IISCaseMM

Slika 7.43. Deo *UniversityDb* relacione šeme baze podataka u Eclipse editoru

Na slici 7.43 može se videti da je od šema relacija *University* i *Faculty* izvornog modela nastao tip forme *FormType_FK_Faculty_University* sa korenskim tipom komponente *ComponentTypeRoot_University* i jednim potomkom *ComponentType_Faculty*. Između šema relacija *University* i *Faculty* postoji ograničenje referencijalnog integriteta *Faculty*[*FacId*] $\subseteq$ *University*[*UniId*], gde je šema relacije *Faculty* tipa *WR* jer primarni ključ šeme relacije *Faculty* sadrži *UniId*. Korenski tip komponente nastao je od šeme relacije *University* koja se nalazi sa desne strane. Tip komponente potomak nastao je od šeme relacije *Faculty*, koja se nalazi sa leve strane ograničenja referencijalnog integriteta.



Slika 7.44. Osobine tipa komponente *Faculty* u okviru tipa forme *FormType_FK_Faculty_University*

Takođe, pošto postoji i ograničenje inverznog referencijalnog integriteta između ove dve šeme relacije, broj pojavljivanja, *NoOfOcurrences*, tipa komponente potomka, *ComponentType_Faculty* inicijalizovan je na 1-N (*OneOrMany*). Osobine ovog tipa komponente prikazane su na slici 7.44.

Na listinzima 7.57 i 7.58 prikazani su primeri sa slike 7.43 a) i b) u XML XMI formatu, respektivno.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ConstraintRDM:Project xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
...
xmlns:URSConstraint="http://relacionimm/1.0/URSConstraint" Name="Project_Oracle10g">
  <URSs Name="URS"> ...  </URSs>
  <DBSchema Name="UniversityDb">
    <RelationSchemes Name="University">
      <PrimaryKey Name="PK_University" KeyAttr=
                                  "//@DBSchema/@RelationSchemes.0/@RSAttributes.0"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                  AttributeName="//@URSs/@URSAttributes.11"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniName"
                                  AttributeName="//@URSs/@URSAttributes.24"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="UniCity"
                                  AttributeName="//@URSs/@URSAttributes.7"/>
    </RelationSchemes>
    <RelationSchemes Name="Faculty">
      <PrimaryKey Name="PK_Faculty" KeyAttr=
            "//@DBSchema/@RelationSchemes.2/@RSAttributes.0//@DBSchema/@RelationSchemes.2/
                                                                  @RSAttributes.1"/>
      <EquivalentKey Name="UQ_Faculty" KeyAttr=
            "//@DBSchema/@RelationSchemes.2/@RSAttributes.0//@DBSchema/@RelationSchemes.2/
                        @RSAttributes.1 //@DBSchema/@RelationSchemes.2/@RSAttributes.2"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="UniId"
                                  AttributeName="//@URSs/@URSAttributes.11"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacId"
                                  AttributeName="//@URSs/@URSAttributes.6"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacShortName"
                                  AttributeName="//@URSs/@URSAttributes.13"/>
      <RSAttributes xsi:type="RelationConstraint:NotNullAttr" Name="FacName"
                                  AttributeName="//@URSs/@URSAttributes.19"/>
      <RSAttributes xsi:type="RelationConstraint:NullAttr" Name="Dean"
                                  AttributeName="//@URSs/@URSAttributes.3"/>
    </RelationSchemes>
    <MRConstraints xsi:type="INDConstraint:ReferentialIntegrityCon" Name=
            "FK_Faculty_University" RHS_Key="//@DBSchema/@RelationSchemes.0/@PrimaryKey"
                        LHS_Attr_RIC="//@DBSchema/@RelationSchemes.2/@RSAttributes.0">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT" RHS_RS_IND=
                                              "//@DBSchema/@RelationSchemes.0"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.2"/>
    </MRConstraints>
    <MRConstraints xsi:type="INDConstraint:InverseReferentialIntegrityCon" Name=
            "IRI_FK_Faculty_University" RHS_Attr_NKB_IND=
            "//@DBSchema/@RelationSchemes.2/@RSAttributes.0" LHS_Key=
            "//@DBSchema/@RelationSchemes.0/@PrimaryKey" RIC=
            "//@DBSchema/@MRConstraints.1">
      <RHS_RS DeleteAction="RESTRICT" UpdateAction="RESTRICT" RHS_RS_IND=
                                              "//@DBSchema/@RelationSchemes.2"/>
      <LHS_RS LHS_RS_IND="//@DBSchema/@RelationSchemes.0"/>
    </MRConstraints>
  </DBSchema>
</ConstraintRDM:Project>
```

Lisitng 7.57. Model dela *UniversityDb* u skladu sa RMM u XML XMI formatu

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project:ISApplicationModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
...
```

```
Name="IISCase_Project_Oracle10g">
  <FundamentalConcepts> ...  </FundamentalConcepts>
  <ApplicationSystems Name="AppSystem_UniversityDb">
    <OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name=
            "FormType_FK_Faculty_University" Title="FormType_FK_Faculty_University"
            Frequency="1" ResponseTime="1" ConsideredINDBSchDesign="true">
      <RootComponentType Name="ComponentTypeRoot_University" Title=
            "ComponentTypeRoot_University" Query="true">
        <ComponentTypeAttributes ComponentTypeAttributeName=
            "//@FundamentalConcepts.0/@FundamentalConcept.11" Title="UniId"
                                                                  Mandatory="true"/>
        <ComponentTypeAttributes ComponentTypeAttributeName=
            "//@FundamentalConcepts.0/@FundamentalConcept.24" Title="UniName"
                                                                  Mandatory="true"/>
        <ComponentTypeAttributes ComponentTypeAttributeName=
            "//@FundamentalConcepts.0/@FundamentalConcept.7" Title="UniCity"/>
        <ComponentTypeKeys Name="PK_University" ComponentTypeKeyAttributes=
            "//@ApplicationSystems.0/@OwnedFormTypes.8/@RootComponentType/
                                              @ComponentTypeAttributes.0" Global="true"/>
        <ComponentTypeChildren Name="ComponentType_Faculty" Title="ComponentType_Faculty"
                                              Query="true" NoOfOcurrences="OneOrMany">
          <ComponentTypeAttributes ComponentTypeAttributeName=
                        "//@FundamentalConcepts.0/@FundamentalConcept.6" Title="FacId"
                                                                  Mandatory="true"/>
          <ComponentTypeAttributes ComponentTypeAttributeName=
            "//@FundamentalConcepts.0/@FundamentalConcept.13" Title="FacShortName"
                                                                  Mandatory="true"/>
          <ComponentTypeAttributes ComponentTypeAttributeName=
            "//@FundamentalConcepts.0/@FundamentalConcept.19" Title="FacName"
                                                                  Mandatory="true"/>
          <ComponentTypeAttributes ComponentTypeAttributeName=
            "//@FundamentalConcepts.0/@FundamentalConcept.3" Title="Dean"/>
          <ComponentTypeKeys Name="UQ_Faculty" ComponentTypeKeyAttributes=
            "//@ApplicationSystems.0/@OwnedFormTypes.8/@RootComponentType/
            @ComponentTypeChildren.0/@ComponentTypeAttributes.0 //@ApplicationSystems.0/
            @OwnedFormTypes.8/@RootComponentType/@ComponentTypeChildren.0/
            @ComponentTypeAttributes.1" Global="true"/>
          <ComponentTypeKeys Name="PK_Faculty" ComponentTypeKeyAttributes=
            "//@ApplicationSystems.0/@OwnedFormTypes.8/@RootComponentType/
            @ComponentTypeChildren.0/@ComponentTypeAttributes.0" Global="true"/>
        </ComponentTypeChildren>
      </RootComponentType>
    </OwnedFormTypes>
    ...
  </ApplicationSystems>
</project:ISApplicationModel>
```

Lisitng 7.58. Model dela *UniversityDb* u skladu sa IISCaseMM u XML XMI formatu

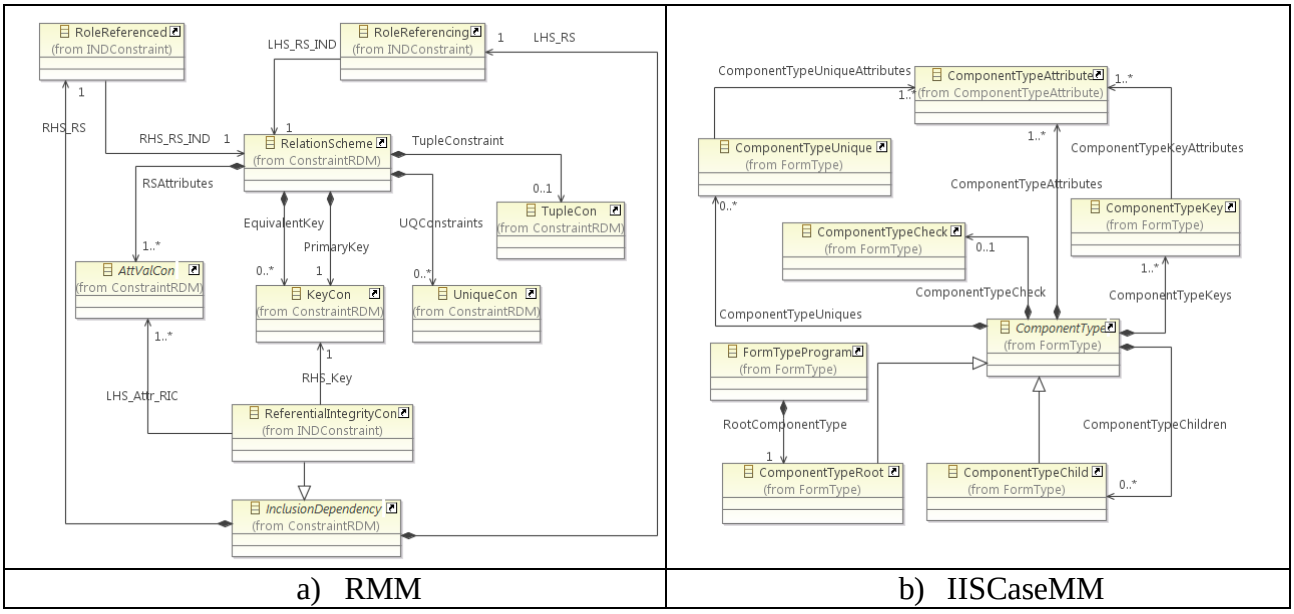
7.3.6. $\mathcal{F}$ _Tree^n tip forme

U ovoj tački biće prezentovani algoritmi za generisanje $\mathcal{F}$ *_Tree*^{n} tipova formi. Ova vrsta tipova formi nastaje na osnovu šeme relacije N_p za koju važi sledeće:

- ako je referencirana od strane dve ili više šema relacija tipa *WR*, ili
- ako je referencirana od strane barem jedne šeme relacije tipa *WR* koju referencira barem jedna šema relacija, takođe, tipa *WR*.

Šeme relacija koje su referencirane od strane samo jedne šeme relacije tipa *WR*, obuhvaćene su pravilom opisanim u prethodnoj tački, zbog čega će ovde biti izostavljene.

Pseudokod algoritma procedure za mapiranje šema relacija u $\mathcal{F}$ *_Tree*^{n} tip forme, *KreiranjeF_Tree*^{n}, prikazan je na slici 7.46. Na slici 7.45 prikazani su delovi meta-modela sa konceptima koji su obuhvaćeni ovim algoritmom za mapiranje.



a) RMM | b) IISCaseMM

Slika 7.45. Delovi meta-modela za mapiranje odgovarajućih koncepata u koncept tipa forme

Ulazni parametri procedure su skupovi šema relacija *S* i ograničenja referencijalnih integriteta *RIC*. Na osnovu skupa *RIC* izdvajaju se šeme relacija od kojih će nastati $\mathcal{F}$ *_Tree*^{n} tipovi formi pomoću procedure *TraženjeSvihRSzaKorenskiC_F*, čiji pseudokod je prikazan na slici 7.47.

U proceduri *TraženjeSvihRSzaKorenskiC_F* proverava se za svaku šemu relacije iz skupa *S*, na osnovu skupa ograničenja referencijalnih integriteta *RIC*, da li postoje ograničenja referencijalnog integriteta *ric* u kojima je šema relacije na levoj strani ograničenja tipa *WR*, a da je na desnoj strani posmatrana šema relacije. Ova provera vrši se pomoću procedure *ProveravanjeKandidataZaF*, čiji pseudokod je prikazan u tački 7.3.5 na slici 7.38.

Ukoliko za posmatranu šemu relacije postoje dve ili više šema relacija koja su slabe od nje (*no* > 1), ona postaje kandidat za korenski tip komponente $\mathcal{F}$ *_Tree*^{n} tipa forme, a slabe šeme relacija postaju kandidati za tipove komponenti potomaka.

Pošto u skup kandidata za korenske tipove komponenti tipova formi ulaze i šeme relacija koje su referencirane od strane barem jedne šeme relacije tipa *WR* koju referencira barem jedna šema relacija, takođe, tipa *WR*, proverava se da li za posmatranu šemu relacije ima takvih šema relacija. Ova provera vrši se pomoću procedure *ProveravanjePotomakImaPotomka*, koja je prikazana na slici 7.48. Pošto su pravilom opisanim u prethodnoj tački odeljka, prikazanim na slici 7.37, već kreirni tipovi formi sa jednim direktno podređenim potomkom, šeme relacija kandidati za potomka, koje nemaju svojih potomaka, ne uzimaju se u obzir.

Procedura:	*KreiranjeF_Tree*^{n}
Formalni parametri:	
Ulazni:	$S = \{N_i(R_i, O_i) \mid i = 1,.., n\}$ $O = K \cup UQ \cup CH$ $RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$
Izlazni:	$FT = \{ \mathcal{F}_i \mid i = 1,.., l\}$
Lokalne deklaracije: *rootRS* – izlazni parametar iz procedure *TraženjeSvihRSzaKorenskiC_F* (*rootRS* = $\{N_i(R_i, K_i) \mid i = 1,.., j\}$) *ct* – izlazni parametar iz procedure *KreiranjeKorenskogC_zaF_Tree*^{n} (*ct* – korenski tip komponente)	
Pseudokod: **POČETAK PROCESA** KreirajTipoveFormi_Tree^n **POZOVI** *TraženjeSvihRSzaKorenskiC_F*(*RIC, S, rootRS*) **RADI** kreiranjeF ($\forall N \in$ *rootRS*) **POSTAVI** *Name* ←'FormTypeN_' + *N* **POSTAVI** *Title* ←'FormTypeN_' + *N* **POSTAVI** *ConsideredINDBSchDesign* ←*true* **POSTAVI** *Frequency* ← 1 **POSTAVI** *ResponseTime* ← 1 **POZOVI** *KreiranjeKorenskogC_zaF_Tree*^{n}(*RIC, S, N, ct*) **POSTAVI** *RootComponentType* ← *ct* **KRAJ RADI** kreiranjeF **KRAJ PROCESA** KreirajTipoveFormi_Tree^n	

Slika 7.46. Algoritam procedure *KreiranjeF_Tree*^{n}

Ulazni parametri procedure *ProveravanjePotomakImaPotomka* su skup svih ograničenja referencijalnog integriteta i šema relacije za koju se proverava postojanje potomaka. Iz skupa ograničenja referencijalnog integriteta traži se da li postoji neko ograničenje *ric* u kojem je na desnoj strani šema relacije *N*, a na levoj šema relacije N_l koja je tipa *WR*. Ukoliko se nađe barem jedno, indikator *ima* biće postavljen na jedan što znači da *N* ima potomka. Proveru da li je N_l slaba šema relacije vrši procedura *ProveravanjeKandidataZaF* koja je objašnjena u tački 7.3.5 i prikazana na slici 7.38.

Nakon što se izdvoje šeme relacija od kojih će nastati $\mathcal{F}$ *_Tree*^{n} tipovi formi, za svaku će u procesu transformacije biti kreiran tip forme sa korenskim tipom komponente koji nastaje od te šeme relacije. Nazivu šeme relacije dodaje se prefiks *FormTupeN_*, čime se formira naziv tipa forme (*Name*), dok naslov tipa forme (*Title*) dobija istu vrednost kao i naziv. Objašnjenje za *ConsideredINDBSchDesign*, *Frequency* i *ResponseTime* je identično kao u prethodno opisanom mapiranju za $\mathcal{F}$ *_Basic* tipove formi.

Procedura:	*TraženjeSvihRSzaKorenskiC_F*
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $S = \{ N_i(R_i, O_i) \mid i = 1,.., n\}$
Izlazni:	*rootRS*

Lokalne deklaracije:
ind – izlazni parametar iz procedure *ProveravanjeKandidataZaF*
(*ind* = 1 – jeste kandidat za kreiranje *F*, *ind* = 0 – nije kandidat za kreiranje *F*)
ima – izlazni parametar iz procedure *ProveravanjeBrojaPotomaka*
(*ima* = 0 – potomak nema potomka, *ima* = 1 – potomak ima potomka)
no – broj direktno podređenih potomaka

Pseudokod:

```
POČETAK PROCESA TraženjeSvihRSzaKorenskiC_F
   RADI tražiC (∀N ∈ S)
      POSTAVI  no ← 0
      RADI proveru (∀ric ∈ RIC)
         AKO JE Nr = N TADA
            POZOVI ProveravanjeKandidataZaF(ric, Nl, ind)
            AKO JE  ind = 1 TADA
               POSTAVI  no ← no + 1
               POZOVI ProveravanjePotomakImaPotomka(RIC, Nl, ima)
            KRAJ AKO
         KRAJ AKO
      KRAJ RADI proveru
      AKO JE  ima  = 1 ∨ no > 1 TADA
         POSTAVI  rootRS ← rootRS ∪ N
      KRAJ AKO
   KRAJ RADI tražiC
KRAJ PROCESA TraženjeSvihRSzaKorenskiC_F
```

Slika 7.47. Algoritam procedure *TraženjeSvihRSzaKorenskiC_F*

Procedura:	*ProveravanjePotomakImaPotomka*
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $N(R, O)$
Izlazni:	*ima*
Lokalne deklaracije: *ind* - izlazni parametar iz procedure *ProveravanjeKandidataZaF* (*ind* = 1 – jeste kandidat za kreiranje *F*, *ind* = 0 – nije kandidat za kreiranje *F*)	
Pseudokod: **POČETAK PROCESA** ProveravanjePotomakImaPotomka **POSTAVI** *ima* ← 0 **RADI** traženje (∀*ric* ∈ *RIC* **DOK JE** *ima* = 0) ***AKO JE*** $N_r = N$ **TADA** **POZOVI** *ProveravanjeKandidataZaF*(*ric*, N_l, *ind*) ***AKO JE*** *ind* = 1 *ima* ← 1 ***KRAJ AKO*** ***KRAJ AKO*** **KRAJ RADI** traženje **KRAJ PROCESA** ProveravanjePotomakImaPotomka	

Slika 7.48. Algoritam procedure *ProveravanjePotomakImaPotomka*

Korenski tip komponente kreira se pomoću procedure *KreiranjeKorenskogC_zaF_Tree*^{n}, čiji pseudokod je prikazan na slici 7.49. Algoritam procedure za kreiranje korenskog tipa komponente za $\mathcal{F}$ *_Tree*^{n} tip forme veoma je sličan algoritmu za kreiranje korenskog tipa komponente $\mathcal{F}$ *_Tree*^{2} tipova formi, prikazanom na slici 7.39. Osnovna razlika je u tome što korenski tip komponente za $\mathcal{F}$ *_Tree*^{n} tip forme ima više tipova komponenti kao potomke, za razliku od $\mathcal{F}$ *_Tree*^{2} tipova formi, kod kojih korenski tip komponente ima samo jednog potomka.

Ulazni parametar procedure *KreiranjeKorenskogC_zaTree*^{2} je šema relacije *N* od koje nastaje korenski tip komponente, kao i skup svih šema relacija i skup svih ograničenja referencijalnog integriteta *RIC*, koji su potrebni za proceduru *KreiranjeC_zaF_Tree*^{n} koja se poziva iz ove procedure. Algoritam procedure *KreiranjeC_zaF_Tree*^{n} biće objašnjen u tekstu koji sledi ispod. Izlazni parametar procedure *KreiranjeKorenskogC_zaTree*^{n} je korenski tip komponente (*C*) sa svim potomcima.

Nazivu šeme relacije *N* dodaje se prefiks *ComponentTypeN_*, čime se formira naziv korenskog tipa komponente (*Name*), dok naslov tipa komponente (*Title*) dobija istu vrednost kao i naziv. Skup ključeva, skup ograničenja jedinstvenosti i ograničenje torke šeme relacije *N*, postaje skup ključeva, skup ograničenja jedinstvenosti i ograničenje torke tipa komponente. Dozvoljene operacije nad bazom podataka (*Delete, Insert, Update*), koje moraju biti pridružene svakom tipu komponente, inicijalizuju se na nedozvoljenu vrednost (*false*), osim operacije čitanja (*Query*) iz baze podataka koja je inicijalno dozvoljena (*true*).

Procedura:	*KreiranjeKorenskogC_zaF_Tree*^{n}
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $S = \{N_i(R_i, K_i) \mid i = 1,.., n\}$ $N(R, O), O = K \cup UQ \cup CH$
Izlazni:	$C(Q, O_C)$ $O_C = K_C \cup UQ_C \cup CH_C$
Lokalne deklaracije: *cN* – izlazni parametar iz procedure *TraženjeSvihDirektnihPotomaka* (*cN* – sve N_c koje su potomci od *N*)	
Pseudokod: **POČETAK PROCESA** KreiranjeKorenskogC **RADI** kreiranjeKorenC **POSTAVI** *Name* ←'ComponentTypeN_' + *N* **POSTAVI** *Title* ←' ComponentTypeN_' + *N* **POSTAVI** $Q \leftarrow R$ **POSTAVI** $K_C \leftarrow K$ **POSTAVI** $UQ_C \leftarrow UQ$ **POSTAVI** $CH_C \leftarrow CH$ **POSTAVI** *Query* ← true **POSTAVI** *Delete* ← false **POSTAVI** *Insert* ← false **POSTAVI** *Update* ← false **POZOVI** *TraženjeSvihDirektnihPotomaka*(*RIC*, *N*, *cN*) **RADI** kreiranjeC ($\forall N_c \in cN$) **POZOVI** *KreiranjeC_zaF_Tree*^{n}(*RIC*, *S*, *N*, N_c, *ct*) $cts \leftarrow cts \cup ct$ **KRAJ RADI** kreiranjeC **POSTAVI** *ComponentTypeChildren* ← *cts* **KRAJ RADI** kreiranjeKorenC **KRAJ PROCESA** KreiranjeKorenskogC	

Slika 7.49. Algoritam procedure *KreiranjeKorenskogC_zaF_Tree*^{n}

Potomci tipa komponente kreiraju se tako što se za sve pronađene šeme relacija, koje su slabe od šeme relacije od koje je nastao korenski tip komponente, poziva procedura *KreiranjeC_zaF_Tree*^{n} . Ova procedura služi za kreiranje tipa komponente potomka, čiji algoritam je prikazan na slici 7.51. Traženje šema relacija od kojih će putem transformacije nastati tipovi komponenti potomci vrši se pomoću procecedure *TraženjeSvihDirektnihPotomaka*, čiji pseudokod je prikazan na slici 7.50.

