UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

Milan Čeliković

Pristup modelovanju specifikacija informacionog sistema putem namenskih jezika

- DOKTORSKA DISERTACIJA -

Mentor

Dr Ivan Luković, red. prof.

Novi Sad, 2018.

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**:		MSc Милан Челиковић
Ментор, **МН**:		др Иван Луковић, редовни професор
Наслов рада, **НР**:		Приступ моделовању спецификација информационог система путем наменских језика
Језик публикације, **ЈП**:		Српски
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Србија
Уже географско подручје, **УГП**:		АП Војводина
Година, **ГО**:		2018
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)		7/121/100/0/33/0/4
Научна област, **НО**:		Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:		Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, **ПО**:		Информациони системи, наменски језици, пројектовање база података, платформски независни модели, MDSD / CASE алати
УДК		
Чува се, **ЧУ**:		Библиотека Факултета техничких наука, Трг Д. Обрадовића 6, 21000 Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		Употреба платформски независног моделовања и наменских језика у поступку развоја информационих система скраћује време њиховог развоја и побољшава квалитет тог процеса. При томе, циљ је обезбеђење могућности да развој свих аспеката информационих система буде подржан оваквим приступом. Ова дисертација даје допринос у остварењу наведеног циља. У дисертацији представљени су наменски језици за моделовање спецификације информационог система засновани на приступу типа форме и EER концептима.
Датум прихватања теме, **ДП**:		13. 7. 2017.
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	др Соња Ристић, ванредни професор
	Члан:	др Борис Милашиновић, доцент
	Члан:	др Славица Кордић, доцент
	Члан:	др Милан Сегединац, доцент
	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**:		Milan Čeliković, M.Sc.	
Mentor, **MN**:		Ivan Luković, Ph.D., Full Professor	
Title, **TI**:		An Approach to Modeling Information System Specifications based on Domain Specific Languages	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Serbia	
Locality of publication, **LP**:		AP Vojvodina	
Publication year, **PY**:		2015	
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)		7/121/100/0/33/0/4	
Scientific field, **SF**:		Software engineering	
Scientific discipline, **SD**:		Applied computer science	
Subject/Key words, **S**/**KW**:		Information Systems, Domain Specific Languages, Database Design, Platform Independent Models, MDSD / CASE tools	
UC			
Holding data, **HD**:		Library of Faculty of Technical Sciences, 21000 Novi Sad, Trg Dositeja Obradovića 6	
Note, **N**:			
Abstract, **AB**:		The usage of platform-independent modelling and domain specific languages in information systems development process reduces the development time and improves the process quality. By that, the goal is to have all elements of an information system supported by this approach. The dissertation provides a contribution towards fulfilling the given goal. In the dissertation, author presents the domain specific languages for modelling information system specification based on the form type approach, and EER concepts.	
Accepted by the Scientific Board on, **ASB**:		July 13th, 2017	
Defended on, **DE**:			
Defended Board, **DB**:	President:	Sonja Ristić, Ph.D., Associate Professor	
	Member:	Boris Milašinović, Ph.D., Assistant Professor	
	Member:	Slavica Kordić, Ph.D., Assistant professor	Mentor's sign
	Member:	Milan Segedinac, Ph.D., Assistant Professor	
	Member, Mentor:	Ivan Luković, Ph.D., Full Professor	

Zahvaljujem se mentoru, prof. dr Ivanu Lukoviću, za veliku količinu znanja i saveta koje mi je preneo tokom doktorskih studija i izrade doktorske disertacije.

Takođe, zahvaljujem se porodici, prijateljima i kolegama na svesrdnoj podršci i pomoći.

Sadržaj

1. Uvod

Savremeno poslovno okruženje karakterišu globalizacija, sve veća konkurencija i tehnološke inovacije, naročito na polju informacionih tehnologija. Razvoj savremenog i složenog informacionog sistema podrazumeva implementaciju velikog broja formi za pregled, unos, izmenu i brisanje podataka, kao i kreiranje kompleksne strukture šeme baze podataka. Tradicionalni pristupi koji podrazumevaju programiranje u jeziku treće generacije, neefikasni su sa stanovišta iskorišćenja vremenskih i ljudskih resursa. Stoga, razvoj metodologija i alata koji automatizuju ovaj proces, bilo delimično ili, u idealnom slučaju potpuno, predstavlja važan segment u unapređenju softverskog inženjerstva, jer se time olakšava i unapređuje proces izgradnje kompleksnih informacionih sistema.

Stalni razvoj informacionih tehnologija dovodi do novih mogućnosti za njihovu upotrebu u informacionim sistemima (IS). Iako to obogaćuje informacione sisteme novim mogućnostima, današnje metode i tehnike razvoja IS i dalje su daleko od „optimalnih“ i univerzalno primenljivih. Brojni pristupi u razvoju IS formulisani su u poslednjih nekoliko decenija. Danas, međutim, nije neuobičajeno za mnoge unapređene starije pristupe da su još uvek u upotrebi. Informaciono društvo je takođe prihvatilo nove ideje uz zadržavanje svojih često primenjihvanih tehnika iz prošlosti. Uprkos pojave paradigmi *Model-Driven Software Engineering* (MDSE) i namenskih jezika (*Domain Specific Languages*, DSLs), projektanti IS i dalje angažuju dosta svog vremena u primeni tradicionalnog procesa konceptualnog modelovanja realnog sistema, koji se ne zasniva na primeni navedenih paradigmi.

1.1 Predmet istraživanja

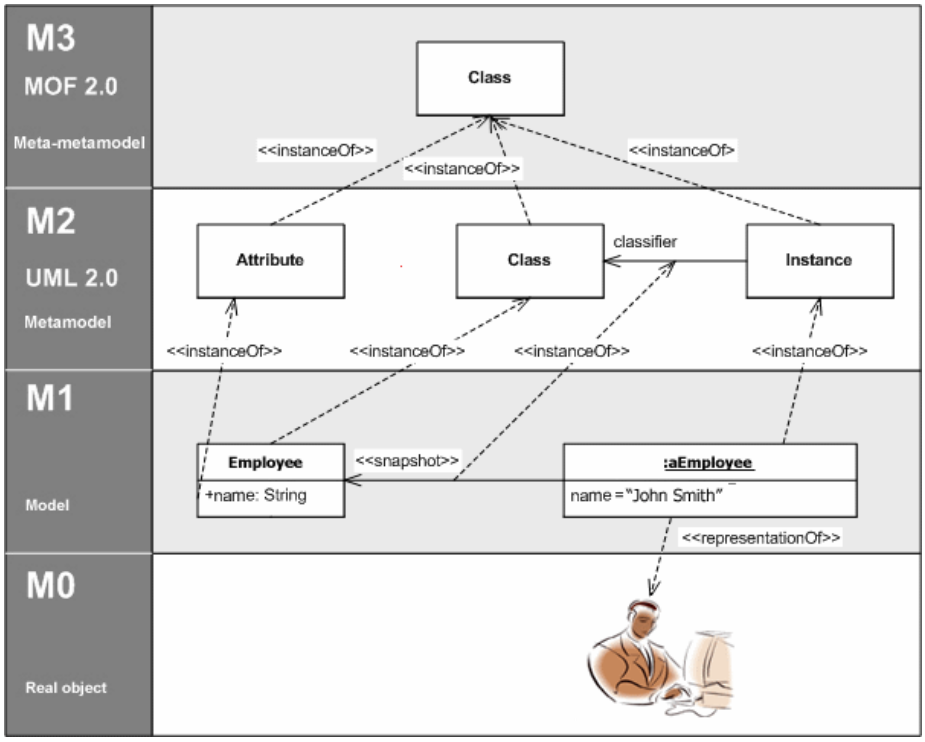
Glavni predmet ove doktorske disertacije je implementacija radnog okruženja za modelovanje specifikacija informacionog sistema putem namenskih jezika, na osnovu pristupa koji se zasniva na razvoju softvera vođenog modelima (*Model Driven Software Development*, MDSD). U ovom odeljku predstavljeni su i osnovni pojmovi – *modeli, jezici za modelovanje* i *koncepti za modelovanje* – koji su u vezi sa predmetom istraživanja. Pored objašenjenja osnovnih pojmova, pokazano je kako su ovi pojmovi organizovani četvoroslojnoj arhitekturi pristupa MDSD.

U MDSD, *modele* treba posmatrati kao glavne činioce u procesu razvoja softvera. Modeli se kreiraju u procesu identifikacije činilaca poslovanja iz realnog sistema, grupisanjem, generalizacijom, i odbacivanjem osobina koje su nebitne za određeni slučaj korišćenja. *Jezici za modelovanje* su neophodni kako bi se obezbedila odgovarajuća notacija za specifikaciju modela. Često, projektanti jezika za modelovanje pružaju korisnicma mogućnost da koriste notaciju koja je vizuelno i semantički bliska konceptima iz oblasti u kojoj se vrši modelovanje. Razvoj jezika za modelovanje zahteva prepoznavanje koncepata iz obalsti u kojoj se vrši modelovanje. *Koncepte za modelovanje* potrebno je preslikati na odgovarajuće koncepte jezika, čime se pruža grafička ili tekstualna notacija.

Jezici za modelovanje se razvijaju i koriste u skladu sa četvoroslojnom arhitekturom pristupa MDSD. Nivoi su numerisani prema stepenu apstrakcije, počevši od najnižeg, iskazanog kao M0.

Na slici 1.1 prikazana je četvoroslojna arhitektura meta-nivoa definisana od strane OMG grupe. Meta-meta-model, meta-model, model sistema i instanca modela formiraju četvoroslojnu arhitekturu meta-nivoa:

- 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, na lingvističkom nivou, izbegava ulazak u problem postojanja beskonačno mnogo meta-nivoa.



Slika 1.1. Ilustracija četvoroslojne arhitekture meta-nivoa

Jezici za modelovanje koji se oslanjaju na znanje iz određene oblasti i sadrže koncepte za modelovanje nazivaju se namenski jezici za modelovanje (*Domain Specific Modeling Language,* DSML) i mogu se posmatrati kao podskup skupa namenskih jezika. Bliskost krajnjeg korisnika konceptima za modelovanje iz određene oblasti predstavlja prednost namenskih jezika za modelovanje u odnosu na jezike opšte namene za modelovanje (*General Purpose Modeling Language,* GPML), kao što je Unified Modeling Language (UML). Korišćenjem takvog jezika, stručnjak za problemsku oblast ili krajnji korisnik iz problemske oblasti u stanju je da specificira rešenje na brži način, s manje grešaka i koristeći poznate koncepte, nego što bi to bio slučaj s jezicima opšte namene za modelovanje.

U okviru ovog istraživanja razvijena je specifikacija platformski nezavisnih modela (*Platform-independent model,* PIM) na M2 nivou, takva da omogućava opis specifikacije informacionog sistema na M1 nivou. Iz specifikacije M1 nivoa izvodi se implementacija informacionog sistema koja predstavlja model M0 nivoa.

Shodno tome, razvijena su tri DSML: 1) jezik za specifikaciju informacionog sistema pomoću koncepta tipa forme; 2) jezik za opis šablona korisničkog interfejsa i 3) jezik za opis informacionog sistema pomoću koncepata proširenog modela tipova entiteta i poveznika.

1.2 Ciljevi istraživanja

Već duži niz godina prisutni su pristupi razvoju informacionih sistema, zasnovani na primeni MDSD pristupa, takvi da obezbeđuju generisanje izvršnog programskog koda za namenska softverska okruženja za implementaciju informacionih sistema. Međutim, ovi pristupi zasnovani su na primeni opšte prihvaćenih specifikacionih tehnika, koje najčešće nemaju sve karakteristike nameskih jezika. S druge strane, namenski jezici imaju brojne prednosti u odnosu na jezike opšte namene (Hudak): 1) koncizniji su; 2) efikasniji za upotrebu; 3) lakši su za održavanje; 4) omogućuju jednostavniju analizu softverskih proizvoda i 5) omogućuju

da se u razvoj uključe i krajnji korisnici. Shodno tome, **glavni cilj** ovog istraživanja je ostvarenje *pristupa za specifikaciju informacionih sistema zasnovanog na upotrebi namenskih jezka za modelovanje*. Takav pristup zasniva se na principima MDSD, koji uključuje platformski nezavisno modelovanje, automatsku transformaciju u platformski zavisan model i automatsko generisanje izvršivog programskog koda.

Prvi prateći cilj ovog istraživanja je *analiza teorijskih osnova i postojećih pristupa zasnovanih na modelima i alate za razvoj IS*. Za svaki pristup, neopohodno je uočiti njegove glavne koncepte, prednosti i nedostatke. Na osnovu ovih nalaza, treba izdvojiti samo one pristupe koji se koriste za specifikaciju IS na konceptulanom nivou jer je to glavni cilj novog pristupa.

Jedna od najvažnijih prednosti namenskih jezika je i to što pružaju mogućnost aktivnog uključivanja krajnjih korisnika u razvoj IS. Krajnji korisnici informacioni sistem vide kroz korisnički intefejs sačinjen od ekranskih formi. Dakle, predloženi pristup treba da omogući modelovanje elemenata ekranskih formi. Prema tome, **drugi prateći cilj** je *razvoj namenskog jezika za modelovanje specifikacije informacionog sistema pomoću platformski nezavisnog tipa forme kojim se opisuje i korisnički interfejs IS-a*.

Pored strukture tipa forme, definisanje korisničkog interfejsa zahteva i specificiranje opštih karakteristika korisničkog interfejsa, kao na primer, koji će elementi korisničkog interfejsa biti iskorišćeni za elelemente forme i kako će elementi korisničkog interfejsa biti raspoređeni. Shodno tome, definišemo **treći prateći cilj** ovog istraživanja kao *razvoj namenskog jezika koji obezbeđuje modelovanje specifikacija šablona korisničkog interfejsa informacionog sistema*.

I pored brojnih prednosti koje namenski jezici pružaju, može se javiti potreba da se delovi informacionog sistema specificiraju i korišćenjem tradicionalnih pristupa. Zbog te potrebe uvodimo **četvrti prateći cilj**: *pružanje mogućnosti za integraciju tradicionalnog pristupa modelovanju informacionog sistema pomoću koncepata proširenog modela tipova entiteta i poveznika i novog pristupa za specifikaciju informacionog sistema pomoću koncepta tipa forme*.

Drugi, treći i četvrti prateći cilj uključuju kreiranje konkretnih i apstraktnih sintaksa namenskih jezika za opis specifikacija. Apstraktne sintakse koje će biti predložene jesu meta-modeli specificirani na PIM nivou.

Peti prateći cilj je *verifikacija predloženog pristupa*, koja će biti realizovana kroz razvoj softverskog alata za projektovanje informacionih sistema. Valjanost predloženog pristupa i softverskog alata će biti pokazana na prikazu slučaja razvoja jednog informacionog sistema.

1.3 Hipoteze u istraživanju

Motivi za istraživanje i primenu oblasti MDSE i razvoja DSL u okviru inženjerstva, tj. razvoja, projektovanja i realizacije IS, bili su problemi na koje se nailazi u primeni tradicionalnih pristupa razvoju IS. Skoro sve aktuelne metode za razvoj IS zasnovane su na modelima. Konceptualno modelovanje je faza koja se posebno ističe u tradicionalnom razvoju IS. Očekivani rezultat ove faze je specifikacija realnog sistema upotrebom odabranih PIM koncepata.

Dugi niz godina, prošireni model tipova entiteta i poveznika *Extended Entity-Relationship Data Model* (EER) je najčešće korišćeni konceptualni model podataka. Tipičan scenario procesa projektovanja šeme baze podataka koji obezbeđuje većina postojećih *Computer-aided software engineering* (CASE) alata polazi od postupka kreiranja EER šeme baze podataka koja se zatim transformiše u relacionu šemu baze podataka. Takav postupak ima mnoge prednosti, ali postoje i ozbiljni nedostaci (Lukovic 2009). Upotreba metoda za razvoj IS, koje su zasnovane na tehnikama kao što je EER modelovanje ili jezika opšte namene za modelovanje kao što je UML, pa čak i relacionog modela podataka i odgovarajućeg alata u tom slučaju, zahteva napredno znanje i odogovarajuće veštine. U slučaju kada je nemoguće pronaći odgovarajući broj projektanata IS koji poseduju ove osobine može doći do rizika od projektovanja IS lošeg kvaliteta (Kosar et al. 2006). Osim toga, ove metode i tehnike su često nerazumljive krajnjim korisnicima. U praksi, ova činjenica može dovesti do problema u komunikaciji i do nesporazuma između projektanata i krajnjih korisnika. Po pravilu, rezultat nesporazuma je loše projektovana šema baze podataka ili softverska aplikacija, jer nije garantovano sagledavanje svih navedenih zahteva krajnjih korisnika. Projektanti i krajnji korisnici obično postaju svesni nedostataka suviše kasno, kada je završen razvoj šeme baze podataka ili softverske aplikacije.

Kako bi ovakvi nedostaci bili svedeni na tolerantni minimum, napravljen je alterantivni pristup za razvoj IS, koji je zasnovan na upotrebi metode za razvoj softvera zasnovanoj na modelima (*Model-Driven Software Development* MDSD) (Bézivin 2004) i namenskih jezika (Deursen et al. 2000, Mernik et al. 2005). Naziv ovog pristupa razvoju IS je *Pristup zasnovan na konceptu tipa forme* (FT) i detaljno je opisan u radovima (Luković et al. 2007, Luković et al. 2006, Aleksić et al. 2007, Aleksić et al. 2011., Luković et al. 2010, Ristić et al. 2003, Ristić et al. 2007). Osnovna ideja je da se projektantima IS obezbede neophodni koncepti za razvoj na meta-nivou. Projektanti IS mogu pomoću ovakvih meta-modela, lako da modeluju realni sistem. Nakon toga, projektanti IS mogu bez ikakvog dodatnog stručnog znanja, da koriste niz formalnih metoda i složenih algoritama za kreiranje specifikacije šeme baze podataka i izvršnog koda IS. Kako bi FT pristup bio praktično podržan i kako bi bilo omogućeno korišćenje PIM koncepata u procesu projektovanja IS, razvijen je od strane autora pristup i softverski alat IIS*Studio. Alat IIS*Studio pruža mogućnost za evolutivni i inkrementalni razvoj IS. Nadalje, podržano je konceptualno projektovanje IS putem koncepata koji su bliski percepciji krajnjeg korisnika i ne zavise od platforme za kasniju implementaciju IS. IIS*Studio danas podržava sledeće funkcionalnosti:

1) konceptualno projektovanje šeme baze podataka, transakcionih programa, poslovnih aplikacija i informacionog sistema;
2) automatizovano projektovanje podšema relacione baze podataka u trećoj normalnoj formi (3NF);

3) automatizovanu integraciju podšema u jedinstvenu šemu baze podataka u trećoj normalnoj formi;
4) generisanje SQL opisa šeme baze podataka za različite sisteme za upravljanje bazama podataka;
5) generisanje XML specifikacije informacionoog sistema;
6) projektovanje aplikacija informacionog sistema i vizuelnih atributa ekranskih formi;
7) zadavanje ograničenja torki, transformacija i generisanje SQL koda za proveru važenja tih ograničenja na nivou konceptualne šeme baze podataka;
8) reverzni inženjering relacionih baza podataka u modele FT;
9) generisanje programskog koda ekranskih formi transakcionih programa; i
10) automatizovanu detekciju formalnih kolizija ograničenja.

U pristupu na kojem se IIS*Studio 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 IS ne navodeći detalje vezane za implementacionu platformu. Počevši od modela IS koji se formira na visokom stepenu nezavisnosti od platforme, nizom transformacija modela, pomoću alata IIS*Studio, može se dobiti više platformski zavisnih modela (*Platform-specific model*, PSM). Model koji je, u krajnjoj liniji, potpuno usmeren na platformu je konkretni implementaciono orijentisan opis aplikacija informacionog sistema ili implementacioni opis šeme baze podataka, iskazan putem koncepata izabranog proizvođača sistema za upravljanje bazama podataka SUBP. (Banović 2010, Ristić et al. 2011, Ristić et al. 2014, Aleksić et al. 2013)

Postoji nekoliko alata koji su integrisani u softversko okruženje IIS*Studio. Neki od njih omogućuju tzv. model-u-model transformacije, dok drugi omogućavaju tzv. model-u-kod transformacije. Na apstraktnom nivou PIM-a, IIS*Studio obezbeđuje konceptualno modelovanje šeme baze podataka koje uključuje specifikaciju raznih vrsta ograničenja baze podataka, kao što su: ograničenje domena, ograničenje nula vrednosti, ograničenje torke (eng. 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, jer je zasnovan na relacionom modelu, ali je nezavisan od konkretnog proizvođača. Dobijeni model se može transformisati model-u-kod transformacijom u SQL opis šeme baze podataka. Generisani SQL opis može biti usaglašen sa standardom i time imati još uvek određeni stepen implementacione nezavisnosti, ili može biti formiran prema sintaksi izabranog proizvođača SUBP, čime će predstavljati potpuno platformski zavisan implementacioni opis šeme baze podataka (Luković et al. 2010). Ovakav pristup razvoju IS predstavlja vrstu MDSD pristupa razvoju sistema (Luković et al. 2008).

Pored potrebe da PIM specifikacije budu zadate na nivou repozitorujma, postoji i snažna potreba da se ove specifikacije reprezentuju i pomoću teksutalnog jezika iz sledećih razloga: 1) Iako možemo očekivati da prosečni korisnici više vole da koriste vizuelno orijentisane alate za kreiranje PIM specifikacije, potrebno je obezbediti iskusnijim korisnicima tekstualni jezik i alat za efikasnije kreiranje PIM specifikacije. 2) Pored definicije IIS*Studio meta-modela na nivou repozitorijuma, nophodno je da postoji ekvivalentna reprezentacija meta-modela predstavljena nezavisno od platfome na kojoj je implementirana. Na ovaj način, meta-model postaje potpuno nezavisan od definicije repozitorijuma, koja obično može da uključi neke detalje na nivou implementacije. 3) Neophodno je definisati osnovu za razvoj različitih algoritama za proveru formalne ispravnosti kreiranih modela, kao i za implementaciju semantičke analize. 4) Zbog prethodnih razloga je neohodna specifikacija

koncepata i pravila na meta-nivou sa stanovišta gramatike jezika. Pomoću takve gramatike, moguće je specificirati namenski jezik koji sadrži koncepte i pravila iz domena koja su primenjena u specifikaciji IS na konceptualnom nivou u alatu IIS*Studio (Luković et al. 2011).

Iz navedenih motiva mogućeg istraživanja u oblasti razvoja IS zasnovanog na MDSD pristupu i primeni namenskih jezika, definisana je početna hipoteza predloženog istraživanja:

H0: Moguće je definisati pristup za specifikaciju informacionog sistema, koji je zasnovan na upotrebi namenskih jezika za modelovanje, primenom principa MDSD, koji uključuje platformski nezavisno modelovanje, automatsku transformaciju u platformski zavistan model i automatsko generisanje izvršivog programskog koda.

Specifikacija informacionog sistema treba aktivno da uključi krajnjeg korisnika. S obzirom da krajnji korisnik informacioni sistem vidi kroz ekranske forme ili dokumenta, centralni koncept u predloženom pristupu specifikaciji informacionih sistema je *tip forme*, koji predstavlja generalizaciju ekranskih formi, ili konkretnih dokumenata u poslovanju. Shodno tome definišemo sledeću hipotezu:

H1: Moguće je razviti namenski jezik za modelovanje specifikacije informacionog sistema pomoću platformski nezavisnih koncepata tipa forme. Takav namenski jezik treba da podrži pristup razvoju softvera koji je zasnovan na modelima. Osim toga, jezik treba da sadrži najvažnije koncepte definisane na PIM nivou koji pružaju dovoljnu izražajnost.

Tipom forme opisuju se ekranske forme koje korisnik upotrebljava u svrhu komunikacije s informacionim sistemom. U predloženom pristupu projektant specificira ekranske forme transakcionih programa, a indirektno, kreira inicijalni skup atributa i ograničenja. Da bi korisnički interfejs bio potpun, pored strukture tipa forme potrebno je zadati i izgled samog interfejsa, što se obezbeđuje primenom *šablona korisničkog interfejsa informacionog sistema*. Šablon korisničkog interfejsa predstavlja specifikaciju opštih karakteristika korisničkog interfejsa. Na osnovu toga uvodimo sledeću hipotezu:

H2: Moguće je razviti namenski jezik koji obezbeđuje modelovanje specifikacija za opis šablona korisničkog interfejsa informacionog sistema.

Predloženi pristup je semantički bogatiji od tradicionalnih pristupa zasnovanih na modelovanju podataka, zato što omogućuje i definisanje elemenata poslovne logike kao i izgleda korisničkog interfejsa. Međutim, može se javiti potreba da se delovi informacionog sistema specificiraju na tradicionalan način. Stoga je potrebno omogućiti da se tradicionalni pristup integriše s predloženim. Shodno tome definišemo i sledeću hipotezu:

H3: Moguće je integrisati rezultate primene tradicionalnog pristupa za modelovanje informacionog sistema pomoću koncepata proširenog modela tipova entiteta i poveznika i novog pristupa za specifikaciju informacionog sistema pomoću koncepta tipa forme.

Potrebno je omogućiti formalnu verifikaciju predloženog pristupa. Verifikacija treba da obuhvati razvoj softverskog alata zasnovanog na predloženom pristupu. Valjanost razvijenog sofveroskog alata će biti proverena kroz prikaz slučaja implementacije jednog informacionog sistema. Prema tome definišemo sledeću hipotezu:

H4: Moguća je praktična provera valjanosti razvijenih namenskih jezika kroz njihovu implementaciju u okviru jednog MDSD alata, namenjenog projektovanju informacionih sistema i baza podataka.

1.4 Metodologija istraživanja

U nastavku, dat je opis metodologije istraživanja po ključnim koracima: a) definisanje postupka analize postojećih softverskih rešenja u oblasti razvoja IS, b) definisanje postupka razvoja pristupa za modelovanje specifikacija IS i c) analiza sa ocenom pristupa za modelovanje specifikacija IS.

1.4.1 Postupak analize postojećih pristupa u oblasti razvoja IS

Prvi korak koji je potrebno sprovesti kao deo predloženog istraživanja je proučavanje postojećih pristupa u oblasti razvoja IS. Prednosti ovakvog istraživanja su dvostruki. Prva prednost je mogućnost uočavanja prednosti i mana, dobrih strana primene, kao i načina upotrebe svakog rešenja. Ovakva vrsta analize je neophodna, jer je planirana pre faze implementacije novog pristuipa. U postupku analize potrebno je uočiti glavne koncepte, koje ova rešenja koriste u fazi specifikacije modela IS. Stečena iskustva i informacije biće od koristi u razvoju novog pristupa. Još jedna prednost ovakve analize je ta da predstavlja osnovu za procenu novog pristupa. Nakon završetka faze razvoja pristupa i odgovarajućeg alata, neophodno je uporediti koncepte, performanse i korisnička iskustva sa softverskim rešenjima koja su analizirana u okviru ovog koraka.

1.4.2 Postupak razvoja pristupa za modelovanje specifikacija IS

Razvoj skupa namenskih jezika za modelovanje specifikacija IS biće sproveden kao iterativni proces sa sledećim aktivnostima: prikupljanje zahteva, analiza zahteva, projektovanje, implementacija, testiranje, otklanjanje nedostataka i stavaljanje u upotrebu.

Zahtevi će biti prikupljeni na na tri načina: 1) Intervju sa stručnjacima u oblasti projektovanja IS, 2) Analiza dokumentacije i 3) Analiza postojećih softverskih rešenja u oblasti projektovanja IS. Najvažniji izvor funkcionalnih i nefunkcionalnih zahteva su stručnjaci iz oblasti poslovanja koju pokriva IS. S obzirom da su oni i krajnji korisnici alata, njihova očekivanja i prethodna iskustva su od izuzetnog značaja za čitav proces razvoja. Zahtevi koji će biti definisani, pokrivaju trenutne metode i tehnike u oblasti softvera za razvoj IS.

Na osnovu definisanih uslova iz prethodnog koraka, potrebno je isprojektovati sledeće elemente sistema: 1) pristup modelovanju specifikacija IS, 2) koncepte jezika za modelovanje, i 3) arhitekturu alata. Faze implentacije, testiranja i otklanjanja grešaka će biti sprovedene nakon faze projekotvanja, u cilju kreiranja alata za specifikaciju modela IS. Programski jezik Java će biti upotrebeljen za implementaciju alata, jer pruža velike mogućnosti za pravljenje namenskih jezika za modelovanje putem EMF okruženja. Korišćenjem EMF okruženja, kreirani alat biće ugrađen, kao dodatak, u Eclipse okruženje.

Aktivnost održavanja kreiranog softverskog alata nije deo ovog istraživanja. Ona može biti samo deo redovne upotrebe alata u njegovoj praktičnoj primeni, što izlazi izvan okvira predloženog istraživanja.

1.4.3 Metod ocenjivanja

Ovaj pristup će biti ocenjen na osnovu unapred definisanih primera. Rezultati ocenjivanja će biti upoređeni sa rezultatima analize softvera iz prvog koraka predložene metodologije istraživanja. Testranjem pristupa i razvijenog alata na unapred definisanim primerima biće proverena njihova funkcionalnost i mogućnosti u izabranom domenu primene. Rezultati testiranja treba da pruže odgovor da li alat poseduje sve neophodne koncepte za modelovanje i funkcionalnosti koje su neophodne u praktičnoj upotrebi. Poređenje sa drugim alatima omogućiće proveru efikasnosti pristupa i alata u kontekstu postojećih alata za modelovanje specifikacija IS.

Pre ocenjivanja, neophodno je pripremiti primere za testiranje. Kako bi procena pružila relevantne informacije, neophodno je prikupiti podatke iz realnog sistema. Neophodno je instalirati alat na „čistom“ operativnom sistemu i testirati ga u kontrolisanim uslovima, kako bi bio obezbeđen rad alata bez ikakvih dodatnih spoljnih uticaja.

Ocenjivanje će obaviti nekoliko stručnjaka iz oblasti razvoja IS u kontrolisanom okruženju u istom vremenskom periodu. Kontrolisano okruženje podrazumeva da korisnici imaju iste računare, dozvole pristupa i podešavanja alata. Nakon završenog ocenjivanja rezultati će biti analizirani. Analiza dobijenih rezultata treba da ukaže na dalja poboljšanja pristupa i alata i naredne pravce istraživanja (Dimitrieski et al. 2015).

1.5 Struktura disertacije

Pored Uvoda i Zaključka, 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 specifikaciji informacionog sistema na konceptualnom nivou. 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, pod nazivom *Pregled karakteristika i mogućnosti okruženja IIS*Studio*, opisane su 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 čak ne moraju biti ni 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 četvrtom poglavlju, pod nazivom *Meta-modeli za specifikaciju informacionih sistema*, data je specifikacija svih meta-modela neophodnih za specifikaciju koncepata namenskih jezika. Biće prikazani sledeći meta-modeli:

- IIS*Case meta-model,
- Meta-model koncepta šablona korisničkog interfejsa i
- Meta-Model tipova entiteta i poveznika

U petom poglavlju, pod nazivom *Konkretne sintakse namenskih jezika za specifikaciju informacionih sistema*, data je specifikacija konkretnih tekstuaknih sintaksa sledećih namenskih jezika:

- za modelovanje specifikacije informacionog sistema pomoću platformski nezavisnih koncepata tipa forme,
- za modelovanje specifikacija za opis šablona korisničkog interfejsa informacionog sistema i
- za modelovanje informacionog sistema pomoću koncepata proširenog modela tipova entiteta i poveznika.

U šestom poglavlju, prikazana je praktična primena razviijenih namenskih jezika na karakterističnom primeru. Pored praktične primene dat je prikaz evaluacije namenskog jezika koji je zasnovan na tipu entiteta i tipu poveznika.

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 četiri priloga. *Prilog 1* sadrži primer specifikacije informacionog sistema opisan pomoću EERDSL namenskog jezika. *Prilog 2* reprezentuje primer specifikacije informacionog sistema koji je specificiran putem FTDSL i UIDSL namenskih jezika. Spisak često korišćenih skraćenica i spisak slika predstavljeni su, redom, u prilozima 3 i 4.

2. Pregled trenutnog stanja u oblasti

U nastavku ovog poglavlja, predstavljena je evolucija tehnika za razvoj IS u tri istorijske faze u skladu sa važnim prelazima koji su se desili u softverskom inženjerstvu i koje su izazvane prihvatanjem osnovnih principa MDSE i namenskih jezika. Pored opisa pristupa tipičnih za te periode, prezentovane su prednosti i nedostaci reprezentativnih metoda. Najveća pažnja je posvećena namenskim jezicima, koji pokušavaju da iskoriste prethodno popularne metode i njihove prednosti.

2.1 Tradicionalni pristupi za razvoj informacionih sistema

Izazovi u razvoju IS nisu vezani isključivo za tehnologije koje se koriste. Postoje brojni problemi u vezi sa poslovanjem, organizacionim i sociološkim aspektima, bitnim za razvoj IS. Uzimajući u obzir širok pogled na informacione sisteme, Hirschheim & Klein (1989) identifikovali su paradigme u razvoju informacionih sistema. Autori ovog rada posmatraju *epistemološku osu* koja saznanja deli na *subjektivna* i *objektivna* i *ontološku osu* koja posmatra odnos *uređenosti* i *neuređenosti* sveta. Shodno tome, definisane su četiri moguće kombinacije između dimenzija: 1) red sukob, ističući stabilnost ili promene u društvu i 2) objektivinost-subjektivnost, naglašavajući mogućnost ili nemogućnost proučavanja društva koristeći objektivne metode prirodnih nauka. Moguće kombinacije su ujedno i paradigme interpretiranja informacionih sistema: 1) *Funkcionalizam* (red, objektivnost); 2) *Društveni relativizam* (red, subjektivnost); 3) *Radikalni strukturalizam* (sukob, objektivnost) i 4) *Neohumanizam* (sukob, subjektivnost). S obzirom da pristup predložen u ovom radu pogoduje okruženjima u kojima se korisnički zahtevi dinamički menjaju, na ontološkoj osi bi predloženi pristup bio pozicioniran na delu koji se odnosi na sukob, a ne na red. Pošto predloženi pristup pogoduje aktivnom uključivanju krajnjeg korisnika u razvoj informacionog sistema i omogućuje mu da iskaže svoje subjektivne stavove, na episetmološkoj osi bismo ga pozicionirali na subjektivni deo. Stoga predloženi pristup u ovoj doktorskoj disertaciji smatramo *neohumanističkim*.

Problemi na koje se nailazi u razvoju IS zajedno sa rešenjima problema, predstavljali su motiv za definisanje discipline u okviru inženjerstva koja je fokusirana na projektovanje i razvoj IS (Brinkkemper 1996). Danas su skoro sve popularne metode za razvoj IS zasnovane na modelima. Jedna od posebno istaknutih faza u razvoju IS je konceptualno modelovanje. Očekivani rezultat ove faze je specifikacija realnog sistema upotrebom odabranih PIM koncepata. U istraživanju konceptualnih modela, četiri važna elementa su identifikovali Wand & Weber (2002): *gramatike* koje predstavljaju skupove konstrukata i pravila koja ih međusobno povezuju, *metode* koje predstavljaju primenu gramatike, *skripte* koje predstavljaju rezultat modelovanja, i *kontekst modelovanja*. U ovom istraživanju biće predložene gramatike namenskih jezika namenjene za modelovanje tipa forme, šablona korisničkog interfejsa i integracije sa tradicionalnim pristupom modelovanju IS. Za te gramatike biće predložene metode primene. Skripte i kontekst modelovanja biće ilustrovani kroz prikaz slučaja.

Jednu klasifikaciju tehnika modelovanja je predložio Giaglis (2001). Autor je dao pregled popularnih tehnika za modelovanje IS kao što su: dijagram toka podataka, dijagram objeti-veze (ER), dijagram prelaza stanja, IDEF1k, i UML. U radu, tehnike su kategorisane prema svojoj “širini” i “dubini” što obuhvata ciljeve i zadatke u modelovanju. U samom radu su

prezentovane prednosti i mane različitih tehnika za modelovanje IS iz ugla projektanta IS i krajnjeg korisnika.

Postoji veliki broj notacija za specifikaciju podataka i baza podataka jer podaci su od suštinskog značaja za svaki IS. Ove notacije se mogu smatrati namenskim jezicima za modelovanje struktura baza podataka. Neke od poznatih metoda uključuju upotrebu hijerarhijskog modela, objektno-orijentisanog modela, ER modela predloženog od Chen-a (1976) sa svojim kasnijim proširenjima i relacioni model podataka predloženog od Codd-a (1970).

Sve veća upotreba konceptulanog modela u procesu razvoja IS je donela razne prednosti, ali istovremeno i nove probleme. Jedna od prednosti konceptulanog modela je da olakša saradnju između programera i krajnjih korisnika. Ovaj pristup pruža mogućnost u kojoj krajnji korisnik može bolje da shvati sistem koji se razvija, pa čak i da učestvuje u procesu modelovanja. Iako ova vrsta modela predstavlja početnu fazu razvoja IS razumljivijom za prosečnog krajnjeg korisnika, stručnjaci u oblasti modelovanja ne treba da budu isključeni iz ove faze. U radu koji objavio je Shanks (1997), zaključeno je da su modeli podataka specificarani od strane stručnjaka iz oblasti modelovanja podataka, tačniji, potpuniji, fleksibilniji i razumljiviji od onih koje su definisali početnici ili krajnji korisnici. Osim toga, konceptualno modelovanje se smatra da je "teško" u praksi i, iz tog razloga, potrebna je konstantna pomoć stručnjaka u oblasti modelovanja (vom Brocke & Buddendick 2006). Neke od opštih preporuka za modelovanje prikupljenim u Uputstvu za modelovanje (GoM) od strane Schuette & Rotthowe (1998) opisuju prikladnost jezika za specifikaciju modela.

