УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Илија Д. Антовић

АУТОМАТСКО ГЕНЕРИСАЊЕ КОРИСНИЧКОГ ИНТЕРФЕЈСА АПЛИКАЦИЈЕ ЗАСНОВАНО НА СЛУЧАЈЕВИМА КОРИШЋЕЊА

докторска дисертација

Београд, 2015.

UNIVERSITY OF BELGRADE

FACULTY OF ORGANIZATIONAL SCIENCES

Ilija D. Antović

AUTOMATIC USER INTERFACE GENERATION BASED ON USE CASES

Doctoral Dissertation

Београд, 2015.

Ментор:

др Владан Девеџић, редовни професор

Факултет организационих наука, Београд

____

Чланови комисије:

др Синиша Влајић, ванредни професор

Факултет организационих наука, Београд

____

др Драган Бојић, доцент

Електротехнички факултет, Београд

____

Датум одбране: ________________

Овај рад посвећујем сину Иву и својим родитељима.

Предговор

Ова дисертација је рађена у периоду од септембра 2011. до августа 2015. године на Факултету организационих наука у Београду. Идеја на основу које је покренуто истраживање, плод је заједничког рада чланова Лабораторије за софтверско инжењерство. Реализација ове идеје, која је представљена у раду, коришћена је у извођењу неколико софтверских пројеката у оквиру Лабораторије, почевши од 2007. године, како би се практично доказала њена примјењивост. Прве резултате истраживања изложио сам у свом Магистарском раду који сам одбранио у јуну 2010. године. Након тога истраживање је настављено, а резултати су у више наврата објављивани у научним часописима и на конференцијама.

Захваљујем се ментору проф. др Владану Девеџићу на указаном повјерењу и прагматичном приступу коју је показао током израде ове дисертације.

Од почетка рада на овом истраживању, огроман допринос дао је мој колега и пријатељ, ментор мог Магистарског рада, професор Синиша Влајић, коме се захваљујем не само на несебичној стручној и моралној подршци коју је пружао током писања ове тезе, већ и на позитивном и конструктивном односу који међу нама траје већ читавих 15 година, и надам се да ће на исти начин још дуго трајати.

Немјерљиву подршку у изради овог рада пружиле су ми колеге мр Милош Милић, мр Војислав Станојевић, мр Душан Савић и др Саша Лазаревић, које су, заједно са професором Влајићем, својим конструктивним предлозима, али и директним учешћем дале велики допринос у спровођењу овог истраживања, па им се овом приликом најтоплије захваљујем. Пивилегија је бити окружен оваквим људима.

Захваљујем се колегама др Нини Турајлић, др Александру Ђоковићу, др Мирјани Меснер и др Милици Булајић који су у сваком тренутку били отворени и спремни да ми изађу у сусрет и пруже помоћ из свог домена и тиме ми значајно олакшају писање дисертације.

Посебно желим да се захвалим нашим бившим студентима Милици Карић, Стевану Бузејићу, Игору Ћировићу и Тамари Валок, који су показали интересовање и активно учествовали у истраживању што је резултирало низом пројектних и Мастер радова којима Факултет организационих наука и Лабораторија за софтверско инжењерство могу да буду поносни.

Аутоматско генерисање корисничког интерфејса апликације засновано на случајевима коришћења

Резиме

Предмет ове дисертације је аутоматизација процеса пројектовања и имплементације корисничког интерфејса, као и утврђивање веза између софтверских захтјева и будућег корисничког интерфејса апликације узимајући у обзир карактеристике циљаних технологија и типова софтверских система. Утврђени су принципи и карактеристике које треба да буду саставни дио алата за аутоматизацију процеса имплементације корисничког интерфејса – генератора корисничког интерфејса.

Дисертација пружа анализу постојећих приступа развоју корисничког интерфејса, метода и техника прикупљања софтверских захтјева, као и преглед досадашњих покушаја рјешавања проблема аутоматизације, са акцентом на актуелне алате који се могу користити у раним фазама софтверског пројекта. Такође, спроведено је испитивање са циљем да се идентификују преовлађујући ставови носилаца софтверске индустрије – искусних софтверских инжењера, по питању елемената тј. карактеристика које алат за генерисање корисничког интерфејса мора да посједује како би га прихватили и користили у процесу развоја софтвера. Уочене карактеристике су посматране као захтјеви које нови приступ генерисању корисничког интерфејса мора да задовољи како би био прихваћен од стране потенцијалних корисника.

На основу уочених захтјева дефинисан је нови – *SilabUI* приступ за аутоматизацију процеса развоја корисничког интерфејса пословних апликација који омогућава извршење *CRUD* (*Create, Read, Update, Delete*) операција. *SilabUI* приступ успоставља формалне везе између случајева коришћења и корисничког интерфејса апликације које су дефинисане семантички богатим мета-моделом.

Мета-модел омогућава спецификацију софтверских захтјева која обухвата све потребне информације за развој корисничког интерфејса, а у исто вријеме његова једноставност га чини погодним за коришћење у раним фазама софтверског пројекта, јер је у потпуности независан од платформи за извршење и имплементационих технологија за ће кориснички интерфејс бити генерисан.

Мета-модел узима у обзир чињеницу да се један исти кориснички захтјев на нивоу корисничког интерфејса може реализовати на различите начине. Ово је постигнуто дефинисањем неколико шаблона корисничког интерфејса.

Реализована су три начина за формирање конкретних модела на основу дефинисаног мета-модела. Ови модели представљају улазну спецификацију на основу које се врши генерисање програмског кода корисничког интерфејса за различите типове апликација. За потребе дисертације развијен је генератор корисничког интерфејса за *Java Swing* десктоп апликације, као и за *Java Server Faces* веб апликације.

Кључне ријечи: кориснички интерфејс, инжењеринг софтверких захтјева, случај коришћења, генератор кода, мета-модел, шаблони корисничког интерфејса, пословне апликације, генератор корисничког интерфејса

Научна област: Рачунарске науке

Ужа научна област: Софтверско инжењерство

УДК број: 004.5

Automatic User Interface Generation Based on Use Cases

Abstract

The subject of this dissertation is the automation of the process of designing and implementing user interfaces. The dissertation also identifies the connections between software requirements, particularly use cases, and the resulting user interface of the application, taking into account the characteristics of the targeted implementation technologies and application types. The principles and characteristics which should be an integral part of the tool for automating the process of designing and implementing user interfaces – the user interface generator – were defined.

The dissertation provides an analysis of contemporary approaches for user interface development, methods and techniques for software requirements elicitation and specification. In addition, we provide an extensive comparative analysis of the previous attempts to solve the problem of automation, with a focus on current tools which could be used in the early stages of a software project. Also, a survey was conducted in order to identify the predominant opinions of key stakeholders in the software industry – experienced software engineers – regarding the elements and features that tools for generating user interface need in order to be accepted and used in software development. The identified characteristics are regarded as requirements that should be met in a new approach to generating the user interfaces in order to be accepted by potential users.

Based on the identified requirements the new SilabUI approach is defined in order to automate the process of development of the user interface of business applications that

enables the execution of CRUD (Create, Read, Update, Delete) operations. SilabUI approach establishes formal corelations between use cases and application user interface defined by a semantically rich meta-model.