Ulazni parametri procedure *TraženjeSvihDirektnihPotomaka* su skup ograničenja referencijalnog integriteta *RIC* i šema relacije *N* za koju se traže identifikaciono zavisne šeme relacija, tj. slabe šeme relacija. Za svako ograničenje referencijalnog integriteta u kojem je na desnoj strani šema relacije *N*, proverava se da li je šema relacije N_l na levoj strani ograničenja tipa *WR*. Ukoliko jeste (*ind* = 1, izlazni parametar procedure *ProveravanjeKandidataZaF* koja je objašnjena u tački 7.3.5 na slici 7.38) šema relacije N_l ulazi u skup šema relacija *cN*, od kojih će nastati tipovi komponenti potomci tipa komponente nastalog od *N*.

Procedura:	*TraženjeSvihDirektnihPotomaka*
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $N(R, O)$
Izlazni:	*cN*
Lokalne deklaracije: *ind* - izlazni parametar iz procedure *ProveravanjeKandidataZaF* (*ind* = 1 – jeste kandidat za kreiranje *F*, *ind* = 0 – nije kandidat za kreiranje *F*)	
Pseudokod: **POČETAK PROCESA** TraženjeSvihDirektnihPotomaka **RADI** traženje ($\forall ric \in RIC$) ***AKO JE*** $N_r = N$ **TADA** **POZOVI** *ProveravanjeKandidataZaF*(*ric*, N_l, *ind*) ***AKO JE*** *ind* = 1 $cN \leftarrow cN \cup N_l$ ***KRAJ AKO*** ***KRAJ AKO*** **KRAJ RADI** traženje **KRAJ PROCESA** TraženjeSvihDirektnihPotomaka	

Slika 7.50. Algoritam procedure *TraženjeSvihDirektnihPotomaka*

U postupku transformacije, pri mapiranju šeme relacije na tip komponente koji će biti potomak, naziv i naslov tipa komponente formiraju se tako što se na naziv šeme relacije dodaje prefiks *ComponentTypeN_*.

Zbog uslova kojim je definisano da presek skupova atributa tipova komponenti na jednom tipu forme mora biti prazan skup, skup atributa tipa komponente formira se tako što se iz skupa atributa šeme relacije *N* od koje nastaje tip komponente potomak izbacuje skup atributa onog stranog ključa šeme relacije *N* koja referencira šemu relacije N_p, a od koje je nastao korenski tip komponente. Ovo je u algoritmu sa slike 7.51 predstavljeno izrazom

$$Q \leftarrow R/LHS$$

Da bi se došlo do skupa *LHS*, potrebno je pronaći ograničenje referencijalnog integriteta u kojem se na levoj strani nalazi šema relacije *N*, a na desnoj N_p. Ovaj postupak vrši se pomoću procedure *PronalaženjeRIC*, čiji pseudokod je prikazan na slici 7.52.

Ključevi, ograničenja jedinstvenosti, ograničenje torke kao i dozvoljene operacije nad bazom podataka mapirane su na isti način kao i kod korenskog tipa komponente.

Broj pojava tipa komponente (*NoOfOcurrences*) potomka u okviru pojave korenskog tipa komponente, kao i u prethodnom slučaju za $\mathcal{F}$ *_Tree*^{2} tip forme, inicijalzuje se na 1-N, ukoliko postoji ograničenje inverznog referencijalnog integriteta, a u suprotnom na 0-N.

Za razliku od slučaja $\mathcal{F}$ *_Tree*^{2} tipova formi, kod kojih tip komponente potomak nije imao svojih potomaka, za slučaj $\mathcal{F}$ *_Tree*^{n} tipova formi, tip komponente potomak može imati svoje potomke. Za pronalaženje svih šema relacija od kojih će nastati potomci tipa komponente, sada potomka, poziva se već objašnjena procedura *TraženjeSvihDirektnihPotomaka*, prikazana na slici 7.50. Za svaku nađenu šemu relacije poziva se ponovo procedura za kreiranje tipa komponente potomka *KreiranjeC_zaTree*^{n}.

Procedura:	*KreiranjeC_zaF_Tree*^{n}
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $S = \{N_i(R_i, K_i) \mid i = 1,.., n\}$ $N_p(R_p, O_p), O_p = K_p \cup UQ_p \cup CH_p$ $N(R, O), O = K \cup UQ \cup CH$
Izlazni:	$C(Q, O_C)$ $O_C = K_C \cup UQ_C \cup CH_C$
Lokalne deklaracije: *cN* – izlazni parametar iz procedure *TraženjeSvihDirektnihPotomaka* (*cN* – sve N_c koje su potomci od *N*)	

Pseudokod:

POČETAK PROCESA KreiranjeC_zaF_Tree^{n}
  RADI kreiranjeC
    POSTAVI *Name* ←'ComponentTypeN_' + *N*
    POSTAVI *Title* ←'ComponentTypeN_' + *N*
    POZOVI *PronalaženjeRIC*(*RIC*, N_p, *N*, *ric*)
    POSTAVI $Q \leftarrow R / LHS$
    POSTAVI $K_C \leftarrow K$
    POSTAVI $UQ_C \leftarrow UQ$
    POSTAVI $CH_C \leftarrow CH$
    POSTAVI *Query* ← true
    POSTAVI *Delete* ← false
    POSTAVI *Insert* ← false
    POSTAVI *Update* ← false
    AKO ∃ *IRIC* **TADA**
      POSTAVI *NoOfOcurrences* ← (1, M)
    INAČE
      POSTAVI *NoOfOcurrences* ← (0, M)
    KRAJ AKO
    POZOVI *TraženjeSvihDirektnihPotomaka*(*RIC*, *N*, *cN*)
    RADI rekurzija ($\forall N_c \in cN$)
      POZOVI *KreiranjeC_zaTree*^{n}(*RIC*, *S*, *N*, N_c, *ct*)
      $cts \leftarrow cts \cup ct$
    KRAJ RADI rekurzija
    POSTAVI *ComponentTypeChildren* ← *cts*
  KRAJ RADI kreiranjeC
KRAJ PROCESA KreiranjeC_zaF_Tree^{n}

Slika 7.51. Algoritam procedure *KreiranjeC_zaF_Tree*^{n}

Procedura:	*PronalaženjeRIC*
Formalni parametri:	
Ulazni:	$RIC = \{ ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}$ $N_r(R_r, O_r), N_l(R_l, O_l)$
Izlazni:	*ricR*

Pseudokod:

POČETAK PROCESA PronalaženjeRIC
  RADI traženjeRIC ($\forall ric: N_c[LHS_c] \subseteq N_p[RHS_p] \in RIC$)
    AKO JE $N_l = N_c \wedge N_r = N_p$ **TADA**
      POSTAVI $ricR \leftarrow ric$
    KRAJ AKO
  KRAJ RADI traženjeRIC
KRAJ PROCESA PronalaženjeRIC

Slika 7.52. Algoritam procedure *PronalaženjeRIC*

Definicija opisanog mapiranja implementirana je pomoću ATL *lazy* pravila *getFormType*, prikazanog na listingu 7.59. Primenom ovog pravila na instance šema relacija koje su zadovoljile uslov da budu korenski tipovi komponente na nekom tipu forme tipa $\mathcal{F}\_Tree^n$ (*p*), nastaje tip forme (*ft*) sa korenskim tipom komponente (*ctr*). Provera koje šeme relacija zadovoljavaju uslov vrši se pomoću ATL pomoćne metode prikazane na listingu 7.34.

Atributi i ograničenja korenskog tipa komponente kreiraju se na isti način kao u ATL pravilima za generisanje $\mathcal{F}\_Basic$ i $\mathcal{F}\_Tree^2$ tipova formi, pozivanjem ATL *lazy* pravila prikazanih na listinzima od 7.43 do 7.48.

Za razliku od pravila za generisanje $\mathcal{F}\_Tree^2$ tipova formi, kod kojih je kreiran samo jedan tip komponente potomak, u ovom pravilu poziva se rekurzivno ATL *lazy* pravilo *ChildComponentType* za kreiranje svih potomaka kreiranog korenskog tipa komponente, prikazano na listingu 7.61. Potomci nekog tipa komponente (*children*) traže se pomođu ATL pomoćne metode *Children*, prikazane na listingu 7.60. Ova pomoćna metoda ima samo jedan ulazni parametar, šemu relacije za koju se traže svi direktni potomci. Osim rekurzivnog pozivanja pravila za kreiranje tipova komponenti potomaka, sve ostalo je isto kao kod pravila za kreiranje tipa komponente potomka opisanog u prethodnoj tački odeljka na slici 7.52.

```
lazy rule getFormType{
from
   p: RM!RelationScheme
using {
   children: Sequence(RM!RelationScheme)= thisModule.Children(p);
}
to
   ft: IISCase!FormTypeProgram(
      Name <- 'FormTypeN_' + p.Name,
      Title <-'FormTypeN_' + p.Name,
      ConsideredINDBSchDesign <- true,
      Frequency <- 1,
      ResponseTime <- 1,
      RootComponentType <- ctr
   ),
   ctr: IISCase!ComponentTypeRoot(
      Name <- 'ComponentTypeRootN_' + p.Name,
      Title <- 'ComponentTypeRootN_' + p.Name,
      ComponentTypeAttributes <-p.RSAttributes -> collect(e|
```

```
         if (e.oclIsTypeOf(RM!NotNullAttr)) then
            thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
            thisModule.RSAttributes2NullCompTypeAttribute(e) endif),
      ComponentTypeKeys <- (p.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(p.PrimaryKey)),
      ComponentTypeUniques <- p.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e)),
      ComponentTypeCheck <- if p.TupleConstraint.oclIsUndefined() then OclUndefined else
         thisModule.TupleConstraints2ComponentTypeCheckCon(p.TupleConstraint) endif,
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false,
      ComponentTypeChildren <- children->collect(c | thisModule.ChildComponentType(c,p))
   )
   do{
      thisModule.attributes <- Sequence{};
   }
}
```

Listing 7.59. ATL *lazy* pravilo za mapiranje koncepta **RelationScheme** u koncepte **FormTypeProgram** i **ComponentTypeRoot**

```
helper def: Children(rs:RM!RelationScheme): Sequence(RM!RelationScheme) =
            let allRIC:Sequence(RM!ReferentialIntegrityCon)=
RM!ReferentialIntegrityCon.allInstances() in
            allRIC->iterate(ric; rez:Sequence(RM!RelationScheme) = Sequence{}|
            if    ric.RHS_RS.RHS_RS_IND=rs    and    thisModule.isRSforFormType(ric)    then
rez.append(ric.LHS_RS.LHS_RS_IND) else rez endif);
```

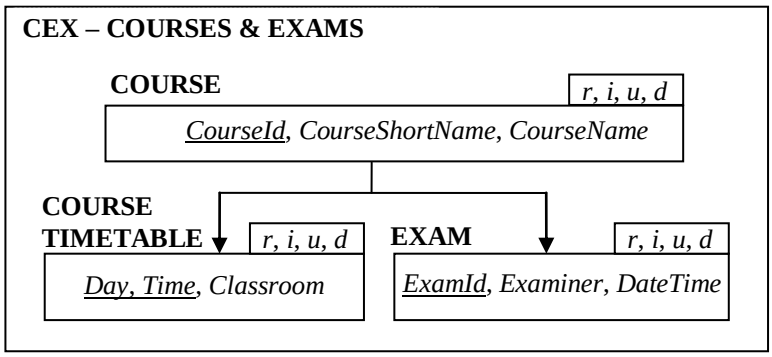
Listing 7.60. ATL *helper* za traženje direktnih potomaka šeme relacije

```
lazy rule ChildComponentType{
from
   rs:RM!RelationScheme,
   p:RM!RelationScheme
using{
   children: Sequence(RM!RelationScheme)= thisModule.Children(rs);
}
To
   ct: IISCase!ComponentTypeChild (
      Name <- 'Child_ComponentTypeN_' + rs.Name,
      Title <- 'Child_ComponentTypeN_' + rs.Name,
      ComponentTypeAttributes <- (rs.RSAttributes.asSet()-
         thisModule.parentKeyAttributes.asSet()) ->collect(e|
         if (e.oclIsTypeOf(RM!NotNullAttr)) then
            thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
            thisModule.RSAttributes2NullCompTypeAttribute(e) endif),
      ComponentTypeAttributes <- if thisModule.RS_AllKeys(rs) then
         thisModule.AttributesOfRSs(rs, p)->collect(e|
         thisModule.RSAttributes2CompTypeAttribute(e)) else rs.RSAttributes ->
         collect(e| if (e.oclIsTypeOf(RM!NotNullAttr)) then
         thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
         thisModule.RSAttributes2NullCompTypeAttribute(e) endif) endif,
      ComponentTypeKeys <- (rs.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(rs.PrimaryKey)),
      ComponentTypeUniques <- rs.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e)),
      ComponentTypeCheck <- if rs.TupleConstraint.oclIsUndefined() then
```

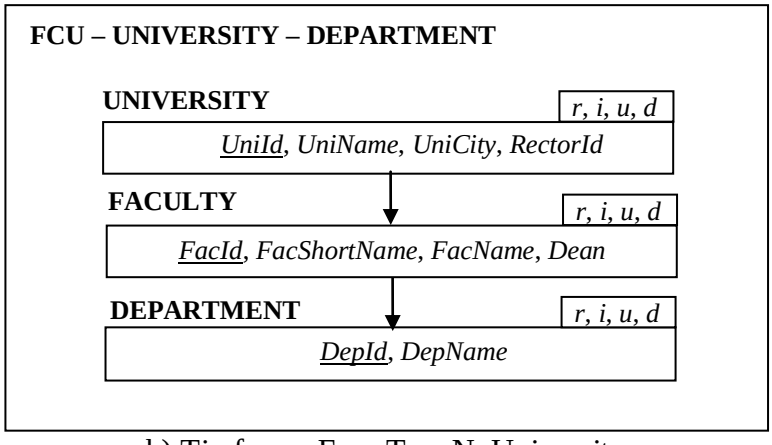
```
            OclUndefined else
            thisModule.TupleConstraints2ComponentTypeCheckCon(rs.TupleConstraint)
         endif,
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false,
      NoOfOcurrences <- #NoneOrMany,
      ComponentTypeChildren <- children->collect(c |
         thisModule.ChildComponentType(c,rs))
   )
   do{
      thisModule.attributes <- Sequence{};
   }
}
```

Listing 7.61. Rekurzivno ATL *lazy* pravilo za mapiranje koncepta **RelationScheme** u koncept **ComponentTypeChild**

Primer 7.10. Za isti skup šema relacija i ograničenja, dat u primerima od 7.1 do 7.9, primenom prethodno opisanih pravila, nakon transformacije kreiraju se i dva $\mathcal{F}\_Tree^n$ tipa forme: *FormTypeN_Course* i *FormTypeN_University*. Struktura ova dva tipa forme, kao i njihova prezentacija u Eclipse editoru, prikazani su na slikama 7.53 i 7.54.



a) Tip forme FormTypeN_Course



b) Tip forme FormTypeN_University

Slika 7.53. Generalizacija ekranskih formi

Na osnovu ograničenja referencijalnog integriteta koja postoje između šema relacija *Exam, CourseTimetable* i *Course*:

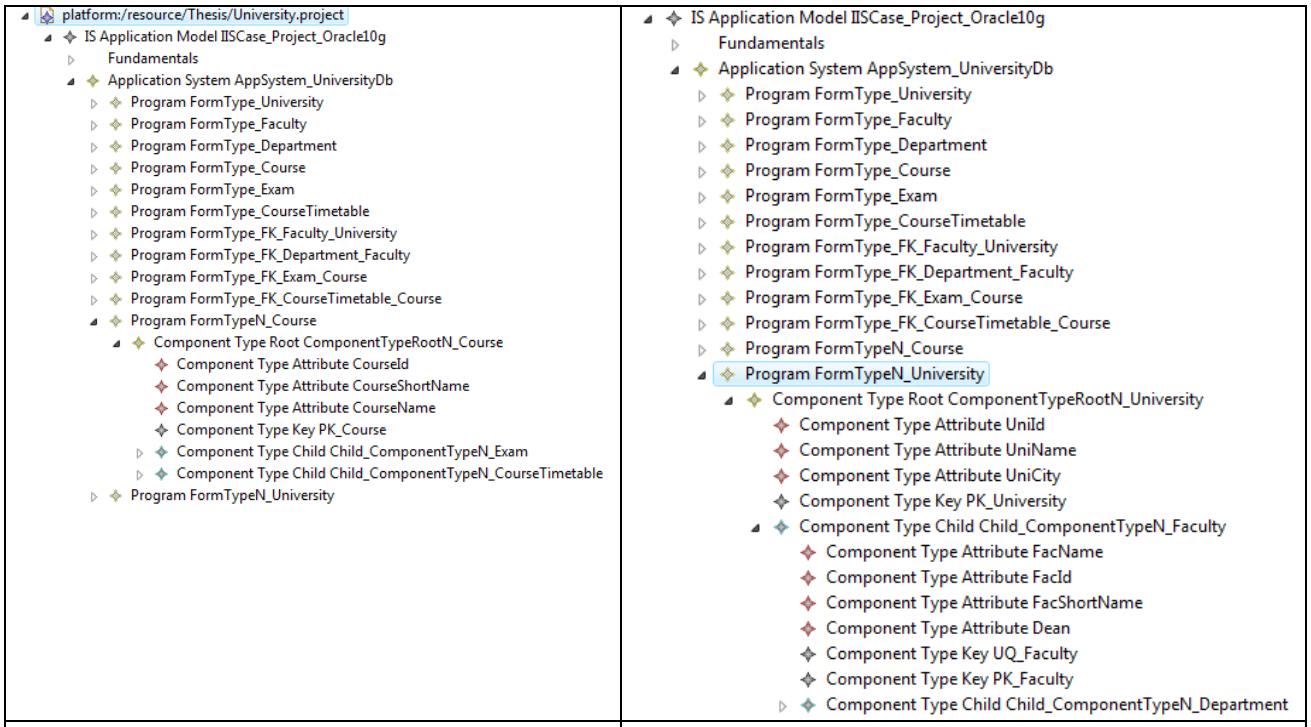
- *FK_Exam_Course*: *Exam*[*CourseId*] ⊆ *Course*[*CourseId*],
- *FK_CourseTimetable _Course*: *CourseTimetable*[*CourseId*] ⊆ *Course*[*CourseId*],

šema relacije *Course* postaje element izvornog modela na koji se primenjuje pravilo za generisanje $\mathcal{F}$ *_Tree*^{n} tipova formi. Od šeme relacije *Course* nastaje korenski tip komponente, dok od šema relacija *Exam* i *CourseTimetable* nastaju tipovi komponenti potomci (slike 7.54 a) i 7.55 a)).

Takođe, na osnovu ograničenja referencijalnog integriteta:

- *FK_ Faculty_University: Faculty*[*FacId*] ⊆ *University*[*UniId*] i
- *FK_Department_Faculty*: *Department*[*UniId, FacId*] ⊆ *Faculty*[*UniId, FacId*],

od šeme relacije *University*, nakon procesa transformacije, nastaje $\mathcal{F}$ *_Tree*^{n} tip forme sa jednim tipom komponente kao direktnim potomkom, nastalim od šeme relacije *Faculty*, i jednim tipom komponente kao posrednim potomkom nastalim od šeme relacije *Department*, koja je direktni potomak od *Faculty* (slike 7.54 b) i 7.55 b)).



a) Tip forme FormTypeN_Course | b) Tip forme FormTypeN_University

Slika 7.54. Deo *UniversityDb* konceptualne šeme baze podataka u Eclipse editoru

Na listingu 7.62 prikazan je primeri sa slike 7.54 b) u XML XMI formatu.

```
...
<OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name="FormTypeN_University"
Title="FormTypeN_University" Frequency="1" ResponseTime="1"
ConsideredINDBSchDesign="true">
  <RootComponentType Name="ComponentTypeRootN_University" Title=
            "ComponentTypeRootN_University" Query="true">
    <ComponentTypeAttributes ComponentTypeAttributeName=
      "//@FundamentalConcepts.0/@FundamentalConcept.11" Title="UniId" Mandatory="true"/>
    <ComponentTypeAttributes ComponentTypeAttributeName=
      "//@FundamentalConcepts.0/@FundamentalConcept.24" Title="UniName" Mandatory="true"/>
    <ComponentTypeAttributes ComponentTypeAttributeName=
```

```
      "//@FundamentalConcepts.0/@FundamentalConcept.7" Title="UniCity"/>
    <ComponentTypeKeys Name="PK_University" ComponentTypeKeyAttributes=
      "//@ApplicationSystems.0/@OwnedFormTypes.14/@RootComponentType/
       @ComponentTypeAttributes.0" Global="true"/>
    <ComponentTypeChildren Name="Child_ComponentTypeN_Faculty" Title=
      "Child_ComponentTypeN_Faculty" Query="true" NoOfOcurrences="NoneOrMany">
      <ComponentTypeAttributes ComponentTypeAttributeName=
        "//@FundamentalConcepts.0/@FundamentalConcept.3" Title="Dean"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
        "//@FundamentalConcepts.0/@FundamentalConcept.11" Title="UniId" Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
       "//@FundamentalConcepts.0/@FundamentalConcept.19" Title="FacName"
                                                                  Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
       "//@FundamentalConcepts.0/@FundamentalConcept.13" Title="FacShortName"
                                                                  Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
       "//@FundamentalConcepts.0/@FundamentalConcept.6" Title="FacId" Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
       "//@FundamentalConcepts.0/@FundamentalConcept.11" Title="UniId" Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
       "//@FundamentalConcepts.0/@FundamentalConcept.6" Title="FacId" Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
       "//@FundamentalConcepts.0/@FundamentalConcept.13" Title="FacShortName"
                                                                  Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
        "//@FundamentalConcepts.0/@FundamentalConcept.19" Title="FacName"
                                                                  Mandatory="true"/>
      <ComponentTypeAttributes ComponentTypeAttributeName=
      "//@FundamentalConcepts.0/@FundamentalConcept.3" Title="Dean"/>
      <ComponentTypeKeys Name="UQ_Faculty" ComponentTypeKeyAttributes=
        "//@ApplicationSystems.0/@OwnedFormTypes.14/@RootComponentType/
         @ComponentTypeChildren.0/@ComponentTypeAttributes.5
        //@ApplicationSystems.0/@OwnedFormTypes.14/@RootComponentType/
         @ComponentTypeChildren.0/@ComponentTypeAttributes.1
        //@ApplicationSystems.0/@OwnedFormTypes.14/@RootComponentType/
         @ComponentTypeChildren.0/@ComponentTypeAttributes.6
...
      <ComponentTypeKeys Name="PK_Faculty" ComponentTypeKeyAttributes=
        "//@ApplicationSystems.0/@OwnedFormTypes.14/@RootComponentType/
...
      <ComponentTypeChildren Name="Child_ComponentTypeN_Department" Title=
         "Child_ComponentTypeN_Department" Query="true" NoOfOcurrences="NoneOrMany">
        <ComponentTypeAttributes ComponentTypeAttributeName=
          "//@FundamentalConcepts.0/@FundamentalConcept.18" Title="Director"
                                                                  Mandatory="true"/>
        <ComponentTypeAttributes ComponentTypeAttributeName=
          "//@FundamentalConcepts.0/@FundamentalConcept.6" Title="FacId"
                                                                  Mandatory="true"/>
        <ComponentTypeAttributes ComponentTypeAttributeName=
         .../>
        <ComponentTypeKeys Name="PK_Department" ComponentTypeKeyAttributes=
         "//@ApplicationSystems.0/@OwnedFormTypes.14/@RootComponentType/ ... />
          </ComponentTypeChildren>
        </ComponentTypeChildren>
      </RootComponentType>
    </OwnedFormTypes>
```