Tradicionalne metode za razvoj IS, koje su zasnovane na tehnikama kao što je EER modelovanje ili jezika opšte namene za modelovanje kao što je UML, zahtevaju napredno znanje i odogovarajuće veštine. Vrlo često nemoguće je pronaći kvalitetne projektante IS koji poseduju ovakve veštine i sposobnosti za njihovu praktičnu primenu, što vodi ka riziku projektovanja IS lošeg kvaliteta. Krajnjim korisnicima su ove metode i tehnike vrlo često nerazumljive, što može dovesti do problema u komunikaciji i do nesporazuma između projektanata i krajnjih korisnika. U ovakvoj situaciji nije garantovano sagledavanje svih navedenih zahteva krajnjih korisnika, a kao krajnji rezultat nastaje loše projektovana šema baze podataka ili softverska aplikacija (Aleksić et al. 2007, Luković 2009, Luković et al. 2003, Luković et al. 2007, Luković et al. 2013).

2.2 MDE pristupi

Važan događaj u istoriji metoda za razvoj softvera bio je uvođenje *Model Driven Architecture* (MDA) od strane Object Management Group (OMG). MDA se može posmatrati kao okvir za razvoj softvera fokusiran na stvaranje PIM modela, koji su nezavisni od tehnologije implementacije. Preko automatizovanih procedura transformacije, PSM izvode se iz PIM. PSM je vezan za određenu tehnologiju implementacije i koristiti se kao osnova za implementaciju odgovarajućih programa. Na ovaj način, kompletna softverska aplikacija može da se generiše korišćenjem lanaca modela transformacija koja polazi od jednog PIM modela. Zbog toga, od projektanta se očekuje da specificira samo PIM, dok se ostatak procesa vrši putem dostupnih procedura transformacije.

Vara et al. (2009) predlažu pristup vođen modelima za automatsko razvijanje objektno-relacionih (OR) šema baza podataka. Dijagram PSM (ili modela) je stvoren kroz

transformaciju modela, gde je ulaz u lanac transformacija PIM u obliku UML klasa. U sledećem koraku SQL kod se generiše putem model-tekst transformacije iz PSM. Još jedan primer pristupa koji sledi logiku sličnu MDA je OO - metod za modelovanje i razvoj IS (Pastor et al., 2001). U radu je opisan objektno orijentisani (OO) pristup koji polazi od skupa sistemskih zahteva definisanih u tri različite vrste konceptualnih modela: objektni, dinamični, i funkcionalni. Ovi modeli se automatski pretvaraju u formalnu OO specifikaciju koja se dalje preslikava u implementaciju u jednom od OO programskih jezika, čime se razvija IS u skladu sa zahtevima. Slično ovom pristupu, u IIS*Studio alatu, razvoj sistema počinje od PIM koji koji predstavlja osnovu za razvoj IS. Razvoj se dalje odvija putem transformacije modela i generisanja kôda. Finalni proizvodi su izvršni kôd i odgovarajuće skripte baze podataka za izabranu platformu (Luković et al. 2007;. Luković et al. 2008.).

Model Driven Engineering (MDE) je termin koji se često koristi u kombinaciji sa MDA. MDE je opisan kao "obećavajući pristup za prevazilaženje problema složenosti platformi i nemogućnosti jezika treće generacije za ublažavanje ove kompleksnosti i predstavljanja koncepata domena na efikasan način" (Schmidt 2006 str. 26-27). MDE pristup kombinuje namenske jezike za modelovanje sa odgovarajućim transformacijama i generatorima koda. Prema ovoj viziji, MDE treba da obuhvati apstrakcije na najvišem nivou i da pomogne da se proces stvaranja softvera olakša. MDE je širok pojam koji obuhvata MDA ali se ne ograničava samo na njega.

Prednost korišćenja MDA pristupa treba da se odnosi na poboljšanje produktivnosti programera, prenosivost generisanog softvera na različite platforme, rad generisanog softvera na različitim platformama, kao i mogućnost dokumentovanja i održavanja softvera (Kleppe et al. 2003). Ipak, MDA pristup je istovremeno kritikovan zbog preambicioznih, možda čak i nedostižnih ciljeva. Neki od osnovnih problema uključuju i pitanje podrške za PSM usmerene na složene platforme, neophodnost stalnog ažuriranja na najnoviju verziju platforme, kao i podršku za kompleksne moderne višeslojne aplikacije (Thomas, 2004). Jedno od mogućih rešenja za ove probleme može biti veće oslanjanje na domenski orijentisane programske jezike (DOP), tj. namenske jezike, koji "omogućuju domenskom stručnjaku preslikavanje apstrakcija domena u DOP apstrakcije" (Thomas & Barry 2003). Ovo rešenje može da posluži kao osnova za implementaciju namenskog jezika. Na sličan način, Kelly (2005) tvrdi da je glavni fokus u MDA, UML, koji obično ne obuhvata znanje iz odgovarajućeg domena koje je prisutno u organizacijama. Autor je naveo da ključ za dalje poboljšanje produktivnosti programera leži u primeni namenskog modela za domen (*Domain Specific Model,* DSM). Primena DSM bi mogla da proizvede modele, koji bi mogli da budu upotrebljeni kao ulazne vrednosti za automatsko generisanje koda. Zagovornici MDA smatraju da su mogućnosti UML profila dovoljni za efikasno prilagođavanje UML-a domenu ili platformi (Fuentes-Fernandez & Vallecillo-Moreno 2004). Pored toga, moguće je stvoriti potpuno nove meta-modele pomoću MOF (“MOF”) specifikacije ako je potrebno. Jedna od implementacija putem ovakvog pristupa je predstavljena u Perišić et al. (2011). Novi pristup za specifikaciju IS koji je predložen u ovom radu, podrazumeva specifikaciju modela pomoću PIM koncepata koji su specificirani na nivou meta-modela koji je opisan pomoću MOF specifikacije.

Ideje i praktična rešenja koja su predstavjena u prethodnom pasusu pokazuju da se čak i sa upotrebom MDA pristupa, javljaju problemi po pitanju složenosti. Ovakva vrsta problema zahteva primenu nameskih jezika, koji omogućuju efikasnije rešavanje problema pomoću domenskog znanja ugrađenog u softver. Spoj ove dve paradigme nije samo obostrano koristan, već predstavlja sledeći logičan korak u evoluciji metoda razvoja IS.

2.3 Namenski jezici u razvoju informacionih sistema

Postoje mnogi primeri primene namenskih jezika u različitim fazama razvoja softvera i kontekstima od značaja za razvoj IS: 1) specifikacija i generisanje softverskih konektora između softverskih komponenti (Bureš et al. 2008.)., 2) integracija heterogenih sistema (Frantz et al. 2008; Milanović et al. 2009), 3) komunikacija između heterogenih zdravstvenih informacionih sistema (Menezes et al. 2010), 4) specifikacija i preslikavanje podataka (Grundy et al. 2004), 5) specifikacija strukture baze podataka (Dejanović et al. 2010a), 6) kroki alat za specifikaciju korisničkog interfejsa poslovnih aplikacija (Filipović et al. 2017, Milosavljević et al. 2013) itd. Sprecifikacija softverskih arhitektura je još jedan domen za koji postoje namenski jezici. Neki od jezika za opis arhitektura (ADL) su AADL ("AADL"), Acme ("Acme"), Aesop ("Aesop") i Wright ("Wright"). Đukić et al. (2011) opisuju upotrebu namenskih jezika u procesu formalne specifikacije i interpretacije dokumenata u oblasti izdavaštva. Autori su razvili četiri jezika za modelovanje: 1) malih oglasa, 2) odgovarajućih dokumenata, 3) specifikacije poslovnih procesa i 4) vizuelno-prostorne organizacije dokumenata. Autori su razvili integrisani meta-model koji predstavlja osnovu za namenske jezike pod nazivom *DVDocLang*. Ovi namenski jezici razvijeni su putem razvojnog okruženja *Metaedit+* (MetaCaseMetaEdit+). Živanov et al. (2008) predstavili su *Kiosk Specification Language* (KSL), namenski jezik za razvoj kiosk aplikacija. KSL je tekstualni namenski jezik koji sakriva sve suvišne detalje implementacije kiosk aplikacija koje podržavaju ekrane osetljive na dodir. Ovakav postupak čini proces razvoja kiosk aplikacija manje sklon greškama. Pomoću tekstualne specifikacije u KSL, kiosk aplikacija se generiše korišćenjem odgovarajućeg šablona za određenu platformu. Putem ovog pristupa, podrška novih odredišnih platformi omogućena je dodavanjem odgovarajućih šablona.

Osim što mogu da posluže za konceptualno modelovanje IS, namenski jezici mogu imati dodatne uloge u radu IS u raznim oblastima. Na primer, oni se mogu koristiti u korporativnim informacionim sistema (KIS). Freudenthal (2010b) je istakao da je KIS obično "plitak, ali širok". Ova tvrdnja podrazumeva potrebu za uključivanjem različitih namenskih jezika i jednog jezika opšte namene tako da KIS može da poseduje potrebnu funkcionalnost. Analiza dostupnih namenskih jezika pokazala je da, u tom trenutku, nije postojao takav skup namenskih jezika koji mogu da podrže sve zahteve za proces integracije namenskih jezika u jedan KIS. Na sličan način, namenski jezik može da se koristi u prilagodljivim IS za određivanje pravila verifikacije dokumenata (Freudenthal, 2010a). Za domen primene informacionih sistema u bolnicama (HIS), razvijen je *Revised Three-layer Graph-based Meta Model* (3LGM2) i odgovarajući alat za njegovu podršku (Wendt et al. 2004;. Winter et al. 2003). Meta-model je napravljen pomoću UML-a kako bi bila podržana tri međusobno povezana sloja: domen, logički sloj i fizički sloj. Pomoću odgovarajućeg alata, domenski stručnjaci mogu da specificiraju HIS model putem predloženog meta modela 3LGM2. Prethodno navedeno istraživanje koje opisuje razvoj informacionih sistema za bolnice je slično predloženom pristupu u ovom radu. Ograničenje prezentovanog HIS modela je domen primene u informacionim sistemima za bolnice. Novi pristup predložen u ovom radu treba da omogući specifikaciju IS u bilo kojem domenu primene.

Integracija MDSE i namenskih jezika u jedan pristup pruža mogućnost specifikacije modela iz različitih pogleda na realan sistem. Jedan primer kombinacije ova dva pristupa je u razvoju *Eclipse* dodataka zasnovanih na repozitorijumu (Sivonen, 2008). Ovi dodaci mogu se specificirati pomoću specijalno razvijenog namenskog jezika za modelovanje (DSML) koji podržava sve koncepte bitne za ovaj domen. Dodaci se generišu na osnovu modela specificaranih putem DSML i predstavljaju proizvode odgovarajućih generatora koda.

Programer jedino mora da specificira model novog *Eclipse* dodatka. Bicevskis et al. (2011) napravili su DSML za modelovanje poslovnih procesa (*Business Process Modeling* BPM) kod kojeg se model sastoji od tri vrste dijagrama: poslovnih procesa (*Business Process* BP), informacionog sistema (IS) i dijagrama korisničkih servisa. Dijagrami IS prvenstveno su namenjeni za upotrebu od strane programera IS i, samim tim, imaju više grafičkih simbola od dijagrama za specifikaciju BP. Viši nivo detalja u dijagramima IS je potreban prilikom automatskog prevođenja modela u izvršni kod. Predloženi pristup u ovom radu zasnovan je na specifikaciji modela IS, koji bi bili opisani pomoću različitih namenskih jezika. Namenski jezici podržali bi odgovarajuće PIM koncepte koji bi bili specificirani na nivou odgovarajućih meta-modela.

U savremenim pristupima razvoja IS, smatra se da je neophodno značajnije oslanjanje na namenske jezike i MDE. Na primer, jezici za transformaciju modela su od suštinskog značaja za razvoj IS koji se zasniva na modelima. Neki od brojnih jezika koji pripadaju pomenutoj grupi su: ATL jezik za specifikaciju trasnsformacija (skraćeno ATL), *Query/View/Transformation* (QVT), *Kermeta* - jezik koji podržava modele transformacija i *Tefkat*. Iako podržavaju transformacije modela, postoje napori da se stvore još specifičniji jezici koji će biti usmereni na podršku transformacija modela koji pripadaju određenoj domenskoj oblasti. Ovi jezici su poznati kao jezici za specifikaciju transformacija u određenoj oblasti primene (DSTL). Irazábal et al. (2010) predlažu pristup razvoju takvih jezika pomoću jezika za opis transformacija, dok Reiter et al. (2006) predlažu primenu generičkog okruženja za opis jezika za specifikaciju modela transformacija u određenoj oblasti primene (DSMTL).

Funkcionalnost koja je posebno relevantna za IIS*Studio, kao i razvoj IS zasnovanog na modelima, je tzv. tekst–model interoperabilnost koja omogućava programerima da pretvore tekstualnu specifikaciju u odgovarajući model i obrnuto. Jedno takvo rešenje za namenske jezike definisano je pomoću *ATLAS Model Management Architecture* (AMMA). Rešenje predstavlja namenski jezik sa tekstualnom konkretnom sintaksom, koje omogućuje specifikaciju konkretnih sintaksa za odgovarajuće meta-modele (Jouault et al. 2006).

Još jedan značajan izazov u savremenim pristupima razvoja IS je efikasnost procesa razvoja samih namenskih jezika. Postoji veliki broj pristupa za meta-modelovanje i alata pogodnih za kreiranje namenskih jezika koji se uglavnom zasnivaju na specifikacijama putem sopstvenih meta-meta-modela. Jedan od alata je *Generic Modeling Environment* (GME) ("Generic Modeling Environment“), koji predstavlja podesiv alat za domensko modelovanje i generisanje programa, zasnovan na UML meta-modelu. Metaedit+ ("MetaCaseMetaEdit+") omogućuje specifikaciju meta-modela u grafičkom editoru koristeći model podataka *Object-Property-Role-Relationship*. *Eclipse Modeling Framevork* (EMF) ("Eclipse Modeling Framevork") je takođe često korišćen alat za meta-modelovanje, u kojem se koristi Ecore meta-meta-model u postupku kreiranja meta-modela. *Ecore* je *Eclipse* implementacija MOF 2.0 specifikacije u programskom jeziku Java. Postoje razni Eclipse dodaci pogodni za krerianje namenskih jezika. Svi dodaci su zasnovani na *Ecore* meta-meta-modelu i *EMF* okruženju. *EMFText* i *Xtext* su pogodni za stvaranje tekstualnih namenskih jezika. *Graphical Modeling Framework* (GMF) je alat koja se koristi za specifikaciju grafičkih namenskih jezika. Xtext alat generiše, pored parsera, model apstraktne sintakse i *Eclipse editor*. U dosadašnjem toku istraživanja autora, već je korišćeno EMF okruženje za specifikaciju IIS*Studio meta-modela pomoću MOF 2.0 specifikacije (Čeliković et al. 2011).

U ovom odeljku prezentovani su najznačajniji pomaci u pristupima razvoju IS, kako tradicionalnim, tako i onim, zasnovanim na modelima. Posebna pažnja posvećena je upotrebi namenskih jezika u kontekstu razvoja IS. Istraživanje dostupnih literaturnih izvora pokazuje da su metode razvoja IS evoluirale ka značajnijoj primeni namenskih jezika. Važan pogled na evoluciju softverskog inženjeringa dat je u radu Kurtev et al. (2006). Autori su u radu predstavili razloge zbog kojih MDE postaje generalizacija MDA. Oni navode da je inženjering uz pomoć namenskih jezika, koji je "pozicioniran na višem apstraktnom nivou, pri čemu koristi različita tehnička rešenja kao što su MDE, KSML, itd" (Str. 602) dalja generalizacija MDE. Jedan od pravaca dalje evolucije MDE obuhvata razvoj softverskih okruženja pod nazivom *Domain Specific Application Development Environment* (*DSADE*), koji podržavaju kreiranje i razvoj različitih aplikacija (France & Rumpe 2007). Ovakva okruženja sadrže skup alata koji pružaju mogućnosti za specifikaciju, razvoj, analizu i transformaciju modela u različite oblike iz kojih je moguće generisati izvršive aplikacije. Jedan takav primer je okruženje za brz razvoj prototipova *web* aplikacija HiperDe (Nunes & Schvabe 2006). Ovo okruženje kombinuje namenski jezik za opis specifikacija aplikacija sa pristupom razvoja koji je zasnovan na modelima i, pri tom, podržava stvaranje malih, ali složenih *web* aplikacija.

Predloženi pristup u ovom radu polazi od problema koji postoje u primeni tradicionalnih metoda za razvoj IS. Upotreba ovakvih metoda neretko dovodi do loše projektovane šeme baze podataka jer nije garantovano sagledavanje svih navedenih zahteva krajnjih korisnika. Projektanti i krajnji korisnici obično postaju svesni nedostataka rešenja, tek kada je završen razvoj šeme baze podataka. Kako bi se otklonili ili makar umanjili ovakvi nedostaci, napravljen je alterantivni pristup za razvoj IS koji je zasnovan na konceptu tipa forme (FT). Predloženi pristup zasnovan je na upotrebi metode za razvoj softvera zasnovanoj na modelima (MDSD) i namenskim jezicima (DSL).

3. Pregled karakteristika i mogućnosti okruženja IIS*Studio

IIS*Studio predstavlja razvojno okruženje koje je nastalo kao rezultat višegodišnjeg istraživanja iz oblasti projektovanja i automatskog generisanja šema baza podataka i transakcionih programa. IIS*Studio podržava metodološki pristup razvoju IS, zasnovan na konceptu tipa forme i služi u postupku projektovanja informacionih sistema i odgovarajuće šeme baze podataka, gde kao krajnji proizvod nastaje funkcionalni prototip informacionog sistema.
Trenutna verzija okruženja IIS*Studio sadrži dva glavna alata:

- IIS*Case - alat za projektovanje informacionih sistema,
- IIS*UIModeler - alat za modelovanje šablona korisničkog interfejsa aplikacija informacionih sistema, i
- IIS*Ree - alat namenjen procesu reinženjeringa informacionih sistema

Alat IIS*UIModeler podržava postupak specifikacije šablona korisničkog interfejsa aplikacija koje krajnji korisnik može da koristi u radu 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 Banovic 2010.

IIS*Case je alat zasnovan na MDSD pristupu. Alat pruža visok nivo automatizacije procesa *forward* inženjeringa informacionih sistema. U daljem tekstu dat je opis IIS*Case alata.

U ovom poglavlju, najpre su ukratko prikazane osnovne mogućnosti i funkcionalnosti alata IIS*Case. Zatim, su opisani osnovni koncepti alata, pri čemu je posebna pažnja posvećena definiciji i opisu koncepta tipa forme, kao glavnog koncepta za modelovanje. Nakon toga su opisane faze *forward* inženjeringa koji kao krajnji rezultat daje implementacioni opis relacione šeme baze podataka i programski kod ekranskih formi transakcionih programa.

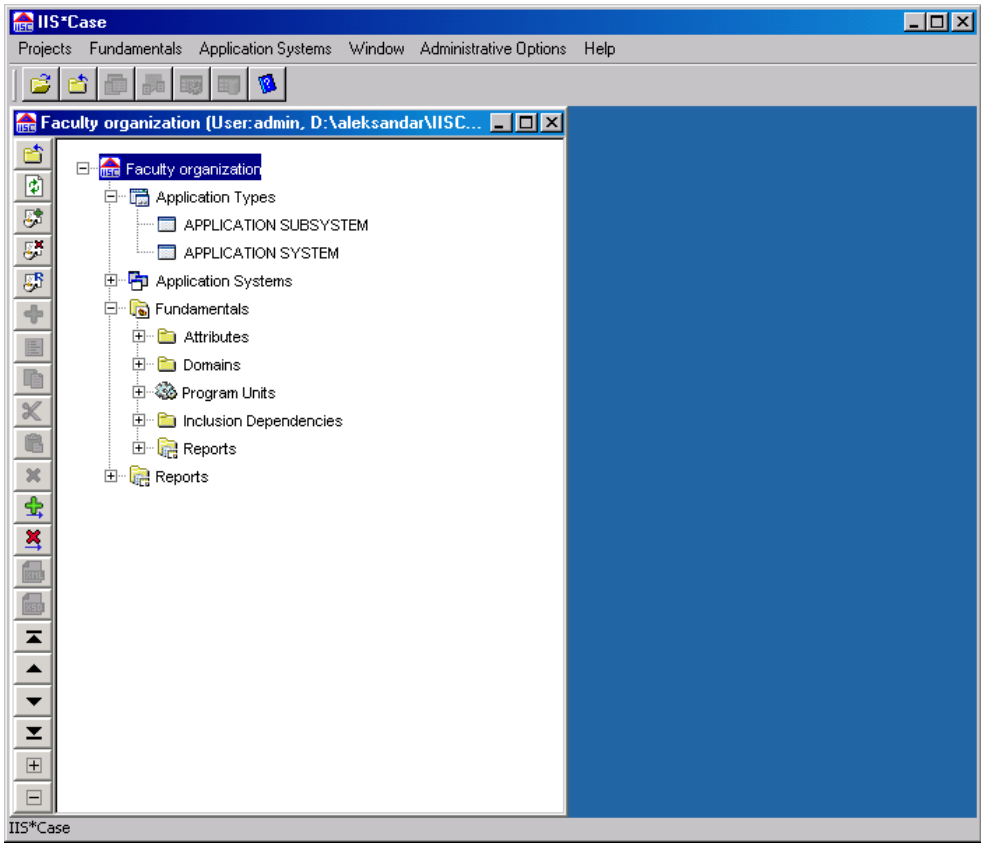
3.1 Alat IIS*Case

Glavni koncepti i formalizmi na kojima je zasnovan metodloški pristup i alat IIS*Case razvijani su od kraja osamdesetih godina na Fakultetu tehničkih nauka u Novom Sadu. U različitim programskim okruženjima razvijeno je nekoliko verzija alata IIS*Case. Prva verzija IIS*Case-a bila je zasnovana na tipovima formi i skupu funkcionalnih zavisnosti, koji predstavlja polazni skup ograničenja. Daljim razvojem alata, skup ogrnaičenja je proširen i nefunkcionalnim odnosima i tzv. specijalnim funkcionalnim zavisnostima.

Prvobitno, IIS*Case je razvijen da podrži automatsko projektovanje šema baza podataka. Želja da se pomogne projektantu u prevazilaženju problema definisanja ograničenja baza podataka predstavlja motiv razvoja ovakvog alata. Ovo je moguće podržati automatskim generisanjem uz minimalnu interakciju sa projektantom. Glavni zadatak projektanta je da specificira pravila korišćenja podataka u realnom sistemu na ispravan način, a alat ima zadatak 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 definisani su potpuno nezavisno od šeme relacije kojoj pripadaju. Daljim razvojem alata, skup koncepata ugrađenih u alat IIS*Case se proširuje, tako da je na sadašnjem stepenu razvoja moguće specificirati većinu aspekata jednog informacionog sistema.

Poslednja verzija alata implementirana je u programskom jeziku *Java* i na slici 3.1 prikazana je glavna forma ovog alata. Na sadašnjem stepenu razvoja alat podržava:

- konceptualno projektovanje šeme baze podataka, transakcionih programa i informacionog sistema,
- automatizovano projektovanje podšema relacione baze podataka u trećoj normalnoj formi
- automatizovanu integraciju podšema u jedinstvenu šemu baze podataka u trećoj normalnoj formi,
- automatizovanu detekciju formalnih kolizija ograničenja,
- generisanje SQL opisa šeme baze podataka za različite sisteme za upravljanje bazama podataka,
- generisanje XML specifikacije informacionoog sistema,
- projektovanje aplikacija informacionog sistema i vizuelnih atributa ekranskih formi,
- reverzni inženjering relacionih baza podataka u modele FT,
- zadavanje *check* ograničenja, transformacija i generisanje SQL koda za proveru važenja tih ograničenja na nivou konceptualne šeme baze podataka i
- generisanje programskog koda ekranskih formi transakcionih programa.



Slika 3.1. Glavna forma alata IIS*Case

U pristupu na kojem se IIS*Case zasniva striktno je razdvojena specifikacija sistema i implementacija na konkretnoj platformi. Postupak modelovanje se obavlja na visokom nivou apstrakcije, jer projektant kreira model informacionog sistema putem koncepata iz problemskog domena. Tokom ovog postupka projektant nema potrebu da navodi detalje vezane za implementacionu platformu. Ovakav model informacionog sistema koji je na viskom stepenu nezavisnosti od platforme (PIM), može se koristiti u nizu transformacija modela u alatu IIS*Case, nakon čega se može dobiti više modela sa različitim stepenom

zavisnosti od platforme (PSM). Model koji je potpuno specifičan za platformu predstavlja konkretni implementaciono orijentisan opis informacionog sistema ili implementacioni opis šeme baze podataka za izabranog proizvođača.

Nekoliko alata je integrisano u alat IIS*Case. Jedan deo njih pruža mogućnost za rad sa model-u-model transformacijama, dok drugi deo alata omogućavaju model-u-kod transformacije. IIS*Case obezbeđuje konceptualno modelovanje šeme baze podataka na apstraktnom nivou PIM-a, koje uključuje specifikaciju raznih vrsta ograničenja baze podataka, kao što su: ograničenje nula vrednosti, ograničenje domena, ograničenje ključa i jedinstvenosti, c*heck* ograničenje, kao i razne tipove ograničenja zavisnosti sadržavanja. Ovakav model se dalje automatski transformiše u model relacione baze podataka, koji je jednim delom platformski zavistan jer je zasnovan na relacionom modelu, ali je nezavistan od konkretnog proizvođača. Dobijeni model može se dalje transformisati model-u-kod transformacijom u SQL opis šeme baze podataka. Dobijeni SQL opis šeme baze podataka je u skladu sa standardom i može da ima određeni stepen implementacione nezavisnosti ali može da predstavlja i SQL opis šeme baze podataka za izabranog proizvođača koji predstavlja implementacioni opis šeme baze podataka sa stepenom 0% platformske nezavisnosti. Ovakav pristup razvoju informacionih sistema saglasan je sa MDSD pristupom razvoja (Lukovic 2008).

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, od kojih su neki: Luković 2007, Luković 2006, Aleksić 2007, Aleksić 2011, Banović 2010.

3.2 Platformski nezavisni koncepti u alatu IIS*Case

IIS*Case je alat koji 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 podskupa:

- osnovni koncepti koji uključuju: domen, atribute, funkcije, pakete i događaje; i
- koncepti specifični 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 Luković 2010.
Paket predstavlja kolekciju određenih funkcija koje su definisane u IIS*Case repozitorijumu. Koncept funkcije može se koristi za specifikaciju kompleksnih funkcionalnosti. Funkcije u IIS*Case-u definišu se na nivou projekta i mogu biti upotrebljene u različitim IIS*Case specifikacijama. Koncept događaja koristi se na nivou platformski nezavisnog modela. Koncept događaja služi da bi se predstavio bilo koji softverski događaj koji može pokrenuti neku akciju pod određenim uslovima. Događaji mogu biti:

- događaji baze podataka (trigeri i izuzeci),
- događaji aplikativnog servera (događaj tastature, događaj miša i izuzeci) i
- događaji klijenta (događaj tastature, događaj miša i izuzeci).

Svaki događaj trebalo bi da bude povezan sa softverskom specifikacijom na nivou platformski nezavisnog modela. Koristeći koncet zavisnosti sadržavanja projektant može da definiše ograničenja univerzalne šeme relacije tipa: skup vrednosti niza atributa je podskup skupa vrednosti drugog niza atributa. Ovakva ograničenja mogu da dovede do pojave odgovarajućih međurelacionih ograničenja implementacione šeme. Preimenovani atributi su specijalni slučajevi zavisnosti sadržavanja i definisanje preimenovanja je analogno zadavanju odgovarajuće zavisnosti.

Primena alata IIS*Case polazi od pretpostavke o postojanju univerzalne šeme relacije. Zbog ovog razloga definicija atributa, pri projektovanju, je nezavisna od šeme relacije kojoj će atribut pripadati. Pre same definicije atributa neophodno je obaviti definisanje potrebnih domena i funkcija.
Projektant je u mogućnosti da definiše naredne osobine za atribute definisane na nivou osnovnih koncepata:

- domena atributa,
- 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 narednom poglavlju.
Koncept projekta koji ima strukturu tipa stabla sadrži sve projektantske specifikacije vezane za jedan informacioni sistem. Aplikativni sistem je osnovni činilac svakog projekta. Projekat može da sadrži više aplikativnih sistema. Aplikativni sistem može da referencira više drugih aplikativnih sistema, koji na taj način postaju njegovi potomci, tj. podsistemi.

Specifikacije koje su vezane za projekat čuvaju se u jedinstvenom repozitorijumu okruženja IIS*Studio. Detaljnije informacije o strukturi repozitorijuma mogu se naći u Banović 2010. ODBC protokol se koristi za komunikaciju sa repozitorijumom, čime je omogućena konekcija na različite sisteme za upravljanje bazama podataka.

U IIS*Case-u, glavni koncept za modelovanje, koji je platformski nezavisan, jeste *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 poglavlju.

3.3 Tip forme

Tip forme je glavni koncept pristupa i alata IIS*Case. Tip forme predstavlja model ekranskih ili izveštajnih formi koje korisnik koristi u postupku komunikacije sa informacionim sistemom. Pomoću IIS*Case alata projektant specificira ekranske ili izveštajne forme transakcionih programa, čime indirektno, definiše inicijalni skup atributa i ograničenja.

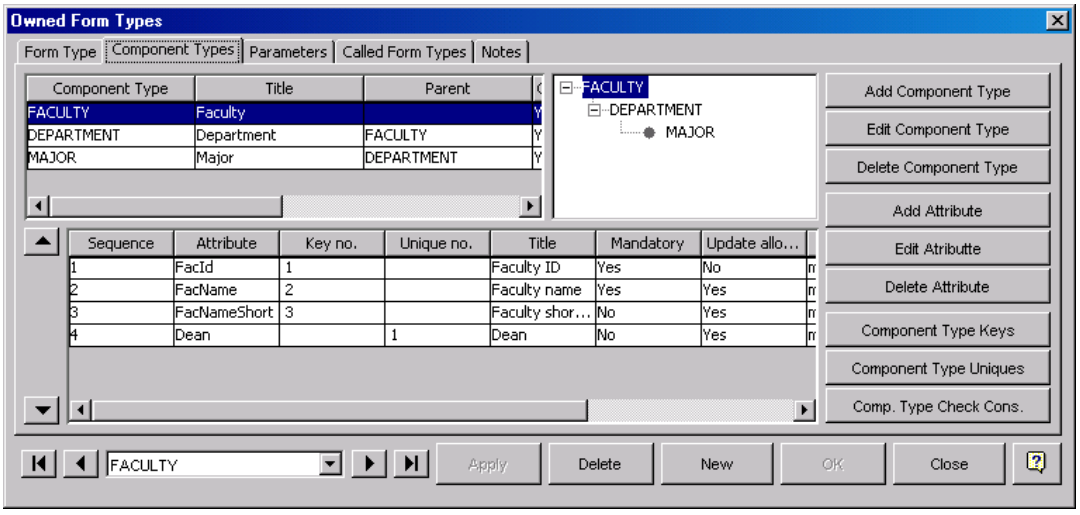
Tradicionalni pristup u razvoju IS podrazumeva projektovanje šeme baze podataka, a zatim i implementaciju ekranskih formi transakcionih programa. Koristeći IIS*Case alat, projektant, putem definiše specifikacije vezane za tipove forme,čime istovremeno specificira šemu

buduće baze podataka, kao i strukturu ekranskih formi koje će biti korišćene za unos, brisanje i izmenu podataka nad tom bazom. U postupku projektovanja projektant ne mora da poseduje napredna znanja, koja su neophodna za projektovanje relacionih baza podataka. Projektant treba dobro da specificira pravila po kojima se koriste podaci jednog realnog sistema, a alat je u stanju da takve specifikacije transformiše u opis implementacione šeme baze podataka sa svim odgovarajućim ograničenjima.

Koncept tipa forme je dovoljno semantički bogat da omogući specifikaciju elemenata statičke strukture realnog sistema, kao i semantike, međuzavisnosti komponenata i pravila poslovanja. Pomoću koncepta tipa forme, postupak projektovanja baze podataka u IIS*Case-u, 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 specificiran kao *menu* ili *program*. Ukoliko je tip forme definisan kao *menu*, u takvoj situaciji taj tip forme u budućem generisanju aplikacije predstavlja osnovu za generisanje menija ekranskih formi. Ako je tip forme označen kao program, onda će taj tip forme biti osnova za generisanje ekranskih formi transakcionih programa. Na slici 3.2 prikazana je ekranska forma koja se koristi za specifikaciju 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. Ukoliko je tip forme specifiran da učestvuje u dizajnu, onda će biti uključen u skup tipova formi koji se koristi pri generisanju implementacione šeme baze podataka.



Slika 3.2. Forma za definisanje tipa forme

Tip forme je specificiran kao imenovana struktura tipa stabla. Čvorovi ovakvog stabla se nazivaju tipovi komponenti. Svaki tip komponente ima jedinstven naziv na nivou tipa forme koji ga sadrži, kao i neprazne skupove atributa i ključeva. Za odgovarajući tip komponente je potrebno definisati i skup operacija koje su dozvoljene nad tim tipom komponente. U okviru aplikativnog sistema projektant može da definiše nove, menja i briše postojeće tipove formi, kopira ih i premešta u druge aplikativne sisteme tekućeg projekta. Jedan aplikativni sistem može da referencira tipove formi koji pripadaju drugim aplikativnim sistemima. Tip forme

koji je specificiran na ovaj način biće ravnopravno uključen u kreiranje polaznog skupa ograničenja.

Atributi iz univerzalnog skupa atributa mogu biti pridruženi tipovima komponenti. Specifikacija atributa na tipu komponente zahteva izbor atributa iz skupa u kojem su definisani svi atributi i definisanje dozvoljenih operacija nad datim atributom. Jedan atribut može da pripada najviše jednom tipu komponente istog tipa forme, a može biti uključen i u više različitih tipova formi. Na nivou tipa komponente može se definisati i redosled atributa, podrazumevana vrednost atributa i ponašanje atributa na tipu komponente. Za svaki tip komponente projektant zadaje osnovne podatke kao što su:

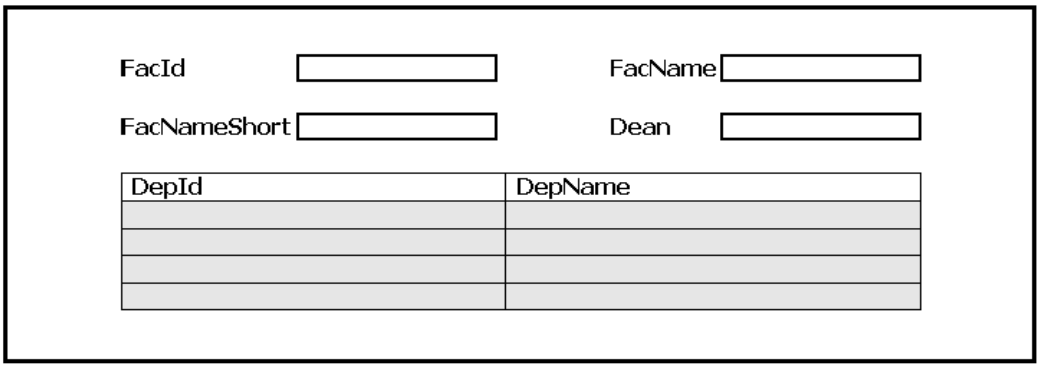
- ime,
- nadređeni tip komponente,
- dozvoljene operacije,
- kardinalitet u okviru nadređenog tipa komponente,
- atributi za vizualizaciju i
- grupe polja.

Jedan tip komponente specifiran je kao korenski, a svi ostali definišu se kao podređeni tom tipu komponente u hijerarhijskoj strukturi tipa forme. Svaki tip komponente može posedovati proizvoljan broj podređenih tipova komponenti. Izborom jedne od opcija *0-N* i *1-N* može se definisati kardinalnost pojave tipa komponente u okviru pojave nadređenog tipa komponente. Ova činjenica posledično ima uticaja na ograničenja u implementacionoj šemi baze podataka.

Na nivou tipa komponente, mogu se definisati ograničenja ključa i ograničenja jedinstvene vrednosti. Ključ predstavlja neprazan skup atributa putem kojeg se jedinstveno identifikuje pojava tog tipa komponente u okviru jedne pojave nadređenog tipa komponente. U okviru jednog tipa komponente moguće je definisati proizvoljan broj ključeva. Jedan atribut može učestvovati u proizvoljnom broju ključeva. Atribut koji nije specificiran da učestvuje u kreiranju implementacione šeme baze podataka, ne može biti deo nekog ključa. Za svaki tip komponente koji je definisan kao korenski projektant mora definisati makar jedan ključ, koji ga jedinstveno identifikuje u okviru jedne pojave njemu nadređenog tipa komponente. Za tip komponente koji je specificiran kao podređen, ključ se definiše kao unija po jednog ključa od korenskog do datog tipa komponente.

Ograničenja jedinstvene vrednosti definišu se na sličan način. Jedno ograničenje jedinstvene vrednosti predstavlja neprazan skup atributa. Na niovu jednog tipa komponente nisu postavljena nikakva ograničenja vezana za broj ograničenja jedinstvene vrednosti koja će biti definisana.

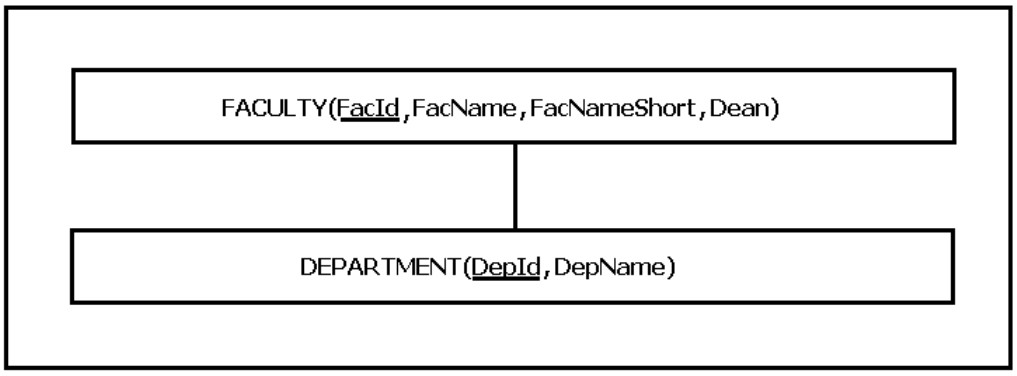
Primer 3.1. U ovom primeru prikazano je na koji način je moguće pomoću alata IIS*Case na osnovu dokumenata koji se koriste u realnom sistemu, specificirati opis tipova formi i tipova komponenti.



