УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Никола Обреновић

Прилог пројектовању, консолидацији и трансформацијама ограничења торке шеме базе података, заснован на платформски независним моделима

ДОКТОРСКА ДИСЕРТАЦИЈА

Нови Сад, 2015.

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 Никола Обреновић
Ментор, **МН**:		др Иван Луковић, редовни професор
Наслов рада, **НР**:		Прилог пројектовању, консолидацији и трансформацијама ограничења торке шеме базе података, заснован на платформски независним моделима
Језик публикације, **ЈП**:		Српски
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Србија
Уже географско подручје, **УГП**:		АП Војводина
Година, **ГО**:		2015
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)		9/107/81/0/50/0/0
Научна област, **НО**:		Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:		Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, **ПО**:		Пројектовање база података, ограничења вредности, платформски независни модели, трансформације модела, генерисање изворног кôда, консолидација модела
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 генерисања прототипова у развоју 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 треба да пружи одговарајући допринос у остварењу наведеног циља. У дисертацији представљени су алгоритми за трансформацију модела ограничења вредности у извршив кôд и консолидацију подшема са јединственом шемом базе података, са аспекта ограничења вредности.
Датум прихватања теме, **ДП**:		8. 5. 2014.
Датум одбране, **ДО**:		10. 10. 2015.
Чланови комисије, **КО**:	Председник:	др Зоран Марјановић, редовни професор
	Члан:	др Силвиа Гилезан, редовни професор
	Члан:	др Соња Ристић, ванредни професор
	Члан:	др Славица Кордић, 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**:		Nikola Obrenović, M.Sc.
Mentor, **MN**:		Ivan Luković, Ph.D., Full Professor
Title, **TI**:		An Approach to Design, Consolidation and Transformations of Database Schema Check Constraints Based on Platform Independent Models
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)		9/107/81/0/50/0/0
Scientific field, **SF**:		Software engineering
Scientific discipline, **SD**:		Applied computer science
Subject/Key words, **S/KW**:		Database Design; Check constraints; Platform Independent Models; Model Transformations; Source code generation; Model consolidation.
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 generation of prototypes in information systems development 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. This dissertation should provide a contribution towards fulfilling the given goal. In the dissertation, author presents algorithms for check constraint model into executable code transformations and algorithms for testing subschema consolidation with respect to check constraints.
Accepted by the Scientific Board on, **ASB**:		May 8th, 2014
Defended on, **DE**:		October 10th, 2015
Defended Board, **DB**:	President:	Zoran Marjanović, Ph.D., Full Professor
	Member:	Silvia Gilezan, Ph.D., Full Professor
	Member:	Sonja Ristić, Ph.D., Associate Professor
	Member:	Slavica Kordić, Ph.D., Assistant Professor
	Member, Mentor:	Ivan Luković, Ph.D., Full Professor

Mentor's sign

Захваљујем свом ментору, проф. др Ивану Луковићу, на великој количини знања и савета које ми је пренео током докторских студија и израде докторске дисертације.

Такође, захваљујем породици, пријатељима и колегама из Schneider Electric DMS NS-а на свесрдној подршци и помоћи.

Садржај

Списак слика

1 Увод

Изградња информационог система започиње прикупљањем корисничких захтева, често у првом моменту исказаних у облику неформализованог и неструктурираног текстуалног записа. На основу прикупљених захтева, приступа се изради формализованог модела информационог система. При томе, циљ је да модел садржи што детаљнију и прецизнију репрезентацију стварног система и укључи што више објеката који ће бити коришћени у развоју. Након завршеног моделовања, почиње развој информационог система у изабраном софтверском окружењу, уз употребу изабраних технологија и програмских језика.

Према „класичном“ и за дужи низ година устаљеном начину развоја информационих система, модели базе података и модели софтверских апликација могу да садрже елементе специфичне за имплементациону платформу, а само програмирање софтвера ради се, најчешће, ручно.

Са друге стране, у последњој деценији, све више су заступљене и пропагирају се методологије развоја софтвера засноване на моделима (енг. *model-driven software development, MDSD*) [1, 2]. Идеја ових методологија је да модел софтвера мора бити независан од имплементационе платформе и да морају постојати аутоматски механизми који моделе преводе у извршиви кôд. Платформска независност модела омогућава да пројектант не буде оптерећен имплементационим детаљима, односно специфичностима платформе, током дизајна система и олакшава се будућа миграција система на нову платформу. Аутоматско генерисање кôда поједностављује и убрзава развој и спречава уношење грешака које настају приликом ручног писања кôда.

Из тих разлога, на Факултету техничких наука, Универзитета у Новом Саду, развијају се технике за платформски независно моделовање информационих система и алгоритми за аутоматску трансформацију платформски независних модела у извршиве прототипове информационих система. Тренутно је омогућено моделовање шеме базе података [3], корисничког интефејса [4] и пословне логике [5]. Такође, развијени су алгоритми за:

1. трансформацију платформски независног модела базе података у релациони модел

података и даље у извршиви *SQL* кôд [3, 6],

2. транформацију модела информационог система и модела корисничког интерфејса у извршиви прототип информационог система [4] и
3. трансформацију модела пословне логике у извршиви *T-Sql* и *PL/SQL* код [7].

Ограничења података представљају део модела информационог система, тачније део модела базе података на коју се информациони систем ослања. Њихова намена је да очувају тачност и интегритет података са аспекта природе података и пословних правила која важе у моделованом систему. Ограничења података обогаћују шему базе података семантиком и ослобађају апликативне слојеве обавезе да брину о интегритету података у бази података.

Модерни системи за управљање базама података (СУБП) заснивају се на релационом моделу података који дозвољава дефинисање различитих врста ограничења података. Са друге стране, у пракси ограничења података ретко се моделују, а често се и не имплементирају. Ово је посебно изражено у случају ограничења вредности (енг. *check constraint*) која представљају битан механизам за дефинисање пословних правила у релационом моделу података. Разлог за избегавање реализације, или чак и прецизног моделовања ограничења вредности, поготово када су она комплекснија, је проблем непостојања једноставних и свеобухватних техника за платформски-независно моделовање и алата и процедура за генерисање извршних спецификација за комерцијалне платформе.

У складу са наведеним проблемом, дефинисана је прва хипотеза ове докторске дисертације:

X1: Могуће је креирати кориснички оријентисан приступ за развој ограничења вредности у релационим базама података, који ће укључити платформски независно моделовање ограничења вредности, аутоматску трансформацију у платформски зависан модел и аутоматско генерисање извршивог *SQL/DDL* кôда.

У релационом моделу података, ограничење вредности може бити специфицирано на нивоу домена, атрибута или шеме релације и представља логички израз који специфицира скуп дозвољених вредности које могу бити додељене атрибуту или торки релације. У случају када је ограничење вредности специфицирано на нивоу шеме релације, ограничење се такође скраћено назива и ограничење торке.

Током ранијих истраживања, развијен је наменски језик (енг. *domain specific language, DSL*) који служи за моделовање ограничења вредности [8] у оквиру платформски независног модела података који се користи у алату *IIS*Case*. У циљу потврђивања или оповргавања хипотезе X1, дефинисан је први задатак ове докторске дисертације да се дефинишу алгоритми за трансформацију платформски независног модела ограничења вредности у релациони модел података и алгоритми за њихову даљу трансформацију у извршиви *SQL/DDL* кôд.

Информациони системи у пракси често обухватају велик број концепата и моделују велик број ентитета, због чега прављење јединственог модела информационог система често превазилази когнитивне способности појединца и за резултат има модел ниског квалитета [9, 10]. Из тог разлога, у [11] и [9] предлаже се поступно и независно моделовање делова информационог система, односно подсистема, од стране више пројектаната, прво на платформски независном нивоу. Надаље, платформски независне спецификације које чине моделе појединачних подсистема, трансформишу се у

подшеме базе података, а унија свих спецификација подсистема трансформише се у јединствену шему базе података, на нивоу целог информационог система. Прецизне дефиниције трансформација могу се наћи у [6], [12] и [3].

Кориснички графички интефејс и пословна логика информационог система специфицира се такође кроз моделе информационих подсистема и имплементира ослањајући се на концепте подшеме базе података која се добија трансформацијом модела одговарајућег подсистема. С друге стране, подшема базе података је логички концепт који представља скуп погледа на јединствену шему базе података целог информационог система [9] и операције над подацима које су издате кроз кориснички интефејс информационог подсистема заправо се извршавају над јединственом шемом базе података информационог система.

Да би операције селекције и ажурирања података покренуте на нивоу информационог подсистема очувале интегритет података јединствене базе података, свака подшема базе података мора бити формално усклађена са јединственом шемом базе података информационог система. Појам и дефиниција формалне усклађености уведени су и детаљно дискутовани у [9], а у даљем тексту овај појам је описан само на нивоу детаљности потребном за овај рад.

Потребан услов да подшема буде формално усклађена са јединственом шемом базе података је да сви њени концепти подшеме буду формално усклађени са истим концептима шеме. У [9], дефинисани су алгоритми за проверу усклађености подшема са аспекта домена, шема релација, примарних кључева, ограничења јединствености обележја, ограничења референцијалног интегритета и ограничења инверзног референцијалног интегритета. Ови алгортими су и имплементирани у алату *IIS*Case*.

Наставак истраживања презентованог у раду [9], реализован је кроз ову докторску дисертацију. Хипотеза тог истраживања представља другу хипотезу ове дисертације и гласи:

X2: Могуће је развити алгоритам за проверу усклађености подшеме са јединственом шемом базе података, са аспекта ограничења вредности.

Сагласно хипотези X2, развој таквог алгоритма постављен је за други задатак истраживања у овој докторској дисертацији и представља други важан допринос ове дисертације. Основни проблем у дефинисању овог алгоритма представља решавање импликационог проблема за ограничења вредности, што ће детаљно бити објашњено у дисертацији. По својој природи, ограничења вредности представљају комплексније логичке изразе у односу на остале типове ограничења у релационим базама података. Такође, ограничења вредности у својој дефиницији дозвољавају изразе аритметике целих и реалних бројева и операције над датумима, скуповима и стринговима, односно изразе који излазе из оквира математичке логике. Стога, провера усклађености ограничења вредности захтева другачије алгоритме у односу на све остале типове ограничења у релационом моделу података.

Истраживање у оквиру ове докторске дисертације спроведено је кроз следеће кораке:

1. из искуства стечених кроз академске и индустријске пројекте, идентификован је проблем: **недостатак кориснички-оријентисаних метода и техника, вођених моделом, за развој ограничења вредности**,
2. на основу проблема, **дефинисане су горенаведене хипотезе X1 и X2**,

3. дефинисан је циљ ове дисертације, **да понуди кориснички-оријентисан приступ за моделовање, генерисање и консолидацију ограничења вредности, који ће омогућити ширу употребу ограничења вредности у моделовању и имплементацији база података**,
4. спроведено је истраживање у области са циљем проналажења постојећег решења за идентификовани проблем или решења сличних или повезаних проблема,
5. дефинисан је задатак: **развити кориснички-оријентисане технике и алгоритме за трансформацију платформски независно моделованих ограничења вредности у релациони модел података, генерисање извршивог кôда и консолидацију подшема са аспекта ограничења вредности**,
6. предложено је сопствено решење: **креиране су потребне технике и алгоритми и дати су математички докази њихове коректности**,
7. имплементирано је решење: **развијене технике и алгоритми имплементирани су у алату за моделом вођено пројектовање информационих система и база података и генерисање прототипова, *IIS*Case***,
8. предложени су правци унапређивања предложеног решења и даљег истраживања.

Резултат истраживања спроведеног у овој докторској дисертацији представљају предложени алгоритми и технике који омогућавају развој ограничења вредности заснован на платформски независним моделима. У академским и привредним круговима, постоји став да развој софтвера заснован на моделима омогућава бржи, прецизнији и једноставнији развој софтвера у односу на директно програмирање на одређеној имплементационој платформи [1, 2, 7]. У складу са тим, резултати ове дисертације требало би да допринесу унапређењу процеса развоја информационих система и обезбеђењу активног коришћења ограничења вредности у постизању одговарајућег квалитета испројектованих шема базе података информационих система.

Текст ове докторске дисертације структуиран је на следећи начин. Поред увода, дисертација садржи још 8 поглаља.

У другом поглављу, дат је преглед стања у области истраживања ограничења вредности, и посебно ограничења торке, као и усклађености ових ограничења у релационим базама података. У истом поглављу, описани су и доступни комерцијални алати за пројектовање база података и њихове функционалности у погледу развоја ограничења вредности. Циљ прегледа стања у области истраживања у овој докторској дисертацији био је анализа постојећих решења за развој ограничења вредности и посебно ограничења торке, заснован на моделима и да се утврди да ли постоји потреба за креирањем сопственог решења.

У трећем поглављу описан је алат *IIS*Case* који служи за моделовање и аутоматско генерисање база података и информационих система. Уз алат, описани су и:

- платформски независан модел релационе базе података, чији део је и модел ограничења вредности,
- постојеће функционалности генерисања релационе шеме базе података, извршивог *SQL/DDL* кôда и консолидације подшема база података са

јединственом шемом.

На тај начин, представљен је контекст у којем су решавани проблеми дисертације и истраживачки рад на који се ова дисертација надовезује.

Платформски независан модел ограничења вредности приказан је у четвртом поглављу. Приказани модел користи се у алату *IIS*Case* за спецификацију ограничења вредности и представља улазни податак за трансформације ограничења вредности у релациони модел података, које су развијене у овој дисертацији.

У петом поглављу, дефинисани су алгоритми за трансформацију платформски независног модела ограничења вредности у релациони модел података и дати су формални докази коректности и оцена њихове сложености.

Развијени алгоритми и шаблони изворног кôда који су употребљени у циљу обезбеђења подршке за генерисање извршивих спецификација моделованих ограничења вредности, представљени су у шестом поглављу.

Кроз развијене алгоритме и шаблоне кôда, који су представљени у поглављима 5 и 6, створен је начин за развој ограничења вредности заснован на платформски независном моделу, што представља први резултат истаживања у оквиру ове докторске дисертације. Основни резултати приказани у поглављима 5 и 6, такође су презентовани и у раду аутора докторске дисертације [13].

Алгоритам за проверу консолидованости подшеме са јединственом шемом базе података са аспекта ограничења вредности представљен је у поглављу 7. У истом поглављу дати су формални доказ коректности алгоритма и оцена његове комплексности. Представљени алгоритам омогућава развој ограничења вредности у склопу паралелног развоја делова информационог система од стране више пројектаната и интеграцију независно моделованих ограничења вредности у јединствену шему базе података. Опис алгоритма презентован је и у раду аутора докторске дисертације [14].

Закључци истаживања спроведеног у оквиру ове дисертације, као и предложени правци даљег истраживања, дати су поглављу 8.

У деветом поглављу дат је преглед литературе која је коришћена у изради ове докторске дисертације.

2 Актуелно стање у области

Током учешћа на великом броју пројеката развоја информационих система у привредном и академском окружењу, примећено је да пројектанти често изостављају ограничења вредности из модела базе података, или да она не буду уопште имплементирана у бази података, или пак да имплементација не одговара моделованом ограничењу. Нежељена последица тога је да подаци у имплементираној бази података могу нарушити пословно правило које би требало да буде моделовано помоћу изостављеног ограничења. Као узрок оваквог стања, а што је уједно препознато и као проблем постављен пред ову дисертацију, идентификовано је непостојање одговарајућих, кориснички-оријентисаних метода и техника за развој ограничења вредности.

Како би постојећа решења наведеног проблема, или бар једног његовог дела била истражена, анализиране су и представљене у даљем тексту следеће категорије радова, као и постојећа софтверска решења:

1. радови на тему платформски независног моделовања база података, и аутоматског генерисања извршивог *SQL/DDL* кôда,
2. доступни комерцијални *CASE* алати за пројектовање релационих база података и
3. радови на тему консолидације независно моделованих подшема база података са јединственом имплементационом шемом.

На тему моделовања информационих система и база података, и аутоматског генерисања извршивог *SQL/DDL* кôда, издвајају се радови описани у наредним пасусима.

Демут, Хусман и Лехер у [15, 16] користе *Object Constraint Language* (*OCL*, [17]) за платформски-независно моделовање ограничења вредности. Наведени аутори користе *Unified Modeling Language* (*UML*) моделе за моделовање шема релационих база података па *OCL* представља природан начин за моделовање ограничења вредности. Такође, аутори су презентовали мапирања типова *OCL* израза на *SQL* и показали да сви

типови *OCL* израза, сем *iterate*, могу бити представљени *SQL* изразима. Предност свог приступа аутори виде у чињеници да су UML модели и OCL језик веома распрострањени у употреби међу пројектантима информационих, или чак уопште, софтверских система.

Са друге стране, наш приступ заснива се на моделу типова форми за моделовање база података и кориснички оријентисаном *DSL* језику за моделовање ограничења вредности. Из тог разлога, модел ограничења вредности, који се предлаже нашим приступом, требао би да буде ближи и разумљивији крајњем кориснику, и да му омогући да учествује у пројектовању информационог система. Такође, Демут, Хусман и Лехер не разматрају проблем консолидације ограничења вредности и идентификације конфликата између ограничења, док је у овом раду представљено једно могуће решење тог проблема.

У [18], Кабот и Тениенте дали су приказ модерних *CASE* алата у којем тврде да алати не пружају задовољавајуће начине за моделовање ограничења вредности и аутоматско генерисање *SQL* кода из модела. Према њиховом мишљењу, идеалан *CASE* алат треба да задовољи, између осталих, следеће особине:

1. платфомски независан модел ограничења вредности, који треба да омогући моделовање свих типова ограничења вредности која се срећу у пракси,
2. генерисани кôд ограничења вредности треба да буде платформски зависан и оптимизован за изабрану платформу и
3. трансформације између различитих модела података и између модела и кода треба да буду независне од циљне платформе за коју се код генерише.

Приступ за развој ограничења вредности у алату *IIS*Case*, који је описан у овој дисертацији, задовољава наведене особине на следеће начине:

1. модел ограничења вредности је део модела типова форми који је у потпуности платформски независан, што је детаљно описано у поглављу 4,
2. трансформација платформски независног модела у платформски зависан модел ограничења вредности је независна од изабране циљне SQL платформе, што је представљено у поглављу 5 и
3. *IIS*Case* може да генерише *SQL/DDL* кôд по *ANSI SQL-2003* стандарду, као и за комерцијалне СУБП *Oracle 9i* и *10g*, и *MS SQL Server 2000* и *2008*. При томе, генерисани кôд узима у обзир специфичности изабране платформе и оптимизован је за њу. Генерисање *SQL/DDL* кôда у алату *IIS*Case* представљено је у поглављу 6.

Од наведених особина, *IIS*Case* и даље не задовољава део 3. особине да трансформација модела ограничења вредности у *SQL/DDL* кôд буде независна од изабране платформе. Оваква трансформација излази из опсега ове дисертација и може представљати један од праваца будућих истраживања.

У склопу истраживања за потребе ове докторске дисертације, разматрано је неколико комерцијалних *CASE* алата који омогућавају моделовање база података и информационих система: *Oracle Designer* [19], *Sybase PowerDesigner* [20], *Enterprise Architect* [21], *IBM Rational Rose Data Modeler* [22], *CA ERwin Data Modeler* [23] и *Toad Data Modeler* [24]. Наведени алати разматрани су са аспекта моделовања ограничења вредности и закључци су дати у наставку текста.

Сви разматрани алати нуде моделовање база података путем модела ентитета и веза (енг. *entity-relationship* (*ER*) *data model*), али не подржавају потпуни животни циклус моделовања ограничења вредности и трансформисања њихових модела у извршиви *SQL* код. На пример, у алатима *Oracle Designer*, *Sybase PowerDesigner* и *Toad*

Data Modeler ограничења вредности могу се специфицирати једино путем *SQL* кода који одговара изабраној платформи на којој ће моделована база података бити имплементирана. Са друге стране, наведени алати не нуде могућност да се ограничења вредности моделују на платформски независан начин, који је подржан у алату *IIS*Case*.

Помоћу алата *ERwin Data Modeler* и *Enterprise Architect* ограничења вредности могу бити специфицирана на платформски независан начин само на нивоу атрибута, док ограничења вредности на нивоу домена и шема релација нису подржана. Због тога, један недостатак ових алата је, на пример, да је немогуће моделовати ограничење вредности чији логички израз референцира више од једног обележја шеме релације.

Са друге стране, алат *IIS*Case* омогућава спецификацију ограничења вредности на сва три нивоа као и ограничења вредности које се односе на више од једне шеме релације. Ова функционалност алата *IIS*Case* је за праксу веома битна, јер се управо у пракси често јавља потреба за моделовањем различитих ограничења вредности.

Алат *IBM Rational Rose Data Modeler* омогућава платфомски-зависно моделовање ограничења вредности на нивоу атрибута и шема релације у форми *SQL* језика. Такође, алат нуди могућност моделовања ограничења вредности који референцирају више шема релација кроз *OCL* језик, што је предност која га издваја од осталих алата. Ипак, овај алат не омогућава трансформацију модела ограничења вредности у *SQL* код.

На основу проучене доступне литературе и анализираних постојећих комерцијалних алата, закључује се да је потребно креирати нов кориснички оријентисан приступ за развој ограничења вредности у релационим базама података, чиме ће бити потврђена хипотеза X1 дефинисана у првом поглављу. При томе, приступ треба да задовољи горенаведене особине из [18].

У даљем тексту представљени су радови на тему интеграције и консолидације независно моделованих шема база података и провере задовољивости скупа ограничења која су по својој намени и структури слична ограничењима вредности.

У [10] дат је исцрпан преглед методологија и техника за интеграцију независно моделованих подшема база података. Већина приказаних методологија укључује корак детекције и разрешавања конфликата између подшема. Разматрани су конфликти са аспекта именовања, домена, кардиналитета веза између ентитета, примарних кључева и намене ентитета у виду уноса и брисања података, али ниједна методологија не разматра конфликте између ограничења вредности.

Такође, у закључку у [10] констатује се да већина методологија не дефинише алгоритме за разрешење конфликата нити предлажу имплементације аутоматизоване интеграције подшема база података. С друге стране, у склопу алата *IIS*Case* развијени су и имплементирани алгоритми за аутоматизовану интеграцију модела информационих подсистема. Поступак интеграције резултује јединственом шемом базе података информационог система. Такође, развијени су и алгоритми за проверу усаглашености подшема са јединственом шемом базе података [9, 12]. Имплементирани алгоритмни за проверу усаглашености укључују детекцију конфликата између подшеме и јединствене шеме базе података, а где је могуће, и поступке за њихово аутоматско разрешавање.

Рам и Бернштајн су у [79] дали преглед постојећих метода за идентификацију и мапирање семантички једнаких елемената различитих шема релационих база података, што представља предуслов за успешну интеграцију шема. У раду се такође констатује да је овај процес потребно аутоматизовати на нивоу независном од конкретног модела података и представљен је генерички алгоритам мапирања, који је применљив и на

релационе шеме база података.

Са аспекта релационих шема база података, предложени генерички алгоритам мапирања обухвата мапирање шема релација и атрибута, користећи ограничења домена, ограничења кључа и ограничења јединствене вредности, али не и ограничења вредности. Такође, ниједна од осталих приказаних метода за мапирање не укључује поређење логичких израза ограничења вредности, па се идеје, садржане у њима, не могу употребити у овој дисертацији.

На пример, у [80] представљен је алат *SEMINT* који служи за проналажење атрибута из различитих релационих база података, који моделују исти атрибут реалног система. Алат користи неуронске мреже како би идентификовао кореспондентне атрибуте а атрибути се пореде преко имена, дефиниције и статистичких одлика својих вредности, попут аритметичке средине и стандардне девијације. Поређење дефиниција два атрибута обухвата и поређење постојања ограничења вредности, али се логички изрази ограничења вредности не пореде.

Ли и Линг представили су у [81] интеграцију концептуалних модела база података, која се врши на нивоу модела типова ентитета и повезника. При томе, аутори су дефинисали алгоритме за разрешавање конфликата у именовању, дефиницији домена, кључевима и следећим типовима ограничења:

1. дозвољене вредности домена,
2. кардиналитет атрибута у оквиру типа ентитета и
3. кардиналитети типова повезника.

По методологији развоја информационих система, имплементираној у алату *IIS*Case*, домени и атрибути моделују се на нивоу целог информационог система чиме се избегавају конфликти на тим елементима. Такође, еквивалент конфликтима између типова повезника, у алату *IIS*Case* представљају конфликти између шема релација. што је већ разматрано у [9]. Са друге стране, аутори Ли и Линг не разматрају идентификацију конфликата између ограничења вредности, што је један од главних задатака ове дисертације.

У [25] и [26], представљена је употреба система за логичко програмирање са ограничењима, ECL^iPS^e,за доказивање задовољивости скупа *OCL* ограничења. При томе, да би наведени језик био искоришћен, потребно је ограничити домене атрибута који се појављују у ограничењима на коначне скупове и задовољивост се проверава само у оквиру тих скупова вредности (енг. *bounded checking*). Односно, ако скуп ограничења није задовољив у оквиру дефинисаних коначних домена, не значи да није задовољив у оквиру оригиналних, неограничених, домена атрибута.

С друге стране, да би била проверена конзистентност ограничења торки, потребно је доказати валидност импликационе формуле између два ограничења торке, што се спроводи доказивањем да негација импликационе формуле није задовољива на целом домену својих атрибута. Стога, употреба система за логичко програмирање са ограничењима није прикладна за практичну проверу консолидованости ограничења торки.

У [27], аутори предлажу метод за проверу задовољивости скупа специфицираних ограничења интегритета у дедуктивним базама података. У њиховом случају ограничења су, по својој структури, формуле предикатске логике првог реда и метод за проверу њихове задовољивости је модификовани метод таблоа. У општем домену релационих база података, овај метод би такође био примењив за доказивање

задовољивости скупа ограничења која могу бити представљена предикатском логиком, попут ограничења примарног кључа или референцијалног интегритета.

Насупрот томе, логички изрази ограничења вредности могу да садрже аритметичке операције и релације или операције и релације над стринговима, датумима и скуповима. Како метод представљен у [27] не узима у обзир значење оператора и релација изван предикатске логике, односно не врши никакво резоновање над њима, он није прикладан за проверу усклађености подшеме са јединственом шемом базе података са аспекта ограничења вредности, и стога, није примењив у овој докторској дисертацији.

У [28], аутори анализирају ограничења интегритета у *XML* базама података која су специфицирана употребом хибридне модалне логике [29] и предлажу метод за доказивање задовољивости скупа ограничења који се такође заснива на методи таблоа. Као и у претходном случају, анализирана ограничења не садрже изразе из аритметике целих или реалних бројева, нити изразе са стринг операторима, који се, пак, често јављају у ограничењима вредности. Из тог разлога, ни овај предложени метод не укључује резоновање над наведеним изразима и стога не може бити употребљен у овој дисертацији.

Формика у [30] представила је алгоритам за детекцију конфликата унутар скупа ограничења интегритета дефинисаних у шеми објектно-оријентисане базе података. При томе, ауторка се ограничила на ограничења интегритета облика

$$i_c\text{: } this.p_1. \ldots .p_n\ \theta\ C$$

где су p_i ($i \in \{1, \ldots, n\}$) обележја, C је константа одговарајућег типа а θ један од оператора поређења из следећег скупа $\{=, <, \leq\}$.

С друге стране, логички изрази који се јављају у ограничењима вредности могу имати комплексније облике од анализираних ограничења и дозвољавају већи скуп оператора, па би алгоритам представљен у [30] могао бити употребљен само над малим подскупом ограничења вредности. Из тог разлога, наведени алгоритам није изабран за проверу консолидованости подшема у односу на ограничења вредности.

У [31], аутори представљају механизам за проверу задовољивости скупа *OCL* ограничења. Аутори прво трансформишу *OCL* ограничења у изразе математичке логике првог реда а затим проверавају задовољивост добијених логичких израза алатом за аутоматско доказивање теорема, *Prover9* [32], и такође *Satisfiability Modulo Theory* (*SMT*) решавачем, *Yices* [33]. При томе, аутори су дошли до закључка да је *SMT* решавач био успешнији од алата за аутоматско доказивање теорема, јер је обучен да резонује, поред математичке логике, и над аксиомима и теоремама аритметике целих бројева.

Бланшет, Беме и Паулсон у [34] представили су интеграцију *SMT* решавача у систем за доказивање задовољивости формула логике вишег реда, *Isabelle/HOL*, односно његов модул *Sledgehammer*. Циљ интеграције био је да се *SMT* решавачи употребе за доказивање формула које садрже изразе и из других теорија поред израза математичке логике.

У овој дисертацији закључено је да алгоритми и методе за проверу задовољивости у оквиру предикатске логике нису довољно моћне за проналажење конфликата између ограничења торки подшеме и ограничења торки интегрисане шеме базе података, и да је потребно укључити и резоновање над аритметичким изразима, као и изразима који садрже знаковне (алфанумеричке), датумске и скуповне операторе. У ту сврху, у дисертацији употребљени су *SMT* решавачи, из истих мотива као и у [31] и [34]. Због

тога, *SMT* решавачи представљени су детаљно у поглављу 7.

На основу проучене доступне литературе, може се закључити да није пронађен одговарајући алгоритам за проверу консолидованости подшема база података са аспекта ограничења вредности. Стога, потребно је развити такав алгоритам, у циљу потврђивања хипотезе X2, дефинисане у првом поглављу.

На основу анализиране литературе и постојећих *CASE* алата, могу се формулисати и следећи општи закључци:

- истраживања у правцу платформски независног моделовања и аутоматског генерисања извршивог кôда за ограничења вредности релационих база података ретко се срећу и у привредном и у академском окружењу,
- постојећи алати такође нуде ограничене могућности по питању развоја ограничења вредности, иако у пракси постоји потреба за тим,
- жељени развој ограничења вредности треба да почне од платформски независног модела, који пројектантски алат аутоматски трансформише у релациони модел података и даље у *SQL/DDL* кôд, што је представљено у поглављима 4, 5 и 6,
- у литератури није пронађен алгоритам за проверу консолидованости подшема са јединственом шемом базе података, из угла ограничења вредности и
- за доказивање задовољивости скупа логичких израза који садрже и изразе других теорија осим математичке логике, потребно је искористити *SMT* решаваче, што је урађено у поглављу 7.

Наведени закључци оправдавају циљеве и задатке ове докторске дисертације, формулисане у претходном поглављу.

3 Развој информационих система у окружењу *IIS*Studio*

На Факултету техничких наука, Универзитета у Новом Саду, развијају се од 80-их година 20. века методе и технике развоја информационих система засноване на платформски-независном моделовању и аутоматизованом генерисању шеме релационе базе података и функционалних прототипова информационих система. Развијене методе су реализоване кроз развојно окружење *IIS*Studio* које се састоји из три главна алата:

- *IIS*Case* – алат за пројектовање и развој информационих система,
- *IIS*UIModeler* – алат за моделовање шаблона корисничког интерфејса генерисаних прототипова информационих система и
- *IIS*Ree* – алат за реинжењеринг релационих база података у модел типова форми.

Пошто су ограничења вредности саставни део и модела и имплементације информационих система, ова ограничења се моделују кроз алат *IIS*Case*. Стога ће овај алат и модел информационог система подржан њиме бити детаљно представљени у наставку овог поглавља.

Са друге стране, ограничења вредности нису директно повезана са корисничким интерфејсом па самим тим ни са алатом *IIS*UIModeler*. Из тог разлога, детаљан опис алата *IIS*UIModeler* биће изостављен из ове дисертације, али се може пронаћи у [4].

Такође, реверзни инжењеринг ограничења вредности излази из опсега ове докторске дисертације, па се опис алата *IIS*Ree* може пронаћи у [35].

У својој првој верзији, алат *IIS*Case* је пружао могућност аутоматског генерисања шеме релационе базе података на основу спецификације функционалних зависности између атрибута универзалне шеме релације. За ту намену, коришћен је Бери-Бернштајнов алгоритам синтезе [36].

Током времена, алат је проширен увођењем великог броја нових концепата за

пројектовање и развој информационих система и шема база података. На пример, оригинални алгоритам синтезе проширен је корацима за очување спојивости без губитака добијених шема релација [37], тако што су функционалним зависностима, као улазном скупу података за алгоритам синтезе, додати нефункционални односи и специјалне функционалне зависности [37, 38]. Такође, уведена је могућност да се информациони систем моделује кроз независно моделовање подсистема који се аутоматски интегришу у јединствени систем, при чему *IIS*Case* води рачуна о валидности интеграције [3].