Listing 7.62. Model dela *UniversityDb* u skladu sa IISCaseMM u XML XMI formatu

7.4. Definicije mapiranja elemenata XML šeme u koncepte RSUBP meta-modela

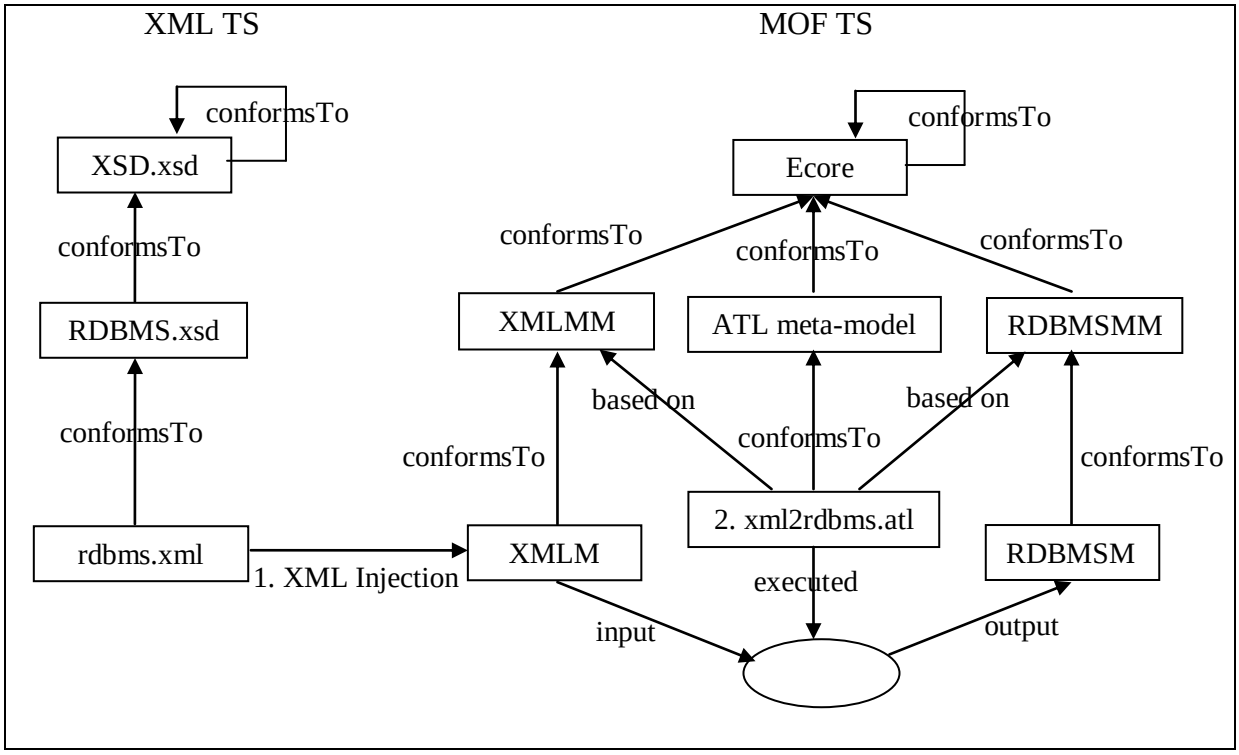
Instanca meta-modela relacione šeme baze podataka dobija se ekstrakcijom meta-podataka iz rečnika podataka nekog RSUBP-a, koji se pomoću modula *XMLTransformer* serijalizuju u XML specifikaciju. XML specifikacija je u skladu sa XML šemom RDBMS.xsd, koja je kompletno prikazana u prilogu A.1.

Kako ova XML specifikacija, koja predstavlja ulaznu instancu transformacije koja se izvršava pomoću modula *M2M Transformer*, pripada XML tehnološkom prostoru (XML TS) neophodno je dobiti odgovarajući model relacione šeme baze podataka koji će pripadati MDA MOF tehnološkom prostoru (MOF TS).

U cilju prevazilaženja problema različitih tehnoloških prostora, u prvom koraku transformacije konvertuje se XML specifikacija sa meta-podacima u model koji je u skladu sa MOF-zasnovanim XML meta-modelom. Konverzija iz XML TS u MOF TS vrši se XML *injector*-om, koji je ugrađen u ATL SDK.

U drugom koraku transformacije, dobijeni XML model, koji je u skladu sa meta-modelom prezentovanim u odeljku 6.1, transformiše se u model RDBMSM, koji je u skladu sa RSUBP meta-modelom, predstavljenim u odeljku 6.2.

Scenario prva dva koraka transformacije prikazan je na slici 7.55.



Slika 7.55. Scenario prva dva koraka transformacije

Pomoću xml2rdbms.atl transformacije elementi iz XMLM modela transformišu se u elemente RDBMSM modela. U tabeli 7.3 prikazani su koncepti koji učestvuju u mapiranjima između elemenata XML šeme (RDBMS.xsd), XML meta-modela (XMLMM) i RSUBP meta-modela (RDBMSMM).

Tabela 7.3. Koncepti za mapiranje XML šeme u RSUBP meta-model

XML šema	XML meta-model	RSUBP meta-model
`xsd:element name="RSUBP"`	`Root`	RDBMS
`xsd:element name="Database"`	`Element name = 'Database'`	Database
`xsd:element name="SystemDataType"`	`Element name= 'SystemDataType'`	SystemDataType
`xsd:element name="UserDefinedDataType"`	`Element name='UserDefinedDataType'`	UserDefinedDataType
`xsd:element name="Table"`	`Element name='Table'`	Table
`xsd:element name="Column"`	`Element name='Column'`	Column
`xsd:element name="PrimaryKeyCon"`	`Element name= 'PrimaryKeyCon'`	PrimaryKeyCon
`xsd:element name="UniqueCon"`	`Element name='UniqueCon'`	UniqueCon
`xsd:element name="CheckCon"`	`Element name='CheckCon'`	CheckCon
`xsd:element name="ForeignKey"`	`Element name='ForeignKey'`	ForeignKey

Mapiranja između elemenata XML meta-modela i elemenata RSUBP meta-modela su definisana pomoću *matched* pravila u ATL jeziku, u kojima je korišćeno više pomoćnih metoda.

Na listingu 7.63 prikazana je definicija mapiranja korenskog elementa XML meta-modela u korenski element RSUBP meta-modela, pomoću ATL pravila *Root2RDBMS*. Korenski element sadrži skup sistemskih tipova podataka (*DataTypes*) i baza podataka (*Databases*). Iz ovog pravila pozivaju se i dve pomoćne metode *getValue* i *testNodeName* za traženje odgovarajućeg elementa i dobijanje tekstualnog sadržaja elementa, prikazane na listinzima 7.64 i 7.65, respektivno.

```
rule Root2RDBMS{
from
   r : xml!Root
to
   out:rdbms!RDBMS (
      Name <- r.getValue('Name'),
      DataTypes <- r.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('SystemDataType')),
      Databases <- r.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('Database'))
   )
}
```

Listing 7.63. ATL pravilo za mapiranje korenskog elementa na koncept RDBMS

```
helper context xml!Element def: getValue(name : String) : String =
if self.testElement(name) then
 self.children->select(c | c.oclIsKindOf(xml!Element) and c.name=thisModule.prefix + name)
            ->first().children ->select(c | c.oclIsKindOf(xml!Text))->first().value
else '' endif;
```

Listing 7.64. ATL *helepr* za dobijanje tekstualnog sadržaja elementa

ATL pravila za mapiranje koncepata sistemskih tipova podataka i baze podataka prikazana su na listinzima 7.67 i 7.68, respektivno. Ova pravila, takođe, pozivaju prethodno pomenute pomoćne metode, kao i pomoćnu metodu *getIntValue*, za dobijanje sadržaja elementa čiji tip podatka je numerički, prikazanu na listingu 7.66. Pravila prikazana na listinzima 7.67 i 7.68 koriste i pomoćni atribut *prefix*, kojim se izdvaja prefiks ispred elementa korišćen u XML šemi. Atribut *prefix* prikazan je na listingu 7.69.

```
helper context xml!Node def: testNodeName(name : String) : Boolean =
if self.name = thisModule.prefix + name then true
else false endif;
```

Listing 7.65. ATL *helepr* za traženje odgovarajućeg elementa

```
helper context xml!Element def: testElement(name : String) : Boolean =
not (self.children ->select(d | d.oclIsKindOf(xml!Element) and d.name = thisModule.prefix
+ name)->first().oclIsUndefined());

helper context xml!Element def: getIntValue(name : String) : Integer =
if self.testElement(name) then
let str : String = self.children->select(c | c.oclIsKindOf(xml!Element) and
c.name=thisModule.prefix + name)->first().children ->select(c | c.oclIsKindOf(xml!Text))
->first().value in
  if not (str = '') then str.toInteger()
  else 0 endif
else 0 endif;
```

Listing 7.66. ATL *helepr*-i za dobijanje sadržaja elementa tipa broja

```
rule SystemDataType{
from
   s: xml!Element (s.name = thisModule.prefix + 'SystemDataType')
to
   out:rdbms!SystemDataType(
      Name <- s.getValue('Name'),
      PredefinedDecPlaces <- s.getIntValue('PredefinedDecPlaces'),
      PredefinedLength <- s.getIntValue('PredefinedLength')
   )
}
```

Listing 7.67. ATL pravilo za mapiranje elementa *SystemDataType* na koncept *SystemDataType*

```
rule Database{
from
   db: xml!Element (db.name = thisModule.prefix + 'Database')
to
   out: rdbms!Database(
      Name <- db.getValue('Name'),
      Tables <- db.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('Table')),
      UserDefinedDataTypes <- db.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('UserDefinedDataType'))
   )}
```

Listing 7.68. ATL pravilo za mapiranje elementa *Database* na koncept *Database*

```
helper def: prefix : String =
let allRootAttributes : Sequence(xml!Attribute) =
xml!Root.allInstances().first().children -> select(attr | attr.oclIsTypeOf(xml!Attribute))
in let prefixName : String =
allRootAttributes -> select(attr | attr.value =
   'http://www.uns.ac.rs/dbXMLSchema').first().name in
    prefixName.substring(prefixName.lastIndexOf(':') + 2, prefixName.size()) + ':';
```

Listing 7.69. ATL *helepr* za izdvajanje prefiksa ispred elementa

Baza podataka sadrži skup tabela i korisnički definisanih tipova podataka. ATL pravila za mapiranje ovih koncepata prikazana su na listinzima 7.70 i 7.71, respektivno.

```
rule Table{
from
   tb: xml!Element (tb.name = thisModule.prefix + 'Table')
to
   out:rdbms!Table(
      Name <- tb.getValue('Name'),
      Columns <- tb.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('Column')),
            TablePK <- tb.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('PrimaryKeyCon'))->first(),
            TableUQ <- tb.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('UniqueCon')),
            TableCHs <- tb.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('CheckCon')),
            TableFKs <- tb.children->select(d | d.oclIsKindOf(xml!Element) and
            d.testNodeName('ForeignKey'))
   )
}
```

Listing 7.70. ATL pravilo za mapiranje elementa *Table* na koncept *Table*

```
rule UserDefinedDataType{
from
   dt: xml!Element (dt.name = thisModule.prefix + 'UserDefinedDataType')
to
   out: rdbms!UserDefinedDataType(
      Name <- dt.getValue('Name'),
      Precision <- dt.getIntValue('Precision'),
      Length <- dt.getIntValue('Length'),
      DefaultValue <- dt.getValue('DefaultValue'),
      DataType <- xml!Element->allInstances()->select(d | d.testNodeName('SystemDataType')
            and d.getValue('Name') = dt.getValue('DataTypeName'))->first()
   )
}
```

Listing 7.71. ATL pravilo za mapiranje elementa *UserDefinedDataType* na koncept *UserDefinedDataType*

Tabela baze podataka sadrži kolone, ograničenje primarnog ključa, skup ograničenja jedinstvenosti, skup *check* ograničenja i skup ograničenja stranog ključa. Definicije pravila za mapiranje ovih koncepata pomoću ATL-a, prikazana su na listinzima 7.72, 7.74, 7.76, 7.77 i 7.78, respektivno. U ovim pravilima koriste se i pomoćne metode prikazane na listinzima 7.73, 7.75 i 7.79 koje se pozivaju radi dobijanja sadržaja elementa logičkog tipa, pronalaženja atributa određenog ograničenja i dobijanja vrednosti elemenata potomaka, respektivno.

```
rule Column {
from
   col: xml!Element (col.name = thisModule.prefix + 'Column')
to
   out: rdbms!Column(
      Name <- col.getValue('Name'),
      ColumnDataType <- xml!Element->allInstances()->select(d |
            d.testNodeName('SystemDataType') and
            d.getValue('Name') = col.getValue('DataType')) -> first(),
      Default <- col.getValue('Default'),
      Length <- col.getIntValue('Length'),
      Precision <- col.getIntValue('Precision'),
      Nullable <- col.getBooleanValue('Nullable')
   )
}
```

Listing 7.72. ATL pravilo za mapiranje elementa *Column* na koncept *Column*

```
helper context xml!Element def: getBooleanValue(name : String) : Boolean =
if self.testElement(name) then
let str : String = self.children->select(c | c.oclIsKindOf(xml!Element) and
c.name=thisModule.prefix + name)->first().children ->select(c | c.oclIsKindOf(xml!Text))
->first().value in if not (str = '') then
      if str = 'true' then true else false endif
   else false endif
else false endif;
```

Listing 7.73. ATL *helepr* za dobijanje sadržaja elementa logičkog tipa

```
rule PrimaryKeyCon{
from
   pk: xml!Element (pk.name = thisModule.prefix + 'PrimaryKeyCon')
to
   out: rdbms!PrimaryKeyCon(
      Name <- pk.getValue('Name'),
      Deferrable <- pk.getValue('Deferrable')
      Deferred <- pk.getValue('Deferred'),
      PKandUQColumns <- thisModule.allConsColumns(pk, pk.parent.getValue('Name'),
                                                                  'PKColumn')
   )
}
```

Listing 7.74. ATL pravilo za mapiranje elementa *PrimaryKeyCon* na koncept *PrimaryKeyCon*

```
helper def: allConsColumns (con : xml!Element, tableName : String, ConsColName : String) :
Sequence(xml!Element) =
let tables : Sequence(xml!Element) = xml!Element->allInstances()->select(tbl |
tbl.testNodeName('Table')) in
   xml!Element->allInstances()->select(col | col.testNodeName('Column') and
   col.getValue('Name') = con.getValue(ConsColName) and tables -> exists(tbl |
   tbl.getValue('Name') = tableName and tbl.children -> includes(col)));
```

Listing 7.75. ATL *helepr* za pronalaženje atributa određenog ograničenja

```
rule UniqueCon{
from
uq:xml!Element (uq.name = thisModule.prefix + 'UniqueCon')
to
   out:rdbms!UniqueCon(
      Name <- uq.getValue('Name'),
      Deferrable <- uq.getValue('Deferrable'),
      Deferred <- uq.getValue('Deferred'),
      PKandUQColumns <- thisModule.allConsColumns(uq, uq.parent.getValue('Name'),
                                                                    'UQColumn')
   )}
```

Listing 7.76. ATL pravilo za mapiranje elementa *UniqueCon* na koncept *UniqueCon*

```
rule CheckCon{
from
   ck:xml!Element(ck.name = thisModule.prefix + 'CheckCon')
to
   out:rdbms!CheckCon(
      Name <- ck.getValue('Name'),
      Deferrable <- ck.getValue('Deferrable'),
      Deferred <- ck.getValue('Deferred'),
      CheckCondition <- ck.getValue('CheckCondition')
   )
}
```

Listing 7.77. ATL pravilo za mapiranje elementa *CheckCon* na koncept *CheckCon*

```
rule ForeignKey{
from
   fk:xml!Element(fk.name = thisModule.prefix + 'ForeignKey')
to
   out:rdbms!ForeignKey(
      Name <- fk.getValue('Name'),
      RefersTo <- xml!Element->allInstances()->select(d | d.testNodeName('Table') and
        d.getValue('Name') = fk.getValue('RefersTo'))->first(),
      RHS_Key <- xml!Element->allInstances()->select(d | d.testNodeName('Table') and
        d.getValueChild('PrimaryKeyCon','Name') = fk.getValue('RHS_Key'))->
        first().getChild('PrimaryKeyCon'),
      LHS_Attr <- fk.getChildren('LHS_Column')->collect(lhs | xml!Element->allInstances()
        ->select(table | table.testNodeName('Table') and table.getValue('Name') =
        fk.parent.getValue('Name'))->first().getChildren('Column')
        ->select(col | col.getValue('Name') = lhs.children.first().value))->flatten(),
      Deferrable <- fk.getValue('Deferrable'),
      Deferred <- fk.getValue('Deferred'),
      Match <- fk.getValue('Match'),
      UpdateActionRHS <- fk.getValue('UpdateAction'),
      DeleteActionRHS <- fk.getValue('DeleteAction'),
      InverseReferentialIntegrityCon <-
                                    fk.getBooleanValue('InverseReferentialIntegrityCon')
   )
}
```

Listing 7.78. ATL pravilo za mapiranje elementa *ForeignKey* na koncept *ForeignKey*

```
helper context xml!Element def: getValueChild(name : String, name1 : String) : String =
if self.testElement(name) then
self.children->select(c | c.oclIsKindOf(xml!Element) and c.name=thisModule.prefix + name)
->first().children->select(c | c.oclIsKindOf(xml!Element) and c.name=thisModule.prefix +
```

```
name1)->first().children->select(c | c.oclIsKindOf(xml!Text))->first().value
else '' endif;

helper context xml!Element def: getChildren(name : String) : Sequence(xml!Element) =
if self.testElement(name) then
self.children->select(c | c.oclIsKindOf(xml!Element) and c.name=thisModule.prefix + name)
else '' endif;

helper context xml!Element def: getChild(name : String) : xml!Element =
if self.testElement(name) then
self.children->select(c | c.oclIsKindOf(xml!Element) and c.name=thisModule.prefix + name)
->first() else '' endif;
```

Listing 7.79. ATL *helepr*-i za dobijanje vrednosti elemenata potomka

U slučaju primera XML specifikacije prikazane na listingu 5.2, nakon konverzije iz XML tehnološkog prostora XML *injector*-om, dobija se XMI specifikacija koja predstavlja XML model u MOF tehnološkom prostoru. Polazna XML specifikacija napravljena je pomoću modula *XML Transformer*, na osnovu XML šeme RDBMS.xsd, date u prilogu A.1. XMI specifikacija, nakon obavljene konverzije, prikazana je na listingu 7.80. Nakon primene prethodno opisanih mapiranja koja čine xml2rdbms.atl transformaciju, dobijeni XML model, koji je u skladu sa meta-modelom prezentovanim u odeljku 6.1, transformiše se u model RDBMSM, koji je u skladu sa RSUBP meta-modelom, predstavljenim u odeljku 6.2. Rezultat ove transformacije u XML XMI formatu, za ulazni model čiji deo je prikazan na listingu 7.80, prikazan je na listingu 7.81.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Root xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="XML" name="db:RSUBP">
  <children xsi:type="Attribute" name="xmlns:db"
            value="http://www.uns.ac.rs/dbXMLSchema"/>
  <children xsi:type="Attribute" name="xmlns:xsi" value="http://www.w3.org/2001/
            XMLSchema-instance"/>
  <children xsi:type="Attribute" name="xsi:schemaLocation"
            value="http://www.uns.ac.rs/dbXMLSchema dbXMLSchema.xsd "/>
  <children xsi:type="Element" name="db:Name">
    <children xsi:type="Text" name="#text" value="Oracle10g"/>
  </children>
  <children xsi:type="Element" name="db:Database">
    <children xsi:type="Element" name="db:Name">
      <children xsi:type="Text" name="#text" value="UniversityDB"/>
    </children>
    <children xsi:type="Element" name="db:Table">
      <children xsi:type="Element" name="db:Name">
        <children xsi:type="Text" name="#text" value="University"/>
      </children>
      <children xsi:type="Element" name="db:Column">
        <children xsi:type="Element" name="db:Name">
          <children xsi:type="Text" name="#text" value="UniId"/>
        </children>
        <children xsi:type="Element" name="db:Nullable">
          <children xsi:type="Text" name="#text" value="false"/>
        </children>
        <children xsi:type="Element" name="db:DataType">
          <children xsi:type="Text" name="#text" value="integer"/>
        </children>
      </children>
      <children xsi:type="Element" name="db:Column">
        <children xsi:type="Element" name="db:Name">
          <children xsi:type="Text" name="#text" value="UniName"/>
```