Slika 3.3. Dokument koji sadrži informacije o fakultetima

Na slici 3.3 prikazan je jedan dokument koji služi za evidenciju fakulteta i njegovih departmana. Dokument je vizuelno podeljen na dva dela. U gornjem delu dokumenta evidentiraju se osnovne informacije o fakultetu, dok se u tabelu u donjem delu unose podaci o departmanima koji pripadaju tom fakultetu.

Na osnovu ovog dokumenta, projektant vrši specifikaciju odgovarajućeg tipa forme. Tip forme koji reprezentuje navedeni dokument sadrži dva tipa komponente *Faculty* i *Department*. Tip komponente *Faculty* je korenski tip komponente i nadređen je tipu komponente *Department*. Na slici 3.4 dat je grafički prikaz navedenih tipova komponenti.



Slika 3.4. Specifikacija tipa forme

Atributi tipova komponenti koji su deo ključa su podvučeni. Atribut *FacId* jedinstveno identifikuje svaku pojavu tipa komponente *Faculty*. Jedna pojava tipa komponente *Department* jedinstveno je identifikovana pomoću atributa *DeptId* u okviru jedne pojave tipa komponente *Faculty*.

3.4 Forward inženjering u alatu IIS*Case

Projektovanje šeme baze podataka prethodi specifikaciji ekranskih formi i izveštaja transkacionih programa u tradicionalnim pristupima projektovanja IS. Za razliku od ovakvih pristupa, u alatu IIS*Case projektant najpre treba da definiše ekranske forme čime indirektno specificira incijalni skup atributa i ograničenja. Skup tipova formi predstavlja platformski nezavisan pogled na IS iz perspektive krajnjeg korisnika. Zbog ove osobine model IS-a specificiran putem alata IIS*Case i reprezentovan skupom specifikacija tipova formi, može biti klasifikovan kao PIM.

IIS*Case u svom postupku projektovanja IS koristi skup tipova formi za generisanje relacionih šema baza podataka i njegov graf zatvaranja. Definisanjem tipova formi, projektant zapravo u isto vreme specificira:

- šemu baze podataka,
- funkcionalne karakteristike budućeg transakcionog programa i
- izgled interfejsa za krajnjeg korisnika.

Na osnovu definicije tipova formi, alat IIS*Case može sam da izvede definiše ograničenja, kandidate za primarne ključeve, itd. čime se štedi značajno vreme potrebni za projektovanje IS.

U postupku *forward* inženjeringa putem alata IIS*Case 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.

3.4.1 Konceptualno modelovanje

Konceptualno modelovanje je korak koji se vrši kreiranjem skupova tipova formi. Neophodno je da projektant kreira novi projekat koji sadrži bar jedan aplikativni sistema, pre nego što započne postupak specifikacije tipova formi. Naredni korak u konceptualnom modelovanju IS-a je specifikacija osnovnih koncepata, koji su opisani u odeljku 3.2. Osnovne koncepte projektant definiše na nivou projekta, nezavisno od aplikativnog sistema.

U narednom koraku, projektant specifira tipove formi na osnovu dokumenata iz realnog sistema. Projektat treba na korektan način da predstavi pravila upotrebe podataka u realnom sistemu pomoću koncepta tipa forme. Koncept tipa forme predstavlja osnovu na kojoj su definisani algoritmi u alatu IIS*Case za generisanje početnog skupa atributa i izdvajanja funkcionalnih zavisnosti kao ulaznih parametra u postupku generisanja šeme baze podataka.

Na nivou tipa forme specificiraju se tipovi komponenti i njihova hijerarhijska struktura tipa stabla kao što je prikazano na slici 3.2. Jedan tip komponente je korenski, u vrhu stabla, a svi ostali su hijerarhijski ispod njega. Za svaki tip komponente može biti definisan proizvoljan broj podređenih tipova komponenti.

Definisanje atributa na tipu komponente predstavlja postupak izbora atributa iz skupa svih atributa koji su definisani kao osnovni koncepti i specifikaciju dozvoljenih akcija nad datim atributom. Jedan atribut iz osnovnih koncepata može biti uključen na različitim tipovima formi, na nivou tipova komponenti, ali ne sme da pripada različitim tipovima komponenti istog tipa forme. Naredni korak predstavlja definiciju ključeva i ograničenja jedinstvene vrednosti na tipu komponente.

Na osnovu definicije tipova formi, može se identifikovati polazni skup ograničenja sledećih tipova:

- funkcionalne zavisnosti,
- nefunkcionalni odnosi,

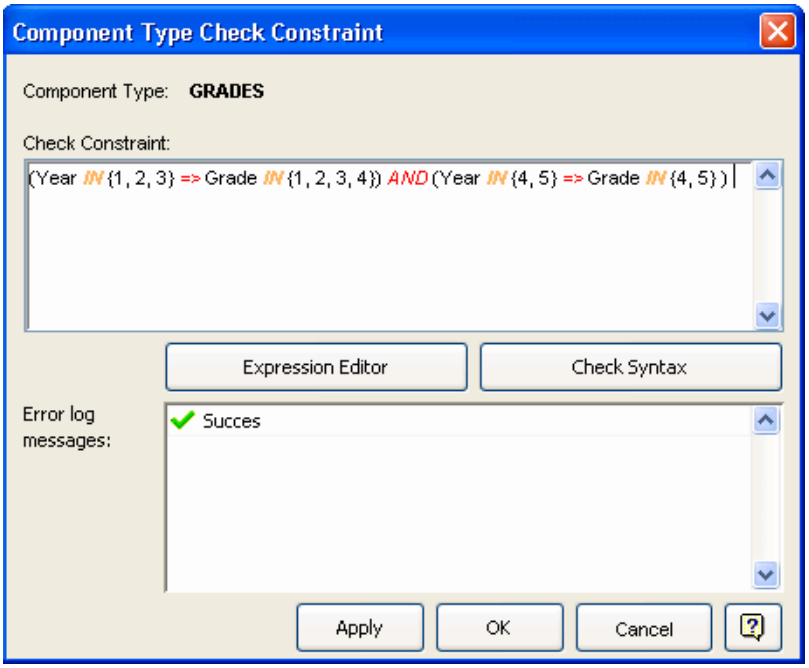
- ugrađene zavisnosti sadržavanja i
- specijalne funkcionalne zavisnosti sa ugrađenim ograničenjima jedinstvene vrednosti.

Alat IIS*Case takođe poseduje koncepte i alate putem kojih projektant može da definiše izraze ograničenja torki prilikom modelovanja konceptualne šeme baze podataka. Postupak ER modelovanja ne dozvoljava definiciju ovakve vrste ograničenja. Za definiciju ograničenja torke razvijen je namenski jezik putem kojeg je moguće definisati ova ograničenja. Glavna namena 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 3.5 prikazana je forma za specifikaciju izraza ograničenja tipa komponente. Pored namenskog jezika, razvijeni su i algoritmi putem kojih je moguće izvršiti generisanje programskog koda na osnovu specifikacije ograničenja torke. Više o ovim transformacijama može se naći u Obrenovic et al. 2012.

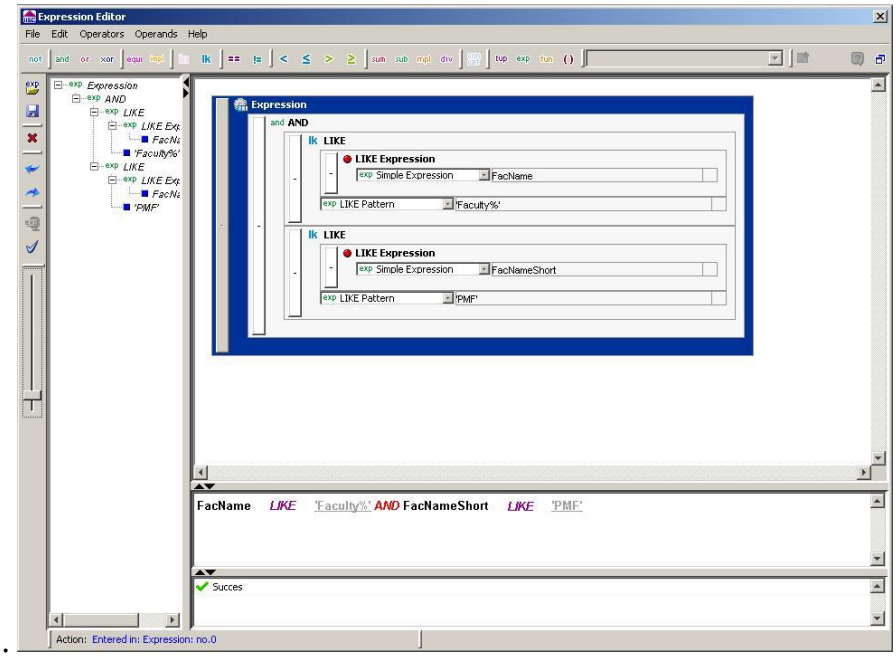
Alat *Expession Editor* namenjen je za definisanje ograničenja torke. Putem ovoga alata izrazi se mogu specificirati na dva načina:

- putem tekstualne sintakse i
- putem grafičke sintakse.

Pored sintaksne analize, alat poseduje i deo koji vrši semantičku analizu. Na primer, za izraze zadate na nivou domena koji je tipa sloga, kada je referencirano polje sloga, vrši se provera da li je polje validno. Alat ne vrši proveru tipova, jer model ne sadrži informaciju o tome da li je neki operator definisan za zadate domene. Na slici 3.6 prikazana je glavna forma ovog alata. Sledeći korak u postupku *forward* inženjeringa je generisanje šeme baze podataka.



Slika 3.5. Forma za specificiranje izraza ograničenja

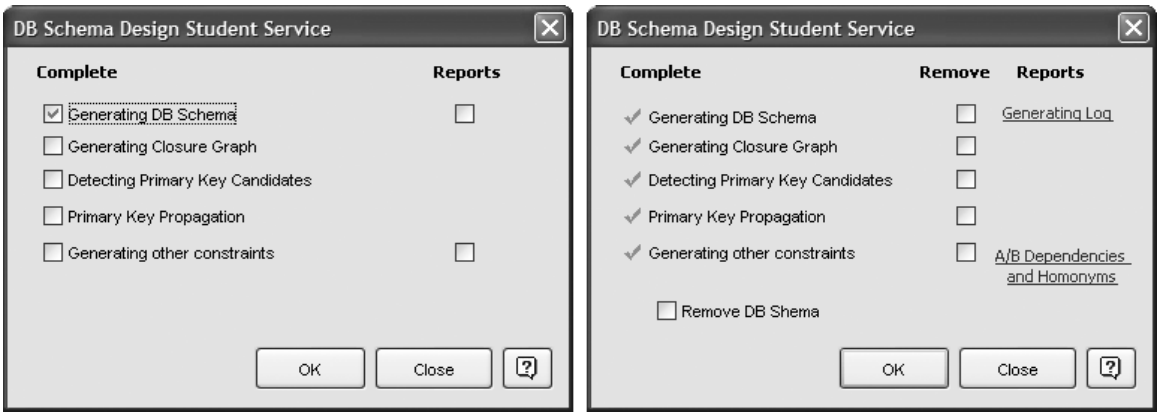


Slika 3.6. Alat Expession Editor

3.4.2 Generisanje šeme baze podataka

Potupak generisanja šeme baze podataka aplikativnog sistema na osnovu polaznog skupa ograničenja izvodi se kroy 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 3.7. Forma za kontrolu procesa generisanja šeme baze podataka

Postupak izvođenja ovih koraka je automatizovan, u određenom redosledu, na zahtev projektanta. Ulazna specifikacija u ovaj postupak je unija skupova tipova formi aplikativnog sistema koji je izabran i svih njegovih aplikativnih podsistema. Na slici 3.7 prikazana je IIS*Case forma koju projektant koristi za kontrolu procesa generisanja šeme baze podataka. Forma na levoj strani slike prikazuje slučaj kada nisu svi koraci kompletirani, dok ista forma, prikazana na slici sa desne strane, prikazuje slučaj kada je čitav postupak kompletiran.

U prvom koraku izvodi se algoritam sinteze, koji je značajno unapređen, specijalno za potrebe praktične primene u pristupu i alatu IIS*Case. U toku ovog koraka, konceptualni model prevodi se u relacioni model podataka. Postupak transformacija započinje izvođenjem skupa funkcionalnih, nefunkcionalnih i specijalnih funkcionalnih zavisnosti iz svih tipova formi uključenih u ulaznu specifikaciju. Kao rezultat ovog koraka dobija se skup šema relacija sa definisanim skupovima atributa, sintetizovanim ključevima i ograničenjima jedinstvene vrednosti. Osnovni algoritam sinteze u alatu IIS*Case je proširen novim koracima koji, omogućuju generisanje i sintetiziovanih i nesintetizovanih ključeva.

Drugi korak omogućuje generisanje grafa zatvaranja koji predstavlja grafičku reprezentaciju generisane šeme. Graf zatvaranja je posebno važan za vizuelizaciju međurelacionih ograničenja. Svaki čvor grafa zatvaranja predstavlja šemu relacije, koja je generisana pomoću algoritma sinteze. Svaka direktna veza između čvorova predstavlja činjenicu da je pravi ili nepravi podskup ključa podređenog čvora propagiran kao strani ključ u nadređeni čvor.

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

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

U narednom, petom koraku izovde 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.

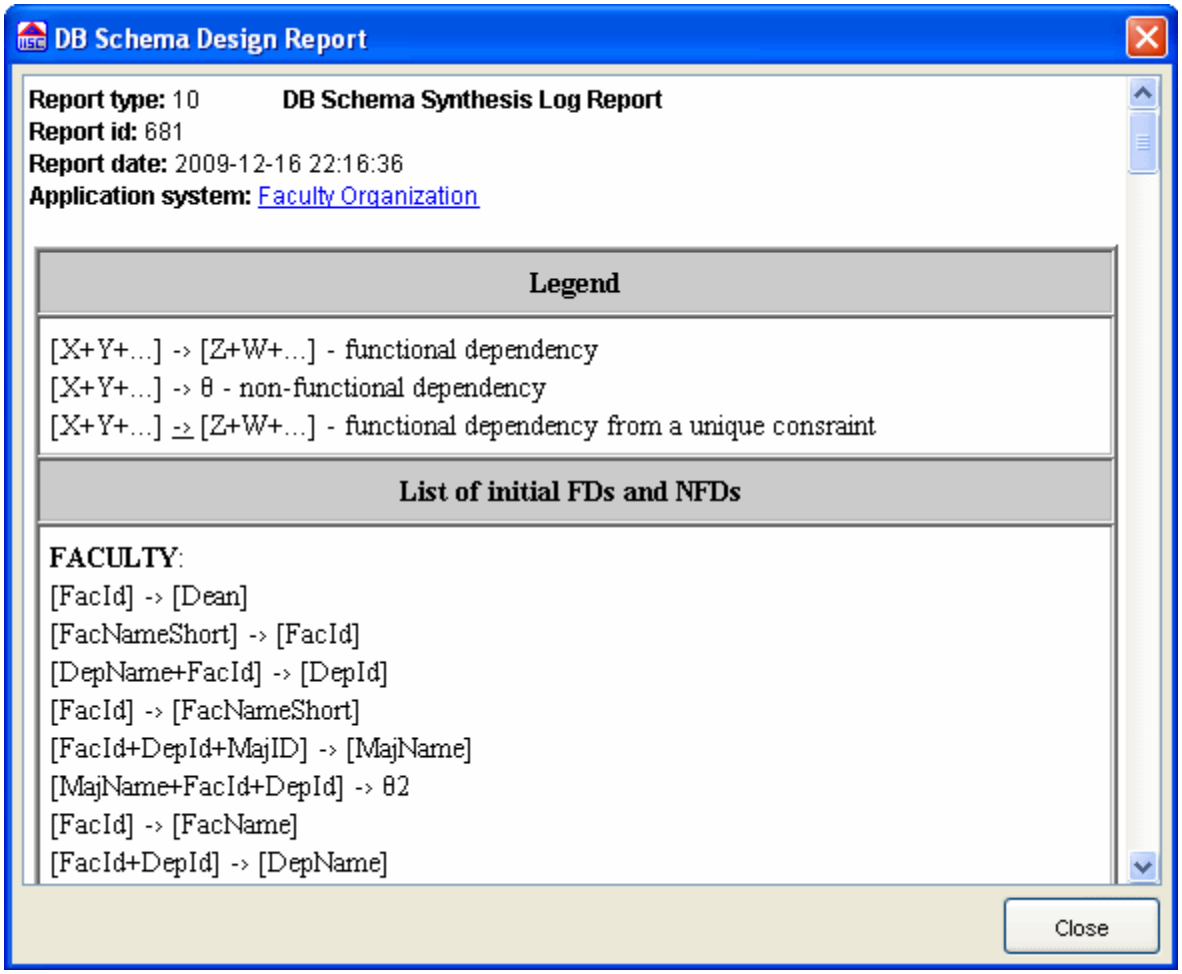
Osnovna i proširena ograničenja referencijalnog integriteta generišu se na osnovu grafa zatvaranja. Postojanje ovih tipova ograničenja je posledica propagacije primarnog ključa. Alat IIS*Case će generisati ograničenje osnovnog referencijalnog integirteta, ukoliko se ceo primarni ključ prostire iz podređene (referencirane) šeme relacije u nadređenu (referencirajuću),. U suprotnom alat detektuje kao ograničenje proširenog referencijalnog integirteta.

Ograničenja referencijalnog integriteta koja su zasnovana na netrivijalnim zavisnostima sadržavanja nastaju od netrivijalnih zavisnosti sadržavanja koje projektant specificira na nivou skupa svih atributa informacionog sistema. Ograničenje inverznog referencijalnog integriteta nastaje od komponenti tipova forme. Ukoliko je tip komponente N_i direktno podređen tipu komponente N_j, tada projektant ima mogućnost da specificira da svaka instanca N_i mora biti povezana sa najmanje jednom instancom od N_j. U toj situaciji alat 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.

Alat IIS*Case obezbeđuje i automatsko generisanje odgovarajućih izveštaja nakon prvog i petog koraka procesa generisanja. Slika 3.8 sadrži prikaz izveštaja koji je generisan tokom

sinteze (prvog koraka). Prikazani izveštaj uključuje informacije o svim transformacijama koje su urađene nad polaznim skupom ograničenja.

Putem prethodnih koraka obezbeđeno je generisanje potencijalne šeme baze podataka izabranog aplikativnog sistema. Ako je aplikativni sistem specifiran istovremeno i kao 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. Ova osobina ne znači da će šema baze podataka aplikativnog sistema biti rezultat proste unije podšema njegovih podsistema. Prosto uniranje može izazvati kolizije i redudantnost podataka pa se podšeme integrišu jednim složenim procesom,. Skup tipova formi izabranog aplikativnog sistema, tj konceptualna šema baze podataka tog sistema, u postupku generisanja i konsolidacije, alat IIS*Case transformiše u implementacionu relacionu šemu baze podataka. Sledeći korak u postupku *forward* inženjeringa je konsolidacija šema baze podataka.



Slika 3.8. Izveštaj o rezultatima i međurezultatima procesa sinteze

3.4.3 Konsolidacija (usaglašavanje) šema baze podataka

Svaka podšema koja odgovara podsistemu aplikativnog sistema mora biti formalno konzistentna sa šemom baze podataka celog sistema, tj. mora postojati usaglašenost svih relacionih i međurelacionih ograničenja. Ako se podšeme integrišu u neku šemu baze podataka, neophodno je u postupku projektovanja nakon integracije proveriti konzistentnost

na višem nivou. Usaglašenost koncepata podšeme sa konceptima potencijalne implementacione šeme mora biti zadovoljeno, da bi se moglo smatrati da su u implementacionoj šemi pravilno integrisane sve njene podšeme. Ovo znači da mora važiti sledeće:

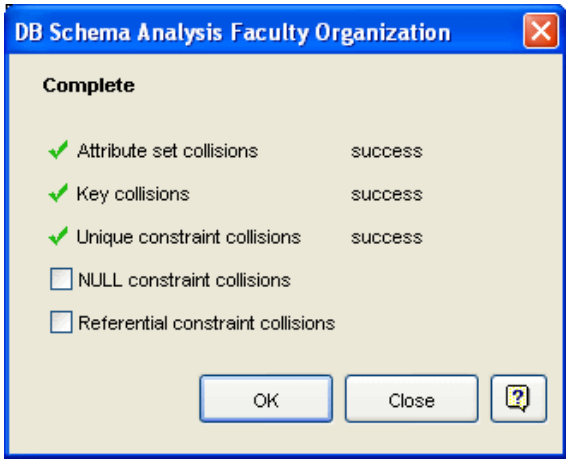
- za svaku šemu relacije podšeme, postoji šema relacije integrisane šeme, takva da je skup obeležja šeme relacije podšeme podskup skupa obeležja šeme relacije implementacione šeme i primarni ključ šeme relacije podšeme je jednak primarnom ključu korespondentne šeme relacije integrisane šeme;
- sva relevantna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

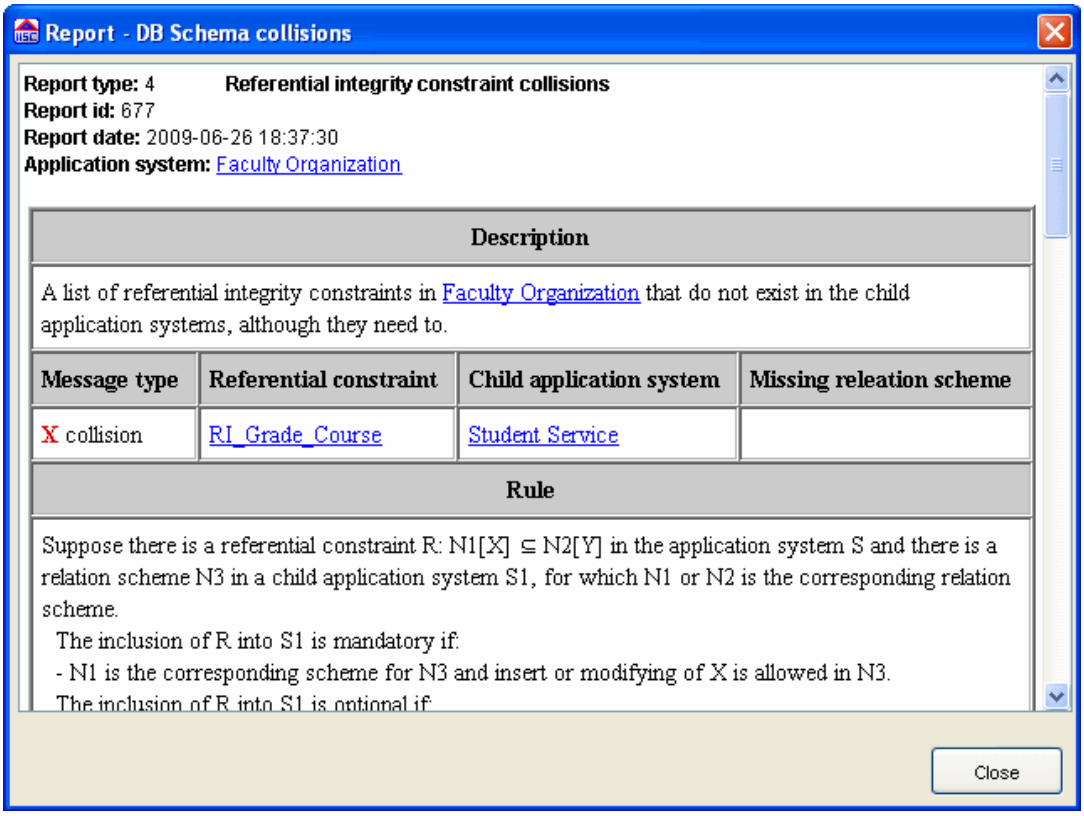
Proces konsolidacije šeme baze podataka sa podšemama podeljen je na korake koji usaglašavaju pojedinačne grupe ograničenja istog tipa. Proces se sastoji od sledećih koraka:

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 u kojem se izvršavavaju ovi koraci je bitan, jer uspešnost provere jedne grupe ograničenja može usloviti mogućnost provere druge grupe. Zbog ovog razloga korake nije moguće izvršiti u proizvoljnom redosledu. Na slici 3.9 prikazana je ekranska forma preko koje projektant zadaje naredbe za izvršavanje pojedinačnih koraka algoritma konsolidacije.



Slika 3.9. Forma za analizu usaglašenosti šeme baze podataka



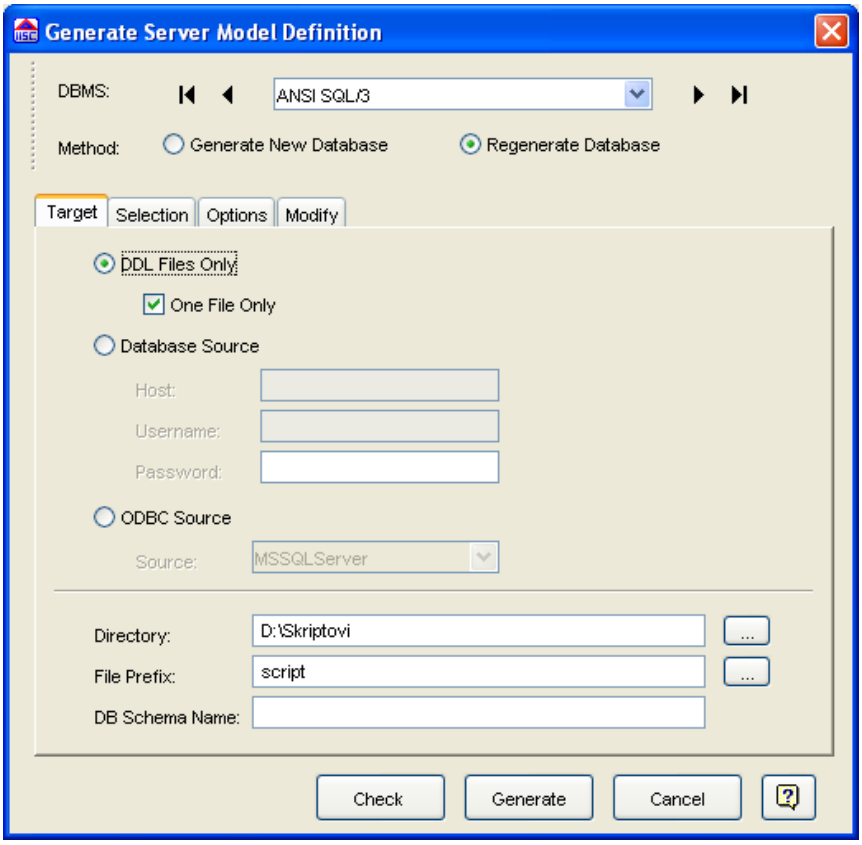
Slika 3.10. Izveštaj o kolizijama ograničenja referencijalnog integriteta

U ovom koraku alat takođe generiše izveštaje u kojima su detaljno opisane neusaglašenosti podšema, koje se odnose na skup ograničenja koji se u datom koraku usaglašava. Svi generisani izveštaji dostupni su projektantu, tako da je moguće pristupiti istoriji promena u svakom trenutku. Pretraga po tipu i sadržaju izveštaja olakšava pregledanje, u situaciji velikog broja generisanih izveštaja. Na slici 3.10 dat je jedan primer izveštaja neusaglašenosti skupova ograničenja referencijalnih integriteta.

Proces projektovanja šeme baze podataka je iterativan. Svaka iteracija predstavlja postupak kreiranja 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, sve dok se ne dobije usaglašena šema baze podataka. Naredni korak u postupku *forward* inženjeringa je generisanje implementacionog opisa šeme baze podataka.

3.4.4 Generisanje implementacionog opisa šeme baze podataka

Alat IIS*Case sadrži ugrađen SQL generator (slika 3.11) pomoću kojeg je moguće implementirati internu šemu baze podataka za izabrani sistem za upravljanje bazama podataka koji je podržan od strane alata. SQL generator obezbeđuje automatsko generisanje SQL koda pri čemu koristi specifikaciju nastalu projektovanjem IS. 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 ima mogućnost da i bez poznavanja sintakse SQL-a kao i mehanizama za implementciju ograničenja, generiše SQL kod za kreiranje tabela, pogleda, indeksa, procedura, funkcija i okidača.



Slika 3.11. Forma SQL generatora

U ovoj realizaciji, SQL generator pruža 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 zaokružen je proces projektovanja šeme baze podataka.

Primer 3.2. Na osnovu tipa forme koja je prikazana na slici 3.4 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*. 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 3.1.

```
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  )
```

Slika 3.12. Primer *generisanog SQL koda*

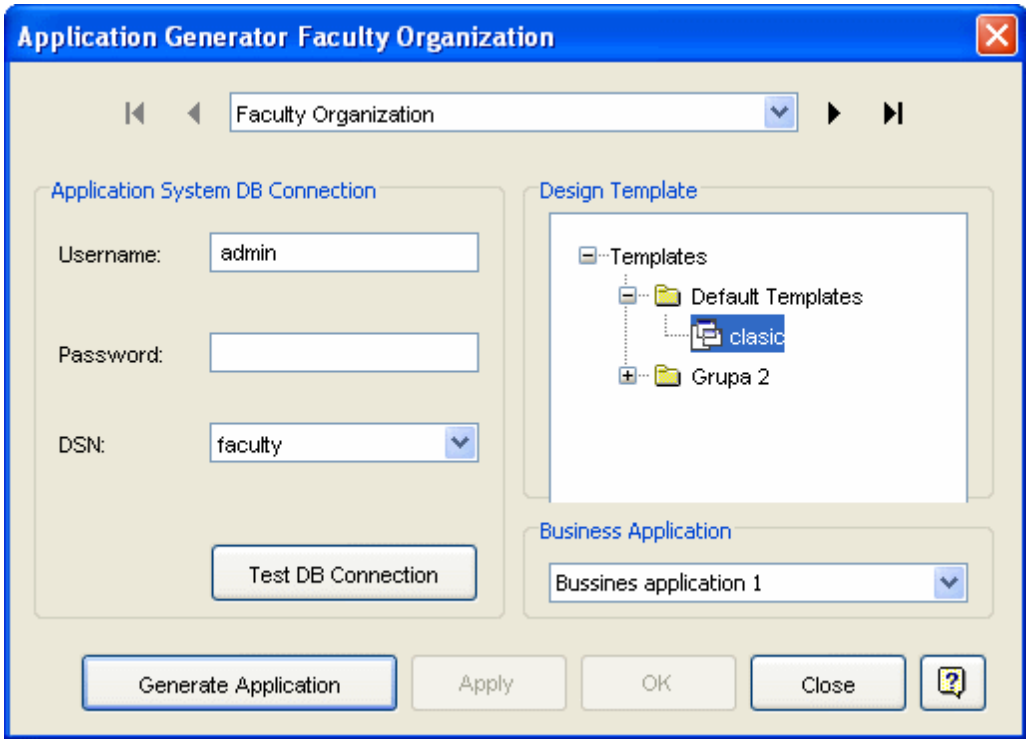
Prethodno opisani koraci *forward* inženjeringa potrebno je izvršiti u navedenom redosledu, a projektant nakon svakog koraka može da pregleda generisane relacione šeme i ograničenja. Putem odgovarajućih formi, projektant može da proveri tačnost specifikacija na nivou projekta, a na određenim mestima i da dopuni generisane specifikacije. Posle svakog koraka, ukoliko projektant nije zadovoljan dobijenim rezultatima, može se vratiti korak unazad, promeniti specifikacije vezane za tipove formi i ponoviti postupak.

Projektantu su u svakom trenutku dostupne informacije o atributima, primarnim ključevima, ograničenjima jedinstvene vrednosti kao i drugim relevantnim ograničenjima. Postoji mogućnost promene redosleda atributa, modifikacije nad ograničenjima primarnog ključa i ograničenjima jedinstvene vrednosti.

3.4.5 Generisanje prototipa aplikacije

Projektant može generisati prototip aplikacije, nakon završenog postupka generisanja i konsolidacije šeme baze podataka i njene implementacije u okviru konkretnog sistema za upravljanje bazama podataka. Na slici 3.12 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.



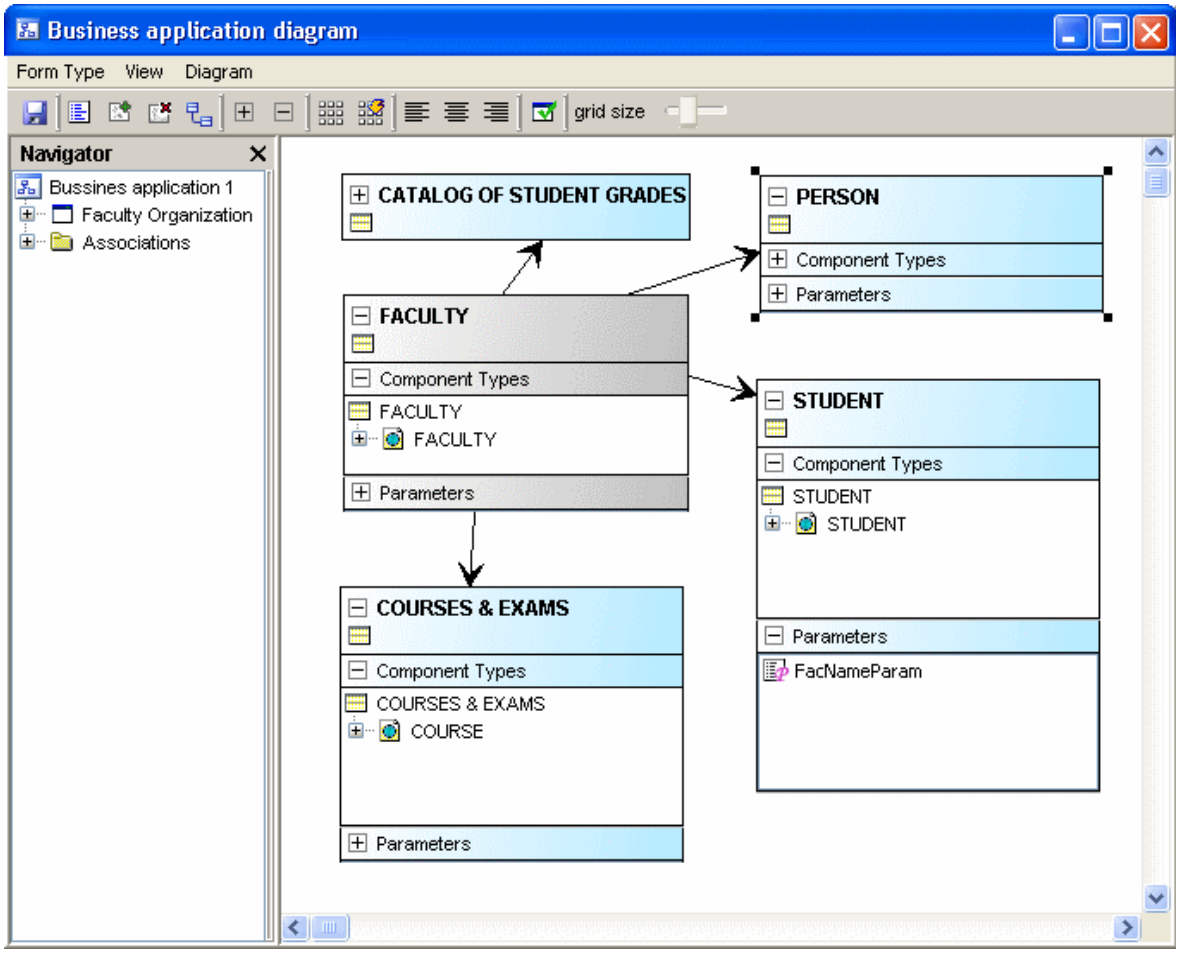
Slika 3.13. Generator aplikacija

Specifikacija poslovne aplikacije

Putem koncepta poslovnih aplikacija, projektant formalno specificira funkcionalnosti informacionog sistema, koje se odnose na uzajamne pozive generisanih ekranskih formi koje odgovaraju različitim tipovima formi. Poslovne aplikacije definišu se na nivou aplikativnog sistema. Pored skupa tipova formi koji učestvuje u poslovnoj aplikaciji, projektant specificira na koji način 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. Konceptu pozivajuće strukture mogu biti pridružene sledeće osobine:

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

Na slici 3.13 dat je primer jedne poslovne aplikacije prikazan u okviru modula za grafičko modelovanje poslovnih aplikacija, koji se naziva *Business Aplication Designer*.