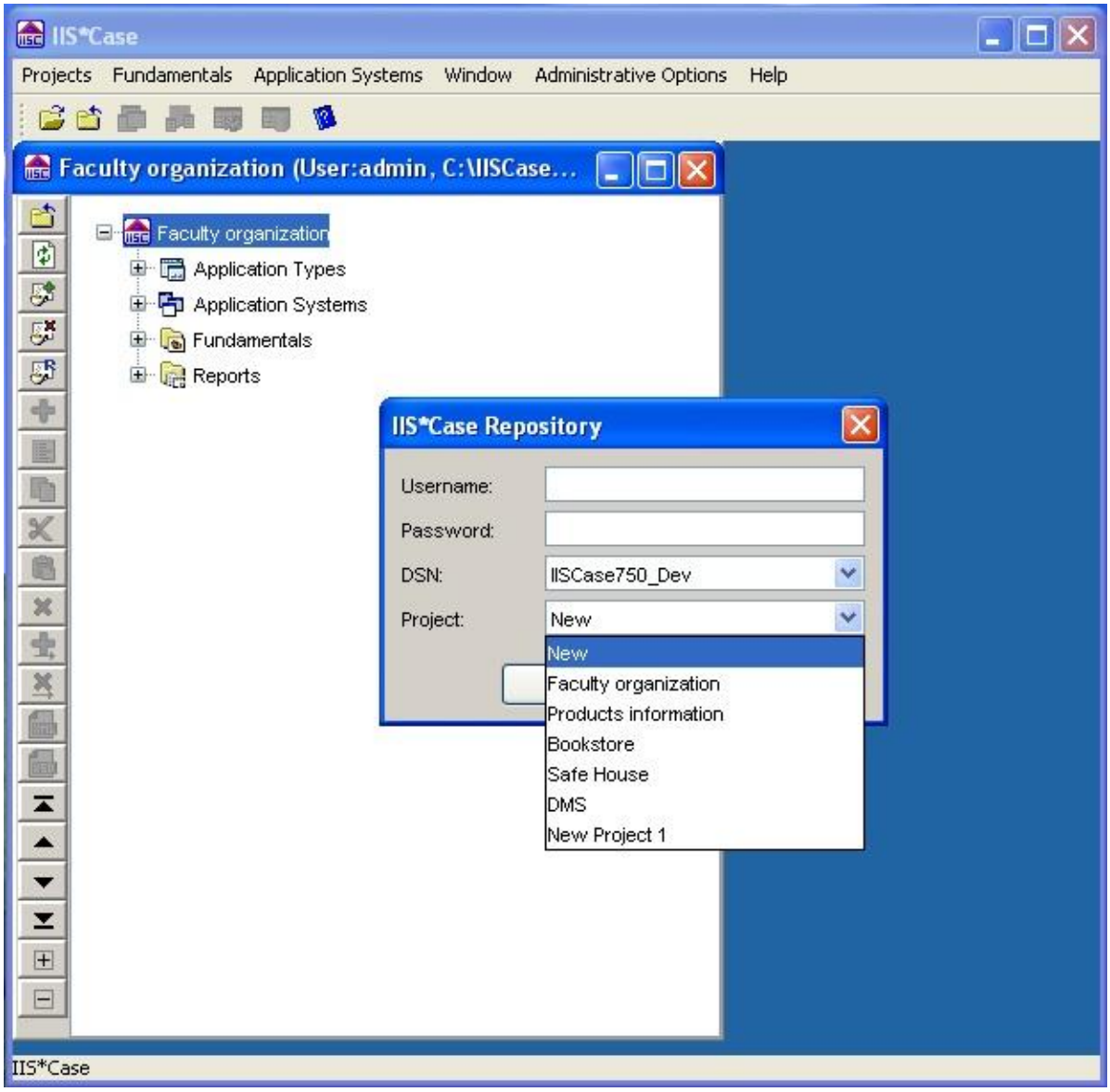
Данас, алат *IIS*Case* нуди платформски независно моделовање већине елемената информационог система и његове шеме базе података, као и аутоматизовано генерисање релационе шеме базе података и прототипа информационог система. У односу на постојеће методологије развоја софтвера, развој информационог система у *IIS*Case*-у одговара *Model Driven Software Development* (*MDSD*) приступу [39], који пропагира развој софтвера вођен моделом. У *IIS*Case*-у, платформски независни модели прво се трансформишу у платформски зависне моделе, који се даље трансформишу у извршиви код, што је у потпуности у складу са *MDSD* приступом. У погледу врста коришћених модела и трансформација, методологија развоја информационог система у *IIS*Case*-у дели исте идеје са *Model-Driven Architecture* (*MDA*) стандардом [40], као једним од најпознатијих *MDSD* приступа. Више детаља о примени *MDSD* у развојном окружењу *IIS*Case* може се наћи у [41].

Главне функционалности алата *IIS*Case* представљене су појединачно у наредним поглављима:

- концептуално моделовање шама база података, трансакционих програма и пословних апликација информационог система – поглавље 3.1,
- аутоматизовано генерисање подшеме релационе базе података у трећој нормалној форми (3НФ) – поглавље 3.2,
- аутоматизована интеграција моделованих подшема у јединствену шему релационе базе података у 3НФ – поглавље 3.3,
- аутоматско генерисање SQL/DDL кôда за јединствену шему базе података по стандарду ANSI SQL-2003 и за различите комерцијалне СУБП – поглавље 3.4 и
- аутоматизовано генерисање извршивих прототипова информационог система са основним функционалностима ажурирања и прегледа података из базе података – поглавље 3.5.

*3.1 Платформски-независни концепти модела информационог система у алату IIS*Case*

Иницијална спецификација различитих аспеката информационог система у алату *IIS*Case* у потпуности је платформски независна и кориснички оријентисана. Рад на спецификацији започиње креирањем пројекта, што је приказано на слици 3.1.

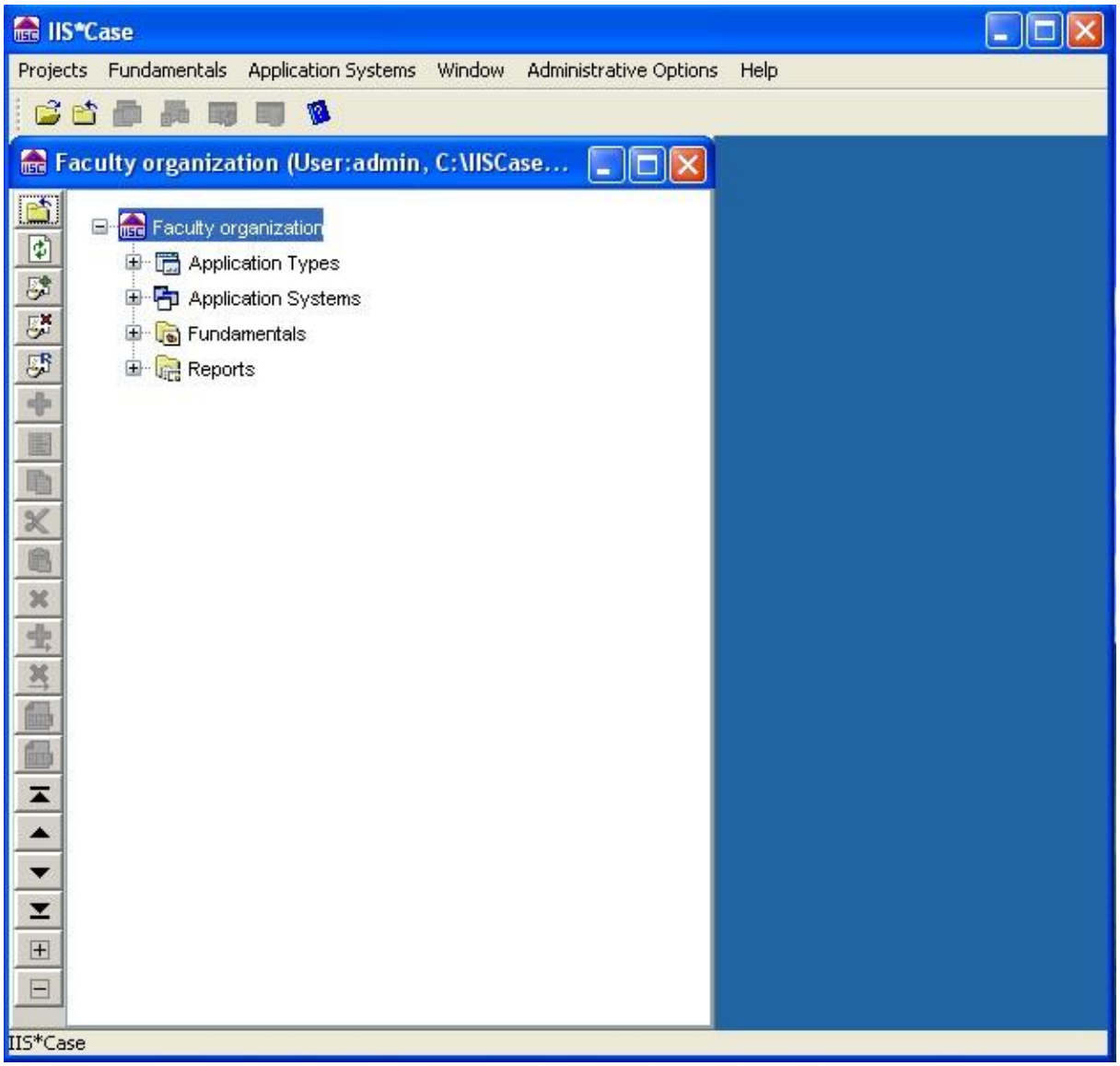


Слика 3.1. Креирање новог пројекта

У оквиру пројекта, пројектант специфицира:

- основне концепте, који се дефинишу на нивоу целог информационог система,
- типове апликативних система и
- апликативне системе, који представљају независне и функционално заокружене подсистеме информационог система.

На слици 3.2 приказан је пројекат *Faculty organization* и његови основни елементи. У следећим одељцима биће представљени основни концепти и апликативни системи.

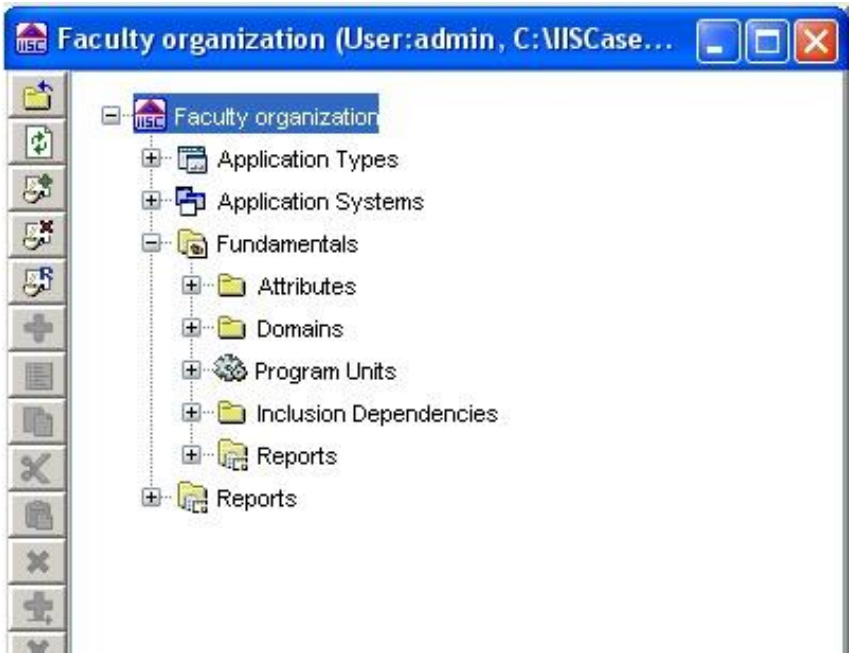


Слика 3.2: Градивни елементи спецификације пројекта

3.1.1 Основни концепти спецификације информационог система

Процес моделовања информационог система започиње спецификацијом основних концепата, који се дефинишу на нивоу целог информационог система, односно *IIS*Case* пројекта, и независни су од апликативних система. На слици 3.3, приказани основни концепти, који се налазе у стаблу пројекта испод чвора *Fundamentals*. У основне концепте спадају:

- уграђени и кориснички дефинисани домени,
- атрибути или обележја,
- пакети, функције и догађаји и
- зависности садржавања.



Слика 3.3: Основни концепти спецификације информационог система

Аналогно појму домена у другим моделима података, домен је овде такође концепт који служи за дефинисање скупа дозвољених вредности атрибута. Имплементација домена врши се путем истоименог ограничења, тј. ограничења домена. У *IIS*Case*-у, постоји скуп уграђених, тј. предифинисаних домена који по својој семантици одговарају доменима који се јављају у већини комерцијалних система за управљање базама података (СУБП). То су домени: целобројни и реални тип, датум, стринг и сл. Уколико је потребно, пројектант може прилагодити иницијални списак домена конкретном пројекту.

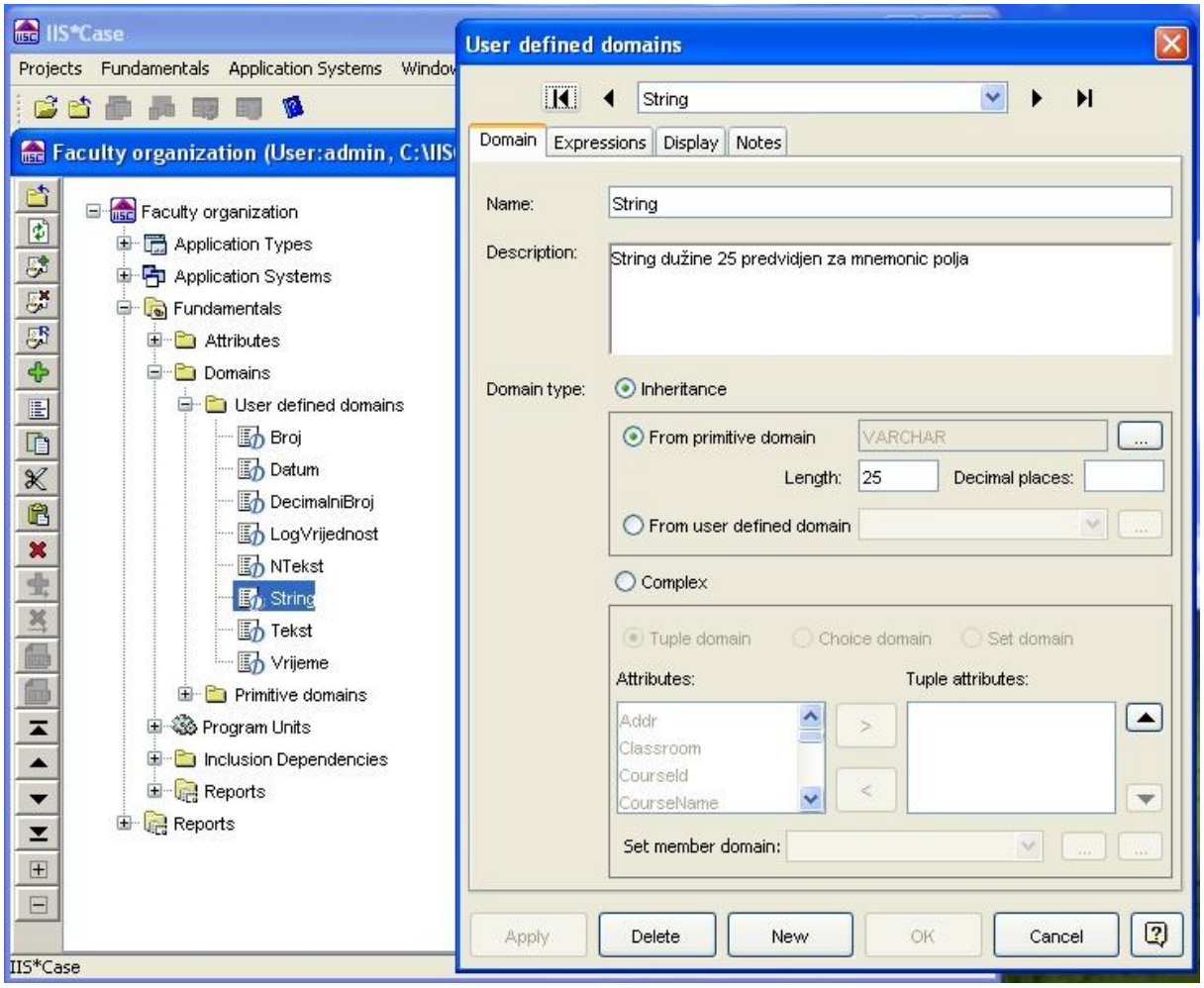
Кориснички дефинисани домени креирају се наслеђивањем постојећих домена или њиховим груписањем у сложене домене, кроз дијалог приказан на слици 3.4. У случају наслеђивања, домен преузима све особине наслеђеног домена и додаје нова ограничења. Концепт наслеђивања домена у *IIS*Case*-у аналоган је концепту наслеђивања класа у објектно-оријентисаном моделу података.

С друге стране, сложени домен може бити:

- домен типа скупа,
- домен типа торке или
- домен типа избора.

Домен типа скупа D омогућава спецификацију вредности које су, поново, скупови вредности неког изворног домена D_s. Формално, $D ::= \{S_i\}$, где је $S_i = \{a \mid a \in D_s\}$.

Домен типа торке D представља скуп уређених n-торки, где сваки члан n-торке припада неком постојећем домену D_i. Формално, домен типа торке D специфицира се као $D ::= Tuple\{(A_1: D_1),..., (A_n: D_n)\}$, где *Tuple* означава домен типа торке, A_i атрибут који је део домена, а D_i основни домен додељен атрибуту A_i.



Слика 3.4. Дијалог зе дефинисање корисничких домена

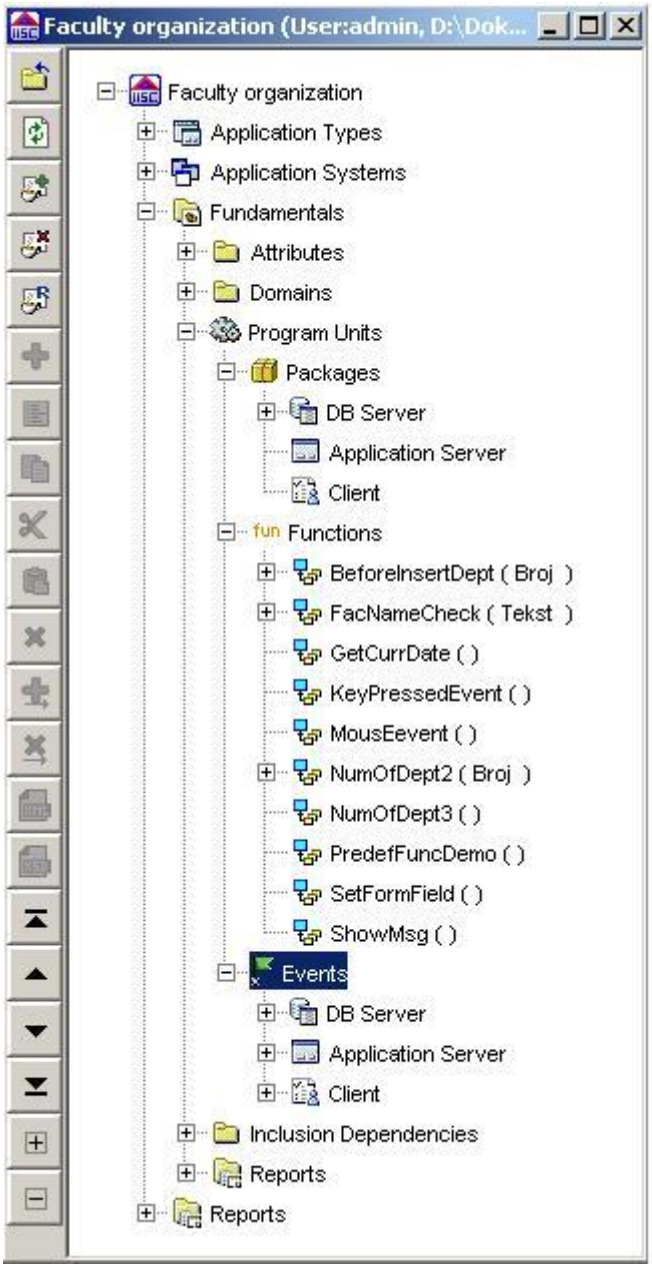
Домен типа избора D представља скуп вредности, где свака вредност припада неком постојећем домену D_i. При томе, вредности из више различитих домена могу се комбиновати у једном домену избора. Домен типа избора формално се специфицира као D ::= *Choice*$\{(A_1: D_1),..., (A_n: D_n)\}$, где *Choice* означава домен типа избора, A_i атрибут који је део домена, а D_i основни домен додељен атрибуту A_i.

Методологија пројектовања шеме базе података, подржана у алату *IIS*Case*, заснива се на претпоставци о постојању шеме универзалне релације (енг. *Universal Relation Schema Assumption*, *URSA*). Стога, име атрибута је јединствено на нивоу целог модела информационог система и атрибут се дефинише независно од елемената модела информационог система у којима ће се касније појавити. Другим речима, истоимени атрибут у различитим елементима модела информационог система носи исту семантику и поседује исту дефиницију.

Ограничење вредности може се појавити и као део спецификације домена и као део спецификација атрибута. Ови типови ограничења вредности и њихова спецификација у оквиру спецификације домена и атрибута биће детаљно представљени у наредном поглављу, док се потпуна спецификација домена и атрибута може наћи у [12].

Концепт функције користи се за спецификацију функционалности информационог система на платформски независном нивоу. При томе, функционалности могу бити, методе пословне логике система, попут одређивања следеће вредности сурогатног кључа неког типа ентитета, или комплексне обраде, попут обрачуна плата запослених.

Функције у *IIS*Case*-у дефинишу се на нивоу пројекта и могу се референцирати из спецификација осталих елемената информационог система који припадају било ком апликативном систему. На слици 3.5 приказан је део навигаторског стабла алата *IIS*Case* у којем се путем контекстног менија на чвору *Functions* могу специфицирати нове функције. Детаљна спецификација концепта функције дата је у [5, 6].



Слика 3.5: Пакети,функције и догађаји у стаблу пројекта

Са аспекта овог рада, концепт функције је битан јер се може појавити у спецификацијама ограничења вредности. Из тог разлога, трансформације модела ограничења вредности у релациони модел података и *SQL/DDL* кôд морају узети у обзир и функције.

Пакет је колекција одабраних функција, дефинисаних у оквиру једног пројекта. Овај концепт служи, слично као и у другим развојним окружењима, да омогући бољу организацију спецификација функција информационог система. У зависности од

циљног слоја у вишеслојној софтверској архитектури, пакет може бити дефинисан на нивоу:

- сервера база података (*DB server*),
- апликативног сервера (*Application Server*) или
- клијентског слоја (*Client*).

Концепт догађаја користи се на нивоу платформски независног модела, да представи било који софтверски догађај који може покренути неку акцију под специфицираних условима. По свом типу, догађаји могу бити:

- догађаји базе података, где спадају тригери и изузеци,
- догађаји апликативног сервера – догађај тастатуре, догађај миша и изузеци и
- догађаји клијента – догађај тастатуре, догађај миша и изузеци.

Догађаји се повезују са спецификацијама осталих елементата моделованог информационог система, попут дела модела који специфицира будућу шему релације или форму корисничког интерфејса. На тај начин, након трансформације модела информационог система у шему базе података и извршиви прототип информационог система, специфицирани догађаји биће додељени одговарајућим софтверским компонентама. На пример, тригер базе података треба да буде повезан с моделом шеме релације, да би, током трансформација, била успостављена веза између тригера и одговарајуће генерисане шеме релације. С друге стране, на пример, догађај тастатуре, чија спецификација је повезана са моделом елемента корисничког интерфејса, биће трансформисан у догађај на екранској форми која одговара датом моделованом елементу.

Користећи зависности садржавања, пројектант задаје ограничења универзалне шеме релације која захтевају да скуп вредности уређене торке атрибута буде подскуп скупа вредности друге уређене торке атрибута. Овај тип ограничења биће трансформисан у одговарајуће међурелационо ограничење у генерисаној релационој шеми базе података. Преименовани атрибути у алату *IIS*Case* представљају специјалне случајеве зависности садржавања и резултују, такође, у међурелационим ограничењима као и сложенији облици зависности садржавања. Детаљнији опис концепта зависности садржавања дат је у [12].

3.1.2 Апликативни системи и типови форми

Основна градивна јединица пројекта је апликативни систем који представља функционално заокружен подсистем информационог система. На пример, у информационом систему електро дистрибуције, апликативни системи могу бити систем енергетских функција, систем наплате, систем материјалних ресурса, итд. При томе, апликативни системи могу бити позиционирани на истом нивоу хијерархије, или се налазити у односу ,,систем-подсистем''.

Основни концепт платформски независног модела апликативног система, а самим тим и приступа моделовања подржаног алатом *IIS*Case*, представља тип форме [3, 6]. Тип форме представља апстракцију екранске форме или документа у електронском или папирном облику, путем којих корисник комуницира са информационим системом, односно ажурира или прегледа податке. Користећи овај концепт, пројектант директно

специфицира екранске форме и извештаје информационог система, док посредно моделује шему базе података и основне трансакционе програме за унос измену и брисање над базом података. Стога, пројектант не мора да поседује напредна знања из области пројектовања база података. Пројектантов задатак је да што прецизније и детаљније специфицира податке информационог система и начине њихове употребе, а *IIS*Case* даље аутоматски трансформише спецификације у имплементациони опис шеме базе података са одговарајучим ограничењима.

Формално, тип форме се дефинише као именована структура типа стабла чији чворови представљају типове компоненти, где тип компоненте моделује скуп појава једног типа ентитета информационог система. У сваком типу форме, постоји један коренски тип компоненте. Остали типови компоненте су њему подређени и формирају структуру типа стабла. Кардиналитет појаве подређеног типа компоненте у оквиру појаве надређеног типа компоненте може битли *0-N* или *1-N*. У имплементационој шеми базе података, веза између надређеног и подређеног типа компоненте биће представљена страним кључем, а кардиналитет *1-N* иницираће креирање додатног ограничења да мора постојати барем једна торка у релацији која одговара шеми релације подређеног типа компоненте, за сваку торку релације која одговара шеми релације надређеног типа компоненте.

Сваки тип компоненте је дефинисан у опсегу типа форме и пројектант за њега дефинише следеће податке:

- име,
- надређени тип компоненте,
- дозвољене операције над подацима,
- кардиналност у оквиру надређеног типа компоненте,
- скуп атрибута,
- скуп кључева,
- атрибуте за визуализацију и
- групе поља.

Атрибути типа компоненте припадају универзалном скупу атрибута, односно атрибути се не дефинишу за сваки тип компоненте посебно. Уз придуживање атрибута типу компоненте, пројектант дефинише и скуп дозвољених акција над тим атрибутом. Један атрибут може бити придружен само једном типу компоненте у оквиру истог типа форме. Са друге стране, исти атрибут може бити додељен различитим типовима форми.

Путем дијалога за спецификацију типа форме, приказаног на слици 3.6, у вези са шемом базе података пројектант специфицира:

- функционалне зависности између обележја пројектованог система,
- ограничења кључа,
- ограничења јединствености вредности,
- ограничења недостајућих (NULL) вредности и
- ограничења вредности на нивоу типова компоненти.

Ове спецификације ограничења представљају улазне податке за алгоритам

имплементиран у алату *IIS*Case* који као излаз даје имплементациону шему релационе базе података, у трећој нормалној форми (3НФ), са свим специфицираним ограничењима израженим путем одговарајућих концепата релационог модела података.

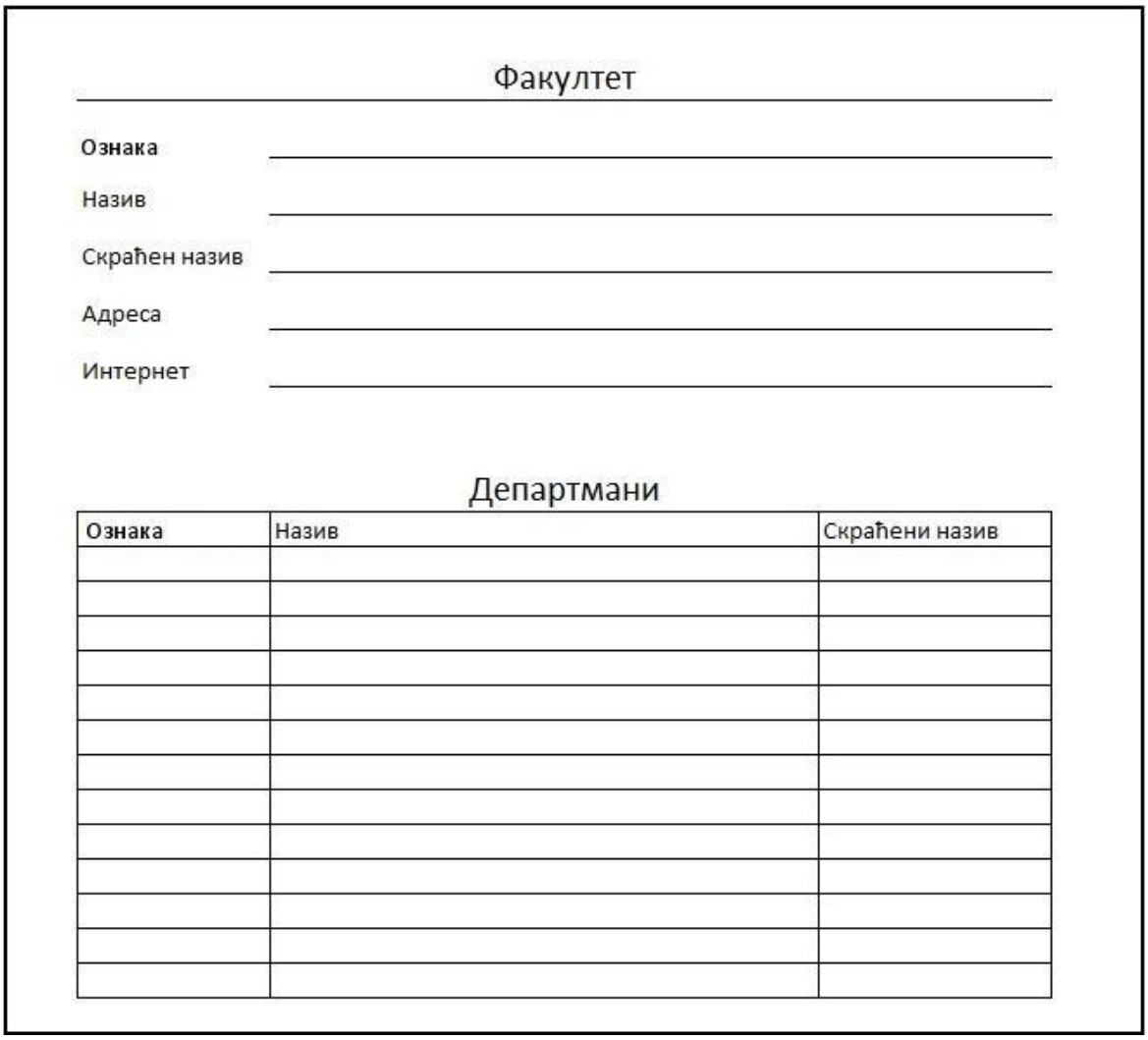
Слика 3.6: Форма за дефинисање типова форми

За сваки тип компоненте, могу се дефинисати ограничења кључа и ограничења јединствености вредности. Кључ представља непразан скуп атрибута који јединствено идентификује појаву тог типа компоненте у оквиру надређеног типа компоненте. За сваки тип компоненте, пројектант мора дефинисати бар један кључ, а могуће је да тип компоненте има и више кључева. За коренски тип компоненте, кључ представља само онај скуп обележја који је дефинисао пројектант. С друге стране, кључ подређеног типа компоненте представља унију по једног кључа из сваког надређеног типа компоненте и посматраног типа компоненте.

Ограничење јединствене вредности састоји се из непразног скупа обележја. Свака торка са задатим конкретним вредностима тог скупа обележја мора бити јединствена у скупу појава типа компоненте. Пројектант може дефинисати произвољан број ограничења јединствене вредности у типу компоненте.

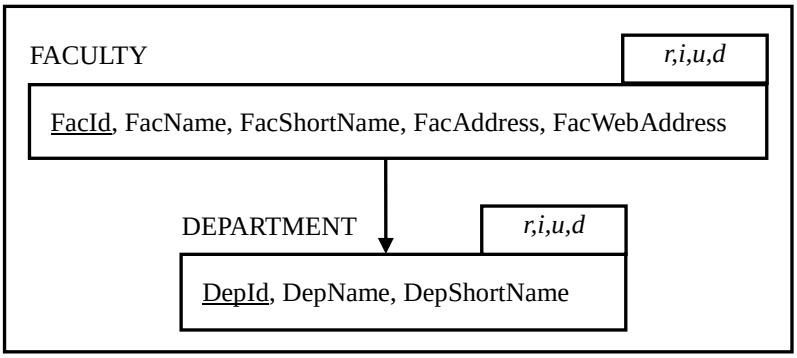
Пример 3.1. У овом примеру илустрован је пројекат типова форми и компоненти на основу документа из реалног система.

Слика 3.7 приказује папирни или електронски документ који служи за евиденцију факултета и његових департмана. У горњем делу документа уносе се информације о факултету, док се испод уносе подаци о припадајућим департманима.



Слика 3.7: Документ који садржи информације о факултетима

На основу овог документа, пројектант дефинише тип форме дат на слици 3.8. Тип форме садржи два типа компоненте *FACULTY*[1] и *DEPARTMENT*. Тип компоненте *FACULTY* је надређен типу компоненте *DEPARTMENT*.



Слика 3.8. Спецификација типа форме

[1] Приликом трансформације модела типова форми у релациони модел базе података и даље у *SQL/DDL* кôд, називи шема релација, обележја, табела и колона граде се од назива типове компоненти и атрибута. Из тог разлога, називи типова форми, типова компоенети и атрибута дати су на енглеском језику, како би називи резултујућих табела били у складу са програмским језиком.

Кључеви типова компоненти су подвучени. Пошто је тип компоненте *FACULTY* коренски, атрибут *FacId* јединствено идентификује сваку појаву овог типа компоненте. Са друге стране, појава типа компоненте *DEPARTMENT* је јединствено идентификована паром атрибута *FacId* и *DeptId*. □

3.2 Генерисање релационе шеме базе података

Путем платформски независног модела у *IIS*Case*-у, пројектант специфицира следеће типове ограничења:

- функционалне зависности,
- нефункционалне зависности,
- зависности садржавања,
- ограничења јединствене вредности, и функционалне зависности које произилазе из њих,
- ограничења домена,
- ограничења NULL вредности и
- ограничења вредности.

Формалне дефиниције наведених типова ограничења се могу наћи у [6, 12, 37, 8].