The meta-model allows specification of the software requirements which includes all necessary information for developing user interfaces. At the same time, its simplicity makes it appropriate for use in the early stages of a software project. It is completely platform-independent, and also independent of implementation technology to be used for generated user interface.

The meta-model takes into account the fact that the same user requirement can be implemented in numerous ways on the presentation level, while maintaining the required functionality. This is achieved by defining several user interface templates.

Three different modes of specifying the models refined from the defined meta-model are implemented. These models represent the input specification for generating user interface source code for different application types. For the purpose of the dissertation two user interface generators are developed: generator for Java Swing desktop applications and for the Java Server Faces web applications.

Keywords: user interface, requirements engineering, use case, code generator, meta-model, user interface template, business applications, user interface generator

Research area: Computer science

Research focus: Software Engineering

UDK number: 004.5

1. УВОД 4

1.1. ПРОБЛЕМ 4
1.2. ПРЕДМЕТ 6
1.3. ЦИЉ РАДА И ПОЛАЗНА ХИПОТЕЗА 10
1.4. ОЧЕКИВАНИ ДОПРИНОС 12
1.5. МЕТОДОЛОГИЈА 13
1.6. СТРУКТУРА 15

2. ЗНАЧАЈ ИСТРАЖИВАЊА И ПРЕГЛЕД ПОСТОЈЕЋИХ ПРИСТУПА У РАЗВОЈУ КОРИСНИЧКОГ ИНТЕРФЕЈСА 19

2.1. КОРИСНИЧКИ ИНТЕРФЕЈС 21
2.2. ЗНАЧАЈ (МОТИВАЦИЈА) 23
2.3. ПОСТОЈЕЋИ ПРИСТУПИ У РАЗВОЈУ КОРИСНИЧКОГ ИНТЕРФЕЈСА 26
2.3.1. АЛАТИ ЗА СКИЦИРАЊЕ (SKETCHING TOOLS) 27
2.3.2. АЛАТИ ЗА ВИЗУЕЛНО КРЕИРАЊЕ КОРИСНИЧКОГ ИНТЕРФЕЈСА (GUI BUILDERS) 29
2.3.3. ЈЕЗИЦИ ЗА СПЕЦИФИКАЦИЈУ КОРИСНИЧКОГ ИНТЕРФЕЈСА 31
2.3.4. РАЗВОЈ КОРИСНИЧКОГ ИНТЕРФЕЈСА ЗАСНОВАН НА МОДЕЛИМА И МОДЕЛОМ ВОЂЕНИ РАЗВОЈ (MDD)33

3. ПОСТОЈЕЋИ ПРИСТУПИ У ИНЖЕЊЕРИНГУ СОФТВЕРСКИХ ЗАХТЈЕВА 49

3.1. ТЕХНИКЕ ЗА ПРИКУПЉАЊЕ ЗАХТЈЕВА 54
3.2. ТЕХНИКЕ ЗА СПЕЦИФИКАЦИЈУ ЗАХТЈЕВА 57
3.3. НАЈЧЕШЋЕ ГРЕШКЕ И ПРОБЛЕМИ ВЕЗАНИ ЗА ЗАХТЈЕВЕ 65
3.4. ДОМЕНСКИ МОДЕЛ У КОНТЕКСТУ ЗАХТЈЕВА 68
3.5. ШАБЛОНИ ЗА СПЕЦИФИКАЦИЈУ СЛУЧАЈЕВА КОРИШЋЕЊА 93
3.5.1. ШАБЛОН ЗА СЛУЧАЈ КОРИШЋЕЊА У ПОТПУНО СРЕЂЕНОМ ОБЛИКУ 94
3.5.2. ШАБЛОН ЗА СЛУЧАЈ КОРИШЋЕЊА У ТАБЕЛАРНОМ ОБЛИКУ СА ЈЕДНОМ КОЛОНОМ 96
3.5.3. ШАБЛОН ЗА СЛУЧАЈ КОРИШЋЕЊА У ТАБЕЛАРНОМ ОБЛИКУ СА ДВИЈЕ КОЛОНЕ 98
3.5.4. ШАБЛОН ЗА СЛУЧАЈ КОРИШЋЕЊА ПРЕПОРУЧЕН ОД СТРАНЕ RUP (RATIONAL UNIFIED PROCESS) МЕТОДЕ 100

3.6. ПРЕПОРУКЕ ЗА СПЕЦИФИКАЦИЈУ КОРАКА У СЦЕНАРИЈУ СЛУЧАЈА КОРИШЋЕЊА 101
3.7. КОРИШЋЕЊЕ ПРОТОТИПОВА У ИНЖЕЊЕРИНГУ ЗАХТЈЕВА 106

4. УПОРЕДНА АНАЛИЗА АЛАТА ЗА АУТОМАТИЗАЦИЈУ РАЗВОЈА КОРИСНИЧКОГ ИНТЕРФЕЈСА 111

4.1. ДЕФИНИСАЊЕ КРИТЕРИЈУМА ЗА ПОРЕЂЕЊЕ АЛАТА ЗА АУТОМАТИЗАЦИЈУ РАЗВОЈА КОРИСНИЧКОГ ИНТЕРФЕЈСА 113
4.2. ПРЕГЛЕД КАРАКТЕРИСТИКА ПОСТОЈЕЋИХ АЛАТА ЗА АУТОМАТИЗАЦИЈУ РАЗВОЈА КОРИСНИЧКОГ ИНТЕРФЕЈСА 141
4.2.1. APACHEISIS FRAMEWORK (NAKED OBJECTS) 142
4.2.2. METAWIDGET 149
4.2.3. WEBRATIO 155
4.2.4. ALPHASIMPLE 159
4.2.5. BIZAGI BPM SUITE 162
4.2.6. NETBEANS 166
4.3. РЕЗУЛТАТИ УПОРЕДНЕ АНАЛИЗЕ ПОСТОЈЕЋИХ АЛАТА НА ОСНОВУ ДЕФИНИСАНИХ КРИТЕРИЈУМА 168

5. АНАЛИЗА КОРИСНИЧКОГ ИНТЕРФЕЈСА РАЗЛИЧИТИХ ТИПОВА АПЛИКАЦИЈА У РАЗЛИЧИТИМ ИМПЛЕМЕНТАЦИОНИМ ТЕХНОЛОГИЈАМА 172

5.1. ПРИНЦИПИ ПРОЈЕКТОВАЊА КОРИСНИЧКОГ ИНТЕРФЕЈСА 174
5.1.1. ПАТЕРНИ ПРОЈЕКТОВАЊА КОРИСНИЧКОГ ИНТЕРФЕЈСА 179
5.2. АНАЛИЗА КАРАКТЕРИСТИКА КОРИСНИЧКОГ ИНТЕРФЕЈСА ДЕСКТОП, ВЕБ И МОБИЛНИХ ПОСЛОВНИХ АПЛИКАЦИЈА 195

6. ШАБЛОНИ КОРИСНИЧКОГ ИНТЕРФЕЈСА 204

6.1. FIELD-FORM ШАБЛОН 221
6.2. FIELD-TAB ШАБЛОН 228
6.3. TABLE-FORM ШАБЛОН 231
6.4. TABLE-TAB ШАБЛОН 236
6.5. KNGUI ШАБЛОН 240