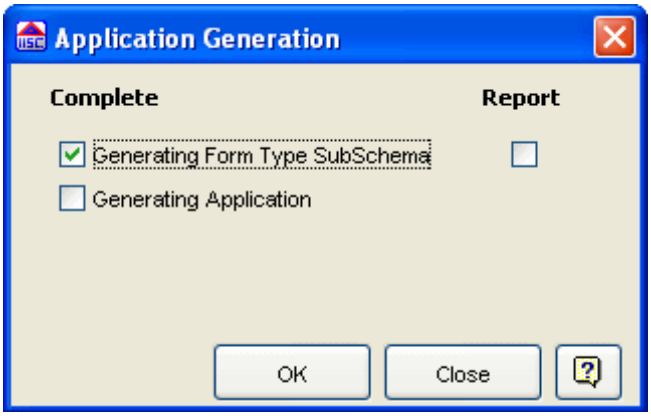


Slika 3.14. Business Aplication Designer

Generisanje transakcionog programa

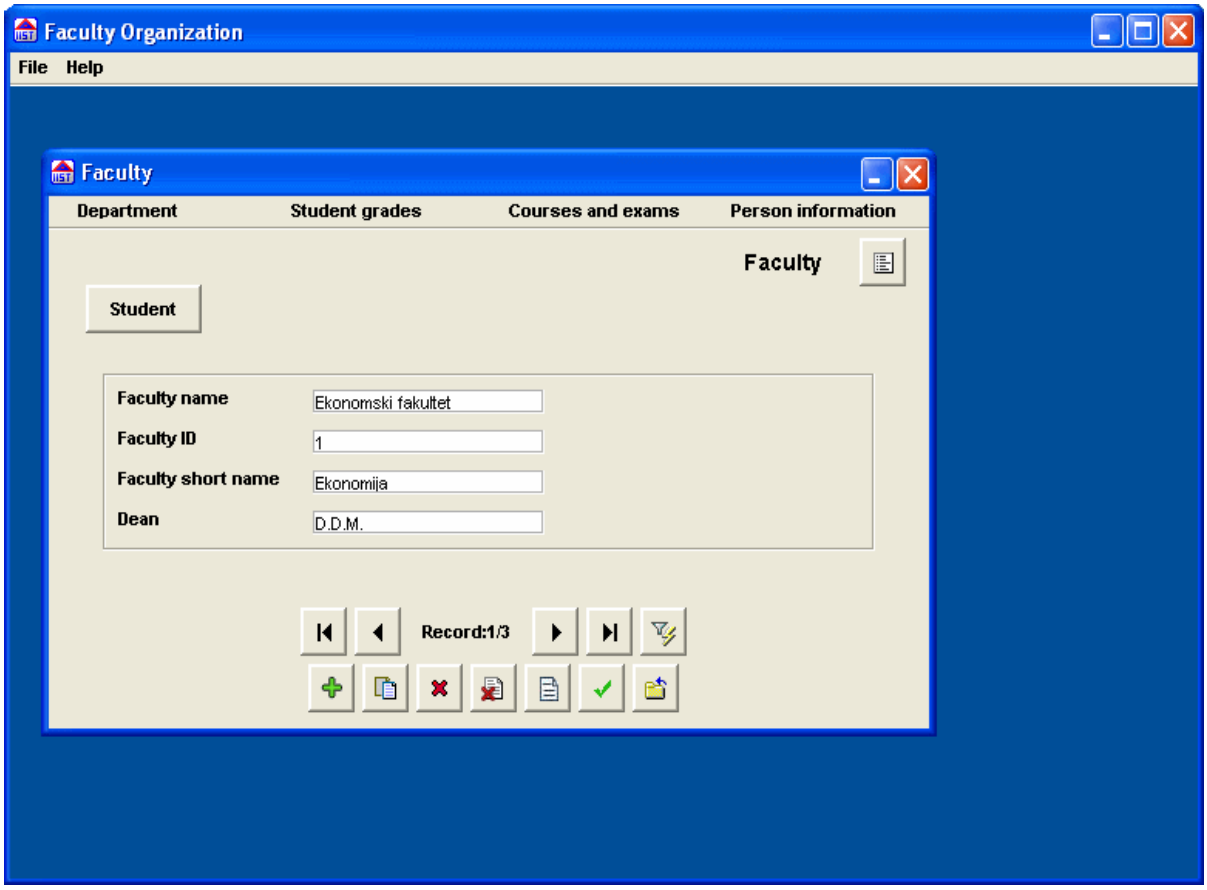
Proces generisanja transakcionog programa sadrži dva koraka (kao što je prikazano na slici 3.14):

- generisanje podšema tipova formi, i
- generisanje aplikacije.



Slika 3.15. Generisanje prototipa aplikacije

Na slici 3.15 prikazan je primer automatski generisane aplikacije.



Slika 3.16. Primer prototipa aplikacije

U ovom poglavlju predstavljen je proces *forward* inženjeringa koji je omogućen pristupom i alatom IIS*Case. Jedan od ciljeva istraživanja u ovoj doktorskoj disertaciji je da se okruženje IIS*Studio proširi novim namenskim jezikom koji će se oslanjati i na mogućnosti alata IIS*Case i njegov proces *forward* inženjeringa. U narednom poglavlju opisan je meta-mode koncepata IIS*Case alata koji predstavlja osnovu za ozgradnju ovakvog jezika.

4. Meta-modeli za specifikaciju informacionih sistema

U prethodnom poglavlju ukratko su prikazani osnovne funkcionalnosti alata IIS*Case. Pristup modelovanju specifikacije informacionog sistema zasniva se na upotrebi odgovarajućih namenskih jezika.

U ovom poglavlju biće detaljno prikazani koncepti za modelovanje specifikacije informacionog sistema na nivou meta-modela. Koncepti definisani na nivou meta-modela su elementi apstraktnih sintaksi namenskih jezika za modelovanje specifikacija informacionog sistema.. Za specifikaciju meta-modela upotrebljeno je EMF okruženje i njegov jezik za meta-modelovanje Ecore. Ecore, namenski jezik za meta-modelovanje je zasnovan na EMOF specifikaciji.
U odeljcima od 4.1 do 4.3 biće prikazani sledeći meta-modeli:

- IIS*Case meta-model,
- Meta-model koncepta šablona korisničkog interfejsa i
- Meta-model proširenog modela zasnovanog na tipu entiteta i tipa poveznika

Meta-modeli predstavljaju osnovu za izgradnju namesnkih jezika, jer se pomoću njih specificira apstraktna sintaksa za modelovanje. Na nivou apstraktne sintakse definasane su konstrukcije koje se mogu koristiti za definisanje validnih modela.

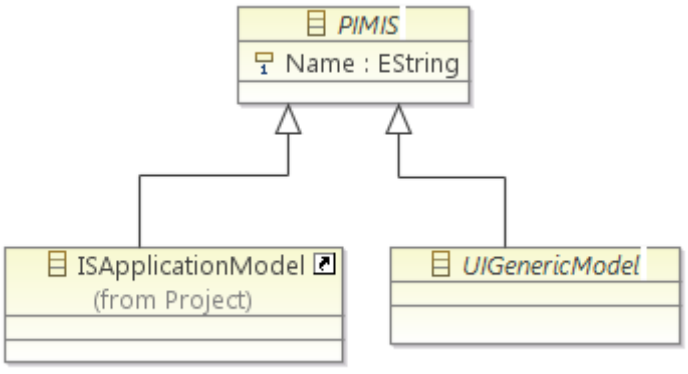
4.1 IIS*Case meta-model

Meta-model okruženja IIS*Studio sadrži veliki broj koncepata, njihovih osobina, veza i pravila. Formalne specifikacije svih koncepata koje okruženje sadrži, predstavljene su putem meta-modela. IIS*Studio PIM koncepti, koji su modelovani apstraktnom klasom ***PIMIS***, podeljeni su na:

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

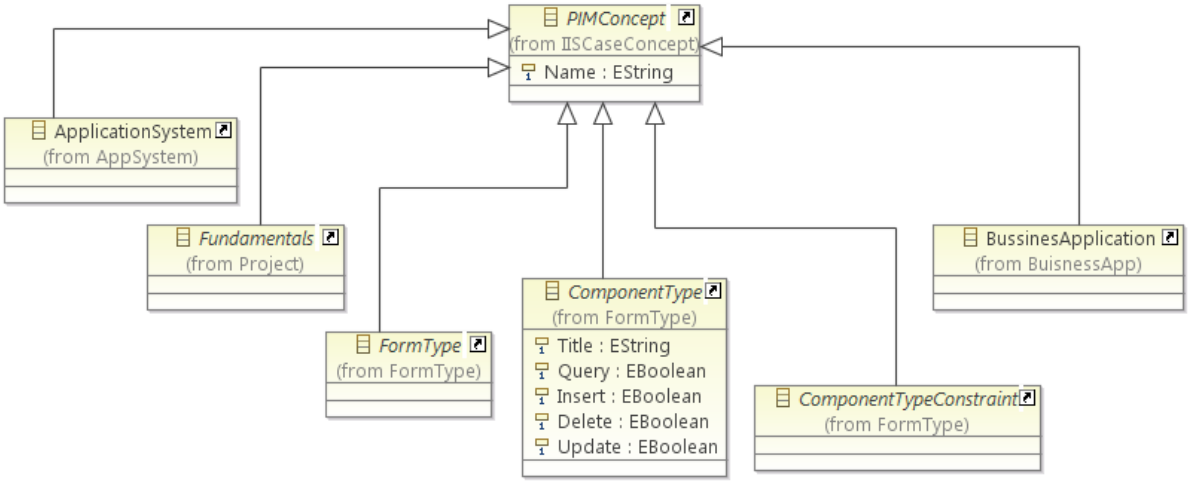
Ovakva podela je napravljena prema nameni alata kojem PIM koncepti pripadaju.

Koncepati koji su u meta-modelu predstavljeni klasom **ISApplicationModel** pripadaju IIS*Case alatu, koji služu za konceptualno modelovanje šema baza podataka. Grupa koncepata, koja je reprezentovana apstraktnom klasom ***UIGenericModel***, pripada alatu UIModeler. Alat UIModeler koji je namenjen konceptualnom modelovanju korisničkih interfejsa. Na slici 4.1 prikazan je meta-model glavnih IIS*Studio PIM koncepata.



Slika 4.1. Meta-model glavnih IIS*Studio PIM koncepata

Detaljan opis svih IIS*Case PIM koncepata može se naći u Čeliković et al. 2012 i Luković et al. 2011.



Slika 4.2. Meta-model glavnih IIS*Case PIM koncepata

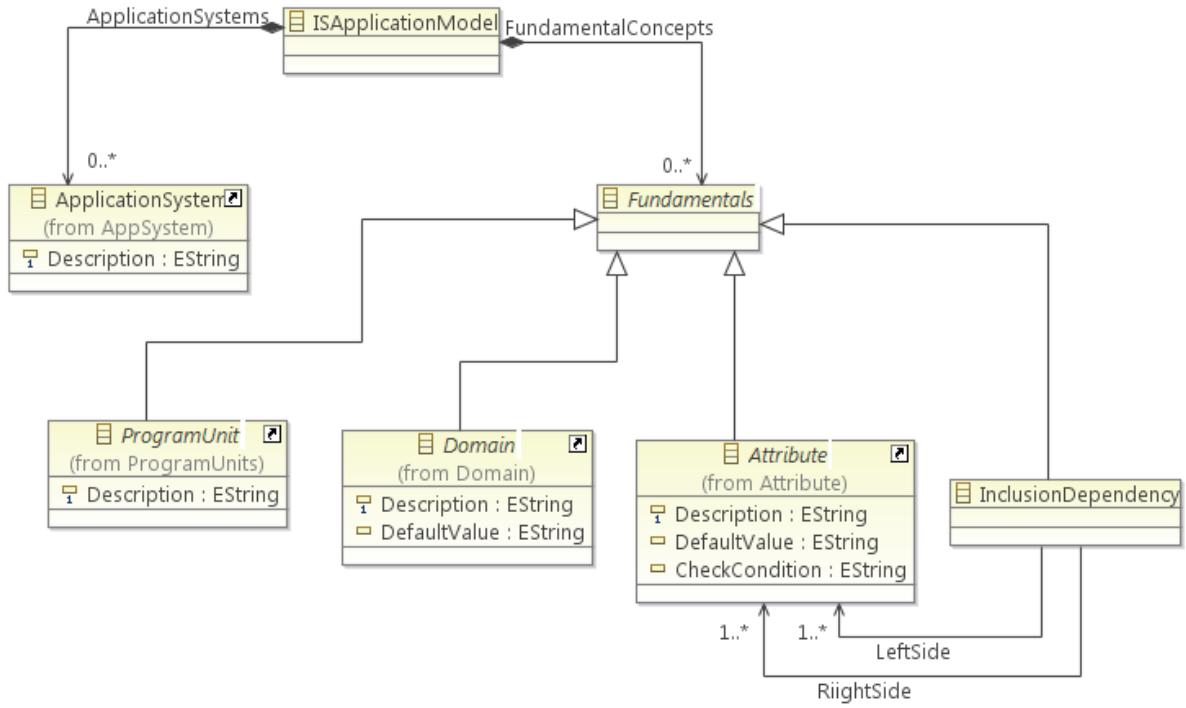
Putem apstraktne klasa ***PIMConcept*** na slici 4.2 modelovani su glavni IIS*Case PIM koncepti: aplikativni sistem (**ApplicationSystem**), tip forme (***FormType***), tip komponente (***ComponentType***), ograničenje tipa komponente (***ComponentTypeConstraint***), poslovna aplikacija (**BussinesApplication**), kao i koncepti koji su svrstani u grupu osnovnih (***Fundamentals***). Nazivi klasa koje reprezentuju odgovarajući koncept u meta-modelu navedeni su u zagradama. Apstraktnu klasu ***PIMConcept*** nasleđuju sve ostale klase i klasa sadrži atribut **Name**. U narednim sekcijama ovog poglavlja svi navedeni koncepti opisani su detaljno.

4.1.1 Meta-model koncepta modela aplikacije IS-a

Klasa **ISApplicationModel** nalazi se na vrhu hijerarhije IIS*Case meta-modela i predstavlja model aplikacije informacionog sistema. U starijim verzijama alata ovo je bio koncept projekta. Svaki model aplikacije informacionog sistema može da sadrži više aplikativnih sistema i osnovnih koncepata. Klasa **ApplicationSystem** modeluje koncept aplikativnog sistema, dok su osnovni koncepti modelovani putem apstraktnom klase ***Fundamentals***. Za svaki model aplikacije potrebno je definisati njegov naziv, što je u meta-modelu predstavljeno atributom **Name** koji je nasleđen iz klase ***PIMConcept***. Meta-model koncepta modela aplikacije informacionog sistema prikazan je na slici 4.3.

Osnovni koncepti (***Fundamentals***) formalno su nezavisni od bilo kog aplikativnog sistema. Ove koncepte projektant definiše na nivou modela aplikacije informacionog sistema i može ih koristiti u različitim aplikativnim sistemima. Osnovni koncepti su: domeni, atributi, zavisnosti sadržavanja i programske jedinice, i na meta-modelu predstavljeni su klasama ***Domain***, ***Attribute***, **InclusionDependency** i ***ProgramUnit***, respektivno.

Aplikativni sistem je organizacioni deo koji može da sadrži druge podsisteme. Razlog za definisanje ovog koncepta za modelovanje u alatu IIS*Case, je obezbeđenje mehanizma za dekompoziciju velikih aplikacija na segmente kojima se može lakše upravljati.



Slika 4.3. Meta-model koncepta modela aplikacije IS-a

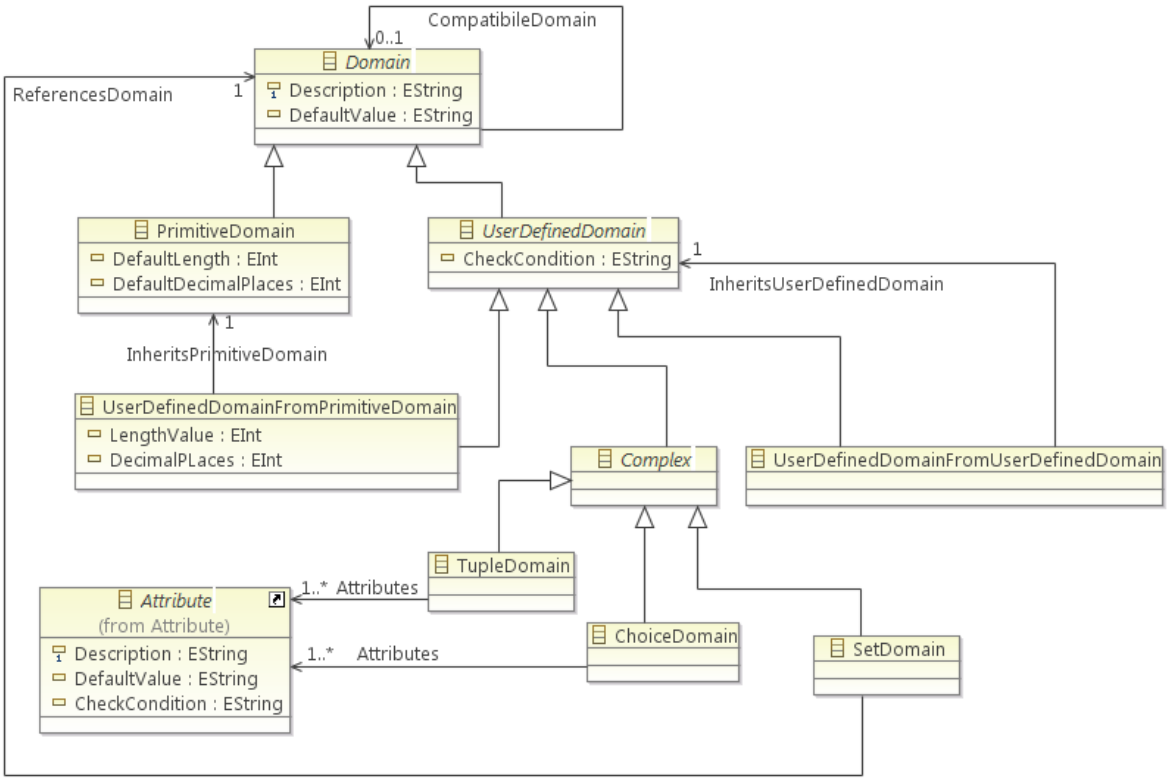
4.1.2 Meta-model koncepta domena

Koncept za modelovanje domen, koji je u meta-modelu predstavljen apstraktnom klasom ***Domain***, pripada osnovnim konceptima (***Fundamentals***). Putem ovog koncepta moguće je definisati skup dozvoljenih vrednosti nekog atributa. Pojam domena koristi se sa značenjem koje je uobičajeno za baze podataka. Na slici 4.4 prikazan je meta-model IIS*Case domen koncepta.

Za svaki domen potrebno je definisati njegov naziv i opis, dok je podrazumevana vrednost opciona. Naziv je u meta-modelu predstavljen atributom **Name**, koji je nasleđen iz klase ***PIMConcept***. Osobine opis i podrazumevana vrednost predstavljene su atributima **Description** i **DefaultValue** na nivou apstraktne klase ***Domain***. Naziv domena mora imati jedinstvenu vrednost na niovu čitavog modela aplikacije informacionog sistema. Na meta-modelu koji se nalazi na slici 4.4 domeni su podelejni na:

- primitivne domene koji su predstavljeni klasom **PrimitiveDomain** i
- korisnički definisane domene definisane apstraktnom klasom ***UserDefinedDomain***.

Primitivni tipovi podataka različitih formalnih jezika kao što su string, integer, float itd. predstavljeni su putem koncepta primitivnog diomena. Definisanjem koncepta primitivnog domena, projektantima je omogućeno da specificiraju potrebne primitivne domene u skladu sa potrebama informacionog sistema koji modeluju. Pomoću ovog koncepta povećava se izražajna moć modela koji projektant specificira. Osobine **DefaultLength** i **DefaultDecimalPlaces** koje se zadaju prilikom definisanja primitivnog domena predstavljaju način definisanja atributa dužina i broj cifara iza decimalne tačke.



Slika 4.4. Meta-model Domain koncepta

Prilikom definisanja korisnički definisanog domena potrebno je zadati uslov (atribut **CheckCondition**). Koncept uslov predstavlja regularni izraz pomoću kojeg je moguće dodatno ograničiti skup mogućih vrednosti nekog domena. Korisnički definisani domeni mogu biti specificirani kao:

- domeni kreirani pravilom nasleđivanja i
- složeni domeni.

Domen koji je kreiran na osnovu pravila nasleđivanja, nasleđuje specifikaciju prethodno definisanog primitivnog domena, koji je specificiran klasom **UserDefinedDomainFromPrimitiveDomain** ili korisnički definisanog domena, koji je definisan putem klase **UserDefinedDomainFromUserDefinedDomain**. Na nivou domena koji nasleđuje primitivni domen, projektant može specificirati dužinu (**LengthValue**) i broj cifara iza decimalne tačke (**DecimalPlaces**).

Složene domene projektant može kreirati putem tri pravila:

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

Domen koji je kreiran putem pravila torke naziva se domen tipa torke, i modelovan je klasom **TupleDomain**. Ovakav domen predstavlja torku vrednosti. Domen torke je vrsta složenog domena koji referencira postojeće atribute kao elemente domena torke. Skupovni domen ili domen tipa skupa koji je predstavljen klasom **SetDomain**, reprezentuje skupove vrednosti nad definisanim domenom. Svaka vrednost definisana u skupovnom domenu predstavlja skup vrednosti iz povezanog domena. Klasom **ChoiceDomain** modeluje domen tipa izbora, i 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.

Prilikom specifikacije atributa projektant može referencirati specificirane domene. Atributi se zatim koriste u specifikacijama tipa forme aplikativnih sistema.

4.1.3 Meta-model koncepta atributa

Na slici 4.5 prikazan je meta-model IIS*Case ***Attribute*** koncepta koji modeluje koncept atributa. Svaki atribut jedinstveno sj identifikovan isključivo svojim nazivom u okviru IIS*Case modela aplikacije. Ova osobina je u skladu sa pretpostavkom o postojanju šeme univerzalne relacije (URS). Za svaki atribut projektant može definisati podrazumevanu vrednost i zadati uslov, pored naziva i opisa, koji su obavezani prilikom specifikacije. Naziv atributa je u meta-modelu specificiran osobinom **Name**, dok su atributi opis, podrazumevana vrednost i zadati uslov modelovni atributima **Description**, **DefaultValue** i **CheckCondition**, respektivno, u apstraktnoj klasi ***Attribute***.

Koncept za modelovanje uslov definiše regularni izraz koji može dodatno ograničiti skup mogućih vrednosti atributa. Projektant definiše uslov na sličan način kao i uslov prilikom kreiranja domena. Ukoliko je definisan uslov prilikom specifikacije atributa, a specificiran je uslov i kod njemu pridruženog domena, alat koristi logičku operacija I (AND) za njihovo spajanje. Ukoliko prilikom specifikacije atributa nije definisana njegova vrednost, vrednost atributa će biti inicijalizovana na podrazumevanu vrednost definisanu na nivou atributa. U slučaju da podrazumevana vrednost atributa nije definisana, njegova vrednost biće inicijalizovana na podrazumevanu vrednost koja je definisana na nivou domena. Prilikom definicije atributa, svakom atributu, takođe, treba pridružiti domen. Ovaj koncept je na meta-modelu predstavljen vezom **AttributeDomain** ka korisnički definisanom domenu.

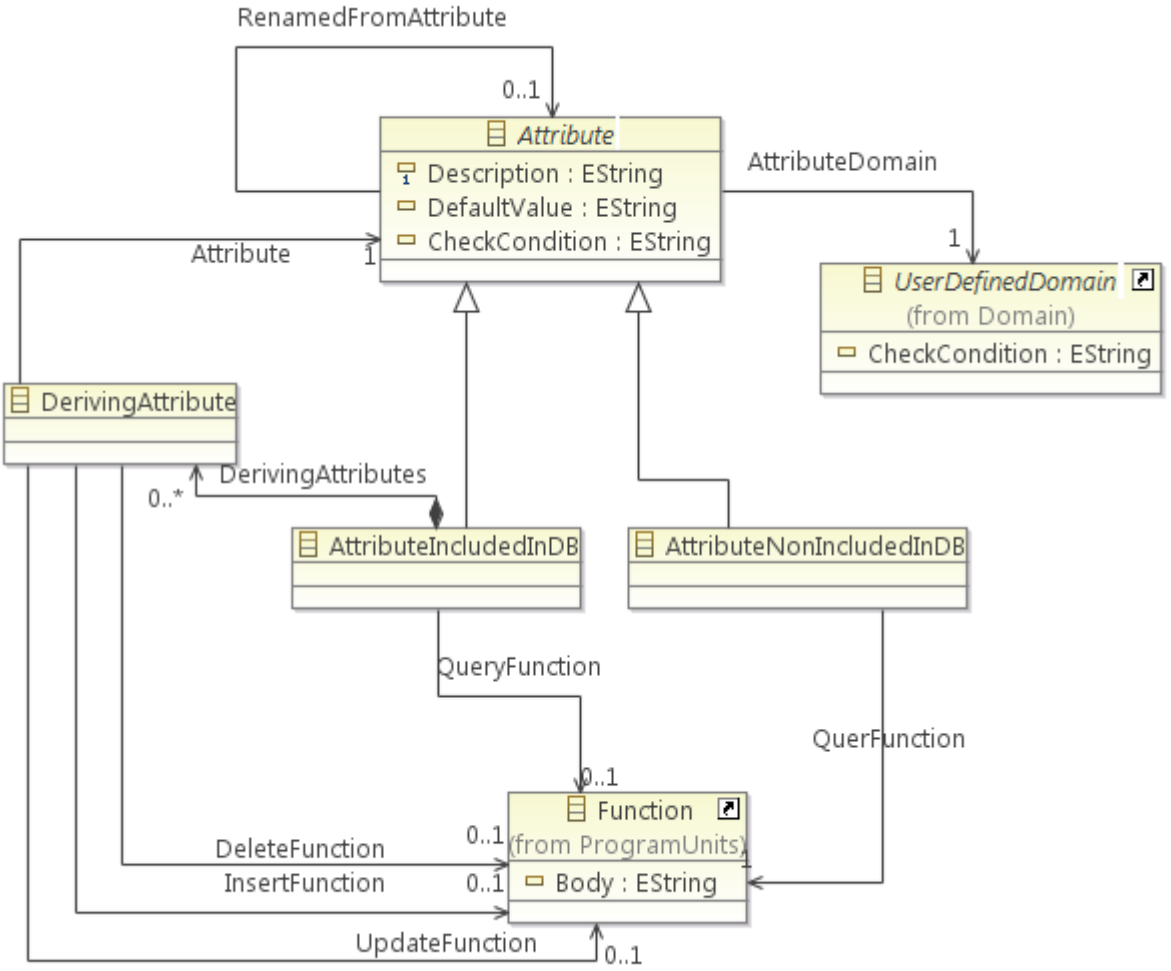
Većina atributa nekog modela aplikacije predstavlja deo šeme baze podataka koja se projektuje u okviru informacionog sistema. Međutim, često postoji potreba za atributima koji predstavljaju neke izračunate vrednosti u okviru izveštaja ili ekranskih formi. Ovakvi atributi ne moraju biti uključeni u šemu baze podataka. Na osnovu ove činjenice, IIS*Case atributi se dele na:

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

Druga podela atributa je prema kriterijumu načina zadavanja vrednosti atributa:

- elementarni ili neizvedeni, koji su predstavljeni u meta-modelu apstraktnom klasom ***Attribute*** i
- izvedeni, predstavljeni u meta-modelu klasom **DerivingAttribute**.

Projektant najčešće definiše vrednost neizvedenih atributa specificira, ili je njihova vrednost preuzeta iz nekog drugog spoljnjeg konteksta. Vrednost izvedenog atributa izračunava se korišćenjem funkcije koja može da predstavlja formulu ili algoritam. U alatu IIS*Case definisano je pravilo da svaki atribut koji nije uključen u šemu baze podataka mora biti definisan kao izveden. Prilikom specifikacije izvedenih atributa projektant može definisati da oni referenciraju neku od funkcija: funkciju unosa, funkciju izmene ili funkciju brisanja, što je u meta-modelu predstavljeno vezama: **InsertFunction**, **UpdateFunction** i **DeleteFunction**, respektivno.



Slika 4.5. Meta-model Attribute koncepta

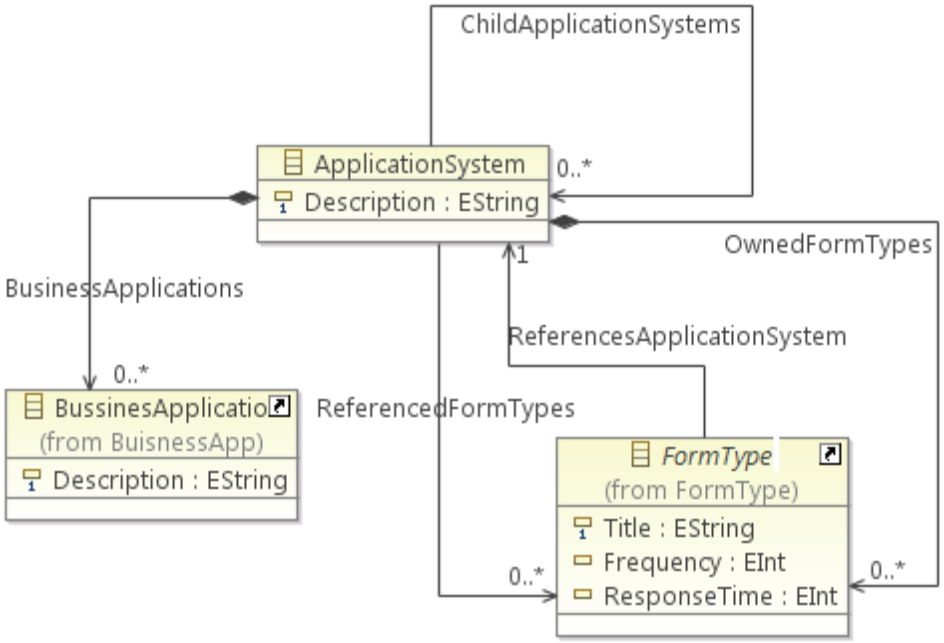
Pored elementarnih, moguće je definisati i preimenovane atribute. Ovaj koncept za modelovanje je predstavljen rekurzivnom vezom **RenamedFromAttribute**. Preimenovani atribut referencira prethodno definisani atribut i mora biti specificiran kao atribut koji je uključen u šemu baze podataka. Osnova za kreiranje takvog atributa je referencirani atribut, koji ima drugačiju semantiku. Drugačija semantika atributa dobija se putem mehanizma preimenovanja. Mehanizam preimenovanja u alatu IIS*Case je sličan mehanizmu preimenovanja koji se koristi u postupku transformacije 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 predstavljena je na slici 4.3 kao osnovni koncept ***InclusionDependency***.

4.1.4 Meta-model koncepta aplikativnog sistema

Koncept za modelovanje koji omogućuje organizaciju projekta po celinama je Aplikativni sistem. Hijerarhijska struktura aplikativnih sistema modelovana je rekurzivnom vezom **ApplicationSystemChildren**. Jedan aplikativni sistem može biti nadređeni za više aplikativnih sistema, dok jedan aplikativni sistem može imati najviše jednog direktno nadređenog.

Koncept **ApplicationSystem** definisan je na slici 4.6. Aplikativni sistem može da ima više tipova formi, a ne mora nijednu, dok jedan tip forme pripada tačno jednom aplikativnom sistemu. Koncept tipa forme u meta-modelu predstavljen je apstraktnom klasom ***FormType***. Na nivou aplikativnih sistema projektant IS može da definiše i poslovne aplikacije (***BusinessApplications***)

Prilikom specifikacije aplikativnog sistema potrebno je definisati njegov naziv i opis. Naziv aplikativnog sistema je u meta-modelu predstavljen atributom **Name** iz nasleđene klase ***PIMConcept***, a opis je modelovan atributom **Description**, koji je specificiran na nivou klase **ApplicationSystem.**



Slika 4.6. Meta-model AplicationSystem koncepta

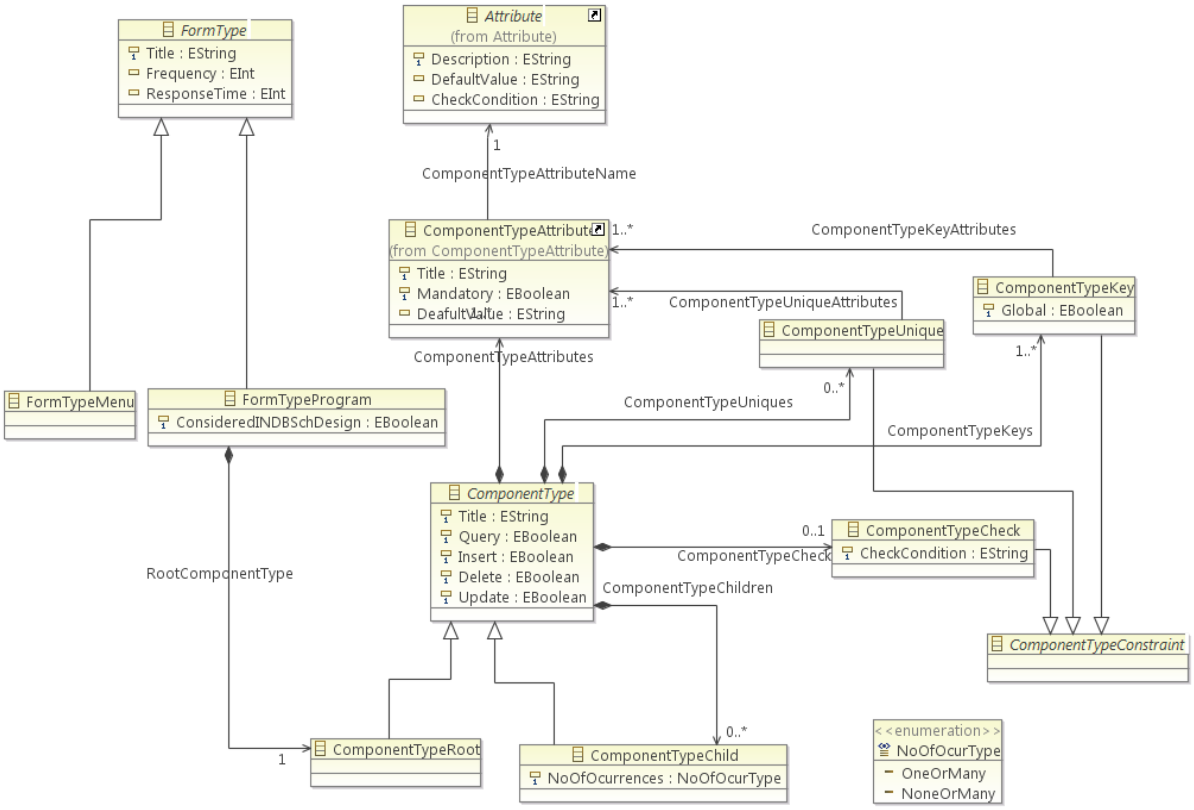
4.1.5 Meta-model koncepta tipa forme

Tip forme predstavlja glavni koncept za modelovanje u IIS*Case alatu. Putem tipa forme proejktant modeluje apstrakciju tipova dokumenata, ekranskih formi ili izveštaja koje krajnji korisnici IS mogu koristiti u obavljanju svakodnevnih poslova. Pomoću koncepta tipa forme, projektant na nivou PIM-a, na idirektan način definiše šemu baze podataka sa atributima i ograničenjima, kao i sa modelima transakcionih programa i aplikacija informacionog sistema koji je predmet modelovanja. Projektant ima mogućnost da u projektovani aplikativni sistem, uključi i tipove formi koji su kreirani za potrebe drugog aplikativnog sistema. Na osnovu ove činjenice, tipovi formi se klasifikuju na:

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

Aplikativni sistem poseduje tip forme ukoliko je tip forme definisan na nivou tog istog sistema. Tip forme definisan na ovaj način može se naknadno izeminit, ali samo u okviru aplikativnog sistema koji ga poseduje bez bilo kakvih ograničenja. Na slici 4.6 ovakav tip forme definisan je je kompozitnom vezom **OwnedFormTypes**. Referencirani tip forme potrebno je najpre kreirati u okviru jednog aplikativnog Sistema. U narednom koraku ovako definisan tip forme je uključen u drugi aplikativni sistem koji je predmet projektovanja. Referencirane tipove formi u aplikativnom sistemu u koji su uključeni, projektat može samo

koristiti bez bilo kakve mogućnosti izmene strukture. Na slici 4.6 referencirani tip forme modelovan je vezom **ReferencedFormTypes**. Meta-model koncepta tipa forme definisan je na slici 4.7.



Slika 4.7. Meta-model koncepta tipa forme

Za svaki tip forme (***FormType***) mogu se definisati sledeće osobine: naziv, koji pomoću koje se jedinstveno identifikuje u okviru modela aplikacije, naslov, učestalost upotrebe i vreme odgovora. Atributi naziv i naslov su obavezni prilikom specifikacije i predstavljeni su u meta-modelu atributima **Name** i **Title**, respektivno. Učestalost upotrebe i vreme odgovora su opcione osobine, i predstavljeni su u meta-modelu atributima **Frequency** i **ResponseTime**. Putem osobine vreme odgovora je projektant može da specificira očekivano vreme odgovora nakon izvršenja nekog programa.

Tipovi formi dele se na menije i programske tipove formi. Meni tipovi formi se koriste za specifikaciju menija aplikacije bez elemenata za manipulaciju podacima. Koncept meni tip forme specificiran je u meta-modelu putem klase **FormTypeMenu.** Specifikacija transakcionih programa sa korisničkim interfejsom obavlja se putem koncepta za modelovanje programski tip formi. Ovakav tip formi modeluje ekranske forme za preuzimanje i ažuriranje podataka, ili izveštaje za prikaz rezultata preuzetih podataka. Koncept za modelovanje programskog tipa forme specificiran je u meta-modelu klasom **FormTypeProgram**.

Programski tip forme može da ima kompleksnu strukturu. Ovakav tip forme može biti uzet ili izostavljen iz razmatranja prilikom projektovanja šeme baze podataka. Ova osobina može biti specificirana atributom **ConsideredInDBSchDesign** iz klase **FormTypeProgram**. Tipovi formi, koji su specifirani da se uzmu u razmatranje prilikom projektovanja šeme baze podataka (atribut **ConsideredInDBSchDesign** ima specificiranu vrednost *true*), kasnije predstavljaju ulazne podatke za proces generisanja šeme baze podataka. Tipovi formi koji su

definisanu da se ne razmatraju prilikom projektovanja šeme baze podataka (atribut **ConsideredInDBSchDesign** ima vrednost *false*), ne koriste se u procesu generisanja šeme baze podataka. Projektant koristi ovakve tipove formi samo za modelovanje izveštaja ili tipova formi za preuzimanje podataka.