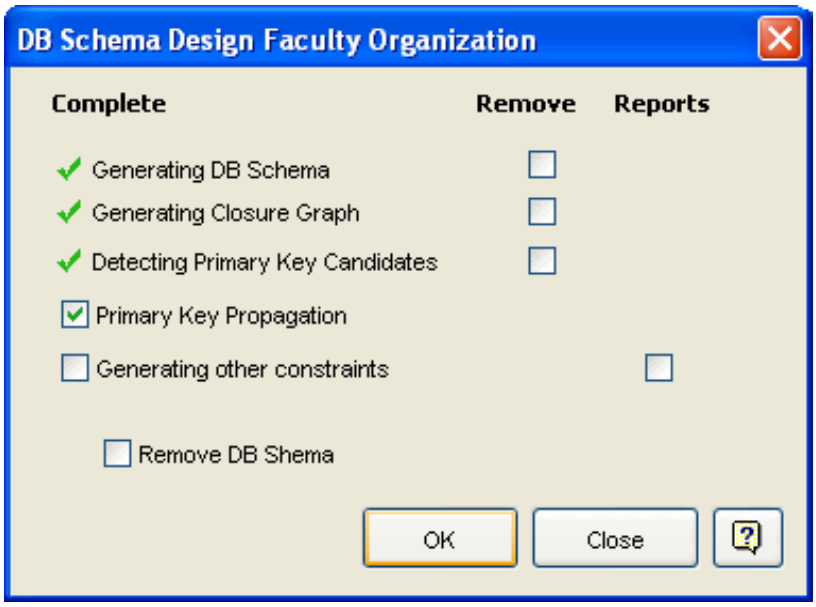
Спецификације ограничења представљају улазне податке за процес генерисања имплементационе шеме релационе базе података. Основни кораци процеса су:

1. трансформација скупа функционалних и нефункционалних зависности у скуп шема релација,
2. конструкције графа затварања,
3. одређивање кандидата за примарне кључеве,
4. пропагација примарних кључева, и
5. генерисање осталих типова ограничења у шеми релационе базе података.

Детаљна спецификација алгоритма и његових корака може се наћи у [3, 6, 12, 13].

Према *Model-Driven Architecture* (*MDA*) приступу [40], процес генерисања релационе шеме базе података представља трансформацију платформски независног модела (енг. *platform-independent model*, *PIM*) у платформски зависан модел (енг. *platform-specific model*, *PSM*). У случају *IIS*Case*-а, платформски независан модел типова форми транформише се у релациону шему базе података. У даљем тексту, ова трансформација се скраћено назива *PIM/PSM* трансформација.

Извођењем корака генерисања релационе шеме базе података управља пројектант користећи форму приказану на слици 3.9. Сви кораци су обавезни и морају се извршити у задатом редоследу.



Слика 3.9. Форма за извршавање корака у генерисању шеме базе података

Први корак представља извршавање алгоритма синтезе [42], који је проширен додатним корацима за очувавање спојивости без губитака [37]. Улазни подаци алгоритма синтезе представљају скупови функционалних, нефункционалних и специјалних зависности који су специфицирани у типовима форми, док је резултат скуп шема релација. Другим речима, алгоритам синтезе трансформише модел базе података заснован на типовима форми у релациони модел базе података.

Други корак генерише граф затварања који омогућава графичку, визуелну репрезентацију генерисане шеме базе података. Чвор графа затварања представља генерисану шему релације док свака грана означава да скуп обележја надређеног чвора функционално одређује, тј. затвара, скуп обележја подређеног чвора.

У трећем кораку се из скупа еквивалентних кључева сваке шеме релације аутоматски бирају кандидати за примарни кључ.

У четвртом кораку, пројектант проглашава један од кандидата кључева за примарни кључ шеме релације. *IIS*Case* даље аутоматски пропагира тај кључ као страни у све директно подређене шеме релација.

Кроз пети корак, *IIS*Case* генерише следеће типове ограничења:

- ограничења основних и проширених референцијалних интегритета,
- ограничења референцијалних интегритета заснованих на нетривијалним зависностима садржавања,
- ограничења инверзних референцијалних интегритета и
- ограничења инверзних референцијалних интегритета заснованих на нетривијалним зависностима садржавања.

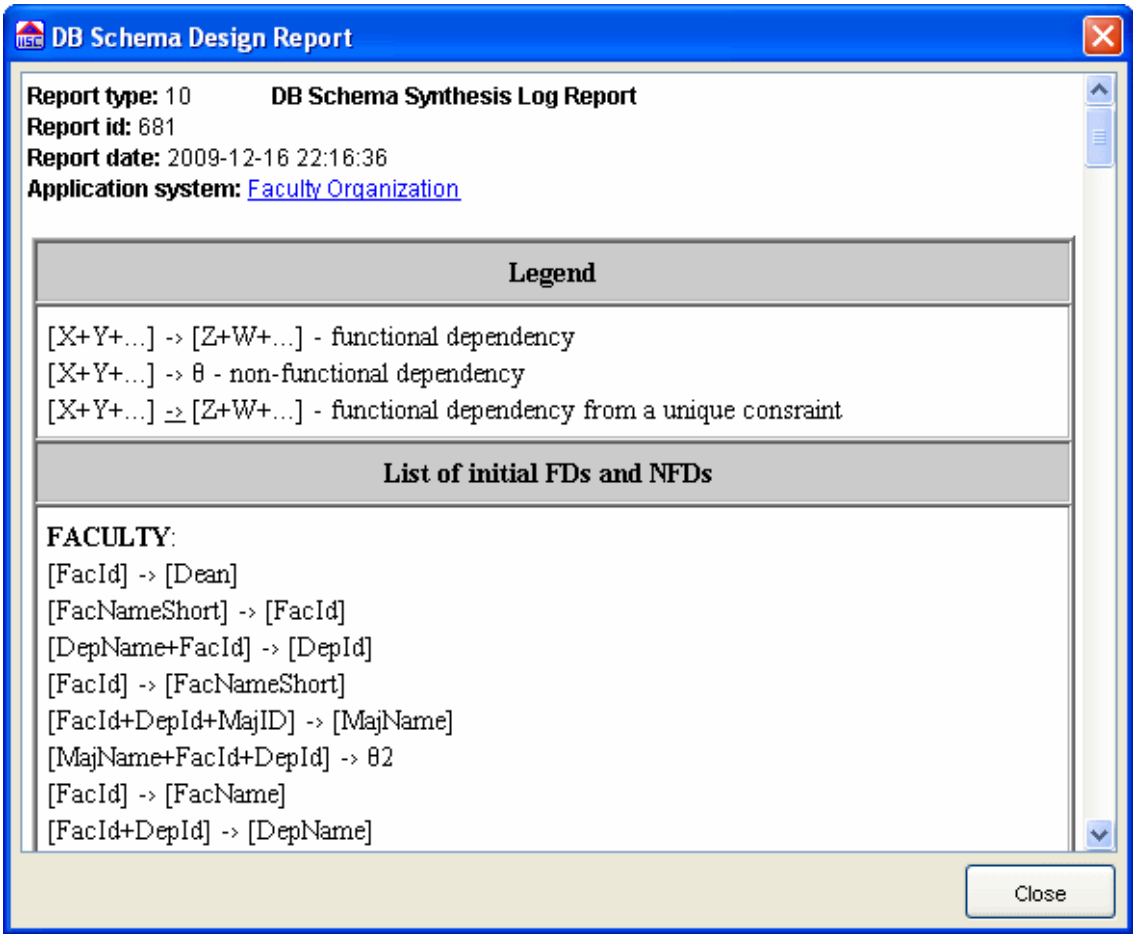
Основна и проширена ограничења референцијалног интегритета су последица пропагације примарног кључа. При том, ако је цео примарни кључ пренет из референциране шеме релације у референцирајућу шему релације, *IIS*Case* генерише ограничење основног референцијалног интегритета. У супротном случају, генерише се ограничење проширеног референцијалног интегритета.

Нетривијалне зависности садржавања дефинише пројектант између обележја информационог система, на основу њихове семантике и употребе у оквиру информационог система. Генерисана ограничења користе исти механизам страног кључа као и у случајевима основних и проширених ограничења референцијалних интегритета.

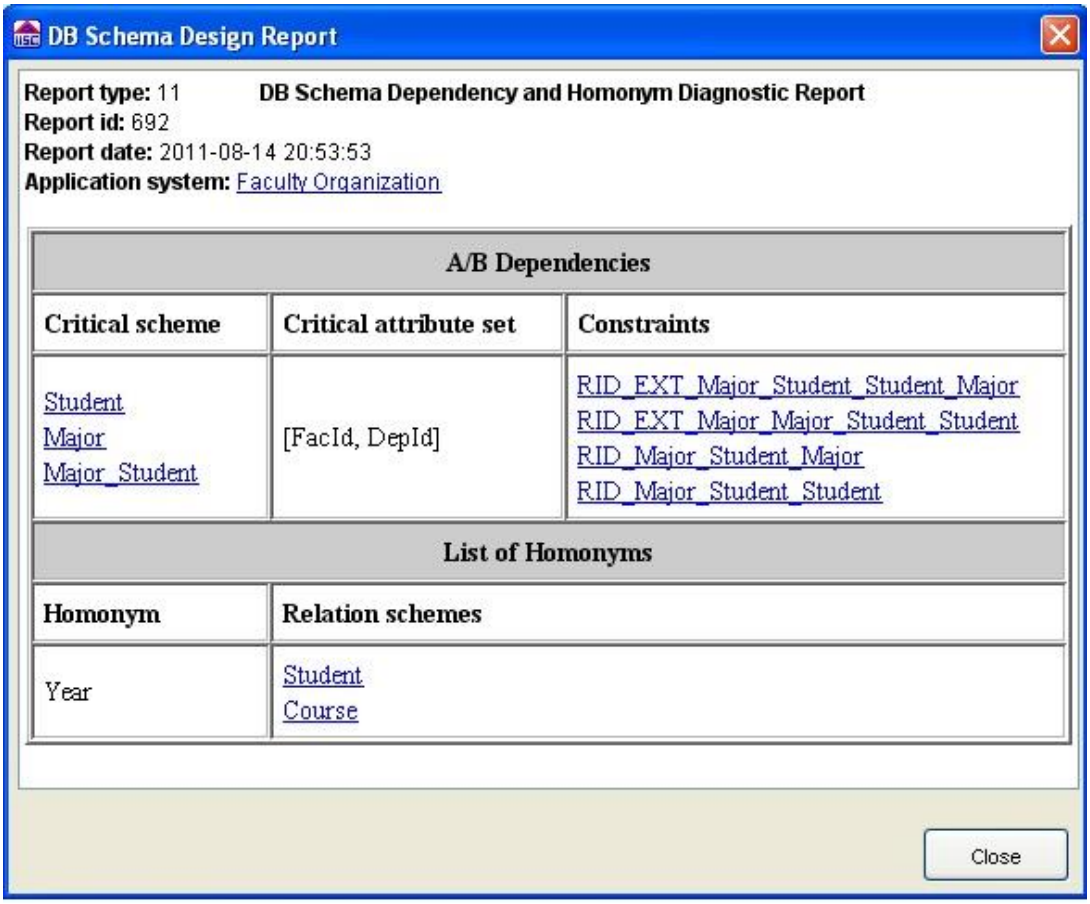
Ограничење инверзног референцијалног интегритета настаје ако свака појава надређеног типа компоненте мора да има макар једну појаву подређеног типа компоненте. Овакво ограничење специфицира пројектант у склопу једног типа форме.

*IIS*Case* такође детектује хомониме и *A* и *B*-зависности шема релација у истом кораку и генерише одговарајућа проширена ограничења референцијалног интегритета [3, 43].

*IIS*Case* аутоматски генерише извештаје након првог и петог корака генерисања шеме релационе базе података. Извештај након првог корака (слика 3.10) садржи информације о коришћеним функционалним зависностима и генерисаним шемама релација. Извештај након петог корака, (слика 3.11) садржи списак идентификованих хомонима и *A* и *B*-зависности.



Слика 3.10. Извештај о генерисаним шемама релација



Слика 3.11. Извештај о хомонима и A/B-зависностима

У овој докторској дисертацији, развијена је *PIM/PSM* трансформација која платформски независан модел ограничења вредности трансформише у модел ограничења вредности у релационом моделу података. Ова трансформација имплементирана је као проширење петог корака генерисања релационе шеме базе података у *IIS*Case*–у. *PIM/PSM* трансформација модела ограничења вредности детаљно је описана у петом поглављу.

Кроз наведене кораке, *IIS*Case* трансформише скуп типова форми моделованог апликативног система у шему релационе базе података апликативног система. Ако је посматрани апликативни систем уједно и подсистем другог апликативног система, генерисана шема је заправо подшема имплементационе шеме надређеног апликативног система.

3.3 Консолидација генерисаних подшема базе података

Моделовање информационог система, укључујући и шему базе података, може да се спроводити на два начина:

- директно, где се цео информациони систем моделује одједном и
- део-по-део, тј. инкрементално, где различити пројектанти независно моделују делове система, који се накнадно интегришу у јединствени систем.

У ранијим истраживањима [9, 3], тврди се да директни развој информационих система, поготово високо сложених, може резултовати моделом са много грешака и недостатака услед људског фактора, односно немогућности лаке и прецизне перцепције велике количине детаља у границама само једног, али по правилу великог креираног модела система. Моделовање целог информационог система одједном захтева од пројектанта да посвети велику и једнаку пажњу свим деловима система, што углавном превазилази људске способности.

Из тог разлога, на Факултету техничких наука у Новом Саду, развијена је методологија [11, 3, 9] за постепено моделовање информационих система и њихових база података. Ова методологија подржана је од стране алата *IIS*Case* који моделоване подсистеме аутоматски интегрише у јединствени систем и води рачуна о усаглашености подсистема са системом.

Модел информационог система у *IIS*Case*-у представља структуру типа стабла, где чворови стабла представљају подсистеме, који се у *IIS*Case* терминологији називају апликативним системима. Сваком апликативном систему одговара једна шема базе података, која представља подшему интегрисане шеме базе података целог информационог система. Како су ограничења торке предмет истраживања ове дисертације, у наставку поглавља посебна пажња посвећена је провери консолидованости подшема база података.

Интегрисана шема базе података не може бити добијена простом унијом подшема, јер би то изазвало колизије у исказивању семантике и фаворизацију редундантности података. Због тога, интегрисана шема добија се трансформацијом уније типова форми из свих апликативних система, која је описана у претходном поглављу. Због краћег писања, интегрисана шема базе података ће надаље у тексту бити називана једноставно шема базе података, осим на местима где је неопходно нагласити о којој се шеми ради.

Приликом имплементације информационог система, имплементира се само шема базе података, док подшеме представљају погледе кроз које корисник види и ажурира податке. На основу подшема имплементира се и кориснички интерфејс. Стога, да би се обезбедило исправно функционисање информационог система, свака подшема базе података мора бити формално конзистентна са шемом базе података целог информационог система [9]. Другим речима, сваки елемент подшеме мора бити усклађен са одговарајућим елементом шеме базе података. При том, елементи су шеме релација и сви типови ограничења, односно, они елементи који нису специфицирани на нивоу целог информационог система.

Посебни услови усаглашености подшеме са шемом базе података су [9]:

- за сваку шему релације подшеме, мора постојати шема релације интегрисане шеме, таква да је скуп обележја шеме релације подшеме подскуп скупа обележја шеме релације интегрисане шеме и примарни кључ шеме релације подшеме једнак је примарном кључу шеме релације интегрисане шеме; и
- за свако ограничење из интегрисане шеме, постоји једнако или строжије ограничење у подшеми ако подшема моделује податке на које се односи ограничење из интегрисане шеме.

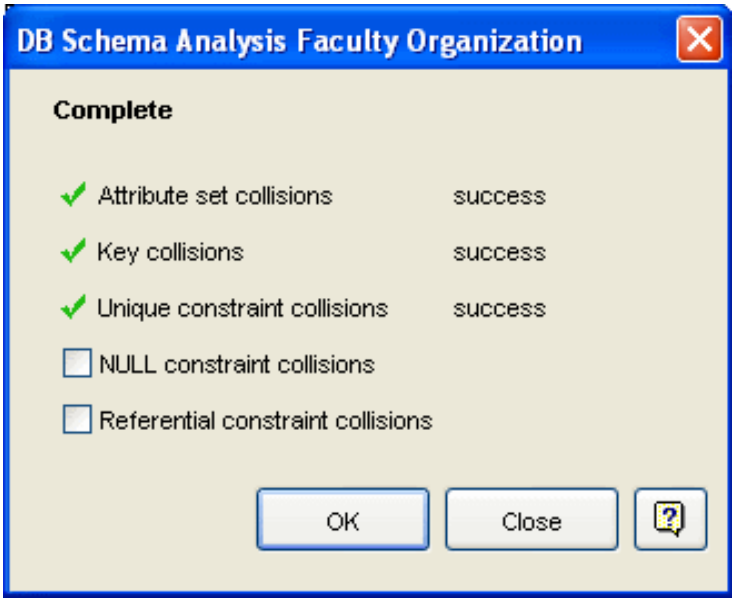
Након сваког генерисања подшеме и шеме базе података из скупова типова форми, пројектант има могућност да провери консолидованост шеме са подшемама. Ако је консолидованост задовољена, генерисана интегрисана шема представља имплементациону шему базе података моделованог информационог система.

Алгоритми за консолидацију подшема са интегрисаном шемом базе података детаљно су изложени и математички формално дефинисани у [9].

Провера консолидованости подшема са интегрисаном шемом базе података одвија се независно по концептима који чине базу података. Кораци провере су:

1. усаглашеност обележја шема релација подшеме и интегрисане шеме базе података,
2. усаглашеност скупова ограничења јединствене вредности обележја,
3. усаглашеност скупова ограничења кључа,
4. усаглашеност скупова ограничења *NULL* вредности обележја и
5. усаглашеност скупова ограничења референцијалних интегритета.

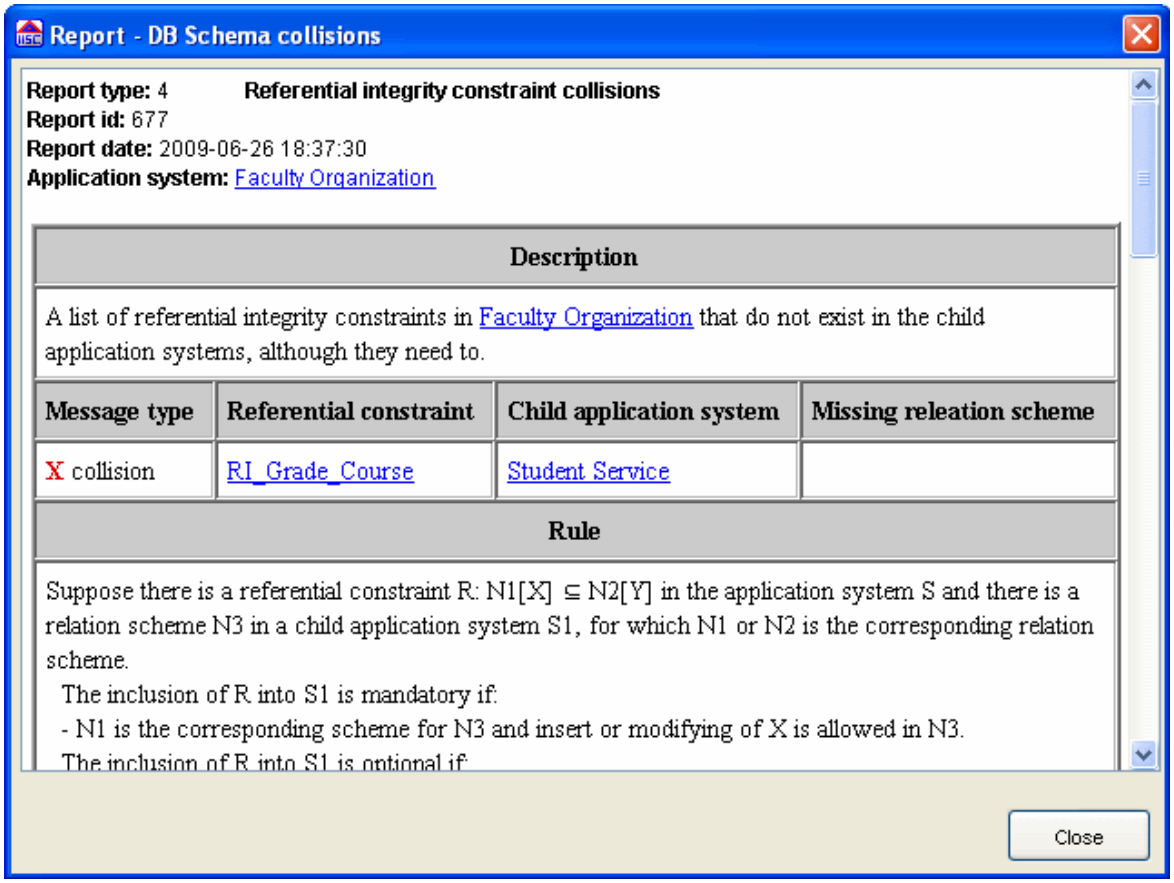
Кораци провере консолидованости морају се изводити у задатом редоследу јер консолидованост једног типа ограничења зависи од консолидованости типова ограничења која се претходно проверавају. На слици 3.12 приказан је дијалог за извршавање корака алгоритма консолидације.



Слика 3.12. Дијалог за прошеру усаглашености подшема базе података

Током провере консолидованости, *IIS*Case* генерише извештаје у којима се детаљно описују разлози неусаглашености подшема. У случајевима када то има смисла, такође предлаже како да се пронађени конфликти реше. У *IIS*Case*-у, извештајима се може приступити кроз стабло пројекта у ком се налази посматрани апликативни систем. На слици 3.13, дат је пример извештаја неусаглашености скупова ограничења референцијалних интегритета.

Процес пројектовања шеме базе података је итеративан. Свака итерација се састоји из поновног моделовања делова информационог система кроз измену типова форми, генерисање подшема и шеме базе података и покретање алгоритма консолидације. Наведени процес се понавља све док се свака подшема не буде формално консолидована са интегрисаном шемом базе података.



Слика 3.13. Извештај о конфликтима ограничења референцијалног интегритета

Један од важних резултата ове докторске дисертације представља алгоритам за проверу консолидованости подшеме са интегрисаном шемом базе података са аспекта ограничења вредности. Дефинисани алгоритам описан је детаљно у седмом поглављу и представља проширење провере консолидованости описане у овом одељку.

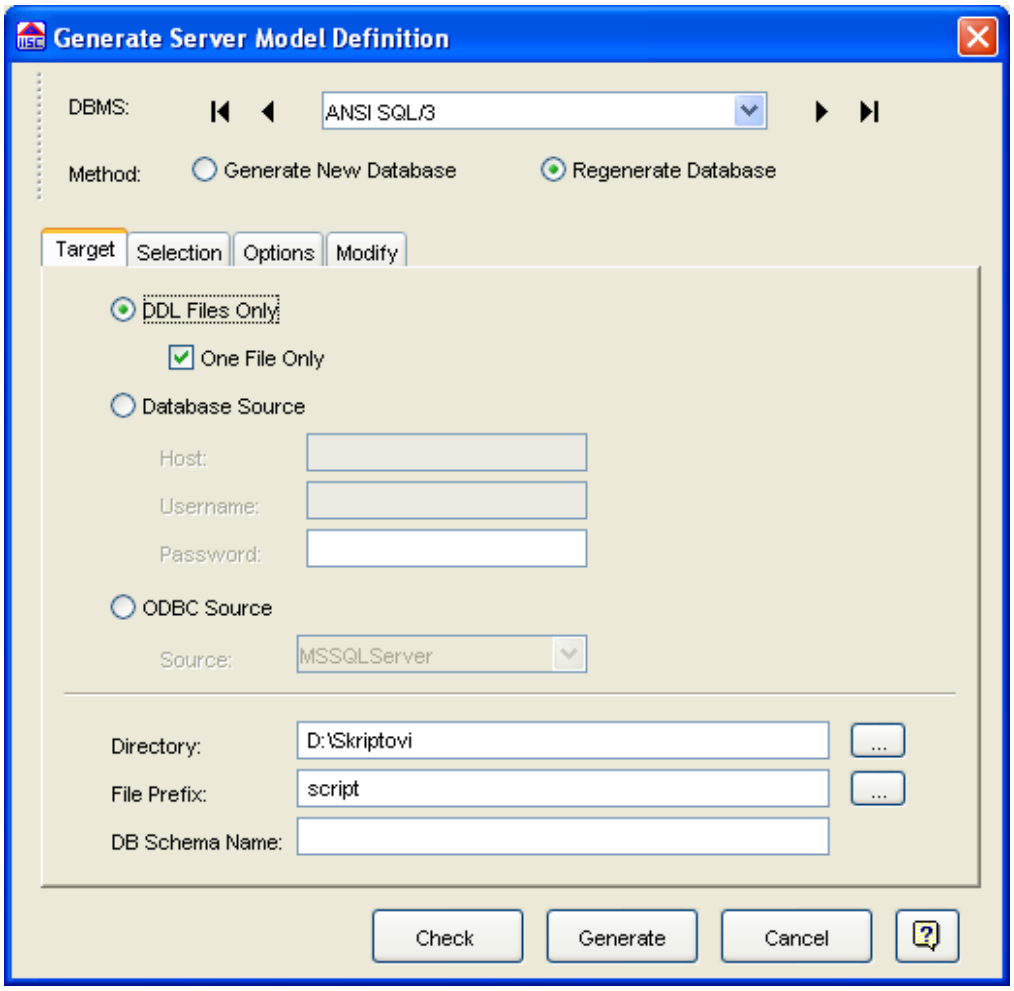
3.4 Генерисање имплементационог описа шеме базе података

У алат *IIS*Case* уграђен је *SQL* генератор који трансформише релациону шему базе података, добијену претходно из модела типова форми, у извршиви *SQL* кôд. Тиме је пројектанту омогућено да без познавања синтаксе *SQL*-а и особина система за управљање базом података, добије *SQL* код и реализује функционалну релациону базу података.

У тренутној верзији, *SQL* генератор омогућава генерисање имплементационог описа шеме базе података по стандарду *ANSI SQL 2003* и *ANSI SQL 2011*, и за комерцијалне СУБП *MS SQL Server* и *Oracle*, односно према синтакси језика *Microsoft T-SQL* и *Oracle PL/SQL*, редом.

SQL генератор се покреће кроз посебан дијалог (слика 3.14) у *IIS*Case*-у. Кроз исти дијалог, корисник може да зада параметре генерисања *SQL* кода, попут локације на којој ће се сместити фајлови са извршивим кодом, адресе сервера на коме је потребно

креирати базу, облик окидача и ограничења вредности и сл. Такође, пројектанту је омогућено и да мења редослед атрибута, као и да мења ограничења примарних кључева и ограничења јединствене вредности.



Слика 3.14. Дијалог за покретање SQL генератора

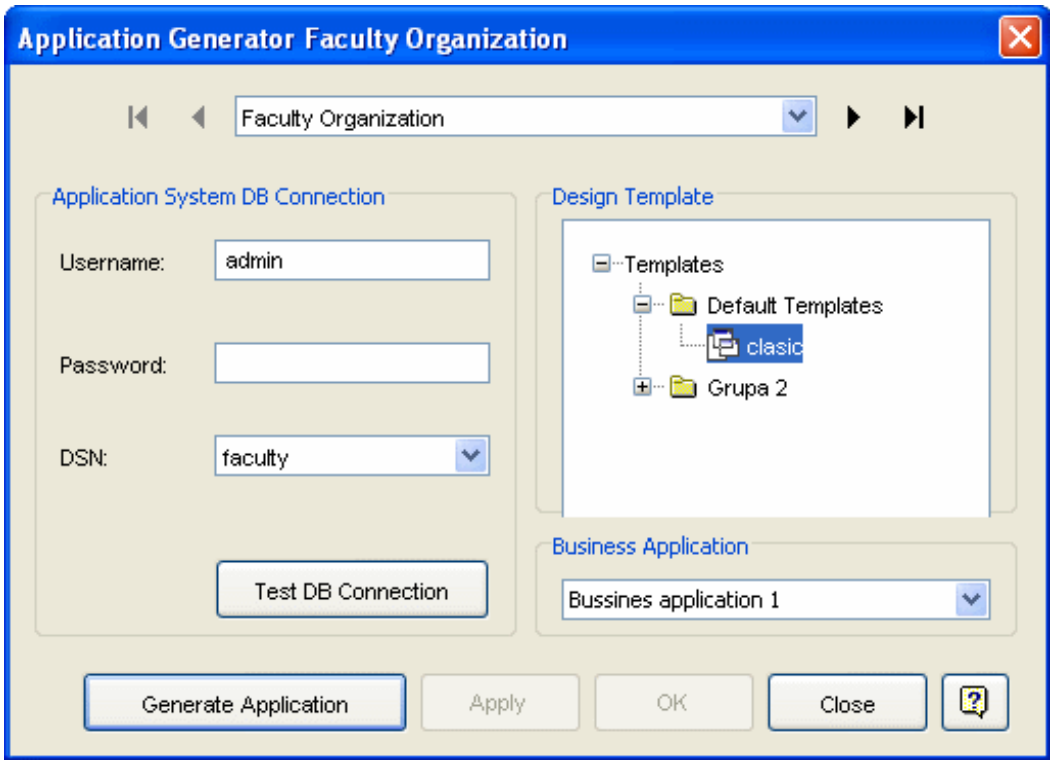
SQL генератор заокружује процес пројектовања шеме базе података информационог система. Детаљан опис генератора и алгоритама и механизама коришћених за генерисање *SQL* кôда се може наћи у [44].

У шестом поглављу ове дисертације представљене су развијене модел-у-кôд трансформације за генерисање *SQL* кôда за ограничења вредности. Ове трансформације имплементиране су као проширење постојећег *SQL* генератора.

3.5 Генерисање прототипа информационог система

Након завршеног генерисања и успешне консолидације шеме базе података, пројектант има могућност да изгенерише прототипове изабраних делова моделованог информационог система. Генерисање прототипа пројектант покреће путем дијалога (слика 3.15) кроз који је потребно да зада следеће параметре:

- параметре за приступ серверу базе података,
- изабрани шаблон корисничког интерфејса који ће се користити за креирање апликације и
- пословну апликацију за коју се генерише прототип.



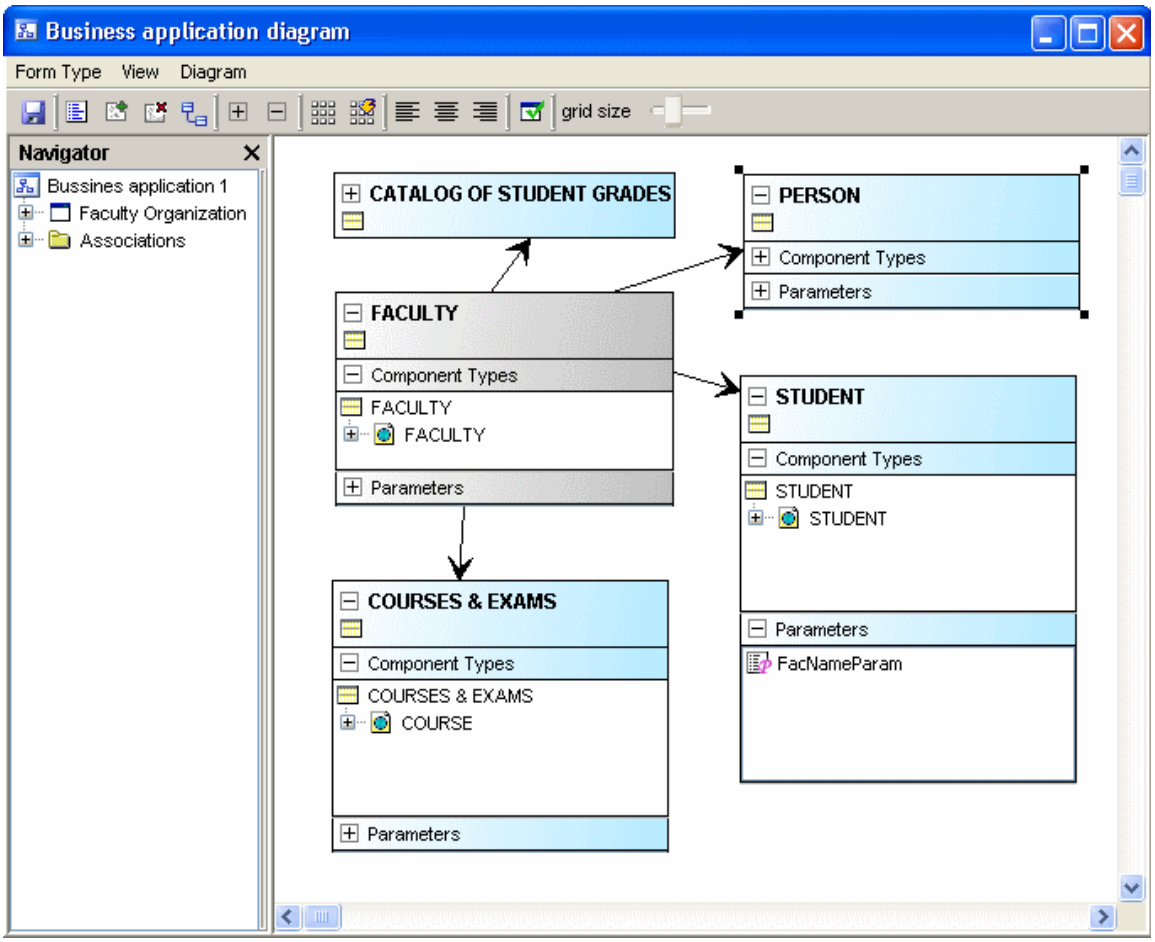
Слика 3.15. Дијалог за генерисање прототипа апликације

Концепт пословне апликације уведен је и детаљно представљен у [45]. Спецификацијом пословних апликација, пројектант формално описује основне функционалности информационог система, које се односе на узајамне позиве екранских форми које су изгенерисане из различитих типова форми. Пословна апликација дефинише се на нивоу апликативног система.

Поред скупа типова форми, спецификација пословне апликације укључује и модел веза које описују како екранске форме међусобно комуницирају. У ту сврху, у оквиру спецификације пословне апликације дефинише се концепт који се назива позивајућа структура. Једна позивајућа структура представља уређени пар типова форми који учествују у пословној апликацији. У оквиру позивајуће структуре, екранска форма која одговара првом елементу пара позива форму која одговара другом а позивајућа структура садржи и следеће спецификације:

- прослеђене вредности,
- начин позива,
- метод позива и
- позиционирање корисничког интерфејса.