7. ПРЕДЛОГ НОВОГ ПРИСТУПА – *SILABUI* 247

7.1. МЕТА-МОДЕЛ СОФТВЕРСКИХ ЗАХТЈЕВА ЗАСНОВАН НА МОДЕЛУ СЛУЧАЈЕВА КОРИШЋЕЊА И ИНФОРМАЦИЈАМА ВЕЗАНИМ ЗА КОРИСНИЧКИ ИНТЕРФЕЈС 254
7.1.1. XML ДЕФИНИЦИЈА МОДЕЛА 261
7.2. АЛАТИ ЗА ФОРМИРАЊЕ УЛАЗНЕ СПЕЦИФИКАЦИЈЕ 286
7.2.1. ДИРЕКТНО ПИСАЊЕ КОДА УЛАЗНЕ СПЕЦИФИКАЦИЈЕ 287
7.2.2. ФОРМИРАЊЕ УЛАЗНЕ СПЕЦИФИКАЦИЈЕ КОРИШЋЕЊЕМ ГРАФИЧКОГ АЛАТА (WIZARD) 287
7.2.3. ФОРМИРАЊЕ УЛАЗНЕ СПЕЦИФИКАЦИЈЕ ВИЗУЕЛНИМ МОДЕЛОВАЊЕМ 297
7.3. АУТОМАТИЗАЦИЈА ПРОЦЕСА ТРАНСФОРМАЦИЈЕ ПРЕДЛОЖЕНОГ МОДЕЛА У ИЗВРШИВИ ПРОГРАМСКИ КОД КОРИСНИЧКОГ ИНТЕРФЕЈСА 300
7.3.1. СТРУКТУРА ГЕНЕРАТОРА 302
7.3.2. СОФТВЕРСКИ ПАТЕРНИ КОРИШЋЕНИ У ПРОЈЕКТОВАЊУ ГЕНЕРАТОРА 306
7.3.3. ДИНАМИКА ПРОЦЕСА ГЕНЕРИСАЊА КОДА КОРИСНИЧКОГ ИНТЕРФЕЈСА 311
7.3.4. ГЕНЕРИСАНИ ПРОГРАМСКИ КОД 314
7.3.5. КОМУНИКАЦИЈА ГЕНЕРИСАНОГ КОРИСНИЧКОГ ИНТЕРФЕЈСА СА АПЛИКАЦИОНОМ ЛОГИКОМ СИСТЕМА 319
7.3.6. ВАЛИДАЦИЈА ПОДАТАКА НА ПРЕЗЕНТАЦИОНОМ НИВОУ 321
7.3.7. ЛОКАЛИШАЦИЈА И ИНТЕРНАЦИОНАЛИЗАЦИЈА 325
7.3.8. ПРОЦЕС РАЗВОЈА КОРИСНИЧКОГ ИНТЕРФЕЈСА SILABUI ПРИСТУПОМ 327

8. ЕВАЛУАЦИЈА ПРЕДЛОЖЕНОГ SILABUI ПРИСТУПА 329

8.1. СТУДИЈСКИ ПРИМЈЕР – РАЗВОЈ ДЕСКТОП АПЛИКАЦИЈЕ КОРИШЋЕЊЕМ *SILABUI* ПРИСТУПА 335
8.2. СТУДИЈСКИ ПРИМЈЕР – РАЗВОЈ ВЕБ АПЛИКАЦИЈЕ КОРИШЋЕЊЕМ *SILABUI* ПРИСТУПА 340

9. ЗАКЉУЧАК 345

10. ЛИТЕРАТУРА 360

11. СПИСАК СЛИКА 375

12. СПИСАК ТАБЕЛА 395

1. УВОД

1.1. Проблем

Полазна тачка у креирању софтверског производа јесте фаза прикупљања софтверских захтјева и резултати ове фазе представљају основу за све остале фазе у животном циклусу софтвера. Из тог разлога спецификација софтверских захтјева (обично представљена моделом случајева коришћења) треба да буде довољно детаљна да пружи потребне информације за пројектовање и имплементацију свих дијелова софтвера – од корисничког интерфејса, преко апликационе логике која учаурује доменску структуру и понашање система, до складишта података. Међутим, чест је случај да спецификација софтверских захтјева не даје довољно информација за пројектовање појединих дијелова система (детаља везаних за кориснички интерфејс, пословна правила и сл.), па се препушта програмеру да на свој начин интерпретира захтјеве и изналази рјешења. Као посљедица тога, добијено рјешење често не задовољава потребе корисника на одговарајући начин. У том смислу, уколико се разматра развој корисничког интерфејса, из перспективе софтверског захтјева ријетко се могу добити информације о изгледу корисничког интерфејса, његовом садржају, врстама компоненти које треба да се користе и сл.

Најчешћи узрок неуспјеха софтверских пројеката јесу недостаци и грешке у корисничким захтјевима [Frost07]. Један од начина за њихово отклањање у раним

фазама јесте прављење прототипова корисничког интерфејса. Међутим, прототипови корисничког интерфејса најчешће нису формално повезани са спецификацијом захтјева па се може десити да настане семантичка неконзистентност између прототипа и спецификације и умјесто повећања прецизности могу да искрсну нови проблеми у разумијевању захтјева [DaCruz10]. Други начин за отклањање недостатака у захтјевима јесте формирање доменског модела система и анализа конзистентности семантике доменског модела са специфицираним захтјевима [Rosemberg07]. Корисник преко корисничког интерфејса ступа у интеракцију са системом и управља његовим стањима. У објектно орјентисаним апликацијама, стања система учаурују се у доменске објекте система који, заједно са везама и ограничењима, чине доменски модел софтверског система. Доменски модел описује статичку структуру софтверског система, тј. основне концепте из домена пословног проблема који се пројектује.

Како случајеви коришћења описују интеракцију између корисника и система, која се реализује преко корисничког интерфејса и има за циљ промјену стања система која су обично учаурена у доменским објектима, веза између ова три дијела система – случајева коришћења, доменског модела и корисничког интерфејса – морала би бити узета у обзир већ у фази прикупљања корисничких захтјева.

До данас је било више покушаја да се направи алат који би аутоматизовао процес имплементације дијела или читавог корисничког интерфејса апликације, али нити један од ових алата није успио да се значајније наметне на тржишту и око себе окупи шири круг корисника [Kennard10]. Анализом ових алата може се утврдити да међу њима постоје значајне разлике. Ове разлике су присутне како у самим полазним основама па све до реализације генерисања корисничког интерфејса. Као полазне основе у постојећим алатима могу се наћи различити модели међу којима су најзаступљенији: модел задатка (Task model), доменски модел (Domain model), кориснички модел (User model), модел дијалога (Dialogue model) и презентациони

модел (Presentation model) [Meixner11] [Schlungbaum96]. Што се тиче начина реализације аутоматског генерисања корисничког интерфејса, поједини алати обезбјеђују динамичко генерисање корисничког интерфејса током извршења апликације, док други предвиђају генерисање програмског кода корисничког интерфејса. Такође, поједини алати омогућавају измјене над генерисаним корисничким интерфејсом, као и неке додатне функционалности као што су флексибилност у начину приказа компоненти, избору компоненти које ће се наћи на корисничком интерфејсу, избору између корисничког интерфејса за десктоп и web апликације, избор између различитих циљних архитектура, док је код других читав интерфејс једнообразан и потпуно дефинисан самим алатом.