Svaki tip forme definisan je kao struktura stabla tipova komponenti, koji su na nivou meta-modelu modelovani putem apstraktne klase ***ComponentType***. Tip forme mora da sadrži makar jedan tip komponente koji je označen kao korenski, a može imati i više podređenih tipova komponenti. Korenski tip komponente modelovan je klasom **ComponentTypeRoot**, dok su ostali tipovi komponenti definisani putem klase **ComponentTypeChild**.

Za svaki tip komponente projektant može da definiše vrednosti za ove osobine naziv (**Name**), naslov (**Title**) i dozvoljene operacije nad pojavama tipa komponente upit (**Query**), unos (**Insert**), izmena (**Update**) i brisanje (**Delete**). U zagradama su definisani nazivi atributa klase **ComponentType** koji modeluju odgovarajuće osobine koncepta tipa komponente. Sve navedene osobine prilikom specifikacije tipa komponente su obavezne.

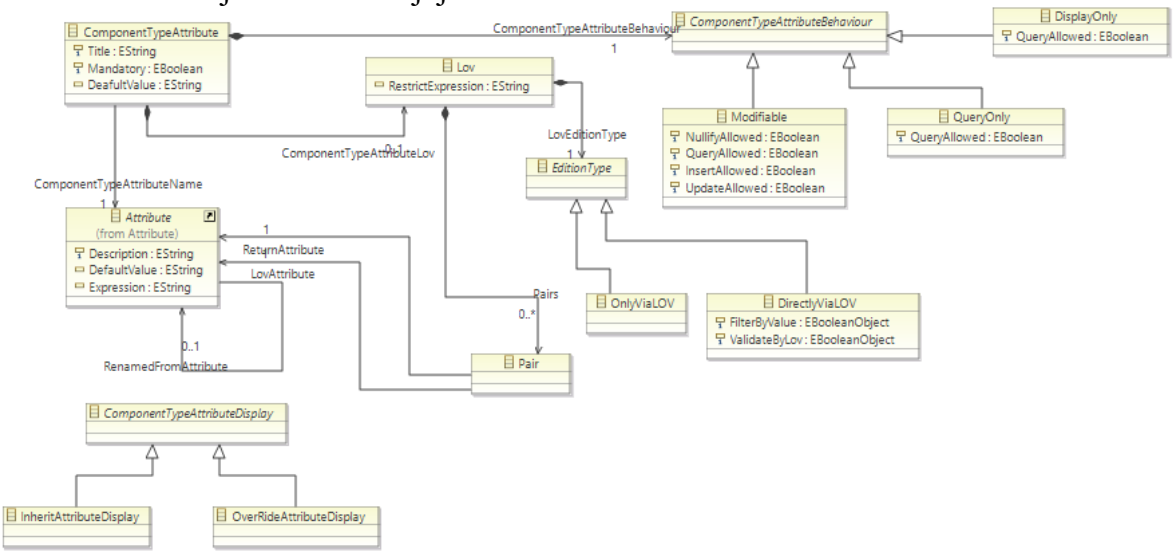
Naziv tipa komponente predstavlja jedinstveni identifikator za svaku instancu tipa komponente. Svaka instanca nadređenog tipa komponente u hijerarhiji može da sarži više podređenih tipova komponenti. Putem osobine broj pojavljivanja, koja je predstavljena u meta-modelu atributom **NoOfOccurrences,** projektant može da ograniči minimalan broj instanci podređenog tipa komponente koje mogu biti povezane sa instancom istog nadređenog tipa u okviru hijerarhije tipa stabla. Osobina broj pojavljivanja može biti definisana sa vrednostima: 0-N ili 1-N. Ove dve vrednosti u IIS*Case meta-modelu modelovane su kao enumeracija **NoOfOcurType**. Vrednost 0-N označava da za svaku instancu nadređenog tipa komponente može da postoji nula ili više instanci podređenog tipa. Vrednost 1- N označava da se za svaku instancu nadređenog tipa komponente mora da bude definisana bar jedna instanca podređenog tipa komponente.

Svaki tip komponente može da sadrži jedan ili više atributa. Atributi tipa komponente modelovani su putem klase **ComponentTypeAttribute** na slici 4.8. Ovaj koncept povezuje atribut (**Attribute**), iz skupa atributa koji su specificirani na nivou modela aplikacije, sa tipom komponente na kojoj će biti prikazan. Ova osobina je modelovana vezom u meta-modelu **ComponentTypeAttributeName**. Za svaki atribut tipa komponente je definisati njegov naslov (**Title**) kojim će atribut biti vidljiv na generisanoj ekranskoj formi kojoj taj tip komponente pripada. Projektant takođe treba da specificira da li je vrednost atributa obavezna ili opciona (**Mandatory**). Podrazumevana vrednost (**DefaultValue**) atributa na tipu komponente je opciona, tj ne mora biti definisana. Na nivou čitavog tipa forme, jedan atribut može biti uključen najviše jednom.

Atribut tipa komponente ima referencu prema atributu koji je specificiran u okviru projekta u na niovu koncepata **Fundamentals**. Atribut tipa komponente ima osobinu naslov koji će biti vidljiv u generisanom ekranu. Atribut, može biti specificiran kao obavezan ili opcioni na ekranskoj formi. Putem koncepta dozvoljene operacije nad atributom tipa komponente, projektant specificira dozvoljene operacije nad bazom podataka, koje se mogu izvršiti na atributu pomoću odgovarajuće stavke ekrana. Projektant može izabrati dozvoljene operacije iz skupa {*query*, *insert*, *update*, *nullify*}. Za atribut tipa komponente moguće je definisati osobine tokom njegovog prikazivanja čime je definisana osobina prezentacije atributa na ekranskoj formi. Ponašanje atributa za vreme njegovog prikazivanja se specificiraju na isti

način kao i kod definicije atributa. Vrednosti osobina prikaza mogu biti nasleđene iz definicije atributa ili modelovane na nivou atributa tipa komponente.

Grupisanje atributa tipa komponenti u grupe stavki, je omogućeno kako bi bilo obezbeđeno objedinjeno pravilo formatiranja rasporeda odabranih atributa tipova komponenti. Svaka grupa stavki može da ima jedan ili više atributa tipa komponente ili grupa stavki iz istog tipa komponente. Za svaku grupu stavki projektant IS specificira vrednosti za: naziv, naslov i kontekst. Specifikacija vrednosti za naziv i naslov je obavezna. Za svaki atribut tipa komponente može se specificirati funkcionalnost "Lista vrednosti" (LOV). Ovim putem projektant treba da definiše tip forme koji ima ulogu tipa LOV. Projektant takođe treba da specificira na koji način krajnji korisnik može da ažurira vrednosti atributa: "Samo preko LOV-a" (**Only via LOV**) ili "Direktno i putem LOV-a" (**Directly & via LOV**). Vrednost "Samo preko LOV-a" specificira da vrednost atributa ne može biti ažurirana putem tastature, već samo na osnovu LOV-a. Vrednost "Direktno i putem LOV-a" označava da ažuriranje vrednosti atributa obezbeđene i preko tastature i LOV-a. Putem svojstva **FilterByValue** projektant specificira da li će biti prikazane sve vrijednosti iz LOV ili samo one filtrirane na osnovu šablona koji definisao krajnji korisnik.



Slika 4.8. Meta-model koncepta atributa tipa komponente

Za svaki tip komponente može biti definisan skup ograničenja, koji je predstavljen apstraktnom klasom **ComponentTypeConstraint**. Ograničenja mogu biti definisana kao:

- ograničenje ključa,
- ograničenje jedinstvenosti i
- ograničenje torke.

U okviru svakog tipa komponente mora biti definisan najmanje jedan ključ. Ključ koji je definisan na nivu tipa komponente mora da sadrži najmanje jedan atribut tipa komponente. Ključ tipa komponente je modelovan u IIS*Case meta-modelu klasom **ComponentTypeKey**. Putem ključa tipa komponente jedinstveno je identifikovana svaka instanca tipa komponente, ali samo u okviru instance nadređenog tipa komponente. Za svaki tip komponente projektant može definisati i ograničenja jedintvenosti. Ova vrsta ograničenja definisana je u meta-modelu klasom **ComponentTypeUniqueness**. Ograničenje jedinstvenosti takođe mora da sadrži najmanje jedan atribut tipa komponente. Za razliku od koncepta ključa, atributi sadržani u ograničenju jedinstvenosti mogi biti opcioni tj. mogu imati nula vrednost. Putem ograničenja jedinstvenosti tipa komponente obezbeđen je mehanizam da je svaka instanca

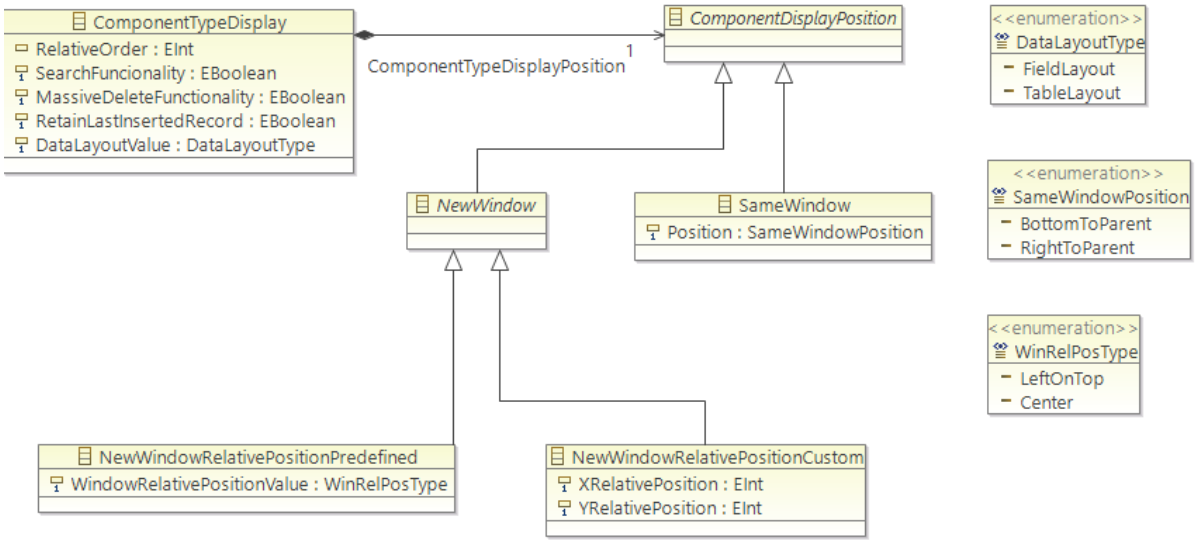
tipa komponente jedinstveno identifikovana na nivou instance nadređenog tipa komponente, u slučaju da atributi koje sadrži nemaju nula vrednost.

Jedna instanca ograničenja torke može biti definisana za tip komponente. Ova vrsta ograničenja specificira ograničenja na moguće vrednosti unutar jedne instance tipa komponente. Klasa **ComponentTypeCheck** predstavlja ovu vrstu ograničenja u meta-modelu. Logički izraz putem kojeg se proverava vrednost svake instance tipa komponente modelovan je atributom **CheckCondition** u klasu **ComponentTypeCheck**.

Projektant može takođe definisati osobine prikaza tipa komponente koje koristi generator programa. Koncept prikaza tipa komponente na slici 4.9 definisan je osobinama: raspored prozora, raspored podataka, relativni redosled, relativna pozicija rasporeda, relativna pozicija prozora, funkcijom pretraživanja, masovnim funkcijama brisanja i čuvanjem poslednjeg umetnutog zapisa.

Raspored prozora ima dve moguće vrednosti: "Novi prozor" i "Isti prozor" i određuje da li će tip komponente biti prikazan u novom prozoru ili u istom prozoru kao i nadređeni tip komponente. Raspored podataka određuje način predstavljanja tipa komponente na ekranskoj formi. Moguće su dve vrednosti: "Polje" ili "Tabela". Pomoću "Polja", moguće je prikazati samo jedan zapis u isto vreme. Pomoću "Tabele", moguće je prikazati skup zapisa u isto vreme na ekranskoj formi, u obliku tabele. Relativni redosled je redni broj koji predstavlja redosled tipa neke komponente u odnosu na druge vrste srodnih komponenti istog roditelja u stablu tipa forme. Relativna pozicija rasporeda predstavlja relativnu poziciju tipa komponente prema tipu roditeljske komponente. Projektant može da izabere vrednost "**Bottom to parent**" ukoliko želi da postavi tip komponente ispod roditeljskog tipa komponente u generisanoj ekranskoj formi. Takođe može da izabere vrednost " Right to parent ", ukoliko je tip komponente postavljen desno u odnosu na roditeljsku. Relativnu poziciju prozora moguće je specificirati samo kada je izabran raspored "Novi prozor". Projektant može odrediti jednu od tri moguće vrednosti: "Centar", "Levo na vrhu" ili "Prilagođeno". Vrednost "Centar" označava da je centar novog prozora pozicioniran tako da odgovara centru roditeljskog prozora. "Levo odozgo" označava da gornji levi ugao novog prozora odgovara gornjem levom uglu roditeljskog prozora. Izborom "prilagođene" vrednosti, relativna pozicija novog prozora u odnosu na nadređeni prozor je eksplicitno određena dajući X i Y relativne pozicije levog gornjeg ugla.

"Funkcija pretrage" predstavlja boolean svojstvo koje omogućava generisanje filtera za odabir podataka. Ako je omogućena funkcionalnost pretrage, krajnjim korisnicima je dozvoljeno da definišu WHERE klauzulu SQL SELECT naredbe. Izborom opcije, "masivna funkcionalnost brisanja" projektant obezbeđuje generisanje opcije brisanja pored svakog zapisa u tabeli. Opcija " retain last inserted record " označava da li će poslednji unesen zapis biti zadržan na ekranu.



Slika 4.9. Meta-model koncepta prikaza tipa komponente

4.1.6 Meta-model koncepta poslovne aplikacije

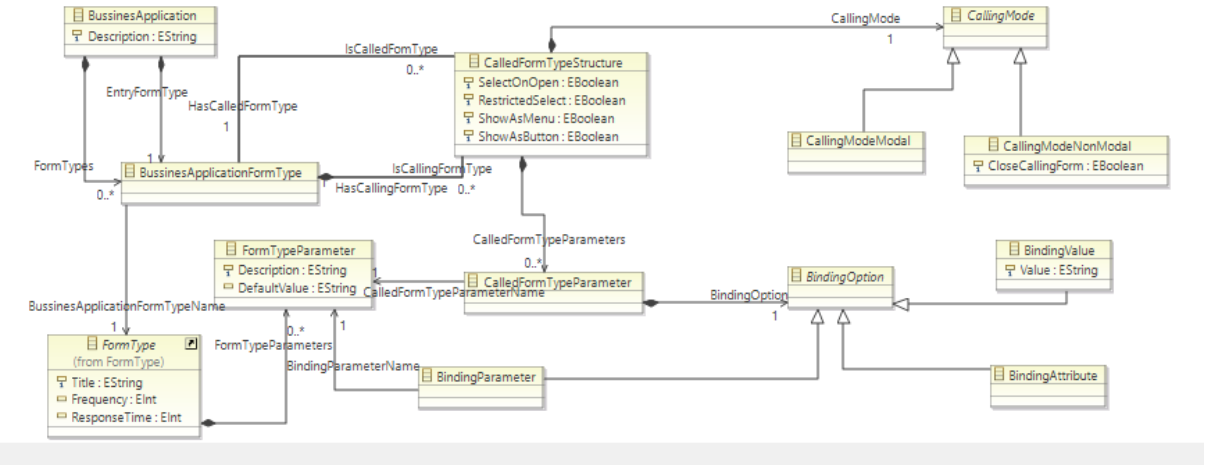
Koncept poslovne aplikacije (**BussinesApplication**) na slici 4.10 predstavlja način formalnog opisa funkcionalnosti IS i organizovan je kroz strukturu tipova formi. Svaka poslovna aplikacija ima naziv (**name**) i opis (**description**). Jedan tip forme koji je uključen u strukturu poslovne aplikacije mora biti deklarisan kao tip forme za ulazak u aplikaciju (**EntryFormType**). Ovaj tip forme predstavlja prvi transakcioni program koji se poziva prilikom pokretanja poslovne aplikacije. Svaka poslovna aplikacija mora imati tip forme za ulazak. Za podršku kreiranju struktura tipa forme unutar jedne poslovne aplikacije, potrebno je koristi koncept poziva tipa forme (**CalledFormTypeStructure**). Korišćenjem ovog koncepta, projektanti modeluju izvršavanje poziva između generisanih transakcionih programa programa. Pomoću ovog koncepta moguće je modelovanje parametara i prosleđivanje vrednosti između dva transakciona programa tokom izvršavanja poziva. Koncept poziva tipa forme sadrži dva tipa forme sa ulogama: tip forme pozivnog oblika (**isCallingFormType**) i tip forme pozvanog oblika (**isCalledFormType**).

Bilo koji tip forme može definisati formalne parametre (**CalledFormTypeParameter**). Svaki formalni parametar ima obavezni naziv kao identifikator. Parametar mora biti povezan sa tačno jednom domenom. U specifikaciji poziva tipa forme, moguće je povezati svaki parametar sa atributom tipa forme. Ovim postupkom, projektant modeluje za koje atribute će se preneti stvarna vrednost parametra tokom izvršavanja poziva.

Za pozvani tip forme u pozivu moramo da odredimo **Binding** opcije. **Binding** opcija sadrži formalne parametre pozvanog tipa forme. Za svaki parametar dizajner specificira kako će se vrednost parametra preneti stvarnom argumentu. Postoje tri moguće opcije: "vrednost" (**BindingValue**), "referenca atributa" (**BindingAttribute**) ili "referenca parametra" (**BindingParameter**). ”Vrednost” je konstanta koja će se preneti tokom izvršavanja poziva. "Referenca atributa" pruža vezu sa atributom povanog tipa forme koja daje vrednost koja se prenosi na parametar tokom izvršavanja poziva. " Referenca parametra" povezuje parametar pozivnog tipa forme koji pruža vrednost koja se prenosi na parametar tokom izvršavanja poziva.

Projektant takođe treba da specificira: metod pozivanja, način pozivanja i položaj UI tipa forme. Metod pozivanja se sastoji od dva logička obeležja: a) "Odabirom otvoreno" (**Select**

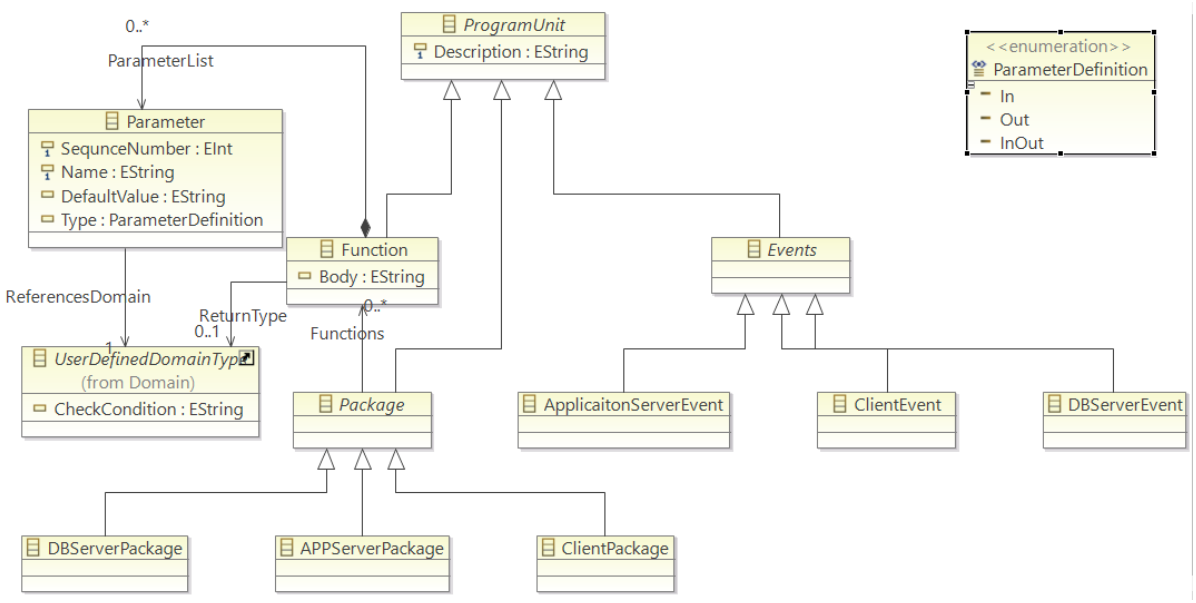
on open) i b) "Ograničen izbor" (**Restricted select**). **Select on open** znači da se pozvani tip forme otvara sa automatskom selekcijom podataka. **Restricted select** dozvoljava odabir podataka u pozivanom tipu forme ograničeno, tj samo po vrednosti prenetih parametara. Režim pozivanja određuje opšte ponašanje poziva tipa forme tokom izvršavanja poziva. Dozvoljene su tri mogućnosti: "**Modal**", "**Non-modal**" ili "**Close call form**". "**Modal**" znači da korisnik ne može aktivirati pozivani tip forme dok se otvara pozvani tip forme. "**Non-modal**" znači da su i pozivni i pozvani tip forme istovremeno aktivni na ekranu. "**Close call form**" se koristi da dovede do zatvaranja pozivnog tipa forme tokom izvršavanja poziva. Položaj UI određuje način na koji će se poziv obezbediti na nivou UI: kao stavka menija (**ShowAsMenu**) ili kao stavka dugmeta (**ShowAsButton**).



Slika 4.10. Meta-model koncepta poslovne aplikacije

4.1.7 Meta-model koncepta programske jedinice

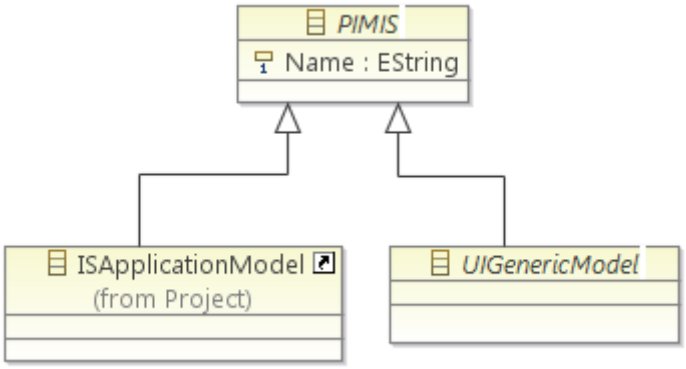
Koncept programske jedinice prikazan je na slici 4.11. Koncept funkcije koristi se za specifikaciju kompleksnih funkcionalnosti. Za svaku funkciju specificira se telo funkcije što je na meta-modelu predstavljeno atributom Body u okviru klase Function. Za svaku funkciju je moguće specificirati tip povratne vrednosti, što je na meta-modelu specificirano vezom ReturnType ka korisnički definisanom domenu. Funkcija može da sadrži listu parametara predstavljenih na meta-modelu klasom Parameter. Svaki parametar opisan je rednim brojem (SequenceNumber), nazivom (Name), predefinisanom vrednošću (DefaultValue) i tipom parametra (Type) čime se definiše da li je parametar ulazni (In), izlazni (Out) ili ulazno-izlazni (InOut). Funkcije u IIS*Case-u definisane su na nivou projekta i mogu biti referencirane iz različitih IIS*Studio specifikacija. Paket je kolekcija odabranih funkcija definisanih u IIS*Studio repozitorijumu. Paketi su na nivou meta-modela specificarani pomoću klase Package.



Slika 4.11. Meta-model koncepta programske jedinice

4.2 Meta-model koncepta šablona korisničkog interfejsa

Grupa koncepata, predstavljena apstraktnom klasom ***UIGenericModel***, pripada alatu UIModeler, koji je namenjen konceptualnom modelovanju šablona korisničkih interfejsa. Na slici 4.12. prikazan je meta-model glavnih IIS*Studio PIM koncepata.



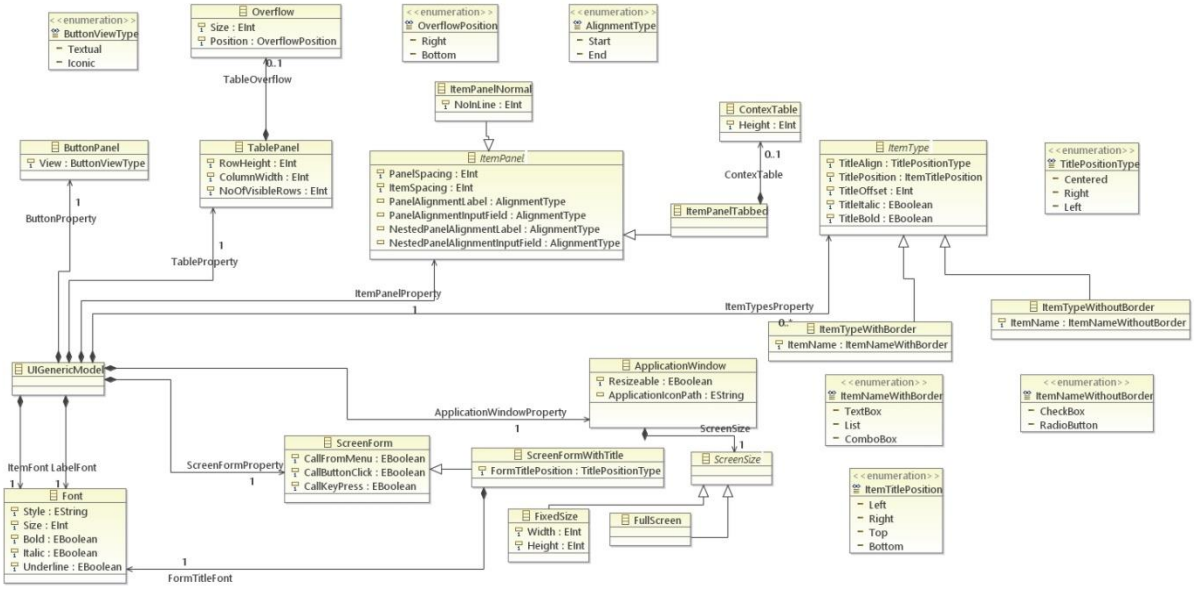
Slika 4.12. Meta-model glavnih IIS*Studio PIM koncepata

U narednom tekstu biće opisani koncepti UIModeler –a. Meta-model koncepata modela šablona korisničkog interfejsa prikazan je na slici 4.13. Klasa **UIGenericModel** je korenski element, koji se nalazi na vrhu hijerarhije UIModeler meta-modela. Ova klasa predstavlja model šablona korisničkog interfejsa. Svaki model šablona korisničkog interefeja sadrži specifikaciju glavne ekranske forme, specifikaciju ostalih ekranskih formi, specifikaciju tabela, specifikaciju panela, specifikaciju polja za prikaz i ažuriranje podataka i specifikaciju dugmadi. Za svaki šablon potrebno je zadati specifikaciju tipa, veličine i formata fonta polja za prikaz i ažuriranje podataka (*ItemFont*), kao i fonta za naziv polja za prikaz i ažuriranje podataka (*LabelFont*). Font je predstavljen klasom **Font** i opisan je stilom (*Style*), veličinom (*Size*), kao i načinom prikaza (*Bold*, *Italic*, *Underline*).

Koncept za opis glavne ekranske forme predstavljen je klasom **ApplicationWindow**. Pri specifikaciji glavne ekranske forme potrebno je zadati putanju do ikone aplikacije (*ApplicationIconPath*) i da li je moguće menjati veličinu prozora (*Resizeable*). Takođe, mora se definisati da li je veličina prozora maksimalna što je u modelu predstavljeno klasom **FullScreen** ili fiksne veličine što je u modelu predstavljeno klasom **FixedSize**. U slučaju izabrane fiksne veličine, potrebno je zadati visinu (*Height*) i širinu (*Width*) glavnog prozora aplikacije.

Specifikacija ostalih ekranskih formi predstavljena je klasom **ScreenForm**. Za svaku ekransku formu potrebno je specificirati da li je moguć poziv ekranskih formi iz menija (*CallFromMenu*), klikom na dugme (*CallButtonClick*) i pritiskom na odgovarajuće dugme sa tastature (*CallKeyPress*). Forme za koje je potrebno prikazati naslov, modeluju se pomoću klase **ScreenFormWithTitle** koja nasleđuje klasu **ScreenForm**. Za naslov ekranske forme potrebno je specificirati tip i veličinu fonta (*FormTitleFont*) kao i poziciju naslova (*FormTitlePosition*). Pozicija naslova može biti sa leve, desne strane ili na sredini. Ove vrednosti definisane su u enumeraciji **TitlePositionType** kao *Left*, *Right* i *Centered*.

Koncept za opis dugmadi predstavljen je klasom **ButtonPanel**. Podržana su dva tipa prikaza dugmadi tekstualni i grafički. Način prikaza zadaje se pomoću atributa *View* koji ima predefinisane vrednosti *Textual* i Iconic iz enumeracije **ButtonViewType.**
Konept za opis tabela predstavljen je klasom **TablePanel**. Za svaki opis tabele potrebno je specificirati visinu redova (*RowHeight*), širinu kolona (*ColumnWidth*) i broj vidljivih redova u tabelama kojim je određena visina tabele (*NoOfVisibleRows*). Specijalno, u slučaju postojanja *overflow* grupe polja, mora se zadati da li će data grupa polja biti prikazana u posebnoj *overflow* oblasti, kao i veličina date oblasti. Koncept *overflow* grupe polja predstavljen je klasom **Overflow**. Pri specifikaciji ovog koncepta potrebno je zadati visinu *overflow* grupe polja (*Size*) i poziciju na kojoj se prikazuje (*Position*). Pozicija *overflow* grupe može biti sa desne ili donje strane u odnosu na tabelu i zadaje se kao predefinisana vrednost *Right* ili *Bottom* iz enumeracije **OverflowPosition**.

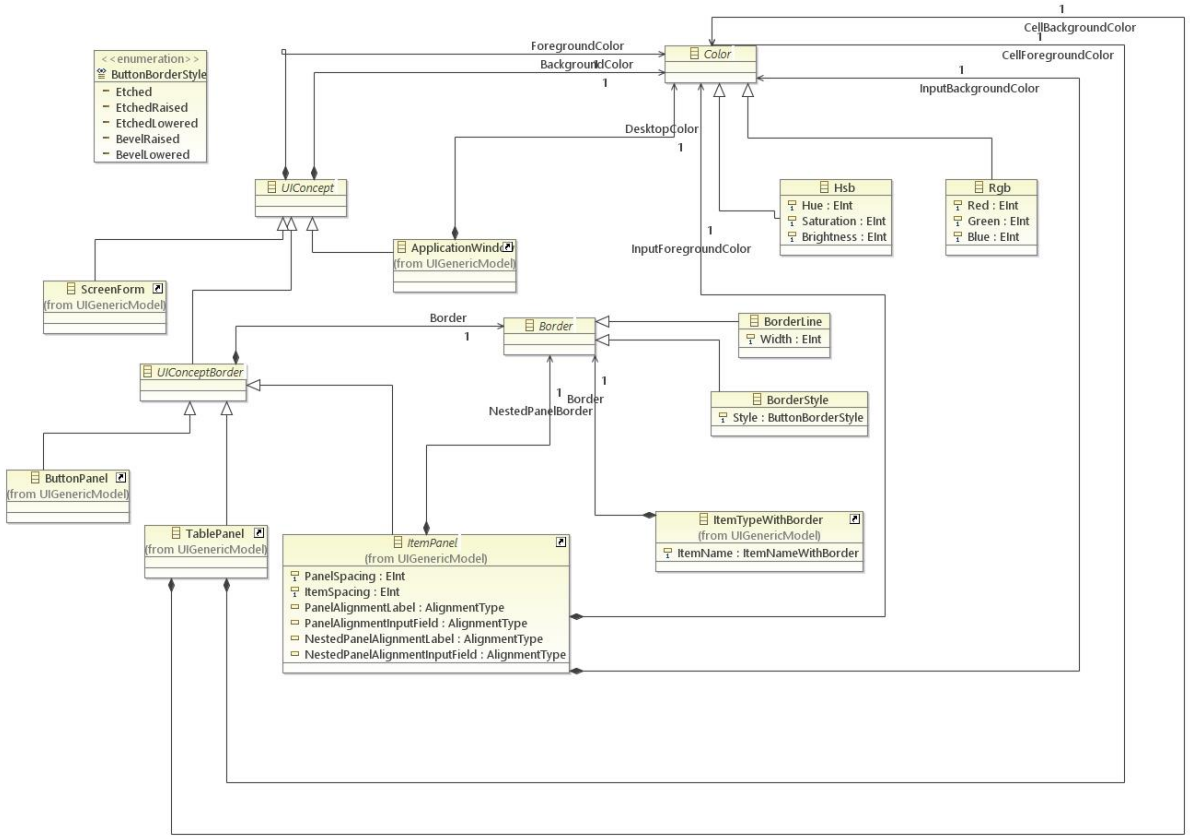


Slika 4.13. Meta-model koncepata modela šablona korisničkog interfejsa

Koncept za opis panela, na dijagramu predstavljen kao apstraktna klasa **ItemPanel**, specificira način na koji će se u okviru korisničkog interfejsa prikazivati grupe polja. Za svaki panel potrebno je specificirati razmak između panela (*PanelSpacing*) i razmak između polja prikazanih u panelu (*ItemSpacing*). Opciono moguće je zadati poravnanje naziva polja za prikaz i ažuriranje podataka unutar panela (*PanelAlignmentLabel*), poravnanje polja za prikaz i ažuriranje podataka unutar panela (*PanelAlignmentInputField*), poravnanje naziva polja za prikaz i ažuriranje podataka unutar ugnježdenih panela (*NestedPanelAlignmentLabel*) i poravnanje polja za prikaz i ažuriranje podataka unutar ugnježdenih panela (Nested*PanelAlignmentInputField*). Moguće vrednosti poravnanja su na početku i na kraju. Ove vrednosti definisane su kao *start* i *end* u okviru enumeracije **AlignementType**. U zavisnosti od tipa panela, panel može biti prikazan kao normalan ili kao panel sa jezičcima. Normalan panel predstavljen je u meta-modelu pomoću klase **ItemPanelNormal,** dok je panel sa jezičcima predstavljen klasom **ItemPanelTabbed**. U slučaju da je za prikaz grupa polja izabran tip panela sa jezičcima može se specificirati kontekstna tabela koja služi za prikaz kontekstne grupe polja.Kontekstna tabela predstavljena je kao klasa **ContexTable**. Za kontekstnu tabelu neophodno je specificirati njenu visinu (*Height*).

Meta-model koncepata modela šablona korisničkog interfejsa za definisanje boja i ivica prikazan je na slici 4.14.Koncept za opis polja panela prikazan je kao klasa **ItemType**. Za

svako polje potrebno je zadati poravnanje naziva polja panela (*TitleAlign*), poziciju naziva u odnosu na polje za prikaz i ažuriranje podataka (*TitlePosition*), razmak naziva od polja za prikaz i ažuriranje podataka (*TitleOffset*) i format naslova italik (*TitleItalic*) i bold (*TitleBold*). Moguće vrednosti za poravnanje naziva polja panela su levo, centrirano i desno i zadaju se kao *Left, Right* i *Centered* iz enumeracije **TitlePositionType.** Moguće vrednosti za poziciju naziva u odnosu na polje za prikaz i ažuriranje podataka su levo, desno, gore, dole i definisane su kao *Left*, *Right*, *Top* i *Bottom* u okviru enumeracije **ItemTitlePosition**. Polje panela može biti predstavljeno kao *Text box*, *Radio button*, *Check box*, *Combo box* i *List.* Polja definisana kao *Text box*,*List* i *Combo box* predstavljena su pomoću klase **ItemTypeWithBorder, a** polja tipa *Radio button*, *Check box* pomoću klase **ItemTypeWithoutBorder.** Vrednosti za tip polja *Text box*,*List,* i *Combo box* definisane su u okviru enumeracije **ItemNameWithBorder,** dok su vrednosti za tip polja *Radio button*, *Check box* definisane u okviru enumeracije **ItemNameWithoutBorder**.



Slika 4.14. Meta-model koncepata modela šablona korisničkog interfejsa za definisanje boja i ivica

Prilikom specifikacije glavne ekranske forme (**ApplicationWindow**), ostalih ekranskih formi (**ScreenForm**), tabela (**TablePanel**), panela (**ItemPanel**) i dugmadi (**ButtonPanel**) potrebno je definisati boju pozadine i boju fonta. Ove zajedničke osobine modelovane su na nivou apstraktne klase **UIConcept** koju nasleđuju prethodno navedene klase. Boja je modelovana pomoću apstraktne klase **Color.** Boju je moguće specificirati na dva načina zadavanjem vrednosti *Hue, Saturation* i Brightness u okviru klase **Hsb** ili zadavanjem vrednosti *Red, Green, Blue* u okviru klase **Rgb**. Klase **Hsb** i **Rgb** nasleđuju klasu **Color.** Za glavni prozor aplikacije potrebno je specificirati i boju desktop-a glavnog prozora aplikacije (*DesktopColor*). Za tabelu potrebno je specificirati i boju pozadine ćelija tabela (*CellBackgroundColor*) i boju fonta teksta u ćelijama tabela (*CellForegroundColor*). Za

panel je potrebno zadati boju pozadine polja prikazanih u panelu (*InputBackgroundColor*) i boju fonta polja prikazanih u panelu (*InputForegroundColor*).