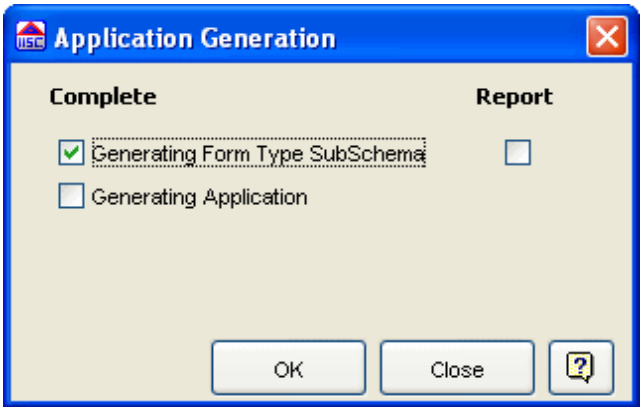
На слици 3.16 дат је пример спецификације пословне апликације приказан у оквиру алата за графичко моделовање пословних апликација, који се назива *Business Application Designer*.



Слика 3.16. Business Application Designer

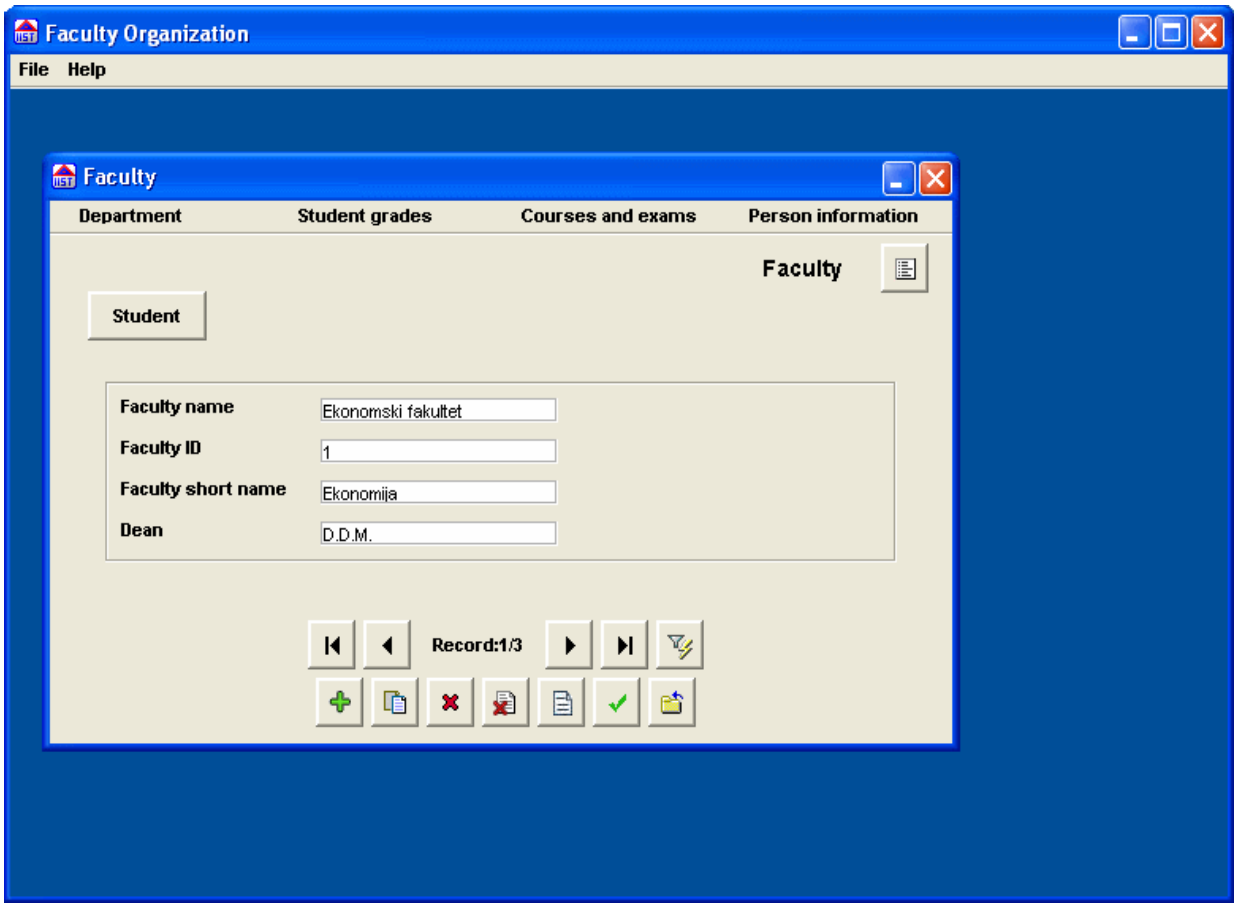
Процес генерисања трансакционог програма састоји се из два корака, који се покрећу путем дијалога са слике 3.17:

- генерисање подшеме типова форми,
- генерисање апликације, односно њеног Јава дела.



Слика 3.17. Покретање генерисања пословне апликације

На слици 3.18, дат је пример аутоматски изгенерисаног прототипа.



Слика 3.18. Пример изгенерисаног прототипа апликације

3.6 Закључак

У трећем поглављу, представљени су основни концепти развоја информационог система у окружењу *IIS*Case*:

- платформски независан модел информационог система, који се специфицира

путем модела типова форми,

- трансформација модела типова форми у релациони модел података, чиме се обезбеђује генерисање релационог модела имплементационе шеме базе података,
- консолидација подшема база података са јединственом, имплементационом шемом базе података,
- генерисање извршивог *SQL/DDL* кôда имплементационе шеме базе података и
- генерисање прототипова моделованог информационог система.

Кроз представљање овог приступа, приказан је уједно и контекст у оквиру кога је реализован задатак ове дисертације: креирање кориснички оријентисаног приступа за развој ограничења вредности. У наредним поглављима представљена су проширења прва четири наведена концепта са аспекта ограничења вредности.

Поступак генерисања извршивих прототипова информационог система није у директној вези са развојем ограничења вредности јер су она у потпуности реализована у имплементационој шеми базе података. Овај поступак описан је у овом поглављу како би читалац овог рада стекао заокружену слику о окружењу *IIS*Case*.

4 Моделовање ограничења вредности путем платформски независних модела

У *IIS*Case*-у, модел информационог система, који специфицира пројектант, представља платформски независан модел, односно модел који не зависи ни од једне конкретне имплементационе технологије или платформе. У моделу се специфицирају различити концепти присутни у информационим системима и базама података, попут домена, атрибута, ограничења, кориснички дефинисаних функција и процедура, догађаја, екранских форми и сл. Формалне дефиниције платформски независних концепата могу се наћи у [3, 4, 5, 6, 7, 8].

У складу са тим, у *IIS*Case*-у могуће је креирати и платформски независне моделе ограничења вредности домена, ограничења вредности атрибута и ограничења вредности торки, који се у моделу типова форми специфицирају на нивоу домена, атрибута и типа компоненте, редом. Наведени модели су представљени и дефинисани у [8]. У наставку поглавља описани су модели сва три типа ограничења. Ови модели представљају полазну основу за развој ограничења вредности, као и улазну спецификацију *PIM/PSM* трансформације ограничења вредности, која је описана у наредном поглављу.

Сваком дефинисаном атрибуту у *IIS*Case*-у, пројектант мора доделити ограничење домена. Осим што дефинише тип податка, спецификација ограничења домена може да садржи и спецификацију ограничења вредности домена. Ограничење вредности домена представља логички израз над уграђеном промењивом *VALUE* која представља дозвољену вредност домена. У *IIS*Case*-у, ограничење вредности домена се дефинише употребом наменског језика (енг. *domain specific language, DSL*) који је уведен и формално дефинисан у [8, 5].

Пример 4.1. Ограничење вредности, задато на нивоу домена, које дозвољава само вредности веће од 7, дефинише се као

$$VALUE > 7. \square$$

У случају наслеђивања домена, домен наследник преузима ограничење вредности

наслеђеног домена. Ако домен наследник има дефинисано сопствено ограничење вредности, наслеђено ограничење придодаје се сопственом уз употребу логичког оператора И (енг. *AND*).

Пример 4.2. Ако наслеђени домен има дефинисано ограничење вредности:

$$VALUE > 7,$$

док домен наследник има дефинисано сопствено ограничење вредности:

$$VALUE < 15,$$

резултујући логички услов ограничења вредности домена наследника биће:

$$VALUE > 7 \text{ AND } VALUE < 15. \square$$

Слика 4.1 приказује формалну дефиницију граматике ограничења вредности домена, преузету из [5]. Оригинално, граматика је развијена у *ANTRL* нотацији [46], али се због лакше читљивости у даљем тексту користи Проширена Бакус-Наурова Форма (ПБНФ).

```
Exp = Exp bin_operator Exp | un_operator Exp | Primary_Exp;
Primary_Exp = constant | 'VALUE'['.'field_name] |  func_name '(' [Exp_List] ')' | '(' Exp ')';
Exp_List = Exp { ',' Exp_List};
```

Слика 4.1. Спецификација граматике ограничења вредности домена у ПБНФ

У граматици са слике 4.1, користе се терми *bin_operator* и *un_operator* које представљају стандардне аритметичке, логичке и стринг операторе:

- сабирање и одузимање (+, -);
- множење и дељење (*, /);
- поређење бројева и датума (<, <=, >, =>);
- једнакост (= =, !=);
- конкатенација (||);
- логичке операције (NOT, AND, OR, XOR, $\Rightarrow$, $\Leftrightarrow$);
- операцију припадности скупу (IN); и
- операцију поређења стрингова путем регуларних израза (LIKE).

Сви наведени оператори су уведени са истом семантиком коју имају у било ком дијалекту *SQL* језика.

Једина дозвољена промењива у изразима ограничења вредности домена је *VALUE*. Ова промењива означава вредност која може бити додељена атрибуту којем је претходно додељено посматрано ограничење домена. Уколико је у питању домен типа избора или торке, промењива *VALUE* може бити додатно квалификована именом члана домена, који је у граматици са слике 4.1 означен термом *field_name*. У том случају ограничење вредности се односи само на референцираног члана домена.

Терми *constant* и *func_name* означавају константу и уграђену или кориснички дефинисану функцију, редом.

Дефиниција атрибута може садржати и ограничење вредности које специфицира

скуп вредности које се могу доделити атрибуту. По структури, ограничење вредности атрибута представља логички израз у коме се као једина промењива појављује сам атрибут. Ако је атрибуту додељен домен торке или избора, логички израз се односи на једног или више чланова домена. Пошто се атрибут дефинише на нивоу модела целог информационог система, ограничење вредности атрибута мора бити задовољено у сваком типу компоненте у ком се појави атрибут.

Пример 4.3. Ако пројектант жели ограничити вредности атрибута *A* на опсег [7,15], у дефиницији атрибута биће специфицирано следеће ограничење вредности:

$$A >= 7 \text{ AND } A <= 15. \square$$

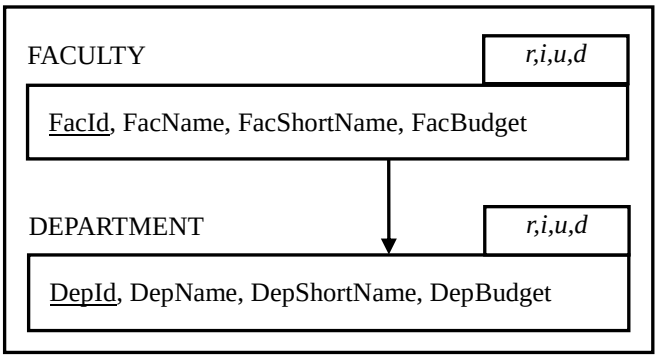
Формална дефиниција граматике за ограничења вредности атрибута уведена је у [5]. Ради целовитости овог текста, дефиниција ће бити поновљена у ПБНФ на слици 4.2. Терм *att_name* представља име атрибута за који се дефинише ограничење вредности док су остали терми дефинисани код граматике ограничења вредности домена.

```
Exp = Exp bin_operator Exp | un_operator Exp | Primary_Exp;
Primary_Exp = constant | att_name ['.'field_name] | func_name '(' [Exp_List] ')' | '(' Exp ')';
Exp_List = Exp { ',' Exp_List};
```

Слика 4.2. Граматика ограничења вредности атрибута у ПБНФ

Појмови типа форме и типа компоненте уведени су у поглављу 3.1.2. У наставку текста дат је следећи пример, који уводи појам ограничења вредности на нивоу типа компоненте.

Пример 4.3. На слици 4.3 дат је тип форме FACULTY DEPARTMENTS, који се састоји из два типа компоненте: FACULTY и DEPARTMENT. Тип компоненте FACULTY моделује факултет са атрибутима: шифру, назив, скраћен назив и буџет факултета. Тип компоненте DEPARTMENT представља модел факултетског департмана и садржи следећа обележја: шифру, име, скраћено име и буџет департмана. Тип форме FACULTY DEPARTMENTS специфицира форму корисничког интерфејса информационог система универзитета. □



Слика 4.3. Тип форме FACULTY DEPARTMENTS

Ограничење вредности на нивоу типа компоненте је формално дефинисано у [5]. Ограничење вредности на нивоу типа компоненте може референцирати атрибуте посматраног типа компоненте и атрибуте надређених типова компоненти у оквиру

истог типа форме.

Пример 4.4. Ако идентификатор департмана мора да буде ненегативан број, пројектант ће на нивоу типа компоненте DEPARTMENT дефинисати следеће ограничење вредности:

$$DepId >= 0.$$

На нивоу истог типа компоненте, пројектант може дефинисати још једно ограничење вредности како би обезбедио да буџет било ког појединачног депатмана мора бити мањи од буџета факултета:

$$FacBudget > DepBudget. \square$$

Ради потпуности ове дисертације, формална дефиниција ограничења вредности на нивоу типа компоненте, оригинално дата у [5], поновљена је на слици 4.4. У граматици, терм *cmpAttName* означава име обележја типа компоненте којег референцира ограничење вредности. Остали терми и операције имају исто значење као и у граматикама на нивоу домена и атрибута.

```
Exp = Exp bin_operator Exp | un_operator Exp | Primary_Exp;
Primary_Exp = constant | cmpAttName ['.' fieldName] |
                    function_name '(' [Exp_List] ')' | '(' Exp ')';
Exp_List = Exp { ',' Exp_List};
```

Слика 4.4. Граматика ограничења торке на нивоу типа компоненте

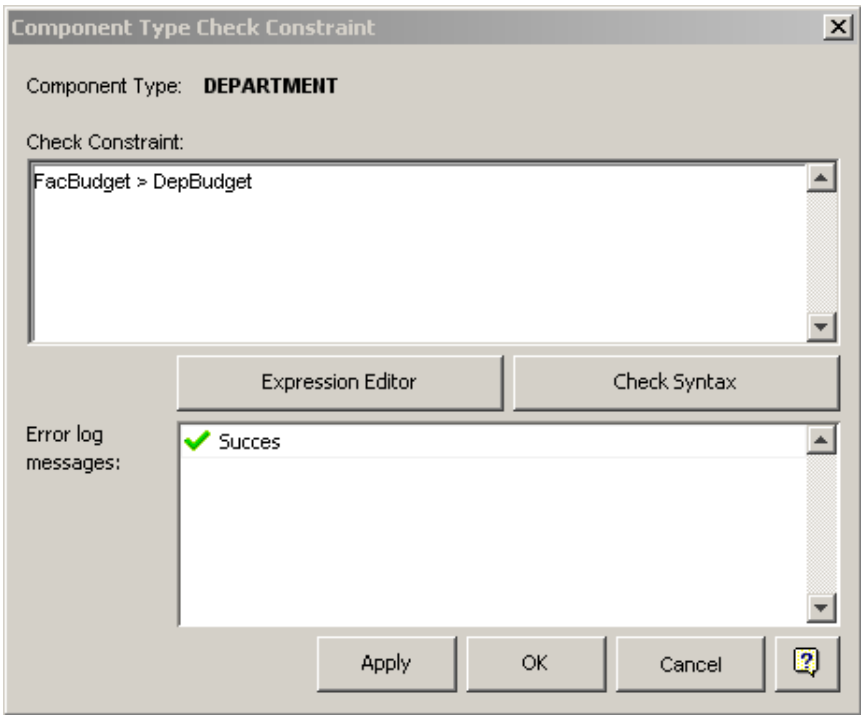
У *IIS*Case*-у постоје два начина да се специфицирају ограничења вредности:

1) директним уношењем логичког израза у дијалогу за дефиницију домена, атрибута или типа компоненте и
2) коришћењем алата *Expression Editor*, који служи за навођено дефинисање ограничења вредности.

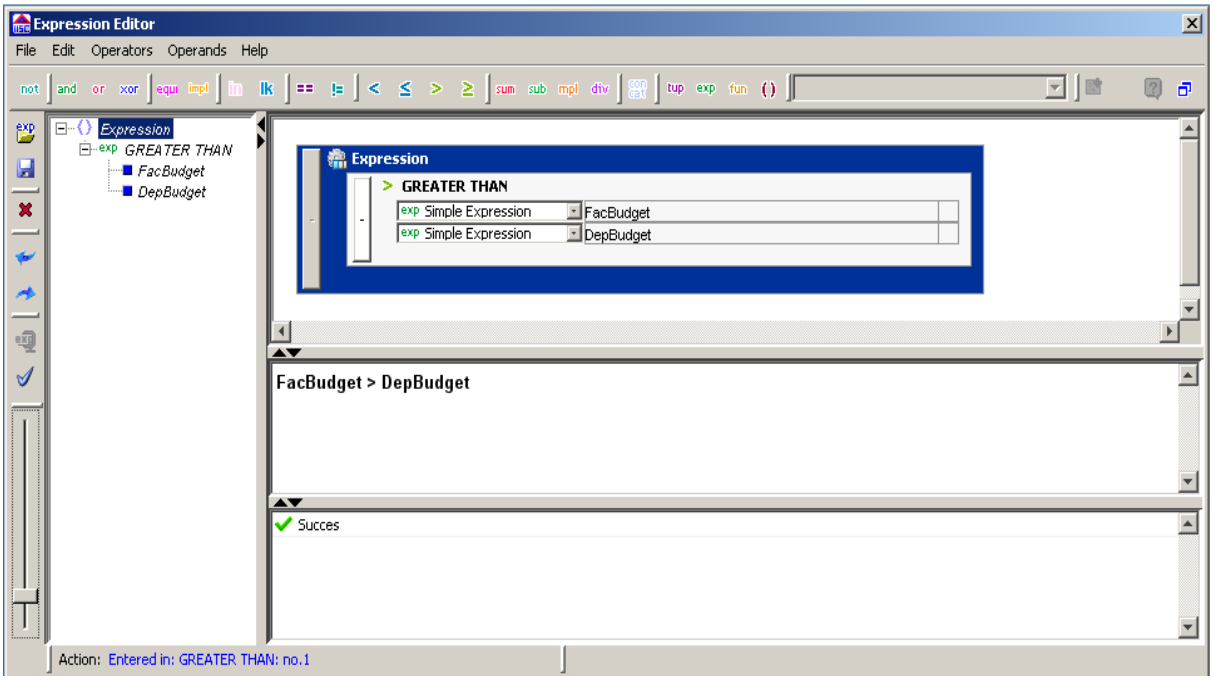
Слика 4.5 приказује дијалог за ручно специфицирање ограничења вредности на нивоу типа компоненте. Овај начин спецификације ограничења вредности захтева познавање синтаксе ограничења те је погоднији за искусне кориснике.

Алат *Expression Editor*, приказан на слици 4.6, омогућава пројектанту да специфицира ограничење вредности користећи графичке команде за генерисање логичких, аритметичких, стринг и датумских оператора. Такође, алат *Expression Editor* нуди синтаксну проверу моделованог ограничења уз јасне поруке о евентуалним грешкама. Из тог разлога, овај алат је погоднији за употребу корисника који имају мање искуства у пројектовању информационих система. Више детаља о алату може се пронаћи у [8, 5].

Да би моделовао ограничења вредности, пројектант не мора да познаје синтаксу *SQL* језика нити одлике СУБП. Једино потребно знање је из домена информационог система и математике, како би ограничење било дефинисано на исправан начин. Стога, наменски језик за спецификацију ограничења вредности омогућава пројектанту да сву пажњу усмери на срж проблема, односно на што верније моделовање информационог система.



Слика 4.5. Дијалог за директно дефинисање ограничења вредности на нивоу типа компоненте



Слика 4.6. Алат Expression Editor

4.1 Закључак

Платформски независан модел ограничења вредности, који је саставни део модела типова форми, представљен је у овом поглављу. При томе, представљени су модели ограничења вредности на нивоу домена, атрибута и типа компоненти.

Кроз *PIM/PSM* трансформације, IIS*Case трансформише спецификације ограничења вредности из модела типова форми у релациони модел података. Ове трансформације описане су у наредном, 5. поглављу.

5 Трансформација ограничења вредности у релациони модел података

У *IIS*Case*-у, платформски независни модел информационог система садржи спецификације будуће шеме базе података, пословне логике, тј. функционалности, и графичког корисничког интерфејса. Делови модела који се односе на базу података трансформишу се, кроз *PIM/PSM* транформације, у концепте релационог модела података који је и даље независан од било ког конкретног СУБП. Алгоритми који се користе у споменутим *PIM/PSM* транформацијама представљени су, уз формални, математички доказ њихове коректности, у [6, 12, 3].

Један од циљева истраживања ове докторске дисертације је да се *PIM/PSM* транформације прошире тако да платформски независне моделе ограничења вредности на нивоу домена, атрибута и типа компоненте транформишу у релациони модел података. Другим речима, процес генерисања релационе шеме базе података треба да укључи и наведена ограничења.

Свако ограничење вредности је дефинисано над одређеним елементом модела, који по свом типу може бити домен, атрибут или тип компоненте. Тај елемент модела представља један део спецификације ограничења вредности и у наставку текста биће за њега коришћен назив контекст ограничења. Други део спецификације ограничења вредности представља сам логички израз који служи за валидацију података путем посматраног ограничења. Приликом превођења ограничења вредности у релациони модел података, потребно је трансформисати и контекст и логички израз ограничења. У наставку текста, прво ће бити представљена *PIM/PSM* транформација контекста ограничења, а затим и трансформација логичког израза.

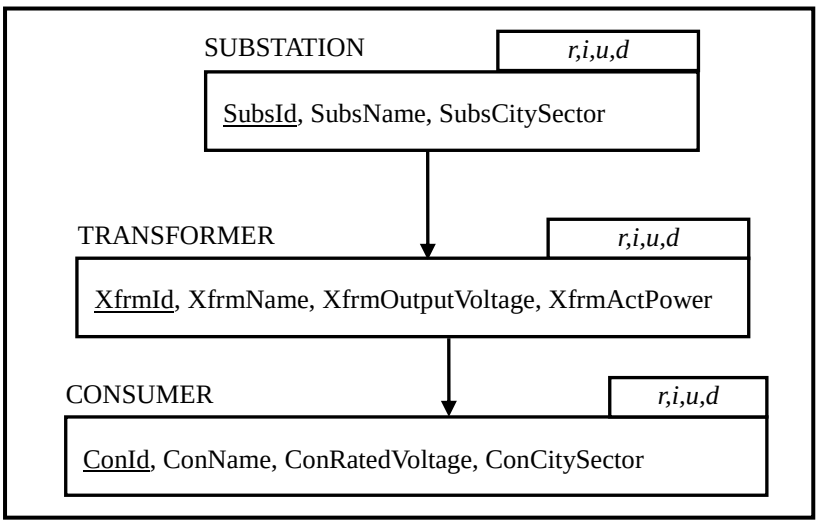
У релационом моделу података постоје концепти домена података и атрибута и носе исту семантику као и истоимени концепти у моделу типова форми. Стога, током трансформације скупа типова форми у релациони модел података, спецификације домена и атрибута остају непромењене, па се самим тим и контекст ограничења вредности над доменом или атрибутом не мења.

С друге стране, контекст ограничења вредности на нивоу типа компоненте мора бити измењен у току генерисања релационе шеме базе података. *PIM/PSM* транформација за ту намену представљена је у даљем тексту.

Типови форми специфицирају, осим осталих ограничења, функционалне зависности између атрибута универзалне шеме релације моделованог информационог система. Функционалне зависности су специфициране задавањем кључева типова компоненти, јер сви остали атрибути типа компоненте функционално зависе од сваког дефинисаног кључа. Користећи прилагођени Бернштајнов алгоритам синтезе [36, 37], *IIS*Case* трансформише скуп функционалних зависности у скуп шема релација. Тиме је скуп типова форми трансформисан у релациони модел података користећи *PIM/PSM* транформацију, која је детаљно представљена у [6, 12]. У даљем тексту, резултујући скуп шема релација обележава се са *S*.

Кардиналитет пресликавања између типова форми и шема релација је *n-n*. Стога, сваком типу форме одговара скуп шема релација, који може бити и јединични али то најчешће није случај. У даљем тексту, тип форме ће бити означен са *F* а скуп кореспондентних шема релације са S_F. Тип форме може се посматрати као дефиниција погледа на кореспондентни скуп шема релација S_F, кроз који корисник може да види и ажурира податке [6]. Дефиниције алгоритама за одређивање S_F за тип форме *F* могу се наћи у [6, 12].

Пример 5.1. Тип форме SERVICE SUBSTATIONS (слика 5.1) представља апстраховану екранску форму на којој се за одабрану сервисну подстаницу дистрибутивне електро мреже приказују списак сервисних трансформатора из подстанице и списак потрошача коју се напајају са трансформатора.



Слика 5.1. Тип форме SERVICE SUBSTATIONS

Тип форме обезбеђује управљање подацима о три ентитета: сервисна подстаница, сервисни трансформатор и потрошач, и сваки од њих је моделован посебним типом компоненте: SUBSTATION, TRANSFORMER и CONSUMER, редом. Ради јасности овог примера, претпостављамо да је тип форме SERVICE SUBSTATIONS и једини тип форме моделованог информационог система.

PIM/PSM транформације имплементиране у *IIS*Case*-у ће дати тип форме пресликати у следећи скуп шема релација:

- *Substation* ({*SubsId, SubsName, SubsCitySector*},{*SubsId*});
- *Transformer* ({*SubsId, XfrmId, XfrmName, XfrmOutputVoltage, XfrmActPower*} {*SubsId+XfrmId*}); и
- *Consumer* ({*SubsId, XfrmId, ConId, ConName, ConRatedVoltage, ConCitySector*}, {*SubsId+XfrmId+ConId*}).

Свака шема релације је написана у облику $N(R,K)$, где је са N означен назив, са R скуп атрибута а са K кључ шеме релације. Ова нотација ће бити коришћена и у наставку текста без поновног дефинисања.

Како модел садржи само један тип форме, наведени резултујући скуп шема релација је уједно и S_F за тип форме SERVICE SUBSTATIONS. □

У циљу проширења постојећих *PIM/PSM* транформација имплементираних у *IIS*Case*-у, развијен је алгоритам за трансформацију контекста ограничења вредности специфицираном на нивоу типа ограничења у релациони модел података. Пошто се контекст ограничења вредности на нивоу домена и атрибута не трансформише, у даљем тексту ће се подразумевати да се ради о ограничењу вредности специфицираном на нивоу типа компоненте.

Развијени алгоритам трансформише ограничење вредности у ограничење торке које је дефинисано над једном или више шема релација у генерисаној шеми базе података. Уколико ограничење торке референцира више шема релација називамо га проширеним ограничењем торке, а шеме релација се називају контекстне шеме релација ограничења торке и скуп тих шема релација означен је са S_{CC}. Како би била очувана семантика моделованог ограничења вредности, шеме релација из S_{CC} морају да очувају спој без губитака, што ће бити детаљно дискутовано у даљем тексту.

Разматрање моделовања и трансформисања проширених ограничења торки пронађено је још само у [15, 16] али због разлике у начину моделовања, исти приступ није могао да буде употребљен у овој докторској дисертацији.

Остатак поглавља је организован на следећи начин. Како би развијени алгоритам могао бити детаљно представљен, у одељку 5.1 дате су дефиниције споја шема релација без губитака података, графа затварања и скупа минималних чворова типа форме. Алгоритам који се користи у *PIM/PSM* транформацији контекста ограничења вредности у релациони модел података дефинисан је у одељку 5.2. Трансформација логичког израза ограничења вредности описана је у одељку 5.3.

5.1 Дефиниције основних појмова

У овом поглављу дате су дефиниције појмова из релационог модела података који се користе приликом дефинисања алгоритма за трансформацију ограничења вредности.

Релациона база података може, на нивоу теорије, бити моделована само једном шемом релације која се назива универзална шема релације. Свака торка универзалне релације садржи вредности свих атрибута свих моделованих ентитета и може се посматрати као један сложени објекат информационог система. Представљање новог објекта, који се разликује од неког другог у само једној вредности неког атрибута, захтева креирање нове торке, код које се вредности свих атрибута целог информационог

система, осим посматраног, понављају, што доводи до проблема појаве високе редундансе података, која је и у теоретском смислу речи и практично непожељна.

Промена вредности неког атрибута једног објекта захтева промену вредности посматраног атрибута у свим торкама универзалне релације у којима се посматрани објекат појављује. Односно, потребно је поновити исте операције приступа торки и промене вредности атрибута, над већим скупом торки, што уноси нежељено продужење времена потребног за измену података.

Слично измени података, уколико се жели обрисати један објекат из универзалне релације, неопходно је обрисати све торке у којима се он појављује. Додатан проблем може да настане уколико остали објекти из брисаних торки носе информацију која не постоји у осталим торкама универзалне релације. У том случају долази до нежељеног губитка те информације.

Наведени проблеми приликом уноса, измене и брисања торки називају се аномалијама ажурирања. Универзална шема релације подложна је овим аномалијама јер универзална релација садржи редундантне податке, односно више торки универзалне релације могу да садрже исте податке о неком ентитету. Опширнија разматрања о универзалној шеми релације и аномалијама ажурирања могу се пронаћи у [37, 47, 48].

Пример 5.2. Универзална шема релације која одговара упрошћеном инфомационом систему електродистрибутивне мреже из примера 5.1. је:

ServiceSubstations({*SubsId, SubsName, SubsCitySector,*
XfrmId, XfrmName, XfrmOutputVoltage, XfrmActPower,
ConId, ConName, ConRatedVoltage, ConCitySector},
{*SubsId*+*XfrmId*+*ConId*}).

Торка која представља подстаницу ,,Грбавица 4'', један њен трансформатор и потрошача Марка Марковића, има следећи облик:

(0, 'Грбавица 4', 'Грбавица', 0, 'ГР4ТР1', 220, 20,
0, 'Марко Марковић', 220, 'Грбавица').

Да би се за исту подстаницу и исти трансформатор унео и потрошач Петар Петровић, потребно унети следећу торку у базу података:

(0, 'Грбавица 4', 'Грбавица', 0, 'ГР4ТР1', 220, 20,
0, 'Петар Петровић', 220, 'Грбавица').

Друга торка садржи идентичне податке о подстаници и трансформатору и тиме ствара непожељну редундантност података. □

Да би непожељна редундантност података и аномалије у ажурирању били спречени, универзална шема релације се декомпонује на скуп мањих шема релација који се назива шема релационе базе података. При томе, релације које се добију декомпозицијом универзалне релације треба, у теоретском смислу речи, да омогуће реконструкцију садржаја универзалне релације, тако што се над њима примени операција природног споја. За шеме релација које задовољавају овај услов каже се да очувавају спој без губитака. Губитак у овом случају не значи да ће резултат природног споја имати мање торки него универзална релација, већ напротив, може имати торке које не постоје у универзалној релацији. Стога, под термином губитак се заправо мисли на губитак информација до којег долази услед стварања непостојећих торки.

Формална дефиниција споја релација без губитака гласи [47]: декомпозиција

$$D = \{N_1(R_1, K_1), N_2(R_2, K_2), \ldots, N_m(R_m, K_m)\}$$

шеме релације $N(R, K)$ поседује својство споја без губитка у односу на скуп функционалних зависности Γ над скупом R ако за сваку релацију r од $N(R, K)$ која задовољава Γ, важи следеће, где је са $\triangleright\triangleleft$ означена операција природног споја:

$$\triangleright\triangleleft(\pi_{R1}(r), ..., \pi_{Rm}(r)) = r.$$

При томе, $\pi_{Ri}(r)$ представља пројекцију релације r на скуп атрибута R_i.

Затварач скупа атрибута X представља скуп атрибута који функционално зависе од X. Такође може се рећи да скуп атрибута X функционално одређује свој затварач. Формална дефиниција затварача скупа атрибута X, у односу на скуп функционалних зависности Γ, је:

$$X_{\Gamma}^{+} = \{A | X \rightarrow A \in \Gamma^{+}\},$$

где је са Γ^{+} означен скуп свих могућих функционалних зависности које се могу извести из Γ.

Граф затварања представља усмерени граф где је сваки чвор једна шема релације која је резултат алгоритма синтезе. Свака усмерена грана представља чињеницу да скуп атрибута надређене шеме релације, из које грана креће, функционално одређује скуп атрибута подређене шеме релације, односно шеме релације у којој се грана завршава. При томе, транзитивне гране у оваквом графу не би носиле било какву нову информацију, те због тога и не постоје. У релационом моделу података, функционална зависност између шема релација реализује се преношењем примарног кључа подређене шеме релације, или једног његовог дела, у надређену шему релације. У случају да се преноси цео примарни кључ, он постаје страни кључ у надређеној шеми релације.

Пример 5.3. За тип форме SERVICE SUBSTATIONS из примера 5.1, граф затварања је приказан на слици 5.2. У графу затварања, шема релације *Consumer* функционално одређује, односно затвара, шему релације *Transformer*. Такође, *Transformer* затвара шему релације *Substation*.