```
        </children>
        <children xsi:type="Element" name="db:Nullable">
          <children xsi:type="Text" name="#text" value="false"/>
        </children>
        <children xsi:type="Element" name="db:DataType">
          <children xsi:type="Text" name="#text" value="String"/>
        </children>
        <children xsi:type="Element" name="db:Length">
          <children xsi:type="Text" name="#text" value="20"/>
        </children>
      </children>
    ...
</Root>
```

Listing 7.80. Model dela *UniversityDb* u XML XMI formatu nakon konverzije *injector*-om

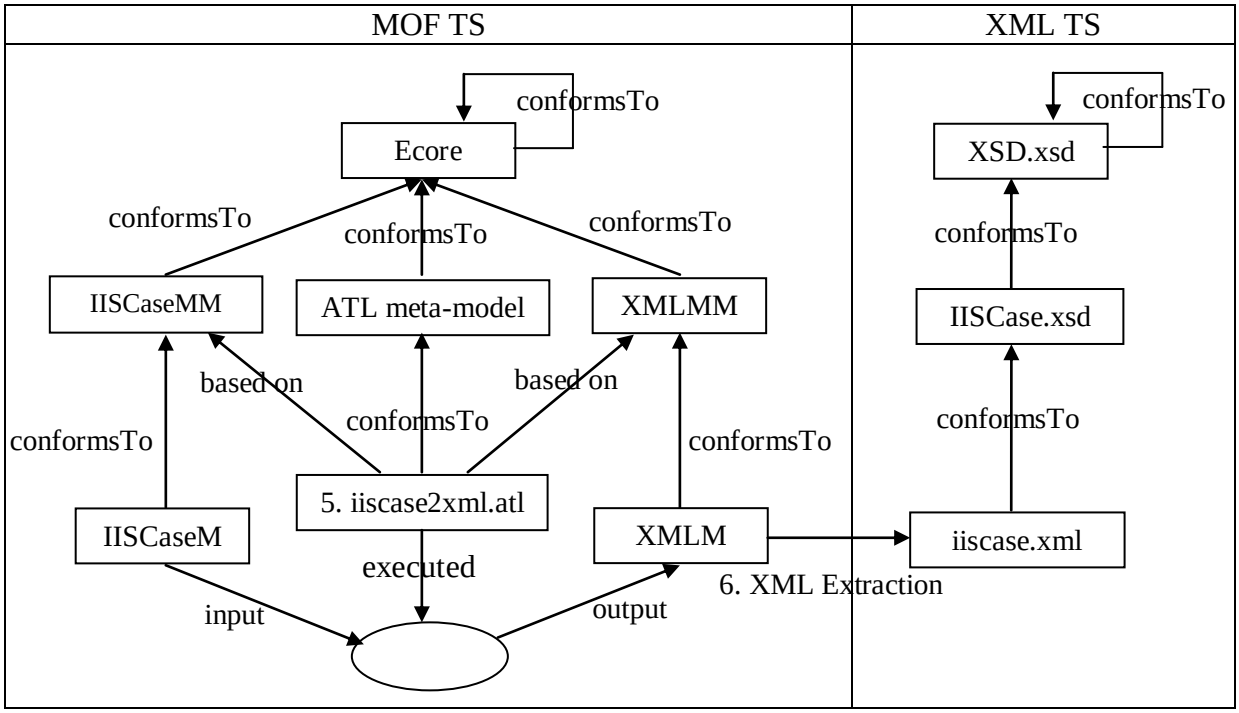
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rdbmsmm:RDBMS xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:rdbmsmm="http://rdbmsmm/1.0" Name="Oracle10g">
  <Databases Name="UniversityDB">
    <Tables Name="University">
      <TablePK Name="PK_University" PKandUQColumns="//@Databases.0/@Tables.0/@Columns.0"/>
      <Columns Name="UniId" Default="" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="UniName" Default="" Length="20" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="UniCity" Default="" Length="20" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="RectorId" Default="" Nullable="true"
                        ColumnDataType="//@DataTypes.0"/>
    </Tables>
    <Tables Name="Faculty">
      <TableFKs Name="FK_University_Faculty" RHS_Key=//@Databases.0/@Tables.0/@TablePK
      LHS_Attr="//@Databases.0/@Tables.1/@Columns.0" RefersTo="//@Databases.0/@Tables.0"/>
      <TablePK Name="PK_Faculty" PKandUQColumns="//@Databases.0/@Tables.1/@Columns.0"/>
      <Columns Name="UniId" Default="" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="FacId" Default="" ColumnDataType="//@DataTypes.0"/>
      <Columns Name="FacShortName" Default="" Length="7" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="FacName" Default="" Length="20" ColumnDataType="//@DataTypes.1"/>
      <Columns Name="Dean" Default="" ColumnDataType="//@DataTypes.0"/>
    </Tables>
  </Databases>
  <DataTypes Name="Integer"/>
  <DataTypes Name="String" PredefinedLength="30"/>
</rdbmsmm:RDBMS>
```

Listing 7.81. Model dela *UniversityDb* u XML XMI formatu nakon transformacije xml2rdbms.atl

7.5. Definicije mapiranja koncepata IISCase meta-modela u elemente XML šeme

U ovoj tački odeljka biće opisani peti i šesti korak transformacije. U petom koraku se model koji je u skladu sa IISCase meta-modelom u MOF tehnološkom prostoru tranformiše u XML model pomoću transformacije iiscase2xml.atl. U poslednjem, šestom koraku, šema baze podataka, predstavljena kao instanca XML meta-modela u MOF tehnološkom prostoru, serijalizuje se u XML specifikaciju iiscase.xml, u XML tehnološkom prostoru, korišćenjem XML *extractor*-a. Ova XML specifikacija zasnovana je na XML šemi IISCase.xsd, koja je kompletno prikazana u prilogu A.2. Ona će predstavljati izlazni rezultat iz modula *M2M Transformer*.

Scenario transformacije poslednja dva koraka prikazan je na slici 7.56.



Slika 7.56. Scenario petog i šestog koraka transformacije

U iiscase2xml.atl transformaciji, koncepti IISCase meta-modela mapiraju se na elemente XML modela. U tabeli 7.4 prikazani su koncepti koji učestvuju u mapiranjima između elemenata IISCase meta-modela, XML meta-modela i IISCase XML šeme.

Tabela 7.4. Koncepti za mapiranje IISCase meta-modela u XML šemu

IISCase meta-model	XML meta-model	IISCase XML šema
ISApplicationModel	`Root`	`xsd:element name= "ISApplicationModel"`
Fundamentals	`Element name= 'Fundamentals'`	`xsd:element name= "Fundamentals"`
ApplicationSystem	`Element name= 'ApplicationSystem'`	`xsd:element name= "ApplicationSystem"`
PrimitiveDomain	`Element name= 'PrimitiveDomain'`	`xsd:element name= "PrimitiveDomain"`
UserDefinedDomainFrom PrimitiveDomain	`Element name= 'UserDefinedDomainFrom PrimitiveDomain'`	`xsd:element name= "UserDefinedDomainFromPrimitiveDomain"`
UserDefinedDomainFromUser DefinedDomain	`Element name=' UserDefinedDomainFromUse r DefinedDomain'`	`xsd:element name= "UserDefinedDomainFromUserDefinedDomain"`
Attribute	`Element name= 'Attribute'`	`xsd:element name="PrimaryKeyCon"`
FormTypeProgram	`Element name= 'FormTypeProgram'`	`xsd:element name= "FormTypeProgram"`
ComponentTypeRoot	`Element name='ComponentTypeRoot'`	`xsd:element name= "ComponentTypeRoot"`

ComponentTypeChild	Element name= 'ComponentTypeChild'	xsd:element name= *"ComponentTypeChild"*
ComponentTypeAttribute	Element name= 'ComponentTypeAttribute'	xsd:element name= *"ComponentTypeAttribute "*
ComponentTypeKey	Element name= 'ComponentTypeKey'	xsd:element name= *"ComponentTypeKey"*
ComponentTypeUnique	Element name= 'ComponentTypeUnique'	xsd:element name= *"ComponentTypeUnique"*
ComponentTypeCheck	Element name= 'ComponentTypeCheck'	xsd:element name= *"ComponentTypeCheck"*

Mapiranja između koncepata IISCase meta-modela i XML meta-modela definisana su pomoću ATL *matched* pravila, u kojima je korišćeno više pomoćnih metoda. ATL specifikacije pravila mapiranja i pomoćnih metoda prikazane su na listinzima od 7.82 do 7.98.

Na listingu 7.82 prikazana je definicija mapiranja koncepta *ISApplicationModel* IISCase mata-modela u korenski element XML meta-modela, pomoću ATL pravila *Root*. Korenski element, tj. model aplikacije sadrži skup osnovnih koncepata (*FundamentalConcepts*) i aplikativnih sistema (*ApplicationSystems*).

Iz ovog pravila pozivaju se pravila *CreateElementWithTextValue* i *CreateAtributeWithTextValue* za kreiranje elemenata i atributa sa tekstualnim sadržajem, prikazane na listingu 7.83.

```
rule Root {
from
   am : IISCase!ISApplicationModel
to
   xam : XML!Root (
      name <- 'ISAplicationModel',
      children <- am.FundamentalConcepts.union(am.ApplicationSystems)
   )
   do {
      thisModule.CreateElementWithTextValue(xam, 'Name', am.Name);
      thisModule.CreateAtributeWithTextValue(xam, 'xmlns',
                                             'http://www.uns.ac.rs/dbXMLSchema');
      thisModule.CreateAtributeWithTextValue(xam, 'xmlns:xsi',
                                    'http://www.w3.org/2001/XMLSchema-instance');
      thisModule.CreateAtributeWithTextValue(xam, 'xsi:schemaLocation',
                                    'http://www.uns.ac.rs/dbXMLSchema dbXMLSchema.xsd');
   }
}
```

Listinig 7.82. ATL pravilo za mapiranje koncepta *ISApplicationModel* na korenski element

```
rule CreateTextValue(element : XML!Node, valueIn : String) {
to
   elem : XML!Text (
      name <- '#text',
      parent <- element,
      value <- valueIn
   )
}

rule CreateElementWithTextValue(element : XML!Node, nameIn : String, value : String) {
to
   elem : XML!Element (
```

```
      name <- nameIn,
      parent <- element
   )
   do {
      thisModule.CreateTextValue(elem, value);
   }
}

rule CreateAtributeWithTextValue(element : XML!Node, nameIn : String, valueIn : String) {
to
   elem : XML!Attribute (
      name <- nameIn,
      value <- valueIn,
      parent <- element
   )
}
```

Listinig 7.83. ATL pravila za kreiranje elemenata i atributa sa tekstualnim sadržajem

ATL pravila za mapiranje osnovnih koncepata: primitivnih domena (*PrimitiveDomain*), korisnički definisanih domena koji nasleđuju primitivni domen (*UserDefinedDomainFromPrimitiveDomain*), korisnički definisanih domena koji nasleđuju korisnički domen (*UserDefinedDomainFromUserDefinedDomain*) i atributa (*AttributeIncludedInDB*) prikazana su na listinzima 7.84, 7.85, 7.87, 7.89 i 7.90, respektivno. Ova pravila pozivaju ATL pomoćne atribute i pomoćne metode prikazane na listinzima 7.86 i 7.88, kojima je rešen rad sa referencama unutar XML specifikacije.

```
rule FundametalConcepts {
from
   funds : IISCase!Fundamentals
to
   out : XML!Element (
      name <- 'Fundamentals',
      children <- funds.FundamentalConcept
   )
}
```

Listinig 7.84. ATL pravilo za mapiranje koncepta *Fundamentals* na element *Fundamentals*

```
rule PrimitiveDomain {
from
   pd : IISCase!PrimitiveDomain
to
   out : XML!Element (
      name <- 'PrimitiveDomain'
   )
   do{
      thisModule.CreateElementWithTextValue(out, 'PrimitiveDomainID', 'pd' +
                                             thisModule.pdidgen.toString());
      thisModule.pdidgen <- thisModule.pdidgen + 1;
      thisModule.CreateElementWithTextValue(out, 'Name', pd.Name.toString());
      thisModule.CreateElementWithTextValue(out, 'DefaultLength',
                                             pd.DefaultLength.toString());
      thisModule.CreateElementWithTextValue(out, 'DefaultDecimalPlaces',
                                             pd.DefaultDecimalPlaces.toString());
      if (not pd.Description.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'Description',
                                             pd.Description.toString());
   }
```

```
      if (not pd.DefaultValue.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'DefaultValue',
                                                pd.DefaultValue.toString());
      }
   }
}
```

Listinig 7.85. ATL pravilo za mapiranje koncepta *PrimitiveDomain* na element *PrimitiveDomain*

```
helper def : pdidgen : Integer = 0;
helper def : udpdidgen : Integer = 0;
helper def : ududidgen : Integer = 0;
helper def : aidbidgen : Integer = 0;
helper def : ctaidgen : Integer = 0;
```

Listinig 7.86. ATL pomoćni atributi

```
rule UserDefinedDomainFromPrimitiveDomain {
from
   udpd : IISCase!UserDefinedDomainFromPrimitiveDomain
to
   out : XML!Element (
      name <- 'UserDefinedDomainFromPrimitiveDomain'
   )
   do{
      thisModule.CreateElementWithTextValue(out, 'UserDefinedDomainFromPrimitiveDomainID',
                                             'udpd' + thisModule.udpdidgen.toString());
      thisModule.udpdidgen <- thisModule.udpdidgen + 1;
      thisModule.CreateElementWithTextValue(out, 'Name', udpd.Name.toString());
      thisModule.CreateElementWithTextValue(out, 'LengthValue',
                                                   udpd.LengthValue.toString());
      thisModule.CreateElementWithTextValue(out, 'DecimalPlaces',
                                             udpd.DecimalPlaces.toString());
      if (not udpd.Description.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'Description',
                                                   udpd.Description.toString());
      }
      if (not udpd.DefaultValue.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'DefaultValue',
                                                   udpd.DefaultValue.toString());
      }
      if (not udpd.CheckCondition.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'CheckCondition',
                                             udpd.CheckCondition.toString());
      }
      thisModule.CreateElementWithTextValue(out, 'InheritsPrimitiveDomain',
      thisModule.getIdFromElement(thisModule.resolveTemp(udpd.InheritsPrimitiveDomain,
                                                         'out')));
   }
}
```

Listinig 7.87. ATL pravilo za mapiranje koncepta *UserDefinedDomainFromPrimitiveDomain* na element *UserDefinedDomainFromPrimitiveDomain*

```
helper def : getIdFromElement(element : XML!Element) : String =
   element.children->select(e | e.name.endsWith('ID')).first().children->select(e |
                                                   e.name='#text').first().value;
```

```
helper def : getIdsFromElement(elementIn : Sequence(XML!Element)) : Sequence(String) =
   if not elementIn.oclIsUndefined() then
      elementIn -> collect(element | element.children->select(e |
      e.name.endsWith('ID')).first().children->select(e |
                                                e.name='#text').first().value + ' ')
   else Sequence{} endif;
```

Listinig 7.88. ATL *helper*-i za rad sa referencama

```
rule UserDefinedDomainFromUserDefinedDomain {
from
   udud : IISCase!UserDefinedDomainFromUserDefinedDomain
to
   out : XML!Element (
      name <- 'UserDefinedDomainFromUserDefinedDomain'
   )
   do{
      thisModule.CreateElementWithTextValue(out,
      'UserDefinedDomainFromUserDefinedDomainID', 'udud' +
      thisModule.ududidgen.toString());
      thisModule.ududidgen <- thisModule.ududidgen + 1;
      thisModule.CreateElementWithTextValue(out, 'Name', udud.Name.toString());
      if (not udud.Description.oclIsUndefined()) {
        thisModule.CreateElementWithTextValue(out, 'Description',
                                                udud.Description.toString());
      }
      if (not udud.DefaultValue.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'DefaultValue',
                                                udud.DefaultValue.toString());
      }
      if (not udud.CheckCondition.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'CheckCondition',
                                                udud.CheckCondition.toString());
      }
      thisModule.CreateElementWithTextValue(out, 'InheritsPrimitiveDomain',
        thisModule.getIdFromElement(thisModule.resolveTemp(udud.InheritsUserDefinedDomain,
                                                                    'out')));
   }
}
```

Listinig 7.89. ATL pravilo za mapiranje koncepta *UserDefinedDomainFromUserDefined Domain* u element *UserDefinedDomainFromUserDefinedDomain*

```
rule AttributeIncludedInDB {
from
   aidb : IISCase!AttributeIncludedInDB
to
   out : XML!Element (
      name <- 'Attribute'
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'AttributeID', 'aidb' +
      thisModule.aidbidgen.toString());
      thisModule.aidbidgen <- thisModule.aidbidgen + 1;
      thisModule.CreateElementWithTextValue(out, 'Name', aidb.Name.toString());
      if (not aidb.Description.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'Description',
                                                        aidb.Description.toString());
      }
      if (not aidb.DefaultValue.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'DefaultValue',
```

```
                                                        aidb.DefaultValue.toString());
      }
      if (not aidb.CheckCondition.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'CheckCondition',
                                                aidb.CheckCondition.toString());
   }
   thisModule.CreateElementWithTextValue(out, 'UserDefinedDomain',
     thisModule.getIdFromElement(thisModule.resolveTemp(aidb.AttributeDomain, 'out')));
   }
}
```

Listinig 7.90. ATL pravilo za mapiranje koncepta *AttributeIncludedInDB* u element *AttributeIncludedInDB*

ATL pravilo za mapiranje koncepta *ApplicationSystem* u odgovarajući element XML meta-modela prikazano je na listingu 7.91. Aplikativni sistem sadrži skup tipova formi koji nastaju mapiranjem pomoću pravila prikazanog na listingu 7.92.

```
rule ApplicationSystem {
from
   aps : IISCase!ApplicationSystem
to
   out : XML!Element (
      name <- 'ApplicationSystem',
      children <- aps.OwnedFormTypes
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'Name', aps.Name.toString());
   }
}
```

Listinig 7.91. ATL pravilo za mapiranje koncepta *ApplicationSystem* u element *ApplicationSystem*

```
rule FormTypeProgram {
from
   ftp : IISCase!FormTypeProgram
to
   out : XML!Element (
      name <- 'FormTypeProgram',
      children <- ftp.RootComponentType
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'Name', ftp.Name.toString());
      thisModule.CreateElementWithTextValue(out, 'Title', ftp.Title.toString());
      thisModule.CreateElementWithTextValue(out, 'Frequency', ftp.Frequency.toString());
      thisModule.CreateElementWithTextValue(out, 'ResponseTime',
                                                        ftp.ResponseTime.toString());
      thisModule.CreateElementWithTextValue(out, 'ConsideredINDBSchDesign',
                                               ftp.ConsideredINDBSchDesign.toString());
   }}
```

Listinig 7.92. ATL pravilo za mapiranje koncepta *FormTypeProgram* u element *FormTypeProgram*

Tip forme sadrži jedan korenski tip komponente (*RootComponentType*) koji može imati više tipova komponenti potomaka (*ComponentTypeChild*). Pravila za mapiranje korenskog tipa komponente i tipa komponente potomka prikazana su na listinzima 7.93 i 7.94. Tipovi

komponenti imaju atribute, skup ekvivalentnih ključeva, skup ograničenja jedinstvenosti i *check* ograničenje. Definicije pravila za mapiranje ovih koncepata pomoću ATL-a prikazana su na listinzima 7.95, 7.96, 7.97 i 7.98, respektivno.

```
rule ComponentTypeRoot {
from
   rct : IISCase!ComponentTypeRoot
to
   out : XML!Element (
      name <- 'ComponentTypeRoot',
      children <- rct.ComponentTypeChildren.union(ctc.ComponentTypeAttributes).
         union(ctc.ComponentTypeKeys).union(ctc.ComponentTypeUniques).
                                                        append(ctc.ComponentTypeCheck)
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'Name', rct.Name.toString());
      thisModule.CreateElementWithTextValue(out, 'Title', rct.Title.toString());
      thisModule.CreateElementWithTextValue(out, 'Query', rct.Query.toString());
      thisModule.CreateElementWithTextValue(out, 'Insert', rct.Insert.toString());
      thisModule.CreateElementWithTextValue(out, 'Delete', rct.Delete.toString());
      thisModule.CreateElementWithTextValue(out, 'Update', rct.Update.toString());
   }
}
```

Listinig 7.93. ATL pravilo za mapiranje koncepta *ComponentTypeRoot* u element *ComponentTypeRoot*

```
rule ComponentTypeChild {
from
   ctc : IISCase!c
to
   out : XML!Element (
      name <- 'ComponentTypeChild',
      children <- ctc.ComponentTypeChildren.union(ctc.ComponentTypeAttributes).
         union(ctc.ComponentTypeKeys).union(ctc.ComponentTypeUniques).
                                                        append(ctc.ComponentTypeCheck)
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'Name', ctc.Name.toString());
      thisModule.CreateElementWithTextValue(out, 'Title', ctc.Title.toString());
      thisModule.CreateElementWithTextValue(out, 'Query', ctc.Query.toString());
      thisModule.CreateElementWithTextValue(out, 'Insert', ctc.Insert.toString());
      thisModule.CreateElementWithTextValue(out, 'Delete', ctc.Delete.toString());
      thisModule.CreateElementWithTextValue(out, 'Update', ctc.Update.toString());
      thisModule.CreateElementWithTextValue(out, 'NoOfOcurrences',
                                             ctc.NoOfOcurrences.toString());
   }
}
```

Listinig 7.94. ATL pravilo za mapiranje koncepta *ComponentTypeChild* u element *ComponentTypeChild*

```
rule ComponentTypeAttribute {
from
   cta : IISCase!ComponentTypeAttribute
to
   out : XML!Element (
      name <- 'ComponentTypeAttribute'
   )
```

```
   do {
      thisModule.CreateElementWithTextValue(out, 'ComponentTypeAttributeID', 'cta' +
            thisModule.ctaidgen.toString());
      thisModule.ctaidgen <- thisModule.ctaidgen + 1;
      thisModule.CreateElementWithTextValue(out, 'Title', cta.Title.toString());
      thisModule.CreateElementWithTextValue(out, 'Mandatory', cta.Mandatory.toString());
      if (not cta.DefaultValue.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'DefaultValue',
                                                        cta.DefaultValue.toString());
      }
      thisModule.CreateElementWithTextValue(out, 'ComponentTypeAttributeName',
        thisModule.getIdFromElement(thisModule.resolveTemp(cta.ComponentTypeAttributeName,
                                                                  'out')));
   }
}
```

Listinig 7.95. ATL pravilo za mapiranje koncepta *ComponentTypeAttribute* u element *ComponentTypeAttribute*

```
rule ComponentTypeKeys {
from
   ctk : IISCase!ComponentTypeKey
   using {
      concat : String = '';
      seq : Sequence(String) = Sequence{};
   }
to
   out : XML!Element (
      name <- 'ComponentTypeKey'
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'Name', ctk.Name.toString());
      thisModule.CreateElementWithTextValue(out, 'Global', ctk.Global.toString());
      seq <- thisModule.getIdsFromElement(ctk.ComponentTypeKeyAttributes -> collect(c |
            thisModule.resolveTemp(c, 'out')));
      concat <- seq.sum();
      if (not concat.oclIsUndefined()) {
         concat <- concat.trim();
      } else {
         concat <- '';}
      thisModule.CreateElementWithTextValue(out, 'ComponentTypeKeyAttributes', concat);
   }
}
```

Listinig 7.96. ATL pravilo za mapiranje koncepta *ComponentTypeKey* u element *ComponentTypeKey*

```
rule ComponentTypeUnique {
from
   ctu : IISCase!ComponentTypeUnique
   using {
      concat : String = '';
      seq : Sequence(String) = Sequence{};
   }
to
   out : XML!Element (
      name <- 'ComponentTypeKey'
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'Name', ctu.Name.toString());
      seq <- thisModule.getIdsFromElement(ctu.ComponentTypeUniqueAttributes -> collect(c |
```

```
            thisModule.resolveTemp(c, 'out')));
      concat <- seq.sum();
      if (not concat.oclIsUndefined()) {
         concat <- concat.trim();
      } else {
         concat <- '';
      }
      thisModule.CreateElementWithTextValue(out, 'ComponentTypeUniqueAttributes', concat);
   }
}
```

Listinig 7.97. ATL pravilo za mapiranje koncepta *ComponentTypeUnique* u element *ComponentTypeUnique*

```
rule ComponentTypeCheck {
from
   ctc : IISCase!ComponentTypeCheck
   using {
      concat : String = '';
      seq : Sequence(String) = Sequence{};
   }
to
   out : XML!Element (
      name <- 'ComponentTypeCheck'
   )
   do {
      thisModule.CreateElementWithTextValue(out, 'Name', ctc.Name.toString());
      if (not ctc.CheckCondition.oclIsUndefined()) {
         thisModule.CreateElementWithTextValue(out, 'CheckCondition',
                                               ctc.CheckCondition.toString());
      }
   }
}
```

Listinig 7.98. ATL pravilo za mapiranje koncepta *ComponentTypeCheck* u element *ComponentTypeCheck*