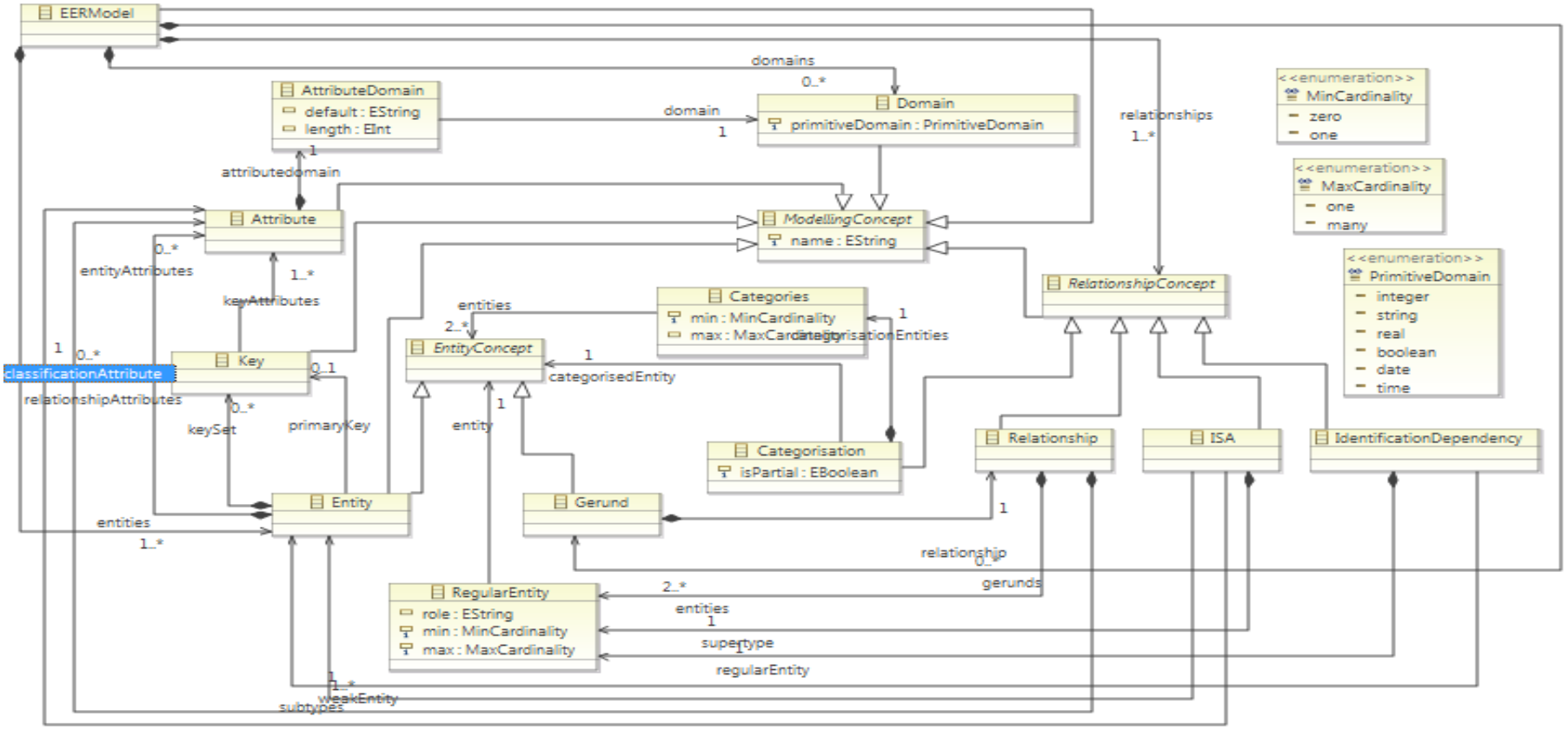
Prilikom specifikacije tabela (**TablePanel**), panela (**ItemPanel**) i dugmadi (**ButtonPanel**) potrebno je definisati ivice panela. Ova zajednička osobina nalazi se u okviru klase **UIConceptBorder** koju nasleđuju prethodno navedene klase. Koncept za specifkaciju ivice predstavljen je klasom **Border.** Ivica može imati predefinisanu vrednost ili može biti zadata njena širina. Predefinisana vrednost predstavljena je pomoću klase **BorderStyle.** Širina ivice (*Width*) definisana je u okviru klase **BorderLine**. Ivicu je neophodno specificirati za ugnježdene panele (*NestedPanelBorder*) kao i za polja panela opisana klasom **ItemTypeWithBorder.**

4.3 Meta-model tipova entiteta i poveznika

U ovom odeljku predstavljeni su koncepti meta-modela EER PIM-a koji su specificirani pomoću Ecore meta-meta-modela. Detaljan opis svih EER PIM koncepata može se naći u Čeliković et al. 2014, Dimitrieski et al. 2014 i Dimitrieski et al. 2015.
Osnovni koncepti modelovanja u EER modelu podataka su: tip entiteta, n-arni tip poveznika, IS-A tip poveznika, identifikaciona zavisnost, kategorizacija, gerund, atribut, ključ i domen. *U ostatku ovog poglavlja predstavljeni su nazivi meta-modela i koncept modela u zagradama i kurzivu.* Korenski koncept u meta-modelu prikazan na slici 4.15 je model (**EERModel**). Svaki EER model sadrži nula ili više tipova entiteta (**Entity**). Koncept tipa entiteta se koristi za modelovanje klase realnih posmatranih entiteta u navedenom IS. Svaki tip entiteta može sadržavati nula ili više atributa (**Attribute**). Koncept atributa se koristi za modelovaanje osobina izabranih stvarnih entiteta u projektovanoj IS.

Domen (**Domain**) predstavlja skup dozvoljenih vrednosti koje atribut može da ima. Svaki domen zasnovan je na primitivnom domenu (**PrimitiveDomain**), kao što su *integer*, *string*, *real*, *boolean*, datum i vreme. Svaki atribut je povezan sa tačno jednim domenom (**AttributeDomain**). Za svaki atribut se može odrediti dužina (**Length**) i podrazumevana vrednost (**Default**). Zbog toga, kako bi se omogućila ponovna upotreba domena, domeni se mogu odrediti jednom na nivou modela EER, a zatim se ponovo koristiti i ograničiti na nivou atributa. Specifikacija entiteta zahteva specifikaciju skupa ključeva (**Key**). Svaki ključ je naveden kao skup atributa tipa entiteta. Samo jedan ključ je proglašen kao primarni ključ (**primaryKey**).

EER model podataka omogućava modelovanje različitih tipova poveznika. Tipovi poveznika između entiteta su klasifikovani kao: n-arni tip poveznika (**Relationship**), identifikacioni tip poveznika (**IdentificationDependency**), IS-A tip poveznika (**ISA**) i kategorizacija (**Categorisation**). N-arni tip poveznika modeluje povezanost između dva ili više tipa entiteta. Za svaki tip entiteta (**RegularEntity**) u vezi potrebno je navesti minimalnu (**MinCardinality**) i maksimalnu (**MaxCardinality**) vrednost kardinaliteta. Minimalni kardinalitet može imati jednu od dve vrednosti: *nula* ili *jedan*, dok maksimalni kardinalitet može dobiti vrednost jedan ili više. Za svaki n-arni tip poveznika, moguće je definisati skup atributa. Gerund (**Gerund**) koncept ima dvojaku ulogu. Gerund, predstavlja istovremeno i tip entiteta i tip poveznika. Koncept gerunda se koristi za specifikaciju pravila koje kombinacije pojava tipova entiteta mogu biti sadržane u pojavi posmatranog tipa poveznika



Slika 4.15. Meta-model tipova entiteta i poveznika

Jedan od glavnih ciljeva ove doktorske teze je definisanje koncepata za modelovanje specifikacija jednog informacionog sistema, na formalan način. Ovako definisani koncepti mogu biti upotrebljeni za izgradnju odgovarajućih namenskih jezika. U skladu sa MDSD, koncepti za modelovanje namenskih jezika definišu se na nivou meta-modela. U ovom poglavlju su predstavljeni meta-modeli neophodni za modelovanje specifikacija jednog informacionog sistema putem namenskih jezika, koji će biti opisani u narednom poglavlju. Svi meta-modeli su izraženi na jedinstven način, pomoću MOF meta-metamodela.

5. Konkretne sintakse namenskih jezika za specifikaciju informacionih sistema

U prethodnom poglavlju prikazani su meta-modeli, koji predstavljaju apstraktnu sintaksu namenskih jezika za modelovanje specifikacija jednog informacionog sistem. Na nivou meta-modela opisani su koncepti koji su namenjeni specifikaciji informaciong sistema. Razvijeni namenski jezici koji su prikazani u ovom poglavlju predstavljaju glavni doprinos ove disertacije. Putem ovih namenskih jezika predstavljen je jedan način implementacije pristupa modelovanju specifikacije informacionog sistema. Predloženi namenski jezici osmišljeni su za primenu u konkretnom domenu izrade aplikacija za rad nad bazama podataka. Saglasno tome, ovakvi jezici svrstavaju se u namenske jezike za domen. U skladu sa tim, koncepti za modelovanje i apstrakcije predstavljenih jezika prilagođeni su domenu u kojem se oni primjenjuju. Namenski jezici povezani su sa domenom problema, čime koncepti i specifičnosti domena postaju deo sintakse i semantike jezika. Na ovaj način, namenski jezici tipično podržavaju koncepte višeg nivoa apstrakcije u odnosu na programske jezike opšte namene. S druge strane, međutim, njihova opštost i izražajnost tipično je sužena u odnosu na programske jezike opšte namene.

Prednosti upotrebe namenskih jezika za modelovanje specifikacije informacionog sistema u odnosu na jezike opšte namene su dvostruke. Namenski jezici usmereni su na konkretan domen primene, pa je skup podržanih koncepata ovog jezika sužen u odnosu na skup podržanih koncepata nekog programskog jezika opšte namene. Ova osobina namenske jezike čini znatno jednostavnijim za učenje i upotrebu. Koncepti jezika bliski su domenu primene što utiče na jednostavnost zadavanja specifikacija. Zadavanje specifikacija putem namenskih jezika poboljšava prenosivost isprojektovanog modela. Specifikacije zadate putem ovakvih jezika ne zavise od konkretne platforme, a takođe mogu da se prevedu i u platformski nezavisan kod. Izborom nove platforme nije potrebno ponovo definisati koncepte za modelovanje specfikacije informacionog sistema.

U ovom poglavlju biće detaljno prikazani konkretna tekstulna sintaksa namenskih jezika za modelovanje specifikacije informacionog sistema. Za specifikaciju konkretne sintakse namenskih jezika upotrebljeno je XText okruženje. XText okruženje poseduje svoju notaciju za specifikaciju konkretne sintakse koja je zasnovana na EBNF notaciji. U postupku specifikacije konkretne sintakse putem XText okruženja mogu se koristiti meta-modeli definisani pomoću Ecore jezika. Na ovaj način meta-modeli opisani u prethodnom poglavlju mogu da posluže kao apstraktna sintaksa, za namenske jezike predtsavljene u ovom poglavlju.

U odeljcima od 5.1 do 5.3 biće prikazane tekstualne konkretne sintakse sledećih namenskih jezika:

- FTDSL namenskog jezika za modelovanje specifikacije informacionog Sistema putem koncepta tipa forme,
- UIDSL namenskog jezika za modelovanje specifikacija šablona korisničkog interfejsa i
- EERDSL namenskog jezika za modelovanje specifikacije informacionog sistema putem koncepata tipova entiteta i poveznika.

5.1 FTDSL konkretna sintaksa

Gramatičko pravilo za opis instance **PrimitiveDomain** koncepta, koji predstavlja primitivni domen pomoću tekstualne sintakse je:

```
PrimitiveDomain returns PrimitiveDomain:
   'PrimitiveDomain'
   Name=EString
   (DefaultLength=EInt)?
   ('.' DefaultDecimalPlaces=EInt)?
   ('default' DefaultValue=EString)?
   ('compatible' CompatibileDomain=[Domain|EString])?
   'description' Description=EString
   ;
```

U okviru pravila za primitvni domen moguće je specificirati: naziv (**Name**), predefinisanu dužinu (**DefaultLength**), prdefinisani broj mesta iza zareza (**DefaultDecimalPlaces**), predefinisanu vrednost (**DefaultValue**), kompatibilni domen (**CompatibileDomain**) i opis (**Description**).

Za opis instance **UserDefinedDomain** koncepta koji reprezentuje korisnički definisan domen putem tekstualne sintakse definisana su dva pravila jer je moguće da ovakav domen nasledi primitivni domen ili neki drugi korisnički definisan domen. Za opis domena koji nasleđuje primitvni domen gramatičko pravilo je:

```
UserDefinedDomainFromPrimitiveDomain returns
UserDefinedDomainFromPrimitiveDomain:
   'UserDefinedDomainFromPrimitiveDomain'
   Name=EString
   ':' InheritsPrimitiveDomain=[PrimitiveDomain|EString]
   (LengthValue=EInt)?
   ('.' DecimalPLaces=EInt)?
   ('default' DefaultValue=EString)?
   ('compatible' CompatibileDomain=[Domain|EString])?
   ('check' CheckCondition=EString)?
   'description' Description=EString
   ;
```

Za opis domena koji nasleđuje korisnički definisan domen gramatičko pravilo glasi:

```
UserDefinedDomainFromUserDefinedDomain returns
UserDefinedDomainFromUserDefinedDomain:
   'UserDefinedDomainFromUserDefinedDomain'
   Name=EString
   ':' InheritsUserDefinedDomain=[UserDefinedDomainType|EString]
   ('default' DefaultValue=EString)?
   ('compatible' CompatibileDomain=[Domain|EString])?
   ('check' CheckCondition=EString)?
   'description' Description=EString
   ;
```

U okviru pravila za korisnički definisan domen moguće je specificirati: naziv (**Name**), predefinisanu vrednost (**DefaultValue**), kompatibilni domen (**CompatibileDomain**), ograničenje (**CheckCondition**), i opis (**Description**). Ukoliko je domen nasleđen iz

primitivnog domena potrebno je specificirati **InheritsPrimitiveDomain** opciju, a ukoliko je nasleđen iz drugog korisnički definisanog domena (**InheritsUserDefinedDomain**).

Gramatičko pravilo za opis instance **Attribute** koncepta koji reprezentuje attribut pomoću tekstualne sintakse je:

```
Attribute returns Attribute:
   'Attribute'
   Name=EString
   ':' AttributeDomain=[UserDefinedDomainType|EString]
   ('default' DefaultValue=EString)?
   ('expression' Expression=EString)?
   ('renamed' RenamedFromAttribute=[Attribute|EString])?
   'description' Description=EString
   ;
```

U okviru pravila za atribut moguće je specificirati: naziv (**Name**), domen atributa (**AttributeDomain**), predefinisanu vrednost (**DefaultValue**), izraz (**Expression**), atribut koji je preimenovan (**RenamedFromAttribute**), i opis (**Description**).

Gramatičko pravilo za opis instance **Package** koncepta koji reprezentuje pakete pomoću tekstualne sintakse glasi:

```
Package returns Package:
   'Package'
   Name=EString
   ('(' Functions+=[Function|EString] ( "," Functions+=[Function|EString])*
')' )?
   'Description' Description=EString
   ;
```

U okviru pravila za paket moguće je specificirati: naziv (**Name**), skup funkcija (**Functions**), i opis (**Description**).

Gramatičko pravilo za opis instance **Function** koncepta koji reprezentuje funkcije pomoću tekstualne sintakse glasi:

```
Function returns Function:
   'Function'
   (ReturnType=[UserDefinedDomainType|EString])?
   Name=EString
   ('(' ParameterList+=Parameter ( "," ParameterList+=Parameter)* ')' )?
   '{'
         (Body=EString)?
   '}'
   'description' Description=EString;
```

U okviru pravila za definiciju funckije moguće je specificirati: naziv (**Name**), skup parametara (**ParameterList**), tip povratne vrednosti (**ReturnType**) i opis (**Description**).

Gramatičko pravilo za opis instance **Parameter** koncepta koji reprezentuje paremetre funkcije pomoću tekstualne sintakse je:

```
Parameter returns Parameter:
   'Parameter'
    SequnceNumber=EInt ')'
```

```
   Name=EString
   ':' ReferencesDomain=[UserDefinedDomainType|EString]
   (Type=ParameterDefinition)?
   ('default' DefaultValue=EString)?
   ;

enum ParameterDefinition returns ParameterDefinition:
                        In = 'In' | Out = 'Out' | InOut = 'InOut';
```

U okviru pravila za definiciju parametra funckije moguće je specificirati: naziv (**Name**), redni broj parametra (**SequnceNumber**), domen (**ReferencesDomain**), predefinisanu vrednost (**DefaultValue**), i tip (**Type**) ako je parametar ulazni (**In**), izlazni (**Out**) ili ulazno-izlazni (**InOut**).

Gramatičko pravilo za opis instance **ApplicationSystem** koncepta koji reprezentuje jedan aplikativni sistem pomoću tekstualne sintakse je:

```
ApplicationSystem returns ApplicationSystem:
   Name=EString
   'description' Description=EString
   '{'
         ('ownedFormTypes' '{' OwnedFormTypes+=FormType ( ","
OwnedFormTypes+=FormType)* '}' )?
         ('referencedFormTypes' '(' ReferencedFormTypes+=[FormType|EString] (
"," ReferencedFormTypes+=[FormType|EString])* ')' )?
         ('businessApplications' '{'
BusinessApplications+=BussinesApplication ( ","
BusinessApplications+=BussinesApplication)* '}' )?
         ('childrenApplicationSystems' '('
ChildrenApplicationSystems+=[ApplicationSystem|EString] ( ","
ChildrenApplicationSystems+=[ApplicationSystem|EString])* ')' )?
   '}';
```

U okviru pravila za definiciju aplikativnog sistema moguće je specificirati: naziv (**Name**), skup tipova formi (**OwnedFormTypes**), skup poslovnih aplikacija (**BusinessApplications**), skup podređenih aplikativnoh sistema, aplikacija (**ChildrenApplicationSystems**), i opis (**Description**).

Gramatičko pravilo za opis instance **FormTypeProgram** koncepta koji reprezentuje jedan tip forme pomoću tekstualne sintakse je:

```
FormTypeProgram returns FormTypeProgram:
   'FormTypeProgram'
    Name=EString
    '(' Title=EString ')'
   (ConsideredINDBSchDesign?='ConsideredINDBSchDesign')?
   ('frequency' Frequency=EInt)?
   ('responseTime' ResponseTime=EInt)?
   'componentTypeRoot' ComponentTypeRoot=ComponentType
   ;
```

U okviru pravila za definiciju tipa forme moguće je specificirati: naziv (**Name**), naslov (**Title**), vreme odziva (**ResponseTime**), učestalost upotrebe (**Frequency**) i korenski tip komponente (**ComponentTypeRoot**).

Gramatičko pravilo za opis instance **ComponentType** koncepta koji reprezentuje tip komponente pomoću tekstualne sintakse je:

```
ComponentType returns ComponentType:
    Name=EString
    '(' Title=EString ')'
    (Query?='Query')?
   (Insert?='Insert')?
   (Delete?='Delete')?
   (Update?='Update')?
   '{'
         'attributes' '{' ComponentTypeAttributes+=ComponentTypeAttribute (
"," ComponentTypeAttributes+=ComponentTypeAttribute)* '}'
         'keys' '{' ComponentTypeKeys+=ComponentTypeKey ( ","
ComponentTypeKeys+=ComponentTypeKey)* '}'
         ('uniques' '{' ComponentTypeUniques+=ComponentTypeUnique ( ","
ComponentTypeUniques+=ComponentTypeUnique)* '}' )?
         ('check' ComponentTypeCheck=ComponentTypeCheck)?
         ('componentTypesChildren' '{' ComponentTypeChildren+=ComponentType (
"," ComponentTypeChildren+=ComponentType)* '}' )?
   '}';
```

U okviru pravila za definiciju tipa komponente moguće je specificirati: naziv (**Name**), skup atributa (**ComponentTypeAttributes**), skup ograničenja kluča (**ComponentTypeKeys**), skup ograničenja jedinstvenosti (**ComponentTypeUniques**), ograničenj torke (**ComponentTypeCheck**), skup podređenih tipova komponenti (**ComponentTypeChildren**), skup dozvoljenih operacija: čitanje (**Query**), unos (**Insert**), izmena (**Update**), brisanje (**Delete**).

Gramatičko pravilo za opis instance **BusinessApplication** koncepta koji predstavlja poslovnu aplikaciju pomoću tekstualne sintakse je:

```
BussinesApplication returns BussinesApplication:
   Name=EString
   'description' Description=EString
   '{'
         ('formTypes' '(' FormTypes+=[FormType|EString] ( ","
FormTypes+=[FormType|EString])* ')' )?
         'entryFormType' EntryFormType=[FormType|EString]
   '}';
```

U okviru pravila za definiciju poslovne aplikacije moguće je specificirati: naziv (**Name**), skup tipova formi (**FormTypes**), ulazni tip forme (**EntryFormType**) i opis (**Description**).

Gramatičko pravilo za opis instance **Key** koncepta koji predstavlja ograničenje ključa pomoću tekstualne sintakse je:

```
ComponentTypeKey returns ComponentTypeKey:
    Name=EString
    '(' ComponentTypeKeyAttributes+=[ComponentTypeAttribute|EString] ( ","
ComponentTypeKeyAttributes+=[ComponentTypeAttribute|EString])* ')'
    (Global?='Global')?
   ;
```

U okviru pravila za definiciju ograničenja ključa moguće je specificirati: naziv (**Name**) i, skup atributa (**ComponentTypeKeyAttributes**).

Gramatičko pravilo za opis instance **Unique** koncepta koji reprezentuje ograničenje jedinstvenosti pomoću tekstualne sintakse glasi:

```
ComponentTypeUnique returns ComponentTypeUnique:
   Name=EString
   '(' ComponentTypeUniqueAttributes+=[ComponentTypeAttribute|EString] ( ","
ComponentTypeUniqueAttributes+=[ComponentTypeAttribute|EString])* ')'
   ;
```

U okviru pravila za definiciju ograničenja jedinstvenosti moguće je specificirati: naziv (**Name**) i, skup atributa (**ComponentTypeKeyAttributes**).

Gramatičko pravilo za opis instance **Check** koncepta koji predstavlja ograničenje torke pomoću tekstualne sintakse glasi:

```
ComponentTypeCheck returns ComponentTypeCheck:
   Name=EString Expression=EString
   ;
```

U okviru pravila za definiciju ograničenja torke moguće je specificirati: naziv (**Name**) i izraz (**Expression**).

5.2 UIDSL Konkretna tekstualna sintaksa

UIGenericModel je pravilo, koje se nalazi na vrhu hijerarhije UIDSL gramatike. Ovo gramatičko pravilo predstavlja model šablona korisničkog interfejsa.

```
UIGenericModel returns UIGenericModel:: UIGenericModel:
   'Template'
   '{'
         'Name' Name=EString
         'ItemFont' ItemFont=Font
         'LabelFont' LabelFont=Font
         'ApplicationWindowProperty'
   '}'
```

Za svaki šablon potrebno je zadati specifikaciju tipa, veličine i formata fonta polja za prikaz i ažuriranje podataka (*ItemFont*), kao i fonta za naziv polja za prikaz i ažuriranje podataka (*LabelFont*).

Gramatičko pravilo za opis instance **Font** koncepta koji predstvalja opis fonta pomoću tekstualne sintakse glasi:

```
Font returns UIGenericModel::Font:
   '{'
         'Style' Style=EString
         'Size' Size=EInt
         'Bold' Bold?=EBoolean
         'Italic' Italic?=EBoolean
         'Underline' Underline?=EBoolean
   '}';
```

U okviru pravila za definiciju fonta moguće je specificirati: stil (**Style**), veličinu (**Size**), kao i način prikaza (**Bold, Italic, Underline**).

Gramatičko pravilo za opis instance **ApplicationWindow** koncepta koji predstvalja opis glavne ekranske forme pomoću tekstualne sintakse glasi:

```
ApplicationWindow returns UIGenericModel::ApplicationWindow:
   '{'
         'Resizeable' Resizeable?=EBoolean
         ('ApplicationIconPath' ApplicationIconPath=EString)?
         'ForegroundColor' ForegroundColor=Color
         'BackgroundColor' BackgroundColor=Color
         'DesktopColor' DesktopColor=Color
         ScreenSize=ScreenSize
   '}';
```

U okviru pravila za definiciju glavne ekranske forme moguće je specificirati: putanju do ikone aplikacije (**ApplicationIconPath**), da li je moguće menjati veličinu prozora (**Resizeable**), kao i veličinu prozora (**ScreenSize**).

Veličina prozora definisana je u okvriru gramatičkog pravila **ScreenSize** koje glasi:

```
ScreenSize returns UIGenericModel::ScreenSize:
   FullScreen | FixedSize;
```

Maksimalna veličina prozora definisana je pomoću gramatičkog pravila **FullScreen** koje glasi:

```
FullScreen returns UIGenericModel::FullScreen:
   { UIGenericModel::FullScreen}
   'FullScreen'
   ;
```

Fiksna veličina prozora definisana je gramatičkim pravilom **FixedSize**.

```
FixedSize returns UIGenericModel::FixedSize:
   'FixedScreenSize'
   '{'
         'Width' Width=EInt
         'Height' Height=EInt
   '}';
```

U okviru pravila za definiciju fiksne veličine prozora, potrebno je zadati visinu (**Height**) i širinu (**Width**) glavnog prozora aplikacije.

Gramatičko pravilo za opis instance **ScreenForm_Impl** koncepta koji reprezentuje ostale ekranske forme pomoću tekstualne sintakse je:

```
ScreenForm_Impl returns UIGenericModel::ScreenForm:
   '{'
         'CallFromMenu' CallFromMenu?=EBoolean
         'CallButtonClick' CallButtonClick?=EBoolean
         'CallKeyPress' CallKeyPress?=EBoolean
         'ForegroundColor' ForegroundColor=Color
         'BackgroundColor' BackgroundColor=Color
```

```
    '}';
```

U okviru pravila za definiciju ostalih ekranskih formi moguće je specificirati: da li je moguć poziv ekranskih formi iz menija (**CallFromMenu**), klikom na dugme (**CallButtonClick**) i pritiskom na odgovarajuće dugme sa tastature (**CallKeyPress**).

Gramatičko pravilo za opis instance **ScreenFormWithTitle** koncepta koji reprezentuje ostale ekranske forme za koje je potrebno prikazati naslov, pomoću tekstualne sintakse je:

```
ScreenFormWithTitle returns UIGenericModel::ScreenFormWithTitle:
   '{'
         'FormTitlePosition' FormTitlePosition=TitlePositionType
         'ForegroundColor' ForegroundColor=Color
         'BackgroundColor' BackgroundColor=Color
         'FormTitleFont' FormTitleFont=Font
         'CallFromMenu' CallFromMenu?=EBoolean
         'CallButtonClick' CallButtonClick?=EBoolean
         'CallKeyPress' CallKeyPress?=EBoolean
   '}';
```

U okviru ovog pravila za definiciju ostalih ekranskih formi moguće je specificirati tip i veličinu fonta (**FormTitleFont**) kao i poziciju naslova (**FormTitlePosition**).

Gramatičko pravilo za opis instance **ItemType** koncepta koji reprezentuje opis polja panela pomoću tekstualne sintakse je:.

```
ItemType returns UIGenericModel::ItemType:
   ItemTypeWithBorder | ItemTypeWithoutBorder;
```

Polje panela može biti predstavljeno kao *Text box*, *Radio button*, *Check box*, *Combo box* i *List.* Polja definisana kao *Text box,List* i *Combo box* predstavljena su pomoću klase **ItemTypeWithBorder, a** polja tipa *Radio button*, *Check box* pomoću klase **ItemTypeWithoutBorder.**

Gramatičko pravilo za opis instance **ItemTypeWithBorder** koncepta koji reprezentuje opis polja panela tipa *Text box,List* i *Combo box* pomoću tekstualne sintakse je:.

```
ItemTypeWithBorder returns UIGenericModel::ItemTypeWithBorder:
   '{'
         'ItemName' ItemName=ItemNameWithBorder
         'TitleAlign' TitleAlign=TitlePositionType
         'TitlePosition' TitlePosition=ItemTitlePosition
         'TitleOffset' TitleOffset=EInt
         'TitleItalic' TitleItalic?=EBoolean
         'TitleBold' TitleBold?=EBoolean
         Border=Border
   '}';
```

U okviru ovog pravila za definiciju polja potrebno je zadati poravnanje naziva polja panela (**TitleAlign**), poziciju naziva u odnosu na polje za prikaz i ažuriranje podataka (**TitlePosition**), razmak naziva od polja za prikaz i ažuriranje podataka (**TitleOffset**) i format naslova italik (**TitleItalic**) i bold (**TitleBold**.

Gramatičko pravilo za opis instance **ItemTypeWithoutBorder** koncepta koji reprezentuje opis polja panela tipa *Radio button, Check box* pomoću tekstualne sintakse je:.

```
ItemTypeWithoutBorder returns UIGenericModel::ItemTypeWithoutBorder:
   '{'
         'ItemName' ItemName=ItemNameWithoutBorder
         'TitleAlign' TitleAlign=TitlePositionType
         'TitlePosition' TitlePosition=ItemTitlePosition
         'TitleOffset' TitleOffset=EInt
         'TitleItalic' TitleItalic?=EBoolean
         'TitleBold' TitleBold?=EBoolean
   '}';
```

Moguće vrednosti za poravnanje naziva polja panela su levo, centrirano i desno i zadaju se kao *Left, Right* i *Centered* u okviru gramatičkog pravila **TitlePositionType.**

```
TitlePositionType returns UIGenericModel::TitlePositionType:
   'Centered' | 'Left' | 'Right';
```

Moguće vrednosti za poziciju naziva u odnosu na polje za prikaz i ažuriranje podataka su levo, desno, gore, dole i definisane su kao *Left*, *Right*, *Top* i *Bottom* u okviru gramatičkog pravila **ItemTitlePosition**.

```
ItemTitlePosition returns UIGenericModel::ItemTitlePosition:
   'Left' | 'Right' | 'Top' | 'Bottom';
```

Vrednosti za tip polja *Text box,List,* i *Combo box* definisane su u okviru gramatičkog pravila **ItemNameWithBorder.**

```
ItemNameWithBorder returns UIGenericModel::ItemNameWithBorder:
   'TextBox' | 'List' | 'ComboBox';
```

Vrednosti za tip polja *Radio button, Check box* definisane su u okviru gramatičkog pravila **ItemNameWithoutBorder.**

```
ItemNameWithoutBorder returns UIGenericModel::ItemNameWithoutBorder:
   'CheckBox' | 'RadioButton';
```

Gramatičko pravilo za opis instance **ItemPanel** koncepta koji reprezentuje opis panela koji specificira način na koji će se u okviru korisničkog interfejsa prikazivati grupe polja pomoću tekstualne sintakse je:.

```
ItemPanel returns UIGenericModel::ItemPanel:
   ItemPanelNormal | ItemPanelTabbed;
```

Panel može biti specificiran kao normalan panel (**ItemPanelNormal),** ili panel sa jezičcima (**ItemPanelTabbed)**.

Gramatičko pravilo za opis opis instance **ItemPanelNormal** koncepta koja reprezentuje normalan panel pomoću tekstualne sintakse je:

```
ItemPanelNormal returns UIGenericModel::ItemPanelNormal:
   '{'
         'PanelSpacing' PanelSpacing=EInt
         'ItemSpacing' ItemSpacing=EInt
```

```
        ('PanelAlignmentLabel' PanelAlignmentLabel=AlignmentType)?
        ('PanelAlignmentInputField' PanelAlignmentInputField=AlignmentType)?
        ('NestedPanelAlignmentLabel'
NestedPanelAlignmentLabel=AlignmentType)?
        ('NestedPanelAlignmentInputField'
NestedPanelAlignmentInputField=AlignmentType)?
        'NoInLine' NoInLine=EInt
        'ForegroundColor' ForegroundColor=Color
        'BackgroundColor' BackgroundColor=Color
        Border=Border
        'NestedPanelBorder' NestedPanelBorder=Border
        'InputBackgroundColor' InputBackgroundColor=Color
        'InputForegroundColor' InputForegroundColor=Color
   '}';
```

Gramatičko pravilo za opis opis instance **ItemPanelTabbed** koncepta koja reprezentuje panel sa jezičcima pomoću tekstualne sintakse je:

```
ItemPanelTabbed returns UIGenericModel::ItemPanelTabbed:
   '{'
        'PanelSpacing' PanelSpacing=EInt
        'ItemSpacing' ItemSpacing=EInt
        ('PanelAlignmentLabel' PanelAlignmentLabel=AlignmentType)?
        ('PanelAlignmentInputField' PanelAlignmentInputField=AlignmentType)?
        ('NestedPanelAlignmentLabel'
NestedPanelAlignmentLabel=AlignmentType)?
        ('NestedPanelAlignmentInputField'
NestedPanelAlignmentInputField=AlignmentType)?
        'ForegroundColor' ForegroundColor=Color
        'BackgroundColor' BackgroundColor=Color
        Border=Border
        'NestedPanelBorder' NestedPanelBorder=Border
        'InputBackgroundColor' InputBackgroundColor=Color
        'InputForegroundColor' InputForegroundColor=Color
        ('ContexTable' ContexTable=ContexTable)?
   '}';
```

U slučaju da je za prikaz grupa polja izabran tip panela sa jezičcima može se specificirati kontekstna tabela koja služi za prikaz kontekstne grupe polja. Gramatičko pravilo za opis instance **ContexTable** koncepta koje reprezentuje boju kontekstnu tabelu pomoću tekstualne sintakse je:

```
ContexTable returns UIGenericModel::ContexTable:
   '{'
        'Height' Height=EInt
   '}';
```

U okviru pravila za opis kontekstne tabele potrebno je specificirati njenu visunu (**Height**).

Gramatičko pravilo za opis opis instance **Color** koncepta koji reprezentuje boju pomoću tekstualne sintakse je:

```
Color returns UIConcept::Color:
   Hsb | Rgb;
```

U okviru gramatičkog pravila boju je moguće specificirati na dva načina zadavanjem vrednosti *Hue, Saturation* i Brightness u okviru gramtičkog pravila **Hsb** ili zadavanjem vrednosti *Red, Green, Blue* u okviru gramtičkog pravila **Rgb**.

```
Hsb returns UIConcept::Hsb:
   'Hsb'
   '{'
         'Hue' Hue=EInt
         'Saturation' Saturation=EInt
         'Brightness' Brightness=EInt
   '}';

Rgb returns UIConcept::Rgb:
   'Rgb'
   '{'
         'Red' Red=EInt
         'Green' Green=EInt
         'Blue' Blue=EInt
   '}';
```

Prilikom specifikacije tabela (**TablePanel**), panela (**ItemPanel**) i dugmadi (**ButtonPanel**) potrebno je definisati ivice panela. Gramatičko pravilo instance **Border** koja reprezenule ivice pomoću tekstualne sintakse je:

```
Border returns UIConcept::Border:
   BorderStyle | BorderLine;
```

Ivicu je neophodno specificirati za ugnježdene panele (**NestedPanelBorder**) kao i za polja panela opisana gramtičkim pravilom **ItemTypeWithBorder.**
Ivica može imati predefinisanu vrednost ili može biti zadata njena širina. Predefinisana vrednost predstavljena je pomoću gramatičkog pravila **BorderStyle.**

```
BorderStyle returns UIConcept::BorderStyle:
   'BorderStyle' Style=ButtonBorderStyle;
```

Širina ivice (*Width*) definisana je u okviru gramatičkog pravila **BorderLine**.

```
BorderLine returns UIConcept::BorderLine:
   'BorderLineWidth' Width=EInt;
```

Gramatičko pravilo za opis opis instance **ButtonPanel** koja reprezentuje opis dugmadi pomoću tekstualne sintakse glasi:

```
ButtonPanel returns UIGenericModel::ButtonPanel:
   '{'
         'View' View=ButtonViewType
         'ForegroundColor' ForegroundColor=Color
         'BackgroundColor' BackgroundColor=Color
         Border=Border
   '}';
```

U okviru pravila podržana su dva tipa prikaza dugmadi tekstualni i grafički. Način prikaza zadaje se pomoću atributa **View.** Predefinisane vrednosti *Textual* i Iconic definisani su u okviru gramatičkog pravila **ButtonViewType.**

```
ButtonViewType returns UIGenericModel::ButtonViewType:
   'Textual' | 'Iconic';
```

Gramatičko pravilo za opis opis instance **TablePanel** koncepta koja reprezentuje opis tabela pomoću tekstualne sintakse glasi:

```
TablePanel returns UIGenericModel::TablePanel:
   '{'
         'RowHeight' RowHeight=EInt
         'ColumnWidth' ColumnWidth=EInt
         'NoOfVisibleRows' NoOfVisibleRows=EInt
         'ForegroundColor' ForegroundColor=Color
         'BackgroundColor' BackgroundColor=Color
         Border=Border
         ('TableOverflow' TableOverflow=Overflow)?
         'CellBackgroundColor' CellBackgroundColor=Color
         'CellForegroundColor' CellForegroundColor=Color
   '}';
```

Za svaki opis tabele potrebno je specificirati visinu redova (**RowHeight**), širinu kolona (**ColumnWidth**) i broj vidljivih redova u tabelama kojim je određena visina tabele (**NoOfVisibleRows**). Specijalno, u slučaju postojanja **Overflow** grupe polja, mora se zadati da li će data grupa polja biti prikazana u posebnoj **Overflow** oblasti, kao i veličina date oblasti. Gramatičko pravilo za opis opis instance **Overflow** koncepta koja reprezentuje Overflow grupe polja pomoću tekstualne sintakse glasi:

```
Overflow returns UIGenericModel::Overflow:
   '{'
         'Size' Size=EInt
         'Position' Position=OverflowPosition
   '}';
```

Pri specifikaciji ovog koncepta potrebno je zadati visinu **Overflow** grupe polja (**Size**) i poziciju na kojoj se prikazuje (**Position**). Pozicija *overflow* grupe može biti sa desne ili donje strane u odnosu na tabelu i zadaje se kao predefinisana vrednost **Right** ili **Bottom** u okviru gramatičkog pravila **OverflowPosition** koje glasi:

```
OverflowPosition returns UIGenericModel::OverflowPosition:
   'Right' | 'Bottom';
```

5.3 EERDSL Konkretna tekstualna sintaksa

Gramatičko pravilo za opis opis instance **ERModel** koncepta koja reprezentuje opis modela pomoću tekstualne sintakse glasi:

```
ERModel returns ERModel:
   name=Estring
   '{'
           ('domains' '{' domains+=Domain ( "," domains+=Domain)* '}' )?
           ('entities' '{' entities+=Entity ( "," entities+=Entity)* '}')?
           ('gerunds' '{' gerunds+=Gerund ( "," gerunds+=Gerund)* '}' )?
           ('relationships' '{' relationships+=RelationshipConcept ( ","
relationships+=RelationshipConcept)* '}')?
   '}';
```

U okviru pravila za definiciju ER modela moguće je specificirati: domene (**domains**), tipove entiteta (**entities**), gerunde (**gerunds**), tipove poveznika (**relationships**).