У генерисаној шеми релационе базе података, гране графа затварања биће реализоване преношењем кључа из подређених шема релација у надређене, у виду страног кључа. Односно, у добијеној релационој шеми биће креирана следећа ограничења референцијалног интегритета:

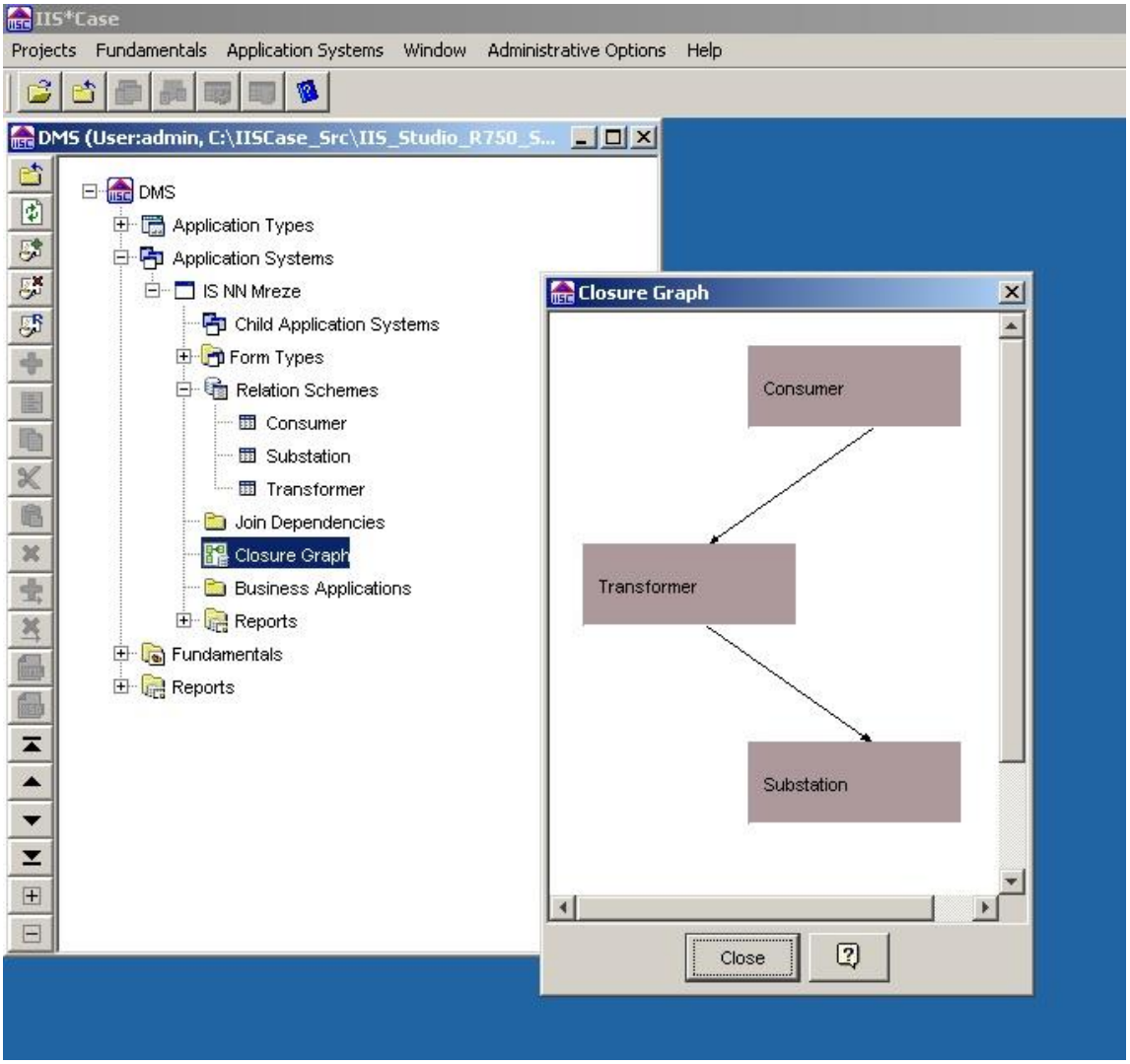
- *Transformer*[*SubsId*] $\subseteq$ *Substation*[*SubsId*] и
- *Consumer*[*SubsId, XfrmId*] $\subseteq$ *Transformer*[*SubsId, XfrmId*)].

У конкретном примеру, кључ шеме релације *Transformer* је уједно и део кључа шема релације *Consumer*. Ипак, у општем случају то није увек тако. □

Појам графа затварања је представљен и дефинисан у [37, 6]. По дефиницији, граф затварања је граф (S, φ), чији чворови су скуп шема релација $S = \{N_i(R_i, K_i) \mid i = 1,..,n\}$ а гране су одређене релацијом φ у S^2 као:

$$\varphi = \{(N_i(R_i, K_i), N_j(R_j, K_j)) \in S^2 \mid R_j \subset (R_i)_{\Gamma}^{+} \wedge (\forall N_k(R_k, K_k) \in (S \setminus \{ N_i(R_i, K_i), N_j(R_j, K_j)\})) (R_j \not\subset (R_k)_{\Gamma}^{+} \vee R_k \not\subset (R_i)_{\Gamma}^{+})\},$$

где је $\Gamma = \{X \rightarrow A \mid (\exists N_i(R_i, K_i) \in S)(A \in (R_i \setminus X) \wedge X \in K_i)\}$.



Слика 5.2. Граф затварања типа форме SERVICE SUBSTATIONS

У алгоритму који ће бити представљен у наставку поглавља користе се две чињенице које следе из графа затварања:

1. скуп атрибута чвора функционално одређује скуп атрибута свих подређених чворова и
2. два повезана чвора у графу затварања представљају две шеме релације за које важи да је кључ подређене шеме релације, или један његов део, пренешен у надређену шему релације.

Ради једноставности излагања материје у даљем тексту, уводе се следеће конвенције у означавању. Ако скуп атрибута R_i затвара скуп атрибута R_j, може се рећи да шема релације $N_i(R_i, K_i)$ затвара шему релације $N_j(R_j, K_j)$. Такође, може се рећи да шема релације $N(R, K)$ затвара скуп атрибута X, када скуп атрибута шеме релације R затвара X.

У оквиру скупа кореспондентних шема релације S_F, могуће је одредити подскуп такав да сви чланови подскупа заједно затварају све остале шеме релације из S_F и да не постоји други подскуп који поседује исто својство и има мањи број чланова. Описани подскуп назива се скуп минималних чворова типа форме F и обележава се са SMN_F. Дефиниција скупа минималних чворова је:

$$SMN_F = \{N_i(R_i, K_i) \in S_F \mid (\forall N_j\,(R_j, K_j) \in (S_F \setminus \{N_i(R_i, K_i)\})\,)(N_j(R_j, K_j), (N_i(R_i, K_i)) \notin \varphi\},$$

где је φ релација графа затварања (S, φ). Алгоритам за проналажење скупа SMN_F дат је у [6].

Пример 5.4. За тип форме SERVICE SUBSTATIONS, из примера 5.1, скуп минималних чворова представља скуп {*Consumer*}.□

5.2 Алгоритам за трансформацију контекста ограничења вредности на нивоу типа компоненте

Алгоритам, оригинално дефинисан у [13], трансформише ограничење вредности на нивоу типа компоненте, специфицираног у моделу типова форми, у ограничење торке у релационом моделу података. Улазни параметри алгоритма су:

- скуп кореспондентних шема релација S_F за посматрани тип форме F,
- скуп минималних чворова SMN_F,
- граф затварања (S, φ) и
- логички израз ограничења вредности LE_{CC}.

Резултат алгоритма је скуп шема релација S_{CC} које референцира добијено ограничење торке у релационој шеми базе података. Елементи скупа S_{CC} називају се контекстне шеме релација резултујућег ограничења торки и очувавају особину споја без губитака. Пошто ограничење вредности може референцирати само атрибуте типа форме F, S_{CC} је увек подскуп од S_F.

Алгоритам се састоји из следећа три корака.

Корак 1. Први корак алгоритма проналази шему релације из S_F, која затвара све атрибуте референциране од стране логичког израза LE_{CC}, такву да у графу затварања не постоји ни једна подређена шема релације која поседује исту особину. Описана шема релације назива се минимални чвор ограничења торке. Како је показано у [37], проналажење минималног чвора је неопходно да би се обезбедио спој контекстних шема релација без губитака.

Унија затварача свих шема релација из SMN_F садржи све атрибуте типа форме и стога садржи и све атрибуте референциране од стране LE_{CC}. Из тог разлога, алгоритам проверава сваку шему релације из SMN_F да ли затвара све атрибуте из LE_{CC}. Када пронађе одговарајућу шему релације, алгоритам се даље „спушта'' по графу затварања и испитује њој подређене шеме релација да ли затварају све атрибуте LE_{CC}. Алгоритам се завршава када се пронађе шема релације чија ниједна подређена шема не затвара све атрибуте LE_{CC} и пронађена шема се проглашава за минимални чвор ограничења торке.

Сврха проналажења минималног чвора ограничења торке је да скуп контекстних шема релација, који је резултат алгоритма, садржи само потребне елементе уз обезбеђивање споја без губитака. □

Корак 2. У другом кораку, алгоритам проналази све шеме релација које су референциране од стране ограничења торке, односно шеме релација које имају непразан пресек са скупом атрибута LE_{CC}. Да би то постигао, алгоритам претражује

скуп S_F. Пронађене шеме релација називају се релевантне шеме или релевантни чворови. □

Корак 3. У последњем кораку, алгоритам проналази путање између минималног чвора ограничења торке и свих релевантних чворова. Те путање обезбеђују природни спој између минималног и релевантних чворова, тј. спој између контекстних шема релације ограничења торке. Чворови који служе да би се обезбедио спој називају се неопходни чворови.

Минимални чвор функционално одређује све атрибуте LE_{CC}, па због дефиниције графа затварања, следи да су релевантни чворови подређени минималном чвору ограничења торке. У скуп неопходних чворова је потребно уврстити што мањи број чворова, односно потребно је одредити минимални скуп шема релација над којима ће се проверавати ограничење торке.

Стога, релеватни чворови се траже употребом прилагођене претраге у ширину, која започиње у минималном чвору и креће се по усмереним гранама графа затварања док не стигне до свих релевантних чворова. На сваком нивоу графа, идентификују се чворови који воде до релевантних чворова и они постају кандидати за неопходне чворове. При томе, услов да би шема релације била кандитат за неопходни чвор је да њен затварач садржи све атрибуте бар једне од релевантних шема релације. Из скупа кандидата, алгоритам бира најмањи подскуп који затвара све релевантне шеме релација и тај подскуп, на следећем нивоу графа затварања, се проверава у наредној итерацији. Претрага се окончава када стигне до свих релевантних чворова. □

Резултат алгоритма S_{CC} представља унију минималног чвора, скупа релевантних чворова и скупа неопходних чворова. Минимални чвор ограничења торке очувава природни спој без губитака шема из S_{CC}, што уједно обезбеђује и да ограничење торке има исту семантику у релационој шеми базе података као и у моделу типова форми. Следећа два примера илуструју рад алгоритма за трансформацију контекста ограничења торке из модела типова форми у релациони модел података.

Пример 5.5. У типу форме SERVICE SUBSTATIONS, из примера 5.1, потребно је специфицирати следеће ограничење:

„Напон на секундару трансформатора мора бити једнак номиналном напону потрошача.”

Ово ограничење се у моделу типова форми представља ограничењем торке на нивоу типа компоненте *Consumer*:

ConRatedVoltage == XfrmOutputVoltage.

На почетку, алгоритам проналази минимални чвор ограничења торке, што је у овом случају шема релације *Consumer*. Разлози за то су:

1. шема релације *Consumer* припада скупу минималних шема релација типа форме SERVICE SUBSTATIONS,
2. изабрана шема затвара све атрибуте референциране од стране LE_{CC} и
3. не постоји ниједна подређена шема релација која такође затвара све атрибуте LE_{CC}.

Такође, једина релевантна шема релације је *Transformer*, јер заједно са шемом *Consumer* садржи све атрибуте LE_{CC}. Пошто су ове две шеме директно повезане графом затварања, скуп неопходних чворова је празан и коначан резултат алгоритма управо

представљају наведене шеме релација *Consumer* и *Transformer*.□

Пример 5.6. У типу форме SERVICE SUBSTATIONS из примера 5.1, потребно је дефинисати још једно правило пословања:

„Потрошач на Грбавици може бити прикључен само на сервисни транформатор који се налази на Грбавици или на Лиману.”

Ово правило може једино бити специфицирано у виду ограничења торке на нивоу типа компоненте *Consumer* на следећи начин:

$$ConCitySector == \text{‘Грбавица’} \Rightarrow$$
$$(SubsCitySector == \text{‘Грбавица’} \vee SubsCitySector == \text{‘Лиман’}).$$

Из истих разлога као и у примеру 5.5, шема релације *Consumer* је минимални чвор ограничења торке. Атрибути LE_{CC} су *SubsCitySector* из шеме *Substation* и *ConCitySector* из шеме *Consumer* па је шема релације *Substation* једина релевантна. Претрага „први у ширину’’ по графу затварања са слике 5.2, која почиње у шеми релације *Consumer* и завршава се у шеми релације *Substation*, означиће шему релације *Trasformer* као неопходну да би био обезбеђен спој без губитака између шема *Consumer* и *Substation*.

На тај начин, коначан резултат алгоритма је скуп контекстних шема релација којем припадају *Consumer*, *Substation* и *Transformer*. □

У даљем тексту, алгоритам за трансформацију ограничења торки из модела типова форми у релациони модел података представљен је у форми псеудо-кôда (слика 5.3).

PROCESS TransformCheckConstraint(**I**(S_F, (S, φ), SMN_F, LE_{CC}) , **O**(S_{CC}), **IO**( ))
BEGIN TransformCheckConstraint
  CALL CheckConstraintMinimalNode (S_F, SMN_F, (S, φ), LE_{CC}, MN_{CC})
  CALL CheckConstraintRelevantNodes (S_F, LE_{CC}, SRN_{CC})
  CALL CheckConstraintNecessaryNodes (S_F, (S, φ), MN_{CC}, SRN_{CC}, SNN_{CC})
  SET $S_{CC} \leftarrow MN_{CC} \cup SRN_{CC} \cup SNN_{CC}$
END TransformCheckConstraint

Слика 5.3. Главна процедура алгоритма за трансформацију ограничења торки

Главна процедура TransformCheckConstraint обједињује читав алгоритам и позива подпроцедуре које представљају кораке алгоритма. Променљиве које су коришћене у псеудо-кôду, а нису уведене раније у тексту, имају следеће значење:

- MN_{CC} – минимални чвор ограничења торке,
- SRN_{CC} – скуп релевантних чворова и
- SNN_{CC} – скуп неопходних чворова.

Процедура CheckConstraintMinimalNode, дата на слици 5.4, проналази минимални чвор ограничења торке. Поред већ уведених, процедура CheckConstraintMinimalNode користи и две нове променљиве:

- Attr(LE_{CC}) – скуп атрибута референцираних од стране логичког израза ограничења торке LE_{CC} и
- Γ – скуп свих функционалних зависности добијених из свих специфицираних типова форми.

```
PROCESS CheckConstraintMinimalNode ( I(S_F, SMN_F, (S,φ), LE_CC) , O(MN_CC ), IO ( ) )
BEGIN PROCESS CheckConstraintMinimalNode
  SET Cand ← SMN_F
  DO FindMinimalNode
    SET Found ← FALSE
    DO CheckCandidates ( ∀ N_i(R_i, K_i) ∈ Cand )
      IF (Attr(LE_CC) ⊆ (R_i)+_Γ ) THEN
        SET MN_CC ← N_i(R_i, K_i)
        SET Found ← TRUE
        EXIT CheckCandidates
      END IF
    END DO CheckCandidates
    IF Found THEN
      CALL Subordinated(MN_CC, S_F, (S,φ), Cand)
    ELSE
      EXIT FindMinimalNode
    END IF
  END DO FindMinimalNode
END PROCESS CheckConstraintMinimalNode
```

Слика 5.4. Процедура за проналажење минималног чвора ограничења торке

У циљу доказивања коректности процедуре CheckConstraintMinimalNode, потребно је кренути од основних корака процеса генерисања релационе шеме базе података, који је имплементиран у *IIS*Case*-у. Основни кораци трансформисања скупа типова форми у скуп шема релација су извођење функционалних зависности из скупа типова форми и примена модификованог Бернштајновог алгоритма синтезе, као што је детаљно описано у [37].

Тип форме је структура типа стабла, чији чворови су типови компоненти а гране су везе типа надређени-подређени. Стога, кључ типа компоненте је унија по једног кључа сваког типа компоненте који се налази на путу од коренског до посматраног типа компоненте. Даље, сваки тип компоненте представља једну класу објеката (ентитета или повезника) информационог система. Тип компоненте, по правилу, трансформише се касније у једну шему релације, применом модификованог алгоритма синтезе. Добијена шема релације такође представља исту класу објеката информационог система, али сада исказану путем концепата релационог модела података. На тај начин, она може да репрезентује било тип ентитета, било тип повезника са кардиналитетима више према више, уколико би као контекст посматрања био модел типова ентитета и повезника.

Да би се наведена трансформација постигла, из сваког типа компоненте изводи се бар једна функционална зависност чија лева страна је кључ типа компоненте а оригинални алгоритам синтезе је модификован да очува посматрану функционалну зависност и њу трансформише у шему релације која ће имати исти кључ. Ако тип компоненте моделује повезник информационог система, десна страна функционалне зависности је празан скуп и таква функционална зависност назива се нефункционалним односом. Детаљна разматрања у вези са нефункционалним односима дата су у [3, 6].

Из претходног пасуса следи да ће модификовани алгоритам синтезе сигурно обезбедити постојање шеме релације чији кључ је унија по једног кључа сваког типа

компоненте који се налази на путу од коренског до посматраног типа компоненте. Пошто ограничење торке може да референцира само атрибуте из типа компоненте којем припада и атрибуте из надређених типова компоненти, посматрана шема релације затвара све атрибуте LE_{CC}. Овим је доказано да минимални чвор ограничења торке мора да постоји.

Шема релације, која је минимални чвор ограничења торке, настала је трансформацијом типа компоненте типа форме F, па припада скупу S_F, односно скуп SMN_F је затвара. Из те чињенице, као и дефиниције графа затварања следи да минимални чвор ограничења торке или припада скупу SMN_F или је подређен чвор скупа SMN_F. Овим је доказана коректност процедуре CheckConstraintMinimalNode.

Процедура Subordinated (слика 5.5) за задату шему релације селектује директно подређене шеме релације. При томе, селектују се само шеме релација које припадају S_F јер су од интереса само шеме релација које могу бити референциране од стране ограничења торки, а које је дефинисано у склопу типа форме. Коректност процедуре произилази из саме дефиниције графа затварања. Све промењиве које се користе у процедури имају исто значење које је уведено у ранијем тексту.

За сваки позив процедуре, петља FindSubordinatedNodes позива се за сваки елемент из S_F по једном. Зато је временска комплексност процедуре процењена са $O(n)$.

```
PROCESS Subordinated ( I(Ni(Ri, Ki),  SF , (S,φ)) , O (Sres ), IO ( ) )
BEGIN PROCESS Subordinated
  SET Sres ← Ø
  DO FindSubordinatedNodes (∀Nj(Rj, Kj)∈ SF)
    IF ((Ni(Ri, Ki), Nj(Rj, Kj)) ⊆ φ) THEN
      SET Sres ← Sres ∪ { Nj(Rj, Kj) }
    END IF
  END DO FindSubordinatedNodes
END PROCESS Subordinated
```

Слика 5.5. Процедура за селекцију директно подређених чворова у графу затварања

Петља FindMinimalNode у процедури CheckConstraintMinimalNode (слика 5.4) се извршава само на делу целог графа затварања (S,φ), који је подређен скупу SMN_F. Нека k буде број извршавања петље. Пошто се за сваки чвор петља понови максимално једном, следи

$$k < |S| = n.$$

Такође, у сваком проласку кроз петљу FindMinimalNode, позива се процедура Subordinated. Следи да је комплексност процедуре CheckConstraintMinimalNode $O(n^2)$.

Процедура CheckConstraintRelevantNodes представљена је у форми псеудо-кôда на слици 5.6. Све промењиве су већ уведене раније у тексту. Процедура проналази релевантне чворове тако што за сваку шему релације из скупа S_F проверава да ли има непразан пресек са скупом атрибута LE_{CC}. Ако је тај услов испуњен, шема релације представља релевантан чвор. Пошто се петља FindRelevantNodes извршава једном за сваку шему релације из скупа S_F, комплексност процедуре је $O(n)$.

```
PROCESS CheckConstraintRelevantNodes ( I(S_F, LE_CC) , O(SRN_CC), IO ( ) )
BEGIN PROCESS CheckConstraintRelevantNodes
  SET SRN_CC ← Ø
  DO FindRelevantNodes (∀N_i(R_i, K_i) ∈ S_F)
    IF (R_i ∩ Attr(LE_CC) <> Ø) THEN
      SET SRN_CC ← SRN_CC ∪ { N_i(R_i, K_i) }
    END IF
  END DO FindRelevantNodes
END PROCESS CheckConstraintRelevantNodes
```

Слика 5.6. Процедура за проналажење релевантних чворова

Процедура CheckConstraintNecessaryNodes је преузета без измена из [6], где је дат формални доказ њене коректности, а комплексност процењена на $O(n^4)$.

Пошто су дати или наведени сви потребни докази коректности корака процедуре TransformCheckConstraint, следи да је читав алгоритам за трансформацију ограничења торке у релациони модел података такође коректан. Пошто се кораци алгоритма извршавају редом и сваки по једном, комплексност алгоритма једнака је комплексности најзахтевнијег корака, односно процењена је са $O(n^4)$.

5.3 Трансформација логичког израза ограничења вредности

Логички изрази ограничења вредности домена, ограничења вредности атрибута и ограничења торке представљени су у поглављу 4 и дефинисани граматикама на сликама 4.1, 4.2 и 4.4, редом. Трансформације логичког израза из модела типова форми, тј. наменског (*DSL*) језика, у релациони модел података је иста за сва три типа ограничења. Стога, у даљем тексту говори се само о логичком изразу без експлицитног навођења ком типу ограничења припада.

У склопу *PIM/PSM* трансформација ограничења вредности и ограничења торки, логички израз се трансформише у дисјунктивну (ДНФ) или конјунктивну нормалну форму (КНФ), примењујући логичке таутологије у датом редоследу:

1. замена импликације негацијом и дисјункцијом:
$$(p \Rightarrow q) \Leftrightarrow (\neg p \vee q),$$
2. замена еквиваленције негацијом, конјункцијом и дисјункцијом:
$$(p \Leftrightarrow q) \Leftrightarrow ((\neg p \vee q) \wedge (\neg q \vee p)),$$
3. правило двоструке негације:
$$\neg\neg p \Leftrightarrow p,$$
4. Де Морганова правила:
$$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q) \text{ и } \neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q) \text{ и}$$
5. дистрибутивни закони:
$$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r) \text{ и}$$
$$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r).$$

Формални опис трансформације може се пронаћи у [49], док су дефиниције употребљених таутологија доступне у [50]. Резултат трансформације је логички израз дат у облику:

$$\wedge (\underset{i}{\vee} l_i),$$

где l_i представља атомски логички израз који не садржи логичке операторе и не може се сводити на простије логичке изразе. Овакви изрази називају се литералима.

Пример 5.7. Следећи изрази могу представљати литерале КНФ или ДНФ логичког израза ограничења торке:

- *A* > 0,
- 0.3**A* + *B* > 15,
- *DOB* > ToDate('1900-01-01'),
- *Name* == 'Joe',
- *Surname* LIKE 'Jo%' или
- *X* IN [1,3,5,7],

где *A*, *B*, *DOB*, *Name*, *Surname* и *X* представљају атрибуте референциране од стране ограничења торке. □

Резултујући логички израз је и даље у складу са граматиком полазног израза, уз разлику да логичке операције еквиваленције (⇔) и импликације (⇒) нису дозвољене.

Ова трансформација је неопходна пошто *SQL* платформе за које ће бити генерисан кôд не подржавају логичке операције еквиваленције и импликације. Насупрот томе, оне су подржане у наменском језику за дефинисање ограничења вредности домена и атрибута и ограничења торки. Такође, други и важнији разлог за то није само техничке природе, као претходно наведени. Ова врста трансформације је потребна и због касније семантичке анализе ограничења торки и проверу консолидованости подшеме са јединственом шемом базе података са аспекта ограничења торки, што је детаљно описано у поглављу 7.

Пример 5.7. Логички израз ограничења торке из примера 5.6, трансформише се у ДНФ:

$$\neg (\textit{ConCitySector} == \text{'Грбавица'}) \vee$$
$$(\textit{SubsCitySector} == \text{'Грбавица'} \vee \textit{SubsCitySector} == \text{'Лиман'}),$$

или КНФ, која за овај конкретан пример има исти облик.

5.4 Закључак

У приступу развоја информационих система који је подржан алатом *IIS*Case*, након моделовања информационог система, следећи корак представља низ *PIM/PSM* трансформација које скуп типова форми трансформишу у релациону шему базе података.

Као први корак ка креирању кориснички оријентисаног развоја ограничења вредности, што представља циљ ове дисертације, развијена је нова *PIM/PSM* трансформација која спецификације ограничења вредности преводи из модела типова форми у релациони модел. Нова трансформација описана је у овом поглављу. Као и

постојеће *PIM/PSM* трансформације, и нова трансформација је у складу са *MDSD* приступом [39].

У моделу типова форми, ограничења вредности могу бити специфицирана на нивоу домена, атрибута или типова компоненти, а трансформација ће таква ограничења превести у ограничења вредности у релационом моделу, на нивоу домена, атрибута или скупа шема релација, редом. Највећи изазов приликом креирања трансформације био је одређивање коректног скупа шема релација, у случају ограничења вредности специфицираног на нивоу типа компоненте, а да се притом обезбеди спој без губитака и очува семантика моделованог ограничења.

Добијене спецификације ограничења вредности у шеми релационе базе података представљају основу за формирање поступка генерисања извршивог *SQL/DDL* кôда, који је описан у наредном поглављу.

6 Генерисање извршивог кôда за ограничења торки

Релациона шема базе података, генерисана кроз *PIM/PSM* трансформације, представља улазну спецификацију за генерисање извршивог *SQL/DDL* кôда. *SQL* генератор у *IIS*Case*-у тренутно подржава следеће СУБП платформе: *ANSI SQL-2003* стандард, *Oracle 9i* и *10g*, и *MS SQL Server 2000* и *2008*.

У склопу ове дисертације, *SQL* генератор проширен је трансформацијама типа модел-у-кôд, које омогућавају генерисање *SQL/DDL* кôда за ограничења вредности домена и атрибута и ограничења торки, за све тренутно подржане платформе СУБП. Ово поглавље садржи описе и дефиниције SQL/DDL шаблона и помоћних процедура које се користе за генерисање кôда. Процедуре и шаблони су такође представљени у [13].

Улазни параметри модел-у-кôд трансформација су резултати *PIM/PSM* трансформација представљених у претходном поглављу:

- граф затварања – φ,
- логички израз – LE_{CC} и
- спецификације домена, атрибута или скуп контекстних шема релација – S_{CC} у зависности од места дефинисања ограничења.

Ово поглавље је подељено у два одељка. У одељку 6.1, приказано је генерисање *SQL/DDL* кôда за ограничења вредности на нивоу домена и атрибута. У одељку 6.2, описано је генерисање кôда за ограничења торке.

6.1 Генерисање ограничења вредности на нивоу домена и атрибута

Према ANSI стандарду, ограничења вредности домена дефинишу се декларативно

употребом `CREATE DOMAIN` наредбе са клаузулом `CHECK`. *SQL/DDL* шаблон ове наредбе дат је на слици 6.1.

```
CREATE DOMAIN <constraint_name>
  AS <data_type> CHECK (<logical_expression>);
```

Слика 6.1. SQL/DDL шаблон за дефинисање ограничења вредности домена по ANSI SQL-2003 стандарду

Терм `<logical_expression>` означава логички израз ограничења вредности домена, чије име је означено са `<constraint_name>`. Са `<data_type>` означен је тип података циљне *SQL* платформе над којим је домен дефинисан.

С друге стране, *Oracle* и *MS SQL Server* СУБП не подржавају наредбу `CREATE DOMAIN`. Из тог разлога, ограничења вредности домена генеришу се као ограничења вредности на нивоу колоне, употребом `CHECK` клаузуле у оквиру наредбе `CREATE TABLE`. Шаблон за генерисање ограничења вредности на нивоу колоне дат је на слици 6.2. Терми <table_name> и <attribute_name> представљају име табеле и име атрибута, редом. Исти шаблон користи се за генерисање ограничења вредности атрибута за све подржане *SQL* платформе.

```
CREATE TABLE <table_name> (
  …
  <attribute_name> <data_type> CHECK (<logical_expression>),
  …
)
```

Слика 6.2. Шаблон за генерисање ограничења вредности атрибута за све SQL платформе

У платформски независном моделу типова форми, за један атрибут могу истовремено бити специфицирани и ограничење вредности домена и ограничење вредности атрибута. Још једна могућност је да атрибуту буду додељена ограничења вредности и од домена и од његовог домена-родитеља. У овим ситуацијама, када за атрибут треба генерисати више од једног ограничења вредности, користи се исти *SQL/DDL* шаблон са слике 6.2, уз измену да се терм `<logical_expression>` генерише конкатенацијом логичких израза свих ограничења вредности који се односе на атрибут. За конкатенацију се користи логички оператор `AND`. Стога, за један атрибут *SQL* генератор прави максимално једно ограничење вредности који укључује сва специфицирана ограничења вредности која се односе на атрибут.

Пример 6.1. Посматрамо претходно моделовано ограничење вредности домена *NOT_NEG_NUM,* које наслеђује уграђени тип података *NUMBER*. Домен ограничава наслеђени тип података на скуп позитивних бројева спецификацијом следећег ограничења вредности:

$$VALUE >= 0.$$

У моделу типова форми, домен *NOT_NEG_NUM* је додељен атрибуту *PERCENTAGE*, који такође има специфицирано сопствено ограничење вредности:

$$PERCENTAGE <= 100.$$

Даље претпостављамо да атрибут *PERCENTAGE* припада шеми релације *EXAMS* која је добијена из модела типова форми *PIM/PSM* трансформацијом. Да би се на СУБП имплементирала ограничења вредности специфицирана уз домен *NOT_NEG_NUM* и атрибут *PERCENTAGE*, *SQL* генератор ће за *Oracle* и *SQL Server* платформе изгенерисати *SQL/DDL* кôд приказан у листингу 6.1.

```
CREATE TABLE EXAMS (
  …
  PERCENTAGE NUMBER CHECK (PERCENTAGE>=0 AND PERCENTAGE<=100),
  …
)
```

Листинг 6.1. SQL/DDL кôд за атрибут PERCENTAGE, генерисан за Oracle and MS SQL Server платформе

SQL/DDL кôд, генерисан према *ANSI* стандарду, се разликује од кôда за *Oracle* and *MS SQL Server* платформе, јер је кôд за домен *NOT_NEG_NUM* и његово ограничење вредности изгенерисан експлицитно (листинг 6.2). □

```
CREATE DOMAIN NOT_NEG_NUM AS NUMBER CHECK (VALUE>=0);

CREATE TABLE EXAMS (
  …
  PERCENTAGE NOT_NEG_NUM CHECK (PERCENTAGE<=100),
  …
)
```

Листинг 6.2. SQL/DDL кôд за домен NOT_NEG_NUM и атрибут PERCENTAGE, генерисан према ANSI SQL-2003 стандарду

6.2 Генерисање SQL/DDL *кôда за ограничења торке*

У случају генерисања *SQL/DDL* кôда ограничења торки, разликују се два случаја:

1) ограничење торке које референцира само једну шему релације и
2) проширено ограничење торке које референцира више контекстних шема релација.

Шаблон за генерисање ограничења торки које референцира само једну шему релације је исти за све три *SQL* платформе и дат је на слици 6.3.

```
ALTER TABLE <table_name> ADD
  CONSTRAINT <constraint_name> CHECK (<logical_expression>);
```

Слика 6.3. SQL/DDL шаблон за генерисање ограничења торке које референцира једну шему релације, за све три циљне платформе

Процес генерисања *SQL/DDL* кôда за проширена ограничења торки је много комплекснији јер у циљним *SQL* платформама не постоје декларативни механизми за спецификацију оваквих ограничења. У *ANSI* стандарду, постоји могућност декларативног дефинисања ограничења на нивоу целе шеме базе података, које може референцирати више шема релација, али природни спој референцираних шема релација не може бити специфициран декларативно. Са друге стране, *Oracle* и *SQL Server* платформе не подржавају оваква ограничења и код њих проширена ограничења торки морају бити имплементирана употребом процедуралног механизма окидача.

Први корак у генерисању *SQL/DDL* кôда за проширена ограничења торки, који је и заједнички за све циљне платформе, је генерисање израза природног споја шема релација из S_{CC}. Атрибути у платформски независном моделу типова форми дефинисани су према *URSA* правилу, што значи да истоимени атрибути из различитих шема релација имају исту семантику. Истоимени атрибути у различитим шемама релација настају простирањем примарног кључа који у циљној шеми релације представља страни кључ. Стога, израз природног споја прави се спајањем парова шема релација, где се спајање две шеме релације врши генерисањем израза једнакости између истоимених атрибута из различитих шема релација уз услов да је атрибут припада кључу у бар једној од две шеме релације. Изрази једнакости спајају се логичким оператором *AND*. Да би били избегнути транзитивни (редундантни) изрази једнакости, изрази једнакости праве се само између шема релација које су директно повезане у графу затварања.

Некључни истоимени атрибути у различитим шемама релација не учествују у креирању израза природног споја из следећег разлога. Они представљају или хомониме [3, 43] или атрибуте које шеме релације чине A-зависним. У случају хомонима, очекује се од пројектанта да један од атрибута преименује у платформски независном моделу, да би резултујућа шема базе података била валидна. Са друге стране, ако су у питању A-зависне шеме релација, *IIS*Case* приликом трансформације модела типова форми у релациони модел, генерише проширена ограничења референцијалног интегритета како би се очувао спој без губитака. Више информација о A-зависним шемама релација и генерисаним ограничењима може се пронаћи у [3, 7].