Za primer modela sa listinga 7.81, koji predstavlja ulazni model u niz transformacija opisanih u prethodnim odeljcima, nakon svih transformacija, tj. nakon primene i ove poslednje transformacije iiscase2xml.atl, dobija se model, čiji je deo u obliku XMI specifikacije prikazan na listingu 7.99. Na lisitngu 7.100 prikazan je deo XML specifikacije ovog modela (deo iiscase.xml), nastale konverzijom pomoću XML *extractor*-a, koja se vraća modulu *XML Transformer* na dalju obradu. Rezultujuća XML specifikacija je u skladu sa XML šemom IISCase.xsd, datom u prilogu A.2.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project:ISApplicationModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:FormType="http://model/1.0/FormType" xmlns:atribute="http://model/1.0/attribute"
xmlns:domain="http://model/1.0/domain" xmlns:project="http://model/1.0/project"
Name="IISCase_Project_Oracle10g">
  <FundamentalConcepts>
    <FundamentalConcept xsi:type="atribute:AttributeIncludedInDB" Name="FacShortName"
      Description="FacShortName" AttributeDomain=
      "//@FundamentalConcepts.0/@FundamentalConcept.14"/>
    ...
  </FundamentalConcepts>
  <ApplicationSystems Name="AppSystem_UniversityDB">
```

```
    <OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name="FormType_University"
         Title="FormType_University" Frequency="1" ResponseTime="1"
         ConsideredINDBSchDesign="true">
      <RootComponentType Name="ComponentType_University" Title="ComponentType_University"
                                                                    Query="true">
        <ComponentTypeAttributes ComponentTypeAttributeName="//@FundamentalConcepts.0/
          @FundamentalConcept.7" Title="UniId" Mandatory="true" DefaultValue=""/>
      ...
      </RootComponentType>
    </OwnedFormTypes>
    <OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name="FormType_Faculty"
      Title="FormType_Faculty" Frequency="1" ResponseTime="1"
          ConsideredINDBSchDesign="true">
      <RootComponentType Name="ComponentType_Faculty" Title="ComponentType_Faculty"
                                                                    Query="true">
        <ComponentTypeAttributes ComponentTypeAttributeName="//@FundamentalConcepts.0/
          @FundamentalConcept.7" Title="UniId" Mandatory="true" DefaultValue=""/>
        ...
    </OwnedFormTypes>
    <OwnedFormTypes xsi:type="FormType:FormTypeProgram" Name=
         "FormType_FK_University_Faculty" Title="FormType_FK_University_Faculty"
         Frequency="1" ResponseTime="1" ConsideredINDBSchDesign="true">
      <RootComponentType Name="ComponentTypeRoot_University" Title=
         "ComponentTypeRoot_University" Query="true">
        ...
        <ComponentTypeChildren Name="ComponentType_Faculty" Title="ComponentType_Faculty"
          Query="true" NoOfOcurrences="NoneOrMany">
        ...
        </ComponentTypeChildren>
      </RootComponentType>
    </OwnedFormTypes>
  </ApplicationSystems>
</project:ISApplicationModel>
```

Listing 7.99. Model dela *UniversityDb* u XML XMI formatu nakon transformacije iiscase2xml.atl

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<ISAplicationModel xmlns="http://www.uns.ac.rs/dbXMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.uns.ac.rs/dbXMLSchema dbXMLSchema.xsd">
  <Fundamentals>
    <Attribute>
      <AttributeID>aidb0</AttributeID>
      <Name>FacShortName</Name>
      <Description>FacShortName</Description>
      <UserDefinedDomain>udpd7</UserDefinedDomain>
    </Attribute>
    <Attribute>
      <AttributeID>aidb1</AttributeID>
      <Name>FacName</Name>
      <Description>FacName</Description>
      <UserDefinedDomain>udpd8</UserDefinedDomain>
    </Attribute>
    ...
  </Fundamentals>
  <ApplicationSystem>
    <FormTypeProgram>
      <ComponentTypeRoot>
        <Name>ComponentType_University</Name>
        <Title>ComponentType_University</Title>
        <Query>true</Query>
        <Insert>false</Insert>
```

```
        <Delete>false</Delete>
        <Update>false</Update>
      </ComponentTypeRoot>
      <Name>FormType_University</Name>
      <Title>FormType_University</Title>
      <Frequency>1</Frequency>
      <ResponseTime>1</ResponseTime>
      <ConsideredINDBSchDesign>true</ConsideredINDBSchDesign>
    </FormTypeProgram>
    <FormTypeProgram>
      <ComponentTypeRoot>
        <Name>ComponentType_Faculty</Name>
        <Title>ComponentType_Faculty</Title>
        <Query>true</Query>
        <Insert>false</Insert>
        <Delete>false</Delete>
        <Update>false</Update>
      </ComponentTypeRoot>
      <Name>FormType_Faculty</Name>
      <Title>FormType_Faculty</Title>
      <Frequency>1</Frequency>
      <ResponseTime>1</ResponseTime>
      <ConsideredINDBSchDesign>true</ConsideredINDBSchDesign>
    </FormTypeProgram>
    ...
    <Name>AppSystem_UniversityDB</Name>
  </ApplicationSystem>
  <Name>IISCase_Project_Oracle10g</Name>
</ISAplicationModel>
```

Listing 7.100. Model dela *UniversityDb* u XML formatu nakon primene XML *extractor*-a

7.6. Zaključak

Pristup reinženjeringu informacionih sistema, koji je prezentovan u ovoj doktorskoj disertaciji, podrazumeva primenu MDA pristupa u procesu ektrakcije i koneptualizacije modela iz postojeće baze podataka. U ovom poglavlju predstavljen je postupak konceptualizacije modela. Prikazani su algoritmi, formalne specifikacije i implementacija metoda transformacija opisa šema baza podataka na različitim meta-nivoima, originalno nastali kao rezultat istraživanja u ovoj doktorskoj disertaciji.

Postupak transformacije, koji je implementiran kao niz nekoliko transformacija, ugrađen je u modul *M2M Transformer*, koji je sastavni deo IIS*Ree alata. Niz transformacija započinje transformacijom ekstrahovanog modela, koji opisuje specifikaciju implementacije sistema na konkretnoj platformi i predstavlja PSM model. Nakon toga slede transformacije čiji ulaz i izlaz predstavljaju modeli koji opisuju sistem sa sve višim stepenom nezavisnosti od platforme. Niz transformacija završava se transformacijom koja generiše model funkcionalnosti sistema koji je nezavisan od platforme na kojoj će biti implementiran i koji predstavlja PIM model. Transformacije su zasnovane na meta-modelima i izvršavaju se u MOF tehnološkom prostoru.

Niz transformacija ugrađenih u modul, u kojima model na izlazu iz jedne transformacije predstavlja model na ulazu u drugu, čine sledeće transformacije:

- XML2RDBMS,
- RDBMS2RM,
- RM2IISCase i
- IISCase2XML.

U prvoj transformaciji, logička šema baze podataka, opisana XML jezikom, transformiše se u model opisan pomoću koncepata relacionih sistema za upravljanje bazama podataka koji

je kompatibilan sa delom SQL standarda. Pomoću naredne transformacije, RDBMS2RM, ovaj model transformiše se u model koji je u skladu sa Generičkim meta-modelom koji reprezentuje koncepte relacione šeme baze podataka zasnovane na teoretskim principima relacionog modela. Novonastali model se potom, trećom, RM2IISCase transformacijom, transformiše u model koji predstavlja konceptualnu šemu baze podataka zasnovanu na PIM konceptima IIS*Case alata. Rezultat ove transformacije transformiše se u XML model pomoću poslednje transformacije, IISCase2XML, i serijalizuje u XML format.

Ovaj rezultat, koji predstavlja izlaz iz modula *M2M Transformer*, vraća se u modul *XML Transformer*. *XML Transformer* parsira XML specifikaciju konceptualne šeme baze podataka i njene elemente smešta u repozitorijum okruženja IIS*Studio, odakle se putem korisničkog interfejsa alata IIS*Case može izvršiti restrukturiranje, a zatim kroz proces *forward* inženjeringa ponovo generisanje šeme baze podataka na implementacionom nivou, za izabranu tehnološku platformu.

8. Zaključak

U ovom, završnom poglavlju disertacije biće rezimirani glavni doprinosi sprovedenog istraživanja, prednosti i nedostaci realizovanog rešenja, kao i pravci budućih istraživanja.

Na početku ovog istraživanja, definisane su polazne hipoteze iz kojih su proizašli osnovni ciljevi doktorske disertacije. Glavni cilj bio je da se pokaže da je proces reinženjeringa informacionih sistema i praktično, a ne samo teoretski, moguće realizovati primenom MD pristupa. Da bi ovakva hipoteza bila opravdana, formulisan je originalni metodološki pristup i kreirano softversko okruženje, namenjeni unapređenju procesa reinženjeringa informacionih sistema. Izabran je pristup koji se zasniva na analizi i transformacijama opisa baza podataka. Baze podataka smatraju se jednim od najosnovnijih artefakata sistema koje sadrže većinu informacija vezanih za postojeći informacioni sistem. U predloženom metodološkom pristupu primenjen je model potkovice kojim je obuhvaćen kompletan životni ciklus reinženjeringa opisa baze podataka. Primenjeni model potkovice obuhvata reverzni inženjering, restrukturiranje i *forward* inženjering zasnovan na upotrebi MDA principa i tehnika.

U okviru istraživanja posebna pažnja posvećena je procesu reverznog inženjeringa koji obuhvata dve faze: fazu ekstrakcije i fazu konceptualizacije modela. Nakon što se model ekstrahuje, koriste se tehnike transformacije modela u cilju njegove konceptualizacije. Jedan od realizovanih ciljeva ove doktorske teze je metod čijom primenom se obezbeđuje, kao krajnji rezultat reverznog inženjeringa, konceptualni model nasleđene baze podataka. Metod se sastoji od niza transformacija. Prva u tom nizu je transformacija ekstrahovanog modela iz nasleđene baze podataka. To je model koji je određen implementacionom platformom i predstavlja PSM model. Nakon ove, slede transformacije čiji ulaz i izlaz predstavljaju modeli koji opisuju sistem sa sve višim stepenom nezavisnosti od platforme. Niz transformacija završava se transformacijom koja generiše model koji je nezavisan od platforme na kojoj će biti implementiran i koji predstavlja PIM model. PIM koji je korišćen u ovoj doktorskoj disertacji je model zasnovan na konceptu tipa forme. Kao predmet reinženjeringa izabran je relacioni model podataka, kao jedna od veoma često korišćenih paradigmi u izgradnji baza podataka informacionih sistema. U skladu sa MDA, pravila transformacije modela definisana su na nivou meta-modela.

U cilju opravdavanja hipoteza H3 i H4, formulisanih u Uvodu ovog rada, razvijeni su neophodni meta-modeli i transformacije zasnovane na tim meta-modelima, kojima je realizovan automatizovani proces konceptualizacije. Meta-modeli su implementirani pomoću Ecore meta-metamodela u EMF okruženju. Transformacije su implementirane pomoću namenskog jezika za transformacije modela ATL, takođe u EMF okruženju. Ecore, kao implementacija MOF meta-metamodela koji je propisan OMG standardom, i ATL, kao implemtacija QVT-a koji je takođe propisan OMG standardom, pripadaju MDA tehnologijama. Ovakvom primenom MDA tehnologija u implementaciji transformacija modela opravdana je i hipoteza ove disertacije H5.

Predloženi pristup postavio je teorijski okvir na osnovu kojeg je razvijen softverski alat IIS*Ree. IIS*Ree je zasnovan na MDSD principima, u cilju omogućavanja visokog nivoa automatizacije procesa reinženjeringa informacionih sistema. Novi alat razvijen je u okviru

okruženja IIS*Studio, namenjenog projektovanju informacionih sistema i njihovih šema baza podataka. Krajnji proizvod, razvijan u ovom okruženju, predstavlja funkcionalni prototip informacionog sistema.

IIS*Ree se sastoji iz četiri glavna modula, originalno nastala u okviru ovog istraživanja, pomoću kojih se vrši postupak reverznog inženjeringa:

- *Legacy System Interface*,
- *MetaData Validator & Constraints Discoverer*,
- *XML Transformer* i
- *M2M Transformer*.

Legacy System Interface je modul u kojem je implementiran skup algoritama za ekstrahovanje svih mogućih informacija iz izvorne, nasleđene relacione baze podataka. Kao izvor informacija koristi rečnik podataka izvorne baze podataka, kao i same podatke.

MetaData Validator & Constraints Discoverer je modul putem kojeg se razrešavaju kolizije u nazivima atributa i pronalaze kandidati za specifikaciju ograničenja inverznog referencijalnog integriteta.

Modul *XML Transformer* ima dve uloge. Prva je da podatke iz repozitorijuma okruženja IIS*Studio, koji predstavljaju formalni opis logičke šeme izvorne baze podataka, serijalizuje u XML specifikaciju na osnovu predefinisane XML šeme ugrađene u alat IIS*Ree. Ova XML specifikacija predstavlja ulaz u proces transformacije koji se obavlja u okviru modula *M2M Transformer*. Takođe, modul *XML Transformer* ima još i ulogu da XML specifikaciju, nastalu kao rezultat procesa transformacije u okviru modula *M2M Transformer*, parsira na osnovu druge predefinisane XML šeme. Rezultat parsiranja smešta ponovo u repozitorijum okruženja IIS*Studio. Modul *XML Transformer* ima veoma važnu ulogu u povezivanju pojedinačnih faza procesa reinženjeringa, u njegovoj praktičnoj realizaciji.

M2M Transformer je modul koji preuzima XML specifikaciju izvornog modela, nad kojom primenjuje niz algoritama za transformacije koje generišu opise šema baza podataka na različitim nivoima apstrakcije. Kao krajnji rezultat rada ovog modula dobija se konceptulani model, tj. instanca meta-modela koji je zasnovan na PIM konceptima alata IIS*Case. Do konceptualnog modela dolazi se pomoću dve osnovne transformacije: RDBMS2RM i RM2IISCase i dve dodatne: XML2RDBMS i IISCase2XML. Transformacija RDBMS2RM transformiše model relacione baze podataka, koji je u skladu sa RSUBP meta-modelom, u model relacione baze podataka koji je u skladu sa Generičkim meta-modelom. Transformacija RM2IISCase transformiše model relacione baze podataka koji je u skladu sa Generičkim meta-modelom u konceptualni model koji je u skladu sa IISCase meta-modelom. XML2RDBMS transformiše šemu baze podataka koja je u skladu sa XML meta-modelom u model relacione baze podataka koji je u skladu sa RSUBP meta-modelom, dok IISCase2XML transformiše opis konceptualne šeme baze podataka koja je u skladu sa IISCase meta-modelom u model koji je u skladu sa XML meta-modelom. Dodatne transformacije bile su neophodne radi usklađivanja različitih tehnoloških prosotra u kojima se nalaze modeli u različitim fazama procesa reinženjeringa. Kao i modul *XML Transformer*, dodatne transformacije imaju ulogu posrednika koji omogućavaju automatsko povezivanje pojedinačnih faza procesa reinženjeringa u njegovoj praktičnoj realizaciji.

Modulima *Legacy System Interface* i *MetaData Validator & Constraints Discoverer* obuhvaćena je prva faza reverznog inženjeringa, odnosno ekstrakcija modela iz nasleđene baze podataka. U ovoj fazi reverznog inženjeringa neophodna je pomoć korisnika-projektanta u podešavanju parametara potrebnih za konekciju na nasleđenu bazu podataka, kao i u razrešavanju kolizija u nazivima atributa i izboru potencijalnih ograničenja referencijalnog integriteta. Pomoću modula *M2M Transformer* realizuje se postupak konceptualizacije. Kako se pokazalo da u ovoj fazi nema potrebe za obezbeđenjem interakcije s korisnikom, postupak konceptualizacije je, u predloženom pristupu, u potpunosti automatizovan. Automatizovani

postupak transformacije koji se odvija u okviru modula *M2M Transformer* implementiran je u EMF okruženju, pomoću ATL jezika.

U cilju obezbeđenja kompletne podrške postupka reinženjeringa, IIS*Ree uključuje kao jedan svoj podsistem, alat IIS*Case. Time su obuhvaćeni postupci restrukturiranja i *forward* inženjeringa. Uključivanjem alata IIS*Case radi kompletiranja procesa reinženjeringa, opravdana je i hipoteza H2. Alati IIS*Ree i IIS*Case, koji pripadaju okruženju IIS*Studio, koriste isti repozitorijum, čime je omogućena izrazito važna prednost u procesu reinženjeringa, a to je obezbeđenje mogućnosti automatskog povezivanja rezultata pojedinačnih faza u praktičnoj realizaciji procesa reinženjeringa. IIS*Case, koji je nastao kao rezultat ranijih istraživanja, implementiran je pomoću Java programskog jezika u objektno-orijentisanom okruženju Oracle Jdeveloper.

U ovom istom okruženju, pomoću Java programskog jezika mogao je biti implementiran čitav alat IIS*Ree, uključujući i modul *M2M Transformer*. Za implementaciju svih potrebnih transformacija mogao je biti primenjen pristup tzv. direktne manipulacije. Prednost takvog pristupa bio bi u uštedi vremena koje je inače utrošeno za savladavanje i praktičnu primenu jednog potpuno novog jezika za transformacije, a što bi svakako pozitivno uticalo na brzinu razvoja rešenja. S druge strane, tako kreirani programski kod bio bi izrazito obimniji, težak za održavanje i neupotrebljiv za ponovno korišćenje u drugim transformacijama. U okviru istraživanja ove doktorske disertacije, pri izboru tehnološkog okruženja za razvoj alata IIS*Ree, namera je bila da se isprobaju nove tehnologije, odnosno alati i jezici, sa željom da se iskoriste njihove prednosti. U izabranom tehnološkom okruženju koje je zasnovano na MDA principima i pomoću jezika za transformaciju modela ATL mogao je da se implementira deo procesa reinženjeringa koji se odnosi na trasformacije šema baza podataka obuhvaćene modulom *M2M Transformer*. Jedna od dobrih osobina primene ATL jezika u implementaciji transformacija je u tome što ATL jezik daje prednost deklarativnom pristupu. U ovom pristupu programer vodi računa o tome *šta* se mapira, a ne na koji način se transformacija izvršava. Čitava transformacija sastoji se od skupa pravila kojima se mapiraju odgovarajući koncepti meta-modela. Za razliku od programskog koda napisanog u Java programskom jeziku, male izmene u delovima ne zahtevaju izmene čitavog sistema. Takođe, napravljeni meta-modeli mogu se koristiti u transformacijama šema baza podataka u obrnutom smeru, u postupku *forward* inženjeringa, kao i u transformacijama u kojima bi izvorni ili ciljni modeli bili zasnovani na nekim drugim modelima podataka.

Glavni nedostatak predstavljenog rešenja je u sledećem. Proces ekstrakcije modela iz nasleđene baze podataka, zbog potrebe za interakcijom sa korisnikom, nije mogao biti realizovan pomoću jezika ATL, već je bilo potrebno koristiti programski jezik opšte namene. Za implementaciju je izabran Java programski jezik i objektno-orijentisano okruženje Oracle Jdeveloper, jer je u tom okruženju implementiran i proces transformacije modela u postupku *forward* inženjeringa. Time je obezbeđeno da se ekstrahovani model, koji treba da predstavlja ulaz u proces transformacije, može relativno jednostavno smestiti u repozitorijum alata IIS*Ree, odnosno u tehnološki prostor RSUBP-a, dok se transformacije mogu izvršavati nad modelima definisanim u MOF tehnološkom prostoru.

Usled korišćenja dva okruženja i različitih tehnika za implementaciju čitavog procesa reinženjeringa šema baza podataka, javlja se problem povezivanja modula, odnosno razmene modela između njih i usaglašavanje tehnoloških prostora. Ovaj problem rešen je na sledeći način. Uveden je dodatni modul *XML Transformer* koji omogućava razmenu modela između modula pomoću XML specifikacija. Izabran je XML kao format za razmenu podataka zbog toga što ATL SDK u kojem je razvijen modul *XML Transformer* automatski podržava uvoz i izvoz XML specifikacija. Generisanjem XML specifikacije, model se iz tehnološkog prostora RSUBP-a prebacuje u XML tehnološki prostor. Međutim, generisana XML specifikacija, koja sadrži opis ulaznog modela, i dalje ne može direktno biti iskorišćena kao ulaz u

transformaciju koja se izvršava u okviru modula *M2M Transformer*. Potrebno je, zbog toga, XML ulaznu specfikaciju prilagoditi MOF tehnološkom prostoru. Takođe, rezultat transformacije koji je u MOF tehnološkom prostoru treba prilagoditi ponovo XML tehnološkom prostoru. Problem prevazilaženja neusklađenosti MOF i XML tehnoloških prosotra rešen je u okviru modula *M2M Transformer* primenom metode koja obuhvata specifikaciju i implementaciju dve dodatne transformacije: XML2RDBMS i IISCase2XML.

U ovoj verziji, kao izlaz iz IIS*Ree alata može se dobiti sledeće:

- konceptualni model podataka opisan PIM konceptima IISCase-a,
- implementacioni opis šeme baze podataka za različite relacione sisteme za upravljanje bazama podataka i
- XML specifikacija informacionog sistema, odnosno XML specifikacija tipova formi i XML specifikacija šeme baze podataka.

Trenutno, pomoću IIS*Ree može se vršiti reinženjering baza podataka implementiranih na sledećim SUBP-ovima: Oracle (verzije 9i, 10g i 11g) i MS SQL Server (verzije 2000, 2005, 2008 i 2012). Alat, takođe, može da generiše implementacioni opis relacionih baza podataka za konkretne proizvođače Oracle (verzije 9i, 10g i 11g) i MS SQL Server (verzije 2000, 2005, 2008 i 2012), kao i ANSI SQL standard.

Praktična provera valjanosti razvijenih transformacija, koje su u potpunosti implementirane u okviru IIS*Ree alata, obavljena je na jednom izabranom primeru veoma pojednostavljenog i hipotetičkog modela informacionog sistema univerziteta. Na ovaj način opravdana je hipoteza H1.

Na osnovu razmatranja u prethodnim poglavljima, sa ciljem sistematizacije prikaza predloženog rešenja, navode se naučni i stručni doprinosi ove disertacije.