Gramatičko pravilo za opis opis instance **RelationshipConcept** koncepta koja reprezentuje opis tipa poveznika pomoću tekstualne sintakse glasi:

```
RelationshipConcept returns RelationshipConcept:
   Relationship | ISA | Categorisation | IdentificationDependency;
```

Tip poveznika može biti specificiran kao n-arni tip poveznika (**Relationship**), IS-A hijerarhija (**ISA**), kategorizacija (**Categorisation**) ili identifikaciona zavisnost (**IdentificationDependency**).

Gramatičko pravilo za opis opis instance **Entity** koncepta koja reprezentuje opis tipa entiteta pomoću tekstualne sintakse glasi:

```
Entity returns Entity:
   name=EString
   '{'
          ('attributeSet' '{' entityAttributes+=Attribute ( ","
entityAttributes+=Attribute)* '}' )?
          ('keySet' '{' keySet+=Key ( "," keySet+=Key)* '}' )?
          ('primaryKey' primaryKey=[Key | EString])?
   '}';
```

U okviru pravila za definiciju tipa entiteta moguće je specificirati: naziv (**name**), skup obeležja (**attributeSet**), skup ključeva (**keySet**), i primarni ključ (**primaryKey**).
Gramatičko pravilo za opis opis instance **Gerund** koncepta koja reprezentuje opis gerunda pomoću tekstualne sintakse glasi:

```
Gerund returns Gerund:
   relationship=Relationship;
```

U okviru pravila za definiciju gerunda potrebno je specificirati n-arni tip poveznika od kojeg nastaje gerund (**relationship**).
Gramatičko pravilo za opis opis instance **Attribute** koncepta koja reprezentuje opis obeležja pomoću tekstualne sintakse glasi:

```
Attribute returns Attribute:
   {Attribute}
   name=EString
   (attributeDomain=AttributeDomain)?;
```

U okviru pravila za definiciju obeležja moguće je specificirati naziv obeležja (**name**) i domen obeležja (**attributeDomain**). Gramatičko pravilo za opis instance **AttributeDomain** koncepta koja reprezentuje opis domena konkretnog obeležja pomoću tekstualne sintakse glasi:

```
AttributeDomain returns AttributeDomain:
   domain=[Domain|EString]
   ('(' length=EInt ')')?
   ('default' default=EString)?;
```

U okviru pravila za definiciju domena konkretnog obeležja moguće je specificirati konkretan domen (**domain**), dužinu (**length**), i predefinisanu vrednost (**default**).

Gramatičko pravilo za opis opis instance **Key** koncepta koja reprezentuje opis ključa tipa entiteta pomoću tekstualne sintakse glasi:

```
Key returns Key:
   name=EString
   '(' keyAttributes+=[Attribute|EString] ( ","
keyAttributes+=[Attribute|EString])* ')';
```

U okviru pravila za definiciju ključa tipa entiteta moguće je definisati naziv ključa (**name**) i skup obeležja koji pripadaju ključu (**keyAttributes**).

Gramatičko pravilo za opis opis instance **Domain** koncepta koja reprezentuje opis domena pomoću tekstualne sintakse glasi:

```
Domain returns Domain:
      {Domain}
      name=EString
      'extends' primitiveDomain=PrimitiveDomain;
```

U okviru pravila za definiciju domena moguće je specificirati naziv domena (**name**) i nasleđeni primitivni domen (**primitiveDomain**). Primitivni domen može biti *integer, string, real, boolean, date, time.* Ove vrednosti definisane su putem gramatičkog pravila **PrimitiveDomain** koje glasi:

```
PrimitiveDomain returns PrimitiveDomain:
                      'integer' | 'string' | 'real' | 'boolean' | 'date' |
'time'
```

Gramatičko pravilo za opis opis instance **Relationship** koncepta koja reprezentuje opis n-arnog tipa poveznika pomoću tekstualne sintakse glasi:

```
Relationship returns Relationship:
   name=EString
   '{'
         'entitiesInRelationship' '{' entities+=RegularEntity ( ","
entities+=RegularEntity)* '}'
         ('attributeSet' '{' relationshipAttributes+=Attribute ( ","
relationshipAttributes+=Attribute)* '}' )?
   '}';
```

U okviru pravila za definiciju n-arnog tipa poveznika moguće je definisati: naziv (**name**), tipove entiteta koje tip poveznika povezuje (**entitiesInRelationship**), skup obeležja (**attributeSet**).

Gramatičko pravilo za opis opis instance **ISA** koncepta koja reprezentuje opis IS-A hijerarhije pomoću tekstualne sintakse glasi:

```
ISA returns ISA:
   'ISA'
   name=EString
```

```
   '{'
         'supertype' supertype=RegularEntity
         'subtypes' '(' subtypes+=[Entity|EString] ( ","
subtypes+=[Entity|EString])* ')'
         'classificationAttribute'
classificationAttribute=[Attribute|EString]
   '}';
```

U okviru pravila za definiciju IS-A hijerarhije moguće je specificirati: naziv (**name**), nadtip (**supertype**), skup podtipova (**subtypes**) i klasifikaciono obeležje (**classificationAttribute**).

Gramatičko pravilo za opis opis instance **Categorisation** koncepta koja reprezentuje opis kategorizacije pomoću tekstualne sintakse glasi:

```
Categorisation returns Categorisation:

   'Categorisation'
   name=EString
   '{'
         'categorisedEntity' categorisedEntity=[EntityConcept|EString]
         'categorisationEntities' categorisationEntities=Categories
         'isPartial' isPartial?=EBoolean
   '}';
```

U okviru pravila za definiciju kategorizacije moguće je specificirati: naziv (**name**), kategorisan tip entiteta (**categorisedEntity**) i kategorije (**categorisationEntities)**, koje su definisane kroz skup instanci koje su definisane putem gramatičkog pravila **Categories** koje glasi:

```
Categories returns Categories:
   '(' entities+=[EntityConcept|EString] ( ","
entities+=[EntityConcept|EString])* ')'
   '('
   min=MinCardinality ','
   max=MaxCardinality
   ')'
   ;
```

U okviru pravila za definiciju tipova entiteta koje su kategorije moguće je specificirati konkretan tip entiteta, kao i vrednosti za minimalan (**min**) i maksimalan (**max**) kardinalitet.

Gramatičko pravilo za opis opis instance **IdentificationDependency** koncepta koja reprezentuje opis identifikacionog tipa poveznika pomoću tekstualne sintakse glasi:

```
IdentificationDependency returns IdentificationDependency:
   'IdentificationDependency'
   name=EString
   '{'
         'weakEntity' weakEntity=[Entity|EString]
         'regularEntity' regularEntity=RegularEntity
   '}';
```

U okviru pravila za definiciju identifikacionog tipa poveznika moguće je specificirati naziv (**name**), slabi tip entiteta (**weakEntity**) i regularni tip entiteta (**regularEntity**).

Gramatičko pravilo za opis instance **RegularEntity** koncepta koji reprezentuje regularni tip entiteta , odnosno tip entiteta koji učestvuje u vezi sa drugim tipom entiteta pomoću tekstualne sintakse glasi:

```
RegularEntity returns RegularEntity:
   entity=[EntityConcept|EString]
          '('
          min=MinCardinality ','
          max=MaxCardinality
          ')'
          (role=EString)?
       ;
```

U okviru pravila za definiciju regularnog tipa entiteta potrebno je specificirati konkretan tip entiteta (**entity**), kao i vrednosti za minimalan (**min**) i maksimalan (**max**) kardinalitet.

Gramatičko pravilo za opis instance **MinCardinality** koncepta koji opisuje vrednosti za minimalan kardinalitet glasi:

```
MinCardinality returns MinCardinality:
   'zero' | 'one';
```

Skup dozvoljenih vrednosti u okviru pravila za minimalan kardinalitet je nula (**zero**) ili jedan (**one**).

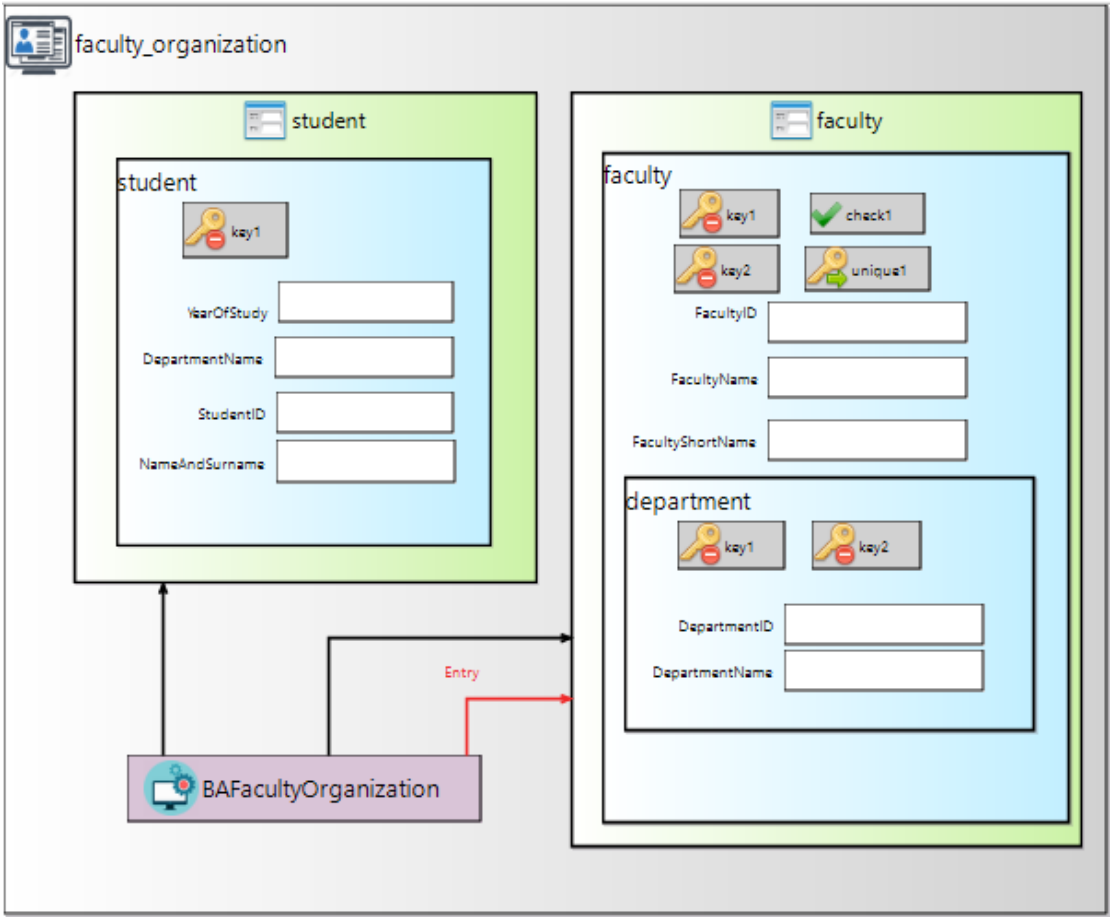
Gramatičko pravilo za opis instance **MaxCardinality** koncepta koji opisuje vrednosti za maksimalan kardinalitet glasi:

```
MaxCardinality returns MaxCardinality:
   'one' | 'many';
```

Skup dozvoljenih vrednosti u okviru pravila za maksimalan kardinalitet je jedan (**one**) ili više (**many**).

Jedan od glavnih ciljeva ove doktorske teze je razvoj namenskih jezika za modelovanje specifikacija jednog informacionog sistema. Specifikacija informacionog sistema predstavlja bitnu ulogu u procesu razvoja informacionih sistema. Meta-modeli predstavljeni u prethodnom poglavlju predstavljaju osnovu za izgradnju namenskih jezika.. Koncepti za modelovanje namenskog jezika definisani na niovu meta-modela predstavljaju njegovu apstraktnu sintaksu. U ovom poglavlju predstavljene su konkretne sintakse tri namenska jezika za modelovanje specifikacija informacionog sistema: FTDSL, UIDSL i EERDSL. Sva tri namenska jezika predstavljaju tekstualne namenske jezike.

6. Primena razvijenog pristupa u razvoju informacionih sistema na karakterističnim primerima



Slika 6.1.-Primer modela IS studentske službe

U prethodnom poglavlju ove disertacije prikazana je konkretna sintaksa namenskih jezika: FTDSL, UIDSL i EERDSL. U daljem tekstu ovog poglavlja opisan je praktičan primer upotrebe ovih namenskih jezika na osnovu primera koji je prikazan na slici 6.1. Cilj ovog poglavlja je da prikaže na koji način se mogu upotrebiti koncepti za modelovanje razvijenih namenskih jezika, i na koji način se mogu međusobno povezati. Takođe ovaj primer treba da pokaže koje su mogunoćsti upotrebe namesnkih jezika: FTDSL, UIDSL i EERDSL. Na osnovu dobijenih specifikacija može se izvesti zaključak, šta je od koncepata podržano u ovim namenskim jezicima.

Modelovanje specfikacije na osnovu primera 6.1 daje uvid u mogućnosti razvijenih namenskih jezika od strane autora ovog rada. Pored primera na slici 6.1 koji je modelovan putem namenskih jezika razvijenim u okviru ove disertacije, u odeljcima od 6.1 do 6.5 predstavljena je evaluacija razvijenih namenskih jezika. Prikazana evaluacija treba da pruži ocenu namenskih jezika od strane drugih korisnika koji nisu svesni načina njihove implementacije kao i dobrih i loših strana. Evaluaciju namenskih jezika obavila je grupa korisnika koja je podeljena u tri kategorije, sa stanovišta iskustva u modelovanju specifikacije IS. Cilj prikazane evaluacije u okviru ove disertacije je da pruži ocenu namenskih jezika sa

stanovišta sledećih karakteristika: funkcionalnost, upotrebljivost, pouzdanost, ekspresivnost i produktivnost.

Na slici 6.1 predstavljen je deo modela IS fakulteta, koji je nazvan FacultyIS. U okviru IS fakulteta FacultyIS, specificiran je aplikativni sistem faculty_organization. Jedan deo aplikativnog sistema faculty_organization treba da omogući evidenciju studenata i departmana unutar jednog fakulteta. Kako bi bila omogućena ova funkcionalnost, na nivou aplikativnog sistema faculty_organization specificirani su tipovi formi student i faculty. Na nivou tipa forme student koja omogućuje rad sa podacima o studentima, specificiran je tip komponente student koji omogućuje evidenciju jednog konkretnog studenta u sistemu. Tip komponente student sadrži skup atributa StudentId, NameAndSurname, DepartmentName, YearOfStudy, koji se modeluju šifra, ime i prezime, naziv departmana i godina studija za svakog studenta. Svaki student je jedinstveno opisan svojom šifrom što je modelovano pomoću ograničenja ključa.

Tip forme faculty omogućuje evidenciju samog fakulteta i skup departmana koji pripadaju tom fakultetu. Kako bi ovaj zahtev bio podržan definisan je tip komponente faculty koji omogućuje evidenciju fakulteta i tip komponente department koji obezbeđuje rad sa podacima o departmanima tog fakulteta. Tip komponente faculty je nadređen tipu komponente department. Tip komponente faculty sadrži skup atributa FacultyID, FacultyName i FacultyShortName kojim se opisuje šifra, naziv i skraćeni naziv jednog fakulteta. Na nivou tipa komponente faculty definisana su dva ograničenja ključa koji obezbeđuju jedinstveni opis svakog fakulteta po šifri ili skraćenom nazivu. Takođe je specificirano ograničenje jedinsvene vrednosti naziva fakulteta kao i ograničenje torke koje obezbeđuje da je vrednost šifre fakulteta uvek veća od nula. Na tipu komponente department specificiran je skup atributa DepartmentID i DepartmentName koji modeluju šifru i naziv jednog departmana. Ovaj tip komponente sadrži ograničenje ključa koje obezbeđuje jedinstvenu šifru za svaki departman. Na nivou aplikativnog sistema faculty_organization specificirana je poslovna aplikacija BAFacultyOrganization koja povezuje tipove formi student i faculty. U okviru poslovne aplikacije BAFacultyOrganization, tip forme faculty predstavlja ulazni tip forme.

U prilogu 1 ovog rada nalazi se fragment programa specificiran putem EERDSL namenskog jezika koji odgovara primeru koji je naveden na Slici 6.1. Najpre je kreirana instanca dijagrama pod nazivom *FacultyIS.* Zatim je kreiran skup tipova entiteta (*entities*): *student, faculty, department.*Za svaki tip entiteta definisan je skup atributa (*attributeSet*), skup ključeva (*keySet*) i primarni ključ (*primaryKey*). Nakon toga definisan je skup tipova poveznika (*relationships*). U okviru svakog tipa poveznika definisani su tipovi entiteta (*entitiesInRelationship*) koje taj tip poveznika povezuje, sa odgovarajućim vrednostima minimalnog i maksimalnog kardinaliteta.

U prilogu 2 ovog rada nalazi se fragment programa specificiran putem FTDSL i UIDSL namenskih jezika koji odgovara primeru koji je naveden na Slici 6.1. Najpre je kreirana instanca projekta pod nazivom *Faculty IS*. Nakon toga su specificirani atributi (*AttributeIncludedInDB*) sa njihovim vrednostima. Pre specifikacije aplikacije IS, potrebno je definisati jedan ili više tipova aplikacija na nivou projekta. U primeru prikazanom na slici 6.1, projekat *Faculty IS* se sastoji od dva aplikativna sistema (*ApplcationSystem*). Dok prvi predstavlja specifikaciju sistema aplikacija organizacije fakulteta, drugi predstavlja aplikativni sistem za studentsku službu. Za svaku instancu aplikacije *ApplcationSystem* neophodno je definisati naziv (*AppSistemName*), opis (*AppSistemDescription*) i tip

(*AppSistemType*). U ovoj fazi specifikacije, u primeru na slici 7, definišemo skup tipova formi za svaki aplikativni sistem. Za svaki tip formi naveden je naziv (*FormTypeName*), naslov (*FormTypeTitle*) i upotreba tipa forme. Svaki tip forme ima vrednosti obeležja za učestalost poziva (*FormTypeFrequency*) i vrieme odziva forme (FormTypeResponseTime). Svaki tip forme uključuje listu specifikacija tipova komponenti (*NewComponentType*). Tip forme faculty sadrži dva tipa komponente, glavni tip komponente (*CompTypeParent*) *faculty* i njegovu podkomponentu *department.*

Za svaki tip komponente, u primeru prikazanom na slici 6.1, definisan je naziv (*CompTypeName*), naslov (*CompTypeTitle*) i skup karakteristika prikaza (*CompTipeCompDisplay*). Na nivou tipa komponentefaculty funkcionalnost pretrage (*SearchFuncionality*) je omogućena. Tip komponente *faculty* treba da bude pozicioniran u novom prozoru (CompDisplaiPosition) i podaci moraju biti prikazani u rasporedu polja podataka (*CompDisplayDataLayout*). Za svaki tip komponente, takođe definišemo atribute tipa komponente tipa (*CompTypeAttribute*). Definicija atributa tipa komponente zahteva naziv (*CompTypeAttribName*) i naslov (*CompTypeAttribTitle*.

Nakon specifikacije skupa atributa tipa komponente, definisana je lista ograničenja tipa komponente. Takođe je moguce specificirati skup ključeva, ograničenja jedinstvenih vrednosti i ograničenja provere. U primeru prikazanom na slici 6.1, samo su ograničenja ključa za definisana za tip komponente student.

6.1 Evaluacija namenskog jezika EERDSL

Tokom i nakon razvoja EERDSL namenskog jezika, testiran je jezik na primeru studentske službe. Glavni nedostatak ovakvog načina evaluacije jezika je taj što izveden od strane autora jezika. Autor jezika je upoznat sa detaljima implementacije jezika i svestan je koje su prednosti i nedostaci. Stoga je neophodno da procenu jezika izvrše drugi korisnici.

Oblast ovog eksperimenta je procena karakteristika kvaliteta namenskog jezika EERDSL analizom percepcije korisnika. Namenske jezike možemo oceniti sa stanovišta sledećih karakteristika: (i) funkcionalnost - stepen do kojeg je zadovoljena implementacija rešenja za potrebe domena aplikacije; (ii) upotrebljivost – karakteristika koja pokazuje na koji način se jezik može koristiti za postizanje određenih ciljeva; (iii) pouzdanost - svojstvo jezika koje pomaže u proizvodnji pouzdanih programa (sposobnost provere modela); (iv) ekspresivnost – karakteristika koja pokazuje način na koji je se mogu određeni pojmovi iskazati putem jezika; i (v) produktivnost - karakteristika koja se odnosi na količinu resursa koje korisnik troši da bi postigao određene ciljeve. Detaljan opis eksperimenta predstavljen je u Dimitrieski et al. 2015.

6.2 Učesnici eksperimenta

Učesnici eksperimenta su izabrani na osnovu nekoliko kriterijuma. Prvi kriterijum je bio da učesnik ne sme biti upoznat sa načinom implementacije jezika. Korisnik koji je upoznat sa detaljima implementacije ima tendenciju da izbegne osobine koje ne rade pravilno. Stoga je gore navedeni kriterijum uspostavljen tako da se omogući nepristrasna procena. Takođe, odabrani su učesnici koji se uklapaju u jednu od sledeće tri kategorije: (i) stručnjaci u oblasti interakcija između čoveka i računara (HCI), koji mogu pružiti procene o vizuelnim aspektima

EERDSL-a, (ii) stručnjaci u obasti dizajna IS, koji mogu pružiti procenu podobnosti EERDSL-a za specifikaciju modela baze podataka, (iii) studenti, koji mogu dati informacije neophodne za primenu jezika u procesu obrazovanja. Od 16 učesnika koji su ocenili pristup, 2 su HCI ekperti, 3 su bili stručnjaci u dizajnu IS, a 11 su studenti. Od 11 studenata, 6 su studenti master studija na predmetima na kojima se izučavaju baze podataka na Fakultetu tehničkih nauka Univerziteta u Novom Sadu, dok je pet njih student na doktorskim studijama sa nekim prethodnim iskustvom u projektovanju šeme baze podataka.

6.3 Priprema i izvođenje eksperimenta

Svi učesnici su procenili jezik u istom, kontrolisanom okruženju. Procena je obavljena na jednom računaru sa svakim učesnikom zasebno. Najpre je korisniku održana kratka uvodna prezentacija sa informacijama o eksperimentu. Takođe su prezentovani zadaci koje su učesnici trebali da urade, kao i detalji o konceptima jezika. U toku prezentacije primenili smo primer koji sadrži dva tipa entiteta i jedan tip poveznika između njih. Za učesnike koji nisu imali ranije iskustvo u radu sa EER modelovanjem objašnjena je sintaksa i semantika svih EER koncepata. Objašnjenje EER koncepta i prezentacija je data svim učesnicima u vidu pisanog dokumenta. Stručnjaci u oblasti interakcija čovek računar su odlučili da preskoče ovu prezentaciju kako bi mogli testirati intuitivnost jezika kada se koristi bez prethodnog objašnjavanja. Nakon prezentacije, od svih učesnika je zatraženo da implementiraju isti, već pripremljeni primjer (IS Studentske službe) koristeći tekstualnu notaciju EERDSL-a. Različite implementacije primera su bile moguće na osnovu iskustva učesnika u radu sa EER modelima podataka. Prosečna implementacija obuhvata 12 tipova entiteta, 10 tipova poveznika i 30 obeležja. Neophodno je koristiti sve EER koncepte kako bi primer bio specificiran na korektan način. Funkcionalni zahtevi primera su definisani na jednoj A4 stranici koja je dostavljena učesnicima na početku evaluacije. Iako su zahtevi bili su striktno definisani, nismo uvodili nikakva vremenska ograničenja za koje učesnici moraju završiti zadatak. Autor jezika je bio prisustan citavo vreme dok su učesnici izvodili eksperiment i davao je odgovore na sva pitanja. Međutim, autor se nije mešao ni na jedan drugi način u toku eksperimenta osim što je davao odgovore učesnicima na postavljena pitanja.

Po završetku praktičnog rada učesnici su bili zamoljeni da popune upitnik. Pitanja su bila podeljena u tri sekcije: (i) pitanja vezana za prethodna uskustva učesnika, (ii) pitanja u pogledu karakteristika namenskog jezika, i (iii) sekcija za komentare u slobodnoj formi. U prvom odjeljku upitnika, učesnici su bili zamoljeni da pruže informacije o njihovom prethodnom iskustvu modelovanju IS korsiteći manje ili više slične CASE alate. Za svako pitanje,učesnici su mogli da obeleže jedan odgovor na uobičajnoj Likertovoj skali sa pet nivoa. Na ovakvoj skali odgovor je moguć u rasponu od 1. Neiskusan do 5. Iskusan. Drugi deo upitnika obezbeđuje evaluaciju pet karakteristika kvaliteta jezika: funkcionalnost, korisnost, pouzdanost, ekspresivnost, i produktivnost. Za svaku karakteristiku obezbeđeno je nekoliko iskaza.. Korisnici su mogli definišu koji nivo jezik ispunjava za svaku karakteristiku označvanjem jednog od pet odgovora u rasponu od 1. Potpuno se ne slažem do 5. Potpuno se slažem. Samo pitanja vezana za karakteristiku produktivnosti nisu pratila pomenutu Likertovu skalu. Učesnici su ocijenili vreme potrebno za kreiranje EER modela podataka koristei namenski jezik odgovrima od 1. Dugo do 5. Kratko. Na kraju, upitnik sadrži odeljak za slobodne komentare gde su učesnici mogli da iskažu komentare i primedbe koji nisu pokriveni pitanjima. Ovaj odeljak je bio velika pomoć za tumačenje odgovora učesnika. Upitnik nije bio anoniman jer je autor ostavio mogućnost da kontaktira učesnike u slučaju da su potrebna dodatna objašnjenja odgovora. Svaki učesnik je trebao da unese svoje ime,

prezime i e-mail adresu. Autor jezika je pružio garancije da će svi podaci biti anonimni i da ih neće dalje distribuirati. Svi učesnici su bili obavešteni i složili su se sa uslovima eksperimenta.

6.4 Rezultati i analiza eksperimenta

EERDSL je ocenjen kao funkcionalno pogodan jezik. Učesnici su prepoznali da EERDSL pokriva većinu koncepata iz domena EER modela podatak i pogodan je za specifikaciju modela podataka. Učesnici nisu napisali bilo kakve dodatne komentare u odeljku slobodnih komentara. Karakteristika kvaliteta upotrebljivost EERDSL-a je ocenjena kao pozitivna od strane velikog broja učesnika. EERDSL je okarakterisan kao pouzdan od strane velikog broja učesnika. Svi učesnici su se složili da EERDSL štiti korisnike od stvaranja sintaksnih grešaka i prikazuje razumljive poruke ako je korisnik napravio semantičku grešku. Poruke o grešci se opisuju kao korisne prilikom prikaza uzroka greške.

Ekspresivnost EERDSL-a je ocenjena pozitivno, uz dodatne komentare stručnjaka u projektovanju IS. Primeba se odnosi na strategiju rešavanja problema prilikom spikifikacije modela. Trenutno, pomoću jezika, celokupni IS treba da se specificira sa jednim EER modelom baze podataka. Stručnjaci u projektovanju IS smatraju da ovakav način rada može predstavljati problem za velike i složene sisteme. Njihov predlog je da se, razdvajanjem modela baze podataka na manje logičke celine, omogući lakše rešavanje složenijih problema. U takvoj situaciji bilo bi potrebno podržati algortime za integraciju modela podataka. Na kraju, učesnici smatraju da je vreme potrebno vreme za specifikaciju modela baze podataka pomoću EERDSL u opsegu od relativno kratkog do srednjeg vremena. Kako odgovor na ovo pitanje zahteva prethodno iskustvo u radu sa sličnim u alatima i tehnikama projektovanja IS, najvažniji odgovori na ovo pitanje su dobijeni od stručnjaka u oblasti projektovanja IS. Sva tri učesnika su odabrala odgovor "kratko" ili "relativno kratko" vreme potrebno za specifikaciju. Odgovor "dugo" vreme potrebno za specifikaciju odabralo je nekoliko studenata.

Nakon sprovedenog ispitivanja učesnika, izračunat je Spirmanov koeficijent korelacije između pitanja u upitniku. Analiza je izvršena pomoću R alata. Jedini interesantan faktor korelacije koji smo pronašli vezan je za pitanje vremena potrebnog za specifikaciju modela baze podataka putem namenskog jezika što obuhvata karakteristiku vezanu za proaktivnost. U tabeli 3, predstavljeni su korelacioni koeficijenti sa odgovarajućim p-vrednostima za svako pitanje koje je vezano vreme potrebno za speciifikaciju modela baze podataka. Što je veća vrednost koeficijenta korelacije. to znači da postoji jača korelacija između dva povezana pitanja. Samo za korelacije sa p-vrednostima manjim ili jednakim 0,01, možemo reći da su statistički značajne. Rezultati evaluacije pokazuju da učesnici koji vide sintaksu jezika razumljivom, smatraju da je vreme potrebno za specifikaciju EER modela manje. U analizi rezultata evaluacije nisu predstavljeni drugi proračuni korelacionih koeficijenata zbog sledećih razloga. Prepoznati su visoki koeficijenti za puno trivijalnih korelacija. Na primer, za većinu kategorija, pitanja unutar kategorije su u korelaciji sa visokim koeficijentom i p-vrednostima manjim od 0.01. Ovo je očekivano jer ovalva pitanja pokrivaju istu karakteristiku kvaliteta i odgovori treba da budu u korelaciji ako korisnici iskreno odgovore na ova pitanja. Ostala pitanja su korelirana sa velikim koeficijentom efikasnosti, ali je p-vrednost veća od 0,1. Ovo čini ove korelacije statistički beznačajnim i stoga nisu ovde predstavljeni.

Na osnovu rezultata evaluacije, možemo zaključiti da EERDSL zadovoljava sledeće karakteristike: funkcionalnu stabilnost, upotrebljivost, pouzdanost, izražajnost i produktivnost. Međutim, može se zaključiti da se jezik može dodatno poboljšati prema komentarima učesnika.

6.5 Pregled ostalih karakteristika kvaliteta

Kako bi bile procenjene druge karakteristike kvaliteta kao što su održavanje, proširivost, ponovna upotrebljivost i mogućnost integracije, učesnici moraju biti upoznati sa detaljima o implementaciji EERDSL-a. Jedan od glavnih kriterijuma za izbor učesnika je bio da učesnici nisu upoznati sa implementacijom EERDSL-a. Zbog ovog razloga ove osobine može oceniti samo autor pristupa.

Održavanje predstavlja mogućnost modifikacije i povezivanja DSL komponenti. Dodavanje nove funkcionalnosti u EERDSL ne degradira postojeću DSL funkcionalnost. Međutim, modifikovanje postojećih koncepata u meta-modelu EER-a može zahtevati neke naknadne izmene u sintaksi jezika. Ovo je posledica tesne povezanosti između EERDSL komponenti. Ovo svakako može uticati na sporiju modifikaciju EERDSL-a. Proširivost se definiše kao mehanizam za dodavanje nove funkcionalnosti u DSL-u od strane korisnika. EERDSL nema ovakvu mogučnost proširenja funkcionalnosti.. Sve izmene moraju biti učinjene od strane autora. Slično tome, nisu krajnjem korisniku nije pružen mehanizam za kreiranje novih jezika za modelovanje na osnovu EERDSL-a. Integracija EERDSL-a je moguća korišćenjem mode u model transformacija čime je omogućeno da se jezik lako integriše sa drugim jezicima u razvoju IS.

Međutim, postoje i neke opasnosti u ovakvom eksperimentu koje je potrebno spomenuti. Složenost zadatka koji su učesnici realizovali može da predstavlja pretnju tokom eksperimenta. Ishod eksperimenta bi mogao biti drugačiji ukoliko bi od učesnika bilo traženo da urade drugačiji primer. Primer je dizajniran sa ciljem da pokrije osnovne koncepte EER-a i dopusti učesnicima koji su iskusni ili neiskusni da ocene EERDSL razumnom vremenu. Primer koji bi bio manje ili vise složen može dovesti do različitih rezultata evaluacije. Iako je obezbeđen pisani dokument koji sadrži prezentaciju EER koncepata, učesnici su imali dozvolu da postavljaju pitanja u neograničenom broju. Na ovaj način neki učesnici su dobili detaljnija objašnjenja od drugih jer su postavljali više pitanja. Još jedna pretnja leži u statističkom delu evaluacije. Tokom eksperimenta nije bilo moguće skupiti dovoljno ljudi po grupama učesnika kako bi se obezbedio isti broj ljudi sa istim prosečnim nivoom stručnosti. Takođe, izabrano je samo 16 učesnika. Nije poznato u kojoj meri rezultati ove evaluacije mogu da se prome ako bi bilo više učesnika koji su ocjenjivali jezik. Takođe nije poznati u kojoj mjeri bi rezultati mogli biti promenjeni ukoliko bi bila različita struktura učesnika. Stoga, ovo može biti još jedna pretnja ovom eksperimentu.

Pristup modelovanju specifikacije informacionog sistema, koji je prezentovan u ovoj disertaciji, podrazumeva primenu razvijenih namenskih jezika. U ovom poglavlju prikazan je praktičan primer upotrebe namenskih jezika:FTDSL, UIDSL i EERDSL na pimeru dela modela informacionog sistema jednog fakulteta. Na nivou primera je pokazano da FTDSL i UIDSL poseduju više koncepata za modelovanje od EERDSL namenskog jezika. FTDSL i UIDSL pored koncepata za modelovanje šeme baze podataka, poseduju koncepte za opis ekranskih formi, tj korisničkog interfejsa.

U ovom poglavlju takođe je predstavljena evaluacija EERDSL namenskog jezika od strane korisnika. Na osnovu analize primera koji je predstavljen na početku ovog poglavlja mogu se izvesti zaključci koji su pre svega subjektivni jer je autor jezika ista osoba koja ocenjuje jezik. Putem evaluacije od strane drugih dobijeni su zaključi koji su u višoj meri objektivizirani. Rezultate ovakvog načina evaluacije možemo smatrati relevantnim jer je analiza obavljena sa tri različite grupe korisnika: studenti, stručnajci u oblasti projektovanja informacionih sistema i stručnjaci u oblasti korisnničkog interfejsa. Korisnici su zajednički ocenili jezik kao funkcionalan i ekspresivan jer pokriva sve EER koncepte za modelovanje na upotrebljiv način. Zamerka koju su izneli stručnjaci s najviše iskustva u oblasti projektovanja je način specificikacije velikih i složenih šema baza podataka, koji se mora obaviti kroz jedan model.

Zbog nedostatka vremena evaluacija je urađena samo za EERDSL namenski jezik. Svakako bi u narednim koracima istraživanja bilo neophodno da korisnici ocene FTDSL i UIDSL. Takođe bi bilo nephodno dobiti njihovu uporednu ocenu za sva tri namenska jezika: EERDSL, FTDSL i UIDSL.
.

7. Zaključak

U ovom, završnom poglavlju doktorske disertacije biće predstavljeni glavni doprinosi sprovedenog istraživanja, pravci budućih istraživanja kao i prednosti i nedostaci razvijenog rešenja.

Na početku ovog istraživanja, autor je postavio polazne hipoteze na osnovu kojih su definisani osnovni ciljevi doktorske disertacije. Glavni cilj bio je da se pokaže da je moguće realizovati pristup za specifikaciju informacionih sistema zasnovan na upotrebi namenskih jezika za modelovanje. Kako bi ovakva hipoteza bila opravdana, formulisan je originalni metodološki pristup i kreirano softversko okruženje, namenjeni unapređenju procesa za specifikaciju informacionih sistema.

U cilju opravdavanja hipoteza H1, H2 i H3, koje su formulisane u Uvodu ovog rada, razvijeni su neophodni meta-modeli kao i tekstualne konkretne sintakse zasnovane na tim meta-modelima, kojima je realizovan proces konceptualizacije. Meta-modeli su implementirani pomoću Ecore meta-metamodela u EMF okruženju. Konkretna sintaksa je implementirana pomoću Xtext razvojnog okruženja koje je zasnovano na Eclipse okruženju. Ecore, kao implementacija MOF meta-metamodela koji je propisan OMG standardom, i XText, pripadaju MDA tehnologijama.

Tokom istraživanja koja su predstavljena 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 koja je upotrebljena u ovom radu treba obogatiti novim metodama i algoritmima za modelovanje specifikacija informacionog sistema putem namenskih jezika. Ovakav pravac daljeg istraživanja bi podrazumevao definisanje novih meta-modela, kojima bi se specificirale apstraktne sintakse jezika za modelovanje šema baza podataka. Kao predmeti specifikacije mogle bi se naći šeme baza podataka zasnovane na objektnom, objektno-relacionom ili XML modelu podataka. Takođe je neophodno podržati koncepte za modelvoanje šema baza podataka koje su zasnovane na NoSQL pristupu. Ovo bi pre svega značilo uvođenje dodatnih koncepata za podšku specifikacija JSON dokumenata. Naknadno bi trebalo razmotriti kako u okviru meta-modela i odgovarajućih namenskih jezika može biti iskazana integracija šema baza podataka koje su zasnovane na hibridnom, SQL i NoSQL pristupu.

Istraživačke aktivnosti u širem smislu, tj. aktivnosti na daljem razvoju predstavljenog pristupa i okruženja IIS*Studio mogu se podeliti u dve grupe:

- proširivanje meta-modela IIS*Case-a i
- dodavanje novih alata u IIS*Case-a, koji su namenjeni modelovanju IS putem vizuelnih i tekstuelnih notacija. Ovakvi alati treba da obezbede i formalnu proveru semantičke validnosti modela u toku modelovanja.