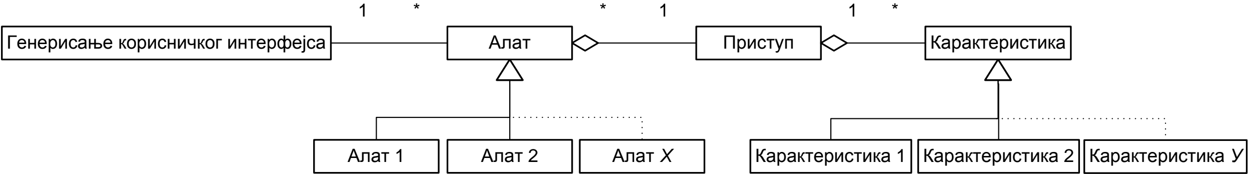
Све ове разлике говоре о томе да још увијек није успостављен широко прихваћен стандард нити јасна правила која би требала да важе приликом генерисања корисничког интерфејса.

1.2. Предмет

У ширем смислу предмет овог истраживања јесте аутоматизација процеса пројектовања и имплементације корисничког интерфејса, као и веза између софтверских захтјева и будућег корисничког интерфејса апликације узимајући у обзир карактеристике циљаних технологија и типова софтверских система. Разматрање овог предмета истраживања има за циљ утврђивање принципа и карактеристика које треба да буду саставни дио будућег алата за аутоматизацију процеса имплементације корисничког интерфејса – генератора корисничког интерфејса. Генератор треба да омогући имплементацију интерфејса у складу са корисничким захтјевима, а у исто вријеме у што већој мјери убрза процес развоја

софтвера, смањи његове трошкове, и омогући његово даље одржавање. При томе корисницима алата генератор треба да омогући слободу избора између различитих циљаних технологија и архитектура.

Како би утврдили који су то принципи и карактеристике које би омогућиле постизање ових задатака, потребно је извршити анализу постојећих алата и приступа који су до сада коришћени и утврдити њихове предности и мане са циљем да се избјегне понављање грешака које су за резултат имале чињеницу да ни један од постојећих алата није стекао значајнију популарност у софтверској индустрији. Зато ће дисертација дефинисати скуп критеријума који ће служити као средство за поређење карактеристика постојећих алата, а касније и за упоређивање алата који ће бити резултат овог истраживања са постојећим алатима (Слика 1). Детаљно ће бити образложен разлог увођења сваког од критеријума, као и предности које се очекују испуњењем сваког од њих.



Слика 1: Однос између алата, приступа и карактеристика различитих приступа за генерисање корисничког интерфејса

Посебна пажња биће посвећена утврђивању веза између корисничких захтјева специфицираних случајевима коришћења са будућим корисничким интерфејсом. Биће разматране везе између структуре (доменски модел) и понашања (системске операције) система, који су изведени из спецификације случајева коришћења, са жељеним корисничким интерфејсом.

Због чињенице да се кориснички интерфејс за један исти кориснички захтјев може реализовати на више различитих начина, биће разматрани кориснички интерфејси неколико софтверских система из различитих домена како би се утврдиле заједничке карактеристике интерфејса и идентификовали најчешће коришћени начини приказа корисничког интерфејса за сличне корисничке захтјеве. Такође ће бити размотрени и основни принципи и препоруке за пројектовање корисничких интерфејса. На овај начин, доћи ће се до скупа шаблона – темплејта корисничких интерфејса, па корисник алата неће бити ограничен на један приказ већ ће моћи да, на брз и једноставан начин, мијења приказ и одабере онај који крајњем кориснику највише одговара. На овај начин постигла би се флексибилност у креирању корисничког интерфејса за исти кориснички захтјев.

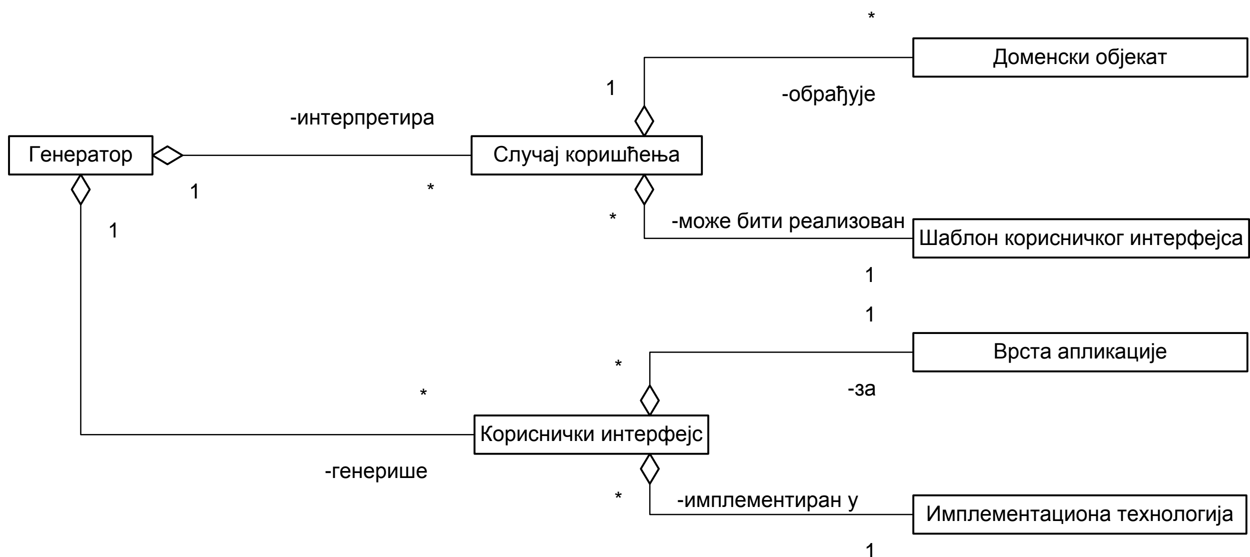
Кориснички интерфејс представља реализацију улаза односно излаза апликационе логике софтверског система. Посматра се као засебан ниво у софтверском систему, али чини његов саставни дио. Данас постоји много различитих типова софтверских система – апликација које се најгрубље могу подијелити на веб, десктоп и мобилне апликације. За прављење сваке од ових апликација развијено је мноштво технологија и платформи за извршење. Свака од ових технологија пружа могућност прављења корисничког интерфејса. Алат за аутоматизацију процеса имплементације корисничког интерфејса треба да пружи могућност одабира између различитих циљних технологија. Како технологија има много, не треба имати амбицију прављења алата који би реализовао генерисање интерфејса за све технологије, али би архитектуру генератора требало пројектовати тако да буде проширива коришћењем софтверских патерна [GOF], како би се за сваку нову технологију могла независно имплементирати компонента која се једноставно може интегрисати у алат (*pluggable component*). На тај начин, пружила би се флексибилност корисницима алата у избору циљане технологије за имплементацију корисничког

интерфејса. Да би се ово остварило, претходно треба утврдити „компатибилност“ графичких компоненти и идентификованих шаблона корисничког интерфејса између различитих типова апликација. Нпр. ако је један шаблон корисничког интерфејса одговарајући за десктоп апликацију, не мора да значи да ће да буде адекватан и за web или мобилну апликацију, најчешће због технолошких ограничења. Зато се мора извршити анализа карактеристика сваке врсте апликација и дефинисати јасна трансформација између основног скупа компоненти и шаблона и изведених шаблона који више одговарају одређеној врсти апликације.