Слика 6.4 приказује ПБНФ израза природног споја.

```
natural_join_expression =
table_name.attribute_name '=' table_name.attribute_name
{'AND' table_name.attribute_name'='table_name.attribute_name};
```

Слика 6.4. Израз природног споја у ПБНФ

У наредним шаблонима користи се терм `list_of_context_relation_schemes` који означава листу контекстних шема релација ограничења торке, тј. листу елемената скупа S_{CC}, чија ПБНФ је дата на слици 6.5.

```
list_of_context_relation_schemes = table_name {','table_name};
```

Слика 6.5. Листа контекстних шема релације ограничења торке

ANSI SQL-2003 стандард пружа могућност дефинисања ограничења на нивоу целе шеме базе података употребом CREATE ASSERTION наредбе. Ова наредба је

употребљена за генерисање проширених ограничења торки кроз шаблон на слици 6.6. Идеја шаблона је да се преброје торке природног споја релација чије шеме припадају S_{CC}, а које не задовољавају логички израз ограничења торке. Да би ограничење било задовољено, број таквих торки мора да буде нула.

```
CREATE ASSERTION ASSERT_<constraint_name> CHECK(
  (SELECT count(*) FROM <list_of_context_relation_schemes>
   WHERE <natural_join_expression>
         AND NOT(<logical expression>)) = 0);
```

Слика 6.6. SQL/DDL шаблон за генерисање проширених ограничења торки према ANSI SQL-2003 стандарду

СУБП *Oracle 9i/10g* и *MS SQL Server 2000/2008* немају подршку за `CREATE ASSERTION` наредбу. Стога за имплементацију проширених ограничења торке користи се механизам окидача тако што се за сваку шему релације из S_{CC} генерише по један окидач који валидира податке који се уносе у релацију или мењају у њој.

Сличан приступ може се пронаћи у [16] уз једну разлику по питању перформанси генерисаних окидача. У [16], генерисани окидачи проверавају све торке природног споја релација из S_{CC} приликом сваке операције уноса или измене података која се тиче релација над шемама из S_{CC}. Насупрот томе, у овој дисертацији предлажу се окидачи који проверавају само нове или измењене торке. То се постиже спајањем само нових или измењених торки са осталим релацијама и валидирањем резултата логичким изразом ограничења торки. Акције брисања података не активирају окидаче пошто брисање података не може нарушити ограничење торке.

Приликом генерисања окидача за *Oracle 9i/10g* СУБП, израз природног споја (слика 6.4) и логички израз ограничења торке биће измењени у односу на исте изразе по ANSI стандарду, на следећи начин. У оба израза, *SQL* генератор замениће назив шеме релације над којом се дефинипе окидач псеудо-слогом `:NEW` који означава само нову или измењену торку због које је покренут окидач.

Такође, *SQL* генератор ће избацити шему релације на којој се дефинише окидач из листе контекстних шема релације ограничења торке, како се не би све торке те релације проверавале окидачем. Добијена листа се назива редукована листа контекстних шема релације ограничења торке.

Слика 6.7 представља *SQL/DDL* шаблон описаног окидача. У шаблону користе се следећи терми:

- <reduced_list_of_context_schemes>, који представља редуковану листу контексних шема релације ограничења торке,
- <natural_join_expression_with_:NEW>, који представља израз природног споја модификованог за *Oracle* платформу и
- <logical_expression_with_:NEW>, који представља логички израз ограничења торке модификованог такође за *Oracle* платформу.

Ако је број торки које не задовољавају логички израз ограничења торке већи од нуле, окидач изазива изузетак и цела наредба која је изазвала окидач поништава се.

```
CREATE OR REPLACE TRIGGER TRG_<constraint_name>
  BEFORE INSERT OR UPDATE ON <table_name>
  FOR EACH ROW
  DECLARE cnt INT;
  exc EXCEPTION;
BEGIN
  SELECT count(*) INTO cnt FROM
                        <reduced_list_of_context_schemes>
  WHERE <natural_join_expression_with_:NEW>
      AND NOT(<logical_expression_with_:NEW>);

  IF cnt<>0 THEN
    RAISE exc;
  END IF;
EXCEPTION
  WHEN exc THEN
    RAISE_APPLICATION_ERROR(-20900,'Check constraint
                                    violated');
END;
```

Слика 6.7. SQL/DDL шаблон окидача за имплементацију проширеног ограничења торке на Oracle платформи

Иста логика окидача примењена је и у случају генерисања проширеног ограничења торки за СУБП *MS SQL Server 2000/2008*. Слика 6.8 приказује шаблон за генерисање окидача за *MS SQL Server* платформу.

```
CREATE TRIGGER TRG_<constraint_name>
  ON <table_name>
  FOR INSERT, UPDATE
AS
  DECLARE @cnt INT

  SELECT @cnt=count(*) FROM
              <list_of_context_schemes_with_Inserted>
  WHERE <natural_join_expression_with_Inserted>
        AND NOT(<logical_expression_with_Inserted >)

  IF (@cnt<>0)
  BEGIN
    RAISERROR('Check constraint violated',16,1)
    ROLLBACK TRAN
  END
GO
```

Слика 6.8. SQL/DDL шаблон за генерисање ограничења торки за MS SQL Server платформу

Са аспекта техничке имплементације, постоје следеће две разлике у односу на шаблон за *Oracle* платформу:

1. У изразу природног споја и логичком изразу ограничења торке, назив шеме релације над којом се имплементира окидач мења се уграђеном табелом `Inserted`. Измењени изрази представљени су у шаблону термима:
   - <natural_join_expression_with_Inserted> и
   - <logical_expression_with_Inserted>.

   Табела `Inserted` има исту намену као и псеудо-слог `:NEW` на *Oracle* платформи: она садржи нове и измењене слогове релације над којом је активиран окидач.
2. У листи контекстних шема релације ограничења торке, шема релације над којом се дефинише окидач мења се табелом `Inserted`, чиме се добија израз који је у шаблону представљен термом <list_of_context_schemes_with_Inserted>. Поређења ради, за Oracle платформу, иста шема релације се избацује из листе.

Следе два примера генерисаног *SQL/DDL* кôда за имплементацију ограничења торки представљених у примерима 5.5 и 5.6.

Пример 6.2. За ограничење торке из примера 5.5, дати су листинзи генерисаног *SQL/DDL* кôда за *ANSI SQL-2003* стандард, *Oracle 9i/10g* и *MS SQL Server* СУБП.

Према *ANSI SQL-2003* стандарду, за имплементацију ограничења торке користи се `CREATE ASSERTION` наредба, што је приказано у листингу 6.3. У листингу, израз природног споја означен је масним словима, логички израз ограничења торке је подвучен док је листа контекстних шема релација дата закошеним словима. Исте ознаке биће употребљене и у осталим примерима до краја овог поглавља.

```
CREATE ASSERTION ASSERT_CHKC_TraCon CHECK(
  (SELECT count(*) FROM Transformer, Consumer
   WHERE Consumer.SubsId=Transformer.SubsId
         AND Consumer.XfrmId=Transformer.XfrmId
         AND NOT(Tansformer.XfrmOutputVoltage =
                 Consumer.ConRatedVoltage))
= 0);
```

Листинг 6.3. SQL/DDL кôд за ограничење торке из примера 5.5 према ANSI SQL-2003 стандарду

За Oracle 9i/10g СУБП, ограничење торке је генерисано као окидачи над шемама релација *Transformer* и *Consumer*, који се активирају приликом уноса или измене торке. Како су окидачи генерисани по истом шаблону, листинг 6.4 приказује само окидач на шеми релације *Consumer*. Треба приметити да се шема релације над којом је дефинисан окидач не појављује у редукованој листи контекстних шема релације ограничења торке, којој је у овом случају припада само шема релације *Transformer*. Уместо тога, торке шеме релације *Transformer* спајају се са псеудо-слогом `:NEW`, што се види из израза природног споја.

Листинг 6.5 приказује окидач на шеми релације *Consumer*, који је генерисан на основу спецификације ограничења торке из примера 5.5, за *MS SQL Server* платформу. *SQL* генератор је поново генерисао окидач и на шеми релације *Transformer*, али како су окидачи генерисани по истом шаблону, он је изостављен из примера. У листингу може се видети да уграђена табела `Inserted` учествује у грађењу измењених израза природног споја, логичког израза ограничења торке и листе контекстних шема релација

ограничења торке уместо шеме релације над којом је дефинисан окидач. □

```
CREATE OR REPLACE TRIGGER TRG_CHKC_TraCon_Consumer
  BEFORE INSERT OR UPDATE ON Consumer
  FOR EACH ROW
  DECLARE cnt INT;
          exc EXCEPTION;
BEGIN
  SELECT count(*) INTO cnt FROM Transformer
  WHERE :NEW.SubsId=Transformer.SubsId
        AND :NEW.XfrmId=Transformer.XfrmId
        AND NOT(Transformer.XfrmOutputVoltage =
                :NEW.ConRatedVoltage);

  IF cnt<>0 THEN
    RAISE exc;
  END IF;
EXCEPTION
  WHEN exc THEN
  RAISE_APPLICATION_ERROR(-20900,Check constraint violated');
END;
```

Листинг 6.4. SQL/DDL кôд за окидач који имплементира ограничење торке из примера 5.5 за Oracle платформу

```
CREATE TRIGGER TRG_CHKC_TraCon_Consumer
  ON Consumer
  FOR INSERT, UPDATE
AS
  DECLARE @cnt INT

  SELECT @cnt=count(*) FROM Transformer, Inserted
  WHERE Inserted.SubsId=Transformer.SubsId
        AND Inserted.XfrmId=Transformer.XfrmId
        AND NOT (Transformer.XfrmOutputVoltage =
                 Inserted.ConRatedVoltage)

  IF (@cnt<>0)
  BEGIN
    RAISERROR('Check constraint violated',16,1)
    ROLLBACK TRAN
  END
GO
```

Листинг 6.5. SQL/DDL кôд за окидач који имплементира ограничење торке из примера 5.5 за MS SQL Server платформу

Пример 6.3. Ограничење торке из примера 5.6 је за потенцијалну *ANSI SQL-2003* платформу генерисано употребом CREATE ASSERTION наредбе (листинг 6.6). При

томе, може се запазити да је логички израз ограничења торке трансформисан у ДНФ (подвучени текст), пошто *ANSI* стандард не подржава операцију импликације.

```
CREATE ASSERTION ASSERT_CHKC_SubTraCon CHECK(
  (SELECT count(*) FROM Consumer,Substation,Transformer
   WHERE Transformer.SubsId=Consumer.SubsId
         AND Transfomer.XfrmId=Consumer.XfrmId
         AND Transformer.SubsId=Substation.SubsId
         AND NOT(NOT(Consumer.ConCitySector='Грбавица')∨
                 (Substation.SubsCitySector = 'Грбавица'∨
                  Substation.SubsCitySector = 'Лиман'))
  )= 0)
```

Листинг 6.6. Имплементација ограничења торке из примера 5.6 према ANSI SQL-2003 стандарду

Ради имплементације истог ограничења торке за *Oracle* и *MS SQL Server* платформе, *SQL* генератор направио је окидаче на шемама релације *Consumer*, *Transformer* и *Substation*. Пошто је логика свих окидача иста, у даљем тексту дати су окидачи на истој шеми релације *Consumer* за *Oracle* (листинг 6.7) и *MS SQL Server* (листинг 6.8) СУБП. □

```
CREATE OR REPLACE TRIGGER TRG_CHKC_SubTraCon_Consumer
  BEFORE INSERT OR UPDATE ON Consumer
  FOR EACH ROW
  DECLARE cnt INT;
          exc EXCEPTION;
BEGIN
  SELECT count(*) INTO cnt FROM Transformer, Substation
  WHERE Transformer.SubsId=Substation.SubsId
        AND :NEW.SubsId=Transformer.SubsId
        AND :NEW.XfrmId=Transformer.XfrmId
        AND NOT(NOT(:NEW.ConCitySector ='Грбавица')∨
                (Substation.SubsCitySector = 'Грбавица'∨
                 Substation.SubsCitySector = 'Лиман'));

  IF cnt<>0 THEN
    RAISE exc;
  END IF;
EXCEPTION
  WHEN exc THEN
    RAISE_APPLICATION_ERROR(-20900, Check constraint
violated');
END;
/
```

Листинг 6.7. Oracle окидач на шеми релације Major настао од ограничења торке из примера 5.6

```
CREATE TRIGGER TRG_CHKC_SubTraCon_Consumer
  ON Chair
  FOR INSERT, UPDATE
AS
  DECLARE @cnt INT

  SELECT @cnt=count(*) FROM Substation, Transformer, Inserted
  WHERE Substation.SubsId=Transformer.SubsId
        AND Inserted.SubsId=Transformer.SubsId
        AND Inserted.XfrmId=Transformer.XfrmId
        AND NOT (NOT(Inserted.ConCitySector ='Грбавица')∨
                 (Substation.SubsCitySector = 'Грбавица'∨
                  Substation.SubsCitySector = 'Лиман'));

  IF (@cnt<>0)
  BEGIN
    RAISERROR('Check constraint violated',16,1)
    ROLLBACK TRAN
  END
GO
```

Листинг 6.8. SQL Server окидач који имплементира ограничење торке из примера 5.6

6.3 Закључак

У овом поглављу описан је поступак генерисања *SQL/DDL* кôда за ограничења вредности и дати су шаблони кôда коришћени у генерисању. Описани поступак представља модел-у-кôд трансформацију и имплементиран је као проширење *SQL* генератора окружења *IIS*Case.*

Са овим проширењем *SQL* генератора, пројектанту, кориснику окружења *IIS*Case*, омогућено је да ограничења вредности развија на платформски независном нивоу а не на нивоу кôда. При томе, сви кораци трансформације ограничења вредности од платформски независног модела до *SQL* кôда су аутоматизовани и спроводе се од стране алата, а не пројектанта. Пошто *SQL* генератор подржава неколико иплементационих платформи, једном моделована ограничења вредности могу бити аутоматски трансформисана у *SQL* кôд за за различите СУБП. Додатно, својствености сваке платформе су искоришћене како би кôд био оптимизован за циљну платформу.

7 Усаглашавање ограничења торки моделованих у оквиру подшема са јединственом шемом базе података

Према *IIS*Case* методологији развоја информационих система [11, 3, 9], пројектанти независно моделују апликативне системе. Скуп свих апликативних система, снабдевен структуром типа стабла, чини модел целог информационог система.

Користећи *PIM/PSM* трансформације, *IIS*Case* преводи скуп типова форми једног апликативног система у подшему базе података, а унију скупова типова форми свих апликативних система у јединствену шему базе података. У циљу обезбеђења исправног функционисања апликација информационог система, укључујући све његове подсистеме, неопходно је да је свака подшема формално консолидована, тј. усаглашена с јединственом шемом базе података. Појам формалне консолидованости подшеме са шемом базе података уведен је и дефинисан у [9].

Како је показано у [9], један од услова да подшема базе података буде усаглашена са јединственом шемом базе података, јесте да за свако нетривијално ограничење шеме базе података мора постојати једнако строго или строже одговарајуће ограничење у подшеми која моделује исте податке. Овај услов је интуитивно једноставно прихватити, пошто подшема представља поглед кроз који корисник прегледа, уноси, мења и брише податке. Наиме, ако податак треба да се нађе у погледу над подшемом базе података, онда мора моћи да се нађе и у бази података, формираној над њеном шемом, док обрнуто не мора да важи. У [9] дефинисани су формални услови усаглашености подшеме и шеме базе података, а такође и алгоритми и процедуре за проверу усаглашености следећих типова ограничења: ограничења примарног кључа, ограничења недостајуће вредности, ограничења јединствености и ограничења референцијалног интегритета.

Надовезујући се на тај рад, у овој дисертацији формулисани су услови консолидованости подшема и методе за проверу консолидованости, са аспекта: а) ограничења вредности домена, б) ограничења вредности атрибута и в) ограничења торки, што све заједно представља један од важних доприноса ове докторске

дисертације. Тиме је омогућено пројектантима да ограничења вредности моделују независно, у склопу засебних модела делова информационог система. При томе, дефинисани алгоритми за проверу консолидованости подшеме са аспекта ограничења вредности обезбеђују исправност моделованих ограничења са аспекта интегрисане шеме базе података.

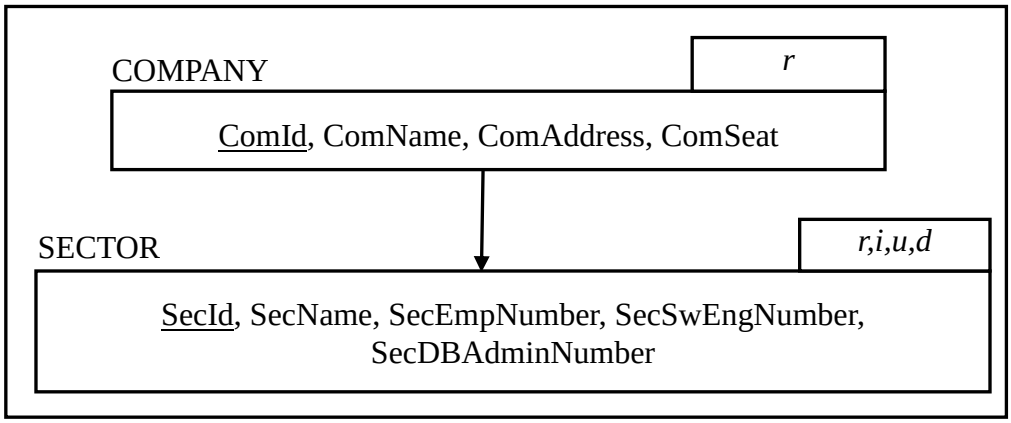
Ограничења разматрана у [9] формализују се путем предикатске логике првог реда. Са друге стране, логички изрази ограничења вредности могу садржати изразе аритметике целих и реалних бројева, изразе над датумима и стринговима, као и кориснички-дефинисане функције, односно изразе који излазе из домена предикатске логике првог реда. Из тог разлога, методе и алгоритми представљени у [9] нису довољни за проверу консолидованости подшеме са јединственом шемом базе података и било је потребно развити нове алгоритме за ту намену.

У платформски независном моделу типова форми, домен и атрибут моделују се на нивоу целог информационог система и њихова спецификација јединствена је за све апликативне системе. Из тог разлога, спецификација домена или атрибута, укључујући и спецификацију припадајућег ограничења вредности, у било којој подшеми иста је као и у шеми базе података. Стога, из самог начина моделовања информационог система у *IIS*Case* алату следи да је подшема консолидована са шемом базе података са аспекта ограничења вредности домена и атрибута.

Насупрот томе, модел ограничења торке специфицира се на нивоу типа компоненте и припада само једном апликативном систему. Како тип форме учествује у генерисању и подшеме и интегрисане шеме базе података, трансформисано ограничење торке постоји након *PIM/PSM* транформација и у подшеми и у шеми.

Различити апликативни системи могу моделовати исте податке у оквиру једног информационог система, сваки са свог аспекта употребе података. При томе, апликативни системи могу унети различита ограничења торки над истим подацима или над скуповима података који се преклапају, па скуп шема релација у интегрисаној шеми базе података може имати више дефинисаних ограничења торки која потичу из различитих апликативних система. Из тог разлога, потребно је усагласити ограничења торки између шеме и свих подшема базе података.

Пример 7.1. У циљу илустрације претходно описаних ситуација са ограничењима торке, посматрамо два типа форме: COMPANY ORGANIZATION (слика 7.1) и DATABASE CLUSTERS (слика 7.2), који припадају различитим апликативним системима у оквиру модела информационог система неке софтверске компаније. Тип форме COMPANY ORGANIZATION користи се за преглед, унос, измену и брисање података о припадајућим секторима, као организационим јединицама. Тип форме DATABASE CLUSTERS моделује кластере база података, које одржава и за које је надлежан један сектор.

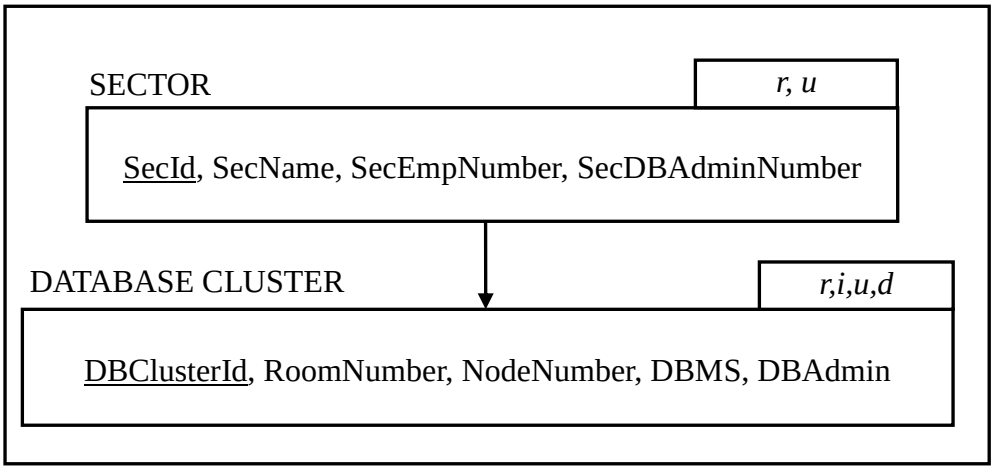


Слика 7.1. Тип форме COMPANY ORGANIZATION

Тип форме COMPANY ORGANIZATION састоји се из типова компоненти COMPANY и SECTOR, где први само приказује податке о компанији, док се други користи за ажурирање података о секторима. Како би обезбедио валидност података о секторима, пројектант је дефинисао следеће ограничење торке на нивоу типа компоненте SECTOR:

SecEmpNumber > SecSwEngNumber AND SecSwEngNumber > SecDBAdminNumber.

Ово ограничење торке уводи следећа пословна правила. Како сектор има запослено и помоћно особље, број запослених у сектору мора бити већи од броја софтверских инжењера. Истовремено, пошто нису сви софтверски инжењери администратори база података, број администратора у сектору мора бити мањи од броја инжењера.



Слика 7.2. Тип форме DATABASE CLUSTERS

Тип форме DATABASE CLUSTERS садржи два типа компоненте, SECTOR и DATABASE CLUSTER. Тип компоненте SECTOR намењен је за преглед и измену података о компанијским секторима, док унос и брисање сектора нису дозвољени. Кроз тип компоненте DATABASE CLUSTER корисник може да види, али и да уноси, мења и брише кластере база података по сваком сектору.

Пошто тип форме DATABASE CLUSTERS припада различитом апликативном систему од типа форме COMPANY ORGANIZATION, он моделује исти скуп података

из другачијег угла посматрања. Стога, на типу компоненте SECTOR пројектант је дефинисао само ограничење:

$$SecEmpNumber > SecDBAdminNumber,$$

пошто атрибут *SecSwEngNumber* није од интереса за овај тип форме.

Путем *PIM/PSM* транформација, из типова форми COMPANY ORGANIZATION и DATABASE CLUSTERS добијен је следећи скуп шема релација:

- Company ({ComId, ComName, ComAddress, ComSeat}, {ComId}),
- Sector ({SecId, SecName, SecEmpNumber, SecSwEngNumber, SecDBAdminNumber, ComId}, {SecId} ∪
  {SecEmpNumber > SecSwEngNumber AND SecSwEngNumber > SecDBAdminNumber, SecEmpNumber > SecDBAdminNumber}),
- DatabaseCluster ({SecId, DBClusterId, RoomNumber, NodeNumber, DBMS, DBAdmin}, { SecId+DBClusterId }) и
- CompanySectorDatabaseCluster ({ComId, SecId, DBClusterId},
  { ComId+SecId+DBClusterId }).

Односно, шема релације Sector наследила је оба ограничења торке, из оба апликативна система. □

За ограничење торке из интегрисане шеме базе података, подшема је од интереса ако је пресек скупа атрибута логичког израза ограничења торке и скупа атрибута подшеме непразан скуп. Алгоритам за проверу консолидованости подшема са интегрисаном шемом базе података [14] проверава да ли за свако ограничење торке шеме базе података, у свакој подшеми од интереса постоји (бар једно) ограничење торке које је једнако строго или строже од ограничења из интегрисане шеме. Ако је тај услов задовољен, подшеме су формално консолидоване са интегрисаном шемом.

Слика 7.3 приказује псеудо-кôд алгоритма. У псеудо-кôду, користе се следеће промењиве:

- S – скуп шема релације интегрисане шеме базе података,
- I_{CC} – скуп ограничења торки из интегрисане шеме,
- SI_{SUB} – скуп уређених парова (S_i, I_i), где S_i представља скуп шема релација подшеме i, а I_i скуп ограничења торки подшеме i;
- Attr – функција чији резултат представља скуп атрибута референцираних од стране задатог аргумента, који је у овом случају подшема или ограничење торке,
- *Report* – скуп уређених парова (S_i, i_S) где је i_S ограничење торке интегрисане шеме због кога је подшема S_i неконсолидована са интегрисаном шемом и
- *Ind* – Булова промењива која представља резултат провере да ли су све подшеме формално консолидоване са шемом.

Коректност алгоритма је директна последица дефиниције формалне консолидованости подшема, дате у [9].

```
PROCESS CheckCheckConstraints(I(S, I_CC, SI_SUB), O(Ind, Report), IO( ))
SET Report ← ∅
SET Ind ← True
DO CheckEachCheckConstraint (∀i_S ∈ I_CC)
  DO CheckEachSubschema (∀(S_i, I_i) ∈ SI_SUB)
    IF Attr(i_S) ∩ Attr(S_i) ≠ ∅ THEN
      IF Attr(i_S) ⊆ Attr(S_i) THEN
        SET Found ← False
        DO CheckSubschemaConstraints (∀i_SS ∈ I_i)
          IF i_SS ⇒ i_S THEN
            SET Found ← True
              BREAK
          END IF
        END DO
        IF Found = False THEN
          SET Ind ← False
          SET Report ← Report ∪ (S_i, i_S)
        END IF
      ELSE
        SET Ind ← False
        SET Report ← Report ∪ (S_i, i_S)
      END IF
    END IF
  END DO
END DO
END PROCESS
```

Слика 7.3. Алгоритам за проверу консолидације подшема са аспекта ограничења торки

Доказивање импликације између два ограничења торке је суштински и уједно најкомпликованији део описаног алгоритма. Овом проблему и начину његовог решевања биће посвећен наредни одељак 7.1. Након тога, у одељку 7.2 описана је имплементација алгоритма у окружењу *IIS*Case*.

7.1 Импликациони проблем ограничења торки

У циљу провере консолидованости подшеме са интегрисаном шемом базе података са аспекта ограничења торки, потребно је доказати да ограничење торке из шеме базе података логички следи из одговарајућег ограничења из подшеме, односно потребно је доказати валидност импликационе формуле ограничења торки:

$$i_{SS} \Rightarrow i_S. \tag{7.1}$$

Формула (7.1) назива се импликациони проблем ограничења торки.

Доказивање валидности логичке формуле, где је сваки литерал исказ, тј. Булова промењива, или Булов предикат, представља проблем доказивања Булове задовољивости (енг. *Boolean satisfiability problem* или *SAT problem*) [51], и припада класи проблема аутоматског доказивања теорема. За решавање овог проблема постоји

велики број алгоритама и алата, названих *SAT* решавачи, који их имплементирају [52].

С друге стране, литерали ограничења торки углавном нису Булове промењиве или предикати, већ:

- изрази аритметике целих и реалних бројева над атрибутима или скаларним функцијама,
- операције над атрибутима типа датума,
- операције над атрибутима типа стринга или
- операције над скуповима.

Ако би била примењена *SAT* техника и *SAT* решавач за доказивање валидности импликационе формуле ограничења торки (7.1), сваки литерал формуле био би третиран као Булова промењива без узимања у обзир његовог правог значења. Такође, односи између литерала били би занемарени, што је илустровано следећим примером.

Пример 7.2. У примеру 7.1 дефинисана су два ограничења торки:

i_1*: SecEmpNumber > SecSwEngNumber AND SecSwEngNumber > SecDBAdminNumber*

и

i_2*: SecEmpNumber > SecDBAdminNumber*.

Ова два ограничења торке, односно њихови логички изрази, садрже следеће литерале:

- l_1: *SecEmpNumber > SecSwEngNumber*;
- l_2: *SecSwEngNumber > SecDBAdminNumber*; и
- l_3: *SecEmpNumber > SecDBAdminNumber*.

Претпоставимо да је i_1 ограничење торке из подшеме а да је i_2 кореспондентно ограничење торке из интегрисане шеме.

Узимајући у обзир транзитивно својство оператора *веће* у аритметици целих и реалних бројева, може се уочити следећи однос између литерала:

$$l_1 \wedge l_2 \Rightarrow l_3.$$

С друге стране, *SAT* решавач би оператор *веће* третирао као неинтерпретирани Булов предикат с два аргумента и не би био свестан горепоменутог односа. Из тога следи да *SAT* решавач не би могао да закључи да из ограничења торке i_1 следи ограничење торке i_2. Ако претпоставимо даље да је i_1 ограничење торке из подшеме а да је i_2 кореспондентно ограничења торке из интегрисане шеме, употребом *SAT* решавача не бисмо могли да закључимо да је подшема консолидована са шемом у односу на ограничења i_1 и i_2. □

На основу претходно наведеног, у циљу доказа валидности (7.1), аутоматски решавач мора да узме у обзир семантику литерала ограничења торки, односно значење оператора и функција из других теорија. Како тај задатак излази из домена *SAT* проблема, за његово решавање искоришћени су резултати научне дисциплине Задовољивост по модулу теорија (енг. *Satisfiability Modulo Theory*, *SMT*) [53, 54], која се надовезује на доказивање Булове задовољивости. *SMT* алгоритми управо проширују могућности *SAT* алгоритама, тако да могу резоновати над додатним теоријама, као што су:

- линеарна и нелинеарна аритметика над скупом целих и реалних бројева,

- теорија неинтерпретираних функција,
- теорија низова и
- теорија бит-вектора.

По аналогији са *SAT* решавачима, алати који имплементирају *SMT* алгоритме називају се *SMT* решавачима.

Сличне идеје примењене су и у [34, 31], где су *SMT* решавачи искоришћени за доказивање формула које садрже изразе и из других теорија, осим математичке логике.

Постојећи *SMT* решавачи углавном подржавају само доказивање задовољивости логичке формуле. Ипак они се могу употребити за доказивање валидности импликационе формуле ограничења торки, пошто је доказивање валидности дуалан проблем у односу на доказивање задовољивости [54]. Односно, формула је валидна акко њена негација није задовољива. Стога, доказивање валидности (7.1) спроводи се доказивањем незадовољивости негације исте формуле.

Начин функционисања *SMT* решавача и њихове особине представљене су у следећем одељку.

7.1.1 *SMT* решавачи

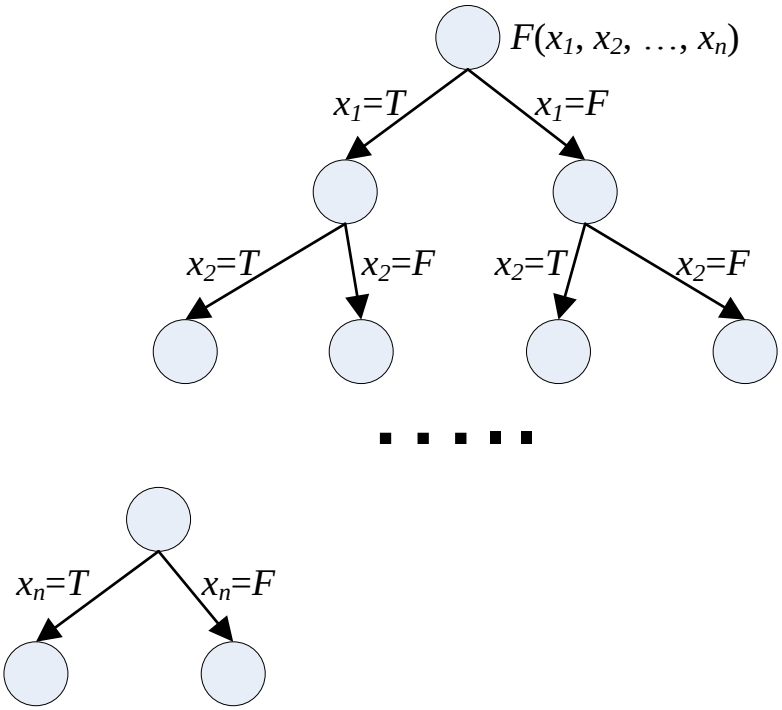
Доказивање задовољивости или валидности логичких формула један је од основних проблема рачунарских наука [53]. Према Хилберту, целокупна класична математика може бити формализована логиком првог реда, па проблем проналажења алгоритма који ће доказати валидност логичке формуле првог реда представља централни проблем математичке логике [55].

За доказивање задовољивости исказних формула и формула логике првог реда користе се *SAT* решавачи. Већина модерних *SAT* решавача заснива се на Дејвис-Патнам-Логемен-Лавленд алгоритму (енг. *Davis-Putnam-Logemann-Loveland*, *DPLL*) [52, 55, 56, 57]. Улаз у *DPLL* алгоритам представља логичка формула у КНФ, где се конјункти формуле називају клаузе а дисјункти унутар конјункта литерали. За логичку формулу $F(x_1, x_2, \ldots, x_n)$ у КНФ, *DPLL* алгоритам прави бинарно стабло претраге, приказано на слици 7.4, где на нивоу стабла i, сваки чвор представља једну могућу комбинацију додељених вредности промењивама $(x_1, x_2, \ldots, x_i)$. Односно, у сваком чвору на нивоу i, промењивој x_i додељују се обе могуће вредности, тачно и нетачно, чиме се праве две излазне гране ка два подређена чвора. У сваком листу стабла, свакој промењивој x_i из F додељена је једна од могућих вредности, што се назива једна интерпретација формуле F. Број листова једнак је броју могућих интерпретација формуле F.