Dominantno naučni doprinisi su:

- analiza postojećih pristupa rešavanju problema relevantnih za predmet istraživanja ove doktorske disertacije,
- formulacija originalnog modela potkovice reinženjeringa IS-a zasnovan na principima primene MDA,
- formulacija originalnog modela arhitekture alata koji podržava model potkovice,
- formulacija originalnog pristupa autora u reverznom inženjeringu baza podataka, zasnovanog na integraciji tradicionalnih tehnika reverznog inženjeringa, naprednih tehnika meta-modelovanja i transformacijama zasnovanim na meta-modelima,
- formulisan predlog klasifikacije meta-modela šema baza podataka,
- specifikacija meta-modela relacione šeme baze podataka na PSM nivou, zasnovanog na standardu sa kojim je u solidnoj meri kompatiblna većina RSUBP-ova (RSUBP meta-model),
- specifikacija generičkog meta-modela relacione šeme baze podataka koji je u skladu sa teoretskim definicijama relacionog modela podataka (Generički meta-model),
- specifikacija pravila za unapređenje semantike meta-modela,
- identifikacija teorijskih okvira i algoritama za transformaciju modela relacione baze podataka, koji je u skladu sa RSUBP meta-modelom, u model relacione baze podataka koji je u skladu sa Generičkim meta-modelom,
- identifikacija teorijskih okvira i algoritama za transformaciju modela relacione baze podataka koji je u skladu sa Generičkim meta-modelom u konceptualni model koji je u skladu sa IISCase meta-modelom,
- specifikacija algoritma za otklanjanje neusaglašenosti koje se odnose na homonime u modelu ekstrahovanom iz nasleđene baze podataka,
- specifikacija algoritma koji proširuje model ekstrahovan iz nasleđene baze podataka informacijama o potencijalnim ograničenjima inverznog referencijalnog integriteta i

- kreiranje postupka za prevazilaženje problema različitih tehnoloških prostora, XML i MOF, koji obuhvata specifikaciju potrebnih XML šema i specifikaciju transformacija:
  - šeme baze podataka koja je u skladu sa XML meta-modelom u model relacione baze podataka koji je u skladu sa RSUBP meta-modelom i
  - opisa konceptualne šeme baze podataka koja je u skladu sa IISCase meta-modelom u model koji je u skladu sa XML meta-modelom.

Ostvareni, dominantno stručni i praktični doprinosi ove disertacije su:

- implementacija MDSD alata IIS*Ree u kojem je primenjen predloženi pristup procesu reinženjeringa IS-a i model potkovice,
- implementacija RSUBP meta-modela i Generičkog meta-modela pomoću Ecore jezika,
- implementacija transformacija modela relacione baze podataka, koji je u skladu sa RSUBP meta-modelom, u model relacione baze podataka koji je u skladu sa Generičkim meta-modelom pomoću jezika ATL,
- implementacija transformacija modela relacione baze podataka koji je u skladu sa Generičkim meta-modelom u konceptualni model koji je u skladu sa IISCase meta-modelom pomoću jezika ATL,
- implementacija transformacija potrebnih za prevazilaženje problema različitih tehnoloških prostora pomoću jezika ATL i
- implementacija pravila za unapređenje semantike prezentovanih meta-modela pomoću jezika OCL.

Tokom istraživanja prezentovanih u ovoj doktorskoj disertaciji identifikovano je nekoliko pravaca za dalja istraživanja.

Jedan od pravaca budućih istraživanja treba da vodi ka daljem razvoju i poboljšanju predloženog teorijskog okvira. Metodologiju korišćenu u ovom radu treba proširiti novim metodama i algoritmima za transformaciju različitih opisa baza podataka koje mogu biti izvor procesa reinženjeringa. Ovo bi podrazumevalo uvođenje novih meta-modela, kojima bi se definisale apstraktne sintakse jezika za modelovanje šema baza podataka, na kojima bi bile zasnovane transformacije. Kao predmeti reinženjeringa mogle bi se naći šeme baza podataka zasnovane na objektnom, objektno-relacionom ili XML modelu podataka.

Dalja istraživanja biće posvećena i proširivanju alata IIS*Ree modulima koji bi podržali na ulazu, kao izvor reinženjeirnga, osim SUBP Oracle i Microsoft SQL Server, i druge proizvođače relacionih sistema za upravljanje bazama podataka ili sisteme za upravljanje bazama podataka zasnovane na nekim drugim modelima podataka, kao što su objektni, objektno-relacioni ili XML modeli. Takođe, alat IIS*Ree treba proširiti i novim generatorima implementacionog opisa šeme baze podataka za različite proizvođače relacionih sistema za upravljanje bazama podataka ili sisteme za upravljanje bazama podataka zasnovane na nekim drugim modelima podataka, kao što su objektni, objektno-relacioni ili XML model.

Jedan od izazova budućeg rada bila bi i implementacija unapređenog algoritma sinteze relacione šeme baze podataka primenom MDA prinicipa, korišćenjem Eclipse okruženja i jezika ATL. Algoritam sinteze je specijalno unapređen za potrebe praktične primene u pristupu i alatu IIS*Case. Transformacije, pomoću kojih bi konceptualni model bio transformisan u relacioni model podataka, bile bi zasnovane na istim meta-modelima koji su predstavljeni u ovoj doktorskoj disertaciji. Transformacija bi obuhvatala izvođenje skupa funkcionalnih, nefunkcionalnih i specijalnih funkcionalnih zavisnosti iz svih tipova formi uključenih u ulaznu specifikaciju, a kao rezultat dobio bi se skup šema relacija sa definisanim skupovima atributa, sintetizovanim ključevima i ograničenjima jedinstvene vrednosti.

Kao jedan od značajnih pravaca istraživanja izdvaja se i obezbeđivanje migracije podataka iz nasleđenih baza podataka u novokreirane baze podatka, nastale kao rezultat reinženjeringa.

Istraživačke aktivnosti u širem smislu, tj. aktivnosti na daljem razvoju odgovarajućeg pristupa i okruženja IIS*Studio mogu se podeliti u dve grupe:

- proširivanje metamodela IIS*Case-a i
- nadogradnja IIS*Case-a novim alatima namenjenim modelovanju putem vizuelnih i tekstuelnih notacija. Ti alati treba da vrše i formalnu proveru semantičke validnosti modela u toku modelovanja.

Proširivanje metamodela alata IIS*Case odnosi se na uvođenje novih koncepata kako bi model, kreiran putem ovog alata, sadržao što potpuniju specifikaciju informacionog sistema. Dalja istraživanja biće posvećena uvođenju koncepata i alata namenjenih modelovanju sistemske arhitekture. Time bi bila stvorena osnova za automatizovanje koraka isporuke i instaliranja generisanih aplikacija. Istraživački napori biće usmereni i na proširivanje metamodela kako bi bilo omogućeno modelovanje onog dela informacionog sistema koji se odnosi na bezbednost programa i podataka. Pored toga, buduća istraživanja biće posvećena formulisanju i realizaciji algoritama namenjenih transformaciji modela kreiranog putem IIS*Case-a u model zadat putem UML-a. Time se postiže bolja razmenljivost specifikacija sa drugim alatima.

Budući istraživački napori biće posvećeni kreiranju novog vizuelno orijentisanog alata koji je namenjen zadavanju specifikacija tipova formi. S druge, strane već je razvijen jezik *IIS*CDesLang* putem kojeg je moguće zadati specifikacije tipova formi. Dalja istraživanja biće posvećena razvoju transformacija čiji je zadatak da na osnovu sadržaja repozitorijuma generišu ekvivalente specifikacije u *IIS*CDesLang*-u, kao i obratan smer, dopunjavanje i modifikacija sadržaja repozitorijuma na osnovu specifikacija u *IIS*CDesLang*-u. Na taj način, korisnicima će biti omogućeno da paralelno koriste i tekstuelnu i vizuelnu notaciju za zadavanje specifikacija, a stvara se i mogućnost vršenja uporedne analize ova dva pristupa zadavanju specifikacija. Pored navedenog, stvara se osnova za razvoj parsera koji će vršiti proveru semantičke ispravnosti modela, kao što je provera da li postoje kolizije *check* ograničenja.

Takođe, jedan od ne manje važnih pravaca daljeg razvoja mogao bi biti i rad na dovođenju laboratorijske verzije alata IIS*Ree i čitavog okruženja IIS*Studio u stanje spremnosti za komercijalnu upotrebu. Poštujući činjenicu da sam čin dovođenja jednog kompleksnog softverskog okruženja, kakvo je i IIS*Studio, do stanja spremnosti za komercijalnu upotrebu uvodi čitav niz novih ili modifikovanih postojećih zahteva kojima se mora pristupiti na posebno sistematičan način, neophodno je aktivno povezati kumulirana, teorijska i interdisciplinarna znanja s jedne strane, i praktično iskustvo s druge strane, što svakako predstavlja i istraživački izazov, a ne samo rutinski inženjerski posao.

Literatura

ADM Architecture-Driven Modernization (ADM), dostupno na: http://adm.omg.org. (Septembar, 2013)

Akehu02 Akehurst D, Kent S, *A relational approach to defining transformations in a metamodel*, in Proceedings UML 2002, Springer-Verlag, LNCS, Vol. 2460, 2002.

Akehu05 Akehurst D, Howells W, McDonald-Maier K, *Kent model transformation language*, Model Transformations in Practice, 2005.

Aksit02 Aksit M, Kurtev I, Bezivin J, *Technological Spaces: an Initial Appraisal*, International, Federated Conf. (DOA, ODBASE, CoopIS), Industrial Track, Los Angeles, 2002.

Aleks05 Aleksić S, Luković I, Pavićević J, *Jedan generator SQL opisa šeme baze podataka*, XIII Konferencija Industrijski sistemi IS 2005, Herceg Novi, Srbija i Crna Gora, Zbornik radova, pp. 341-348, 2005.

Aleks06 Aleksić S, *Jedan SQL generator implementacionog opisa šeme baze podataka CASE alata IIS*Case*, Magistarski rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Novi Sad, 2006.

Aleks07 Aleksić S, Luković I, Mogin P, Govedarica M, *A Generator of SQL Schema Specifications*, Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, ISSN: 1820-0214, Vol. 4, No. 2, pp. 77-96, 2007.

Aleks07a Aleksić S, Luković I, Govedarica M, Mogin P, *A Generator of the SQL Specification of a Database Schema*, International Multiconference on Computer Science and Information Technology, October 15-17, Wisla, Poland, ISSN 1896-7094, Vol. 2, pp. 1113-1122, 2007.

Aleks13 Aleksić S, Ristić S, Luković I, Čeliković M, *A Design Specification and a Server Implementation of the Inverse Referential Integrity Constraints*, Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, ISSN: 1820-0214, Vol. 10, No.1, pp. 283-320, 2013.

Alhaj01 Alhajj R, Polat F, *Reengineering relational databases to object-oriented: Constructing the class hierarchy and migrating the data*, Eighth Working Conference on Reverse Engineering (*WCRE'01*), IEEE Computer Society: Silver Spring MD, pp. 335–344, 2001.

Alhaj03 Alhajj R, *Extracting the extended entity-relationship model from a legacy relational database*, Information Systems, Vol. 28, No. 6, pp. 597-618, 2003.

Ander94 Andersson M. *Extracting an entity relationship schema from a relational database through reverse engineering*, Thirteenth International Conference on Entity-relationalship Approach (Lecture Notes in Computer Science, Vol. 881), Springer, Berlin, pp. 403–419, 1994.

Assma06 Assmann, U., Zchaler and S., Wagner, G, Ontologies, *Meta-Models, and the Model-Driven Paradigm*, In: Calero, C., Ruiz, F., Piattini, M. (eds.) Ontologies for Software Engineering and Software Technology, 2006.

Atkin05 Atkinson C, Kuhne T, *A Generalized Notion of Platforms for Model-Driven Development*, In Model-Driven Software Development, ed. Sami Beydeda, Matthias Book and Volker Gruhn, Berlin Heidelberg: Springer, pp. 119−136, 2005.

ATL10 Eclipse Foundation, *ATL (ATLAS Transformation Language) Project, ATL/User Guide*, dostupno na: http://wiki.eclipse.org/ATL/User_Guide, 2010. (Septembar, 2013)

Atzeni06 Atzeni P, Cappelari P, Bernstein P.A, *Model Independent Schema and Data Translation*, EDBT Conference, Springer, LNCS 3896, pp. 368-385, 2006.

Atzeni07 Atzeni P, Cappelari P, Gainforme G, *MIDST: Model Independent Schema and Data Translation*, In Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data (Beijing, China, June 11 - 14, 2007), SIGMOD '07, ACM, New York, NY, pp. 1134-1136, 2007.

Atzeni07a Atzeni P, *Schema and Data Translation: A Personal Perspective*, Y. Ioannidis et al. (Eds.) ADBIS 2007, LNCS 4690, pp. 14-27, 2007.

Atzeni08 Atzeni P, Cappelari P, Torlone R, Bernstein P.A, Gainforme G, *Model-independent schema translation*, The VLDB Journal, DOI 10.1007/s00778-008-0105-2, 2008.

Banov10 Banović J, *Jedan pristup generisanju izvršnih softverskih specifikacija informacionog sistema*, Doktorski rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Novi Sad, Srbija, 2010.

Barbi10 Barbier G, Bruneliere H, Jouault F, Lennon Y, Madiot F, *Modisco, a model-driven platform to support real legacy modernization uses cases*, In: Information Systems Transformation: Architecture-Driven Modernization Case Studies. Elsevier Science, Amsterdam, 2010.

Beeri79 Beeri C, Bernstein P. A, *Computational Problems Related to the Design of Normal Form Relational Schemas*, ACM Transactions on Database Systems, Vol.4, No.1, March, pp. 30-59, 1979.

Berg99 Bergey J, Smith D, Weiderman N, Woods S, *Options Analysis for Reengineering (OAR): Issues and Conceptual Approach*, The Software Engineering Institute, Carnegie Mellon University, Technical Note CMU/SEI-99-TN-014, 1999.

Beziv01 Bezivin J, Gerb O, *Towards a Precise Definition of the OMG/MDA Framework*, in Proceedings of the 16th IEEE international conference on Automated software engineering, IEEE Computer Society, dostupno na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.6645&rep=rep1&type=pdf, 2001. (September, 2013)

Beziv03 Bezivin, J, Farcet, N, Jezequel J.M, Langlois B, Pollet D, *Reflective Model Driven Engineering.* In UML, volume 2863 of LNCS, pages 175-189, 2003.

Beziv03a Bezivin J, Dupe G, Jouault F, Pitette G, Rougui J.E, *First experiments with the ATL model transformation language: Transforming XSLT into XQuery*, dostupan na: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.129.6724, 2003. (September, 2013)

Beziv04 Bezivin J, *In Search of a Basic Principle for Model-Driven Engineering*, Novatica Journal, Special Issue, March-April 2004.

Blaha01 Blaha M, *A retrospective on industrial database reverse engineering projects—Part 1*, Proceedings 8th Working Conference on Reverse Engineering (WCRE'01), IEEE Computer Society: Stuttgart, Germany, pp. 136–147, 2001.

Blaha01a Blaha M, *A retrospective on industrial database reverse engineering projects—Part 2*, Proceedings 8th Working Conference on Reverse Engineering (WCRE'01), IEEE Computer Society: Stuttgart, Germany, pp. 147–156, 2001.

Booch04 Booch G, Brown A, Iyengar S, Rumbaugh J, Selic B, *An MDA Manifesto*, MDA Journal, May 2004.

Boron04 Boronat A, Perez J, Carsı J. A, Ramos I, *Two Experiences in Software Dynamics*, Journal of Universal Computer Science, Vol. 10, No. 4, 428-453, 2004.

Boron05 Boronat A, Carsı J.A, Ramos I, *Automatic reengineering in MDA using rewriting logic as transformation engine*, Ninth European Conference on Software Maintenance and Reengineering (CSMR'05). IEEE Computer Society: Manchester, U.K., pp. 228–231, 2005.

Cari04 Cariou E, Marvie R, Seinturier L, Duchien L, *OCL for the Specification of Model Transformation Contracts*, OCL and Model Driven Engineering, UML 2004 Conference Workshop, October 12, 2004, Lisbon, Portugal, pp. 69-83, 2004.

Casti09 del Castillo R.P, Rodríguez I.G, Caballero I, *PRECISO: A Reengineering Process and a Tool for Database Modernisation through Web Services*, *SAC'09*, March 8-12, 2009, Honolulu, Hawaii, U.S.A., ACM 978-1-60558-166-8/09/03, 2009.

Čelik11 Čeliković M, Luković I, Aleksić S, Ivančević V, *A MOF based meta-model of IIS*Case PIM Concepts*, In Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 833-840, Los Alamitos, CA: IEEE Computer Society Press, 2011.

Čelik12 Čeliković M, Luković I, Aleksić S, Ivancević V, *A MOF based Meta-Model and a Concrete DSL Syntax of IIS*Case PIM Concepts*, Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, DOI: 10.2298/CSIS120203034C, ISSN: 1820-0214, Vol. 9, No. 3, pp. 1075-1103, 2012.

Chan03 Chan K, Liang Z, Michail A, *Design recovery of interactive graphical application,*International Conference on Software Engineering, IEEE Computer Society: Silver Spring MD, pp. 114–125, 2003.

Chiang95 Chiang R.H, *A knowledge-based system for performing reverse engineering of relational database*, Decision Support Systems, Vol. 13, pp. 295-312, 1995.

Chikof90 Chikofsky EJ, Cross JH, *Reverse engineering and design recovery: A taxonomy*, *IEEE Software*, **7**(1):13–17, dostupno na: http://www.cs.cmu.edu/~aldrich/courses/654sp05/ReengineeringTaxonomy.pdf, 1990. (Septembar, 2013)

Csert02 Csertan G, Huszerl G, Majzik I, Pap Z, Pataricza A, Varro D, *VIATRA — Visual Automated Transformations for Formal Verification and Validation of UML Models*, In Proceedings of 17th IEEE International Conference on Automated Software Engineering (ASE'02), IEEE Computer Society, Los Alamitos, CA, USA, 2002.

CWM Common Warehouse Metamodel (CWM), dostupno na: http://www.omg.org/spec/CWM/, 2003. (Septembar, 2013)

Czarn03 Czarnecki K, Helsen S, *Classification of Model Transformation Approaches*, OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, dostupno na: http://citeseerx.ist.psu.edu/viewdoc/summary?doi =10.1.1.122.8124, 2003. (Septembar, 2013)

Czarne06 Czarnecki K, Helsen S, *Feature-based Survey of Model Transformation Approaches*, *IBM Systems J.*, 45(3), June 2006.

Date06 Date C.J, Darwen H, *Types and the Relational Model. The Third Manifesto*, 3rd ed. Addison Wesley, Reading, 2006.

Diaz11 Díaz O, Puente G, Luis J, Izquierdo C,·Molina J.G, *Harvesting models from web 2.0 databases*, Softw Syst Model, DOI 10.1007/s10270-011-0194-z, Springer-Verlag, 2011.

Diedr88 Diedrich I, Milton J, *New Methods and Fast Algorithms for Database Normalization*, ACM Transactions on Database Systems, Vol. 13, No. 3, pp. 339-365, Sept. 1988.

Duge06 Dugerdil P, *Architecture-based software reengineering*, Isnet62 Project, dostupno na: http://campus.hesge.ch/dugerdilp/_private/ISNet62.pdf, 2006. (Septembar, 2013)

Eass07 Eessaar, E, *Using Meta-modeling in order to Evaluate Data Models*, In Proceedings of the 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Corfu Island, Greece, February 16-19, 2007.

Elmas11 Elmasri R, Navathe B. S, *Database Systems: Models, Languages, Design and Application Programming, Sixth Edition*, Pearson Global Edition, ISBN 978-0-13-214498-8, 2011.

EMF13 Eclipse Modeling Framework, dostupno na: http://www.eclipse.org/modeling/emf/. (Septembar, 2013)

FavreJ05 Favre J.M, *Foundations of Model (Driven) (Reverse) Engineering: Models*, Dagstahl Seminar Proceedings 4101, 2005.

FavreL10 Favre L, *Model Driven Architecture for Reverse Engineering Technologies: strategic directions and system evolution*, Engineering Science Reference, 2010.

FavreL10a Favre L, Pereira C. T, Martinez L. I, *Foundations for MDA Case Tools*, Model Driven Architecture for Reverse Engineering Technologies: strategic directions and system evolution, Engineering Science Reference, pp. 242-252, 2010.

Fong06 Fong J, *Information Systems Reengineering and Integration*, Second Edition, Springer-Verlag, 2006.

Fowler99 Fowler M, *Refactoring: Improving the Design of Existing Programs*, Addison-Wesley, 1999.

Gronb09 Gronback R. C, *Eclipse modeling project : a domain-specific language (DSL) toolkit*, Pearson Education, Inc, ISBN 0-321-53407-7, 2009.

Guerr10 Guerra E, de Lara J, Kolovos D, Paige R, Marchi dos Santos O, *transML: A family of languages to model model transformations*, MODELS 2010, LNCS 6394, 2010.

Guzm07 de Guzman G.R.I, Polo M, Piattini M, *An ADM Approach to Reengineer Relational Database towards Web Services*, Proceeding WCRE '07 Proceedings of the 14th Working Conference on Reverse Engineering, pp. 90-99, IEEE Computer Society Washington, DC, USA, ISBN:0-7695-3034-6 doi>10.1109/WCRE.2007.9, 2007.

Haina06 Hainaut J, *The Transformational Approach to Database Engineering*, Generative and Transformational Software Engineering, Springer, LNCS, 2006.

Haina96 Hainaut JL, Henrard J, Hick JM, Roland D, Englebert V, *Database design recovery*, Eighth Conferences on Advance Information Systems Engineering, Springer, Berlin, pp. 463–480, 1996.

Hamm02 Hammer M, Schmalz M, O'Brien W, Shekar S, Haldevnekar N, *Knowledge Extraction in the SEEK Project Part I*, Tecnical Report TR-02-008, July 2002.

Henra07 Henrard J, Hick J-M, Thiran P, Hainaut J-L. *Strategies for data reengineering*, Ninth Working Conference on Reverse Engineering, IEEE Computer Society: Richmond VA, 2002.

Henra98 Henrard J, Englebert V, Hick JM, Roland D, Hainaut JL, *Program understanding in database reverse engineering*, Database and Expert Systems Applications (DEXA), pp. 70–79, 1998.

Honey83 Honeyman P, Scoire E, *New Characterization of Independence*, Proceedings of ACM SIGMOD, San Jose, California, USA, pp. 92-96, 1983.

Jesus98 de Jesus P. L, Sousa P, *Selection of reverse engineering methods for relational databases*, Third European Conference on Software Maintenance, Nesi, Verhoef: Los Alamitos CA, pp. 194–197, 1998.

Jouau06 Jouault F, Kurtev I, *Transforming Models with ATL***,** MoDELS 2005 Workshops, Springer-Verlag Berlin Heidelberg, LNCS 3844, pp. 128–138, 2006.

Jouau08 Jouault F, Allilaire F, Bezivin J, Kurtev I, *ATL: A model transformation tool*, *Science of Computer Programming*, dostupno na: www.elsevier.com/locate/scico, DOI:10.1016/j.scico.2007.08.002, 2008. (Septembar, 2013)

KDM12 ISO/IEC 19506:2012(E), Information technology - Architecture-Driven Modernization (ADM): Knowledge Discovery Meta-Model (KDM), 2012.

Kermeta Kermeta, dostupno na: http://www.kermeta.org, 2010. (Septembar, 2013)

Kernel ATLAS group, LINA & INRIA, *KM3: Kernel MetaMetaModel*, version 0.3, dostupno na: http://www.eclipse.org/gmt/am3/km3/doc/ KernelMetaMetaModel%5Bv00.06%5D.pdf, 2005. (Septembar, 2013)

Kleppe03 Kleppe A, Warmer J, and Bast W, *MDA Explained: The Model-Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.

Kolah11 Kolahdouz-Rahimi S, Lano K, *A Model-based Development Approach for Model Transforamtion*, Fundamentals of Software Engineering-4th IPM International Conference, FSEN 2011, Tehran, Iran, pp. 48-63, 2011.