Посебна пажња у истраживању биће посвећена структури генерисаног програмског кода. Биће разматрани различити софтверски патерни пројектовања како би се идентификовало који су патерни најпогоднији за програмски код генерисаног корисничког интерфејса, како би се обезбиједило његово лакше одржавање.

Како би оваква аутоматизација била могућа, потребно је дефинисати семантички богат модел корисничких захтјева који ће бити обогаћен информацијама везаним за кориснички интерфејс. На тај начин омогућиће се генерисање извршивог кода корисничког интерфејса независно од циљане технологије или типа апликације (десктоп апликације у различитим технологијама, web апликације у различитим технологијама итд.). Идеја потиче од чињенице да кориснички захтјев садржи информацију о корисниковим намјерама, функционалностима и циљевима које жели да постигне коришћењем система, без обзира на начин имплементације тог система. Алат који ће вршити аутоматизацију биће пројектован тако да омогући трансформацију модела у извршиви програмски код корисничког интерфејса за различите имплементационе технологије.

Однос између концепата везаних за предмет истраживања приказан је на следећем дијаграму (Слика 2).



Слика 2: Однос између генератора програма, случајева коришћења и корисничког интерфејса

1.3.Циљ рада и полазна хипотеза

Циљ овог рада је да се развије модел за спецификацију корисничког захтјева који ће бити довољно семантички богат како би на основу њега било могуће не само пројектовање и имплементација корисничког интерфејса, већ и аутоматизација овог процеса. Циљ је да модел на основу којег ће се вршити генерисање програмског кода корисничког интерфејса садржи у себи све потребне информације како би било могуће задовољење свих постављених критеријума. Како би се олакшало креирање овог модела биће развијени софтверски алати који имају улогу да воде корисника

кроз процес спецификације корисничких захтјева и да конструише семантички богат модел који је потом спреман за трансформацију у извршиви код.

Полазна хипотеза истраживања је да је могуће успоставити везу између модела случајева коришћења и жељеног корисничког интерфејса. Доказивање постојања јасних веза између ових елемената омогућава формирање семантички богатог мета-модела заснованог на моделу случајева коришћења и информацијама везаним за кориснички интерфејс који омогућава аутоматизацију процеса генерисања извршивог програмског кода корисничког интерфејса. Модел једног корисничког захтјева формиран коришћењем поменутог мета-модела може се аутоматски трансформисати у више шаблона корисничког интерфејса, задржавајући жељену функционалност. Овај модел се, такође, може аутоматски трансформисати у програмски код корисничког интерфејса на различитим имплементационим технологијама за различите типове апликација

Апликација развијена током овог истраживања – генератор – треба да покаже да је могућа аутоматизација процеса креирања корисничког интерфејса на основу предложеног модела у складу са дефинисаним критеријумума, како би се добио извршиви програмски код у различитим имплементационим технологијама.

Поред тога, на основу анализе начина интеракције корисника са системом, циљ је да се дође до скупа могућих шаблона корисничког интерфејса којим се различити кориснички захтјеви могу приказати.

Крајњи циљ овог рада је да се у што већој мјери смање вријеме и напор, а тиме и трошкови процеса имплементације корисничког интерфејса апликације као најзахтјевнијег дијела процеса имплементације читавог софтверског система [Myers]. Као доказ успјешности овог истраживања кроз два студијска примјера ће бити

приказан процес развоја десктоп и веб апликације коришћењем предложеног приступа.

1.4. Очекивани допринос

Резултати истраживања презентованог у дисертацији прије свега треба да допринесу убрзавању процеса развоја софтвера, првенствено корисничког интерфејса који директно одговара на постављене захтјеве корисника. Поред тога дисертација ће пружити следеће доприносе:

- Преглед различитих техника спецификације корисничких захтјева, техника за креирање модела података и значај прототиповања.
- Преглед и упоредна анализа постојећих приступа и алата који аутоматизују процес имплементације корисничког интерфејса.
- Идентификација и образложење критеријума које генератор корисничког интерфејса мора да задовољава како би могао бити прихваћен од стране ширег круга корисника у односу на постојеће генераторе.
- Откривање веза између модела случајева коришћења и будућег корисничког интерфејса.
- Откривање правила пресликавања између корисничког интерфејса развијеног за различите типове апликација (desktop, web и мобилне апликације).
- Дефинисање семантички богатог модела корисничких захтјева заснованог на моделу случајева коришћења и информацијама везаним за кориснички интерфејс како би на основу њега било могуће не само

пројектовање и имплементација корисничког интерфејса, већ и аутоматизација овог процеса.

- Аутоматизација процеса креирања корисничког интерфејса апликације у различитим имплементационим технологијама уз задовољење дефинисаних критеријума.
- Идентификација скупа могућих шаблона корисничког интерфејса којим се различити кориснички захтјеви могу приказати задржавајући жељену функционалност.

1.5. Методологија

У наставку ће бити приказан скуп метода које ће бити коришћене и које чине методолошки оквир за реализацију истраживања.

Преглед:

(по функцији: експлораторна, по циљу и улози: верификаторна, по садржају: теоријска истраживања)

- Историјата корисничког интерфејса.
- Начина прикупљања корисничких захтјева.

Преглед и упоредна анализа:

(по функцији: акциона, по циљу и улози: хеуристичка, по садржају: теоријска истраживања)

- Постојећих приступа који се баве сличним проблемом.
- Техника спецификације корисничких захтјева.

- Техника и нотација за креирање модела података.
- Елемената корисничког интерфејса различитих типова апликација (desktop, web, мобилних апликација).
- Формалних спецификација корисничког интерфејса.

Истраживање:

(по функцији: интегрално, по циљу и улози: хеуристичка, по садржају: теоријска истраживања)

- Веза између елемената спецификације случајева коришћења, доменског модела и корисничког интерфејса.
- Карактеристике корисничких интерфејса различитих типова софтверских система.
- Шаблона корисничког интерфејса.
- Семантички богат модел корисничког захтјева за генерисање корисничког интерфејса.
- Трансформација модела у извршиви програмски код.

Научно испитивање (анкета):

(по функцији: интегрално, по циљу и улози: хеуристичка, по садржају: емпиријско истраживање)

- Дефинисање критеријума које алат за аутоматизацију мора да поштује (метода: анкета).

Студија случаја:

- Поступак пројектовања и имплементације софтверског система предложеним приступом.

1.6. Структура

Након увода, у другом поглављу дисертације биће разматрани различити приступи у развоју корисничког интерфејса (скицирање, визуелно креирање, коришћење језика за спецификацију, као и коришћење модела и моделом вођени развој), а за сваки приступ биће приказане уочене предности и недостаци, са циљем да нови приступ аутоматизацији процеса генерисања корисничког интерфејса искористи предности постојећих приступа развоју корисничког интерфејса.