DPLL алгоритам завршава се кад пронађе интерпретацију која задовољава формулу F, односно за коју формула F даје тачно као резултат. Како би се избегло прављење целог стабла претраге, *DPLL* алгоритам примењује две технике оптимизације:

- Пропагација једночланих клауза (енг. *unit propagation*). Ако клауза садржи само једну промењиву, он може имати само једну истинитосну вредност да би цела формула била задовољена. Из тог разлога, гране стабла претраге у којима се посматраној промењивој додељује друга истинитосна вредност не развијају се.

- Елиминација једнополних литерала (енг. *pure literal elimination*). Ако се један литерал, тј. промењива формуле *F*, појављује увек са или увек без негације испред, њој се унапред може доделити истинитосна вредности тако да све клаузе у којима се појављује буду задовољене.



Слика 7.4. DPLL бинарно стабло претраге

У унапређеним верзијама *DPLL* алгоритма, додати су нови кораци за оптимизацију претраге:

- извођење закључака из формуле *F* и одбацивање редудантних клауза (енг. *learning and forgetting*),
- избор следећег литерала којем ће бити додељене истинитосне вредности (енг. *decide next branch*) употребом хеуристике уместо случајног одабира,
- анализа конфликата и одређивање нивоа стабла претраге од којег ће се претрага наставити (енг. *conflict-driven backjumping*) у случају проналажења интепретације која не задовољава формулу и
- отпочињање претраге испочетка са доделом супротне истинитосне вредности првој промењивој (енг. *restarting*).

Формалне дефиниције и детаљи корака за оптимизацију претраге могу се наћи у [54, 52, 58]. Оцена комплексности *DPLL* алгоритма, у најгорем случају када је потребно изградити цело стабло претраге, је експоненцијална $O(2^n)$.

SAT решавачи доказују задовољивост формуле у оквиру логике првог реда, не узимајући у обзир значење предиката и функција употребљених у формули, а које постоје у оквиру неке друге математичке теорије. На пример, *SAT* решавач операцију аритметичког сабирања или релацију ≤ посматраће као неинтерпретиране предикате, редом $P(a,b)$ и $Q(a,b)$.

С циљем да се и знање из осталих теорија, поред математичке логике првог реда, искористи за доказивање задовољивости логичких формула, покренут је развој научне дисциплине „Задовољивост по модулу теорија“ (енг. *Satisfiability Modulo Theories, SMT*). При томе, теорија која се користи за извођење закључака уз логику првог реда назива се, у *SMT* терминологији, позадинска теорија а знање које се из ње користи представљају њене аксиоме и теореме. Према [54], први радови на ову тему су настали крајем 70-их и почетком 80-их година 20. века а пионирима у области се сматрају Нелсон (енг. *Nelson*) и Опен (енг. *Oppen*) са радовима [59, 60], Шостак (енг. *Shostak*, [61, 62, 63]), и Бојер (енг. *Boyer*) и Мур (енг. *Moore*) са радовима [64, 65]. Према аналогији са *SAT* решавачима, софтверски алати који имплементирају *SMT* алгоритме названи су *SMT* решавачи (енг. *SMT solver*).

Различити *SMT* решавачи подржавају различит скуп позадинских теорија. Углавном, модерни *SMT* решавачи могу да резонују над следећим позадинским теоријама [53]:

- линеарна аритметика над скупом целих и реалних бројева,
- нелинеарна аритметика над скупом реалних бројева,
- теорија неинтерпретираних функција,
- теорија низова и
- теорија бит-вектора.

За потребе ове дисертације од интереса су прве три наведене теорије јер се изрази који припадају овим теоријама могу појавити у логичким изразима ограничења вредности. С друге стране, то не важи за изразе теорије низова и теорије бит-вектора па те могућности *SMT* решавача нису од интереса за овај рад.

Две основне стратегије које примењују *SMT* решавачи су проактивна (енг. *eager*) и лења (енг. *lazy*).

Проактивна стратегија састоји се из два корака:

1. превођење оригиналне формуле F_{orig}, која садржи изразе позадинских теорија у формулу F_{Bool}, која садржи само логичке предикате и промењиве и
2. провера задовољивости резултујуће формуле F_{Bool} произвољним и немодификованим *SAT* решавачем.

Кроз корак 1, изрази позадинских теорија замењују се логичким промењивама и предикатима. При томе, информација која постоји у изразима позадинских теорија и односи између израза морају бити пренешени у резултујућу формулу. То се постиже додавањем нових клауза у формулу F_{Bool} који описују однос између уведених логичких промењивих и предиката, што је описано у следећем примеру.

Пример 7.3. Ако се у формули F_{orig} појављују следећа два израза позадинске теорије, у овом случају аритметике целих бројева:

1. $a > b$ и
2. $a + 1 > b$,

приликом прављења F_{Bool}, потребно је изразе заменити логичким променљивама p и q, редом, и у формулу F_{Bool} додати нову клаузу

$$p \Rightarrow q$$

која описује однос између израза позадинске теорије који произилази из саме позадинске теорије. □

Предност проактивне стратегије је могућност коришћења постојећих *SAT* решавача, без потребе да се они прилагоде употреби *SMT* решавача. С друге стране, превођење оригиналне формуле у исказну формулу може да резултује великим бројем додатних логичких промењивих и предиката, као и нових клауза које служе да се представе односи између израза позадинске теорије [54]. Последица повећања броја промењивих, предиката и клауза резултује дужим временом извршавања *SAT* решавача и у екстремним случајевима неуспелим извршавањем због меморијских или временских ограничења.

Из тог разлога, модерни *SMT* решавачи примењују лењу стратегију која представља интеграцију посебних решавача позадинских теорија, надаље *T*-решавача, у *SAT* решавач. Задатак *T*-решавача је да провери да ли је скуп израза позадинске теорије задовољив са аспекта саме теорије и резултат провере врати *SAT* решавачу.

Постоји неколико начина интеграције *SAT* и *T*-решавача, а основни приступ је да се *SAT* решавач покрене над исказном формулом F_{Bool} која је добијена од оригиналне формуле F_{orig} тако што је сваком јединственом литералу, тј. изразу позадинске теорије, додељена различита Булова промењива. Свака тачна интерпретација F_{Bool} одређује за сваки израз позадинске теорије из F_{orig} да ли треба да буде тачан или не. Тако одређен скуп израза позадинске теорије се даље прослеђује специјализованим *T*-решавачима. Уколико сви позвани *T*-решавачи закључе да су изрази задовољиви, F_{orig} је такође задовољива.

У напреднијем приступу интеграцији *SAT* и *T*-решавача, названим *online, SMT* решавач користи проширени *DPLL* алгоритам, назван *T-DPLL*, који у току свог извршавања позива *T*-решавач и пре него што је пронађена тачна евалуација формуле F_{Bool}.

Како би проширени *DPLL* алгоритам могао бити представљен, потребно је увести појам Булове апстракције. Израз који садржи елементе који припадају позадинској теорији могуће је претворити у чисто логички израз тако што ће сваки различит израз позадинске теорије бити замењен посебном логичком промењивом. Добијени логички израз назива се Булова апстракција оригиналног израза. На пример, Булова апстракција израза

$$A+B>7 \text{ AND } B*C<123$$

је израз

$$p \text{ AND } q,$$

где су p и q логичке промењиве, и при томе p замењује $A+B>7$ а q замењује израз $B*C<123$. Саме промењиве p и q су такође Булове апстракције израза које замењују.

Псеудо-кôд проширеног *DPLL* алгоритма [54] дат је на слици 7.5. У алгоритму, употребљене промењиве и помоћне функције имају следеће значење:

- φ – логичка формула са изразима позадинске теорије, чија се задовољивост доказује,
- μ – скуп литерала позадинске теорије којима је *T-DPLL* алгоритам одредио истинитосну вредност,
- *satResult* – индикатор који говори да ли је формула задовољива,
- φ^p – Булова апстракција формуле φ, тј. чисто логичка формула која је добијена

од φ тако што је сваки литерал позадинске теорије замењен логичком промењивом,

- μ^p- Булова апстракција скупа μ, тј. скуп логичких промењивих у које су пресликани литерали позадинске теорије из скупа μ,
- *blevel* – ниво стабла претраге од којег треба наставити претрагу у случају детекције конфликта,
- T2B – функција која логичку формулу са изразима позадинске теорије преводи у Булову апстракцију и
- B2T – инверзна функција $T2B^{-1}$, која добијену Булову апстракцију враћа у оригиналну логичку формулу која садржи изразе позадинске теорије.

```
PROCESS T-DLPP ( I(φ, μ), O(satResult), IO () )
BEGIN PROCESS T-DLPP
  IF(T-Preprocess(φ, μ) = Conflict) THEN
    SET satResult ← Unsat
  ELSE
    SET φ^p ← T2B(φ)
    SET μ^p ← T2B(μ)

    WHILE (True) DO

      CALL T-DecideNextBranch(φ^p, μ^p)
      WHILE (True) DO
        SET status ← T-Deduce(φ^p, μ^p)
        IF (status = Sat) THEN
          SET μ ← B2T (μ^p)
          SET satResult ← Sat
          EXIT
        ELSIF (status = Conflict) THEN
          blevel ← T-AnalyzeConflict(φ^p, μ^p)
          IF (blevel = 0) THEN
            SET satResult ← Unsat
            EXIT
          ELSE
            CALL T-Backtrack(blevel, φ^p, μ^p)
          END IF
        ELSE
          BREAK
        END IF
      END WHILE

    END WHILE
  END IF
END PROCESS T-DLPP
```

Слика 7.5. Проширени DPLL алгоритам

Основни кораци алгоритма изводе се кроз позив неколико процедура које су

описане у даљем тексту.

Процедура T-Preprocess поједностављује формулу φ користећи аксиоме и теореме позадинске теорије, као и методе претпроцесирања *SAT* решавача. При томе, алгоритам може да неким литералима одреди истинитосну вредност која мора бити додељена да би формула била T-задовољива. Такви литерали, уписују се у скуп μ.

Процедура T-DecideNextBranch бира следећи литерал којем ће алгоритам доделити истинитосну вредност.

Процедура T-Deduce изводи закључке о истинитосним вредностима још увек неодређених литерала, на основу делимично одређене формуле φ^p. Када пронађе тачну интерпретацију формуле φ^p, алгоритам позива T-решавач над скупом μ. У супротном, формула φ је незадовољива. Ако T-решавач буде позван, његов резултат је уједно и резултат процедуре T-Deduce.

Задатак процедуре T-AnalyzeConflict је да идентификује скуп литерала η због којих парцијална интерпретација логичке формуле не води до тачне интерпретације целе формуле и на основу тога одреди чвор у *T-DPLL* стаблу одакле треба наставити претрагу. Да претрага не би дошла до истог стања, алгоритам Буловој апстракцији φ^p додаје клаузу $\neg\eta^p$, где η^p означава Булову апстракцију конфликтног скупа литерала η.

Алгоритам *T-DPLL* суштински разликује се од класичног *DPLL* алгоритма у два аспекта [5]:

1. приликом извођења истинитосних вредности неодређених литерала користи се и знање позадинских теорија, а не само теореме логике првог реда и
2. аксиоми и теореме позадинских теорија користе се у детекцији конфликата између литерала, односно незадовољивих клауза.

За описани *online* приступ лење стратегије такође постоји неколико метода убрзања и повећања ефикасности алгоритма:

- евалуација парцијално одређене исказне формуле F_{Bool} употребом *T*-решавача која омогућава да се неконзистенти искази позадинске теорије идентификују раније (енг. *early pruning*), што представља проширење T-Deduce процедуре,
- идентификација израза позадинске теорије који је довео до неконзистентности и наставак претраге са негираним изразом од места где је изразу додељена вредности које је довела до конфликта (енг. *T-learning* и *T-backjumping*), које је унапређење T-AnalyzeConflict процедуре,
- идентификација неконзистентних израза позадинске теорије пре покретања *SAT* решавача, како би се *SAT* решавач унапред обучио да пронађени неконзистентни изрази не могу бити истовремено тачни (енг. *static learning*), у склопу процедуре T-Preprocess и
- употреба *T*-решавача за одређивање истинитосне вредности израза којима *SAT* – решавач још није доделио вредност (енг. *T-propagation*), која се имплементира као проширење T-Deduce процедуре.

Детаљан опис алгоритма лењог *SMT* решавача и унапређења алгоритма може се пронаћи у [54].

7.1.2 Остали разматрани приступи решавању импликационог проблема ограничења торки

За потребе решавања импликационог проблема ограничења торки разматрани су такође алати и језици за аутоматско доказивање теорема (енг. *Automated theory proving, ATP*): *Prolog* [66], *Prover9* [32] и *Vampire* [67, 68]. Као и *SMT* решавачи, ови алати имају за циљ да докажу валидност логичке формуле над скупом свих могућих вредности промењивих које се јављају у формули.

Међутим, наведени алати или не садрже потребно знање из позадинских теорија, попут аксиома линеарне аритметике, или је та функционалност тек у зачетку развоја као што је у случају алата *Vampire* [68]. Стога, изабрани алат за аутоматско доказивање теорема би морао да се обогати потребним аксиомама и теоремама из позадинских теорија, а резултат тога би практично био развој још једног *SMT* решавача, што није био основни циљ овог истраживања.

Такође, разматран је језик *Constraint Handling Rules* (*CHR*, [69, 70]), који спада у класу језика за логичко програмирање са ограничењима [71] (енг. *constraint logic programming*). *CHR* програм представља скуп правила за извођење нових и одбацивање постојећих ограничења, које дефинише програмер за конкретан проблем. Улазни подаци *CHR* програма су скуп иницијалних ограничења и опционо циљно ограничење за које се проверава да ли је последица иницијалног скупа. Приликом извршавања, *CHR* програм генерише нова ограничења користећи дефинисана правила извођења и завршава се са једним од следећа три исхода:

- циљно ограничење је изведено,
- изведено је уграђено ограничење *false*, које означава да почетни скуп ограничења није конзистентан, или
- ако су употребом правила редукције избачена сва ограничења на која могу да се примене правила извођења, односно када ниједно од дефинисаних правила није применљиво на преостали скуп ограничења.

CHR програм би могао бити употребљен за решавање импликационог проблема ограничења торки тако што би ограничење торке из подшеме било задато као иницијално ограничење док би одговарајуће ограничење из интегрисане шеме било циљно ограничење. Међутим, као и у случају алата за аутоматско доказивање теорема, аксиоми и теореме позадинских теорија би морале да буду ручно имплементиране у форми правила за извођење нових ограничења, што би се опет свело на имплементацију још једног *SMT* решавача.

*7.2 Интеграција SMT решавача у IIS*Case*

У циљу обезбеђења провере консолидованости подшема са интегрисаном шемом базе података, *SMT* решавач је интегрисан у *IIS*Case*. Интеграција је спроведена на начин што се негација импликационе формуле ограничења торки (7.1) преведе у облик и језик разумљив *SMT* решавачу и упише у текстуалну датотеку. Та датотека представља улазни податак за *SMT* решавач који се покрене из *IIS*Case*-а као екстерни процес. *SMT* решавач покуша да докаже задовољивост негације (7.1) и ако не успе, (7.1)

проглашава се валидном формулом. Резултат доказивања задовољивости *SMT* решавач упише у излазну текстуалну датотеку коју *IIS*Case* парсира.

Креирање садржаја улазне датотеке *SMT* решавача састоји се из следећих корака:

1. трансформација негације (7.1) у КНФ,
2. претпроцесирање негације (7.1) како би се из формуле избацили оператори и функције који нису подржани од стране *SMT* решавача и
3. трансформације претпроцесиране формуле у језик који разуме *SMT* решавач.

Ови кораци детаљно су описани у даљем тексту.

7.2.1 Трансформације негације импликационе формуле у КНФ

Сви испитани *SMT* решавачи захтевају да улазна формула чија се задовољивост испитује буде дата као сет клауза, где свака клауза одговара једном конјункту улазне формуле у КНФ облику.

Пошто је потребно доказати незадовољивост негације (7.1), што је:

$$\neg\ (i_{SS} \Rightarrow i_S), \tag{7.2}$$

ову формулу је потребно трансформисати у КНФ:

$$i_{SS} \wedge \neg i_S. \tag{7.3}$$

Надаље, формуле i_{SS} и $\neg i_S$ замене се са својим КНФ облицима како би се добио КНФ од (7.2), односно сет клауза које ће бити улаз за *SMT* решавач.

7.2.2 Претпроцесирање оригиналне формуле

Модерни *SMT* решавачи подржавају велики број позадинских теорија [54]. Ипак, није пронађен ниједан *SMT* решавач који подржава операције и функције над датумима, стринговима и скуповима. Пошто су те операције и функције дозвољене у логичким изразима ограничења торки, литерали који их садрже морају бити уклоњени из формуле (7.3). При томе, потребно је очувати задовољивост полазне формуле и што више информација из одбачених литерала и односе између њих. Ово је идеја и проактивних *SMT* решавача [54].

Стога, литерали који садрже операције над датумима, стринговима и скуповима трансформишу се у Булове промењиве, на начине који су описани у наставку поглавља.

7.2.2.1 Трансформација литерала који садрже промењиве типа датум

Литерали који садрже промењиве типа датума и операције над датумима задржавају исти оператор. Са друге стране, промењиве типа датума се трансформишу у целобројне промењиве, задржавајући исто име, а константе типа датума се замењују бројем милисекунди од 1. јануара 1970. године. На овај начин, литерали који садрже датумске промењиве и константе трансформишу се у изразе из домена целобројне аритметике.

Пример 7.4. Литерал

DOB > ToDate('1969-01-01')

трансформише се на овај начин у литерал

DOB > -31536000000. □

Слика 7.6 приказује псеудо-кôд алгоритма за трансформацију литерала који садрже промењиве типа стринг. У алгоритму користе се следеће функције, које нису представљене у досадашњем тексту:

- GetLiterals – функција која враћа скуп литерала ограничења торке које је задато као улазни аргумент функције,
- Type – функција која враћа тип задатог параметра,
- GetAttributes – функција која враћа скуп атрибута прослеђеног литерала и
- GetConstants – функција која враћа скуп константи прослеђеног литерала.

```
PROCESS TransformDateLiterals ( I() , O(), IO (i_S, i_SS) )
BEGIN PROCESS TransformDateLiterals

  DO TransformDateLiterals ( ∀ lit ∈ GetLiterals(i_S) ∪ GetLiterals(i_SS))

    SET litAttrs ← GetAttributes (lit.leftOperand) ∪ GetAttributes (lit.rightOperand)
    DO TransformDateAttributes ( ∀ attr ∈ litAttrs )
      IF (Type(attr) = Date) THEN
        SET Type(attr) ← Int
      END IF
    END DO TransformDateAttributes


    SET litConstants← GetConstants(lit.leftOperand) ∪ GetConstants(lit.rightOperand)
    DO TransformDateConstants ( ∀ const ∈ litConstants)
      IF (Type(const) = Date) THEN
        SET const ← ToInt(const)
        SET Type(const) ← Int
      END IF
    END DO TransformDateAttributes

  END DO TransformDateLiterals
END PROCESS TransformDateLiterals
```

Слика 7.6. Алгоритам за трансформацију литерала који садрже промењиве типа датум

7.2.2.2 Трансформација литерала који садрже промењиве типа стринг

Литерали који садрже стринг промењиве трансформишу се у Булове промењиве путем следећих корака који се спроводе у датом редоследу:

Корак 1. Сваки пар различитих литерала l_i и l_j трансформише се у логичке промењиве p_i и p_j, редом, и формула (7.3) проширује се конјунктом

$$p_i \Rightarrow p_j, \text{ тј., } \neg p_i \vee p_j,$$

акко и l_i и l_j садрже оператор LIKE и l_j може да се изведе из l_i према следећем услову.

Литерал l_j може бити изведен из l_i акко важи следећи однос између десног операнда ro_i од l_i и десног операнда ro_j од l_j. Нека је s_i низ стрингова који се добија раздвајањем ro_i по карактеру '%' и нека је s_{ik} k-ти члан тог низа. Нека су s_j и s_{jk} дефинисани аналогно за ro_j. Литерал l_j следи из l_i ако:

- низови s_i и s_j имају исте дужине n,
- s_{i1} је једнак или проширује s_{j1} са десне стране,
- s_{in} је једнак или проширује s_{jn} са леве стране и
- сваки s_{jk} је подстринг од s_{ik}, $k \in \{2, 3, ..., n\text{-}1\}$. □

Корак 2. Сваки литерал l_i који садржи стринг промењиву, а није процесиран у кораку 1, трансформише се у Булову промењиву на један од следећа два начина:

- ако постоји идентичан литерал l_j, који је процесиран у кораку 1 и раније у овом кораку, l_i трансформише се у Булову промењиву p_j, која је већ додељена литералу l_j или
- ако не постоји идентичан литерал l_j, l_i трансформише се у нову Булову промењиву p_i. □

Пример 7.5. Претпоставимо да су дефинисана следећа ограничења торке и да свако ограничење садржи по један литерал:

$i_1 = l_1$: *NAME* LIKE 'J% D% DOE' и

$i_2 = l_2$: *NAME* LIKE 'JO% DO% DOE'.

Претпоставимо даље да је i_2 ограничење торке из подшеме а i_1 ограничење торке из интегрисане шеме базе података, и да је потребно доказати валидност формуле (7.4):

$$i_2 \Rightarrow i_1, \tag{7.4}$$

односно незадововољивост њене негације (7.5)

$$l_2 \wedge \neg l_1. \tag{7.5}$$

Ако се десни операнди сваког литерала i_k, $k \in \{1,2\}$, разделе по карактеру '%', добијају се следећи низови:

$s_1 = \{$'J', ' D', ' DOE'$\}$ и $s_2 = \{$'JO', ' DO', ' DOE'$\}$.

Анализирамо елементе добијених низова s_1 и s_2. Елемент s_{21} проширује s_{11} са десне стране, елемент s_{23} је идентичан елементу s_{13}, док је елемент s_{22} је подстринг елемента s_{12}. Из ове анализе и горенаведеног корака 1, следи да литерал l_1 следи из литерала l_2, односно да стринг који задовољава ограничење i_2 сигурно задовољава и ограничења i_1.

Стога, литерал l_k, $k \in \{1,2\}$, замењен је Буловом промењивом p_k и (7.5) ће бити трансформисана у формулу:

$$p_2 \wedge \neg p_1 \wedge (\neg p_2 \vee p_1). \tag{7.6}$$

Како (7.6) није задовољива формула, следи да је (7.4) валидна. □

Слика 7.7 приказује псеудо-кôд алгоритма за трансформацију литерала који садрже промењиве типа стринг. У алгоритму користе се следеће промењиве и процедуре, које нису представљене раније у тексту:

- *transLiterals* – мапа која чува већ трансформисане литерале и њима додељене логичке (Булове) промењиве,
- *newPropCnt* – бројач који одређује назив следеће логичке промењиве,
- *allLiterals* – скуп свих литерала ограничења торке из подшеме и ограничења торке из шеме
- ContainsString – предикат који проверава да ли литерал садржи стринг промењиву и
- CheckLikeLiteralInference – функција која проверава да ли се један литерал може извести из другог на основу услова описаног у кораку 1. Псеудо-кôд ове функције дат је на слици 7.8.

```
PROCESS TransformStringLiterals ( I() , O(), IO (iS, iSS) )
BEGIN PROCESS TransformStringLiterals

  SET transLiterals ← ∅
  SET newPropCnt ← 1
  SET allLiterals ← GetLiterals(iS) ∪ GetLiterals(iSS)

  DO IterateLiterals1 ( ∀ li ∈ allLiterals)
    DO IterateLiterals2 ( ∀ lj ∈ allLiterals\ li)
      IF (li.Operator = 'LIKE' AND lj.Operator = 'LIKE') THEN
        IF (CheckLikeLiteralInference (li, lj) THEN
          SET transLiterals ← transLiterals ∪ (li, 'P' + newPropCnt)
          SET li.leftOperand ← 'P' + newPropCnt
          SET newPropCnt ← newPropCnt + 1
          SET li.Operator ← NULL
          SET li.rightOperand ← NULL

          SET transLiterals ← transLiterals ∪ (lj, 'P' + newPropCnt)
          SET lj.leftOperand ← 'P' + newPropCnt
          SET newPropCnt ← newPropCnt + 1
          SET lj.Operator ← NULL
          SET lj.rightOperand ← NULL

          SET iSS ← iSS + 'AND (NOT(' + li.leftOperand + ') OR ' + lj.leftOperand + ')'
        END IF
      END IF
    END DO IterateLiterals2
  END DO IterateLiterals1
```

```
  DO IterateLiterals ( ∀ l_i ∈ allLiterals)
    IF (ContainsString(l_i)) THEN
      IF (transLiterals.Contains(lit_i)) THEN

        SET l_i.leftOperand ← transLiterals.Get(l_i)
        SET l_i.Operator ← NULL
        SET l_i.rightOperand ← NULL

      ELSE

        SET transLiterals ← transLiterals ∪ (l_i, 'P' + newPropCnt)
        SET l_i.leftOperand ← 'P' + newPropCnt
        SET l_i.Operator ← NULL
        SET l_i.rightOperand ← NULL
        SET newPropCnt ← newPropCnt + 1

      END IF
    END IF
  END DO IterateLiterals

END PROCESS TransformStringLiterals
```

Слика 7.7. Алгоритам за трансформацију литерала који садрже промењиве типа стринг

У функцији CheckLikeLiteralInference (слика 7.8) користе се следеће промењиве, функције и процедуре:

- *IND* – индикатор који носи информацију да ли се литерал l_j може извести из литерала l_i,
- Split – функција која дели стринг задат као први аргумент по карактеру који је задат као други аргумент функције,
- Length – функција која враћа дужину прослеђеног низа,
- EqualOrRightExtends – функција чији су параметри два стринга и која враћа као резултат тачну истинитосну вредност ако су стринови идентични или ако први аргумент проширује са десне стране други аргумент,
- EqualOrLeftExtends – функција чији су параметри два стринга и која враћа као резултат тачну истинитосну вредност ако су стринови идентични или ако први аргумент проширује са леве стране други аргумент и
- Substring – функција чији су параметри два стринга и која проверава да ли је први аргумент подстринг другог аргумента.

```
PROCESS CheckLikeLiteralInference ( I(l_i, l_j) , O(IND), IO () )
BEGIN PROCESS CheckLikeLiteralInference

  SET IND ← FALSE
  SET ro_i ← l_i.rightOperand
  SET ro_j ← l_j.rightOperand
  SET s_i ← Split(ro_i, '%')
  SET s_j ← Split(ro_i, '%')
  SET m ← Length(s_i)
  SET n ← Length(s_j)

  IF (m = n AND EqualOrRightExtends(s_i1, s_j1) AND EqualOrLeftExtends(s_in, s_jn)) THEN
    SET IND ← TRUE
    DO CheckInnerParts ( ∀ k ∈ {2,..., n-1})
      IF NOT(Substring(s_j1, s_i1)) THEN
        SET IND ← FALSE
      END IF
    END DO CheckInnerParts
  END IF

END PROCESS CheckLikeLiteralInference
```

Слика 7.8. Функција CheckLikeLiteralInference

7.2.2.3 Трансформација литерала који садрже IN операторе

Литерали који садрже IN операторе, односно операције над скуповима, трансформишу се у Булове промењиве путем следећа два корака, који се спроводе у наведеном редоследу.

Корак 1. Сваки пар литерала l_i и l_j, који садрже IN операторе, трансформише се у Булове промењиве p_i и p_j, редом, и формули (7.3) додаје се конјункт

$$p_i \Rightarrow p_j, \text{ тј., } \neg p_i \vee p_j,$$

акко је десни операнд од l_i подскуп десног операнда од l_j. □

Корак 2. Сваки литерал l_i који садржи IN оператор, а није трансформисан кроз корак 1, трансформише се у Булову промењиву на један од следећа два начина:

- ако постоји идентичан литерал l_j, који је процесиран у кораку 1 или раније у овом кораку, l_i трансформише се у Булову промењиву p_j, која је већ додељена литералу l_j или
- ако не постоји идентичан литерал l_j, l_i трансформише се у нову Булову промењиву p_i. □

Пример 7.6. Посматрајмо два литерала која припадају истој импликационој формули ограничења торки:

$$l_1: X \text{ IN } [1,3,5,7,9] \text{ и } l_2: X \text{ IN } [1,5,9].$$

Пошто је скуп [1,5,9] поскуп од [1,3,5,7,9], корак 1 трансформације литерала који

садрже IN оператор трансформисаће литерал l_k, $k \in \{1,2\}$, у Булову промењиву p_k, а имлпикациона формула ограничења торки биће проширена конјунктом:

$$\neg p_2 \vee p_1. \square$$

Псеудо-кôд алгоритма за трансформацију литерала који садрже операторе *IN* приказан је на слици 7.9. Све промењиве које се користе у псеудо-кôду већ су уведене раније у тексту. С друге стране, у алгоритму користи се функција CheckInLiteralInference која проверава да ли се један литерал може извести из другог на основу услова описаног у кораку 1. Псеудо-кôд функције CheckInLiteralInference дат је на слици 7.10.

```
PROCESS TransformInLiterals ( I() , O(), IO (iS, iSS) )
BEGIN PROCESS TransformInLiterals

  SET transLiterals ← ∅
  SET newPropCnt ← 1
  SET allLiterals ← GetLiterals(iS) ∪ GetLiterals(iSS)

  DO IterateLiterals1 ( ∀ li ∈ allLiterals)
    DO IterateLiterals2 ( ∀ lj ∈ allLiterals\ li)
      IF (li.Operator = 'IN' AND lj.Operator = 'IN') THEN
        IF (CheckInLiteralInference (li, lj) THEN
          SET transLiterals ← transLiterals ∪ (li, 'P' + newPropCnt)
          SET li.leftOperand ← 'P' + newPropCnt
          SET newPropCnt ← newPropCnt + 1
          SET li.Operator ← NULL
          SET li.rightOperand ← NULL

          SET transLiterals ← transLiterals ∪ (lj, 'P' + newPropCnt)
          SET lj.leftOperand ← 'P' + newPropCnt
          SET newPropCnt ← newPropCnt + 1
          SET lj.Operator ← NULL
          SET lj.rightOperand ← NULL

          SET iSS ← iSS + 'AND (NOT(' + li.leftOperand + ') OR ' + lj.leftOperand + ')'
        END IF
      END IF
    END DO IterateLiterals2
  END DO IterateLiterals1

  DO IterateLiterals ( ∀ li ∈ allLiterals)
    IF (li.Operator = 'IN') THEN
      IF (transLiterals.Contains(liti)) THEN

        SET li.leftOperand ← transLiterals.Get(li)
        SET li.Operator ← NULL
        SET li.rightOperand ← NULL
```

```
        ELSE

          SET transLiterals ← transLiterals ∪ (l_i, 'P' + newPropCnt)
          SET l_i.leftOperand ← 'P' + newPropCnt
          SET l_i.Operator ← NULL
          SET l_i.rightOperand ← NULL
          SET newPropCnt ← newPropCnt + 1

        END IF
      END IF
    END DO IterateLiterals

END PROCESS TransformInLiterals
```

Слика 7.9. Алгоритам за за трансформацију литерала који садрже IN оператор

Све промењиве које се користе у функцији CheckInLiteralInference (слика 7.10) имају исто значење као што је претходно назначено у тексту.

```
PROCESS CheckInLiteralInference ( I(l_i, l_j) , O(IND), IO () )
BEGIN PROCESS CheckInLiteralInference

   SET IND ← FALSE
   SET ro_i ← l_i.rightOperand
   SET ro_j ← l_j.rightOperand

   IF (ro_i ⊆ ro_j) THEN
      SET IND ← TRUE
   END IF

END PROCESS CheckInLiteralInference
```

Слика 7.10. Функција CheckInLiteralInference

7.2.3 Превођење формуле у језик разумљив SMT решавачу

SMT је област која се и даље интензивно развија и тренутно ниједан *SMT* решавач није у стању да докаже или оповргне задовољивост сваке логичке формуле. Уколико један *SMT* решавач не успе да докаже задовољивост изабране формуле, постоји могућност да ће неки други успети. Због тога, у *IIS*Case* интегрисана су два решавача: *MathSAT* [72] и *CVC3* [73], и направљена је основа да се нови решавачи додају на једноставан начин.

Сваки *SMT* решавач на улазу очекује *SMT* проблем описан у неком наменском језику за дефинисање ове класе проблема. Један део решавача подржава само наменске језике који су развијени за сваки решавач појединачно. Са друге стране, велики број решавача подржава стандардизовани *SMT-LIB* [74, 75] језик. Овај језик је резултат

међународне иницијативе *SMT* заједнице која има три циља:

- дефинисање прецизног описа позадинских теорија, односно дефинисање скупа аксиома,
- дефинисање референтне библиотеке *SMT* проблема који служе за поређење ефикасности и успешности различитих *SMT* решавача и
- дефинисање језика који би се наметнуо као стандардни језик који разумеју *SMT* решавачи.

Због своје распрострањености, *SMT-LIB* језик је изабран за наменски језик у који *IIS*Case* трансформише импликациону формулу ограничења торки. Тиме је омогућено да се валидност формуле провери са великим бројем решавача. Из те групе, изабран је *MathSAT* због добрих резултата на такмичењу решавача *SMT-COMP 2012* [76].

Улазна *SMT-LIB* датотека састоји се из три секције:

1. секције са декларацијама атрибута и функција које се користе у клаузама,
2. скупа клауза, где свака клауза је један конјункт формуле (7.3) и
3. команде која покреће проверу задовољивости скупа клауза.

Пример 7.7. Посматрајмо ограничења торки која су дефинисана у примеру 7.2 и претпоставимо да је потребно доказати да из

i_1*: SecEmpNumber > SecSwEngNumber AND SecSwEngNumber > SecDBAdminNumber*

следи

i_2*: SecEmpNumber > SecDBAdminNumber*.