Kolov08 Kolovos D, Paige R, Polack F, *The Epsilon Transformation Language*, in ICMT 2008, Springer-Verlag, LNCS Vol. 5063, pp. 46–60, 2008.

Kurtev05 Kurtev I, *Adaptability of Model Transformations*, PhD Thesis, University of Twente, 2005.

Kuster05 Küster J. M, Ryndina K, Hauser R, *A Systematic Approach to Designing Model Transformations*, Report RZ 3621, IBM, Zurich, July 2005.

Kuster04 Küster J. M, Sendall S, Wahle M, *Comparing Two Model Transformation Approaches*, Proc. Workshop on OCL and Midel Drive Engineering, dostupno na: http://citeseerx.ist.psu.edu/viewdoc/summary?doi= 10.1.1.103.3078, 2004. (Septembar, 2013)

Lano11 Lano K, Kolahdouz-Rahimi S, *Model-driven development of model transformation*, Theory and Practice of Model Transformations, 47-61, 2011.

Lano12 Lano K, Kolahdouz-Rahimi S, and Poernomo I, *Comparative Evaluation of Model Transformation Specification Approaches*, International Journal of Software and Informatics, Vol. 6, Issue 2, ISSN 1673-7288, 2012.

Lano13 Lano K, Kolahdouz-Rahimi S, *Constraint-based specification of model transformations*, Journal of Systems and Software, Volume 86, Issue 2, Pages 412–436, 2013.

Lara02 de Lara J, Vangheluwe H, *AToM3: A tool for multi-formalism and meta-modelling*, In R.-D. Kutsche and H. Weber, editors, *FASE*, volume 2306 of *LNCS*, pages 174–188. Springer, 2002.

Lukov02 Luković I, Govedarica M, Mogin P, Ristić S, *The Structure of A Subschema and Its XML Specification*, Journal of Information and Organizational Sciences (JIOS), Varazdin, Croatia, Vol. 26, No. 1-2, pp. 69-85, 2002.

Lukov03 Luković I, Ristić S, Mogin P, *A methodology of a database schema design using the subschemas*, In Proceedings of IEEE International Conference on Computational Cybernetics (in CD ROM). Budapest, Hungary: Budapest Polytechnic, 2003.

Lukov06 Luković I, Ristić S, Mogin P, Pavićević J, *Database Schema Integration Process – A Methodology and Aspects of Its Applying*, Novi Sad Journal of Mathematics (Formerly Review of Research, Faculty of Science, Mathematic Series), Serbia, ISSN: 1450-5444, Vol. 36, No. 1, pp 115-150, 2006.

Lukov07 Luković I, Mogin P, Pavićević J, Ristić S, *An Approach to Developing Complex Database Schemas Using Form Types*, Software: Practice and Experience, John Wiley & Sons Inc, Hoboken, USA, ISSN: 0038-0644, Published Online, May 29, DOI: 10.1002/spe.820, Vol. 37, No. 15, pp. 1621-1656, 2007.

Lukov08 Luković I, Ristić S, Aleksić S, Popović A, *An application of the MDSE principles in IIS*Case*, In Proceedings of the 3rd Workshop on Model Driven Software Engineering, pp. 53-62, Berlin, Germany: TFH, University of Applied Sciences Berlin, 2008.

Lukov09 Lukovic I, "*From the Synthesis Algorithm to the Model Driven Transformations in Database Design*", 10th International Scientific Conference on Informatics (Informatics 2009), November 23-25, 2009, Herlany, Slovakia, Proceedings, Slovak Society for Applied Cybernetics and Informatics and Technical University of Košice - Faculty of Electrical Engineering and Informatics, ISBN 978-80-8086-126-1, pp. 9-18, Invited Talk, 2009.

Lukov10 Luković I, Ristić S, Aleksić S, Banović J, Popović A, *A Chain of Model Transformations in IIS*Case*, Scripta Scientiarum Naturalium, University of Montenegro, Faculty of Science, Podgorica, Montenegro, ISSN: 1880-8356, Vol. 1, No. 1, pp. 59-76, 2010.

Lukov10a Luković I, Popović A, Mostić J, Ristić S, *A Tool for Modeling Form Type Check Constraints and Complex Functionalities of Business Applications*, Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia and Montenegro, ISSN: 1820-0214, Vol. 7, No. 2, pp. 359-385, 2010.

Lukov13 Luković I, Ivančević V, Čeliković M, Aleksić S, *DSLs in Action with Model Based Approaches to Information System Development*, Formal and Practical Aspects of Domain-Specific Languages: Recent Developments, IGI Global, USA, ISBN: 978-1-4666-2092-6, DOI: 10.4018/978-1-4666-2092-6, pp. 502-532, 2013.

Lukov93 Luković I, *Automatizovano generisanje podšeme relacione baze podataka putem formi*, magistarski rad, Univerzitet u Beogradu, Elektrotehnički fakultet, Smer Informatika, Beograd, Mentor: prof. dr Pavle Mogin, 1993.

Lurd99 Lurdes P. J, Sousa P, *Selection of Reverse Engineering Methods for Relational Databases*, dostupno na: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9316, 1999. (Septembar, 2013)

MDA03 Object Management Group, *MDA Guide*, version 1.0.1, volume 1, document omg/03-06-01, dostupno na: http://www.omg.org/cgi-bin/doc?omg/03-06-01, 2003. (Septembar, 2013)

Mellor04 Mellor S. J, Scott K., Uhl A, Weise D, *MDA Distilled: Principles of Model-Driven Architecture*, Addison Wesley, 2004.

Mens05 Mens T, Czarnecki K, and Van Gorp P, *A taxonomy of model transformations*, In Jean Bezivin and Reiko Heckel, editors, Language Engineering for Model-Driven Software Development, number 04101 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany, 2005.

MOF13 Meta-Object Facility, dostupno na: http://www.omg.org/mof/. (Septembar, 2013)

Mogin00 Mogin P, Lukovic I, Govedarica M, *Extended Referential Integrity*, Novi Sad Journal of Mathematics (Formerly Review of Research, Faculty of Science and Mathematics, Mathematics Series), Novi Sad, Yugoslavia, Vol. 30, No. 3, pp. 111-122, 2000.

Mogin04 Mogin P, Luković I, Govedarica M, *Principi projektovanja baza podataka*, II izdanje, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Novi Sad, 2004.

Mosti09 Mostić J, *Specifikacija regularnih izraza i funkcija u alatu IIS*Case*, magistarski rad, Univerzitet u Novom Sadu, Fakultet Tehničkih nauka, Novi Sad, 2009.

Mull00 Müller H. A, Jahnke J. H, Smith D. B, Storey M.-A, Tilley S. R, Wong K, *Reverse engineering: A roadmap*, Proceedings of the Conference on The Future of Software Engineering. Limerick, Ireland: ACM, 2000.

Neško07 Nešković S, Vučković M, Turajlić N, *Transformacija XML šeme u relacioni model zasnovana na OMG MDA pristupu i apstraktnom nivou*, Infoteh-Jahorina Vol. 6, Ref. E-II-3, p. 343-347, 2007.

Obren12 Obrenović N, Aleksić S, Popović A, Luković I, *Transformations of Check Constraint PIM Specifications*, Computer and Informatics (CAI), Slovak Academy of Sciences, Institute of Informatics, Bratislava, Slovakia, ISSN: 1335-9150, Vol. 31, No. 5, pp. 1045-1079, 2012.

OCL12 Object Management Group, *Object Constraint Language*, Version 2.3.1, document formal/2012-01-01, dostupno na: http://www.omg.org/cgi-bin/doc?formal/06-05-01.pdf, 2012. (Septembar, 2013)

Pavić05 Pavićević J, *Razvoj CASE alata za automatizovano projektovanje i integraciju šema baza podataka*, Magistarski rad, Univerzitet Crne Gore, Prirodno-matematički fakultet, Podgorica, 2005.

Pavić06 Pavićević J., Luković I., Mogin P., Ristić S., *Generisanje relacione šeme baze podataka zasnovano na konceptu tipa forme*, Naučno-stručni simpozijum Informacione tehnologije INFOTEH 2006, 22-24. 3. 2006, Jahorina, Bosna i Hercegovina, Zbornik radova, pp. 420-424, 2006.

Perez02 Perez J, Ramos I, Anaya V, Cubel J, Domınguez F, Boronat A, Carsı J. A, *Data reverse engineering of legacy databases to object oriented conceptual schemas*, Electronic Notes in Theoretical Computer Science, 74(4):1–13, 2002.

Perez09 Perez-Castillo R, Rodriguez I.G, Caballero I, *PRECISO: a reengineering process and a tool for database modernisation through web services*, SAC '09, Proceedings of the 2009 ACM symposium on Applied Computing, pp. 2126-2133, 2009.

Perez10 Perez-Castillo R, de Guzman I, Piattini M, *Implementing Buisiness Process Recovery Patterns through QVT Transformations*, Theory and Practice of Model Transformations, Springer, LNCS 6142, Third International Conference, ICMT 2010, Malaga, Spain, June 28-July 2, 2010, Proceedings, pp. 168-183, 2010.

Polo07 Polo M, Garcıa-Rodrıguez I, Piattini M, *An MDA-based approach for database re-engineering*, Wiley InterScience, dostupno na: www.interscience.wiley.com, DOI: 10.1002/smr.353, 2007.

Popov06 Popović A, *Specifikacija vizuelnih atributa i struktura poslovnih aplikacija u alatu IIS*Case*, magistarski rad, Univerzitet u Novom Sadu, Fakultet Tehničkih nauka, Novi Sad, 2006.

Popov08 Popović A, Luković I, Ristić S, *A Specification of the Structures of Business Applications in the IIS*Case Tool*, Info M – Journal of Information Technology and Multimedia Systems, Belgrade, Serbia, ISSN: 1451-4397, No. 25, pp. 17-24, 2008.

Popov08a Popović A, *Specifikacija vizuelnih atributa i struktura poslovnih aplikacija u alatu IIS*Case*, Magistarski rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, 2008.

Popov13 Popović A, *Jedan pristup specificiranju izvršnih modela aplikacija informacionog sistema*, Doktorski rad, Univerzitet Crne Gore, Prirodno-matematički fakultet, Podgorica, Crna Gora, 2013.

Preme94 Premerlani W. J, Blaha M. R, *An Approach for Reverse Engineering of Relational Databases*, Communications of the ACM, 37(5):42–49, 1994.

QVT09 OMG, *Query/View/Transformation Specification*, dostupno na: http://www.omg.org/spec/QVT/1.0/, 2009. (Septembar, 2013)

Reus06 Reus T, Geers H, van Deursen A, *Harvesting software systems for MDA-Based reengineering*, Springer-Verlag Berlin / Heidelberg, Proceeding ECMDA-FA'06 Proceedings of the Second European conference on Model Driven Architecture: foundations and Applications, ISBN:3-540-35909-5 978-3-540-35909-8, doi: 10.1007/11787044_17, pp. 213-225, 2006.

Reus06a Reus T, *Harvesting existing software systems for MDA-based reengineering*, master thesis, 2006.

Ristic03 Ristić S, *Istraživanje problema konsolidacije podšema baza podataka*, doktorska disertacija, Univerzitet u Novom Sadu, Ekonomski fakultet Subotica, 2003.

Ristic03a Ristić S, Mogin P, Luković I, *Specifying Database Updates Using A Subschema*, VII IEEE International Conference on Intelligent Engineering Systems, Assiut – Luxor, Egypt, March 4-6, Proceedings, pp. 203-212, 2003.

Ristic06 Ristić S, Luković I, Pavićević J, Mogin P, *Resolving Database Constraint Collisions Supported by IIS*Case Tool*, XVII International Conference on Information and Intelligent Systems, Varazdin, Croatia, September 20-22, Proceedings, pp. 43-52, 2006.

Ristic07 Ristić S, Luković I, Pavićević J, Mogin P, *Resolving Database Constraint Collisions Using IIS*Case Tool*, Journal of Information and Organizational Sciences (JIOS), University of Zagreb, Faculty of Organization and Informatics, Varaždin, Croatia, ISSN: 1846-3312, Vol. 31, No. 1, pp. 187-206, 2007.

Ristic13 Ristić S, Aleksić S, Čeliković, M, Lukovic I, *An EMF Ecore based relational dB schema meta-model*, In Proceedings of the 6th International Conference on Information Technology ICIT 2013, Amman, Jordan, ISSN 2306-6105, ISBN 978-9957-8583-1-5, Proceedings on CD, 2013.

Ristic13a Ristić S, Aleksić S, Čeliković M, Luković I, 2013, *Meta-modeling of inclusion dependency constraints*, In /Proceedings of the 6th Balkan Conference in Informatics/ (BCI '13). ACM, New York, NY, USA, 114-121, DOI=10.1145/2490257.2490265, dostupno na: http://doi.acm.org/10.1145/2490257.2490265. (Septembar, 2013)

Ruscio07 129. Di Ruscio D, *Specification of Model Transformation and Weaving in Model Driven Engineering*, PhD Thesis, University of di L'Aquila, 2007.

Senda03 Sendall, S, Kozaczynski W, *Model Transformation–the Heart and Soul of Model-Driven Software Development*, *IEEE Software*, *20*(5), pp. 42-45, 2003.

Singh09 Singh Y, Sood M, *Models and Transformations in MDA*, First International Conference on Computational Intelligence, Communication Systems and Networks, pp. 253-258, 2009.

SQL:1999 ISO/IEC 9075-(1,2):1999 (ANSI SQL:1999), American National Standards Institute, 1999-2002.

SQL:2003 INCITS/ISO/IEC 9075-1-2003 (ANSI SQL:2003), American National Standards Institute, 2003.

SQL:2011 ISO/IEC 9075:2011 (ANSI SQL:2011), American National Standards Institute, 2011.

Stahl06 Stahl T, Völter M, *Model-Driven Software Development : technology, engineering, management*, John Wiley & Sons Inc, Hoboken, USA, ISBN: 0-470-02570-0, 2006.

Taent03 Taentzer G, *AGG: A graph transformation environment for modeling and validation of Software*, In J. L. Pfaltz, M. Nagl, and B. B¨ohlen, editors, *AGTIVE*, volume 3062 of *LNCS*, pp. 446–453, Springer, 2003.

Tratt05 Tratt, L, *Model transformations and tool integration*, Journal of Software and Systems Modeling, 4(2), pp. 112-122, 2005.

Ulrich10 Ulrich W.M, Newcomb P.H, *Information Systems Transformation: ADM Case Studies*, Morgan Kaufmann, USA, 2010.

UML11 Object Management Group, *OMG Unified Modeling Language (UML)*, Version 2.4.1, document formal/2011-08-05, dostupno na: http://www.omg.org/spec/UML/2.4.1/, 2011. (September, 2013)

Vara09 Vara J, Vela B, Bollati V.A, Marcos E, *Supporting model-driven development of object-relational database schemas: a case study*, in: R. Paige (Ed.), Theory and Practice of Model Transformations, Heidelberg, Springer Berlin, pp. 181–196, 2009.

Vara09a Vara J, *M2DAT: a Technical Solution for Model-Driven Development of Web Information Systems*, PhD Thesis, University of Rey Juan Carlos, 2009.

Vara12 Vara J.M, Marcos E, *A framework for model-driven evelopment of information systems: Technical decisions and lessons learned*, The Journal of Systems and Software 85, pp. 2368– 2384, 2012.

XMI05 ISO/IEC 19503:2005, document formal/05-05-06, dostupno na: http://www.omg.org/spec/XMI/ISO/19503/PDF/. (September, 2013)

XMI11 Object Management Group, OMG MOF 2.0 XMI Mapping Specification, Version 2.4.1, document formal/05-09-01, dostupno na: http://www.omg.org/spec/XMI/2.4.1, August 2013. (September, 2013)

XML06 *XML meta-model*, netBeans.org, dostupno na: http://mdr.netbeans.org/mdrxml.html, 2006. (September, 2013)

XML13 *Extensible Markup Language (XML) 2.0*, World Wide Web Consortium (W3C), dostupno na: http://www.w3.org/TR/xml-c14n2/. (Avgust 2013)

Prilog A – XML šeme modula *XML Transformer*

U ovom dodatku prikazane su XML šeme primenjene u modulu *XML Transformer* za kreiranje XML dokumenata potrebnih za razmenu podataka sa modulom *M2M Transformer*. Na slici A.1 prikazana je XML šema na osnovu koje se kreira XML dokument u koji se smeštaju podaci relacione šeme baze podataka koja je predmet reinženjeriga.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.uns.ac.rs/dbXMLSchema"
xmlns="http://www.uns.ac.rs/dbXMLSchema" elementFormDefault="qualified">
  <xsd:simpleType name="DeferrableAct">
      <xsd:restriction base="xsd:string">
      <xsd:enumeration value="DEFFERABLE"/>
      <xsd:enumeration value="NOT_DEFFERABLE"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="DeferredAct">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="INITIALLY_IMMEDIATE"/>
      <xsd:enumeration value="INITIALLY_DEFERRED"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="Action">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="RESTRICT"/>
      <xsd:enumeration value="NO_ACTION"/>
      <xsd:enumeration value="SET_NULL"/>
      <xsd:enumeration value="SET_DEFAULT"/>
      <xsd:enumeration value="CASCADE"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="ReferencingType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="DEFAULT"/>
      <xsd:enumeration value="PARTIAL"/>
      <xsd:enumeration value="FULL"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="RSUBP">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element ref="Database" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="SystemDataType" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Database">
    <xsd:complexType>
      <xsd:sequence>
```

```
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element ref="Table" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="UserDefinedDataType" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="SystemDataType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="PredefinedDecPlaces" type="xsd:integer" maxOccurs="1"
                                                                   minOccurs="0"/>
        <xsd:element name="PredefinedLength" type="xsd:integer" maxOccurs="1"
                                                                   minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="UserDefinedDataType">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Precision" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="Length" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="DefaultValue" type="xsd:string" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="DataTypeName" type="xsd:string" maxOccurs="1" minOccurs="1"/>
       </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Table">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element ref="Column" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="PrimaryKeyCon" minOccurs="1" maxOccurs="1"/>
        <xsd:element ref="UniqueCon" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="CheckCon" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="ForeignKey" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Column">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Nullable" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Default" type="xsd:string" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="DataType" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Precision" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="Length" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="PrimaryKeyCon">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="PKColumn" type="xsd:string" maxOccurs="unbounded"
                                                                   minOccurs="1"/>
        <xsd:element name="Deferrable" type="DeferrableAct" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="Deferred" type="DeferredAct" maxOccurs="1" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Slika A.1. XML šema za serijalizaciju logičke šeme relacione baze podataka

Na slici A.2 prikazana je XML šema na osnovu koje se kreira XML dokument koji predstavlja izlaz iz modula *M2M Transformer*, u kojem se nalazi konceptualni model šeme baze podataka opisan PIM konceptima alata IIS*Case.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.uns.ac.rs/dbXMLSchema"
xmlns="http://www.uns.ac.rs/dbXMLSchema" elementFormDefault="qualified">
  <xsd:simpleType name="NoOfOcurType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="OneOrMany"/>
      <xsd:enumeration value="NoneOrMany"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="ISApplicationModel">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element ref="Fundamentals" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="ApplicationSystem" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Fundamentals">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="PrimitiveDomain" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="UserDefinedDomainFromPrimitiveDomain" minOccurs="0"
                                                          maxOccurs="unbounded"/>
        <xsd:element ref="UserDefinedDomainFromUserDefinedDomain" minOccurs="0"
                                                          maxOccurs="unbounded"/>
        <xsd:element ref="Attribute" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="PrimitiveDomain">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PrimitiveDomainID" type="xsd:ID" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="DefaultLength" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="DefaultDecimalPlaces" type="xsd:integer" maxOccurs="1"
                                                                  minOccurs="0"/>
        <xsd:element name="Description" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="DefaultValue" type="xsd:string" maxOccurs="1" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="UserDefinedDomainFromPrimitiveDomain">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="UserDefinedDomainFromPrimitiveDomainID" type="xsd:ID"
                                                          maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="LengthValue" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="DecimalPlaces" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="Description" type="xsd:string" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="DefaultValue" type="xsd:string" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="CheckCondition" type="xsd:string" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="InheritsPrimitiveDomain" type="xsd:IDREF" maxOccurs="1"
                                                                  minOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