Proširivanje metamodela alata IIS*Case može značiti uvođenje novih koncepata za modelovanje kako bi model, koji je specificiran putem ovog alata, sadržao što potpuniju specifikaciju informacionog sistema. Jedan od daljih pravaca istraživanja treba da bude posvećen uvođenju koncepata i alata koji su namenjeni modelovanju sistemske arhitekture. Na ovaj način bila bi definisana osnova za automatizovanje koraka isporuke i instaliranja generisanih aplikacija. Takođe istraživanja mogu biti usmerena i na proširivanje meta-modela konceptima koji bi omogućili modelovanje jednog dela informacionog sistema koji je zadužen za bezbednost programa i podataka. Ovakav pravac istraživanja predstavljao bi

važan aspekt za razvoj svakog softverskog proizvoda, bez koga je nemoguće stići do komercijalnog rešenja projektovanog dela IS-a. Pored toga, dalja istraživanja mogu ići u pravcu formulisanja i realizacije algoritama koi su namenjeni transformaciji modela kreiranog putem IIS*Case-a u model zadat putem UML-a. Na ovaj način bila bi podržana 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 su razvijeni jezici *FTDSL i UIDSL* putem kojeg je moguće zadati specifikacije tipova formi. Dalja istraživanja mogu biti posvećena razvoju transformacija pomoću kojih bi mogle da se generišu ekvivalente specifikacije u *FTDSL-u i UIDSL*-u na osnovu sadržaja repozitorijuma. Ovakav skup transformacija podržao bi i obratan smer, tj. dopunjavanje i modifikacija sadržaja repozitorijuma na osnovu specifikacija u *FTDSL-u i UIDSL-u*. Na ovaj način, korisnicima bi bilo omogućeno da paralelno koriste i tekstuelnu i vizuelnu notaciju za definisanje modela specifikacija, a stvara se i mogućnost vršenja uporedne analize ova dva pristupa zadavanju specifikacija. Ovaj prvac istraživanja predstavlja osnovu za razvoj parsera koji bi vršio proveru semantičke ispravnosti modela, kao što je provera da li postoje kolizije ograničenja torki. Dalja istraživanja mogla bi takođe da idu u pravcu evaluacije i ocene namenskih jezika na formalan način. Ovakav pravac istraživanja podrazumevao bi specifikaciju koncepata za modelovanje karakteristika koje želimo da ocenimo na nivou jednog namenskog jezika. Takođe bi koncepti na nivou meta-modela trebalo da obezbede modelovanje odgovarajućih metrika putem kojih bi bilo moguće iskazati karakteristika koncepata namenskog jezika na merljiv način.

Takođe, jedan od bitnih pravaca daljeg razvoja može biti i rad na razvoju laboratorijske verzije alata čitavog okruženja IIS*Studio u stanje koje bi omogućilo komercijalnu upotrebu alata. Kako sam postupak dovođenja jednog složenog softverskog okruženja, kakvo je i IIS*Studio, do stanja spremnog za komercijalnu upotrebu zahteva čitav niz novih ili modifikovanih postojećih zahteva kojima se mora pristupiti na posebno sistematičan način, bilo bi neophodno povezati kumulirana, teorijska i interdisciplinarna znanja s jedne strane, i praktično iskustvo s druge strane. Ovakav postupak svakako predstavlja i istraživački izazov, a ne samo rutinski inženjerski posao.

Novi pristup modelovanju specifikacija IS putem namenskih jezika, koji je predstavljhen u ovoj disertaciji, treba da bude korišćen u praksi kroz implementirano softversko rešenje koje ga podržava. Domen primene namenskog jezika koji omogućuje modelovanje specifikacija IS pomoću koncepta tipa forme ograničen je na postojeći alat za projektovanje IS, IIS*Studio. Tekstualni namenski jezik koji omogućuje modelovanje specifikacija informacionog sistema pomoću koncepata proširenog modela tipova entiteta i poveznika nije uslovljen upotrebom alata IIS*Studio.

Apstraktna sintaksa tekstualnog namenskog jezika za modelovanje specifkacije IS koja je definisana putem meta-modela predstavlja osnovu za alat koji bi mogao da vrši proveru modela IS. IIS*Studio PIM meta-model može da posluži u postupku verifikacije generisanih relacionih modela šema baza podataka. Trenutno, IIS*Studio pomaže projektantima u otkrivanju formalnih grešaka na nivou relacionog modela baze podataka. Novi pristup koji je rezultat ovog istraživanja mogao bi da omogući projektantima otkrivanje i razrešenje kolizija ograničenja na nivou PIM modela.

Predloženi pristup takođe može da ima primenu i u edukaciji studenata. Kako je EER pristup prisutan u gotovo svakoj knjizi o bazama podataka, tekstualni namenski jezik koji omogućuje modelovanje specifikacija informacionog sistema pomoću koncepata proširenog modela tipova entiteta i poveznika biće od pomoći u učenju EER koncepata. Ovaj namenski jezik takođe će biti od pomoći studentima u procesu usvajanja tehnika konceptualnog modelovanja baze podataka. Pored toga, razvijeni namenski jezici za modelovanje i alat koji ih podržava može da posluži kao ugledni primer studentima u kursevima koji se bave razvojem softvera pomoću MDSD pristupa.

Literatura

AADL: Architecture Analysis and Design Language. Retrieved November 6, 2016, from http://www.aadl.info/aadl/currentsite/.

ACME: ACME. Retrieved November 6, 2016, from http://www.cs.cmu.edu/~acme/.

AESOP: AESOP. Retrieved November 6, 2016, from https://www.cs.cmu.edu/~able/aesop/.

Aleksić, S., Luković, I., Mogin, P., & Govedarica, M. (2007). A generator of SQL schema specifi-cations. Computer Science and Information Systems, 4(2), 79-98.

Aleksić, S., Ristić, S., & Luković, I. (2011). An approach to generating server implementation of the inverse referential integrity constraints.In Proceedings of the 5th International Conference on Information Technology, (pp. 1-7). Amman: Al-Zaytoonah University of Jordan.

Aleksić, S., Ristić, S., Luković, I., & Čeliković, M. (2013). A design specification and a server implementation of the inverse referential integrity constraints. Computer Science and Information Systems, 10(1), 283-320.

Banović, J. (2010). An approach to generating executable software specifications of an information system. Unpublished doctoral dissertation, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia.

Bézivin, J. (2004). In search of a basic principle for model driven engineering. UPGRADE - The European Journal for the Informatics Professional, 5(2), 21-24.

Bicevskis, J., Cerina-Berzina, J., Karnitis, G., Lace, L., Medvedis, I., & Nesterovs, S. (2011). Practitioners view on domain specific business process modeling. In Proceeding of the 2011 conference on Databases and Information Systems VI: Selected Papers from the Ninth International Baltic Conference (pp. 169-182). Amsterdam: IOS Press.

Brambilla, M., Cabot, J., Wimmer, M. (2012). Model-Driven Software Engineering in Practice (Synthesis Lectures on Software Engineering). Morgan & Claypool Publishers.

Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. Information and Software Technology, 38(4), 275-280.

Bureš, T., Malohlava, M., & Hnětynka, P. (2008). Using DSL for automatic generation of software connectors. In Proceedings of the Seventh International Conference on Composition-Based Software Systems (pp. 138-147). Washington, DC: IEEE Computer Society.

Chen, P. P. S. (1976). The entity-relationship model – Toward a unified view of data. ACM Transactions on Database Systems, 1(1), 9-36.

Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377-387.

Čeliković, M., Luković, I., Aleksić, S., & Ivančević, V. (2011). 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.

Čeliković, M., Dimitrieski, V., Aleksić, S., Ristić, S., & Luković, I. (2014). A DSL for EER Data Model Specification. In Proceedings of the 23rd International Conference On Information Systems Development (ISD 2014), September 2-4, 2014, University of Zagreb, Faculty of Organization and Informatics, Varazdin, pp. 290-297.

Čeliković, M., Luković, I., Aleksić, S., & Ivančević, V. (2012). 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.

Dejanović, I., Milosavljević, G., Tumbas, M., & Perišić B. (2010). A domain-specific language for defining static structure of database applications. Computer Science and Information Systems, 7(3), 409-440.

Deursen van, A., Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices, 35(6), 26-36.

Dimitrieski, V., Čeliković, M., Aleksić, S., Ristić, S., Alargt, A., & Luković, I. (2015). Concepts and evaluation of the extended entity-relationship approach to database design in a multi-paradigm information system modeling tool. Computer Languages, Systems & Structures, 44, 299-318.

Dimitrieski, V., Čeliković, M., Aleksić, S., Ristić, S., & Luković, I. (2014). Extended entity-relationship approach in a multi-paradigm information system modeling tool. In Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on (pp. 1611-1620). IEEE.

Đukić, V., Luković, I., & Popović, A. (2011). Domain-specific modeling in document engineering. In Proceedings of the Federated Conference on Computer Science and Information Systems (pp. 825-832). Los Alamitos, CA: IEEE Computer Society Press.

EMF: Eclipse Modeling Framework. Retrieved October 6, 2016, from http://www.eclipse.org/modeling/emf/

Filipović, M., Vaderna, R., Ivković, Ž., Kaplar, S., Vuković, Ž., Dejanović, I., ... & Ivanović, D. (2017). Application of Kroki Mockup Tool to Implementation of Executable CERIF Specification. Procedia Computer Science, 106, 245-252.

Fowler, M., Parsons, R. (2010). Domain Specific Languages. Addison-Wesley Signature

France, R., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In Future of Software Engineering (pp. 37-54). Washington, DC: IEEE Computer Society.

Frantz, R. Z., Corchuelo, R., & Gonzáles J. (2008). Advances in a DSL for application integration. Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos, 2(2), 54-66.
Freudenthal, M. (2010a). Domain-specific languages in a customs information system. IEEE Software, 27(2), 65-71.

Freudenthal, M. (2010b). Using DSLs for developing enterprise systems. In Proceedings of the 10th Workshop on Language Descriptions, Tools and Applications, article 11. New York: ACM Press.

Fuentes-Fernández, L., & Vallecillo-Moreno, A. (2004). An introduction to UML profiles. UPGRADE - The European Journal for the Informatics Professional, 5(2), 6-13.

Giaglis, G. M. (2001). A taxonomy of business process modeling and information systems modeling techniques. The International Journal of Flexible Manufacturing Systems, 13(2), 209-228.

GME: Generic Modeling Environment. Retrieved November 6, 2016, from http://www.isis.vanderbilt.edu/Projects/gme/

Grundy, J., Hosking, J. G., Amor, R. W., Mugridge, W. B., & Li, Y. (2004). Domain-specific visual languages for specifying and generating data mapping systems. Journal of Visual Languages Computing, 15(3-4), 243-263.

Hirschheim, R., & Klein, H. K. (1989). Four paradigms of information systems development. Communications of the ACM, 32(10), 1199-1216.

Hudak, P. (1998). Domain Specific Languages. Handbook of Programming Languages, Vol. III: Little Languages and Tools, Peter H. Salas, ed. MacMillan, Indianapolis, 39–60.

Irazábal, J., Pons, C., & Neil, C. (2010). Model transformation as a mechanism for the implementation of domain specific transformation languages. SADIO Electronic Journal of Informatics and Operations Research, 9(1), 49-66.

Jouault, F., Bézivin, J., & Kurtev, I. (2006). TCS: A DSL for the specification of textual concrete syntaxes in model engineering. In Proceedings of the 5th International Conference on Generative Programming and Component Engineering (pp. 249-254). New York: ACM Press.

Kelly, S. (2005). Improving developer productivity with domain-specific modeling languages. Developer.*. Retrieved September 23, 2011, from http://www.developerdotstar.com/mag/articles/ domain_modeling_language.html

Kleppe, A., Warmer, J., & Bast, W. (2003). MDA explained: The model driven architecture - practice and promise. New York: Addison-Wesley.

Kosar, T., Oliveira, N., Mernik, M., Varanda Pereira, M. J., Črepinšek, M., da Cruz, D., & Henriques, P. R. (2010). Comparing general-purpose and domain-specific languages: An empirical study. Computer Science and Information Systems, 7(2), 247-264.

Kurtev, I., Bézivin, J., Jouault, F., & Valduriez, P. (2006). Model-based DSL frameworks. In OOPSLA '06 Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (pp. 602-615). New York: ACM Press.

Luković, I. (2009). From the synthesis algorithm to the model driven transformations in database design. In Proceedings of 10th International Scientific Conference on Informatics (pp. 9-18). Košice, Slovakia: Slovak Society for Applied Cybernetics and Informatics and Technical University of Košice - Faculty of Electrical Engineering and Informatics.

Luković, I., Mogin, P., Pavićević, J., & Ristić, S. (2007). An approach to developing complex database schemas using form types. Software: Practice and Experience, 37(15), 1621-1656.

Luković, I., Ristić, S., Aleksić, S. & Popović, A. (2008). 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.

Luković, I., Ristić, S., & Mogin, P. (2003). 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.

Luković, I., Ristić, S., Mogin, P., & Pavićević, J. (2006). Database schema integration process – A methodology and aspects of its applying. Novi Sad Journal of Mathematics, 36(1), 115-150.

Luković, I., Popović, A., Mostić, J., & Ristić, S. (2010). A tool for modeling form type check constraints and complex functionalities of business applications. Computer Science and Information Systems, 7(2), 359–385.

Luković, I., Pereira, V. J. M., Oliveira, N., & Henriques, R. P. (2011). A DSL for PIM specifications: Design and attribute grammar based implementation. Computer Science and Information Systems, 8(2), 379-403.

Luković, I., Ivancević, V., Čeliković, M., & Aleksić, S. (2013). DSLs in action with model based approaches to information system development. In Formal and Practical Aspects of Domain-specific Languages: Recent Developments (pp. 502-532). IGI Global.

Menezes, A. L., Cirilo, C. E., Moraes, J. L. C. D., Souza, W. L. D., & Prado, A. F. D. (2010). Using archetypes and domain specific languages on development of ubiquitous applications to pervasive healthcare. In Proceedings of the 23rd IEEE International Symposium on Computer-Based Medical Systems (pp. 395-400). Washington, DC: IEEE Computer Society Press.

Mernik, M., Heering, J., & Sloane, M. A. (2005). When and how to develop domain-specific languages. ACM Computing Surveys, 37(4), 316-344.

MetaCaseMetaEdit+ . Retrieved November 6, 2016, from http://www.metacase.com/

MOF: Meta-Object Facilty. Retrieved November 6, 2016, from http://www.omg.org/mof/

Milanović, N., Cartsburg, M., Kutsche, R., Widiker, J., & Kschonsak, F. (2009). Model-based interoperability of heterogeneous information systems: An industrial case study. In R. F. Paige, A. Hartman, & A. Rensink (Eds.), Model driven architecture - Foundations and Applications, Lecture Notes in Computer Science, Volume 5562/2009 (pp. 325-336). Berlin Heidelberg, Germany: Springer-Verlag.

Milosavljević, G., Filipović, M., Marsenić, V., Pejaković, D., & Dejanović, I. (2013). Kroki: A mockup-based tool for participatory development of business applications. In Intelligent Software Methodologies, Tools and Techniques (SoMeT), 2013 IEEE 12th International Conference on (pp. 235-242).

Nunes, D. A., & Schwabe, D. (2006). Rapid prototyping of web applications combining domain specific languages and model driven design. In Proceedings of the 6th International Conference on Web engineering (pp. 153-160). New York: ACM Press.

Obrenovic, N., Popovic, A., Aleksic, S., & Lukovic, I. (2012). Transformations of check constraint PIM specifications. Computing and Informatics, 31(5), 1045-1079.

Pastor, O., Gómez, J., Insfrán, E., & Pelechano V. (2001). The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. Information Systems, 26(7), 507-534.

Perišić, B., Milosavljević, G., Dejanović, I., & Milosavljević, B. (2011). UML profile for specifying user interfaces of business applications. Computer Science and Information Systems, 8(2), 405-426.

Reiter, T., Kapsammer, E., Retschitzegger, W., Schwinger, W., & Stumptner, M. (2006). A generator framework for domain-specific model transformation languages. In Proceedings of the Eighth International Conference on Enterprise Information Systems Databases and Information Systems Integration (pp. 27-35). Paphos, Cyprus: ICEIS Press.

Ristić, S., Aleksić, S., Luković, I., Banović, J. (2011). Form-Driven Application Generating: A Case Study. In Proceedings of 11th International Conference on Informatics (pp. 115-120). Košice, Slovakia: Slovak Society for Applied Cybernetics and Informatics and Technical University of Košice - Faculty of Electrical Engineering and Informatics.

Ristić, S., Luković, I., Pavićević, J., Mogin, P. (2007). Resolving Database Constraint Collisions Using IIS*Case Tool. Journal of Information and Organizational Sciences, 31(1), 187-206.

Ristić, S., Mogin, P., & Luković, I. (2003). Specifying database updates using a subschema. In Proceedings of VII IEEE International Conference on Intelligent Engineering Systems (pp. 203-212). Assiut-Luxor, Egypt: IEEE, Assiut University, Assiut, Egypt, and Budapest Polytechnic, Budapest, Hungary.

Ristić, S., Aleksić, S., Čeliković, M., & Luković, I. (2014). Generic and standard database constraint meta-models. Computer Science and Information Systems, 11(2), 679-696.

Sivonen, S. (2008). Domain-specific modelling language and code generator for developing repository-based Eclipse plug-ins. Espoo, Finland: VTT Publications.

Schmidt, D.C. (2006). Model-driven engineering. IEEE Computer, 39(2), 25-31.

Schuette, R., & Rotthowe, T. (1998). The Guidelines of Modeling - An approach to enhance the quality in information models. In ER '98 Proceedings of the 17th International Conference on Conceptual Modeling (pp. 240-254). London: Springer-Verlag.

Shanks, G. (1997). Conceptual data modelling: an empirical study of expert and novice data modellers. Australasian Journal of Information Systems, 4(2), 63-73.

Thomas, D. (2004). MDA: Revenge of the modelers or UML utopia? IEEE Software, 21(3), 22-24.

Thomas, D., & Barry, B.M. (2003). Model driven development: The case for domain oriented programming. In Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (pp. 2-7). New York: ACM Press.

Vara, J. M., Vela, B., Bollati, V., & Marcos, E. (2009). Supporting model-driven development of object-relational database schemas: A case study. In R. Paige (Ed.), International Conference on Model Transformation, Lecture Notes in Computer Science, Volume 5563/2009 (pp. 181-196). Berlin Heidelberg, Germany: Springer-Verlag.

vom Brocke, J., & Buddendick, C. (2006). Reusable conceptual models. requirements based on the design science research paradigm. Paper presented at the First International Conference on Design Science Research in Information Systems and Technology, Claremont, CA.
Wand, Y., & Weber, R. (2002). Research commentary: Information systems and conceptual modeling – a research agenda. Information Systems Research, 13(4), 363-376.

Wendt, T., Häber, A., Brigl, B., & Winter, A. (2004). Modeling hospital information systems (Part 2): Using the 3LGM2 tool for modeling patient record management. Methods of Information in Medicine, 43(3), 256-267.

Winter, A., Brigl, B., & Wendt, T. (2003). Modeling hospital information systems. Part 1: The revised three-layer graph-based meta model 3LGM2. Methods of Information in Medicine, 42(5), 544-551.

Wright: Wright. Retrieved November 6, 2016, from http://www.cs.cmu.edu/~able/wright/.

Živanov, Ž., Rakić, P., & Hajduković, M. (2008). Using code generation approach in developing kiosk applications. Computer Science and Information Systems, 5(1), 41-59.

Prilog 1 – Primer specifikacije informacionog sistema putem EERDSL namenskog jezika

```
FacultyIS {
	entities {
		student {
			attributeSet {
				StudentID,
				NameAndSurmname,
				YearOfStudy
			}
			keySet {
				PK_student ( StudentID )
			}
			primaryKey PK_student
		},
		faculty {
			attributeSet {
				FacultyID,
				FacultyName,
				FacultyShortName
			}
			keySet {
				PK_fac ( FacultyID )
			}
			primaryKey PK_fac
		},
		department {
			attributeSet {
				DepartmentID,
				DepartmentName
			}
			keySet {
				PK_dep ( DepartmentID )
			}
			primaryKey PK_dep
		}
	}
	relationships {
		has {
			entitiesInRelationship {
				faculty ( one,
				many ),
				department ( one,
				one )
			}
		},
		studies {
			entitiesInRelationship {
```

```
                    student ( one,
                    one ),
                    department ( zero,
                    many )
                }
        }
}
```

Prilog 2 – Primer specifikacije informacionog sistema putem FTDSL i UIDSL namenskih jezika

```
Repository {
	Name 'IISCase Repository'

	Projects {
		Project {
			Name projekat

			//application types
			ApplicationTypes{
				ApplicationType {
					Name system
					Description 'application system'
				},
				ApplicationType {
					Name subsystem
					Description 'application subsystem'
				}
			}

			//fundamental concepts
			FundamentalConcepts {
				//primitive domains
				PrimitiveDomain {
					Name int
					Description 'integer value'
				},
				PrimitiveDomain {
					Name varchar2
					Description 'array of characters'
					DefaultLength 20
				},

				//user defined domains
				UserDefinedDomainFromPrimitiveDomain {
					Name number
					Description 'number value'
					InheritsPrimitiveDomain int
				},
				UserDefinedDomainFromPrimitiveDomain {
					Name text
					Description 'text value'
					LengthValue 250
					InheritsPrimitiveDomain varchar2
				},

				//attributes
				AttributeIncludedInDB{
					Name facId
					Description 'faculty id'
					AttributeDomain number
				},
				AttributeIncludedInDB{
					Name facName
					Description 'faculty name'
```

```
                        AttributeDomain text
                    },
                    AttributeIncludedInDB{
                        Name facShortName
                        Description 'faculty short name'
                        AttributeDomain text
                    },
                    AttributeIncludedInDB{
                        Name depId
                        Description 'department id'
                        AttributeDomain number
                    },
                    AttributeIncludedInDB{
                        Name depName
                        Description 'department name'
                        AttributeDomain text
                    },
                    AttributeIncludedInDB{
                        Name studentId
                        Description 'student id'
                        AttributeDomain number
                    },
                    AttributeIncludedInDB{
                        Name nameSurname
                        Description 'student name and surname'
                        AttributeDomain text
                    },
                    AttributeIncludedInDB{
                        Name year
                        Description 'year of studies'
                        AttributeDomain number
                    }
                }

                //application systems
                ApplicationSystems{
                    ApplicationSystem {
                        Name faculty_organization
                        Description 'faculty organization system'
                        ApplicationType system
                        ChildApplicationSystems (
                            student_service
                        )
                        BusinessApplications {
                            BussinesApplication {
                                Name
BusinessApplicationFacultyOrganization
                                Description 'Business Application
Faculty Organization'
                                FormTypes{
                                    BussinesApplicationFormType {

    BussinesApplicationFormTypeName student
                                    }
                                }
                                EntryFormType
                                    BussinesApplicationFormType {

    BussinesApplicationFormTypeName  faculty
```

```
                                                        IsCallingFormType (

        CalledFormTypeStructure {

        SelectOnOpen false

        RestrictedSelect true
                                                                    ShowAsMenu
false

        ShowAsButton true

        HasCalledFormType student

        CalledFormTypeParameters {

        CalledFormTypeParameter {

        CalledFormTypeParameterName FacNameParam

        BindingOption

                BindingAttribute {

                        BindingAttributeName facId

                }
                                                                          },

        CalledFormTypeParameter {

        CalledFormTypeParameterName  FacShortNameParam

        BindingOption

                BindingParameter {

                        BindingParameterName P_1

                }
                                                                          }
                                                                    }

        CallingMode

        CallingModeNonModal

        CloseCallingForm true
                                                              }
                                                          )
                                                    }
                                        }//business application faculty
organization
                                    }//business applications faculty organization

                                    //owned formtypes
                                    OwnedFormTypes {
                                            //form type faculty
                                            FormTypeProgram {
```

```
                                                Name faculty
                                                Title Faculty
                                                ConsideredINDBSchDesign true
                                                FormTypeParameters {
                                                       FormTypeParameter {
                                                              Name P_1
                                                              Description "Parameter
decimal number"

       FormTypeParameterAttribute facId
                                                              FormTypeParameterDomain
number
                                                       }
                                                }
                                                ComponentTypeRoot {

                                                       //component type faculty
                                                       //ComponentType {
                                                              Name faculty
                                                              Title Faculty
                                                              Query true
                                                              Insert false
                                                              Delete true
                                                              Update true
                                                              ComponentTypeAttributes
{

       ComponentTypeAttribute {
                                                                                     Title
FacultyID

       ComponentTypeAttributeName facId
                                                                                     Mandatory
true

       ComponentTypeAttributeBehaviour

       Modifiable {

       QueryAllowed true

       InsertAllowed true

       NullifyAllowed false

       UpdateAllowed true
                                                                                            }
                                                                              },

       ComponentTypeAttribute{
                                                                                     Title
FacultyName

       ComponentTypeAttributeName facName
                                                                                     Mandatory
true

       ComponentTypeAttributeBehaviour
```

```
	Modifiable {
	QueryAllowed true
	InsertAllowed true
	NullifyAllowed false
	UpdateAllowed true
	}
	},
	ComponentTypeAttribute{
		Title
FacultyShortName
	ComponentTypeAttributeName facShortName
		Mandatory
true
	ComponentTypeAttributeBehaviour
	Modifiable {
	QueryAllowed true
	InsertAllowed true
	NullifyAllowed true
	UpdateAllowed true
	}
	}
}
ComponentTypeKeys {
	ComponentTypeKey
{
		Name key1
	ComponentTypeKeyAttributes (
	facId
		)
		Global
true
	},
	ComponentTypeKey
{
		Name key2
	ComponentTypeKeyAttributes (
	facShortName
		)
```

```
                                                                Global
true
                                                        }
                                                }
                                                ComponentTypeUniques {

        ComponentTypeUnique {
                                                                Name
unique1

        ComponentTypeUniqueAttributes (

        facName
                                                                )
                                                        }
                                                }
                                                ComponentTypeCheck
                                                                {

        Name check1

        Expression 'facid>0'
                                                                }
                                                ItemGroups {
                                                        ItemGroup {

        ItemGroupName 'Faculty details'

        ItemGroupTitle 'Faculty details'
                                                                Context
false
                                                                OverFlow
false

        ItemGroupAttributes {

        ItemAttribute {

        ItemAttributeName facId

        BreakLine true
                                                                        },

        ItemAttribute {

        ItemAttributeName facName

        BreakLine true
                                                                        },

        ItemAttribute {

        ItemAttributeName facShortName

        BreakLine true
                                                                        }
                                                                }
                                                        }
                                                }
```

```
                                                        ComponentTypeDisplay {

        SearchFuncionality true

        MassiveDeleteFunctionality true

        RetainLastInsertedRecord false

        DataLayoutValue FieldLayout

        ComponentTypeDisplayPosition

        NewWindowRelativePositionPredefined {

        WindowRelativePositionValue LeftOnTop
                                                                    }
                                                        }
                                                        ComponentTypeChildren {

                                                               //child
component type department

        ComponentTypeChild {
                                                                    Name
department
                                                                    Title
Department
                                                                    Query true
                                                                    Insert
true
                                                                    Delete
true
                                                                    Update
true

        NoOfOcurrences OneOrMany

        ComponentTypeAttributes{

        ComponentTypeAttribute {

        Title DepartmentID

        ComponentTypeAttributeName depId

        Mandatory true

        ComponentTypeAttributeBehaviour

               Modifiable {

                      QueryAllowed true

                      InsertAllowed true

                      NullifyAllowed true

                      UpdateAllowed true
```

```
            }
                                                                    },
        ComponentTypeAttribute {
        Title DepartmentName
        ComponentTypeAttributeName depName
        Mandatory true
        ComponentTypeAttributeBehaviour
               Modifiable {
                      QueryAllowed true
                      InsertAllowed true
                      NullifyAllowed false
                      UpdateAllowed true
               }
                                                                    }
                                                                }
        ComponentTypeKeys {
        ComponentTypeKey {
        Name key1
        ComponentTypeKeyAttributes (
                depId
        )
        Global true
                                                                    },
        ComponentTypeKey {
        Name key2
        ComponentTypeKeyAttributes (
               depName
        )
        Global true
                                                                    }
                                                                }
                                                                ItemGroups
{
```

```
	ItemGroup {
	ItemGroupName 'Department details'
	ItemGroupTitle 'Department details'
	Context false
	OverFlow false
	ItemGroupAttributes {
		ItemAttribute {
			ItemAttributeName depId
			BreakLine true
		},
		ItemAttribute {
			ItemAttributeName depName
			BreakLine true
		}
	}
	ComponentTypeDisplay {
	SearchFuncionality false
	MassiveDeleteFunctionality true
	RetainLastInsertedRecord true
	DataLayoutValue TableLayout
	ComponentTypeDisplayPosition
		SameWindow {
			Position BottomToParent
		}
```

```
}
}
}
}
}//component type department
}//child component types faculty
}//component type faculty
},// owned form type faculty
```

```
                                        //owned form type student
                                        FormTypeProgram {
                                               Name student
                                               Title Student
                                               ConsideredINDBSchDesign true
                                               FormTypeParameters {
                                                      FormTypeParameter {
                                                             Name FacNameParam
                                                             Description "Faculty
name parameter"
                                                             DefaultValue faculty

       FormTypeParameterAttribute facName
                                                             FormTypeParameterDomain
number
                                                      },
                                                      FormTypeParameter {
                                                             Name FacShortNameParam
                                                             Description "Faculty
short name parameter"
                                                             DefaultValue fac

       FormTypeParameterAttribute facShortName
                                                             FormTypeParameterDomain
text
                                                      }
                                               }
                                               //component types form type student


                                                      //component type student
                                                      ComponentTypeRoot {
                                                             Name student
                                                             Title Student
                                                             Query true
                                                             Insert true
                                                             Delete true
                                                             Update true
                                                             ComponentTypeAttributes
{

       ComponentTypeAttribute {
                                                                           Title
StudentID

       ComponentTypeAttributeName studentId
                                                                           Mandatory
true

       ComponentTypeAttributeBehaviour

       Modifiable {

       QueryAllowed true

       InsertAllowed true

       NullifyAllowed false
```

```
        UpdateAllowed true
                                                                    }
                                                                },
        ComponentTypeAttribute{
                                                                    Title
FacultyShortName
        ComponentTypeAttributeName facShortName
                                                                    Mandatory
true
        ComponentTypeAttributeBehaviour
        Modifiable {
        QueryAllowed true
        InsertAllowed true
        NullifyAllowed true
        UpdateAllowed true
                                                                    }
                                                                },

        ComponentTypeAttribute {
                                                                    Title
DepartmentName
        ComponentTypeAttributeName depName
                                                                    Mandatory
true
        ComponentTypeAttributeBehaviour
        Modifiable {
        QueryAllowed true
        InsertAllowed true
        NullifyAllowed false
        UpdateAllowed true
                                                                    }
                                                                },
        ComponentTypeAttribute{
                                                                    Title
NameSurname
        ComponentTypeAttributeName nameSurname
                                                                    Mandatory
true
```

```
        ComponentTypeAttributeBehaviour
        Modifiable {
        QueryAllowed true
        InsertAllowed true
        NullifyAllowed true
        UpdateAllowed true
                                                                    }
                                                            },

        ComponentTypeAttribute {
                                                                    Title
YearOfStudy
        ComponentTypeAttributeName year
                                                                    Mandatory
false
        ComponentTypeAttributeBehaviour
        Modifiable {
        QueryAllowed true
        InsertAllowed true
        NullifyAllowed false
        UpdateAllowed false
                                                                    }
                                                            }
                                                    }
                                                    ComponentTypeKeys {
                                                            ComponentTypeKey
{
                                                                    Name key1
        ComponentTypeKeyAttributes (
        studentId
                                                                    )
                                                                    Global
true
                                                            }
                                                    }

                                                    ItemGroups {
                                                            ItemGroup {
```

```
       ItemGroupName studentPersonalDetails

       ItemGroupTitle "Student personal details"
                                                                    Context
true
                                                                    OverFlow
false

       ItemGroupAttributes {

       ItemAttribute {

       ItemAttributeName studentId

       BreakLine true
                                                                           },

       ItemAttribute {

       ItemAttributeName nameSurname

       BreakLine false
                                                                           }
                                                                    }
                                                                 },
                                                                 ItemGroup {

       ItemGroupName studentOtherDetails

       ItemGroupTitle "Student other details"
                                                                    Context
false
                                                                    OverFlow
false

       ItemGroupAttributes {

       ItemAttribute {

       ItemAttributeName facShortName

       BreakLine false
                                                                           },

       ItemAttribute {

       ItemAttributeName depName

       BreakLine false
                                                                           },

       ItemAttribute {

       ItemAttributeName year

       BreakLine false
                                                                           }
                                                                    }
```

```
                                                                        }
                                                                }

                                                                ComponentTypeDisplay {

SearchFuncionality false

MassiveDeleteFunctionality false

RetainLastInsertedRecord false

DataLayoutValue FieldLayout

ComponentTypeDisplayPosition

NewWindowRelativePositionCustom {

XRelativePosition 42

YRelativePosition 90
                                                                                                }
                                                                }

                                                        }//component type student
                                                } //owned form type student
                                        }//owned form types faculty organization
                                },//application system faculty organization
                                ApplicationSystem {
                                        Name student_service
                                        Description 'student service subsystem'
                                        ApplicationType subsystem

                                }
                        }//application system project
                }//project
}
Templates {
        Template {
                Name 'Classic Template'
                ItemFont
                        {
                                Style Arial
                                Size 12
                                Bold true
                                Italic false
                                Underline false
                        }
                LabelFont
                        {
                                Style TimesNewRoman
                                Size 12
                                Bold false
                                Italic false
                                Underline false
                        }
                ApplicationWindowProperty
                        {
                                Resizeable true
                                ApplicationIconPath 'C:/icons/icon.gif'
```

```
                        ForegroundColor
                                Hsb {
                                        Hue 10 Saturation 20 Brightness 60
                                }
                        BackgroundColor
                                Hsb {
                                        Hue 20 Saturation 40 Brightness 60
                                }
                        DesktopColor
                                Hsb {
                                        Hue 30 Saturation 20 Brightness 60
                                }
                        FixedScreenSize {
                                        Width 200
                                        Height 600
                                }
                }
        ScreenFormProperty
                {
                        CallFromMenu true
                        CallButtonClick false
                        CallKeyPress false
                        ForegroundColor
                                Rgb {
                                        Red 100 Green 200 Blue 100
                                }
                        BackgroundColor
                                Rgb {
                                        Red 255 Green 255 Blue 255
                                }
                }
        TableProperty
                 {
                        RowHeight 15
                        ColumnWidth 50
                        NoOfVisibleRows 20
                        ForegroundColor
                                Rgb {
                                                Red 100 Green 200 Blue 100
                                        }
                        BackgroundColor
                                Rgb {
                                                Red 100 Green 200 Blue 100
                                        }
                        BorderStyle   EtchedRaised

                        TableOverflow
                                 {
                                        Size 90
                                        Position Right
                                }
                        CellBackgroundColor
                                Rgb {
                                                Red 100 Green 200 Blue 100
                                        }
                        CellForegroundColor
                                Rgb {
                                                Red 100 Green 200 Blue 100
                                        }
```

```
        }
ItemPanelProperty
        {
                PanelSpacing 15
                ItemSpacing 20
                PanelAlignmentLabel Start
                PanelAlignmentInputField Start
                NestedPanelAlignmentLabel End
                NestedPanelAlignmentInputField Start
                ForegroundColor
                        Rgb {
                                        Red 100 Green 200 Blue 100
                                }
                BackgroundColor
                        Rgb {
                                        Red 100 Green 200 Blue 100
                                }
                BorderStyle BevelRaised
                NestedPanelBorder
                                BorderStyle BevelRaised

                InputBackgroundColor
                        Rgb {
                                        Red 100 Green 200 Blue 100
                                }
                InputForegroundColor
                        Rgb {
                                        Red 100 Green 200 Blue 100
                                }
                ContexTable
                        {
                                Height 15
                        }
        }
ItemTypesProperty {
        {
                ItemName CheckBox
                TitleAlign Left
                TitlePosition Left
                TitleOffset 10
                TitleItalic false
                TitleBold true
        },
        {
                ItemName RadioButton
                TitleAlign Left
                TitlePosition Left
                TitleOffset 10
                TitleItalic false
                TitleBold true
        },
        {
                ItemName TextBox
                TitleAlign Left
                TitlePosition Left
                TitleOffset 10
                TitleItalic false
                TitleBold true
                BorderStyle Etched
```

```
                        },
                         {
                              ItemName List
                              TitleAlign Left
                              TitlePosition Left
                              TitleOffset 10
                              TitleItalic false
                              TitleBold true
                              BorderStyle Etched
                        },
                         {
                              ItemName ComboBox
                              TitleAlign Left
                              TitlePosition Left
                              TitleOffset 10
                              TitleItalic false
                              TitleBold true
                              BorderStyle Etched
                        }
                  }
                  ButtonProperty
                         {
                              View Textual
                              ForegroundColor
                                    Rgb {
                                          Red 100 Green 200 Blue 100
                                    }
                              BackgroundColor
                                    Rgb {
                                                Red 100 Green 200 Blue 100
                                          }
                              BorderStyle EtchedRaised
                        }
            }
      }
}
```

Prilog 3 – Spisak često korišćenih skraćenica

DSL	Domain Specific Language
EER	Extended Entity-Relationship
EMF	Eclipse Modeling Framework
GMF	Graphical Modeling Framework
IIS*Case	Integrated Information Systems CASE Tool
IS	Informacioni sistem
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
MDSE	Model Driven Software Engineering
MOF	Meta Object Facility
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
SUBP	Sistem za upravljanje bazama podataka
UML	Unified Modeling Languange
CASE	Computer-aided software engineering

Prilog 4 – Spisak slika