У трећем поглављу биће разматрани постојећи приступи инжењерингу софтверских захтјева, техника за прикупљање захтјева, а биће направљен и преглед карактеристика двије доминантне технике за спецификацију захтјева (случајеви коришћења и корисничке приче). Биће идентификоване предности технике случајева коришћења у контексту аутоматизације процеса креирања корисничког интерфејса засноване на захтјевима. Поред тога, биће објашњена и улога доменског модела и његове везе са случајевима коришћења. На крају ће бити размотрена спецификација корака у сценарију случаја коришћења и њихов утицај на будући кориснички интерфејс, са циљем да се на основу изведених закључака формира мета-модел за креирање улазне спецификације која ће у потпуности бити заснована на случајевима коришћења. Такође, биће објашњена и улога прототипова у инжењерингу захтјева, са посебним акцентом на еволутивне прототипове, који су нарочито интересантни у контексту генерисања корисничог интерфејса. Наиме, уколико генерисани кориснички интерфејс не задовољава у потпуности захтјеве корисника, треба

омогућити његово побољшавање – еволуцију до крајње верзије, која задовољава све постављене захтјеве.

У поглављу 4 биће покушано да се дође до одговора на питања који су разлози за чињеницу да ни један од постојећих алата није успио да око себе окупи широк круг корисника, односно да задовољи потребе привреде, који су то захтјеви које алати за генерисање корисничког интерфејса морају да испуне како би били прихваћени од стране потенцијалних корисника и да ли данас актуелни алати испуњавају ове захтјеве. Зато ће прво бити спроведено испитивање – анкета. Циљ анкете је да се утврди који су то преовлађујући ставови носилаца софтверске индустрије – искусних софтверских инжењера, по питању елемената тј. карактеристика које алат за генерисање корисничког интерфејса мора да посједује како би га прихватили и користили у процесу развоја софтвера. Уочене карактеристике ће касније, у седмом поглављу, бити посматране као захтјеви које нови приступ генерисању корисничког интерфејса мора да задовољи како би био прихваћен од стране потенцијалних корисника. На основу уочених захтјева биће дефинисани критеријуми на основу којих ће бити поређени постојећи алати за генерисање корисничког интерфејса.

У петом поглављу биће размотрени принципи пројектовања корисничког интерфејса који су примјењиви на било који од типова софтверских система (десктоп, веб и мобилне апликације), а биће приказане и препоруке за рјешавање уобичајених проблема при пројектовању које се често у литератури [Tidwell05][UIPatterns] називају патернима корисничког интерфејса (*User Interface Patterns*). Како је један од циљева дисертације омогућавање генерисања корисничког интерфејса за различите типове софтверских система у различитим имплементационим технологијама, биће размотрене специфичности корисничких интерфејса различитих типова софтверских система и имплементационих технологија које се користе за њихов развој. Циљ ове анализе је идентификација карактеристика корисничког интерфејса различитих

типова софтверских система. Будући алат за аутоматизацију генерисања корисничког интерфејса мора узети у обзир идентификоване карактеристике како би на одговарајући начин одговорио на захтјеве корисника.

У шестом поглављу биће анализиране везе које постоје између случајева коришћења и елемената корисничког интерфејса, са циљем да се дође до скупа шаблона корисничког интерфејса који се аутоматски могу повезати са одређеним корисничким захтјевом. Ове везе ће бити формализоване у моделу за спецификацију, па ће тиме бити омогућено да се једноставном и брзом промјеном шаблона у моделу изгенерише потпуно другачији кориснички интерфејс који задовољава специфициране функционалности. Сви дефинисани шаблони морају уважити принципе, препоруке и специфичности у пројектовању корисничког интерфејса.

Седмо поглавље ће представљати синтезу закључака претходних поглавља. Биће приказане карактеристике новог приступа аутоматизацији процеса генерисања корисничког интерфејса. Централна тачка овог поглавља биће дефинисање мета-модела улазне спецификације која ће се користити за аутоматизацију генерисања корисничког интерфејса. Биће приказани различити начини креирања улазне спецификације, а након тога биће приказане карактеристике алата за генерисање програмског кода корисничког интерфејса, као и карактеристике генерисаног програмског кода. Све карактеристике новог приступа треба да буду у складу са захтјевима који су приказани у четвртом поглављу, а који су добијени анализом одговора испитаника. Ка крају поглавља биће приказан модел процеса развоја корисничког интерфејса коришћењем новог – *SilabUI* приступа.

У осмом поглављу, користећи методу студије случаја, биће приказано коришћење новог приступа на конкретном примјеру, са приказом свих акивности дефинисаних моделом процеса.

На крају ће бити дата закључна разматрања, преглед постигнутих циљева, остварени доприноси, као и правци будућих истраживања.

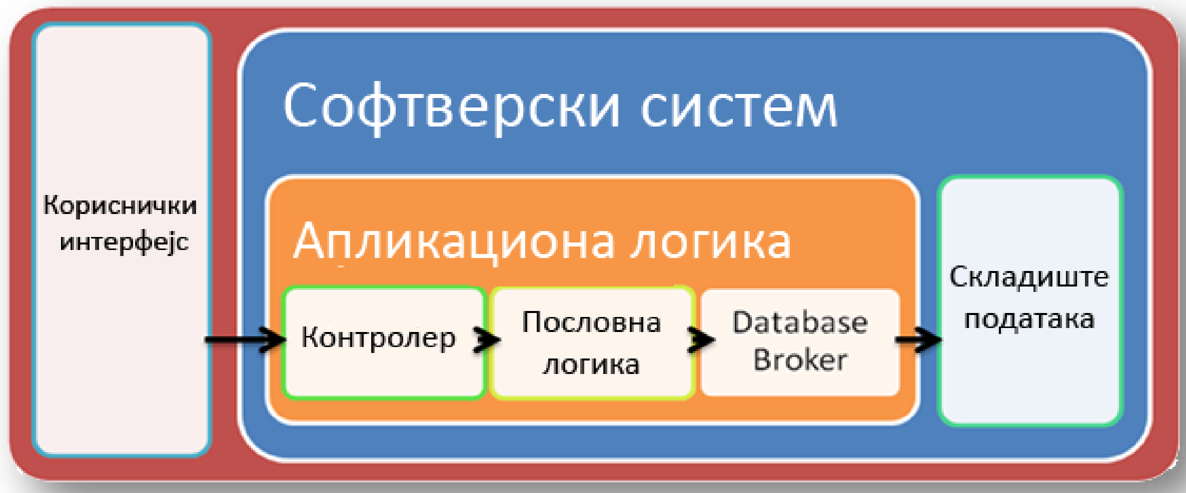
2. ЗНАЧАЈ ИСТРАЖИВАЊА И ПРЕГЛЕД ПОСТОЈЕЋИХ ПРИСТУПА У РАЗВОЈУ КОРИСНИЧКОГ ИНТЕРФЕЈСА

Развој пословних апликација данас подразумијева коришћење тронивојске архитектуре [Freeman12].

Овај концепт предвиђа постојање три логичке цјелине – нивоа: ниво корисничког интерфејса, апликационе логике и складишта података [Larman98].

- Кориснички интерфејс представља улазно – излазну репрезентацију софтверског система.
- Апликациона логика представља структуру и понашање софтверског система.
- Складиште података одговорно је за чување стања софтверског система.

Тронивојска архитектура може графички бити представљена на сљедећи начин:



Слика 3: Упрошћен приказ трослојне архитектуре

Апликациона логика се најчешће састоји од три логичке цјелине и то:

Контролер апликационе логике – представља јединствену тачку приступа операцијама које софтверски систем треба да омогући. Најчешће представља реализацију façade патерна, из скупа тзв. ГОФ патерна [GOF].