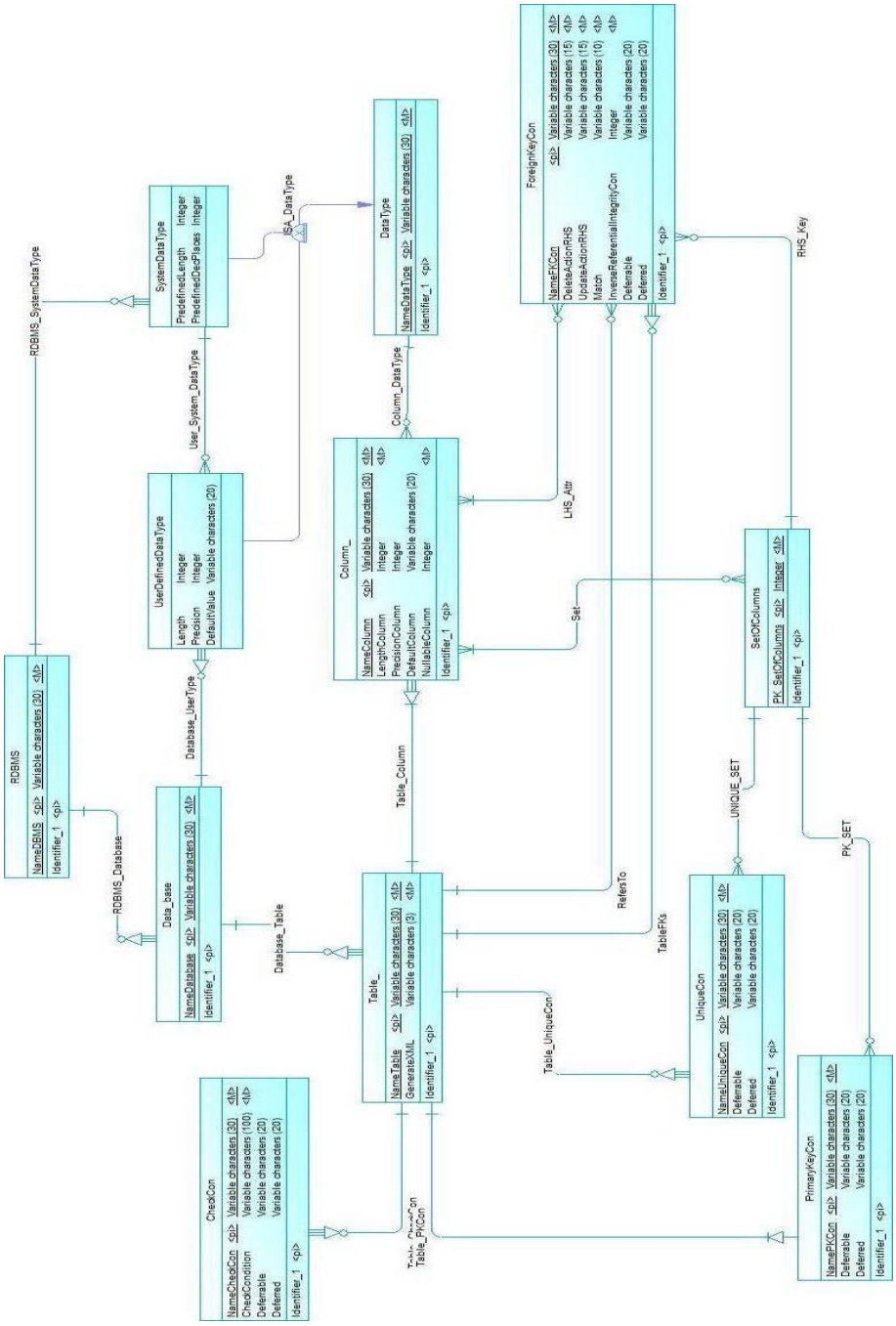
```
<xsd:element name="UserDefinedDomainFromUserDefinedDomain">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="UserDefinedDomainFromUserDefinedDomainID" type="xsd:ID"
                                                    maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Description" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="DefaultValue" type="xsd:string" maxOccurs="1" minOccurs="0"/>
      <xsd:element name="CheckCondition" type="xsd:string" maxOccurs="1" minOccurs="0"/>
      <xsd:element name="InheritsUserDefinedDomain" type="xsd:IDREF" maxOccurs="1"
                                                                  minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Attribute">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="AttributeID" type="xsd:ID" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Description" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="DefaultValue" type="xsd:string" maxOccurs="1" minOccurs="0"/>
      <xsd:element name="CheckCondition" type="xsd:string" maxOccurs="1" minOccurs="0"/>
      <xsd:element name="UserDefinedDomain" type="xsd:IDREF" maxOccurs="1"
                                                                  minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ApplicationSystem">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element ref="FormTypeProgram" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="FormTypeProgram">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Title" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Frequency" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
      <xsd:element name="ResponseTime" type="xsd:integer" maxOccurs="1" minOccurs="0"/>
      <xsd:element name="ConsideredINDBSchDesign" type="xsd:boolean" maxOccurs="1"
                                                                  minOccurs="1"/>
      <xsd:element ref="ComponentTypeRoot" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ComponentTypeRoot">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Title" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Query" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Insert" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Delete" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Update" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
      <xsd:element ref="ComponentTypeChild" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ComponentTypeChild">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
      <xsd:element name="Title" type="xsd:string" maxOccurs="1" minOccurs="1"/>
```

```
        <xsd:element name="Query" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Insert" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Delete" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Update" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="NoOfOcurrences" type="NoOfOcurType" maxOccurs="1"
                                                                minOccurs="1"/>
        <xsd:element ref="ComponentTypeAttribute" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="ComponentTypeKey" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="ComponentTypeUnique" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="ComponentTypeCheck" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="ComponentTypeChild" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="ComponentTypeAttribute">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ComponentTypeAttributeID" type="xsd:ID" maxOccurs="1"
                                                                minOccurs="1"/>
        <xsd:element name="Title" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Mandatory" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="DefaultValue" type="xsd:string" maxOccurs="1" minOccurs="0"/>
        <xsd:element name="ComponentTypeAttributeName" type="xsd:IDREF" maxOccurs="1"
                                                                minOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="ComponentTypeKey">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="Global" type="xsd:boolean" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="ComponentTypeKeyAttributes" type="xsd:IDREFS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="ComponentTypeUnique">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="ComponentTypeUniqueAttributes" type="xsd:IDREFS"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="ComponentTypeCheck">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
        <xsd:element name="CheckCondition" type="xsd:string" maxOccurs="1" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

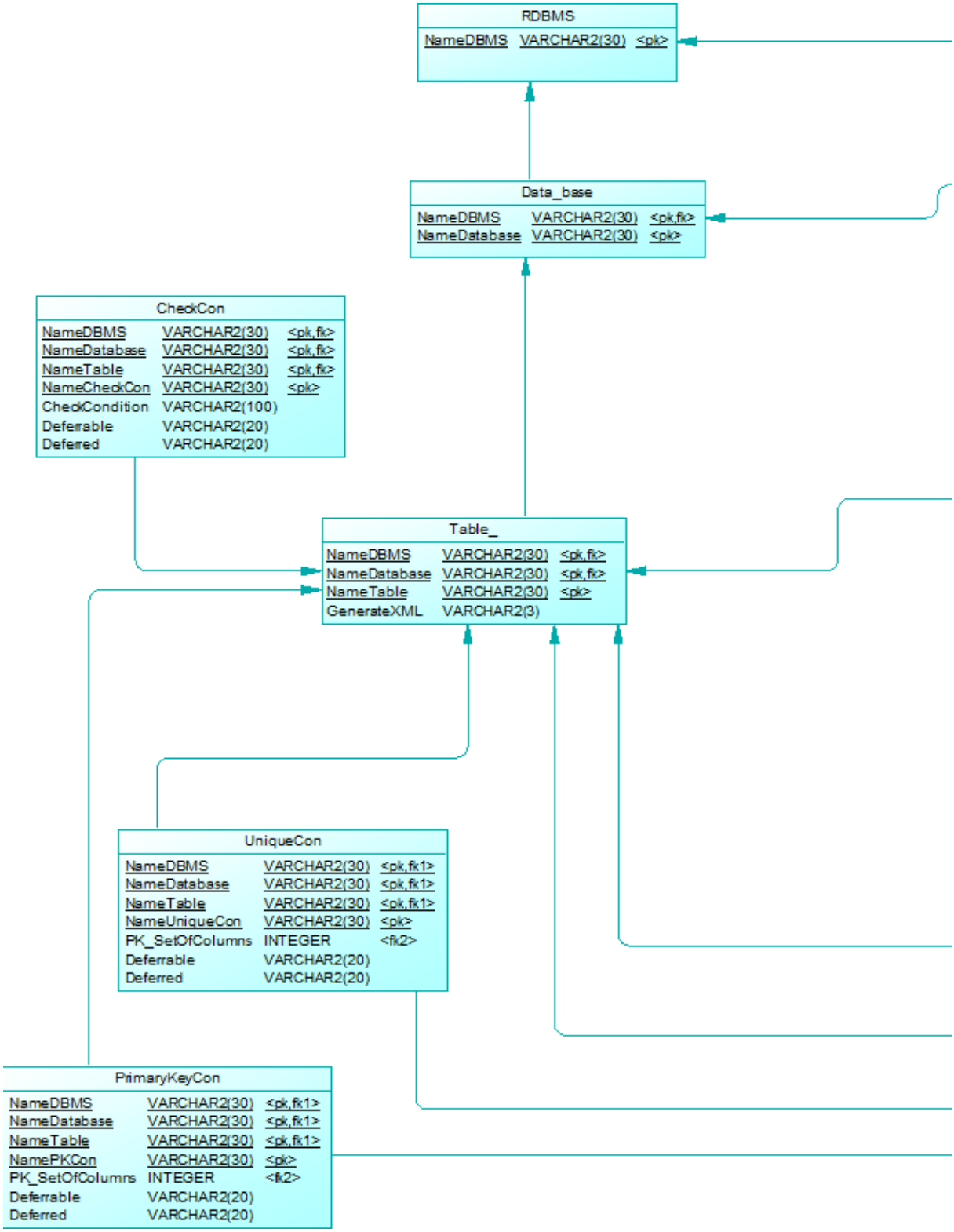
Slika A.2. XML šema za serijalizaciju konceptualne šeme

Prilog B – Specifikacija repozitorijuma IIS*Ree

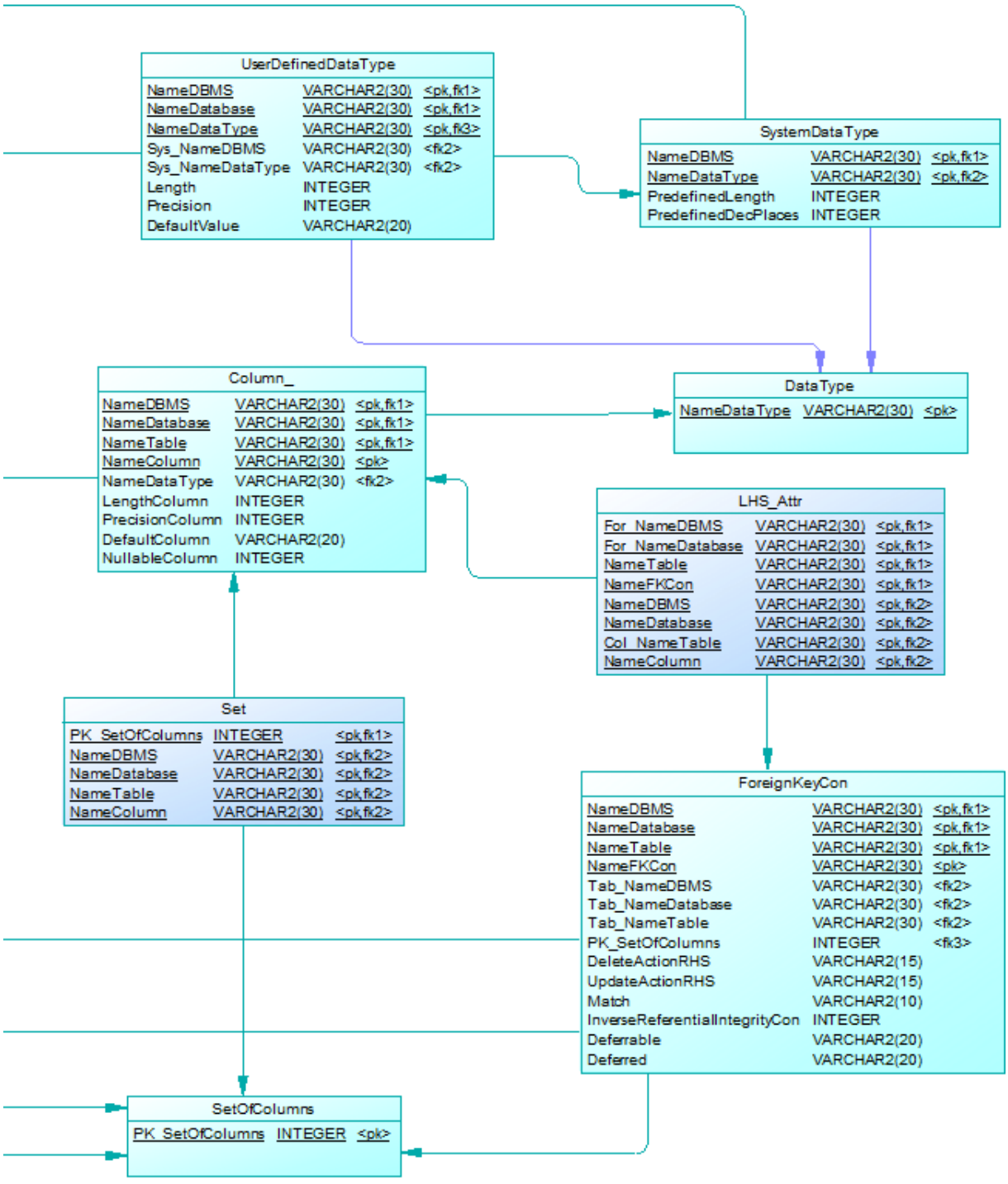
Na slici B.1 prikazan je izvod iz specifikacije repozitorijuma koji se odnosi na IIS*Ree.



Slika B.1. Konceptualna specifikacija repozitorijuma koja se odnosi na IIS*Ree



Slika B.2.a) Specifikacija repozitorijuma koja se odnosi na IIS*Ree



Slika B.2.b) Specifikacija repozitorijuma koja se odnosi na IIS*Ree

Tabela RDBMS

RDBMS		
NameDBMS		Naziv RSUBP-a;
Ključevi		
1.	[NameDBMS]	PK

Tabela Data_base

DATA_BASE		
NameDBMS		Naziv RSUBP-a;
NameDatabase		Naziv šeme baze podataka;
Ključevi		
1.	[NameDBMS+NameDatabase]	PK
Međurelaciona ograničenja		
DATA_BASE[NameDBMS] ⊆ RDBMS[NameDBMS]		

Tabela DataType

DATATYPE		
NameDataType		Naziv tipa podatka;
Ključevi		
1.	[NameType]	PK

Tabela Table_

TABLE_		
NameDBMS		Naziv RSUBP-a;
NameDatabase		Naziv šeme baze podataka;
NameTable		Naziv tabele;
GenerateXML		Indikator za generisanje informacija u XML fajl;
Ključevi		
1.	[NameDBMS+NameDatabase+NameTable]	PK
Međurelaciona ograničenja		
TABLE_[NameDBMS,NameDatabase]⊆ DATA_BASE[NameDBMS, NameDatabase]		

Tabela Column_

COLUMN_		
NameDBMS		Naziv RSUBP-a;
NameDatabase		Naziv šeme baze podataka;
NameTable		Naziv tabele;
NameColumn		Naziv kolone;
NameDataType		Naziv tipa podatka;
LengthColumn		Maksimalna dužina;
PrecisionColumn		Preciznost;
DefaultColumn		Predefinisana vrijednost;
NullableColumn		Dozvola nula vrijednosti;
Ključevi		
1.	[NameDBMS+NameDatabase+NameTable+NameColumn]	PK
Međurelaciona ograničenja		
COLUMN_[NameDBMS, NameDatabase, NameTable]⊆ TABLE_[NameDBMS, NameDatabase, NameTable]		
COLUMN_[NameDataType] ⊂DATATYPE[NameDataType]		

Tabela SystemDataType

SYSTEMDATATYPE		
NameDBMS		Naziv RSUBP-a;
NameDataType		Naziv tipa podatka;
PredefinedLength		Dužina tipa podatka;
PredefinedDecPlaces		Broj cifara iza decimalne tačke;
Ključevi		
1.	[NameDBMS+NameDataType]	PK
Međurelaciona ograničenja		
SYSTEMDATATYPE[NameDBMS] ⊂ RDBMS[NameDBMS]		
SYSTEMDATATYPE[NamaDataType] ⊂DATATYPE[NameDataType]		

Tabela UserDefinedDataType

USERDEFINDEDDATATYPE	
NameDBMS	Naziv RSUBP-a;
NameDatabase	Naziv šeme baze podataka;
NameDataType	Naziv tipa podatka;
Sys_NameDBMS	Naziv RSUBP-a sistemskog tipa;
Sys_NameDataType	Naziv sistemskog tipa podatka;
Length	Dužina tipa podatka;
Precision	Preciznost;
DefaultValue	Predefinisana vrijednost;
Ključevi	
1. [NameDBMS+NameDatabase+NameDataType]	PK
Međurelaciona ograničenja	
USERDEFINEDDATATYPE[NameDBMS, NameDatabase] ⊆ DATA_BASE[NameDBMS, NameDatabase]	
USERDEFINEDDATATYPE [NamaDataType] ⊆DATATYPE[NameDataType]	
USERDEFINEDDATATYPE[Sys_NameDBMS, Sys_NameDataType] ⊆ SYSTEMDATATYPE[NameDBMS, NameDataType]	

Tabela CheckCon

CHECKCON	
NameDBMS	Naziv RSUBP-a;
NameDatabase	Naziv šeme baze podataka;
NameTable	Naziv tabele;
NameCheckCon	Naziv ograničenja torke;
CheckCondition	Logički uslov;
Defferable	Specifikacija trenutka provjere ograničenja (DEFFERABLE, NOT_DEFFERABLE);
Deffered	Specifikacija trenutka provjere ograničenja (INITIALLY_IMMEDIATE, INITIALLY_DEFFERED);
Ključevi	

1.	[NameDBMS+NameDatabase+NameTable+NameCheckCon]	PK
Međurelaciona ograničenja		
CHECKCON[NameDBMS, NameDatabase,NameTable] ⊆ TABLE_[NameDBMS, NameDatabase,NameTable]		

Tabela SetOfColumns

SETOFCOLUMNS		
PK_SetOfColumns		Vještački ključ skupa kolona;
NameOfConstraint		Ime ograničenja za koje je vezan skup kolona;
Ključevi		
1.	[PK_SetOfColumns]	PK

Tabela Set_

SET_		
PK_SetOfColumns	Skup kolona ograničenja PK-a ili ograničenja jedinstvenosti;	
NameDBMS	Naziv RSUBP-a;	
NameDatabase	Naziv šeme baze podataka;	
NameTable	Naziv tabele;	
NameColumn	Naziv kolone;	
Ključevi		
1.	[PK_SetOfColumns+NameDBMS+NameDatabase+ NameTable+NameColumn]	PK
Međurelaciona ograničenja		
SET_[PK_SetOfColumns]⊆ SETOFCOLUMNS[PK_SetOfColumns]		
SET_[NameDBMS,NameDatabase,NameTable,NameColumn] ⊆ COLUMN_[NameDBMS,NameDatabase,NameTable,NameColumn]		

Tabela UniqueCon

UNIQUECON	
NameDBMS	Naziv RSUBP-a;
NameDatabase	Naziv šeme baze podataka;
NameTable	Naziv tabele;

NameUniqueCon	Naziv ograničenja jedinstvenosti;
PK_SetOfColumns	Skup kolona koje pripadaju ograničenju;
Defferable	Specifikacija trenutka provjere ograničenja (DEFFERABLE, NOT_DEFFERABLE);
Deffered	Specifikacija trenutka provjere ograničenja (INITIALLY_IMMEDIATE, INITIALLY_DEFFERED);
Ključevi	
1.	[NameDBMS+NameDatabase+NameTable+NamUniqueCon] PK
Međurelaciona ograničenja	
UNIQUECON[NameDBMS, NameDatabase,NameTable] ⊆ TABLE_[NameDBMS, NameDatabase,NameTable]	
UNIQUECON[PK_SetOfColumns]⊆ SETOFCOLUMNS[PK_SetOfColumns]	

Tabela PrimaryKeyCon

PRIMARYKEYCON	
NameDBMS	Naziv RSUBP-a;
NameDatabase	Naziv šeme baze podataka;
NameTable	Naziv tabele;
NamePKCon	Naziv ograničenja primarnog ključa;
PK_SetOFColumn	Skup kolona koje pripadaju ograničenju;
Defferable	Specifikacija trenutka provjere ograničenja (DEFFERABLE, NOT_DEFFERABLE);
Deffered	Specifikacija trenutka provjere ograničenja (INITIALLY_IMMEDIATE, INITIALLY_DEFFERED);
Ključevi	
1.	[NameDBMS+NameDatabase+NameTable+NamePKCon] PK
Međurelaciona ograničenja	
PRIMARYKEYCON[NameDBMS,NameDatabase,NameTable] ⊆ TABLE_[NameDBMS,NameDatabase,NameTable]	
PRIMARYKEYCON[PK_SetOfColumns]⊆ SETOFCOLUMNS[PK_SetOfColumns]	

Tabela LHS_Attr

LHS_ATTR		
For_NameDBMS	Naziv RSUBP-a za ograničenje stranog ključa;	
For_NameDatabase	Naziv šeme baze podataka za ograničenje stranog ključa;	
NameTable	Naziv tabele koja je vezana za ograničenje stranog ključa;	
NameFKCon	Naziv ograničenja stranog ključa;	
NameDBMS	Naziv RSUBP-a kome pripada kolona LHS-a;	
NameDatabase	Naziv šeme baze kojoj pripada kolona LHS-a;	
Col_NameTable	Naziv tabele kojoj pripada kolona LHS-a;	
NameColumn	Naziv kolone koja pripada lijevoj strani ograničenja stranog ključa;	
Ključevi		
1.	[For_NameDBMS+For_NameDatabase+NameTable+NameFKCon+ NameDBMS+NameDatabase+Col_NameTable+NameColumn]	PK
Međurelaciona ograničenja		
LHS_ATTR[For_NameDBMS,For_NameDatabase,NameTable,NameFKCon] ⊆ FOREIGNKEYCON[NameDBMS, NameDatabase,NameTable,NameFKCon]		
LHS_ATTR[NameDBMS, NameDatabase,Col_NameTable,NameColumn]⊆ COLUMN_[NameDBMS, NameDatabase,NameTable,NameColumn]		

Tabela ForeignKeyCon

FOREIGNKEYCON	
NameDBMS	Naziv RSUBP-a kome pripada šema baze kojoj pripada ograničenje stranog ključa;
NameDatabase	Naziv šeme baze podataka kojoj pripada tabela u kojoj je dato ograničenje stranog ključa;
NameTable	Naziv tabele kojoj pripada ograničenje stranog ključa;
NameFKCon	Naziv ograničenja stranog ključa;
Tab_NameDBMS	Naziv RSUBP-a kome pripada šema baze koja se odnosi na tabelu sa desne strane ograničenja;
Tab_NameDatabase	Naziv šeme baze kojoj pripada tabela sa desne strane ograničenja;
Tab_NameTable	Naziv tabele sa desne strane ograničenja;

PK_SetOfColumns	Skup kolona sa desne strane ograničenja;
DeleteActionRHS	Specifikacija akcije koja se vrši u slučaju brisanja atributa sa desne strane ograničenja (NOACTION, RESTRICT, SETNULL, SETDEFAULT, CASCADE);
UpdateActionRHS	Specifikacija akcije koja se vrši u slučaju modifikacije atributa sa desne strane ograničenja (NOACTION, RESTRICT, SETNULL, SETDEFAULT, CASCADE);
Match	Specifikacija tipa referenciranja;
InverseReferentialIntegrityCon	Postojanje inverznog referecijalnog integriteta za dato ogranilenje stranog ključa;
Defferable	Specifikacija trenutka provjere ograničenja (DEFFERABLE, NOT_DEFFERABLE);
Deffered	Specifikacija trenutka provjere ograničenja (INITIALLY_IMMEDIATE, INITIALLY_DEFFERED);

Ključevi		
1.	[NameDBMS+NameDatabase+NameTable+NameFKCon]	PK

Međurelaciona ograničenja
FOREIGNKEYCON [NameDBMS, NameDatabase,NameTable] ⊆ TABLE_[NameDBMS, NameDatabase,NameTable]
FOREIGNKEYCON [Tab_NameDBMS, Tab_NameDatabase,Tab_NameTable] ⊆ TABLE_[NameDBMS, NameDatabase,NameTable]
FOREIGNKEYCON[PK_SetOfColumns]⊆SETOFCOLUMNS[PK_SETOFCOLUMNS]

Prilog C – Spisak često korišćenih skraćenica

CMOF	complete MOF
DBRE	Database Reverse Engineering
DSL	Domain Specific Language
EER	Extended Entity-Relationship
EMF	Eclipse Modeling Framework
EMOF	essential MOF
ER	Entity-Relationship
GMF	Graphical Modeling Framework
GPL	General Programming Language
IIS*Case	Integrated Information Systems CASE Tool
IIS*Ree	Integrated Information Systems Reenginering
IRIC	Inverse Referential Integrity Constraint
IS	Informacioni sistem
M2C	Model-to-code
M2M	Model-to-model
MD	Model Driven
MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
MDSE	Model Driven Software Engineering
MOF	Meta Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
RIC	Referential Integrity Constraint
RSUBP	Relacioni sistem za upravljanje bazama podataka
SEI	Software Engineering Institute
SUBP	Sistem za upravljanje bazama podataka
UML	Unified Modeling Languange
XMI	XML Metadata Interchange

Prilog D – Spisak slika

Prilog E – Spisak listinga

Prilog F – Spisak tabela