За ту намену, *IIS*Case* креира *SMT-LIB* датотеку чији садржај је дат на слици 7.11. Свака секција обележена је коментаром. Према *SMT-LIB* језику, скуп клауза дат је у префиксној нотацији. □

```
;deklaracije promenjivih i funkcija
(declare-fun SecEmpNumber () Real)
(declare-fun SecSwEngNumber () Real)
(declare-fun SecDBAdminNumber () Real)

;skup klauza
(assert (>SecEmpNumber SecSwEngNumber))
(assert (>SecSwEngNumber SecDBAdminNumber))
(assert (not(>SecEmpNumber SecDBAdminNumber)))

;komanda za pokretanje provere zadovoljivosti
(check-sat)
```

Слика 7.11. SMT-LIB опис ограничења торки из примера 7.2

Детаљан опис *SMT-LIB* језика и синтаксе дат је у [74].

CVC3 [77] је наменски језик пројектован за истоимене *SMT* решаваче. Као и *MathSAT*, *CVC3* је изабран због добрих резултата на такмичењу решавача *SMT-COMP 2012* [76].

Док *MathSAT* и остали решавачи који подржавају *SMT-LIB* језик у основи доказују

задовољивост скупа клауза, основна функционалност *CVC3*-а је доказивање валидности логичке формуле уз важење задатих логичких претпоставки. У случају ограничења торки, логичка претпоставка је ограничење торке из подшеме а формула чија валидност се проверава је ограничење торке из интегрисане шеме. Односно, да би *CVC3* био употребљен, не прави се импликациона формула, али се логичке формуле оба ограничења претпроцесирају као што је описано у поглављу 7.2.2.

Улазна *CVC3* датотека састоји се такође из три секције:

4. декларације атрибута и функција које се користе у клаузама,
5. скупа логичких претпоставки, тј. скупа клауза ограничења торке из подшеме и
6. логичке формуле чија се валидност доказује, тј. логичке формуле ограничења торке из интегрисане шеме.

Пример 7.8. Посматрајмо иста два ограничења торки као у примеру 7.7. Да би било доказано да из i_1 следи i_2, *IIS*Case* креира следећу *CVC3* датотеку, чији садржај је дат на слици 7.12. Секције су означене коментарима. □

Формални опис *CVC* језика и синтаксе дат је у [77].

```
%%% deklaracije promenjivih i funkcija
SecEmpNumber: REAL;
SecSwEngNumber: REAL;
SecDBAdminNumber: REAL;

%%% klauze ogranicenja iz podseme - pretpostavke
ASSERT (SecEmpNumber > SecSwEngNumber);
ASSERT (SecSwEngNumber > SecDBAdminNumber);

%%% ogranicenje iz seme - posledica
QUERY (SecEmpNumber > SecDBAdminNumber);
```

Слика 7.12. CVC3 опис ограничења торки из примера 7.2

7.3 Закључак

Путем алгоритама који су представљени у овом поглављу, омогућена је аутоматска провера консолидованости независно моделованих подшема са јединственом шемом базе података са аспекта ограничења вредности. Употребом ових алгоритама, пројектантима информационог система омогућено је да ограничења вредности моделују независно, у оквиру модела делова информационог система, док алат *IIS*Case* води рачуна да имплементација јединствене шема базе података буде исправна и да трансакциони програми, који се заснивају на подшемама, на исправан начин читају податке из јединствене базе података и ажурирају их. На тај начин, олакшан је и убрзан развој информационих система јер развој ограничења вредности од стране различитих пројектаната није више потребно ручно синхронизовати.

Развијени алгоритми узимају у обзир ограничења вредности која су дефинисана у складу са граматикама датим на сликама 4.1, 4.2 и 4.4, уз ограничење да је код оператора LIKE разматрана употреба само специјалног знака „%”, који замењује један

или више знакова. Стога, граматике из поглавља 4, као и алгоритме дате у овом поглављу, могуће је додатно проширити да подрже и следеће изразе у ограничењима вредности:

1. изразе са оператором LIKE и регуларним изразом који садржи специјални знак „_”, који представља ниједан или један, произвољни знак и
2. изразе са операторима ALL, ANY или EXISTS.

Ова проширења представљају могуће правце будућег рада.

8 Закључци и правци даљег истраживања

Савремени приступи у развоју софтвера, како у академском, тако и у привредном окружењу, све више се ослањају на развој софтвера по некој од *MDSD* методологија и употребу *MDSD* алата. Ипак, *MDSD* алати често имају врло сиромашну или практично никакву подршку за развој ограничења вредности, која треба да постану саставни део једне шеме базе података неког информационог система. Због таквог стања, ограничења вредности често се изостављају из модела базе података, или чак и из саме имплементације базе података. У том случају, имплементирана база података не проверава пословна правила која је могуће имплементирати само употребом ограничења вредности и омогућава да неко од пословних правила буде нарушено.

У циљу пружања боље подршке за пројектовање и генерисање ограничења вредности, спровођена су истраживања на Факултету техничких наука у Новом Саду, с циљем обезбеђења одговарајуће технике за платформски независно моделовање ограничења вредности [5, 8]. Креирани платформски независни модел ограничења вредности представља део платформски независног модела типова форми, који се користи за моделовање информационих система у *MDSD* алату *IIS*Case* [3].

Ова дисертација представља део тих истраживања и у њој су представљени и формално доказани алгоритми за трансформацију платформски независних модела ограничења вредности у релациони модел података. Развијене трансформације покривају ограничења вредности на нивоу домена, атрибута и типа компоненте, односно на сва три нивоа на којима ограничење вредности може бити специфицирано у оквиру модела типова форми, а резултат трансформација су ограничења вредности дефинисана на нивоу домена, обележја или скупа шема релација у релационом моделу, редом. Према *MDSD* терминологији, развијени алгоритми представљају трансформације платформски независног у платформски зависан модел ограничења вредности, тј. *PIM/PSM* трансформације.

Такође, у оквиру дисертације, развијени су алгоритми и шаблони кôда за аутоматско генерисање извршивог *SQL/DDL* кôда ограничења вредности за стандардизоване и комерцијалне СУБП. *SQL/DDL* кôд генерише се полазећи од

спецификација ограничења вредности у релационој шеми базе података, која је претходно добијена као резултат *PIM/PSM* трансформација. Ови алгоритми представљају трансформацију платформски зависног модела у софтверски кôд, према *MDSD* приступу. При томе, посебна пажња посвећена је перформансама генерисаног кôда.

Уз примену креираних трансформација модела, пројектант има могућност да развија ограничења вредности кроз платформски независне моделе. При томе, имплементација ограничења вредности за конкретну имплементациону платформу добија се аутоматски, а генерисани кôд је оптимизован за изабрану платформу. Аутор ове дисертације очекује да ће овај приступ омогућити пројектантима лакши и бржи развој ограничења вредности у односу на технике које нуде постојећи *CASE* алати, што ће допринети широј употреби ограничења вредности у развоју шема база података и информационих система, уопште.

Развој информационог система је тимски рад и не може бити спроведен од стране само једног пројектанта. Стога, методологија развоја информационих система и њихових база података подржана у *IIS*Case*-у подразумева независан развоја делова информационог система, тј. подсистема, од стране различитих пројектаната. Сваки модел дела информационог система пресликава се у подшему базе података, док се унија модела трансформише у јединствену шему базе података.

Предуслов за исправну имплементацију моделованог информационог система је да свака подшема базе података буде формално консолидована са јединственом шемом базе података [9]. Из тог разлога, у *IIS*Case*-у од раније постоје имплементирани алгоритми [9] који проверавају наведену консолидованост. Провера консолидованости спроводи се одвојено за сваки тип ограничења шеме базе података и подшема информационих подсистема, моделованих у оквиру спецификације целокупног информационог система. Претходно развијени алгоритми проверавали су консолидованост са аспекта структуре сваке шеме релације, ограничења кључа, ограничења јединствене вредности, ограничења референцијалног интегритета и ограничења инверзног референцијалног интегритета.

У овој дисертацији, развијени су алгоритми за проверу консолидованости подшема са генералног аспекта ограничења вредности, и посебно ограничења торки. У том случају, консолидација значи да свако ограничење торке из интегрисане шеме базе података, мора у подшеми имати једнако строго или строже ограничење торке ако подшема моделује податке које референцира посматрано ограничење. Односно, ограничење торке из интегрисане шеме мора да следи из кореспондентног ограничења, укљученог у одговарајуће подшеме.

Доказивање да једно ограничење торке представља логичку последицу других, назива се импликациони проблем ограничења торке и решавање тог проблема је неопходни део провере консолидованости подшеме са јединственом шемом базе података. У овој докторској дисертацији закључено је да овај проблем представља *SMT* проблем и да се може решити употребом *SMT* решавача [53, 54], док „конвенционалне“, синтаксно оријентисане методе решавања импликационог проблема за овај тип ограничења, као што је случај за нпр. функционалне зависности и кључеве, нису применљиве. Из тог разлога, у дисертацији дефинисане су трансформације импликационе формуле платформски независних спецификација ограничења торки у захтевани облик, а затим и у захтеване наменске језике, које подржавају *SMT* решавачи.

Да би *SMT* решавач био употребљен за доказивање валидности импликационе

формуле ограничења торки, формула мора бити преведена у скуп клаузула, а литерали који садрже операције над промењивама типа стринг и литерали који садрже IN оператор, морају бити трансформисани у Булове промењиве. Из тог разлога, у дисертацији дефинисане су трансформације које поменуте типове литерала преводе у Булове промењиве уз очување информације из позадинске теорије коју литерали носе.

Након трансформација у циљни облик, потребно је импликациону формулу трансформисати даље у наменске језике разумљиве *SMT* решавачима. При томе, изабрани су стандардизовани *SMT-LIB* језик [74] и *CVC3* језик [77].

Како ниједан постојећи *SMT* решавач не може да реши произвољни *SMT* проблем, тј. све могуће *SMT* проблеме, избор *SMT-LIB* језика као циљног језика за трансформацију импликационе формуле ограничења торки омогућава да се импликациони проблем ограничења торки покуша решити употребом више од једног решавача.

Са друге стране, *CVC3* решавач [73] не подржава *SMT-LIB* језик али се показао као врло успешан решавач на годишњим такмичењима *SMT* решавача [76]. Из тог разлога, развијена је и трансформација импликационе формуле ограничења торки у наменски језик овог решавача.

Представљени алгоритам за проверу консолидованости гарантује да су моделована ограничења вредности, која су специфицирана на нивоу информационог подсистема, у складу са интегрисаном шемом базе података целог информационог система. Самим тим, пројектанти су ослобођени обавезе да сами брину о консолидованости подшема са аспекта ограничења вредности. На тај начин, унапређен је приступ независног моделовања делова информационог система од стране више пројектаната, који је један од најбитнијих аспеката развоја информационих система у алату *IIS*Case*.

Такође, употребом *SMT* решавача у алгоритму за проверу консолидованости, процес развоја информационих система у *IIS*Case*-у проширен је и једном методом рачунарске интелигенције, што представља помак ка примени модерних приступа моделовања и развоја софтвера [25, 26, 31] и база података [27, 28], где су методе рачунарске интелигенције употребљене у циљу решавања проблема које није било могуће решити употребом класичних метода.

Алгоритам за трансформацију ограничења вредности из платформски независног модела типова форми у релациони модел података и алгоритми за генерисање извршивог кôда за ограничења вредности имплементирани су оквиру *MDSD* алата *IIS*Case*, чиме је омогућена и њихова практична верификација и примена. Имплементација алгоритма за консолидацију подшема са аспекта ограничења торки је у току и први тестови нису указали ни на један проблем са алгоритмом који је предложен у овој дисертацији.

Током израде дисертације, идентификовано је више праваца за даља истраживања.

Један од њих представља омогућавање моделовања ограничења вредности која садрже агрегационе функције. Поред тога, потребно је обезбедити трансформације и, коначно, генерисање кôда за оваква ограничења, у алату *IIS*Case*. Такође, неопходно је проширити алгоритам за проверу консолидованости тако да подржи овако задата ограничења вредности.

Пословна правила неког система могу изискивати таква ограничења вредности чији логички израз садржи агрегациону функцију. Хипотетички пример оваквог ограничења би био да годишња премија радника не сме бити већа од просечне плате у предузећу.

Међутим, на основу досадашњег истраживања доступне литературе, ни у једном стандардизованом или комерцијалном СУБП није подржана употреба агрегационих функција у ограничењима вредности, што представља мотивацију за овај правац даљег истраживања. Разлог за овакво стање би могао бити што се ограничења вредности у СУБП проверавају на нивоу торке док би агрегациона функција захтевала пролаз кроз читаву табелу. Ипак, и сам наведени разлог је претпоставка коју треба додатно проверити.

Иницијална идеја је да се овај проблем може решити генерисањем корисничких функција које ће омогућити израчунавање резултата агрегационе функције. У *SQL/DDL* кôду ограничења вредности би позив агрегационе функције био замењен позивом изгенерисане функције. Тиме би се избегло да ограничење вредности директно позива агрегациону функцију што није ни подржано од стране постојећих СУБП. Овакав приступ би био примењив на *Microsoft SQL Server* платформи, док *Oracle* платформа не дозвољава ни употребу корисничких функција у дефиницији ограничења вредности, уколико таква функција реализује било какве операције упита или ажурирања над базом података. Стога, у случају *Oracle* платформе било би потребно пронаћи алтернативно решење, које би могло да иде у правцу имплементације ограничења вредности употребом механизма окидача.

Слично агрегационим функцијама, комерцијални СУБП не дозвољавају ни употребу угњеждених упита у логичким изразима ограничења вредности. Из тог разлога, било би важно размотрити и начине да се омогући моделовање, генерисање и консолидација оваквих ограничења вредности.

Још један идентификовани правац даљег истраживања је развој наменског *SMT* решавача или проширење неког од постојећих, тако да подржава операције над подацима типа датум, стринг и скуп, у циљу побољшања перформанси алгоритма за проверу консолидованости подшема са аспекта ограничења торки. Тиме би се избегла потреба за претпроцесирањем логичких формула ограничења торки које је описано у поглављу 7.2.2. Слична идеја описана је већ у [30, 78] где је аутор представио специјализовани метод за проверу задовољивости појединих типова ограничења интегритета у објектно-оријентисаним базама података. Аутор се одлучио за развој сопственог метода резоновања због незадовољавајућих перформанси општих алата за аутоматско доказивање теорема [78].

Развој *SMT* решавача специјализованог за ограничења вредности захтевао би исцрпно истраживање из области математичких теорија које би биле подржане решавачем. Циљ истраживања био би да се дефинише одговарајући скуп аксиома и теорема позадинске теорије које ће бити искоришћене као метод резоновања *T*-решавача.

У анализираној литератури представљеној у другом поглављу, пронађено је мало навода на тему развоја ограничења вредности, који уз то недовољно покривају животни циклус ограничења вредности, од платформски независног моделовања до имплементације за конкретну платформу. Додатно, није пронађен ниједан навод који се бави консолидацијом ограничења вредности у релационим базама података. Све ово чини простор за додатна истраживања на ову тему великим и изазовним. Стога, узимајући у обзир потребу за коришћењем ограничења торки у привредном и академском окружењу, може се донети закључак да је даље истраживање и унапређивање алата *IIS*Case* за потребе развоја ограничења вредности оправдано.

9 Литература

[1] Atkinson C, Kuhne T: *Model-Driven Development: A Metamodeling Foundation*. IEEE Software, IEEE Computer Society Press, Los Alamitos, CA, USA, ISSN: 0740-7459, Vol. 20, No. 5, September 2003, pp. 36–41.

[2] Selic B: *The Pragmatics of Model-Driven Development*. IEEE Software, IEEE Computer Society Press, Los Alamitos, CA, USA, ISSN: 0740-7459, Vol. 20, No. 5, September 2003, pp. 19–25.

[3] Luković I, Mogin P, Pavićević J, Ristić S: *An Approach to Developing Complex Database Schemas Using Form Types*. Software: Practice and Experience, John Wiley & Sons, Inc. New York, NY, USA, ISSN: 0038-0644 Vol. 37, No. 15, 2007, pp. 1621-1656.

[4] Бановић Ј: *Један приступ генерисању извршних софтверских спецификација информационог система*. Докторска дисертација, Универзитет у Новом Саду, Факултет техничких наука, Нови Сад, 2010.

[5] Luković I, Popović A, Mostić J, Ristić S: *A Tool for Modeling Form Type Check Constraints and Complex Functionalities of Business Applications*. Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, DOI: 10.2298/CSIS111102003A, ISSN: 1820-0214,Vol. 7, No. 2, April 2010, pp. 359-385.

[6] Луковић И: *Аутоматизовано генерисање подшеме релационе базе података путем типова форми*. Магистарски рад, Универзитет у Београду, Електротехнички факултет, Београд, 1993.

[7] Поповић А: *Један приступ специфицирању извршних модела апликација информационог система*. Докторска дисертација, Универзитет Црне Горе, Природно-математички факултет, Подгорица, 2013.

[8] Мостић Ј: *Спецификација регуларних израза и функција у алату IIS*Case*. Магистарски рад, Универзитет у Новом Саду, Факултет техничких наука, Нови Сад, 2009.

[9] Ристић С: *Истраживање проблема консолидације подшема база података*.

Докторска дисертација, Универзитет у Новом Саду, Економски факултет, Суботица, 2003.

[10] Batini C, Lenzerini M, Navathe S. B: *A Comparative Analysis of Methodologies for Database Schema Integration*. ACM Computing Surveys, ACM, New York, USA, ISSN: 0360-0300, Vol. 18, No. 4, 1986, pp. 323-364.

[11] Luković I, Ristić S, Mogin P, Pavićević J: *Database Schema Integration Process – A Methodology and Aspects of Its Applying*. Novi Sad Journal of Mathematics, Faculty of Science, Novi Sad, Serbia, ISSN: 1450-5444, Vol. 36, No. 1, 2006, pp. 115-150.

[12] Павићевић Ј: *Развој CASE алата за аутоматизовано пројектовање и интеграцију шема база података*. Магистарски рад, Универзитет Црне Горе, Природно-математички факултет, Подгорица, 2005.

[13] Obrenović N, Aleksić S, Popović A, Luković I: *Transformations of Check Constraint PIM Specifications*. Computing and Informatics (CAI), Slovak Academy of Sciences, Institute of Informatics, Bratislava, Slovak Republic, ISSN: 1335-9150, Vol. 31, No. 5, December 2012, pp. 1045-1079.

[14] Obrenović N, Luković I: *An Approach to Consolidation of Database Check Constraints*. 4th International Conference on Information Society Technology and Management (ICIST 2014), Kopaonik, Serbia, March 9-13, 2014, Proceedings, Association for Information systems and Computer networks, Belgrade, Serbia, ISBN: 978-86-85525-14-8, pp. 210-215.

[15] Demuth B, Hussmann H: *Using UML/OCL Constraints for Relational Database Design*. 2nd International Conference on The Unified Modeling Language - Beyond the Standard (UML 99), Fort Collins, CO, USA, October 1999, Proceedings, Springer, Berlin, Heidelberg, Germany, ISBN: 3-540-66712-1, pp. 598-613.

[16] Demuth B, Hussmann H, Loecher S: *OCL as a Specification Language for Business Rules in Database Applications*. 4th International Conference on The Unified Modeling Language - Modeling Languages, Concepts, and Tools (UML 2001), Toronto, Canada, October 2001, Proceedings, Springer, London, UK, ISBN: 3-540-42667-1, pp. 104-117.

[17] Warmer J, Kleppe A: *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, ISBN: 0-201-37940-6, 1999.

[18] Cabot J, Teniente E: *Constraint Support in MDA tools: a Survey*. 2nd European conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA'06), Bilbao, Spain, July 10-13, 2006, Proceedings, Springer, Berlin, Heidelberg, Germany, ISBN: 3-540-35909-5, pp. 256-267.

[19] *Oracle Designer web page*. Доступно на интернету: http://wiki.oracle.com/page/Oracle+Designer, приступљено: септембар 2010.

[20] *Sybase PowerDesigner web page*. Доступно на интернету: http://www.sybase.com/products/modelingdevelopment/powerdesigner, приступљено септембар 2010.

[21] *Enterprise Architect Check Constraints*. Доступно на интернету: http://www.sparxsystems.com/enterprise_architect_user_guide/9.3/database_engineering/check_constraints.html, приступљено: август 2014.

[22] *IBM Rational Rose web page*. Доступно на интернету: http://www-01.ibm.com/software/awdtools/developer/rose, приступљено: септембар 2010.

[23] *CA ERwin Data Modeler web page*. Доступно на интернету: http://erwin.com/products/detail/ca_erwin_data_modeler/, приступљено: новембар 2010.

[24] *Check Constraints - Toad Data Modeler 5.2 – User Guide*. Доступно на интернету: http://documents.software.dell.com/DOC164562, приступљено: август 2014.

[25] Cabot J, Clarisó R, Riera D: *UMLtoCSP: A Tool for The Formal Verification of UML/OCL Models Using Constraint Programming*. 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE '07), Atlanta, GA, USA, November 5-9, 2007, Proceedings, ACM, New York, NY, USA, 2007, ISBN: 978-1-59593-882-4, pp. 547-548.

[26] Cabot J, Clarisó R, Riera D: *Verification of UML/OCL Class Diagrams using Constraint Programming*. IEEE International Conference on Software Testing Verification and Validation (ICSTW '08), Lillehammer, Norway, April 9-11, 2008, IEEE Computer Society Washington, DC, USA, ISBN: 978-0-7695-3388-9, pp. 73-80

[27] Bry F, Decker H, Manthey R: *A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases*. Advances in Database Technology - EDBT'88, International Conference on Extending Database Technology, Venice, Italy, March 14-18, 1988, Proceedings, Springer, Heidelberg, West Germany, 1988, ISBN: 3-540-19074-0, pp. 488-505.

[28] Bidoit M, Colazzo D: *Testing XML Constraint Satisfiability*. Electronic Notes in Theoretical Computer Science (ENTCS), Elsevier Science Publishers B. V., Amsterdam, The Netherlands, ISSN: 1571-0661, Vol. 174, No. 6, June 2007, pp. 45-61

[29] Blackburn P: *Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto*. Logic Journal of the IGPL, Oxford Journals, Oxford, UK, ISSN: 1367-0751, Vol. 8, No. 3, 2000, pp. 339–365.

[30] Formica A: *Satisfiability of Object-Oriented Database Constraints with Set and Bag Attributes*. Information Systems, Elsevier Science Ltd. Oxford, UK, ISSN: 0306-4379, Vol. 28, No. 3, May 2003, pp. 213-224.

[31] Clavel M, Egea M, De Dios M. A. G: *Checking Unsatisfiability for OCL Constraints*. Electronic Communications of the EASST (ECEASST), European Association of Software Science and Technology, Berlin, Germany, ISSN: 1863-2122, Vol. 24, 2009.

[32] *Prover 9 Web Page*. Доступно на интернету: http://www.cs.unm.edu/~mccune/prover9/, приступљено: септембар 2012.

[33] *The Yices SMT Solver*. Доступно на интернету: http://yices.csl.sri.com/, приступљено: јул 2014.

[34] Blanchette J C, Böhme S, Paulson L C: *Extending Sledgehammer with SMT Solvers*. Journal of Automated Reasoning, Springer-Verlag, New York, USA, ISSN:0168-7433, Vol. 51, No. 1, June 2013, pp 109-128.

[35] Алексић С: *Методе трансформација шема база података у обезбеђењу реинжењеринга информационих система*. Докторска дисертација, Универзитет у Новом Саду, Факултет техничких наука, Нови Сад, 2013.

[36] Bernstein P. A: *Synthesizing Third Normal Form Relations from Functional Dependencies*. ACM Transactions on Database Systems, ACM, New York, NY, USA, ISSN: 0362-5915, Vol. 1, No. 4, December 1976, pp. 277-298.

[37] Могин П, Луковић И, Говедарица М: *Принципи пројектовања база података*. Друго издање, Универзитет у Новом Саду, Факултет техничких наука, Нови Сад, 2004.

[38] Luković I: *From the Synthesis Algorithm to the Model Driven Transformations in Database Design*. 10th International Scientific Conference on Informatics (Informatics 2009), Herľany, Slovak Republic, November 23-25, 2009, Proceedings, Slovak Society for Applied Cybernetics and Informatics and Technical University of Košice - Faculty of Electrical Engineering and Informatics, ISBN 978-80-8086-126-1, pp. 9-18, Invited talk.

[39] Völter M, Stahl T, Bettin J, Haase A, Helsen S: "Model Driven Software Development: Technology, Engineering, Management", John Wiley & Sons, Ltd., 2006, ISBN:978-0-470-02570-3.

[40] Object Management Group: *MDA Guide Version 1.0.1*. Доступно на интернету: http://www.omg.org/cgi-bin/doc?omg/03-06-01, приступљено: јануар 2011.

[41] Luković I, Ristić S, Aleksić S, Popović A: *An Application of the MDSE Principles in IIS*Case*. 3rd Workshop of the Special Interest Group "Model Driven Software Engineering" (SIG MDSE 2008), Berlin, Germany, December 11-12, 2008, in the book: Model Driven Software Engineering - Transformations and Tools, Logos Verlag Berlin GmbH, 2009, ISBN: 9783832521875, pp. 85-95.

[42] Beeri C, Bernstein P. A: *Computational Problems Related to the Design of Normal Form Relational Schemas*. ACM Transactions on Database Systems, ACM, New York, NY, USA, ISSN: 0362-5915, Vol.4, No.1, March 1979, pp. 30-59.

[43] Mogin P, Lukovic I, Govedarica M: *Extended Referential Integrity*. Novi Sad Journal of Mathematics, Faculty of Science, Novi Sad, Serbia, ISSN: 1450-5444, Vol. 30, No. 3, 2000, pp. 111-122.

[44] Алексић С: *Један SQL генератор имплементационог описа шеме базе података CASE алата IIS*Case*. Магистарски рад, Универзитет у Новом Саду, Факултет техничких наука, Нови Сад, 2006.

[45] Поповић А: *Спецификација визуелних атрибута и структура пословних апликација у алату IIS*Case*. Магистарски рад, Универзитет у Новом Саду, Факултет техничких наука, Нови Сад, 2006.

[46] Parr T: *Grammars*. Доступно на интернету: http://www.antlr.org/wiki/display/CS652/Grammars, приступљено: јун 2011.

[47] Elmasri R, Navathe S: *Fundamentals of Database Systems*. 4th Edition, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003 ISBN:0321122267.

[48] Могин П, Луковић И: *Принципи база података*. Прво издање, Универзитет у Новом Саду, Факултет техничких наука, Нови Сад, 1996.

[49] Colton S, Gow J: *The Resolution Method*. доступно на интернету: http://www.doc.ic.ac.uk/~sgc/teaching/v231/lecture8.html, приступљено: фебруар 2011.

[50] Mendelson E: *Introduction to Mathematical Logic*. 4th Edition. Chapman & Hall, London, United Kingdom, ISBN: 9780412808302, 1997.

[51] S. A. Cook: *The Complexity of Theorem-Proving Procedures*. 3rd annual ACM symposium on Theory of computing (STOC '71), 1971, Proceedings, ACM, New York, USA, DOI: 10.1145/800157.805047, pp.151–158.

[52] Marić F: *Formalization and Implementation of Modern SAT Solvers*. Journal of Automated Reasoning, Springer-Verlag, New York, USA, ISSN:0168-7433, Vol. 43, No. 1, June 2009, pp. 81-119.

[53] De Moura L, Bjørner N: *Satisfiability Modulo Theories: An Appetizer*. Formal Methods: Foundations and Applications, 12th Brazilian Symposium on Formal Methods (SBMF 2009), Gramado, Brazil, August 19-21, 2009, Revised Selected Papers, Springer, Berlin, Heidelberg, Germany, ISBN 978-3-642-10451-0, pp. 23-36.

[54] Barrett C, Sebastiani R, Seshia S. A, Tinelli C: *Satisfiability Modulo Theories*. Chapter 26, in book Biere A, Heule M, Van Maaren H, Walsh T: *Handbook of Satisfiability*. IOS Press, Amsterdam, The Netherlands, February, 2009, ISBN 978-1-58603-929-5, pp. 825-885.

[55] Davis M, Putnam H: *A Computing Procedure for Quantification Theory*. Journal of ACM, ACM, New York, NY, USA, ISSN: 0004-5411, Vol. 7, No. 3, July 1960, pp. 201–215.

[56] Davis M, Logemann G, Loveland D: *A Machine Program for Theorem Proving*. Communications of the ACM, ACM, New York, NY, USA, ISSN: 0001-0782, Vol. 5, No. 7, July 1962, pp. 394–397.

[57] Јаничић П: *Математичка логика у рачунарству*. Четврто издање, Универзитет у Београду, Математички факултет, Београд, 2008.

[58] Марић Ф: *Формализација, имплементација и примене SAT решавача*. Докторска дисертација, Универзитет у Београду, Математички факултет, Београд, 2009.

[59] Nelson G, Oppen D. C: *Simplification by Cooperating Decision Procedures*. ACM Transactions on Programming Languages and Systems, ACM, New York, USA, ISSN: 0164-0925, Vol. 1, No. 2, October 1979, pp. 245–257.

[60] Nelson G, Oppen D. C: *Fast Decision Procedures Based on Congruence Closure*. Journal of the ACM, ACM, New York, USA, ISSN: 0004-5411, Vol. 27, No. 2, April 1980, pp. 356–364.

[61] Shostak R. E: *An Algorithm for Reasoning About Equality*. Communications of the ACM, ACM, New York, USA, ISSN: 0001-0782, Vol. 21, No. 7, July 1978, pp. 583-585.

[62] Shostak R. E: *A Practical Decision Procedure for Arithmetic with Function Symbols*. Journal of the ACM, ACM, New York, USA, ISSN: 0004-5411, Vol. 26, No. 2, April 1979, pp. 351–360.

[63] Shostak R. E: *Deciding Combinations of Theories*. Journal of the ACM, ACM, New York, USA, ISSN: 0004-5411, Vol. 31, No. 1, January 1984, pp. 1–12.

[64] Boyer R. S, Moore J. S: *Integrating Decision Procedures into Heuristic Theorem Provers: A Case Study of Linear Arithmetic*. in book Hayes J. E, Michie D, Richards J, *Machine Intelligence 11*, Oxford University Press, Inc., New York, USA, 1988, ISBN 0-19-853718-2, pp. 83–124.

[65] Boyer R. S, Moore J. S: *A Theorem Prover for a Computational Logic*. 10th International Conference on Automated Deduction (CADE), Kaiserslautern, Germany, July 24–27, 1990, Lecture Notes in Artificial Intelligence No. 449, Springer Berlin Heidelberg, Germany, pp. 1–15.

[66] Хотомски П: *Системи вештачке интелигенције*. Универзитет у Новом Саду, Технички факултет „Михајло Пупин'', Зрењанин, 2006.

[67] *Vampire's Home Page*. Доступно на интернету: http://www.vprover.org/, приступљено: септембар 2012.

[68] Kovacs L, Voronkov A: *First-Order Theorem Proving and VAMPIRE*. 25th International Conference on Computer Aided Verification (CAV'13), St Petersburg, Russia, July 13-19, 2013, Proceedings, Springer, Berlin, Heidelberg, Germany, ISBN: 978-3-642-39798-1, pp 1-35.

[69] Frühwirth T: *Welcome to Constraint Handling Rules.* in book Schrijvers T, Frühwirth T: *Constraint Handling Rules*. Springer, Berlin, Heidelberg, Germany, 2008, ISBN 978-3-540-92242-1, pp. 1–15.

[70] Frühwirth T: *Theory and Practice of Constraint Handling Rules*. The Journal of Logic Programming, Cambridge University Press, Cambridge, UK, Vol. 37, No. 1-3, 1998. pp 95–138.

[71] Frühwirth T, Abdennadher S: *Essentials of Constraint Programming*. Springer-Verlag, New York, USA, ISBN: 3540676236, 2003.

[72] *MathSAT 5 интернет страница*. Доступно на интернету: http://mathsat.fbk.eu/, приступљено: јул 2014.

[73] *CVC3 интернет страница*. Доступно на интернету: http://www.cs.nyu.edu/acsys/cvc3/, приступљено јул 2014.

[74] Barrett C, Stump A, Tinelli C: *The SMT-LIB Standard Version 2.0*. Доступно на интернету: http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.0-r10.12.21.pdf, приступљено: август 2013.

[75] Cok D. R: *The SMT-LIB v2 Language and Tools: A Tutorial*. доступно на интернету: http://www.grammatech.com/resource/smt/SMTLIBTutorial.pdf, приступљено: август 2013.

[76] *SMT-COMP 2012*. Резултати такмичења, доступно на интернету: http://smt-exec.org/, приступљено: јул 2014.

[77] *CVC3 User Manual*. Доступно на интернету http://www.cs.nyu.edu/acsys/cvc3/doc/user_doc.html, приступљено: мај 2014.

[78] Formica A: *Finite Satisfability of Integrity Constraints in Object-Oriented Database Schemas.* IEEE Transactions on Knowledge & Data Engineering, IEEE Educational Activities Department, Piscataway, NJ, USA, ISSN: 1041-4347, Vol.14, No.1, January 2002, pp.123-139.

[79] Rahm E, Bernstein P. A: *A survey of approaches to automatic schema matching*. The VLDB Journal – The International Journal on Very Large Data Bases, Springer-Verlag

New York, Inc., Secaucus, NJ, USA, ISSN: 1066-8888, Vol. 10, No. 4, December 2001, pp. 334-350.

[80] Li W-S, Clifton C: *SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks*. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, ISSN: 0169-023X, Vol. 33, No. 1, April 2000, pp 49-84.

[81] Lee M-L, Ling T. W: *Resolving Constraint Conflicts in the Integration of Entity-Relationship Schemas*. Conceptual Modeling - ER '97, 16th International Conference on Conceptual Modeling, Los Angeles, California, USA, November 3-5, 1997, Proceedings, Springer, ISBN: 3-540-63699-4, pp 394-407.