Пословна логика – која чини језгро система и најчешће се састоји од два важна дијела:

- Објекти домена – класе које презентују доменске концепте;
- Сервиси – сервисни објекти за функције за рад над доменским објектима, представљају реализацију системских операција (могу бити имплементиране коришћењем веб сервиса [Zukowski06], *session bean* компоненти као и *servlet* компоненти [Jendrock08]).

Перзистентни оквир – треба да омогући перзистенцију доменских објеката система у одабраном складишту података. Поставља се као ниво између програма и складишта података.

2.1. Кориснички интерфејс

Кориснички интерфејс представља дио апликације који омогућава кориснику интеракцију са софтверским системом. Кориснички интерфејс се у претходних неколико деценија јављао у различитим облицима, да би данас потпуно преовладао графички кориснички интерфејс[1] (*Graphical User Interface – GUI*). Појам кориснички интерфејс ће се у тези искључиво односити на графички кориснички интерфејс.

Концепт корисничког интерфејса има више значења. Може да се односи на изглед екрана, распоред графичких компоненти, а може да буде посматран као један подсистем или ниво у архитектури софтверског система.

Са становишта корисника, разлика између нивоа корисничког интерфејса и других нивоа софтверског система није јасно уочљива, а најчешће је потпуно невидљива. За већину корисника софтверски систем представља средство помоћу

---

[1] Почеци графичког корисничког интерфејса везују се за истраживања на *Stanford Research Laboratory* (SRL) и *Massachusetts Institute of Technology* (MIT) шездесетих и седамдесетих година XX вијека, да би крајем седамдесетих и почетком осамдесетих година истраживања у истраживачком центру *Palo Alto Research Center Incorporated* (бивши *Xerox PARC*) довела до првих комерцијалних система прије свега *Xerox Star* (1981), the *Apple Lisa* (1982) и *Macintosh* (1984). [Myers98] Ова истраживања поставила су основне парадигме развоја графичког корисничког интерфејса као што су „WYSIWYG“ (*What You See Is What You Get*) и WIMP (Window, Icon, Menu, Pointing device) које и данас представљају основу сваког графичког корисничког интерфејса.

којег се извршавају одређени задаци. За њих, кориснички интерфејс није само један дио софтверског система, већ читав систем [Vliet08].

Са друге стране, са становишта пројектанта система, кориснички интерфејс се посматра као реализација улаза и/или излаза апликационе логике софтверског система [Vlajic03]. Методе развоја софтвера предвиђају, свака на свој начин, одређене фазе и активности унутар сваке фазе развоја софтверског система. Карактеристика која повезује различите методе развоја софтвера јесте раздвајање пројектовања функционалности система од пројектовања начина интеракције корисника са системом. Како животни циклус софтвера подразумијева фазе прикупљања корисничких захтјева, анализу, пројектовање, имплементацију и тестирање, пројектовање корисничког интерфејса обично започиње тек након завршетка фазе прикупљања корисничких захтјева, или након пројектовања апликационе логике. Пројектовање корисничког интерфејса тада се своди на одабир и распоређивање графичких компоненти у циљу извршавања претходно пројектованих функционалности апликационе логике система.

Насупрот томе, постоји и другачији приступ који предлаже пројектовање корисничког интерфејса паралелно са спецификацијом корисничких захтјева. Овај приступ заснован је на корисничкој перспективи схватања софтверског система, која се може сумирати тезом да кориснички интерфејс за корисника не представља само један дио софтверског система већ је за њега то систем у цјелини [Vliet08]. Разлог томе јесте што корисник не познаје детаље структуре и начина реализације система. Специфицирањем корисничких захтјева даје се приказ начина на који корисник жели да изврши одређени задатак помоћу софтверског система. Извршење сваког од ових задатака подразумијева интеракцију корисника и система која има одређену структуру. Кориснички интерфејс, по овом приступу, треба да прати ову структуру како би се достигао што виши ниво корисности (*usability*) софтверског система.

Креирање корисничког интерфејса у раним фазама не подразумијева потпуно функционалан интерфејс, већ прототип који првенствено има за циљ да на површину избаци питања и претпоставке које можда нису биле јасне примјеном других приступа у прикупљању и анализи захтјева [Pfleeger06].

2.2. Значај (мотивација)

Веома битан аспект код развоја софтверских система јесте цијена извођења софтверског пројекта. Када се узме у обзир чињеница да израду софтверског производа чини низ интелектуалних и техничких активности у којима учествују високо школовани инжењери, јасно је да од времена и напора које је потребно уложити у производњу система у великој мјери зависи његова цијена. Израда корисничког интерфејса софтверског система има значајан удио у читавом процесу. Када се погледају резултати истраживања ове проблематике, интересантан је податак да 48% програмског кода софтверског система чини дио везан за кориснички интерфејс, а чак 50% времена од укупног времена развоја одлази на његову имплементацију [Myers]. Слични резултати добијени су и у другим истраживањима у овој области [Kivisto00], без обзира на то што се у развоју користе алати који убрзавају и олакшавају овај процес.

Због чињенице да кориснички интерфејс представља директну везу корисника са системом, често се у току израде софтверског система дешава да корисник мијења захтјеве везане за његов изглед и структуру. Ове измјене понекад изискују промјене над имплементираним системом које неријетко резултују одбацивањем до тада развијаног интерфејса и понављањем цјелокупног процеса. Чак и након тога, дешава се да корисник открије да ни ново рјешење не одговара његовим потребама, јер

корисник често није у могућности да своје потребе довољно добро искаже кроз спецификацију захтјева.

Све ове чињенице намећу потребу за аутоматизацијом овог процеса која треба да доведе до рјешења које ће убрзати процес креирања корисничког интерфејса и смањити његове трошкове. При томе, рјешење мора да у што већој мјери максимизира атрибуте квалитета корисничког интерфејса, прије свега атрибут корисности[2] (*usability*). Добијени кориснички интерфејс мора бити у потпуности конзистентан са постављеним корисничким захтјевима. На тај начин, убрзао би се сам развој корисничког интерфејса, али и добио алат којим би се ефикасније извршио процес спецификације корисничких захтјева. У том смислу корисник би брзо могао да уочи недостатке у спецификацији захтјева и отклони их, како би добио систем који што ближе задовољава његове потребе. Аутоматизација овог процеса имала би утицаја на све фазе животног циклуса софтвера: од прикупљања захтјева до његове имплементације и тестирања.

Како је на самом почетку наведено, полазна тачка у креирању софтверског производа јесте фаза прикупљања корисничких захтјева и резултати ове фазе представљају основу за све остале фазе у животном циклусу софтвера. Из тог разлога спецификација корисничких захтјева треба да буде довољно детаљна да пружи потребне информације за пројектовање и имплементацију свих дијелова софтвера – од корисничког интерфејса, преко апликационе логике која учаурује доменску структуру и понашање система, до складишта података. Међутим, често спецификација корисничког захтјева не даје довољно информација за пројектовање појединих дијелова система, па се препушта програмеру да на свој начин

[2] Корисност (*usability*) је атрибут квалитета који оцјењује лакоћу коришћења корисничког интерфејса. [Nielsen12]

интерпретира захтјеве и изналази рјешења, а често добијено рјешење не задовољава потребе корисника [Frost07]. У том смислу, уколико се разматра развој корисничког интерфејса, из корисничког захтјева ријетко се могу добити информације о изгледу корисничког интерфејса, његовом садржају, врстама компоненти које треба да се користе и сл. Такође, многи модели, развијени за подршку пројектовању софтверских компоненти и њихових међусобних интеракција, не обрађују довољно организовање компоненти корисничког интерфејса као ни интеракцију корисника са овим компонентама [Harmelen01] [DaSilva00]. Зато је потребно одабрати метод којим ће се што цјеловитије специфицирати кориснички захтјев, како би било могуће пројектовање различитих дијелова софтверског система. Данас, најраспрострањенији методи за прикупљање корисничких захтјева јесу спецификација захтјева помоћу случајева коришћења – *use case specification* [Cockburn00], и преко корисничких прича – *user stories* [Cohn04]. Случајеви коришћења и корисничке приче имају доста додирних тачака (засновани су на сценаријима и усмјерени ка циљевима корисника), али и битних разлика (у структурираности, опсегу који покривају, цјеловитости, трајности и детаљности) [Stellman09]. И једна и друга техника имају за циљ да специфицирају намјеру корисника тј. циљ који корисник жели да оствари кроз жељену функционалност.

Како је раније поменуто, најчешћи узрок неуспјеха софтверских пројеката јесу недостаци и грешке у корисничким захтјевима [Frost07]. Један од начина за њихово отклањање у раним фазама јесте прављење прототипова корисничког интерфејса. Корисник преко корисничког интерфејса ступа у интеракцију са сотверским системом. Током ове интеракције корисник може упутити позив систему за извршење одређене операције која мијења стање система. У објектно орјентисаним апликацијама, стања система учаурују се у доменске објекте система који, заједно са везама и ограничењима, чине доменски модел софтверског система. Доменски модел

описује статичку структуру софтверског система, тј. основне концепте из домена пословног проблема који се пројектује. Овај модел се још назива и концептуални модел и прави се у раној фази (фаза анализе) животног циклуса софтвера. Доменски модел представља полазну тачку за даље пројектовање модела података који ће бити основа за пројектовање складишта података софтверског система. Како случајеви коришћења описују интеракцију између корисника и система, која се реализује преко корисничког интерфејса и има за циљ промјену стања система која су учаурена у доменским објектима, веза између ова три дијела система – случајева коришћења, корисничког интерфејса и доменског модела, морала би бити узета у обзир већ у фази прикупљања корисничких захтјева. Када би се узела у обзир ова чињеница приликом креирања прототипа корисничког интерфејса, такав интерфејс би се могао користити за валидацију како случајева коришћења, тако и модела података [Fitzgerald05]. Под валидацијом се подразумијева провјера усаглашености корисничких захтјева са пројектованим доменским моделом. Међутим, прототипови корисничког интерфејса најчешће нису формално повезани са спецификацијом захтјева па измјенама захтјева или прототипа може доћи до неусаглашености између прототипа и спецификације и умјесто повећања прецизности могу да искрсну нови проблеми у разумијевању захтјева [DaCruz10].

2.3. Постојећи приступи у развоју корисничког интерфејса

Постоји више различитих приступа развоју корисничког интерфејса, па су различити аутори на различите начине извршили категоризацију постојећих приступа. У својој докторској дисертацији [DaCruz10] аутор наводи четири категорије приступа:

- коришћење алата за скицирање (*Sketching tools*),
- коришћење алата за визуелно креирање корисничког интерфејса (*UI Builders*),
- коришћење језика за опис корисничког интерфејса засноване на *XML*-у,
- развој корисничког интерфејса заснован на моделима

Слична категоризација направљена је у докторској дисертацији [Kennard11] у којој аутор закључује да се приступи могу груписати у једну од три категорије:

- Интерактивни алати за спецификацију корисничког интерфејса (*Interactive Graphical Specification Tools*),
- развој корисничког интерфејса заснован на моделима,
- коришћењем алата заснованим на језицима

У наставку ће детаљније бити објашњене наведене категорије приступа како би се утврдило у ком контексту се користе и који од њих одговарају постављеним циљевима овог рада.

2.3.1. Алати за скицирање (Sketching tools)

Алати за скицирање корисничког интерфејса обично се користе у фази пројектовања, приликом пројектовања екранских форми, али због једноставности коришћења често се користе и за прављење прототипова током фазе прикупљања и анализе захтјева у комуникацији са крајњим корисницима система, како би се захтјеви што тачније и прецизније дефинисали. Најпростији „алат“ за скицирање, а уједно и најкоришћенији је коришћење папира и оловке, при чему се слободним покретом руке цртају графичке форме и компоненте будућег корисничког интерфејса. За ову сврху могу се користити и софтверски алати намијењени за цртање, или графички алати опште намјене који подржавају дефинисање графичких

компоненти као што су *Microsoft Visio* [MSVisio] или *Creately* [Creately]. Поред алата опште намјене постоје и специјализовани алати за скицирање корисничког интерфејса (*Silk designer* [SilkDesigner], *Denim* [DENIM]), који поред скицирања појединачних корисничких интерфејса омогућавају и њихово повезивање чиме се дефинише и будућа навигација кроз кориснички интерфес читавог система. Оваквим повезивањем скица интерфејса формира се тзв. *storyboard* – табла на којој су представљене скице корисничких интерфејса и везе које између њих постоје.

Нарочито су интересантни алати (нпр. *Silk designer*) који омогућавају скенирање и препознавање ручно цртаних скица корисничког интерфејса, који се тада могу трансформисати у изворни код конкретне имплементационе технологије (нпр. *Java Swing*).

Иако је скицирање корисничког интерфејса тј. прављења прототипа којим се јасније дефинишу софтверски захтјеви пожељно, постоји неколико проблема који могу настати њиховим коришћењем. Основни проблем је непостојање формалних веза између скица и софтверских захтјева представљених текстуалном специфицикацијом (случајеви коришћења, корисничке приче...) или моделима (UML дијаграм класа, UML дијаграм секвенци и сл.). Свака промјена над захтјевима може да доведе до неконзистентности између ових различитих начина представљања захтјева, што касније током анализе и пројектовања система може довести до погрешне интерпретације захтјева. Поред овога, скице корисничког интерфејса се најчешће као такве не могу употребити за даљу обраду, већ се прије свега користе за прављење прототипова који се бацају, о чему ће више ријечи бити у наставку.

2.3.2. Алати за визуелно креирање корисничког интерфејса (GUI Builders)

Алати за визуелно креирање корисничког интерфејса представљају најчешће коришћене алате за имплементацију корисничког интерфејса. Ови алати омогућавају графичку конструкцију корисничког интерфејса избором из палете конкретних предефинисаних графичких компоненти (текстуална поља, падајуће листе, лабеле...) које се обично превлаче (*drag and drop*) на жељену позицију на радној површини (обично на контејнерске компоненте – компоненте које могу садржати друге компоненте: форме, дијалози, панели и сл.) и на тај начин постају дио новог интерфејса. [DaCruz10] Алати омогућавају да се након одабира компоненти њихов изглед подешава измјенама предефинисаних особина за сваку компоненту.

Разлог за популарност ових алата лежи у њиховој интуитивности, што их чини лаким за коришћење, али и у конзистентности интерфејса који се пројектује са резултујућим интерфејсом, а то је постигнуто коришћењем принципа „оно што видиш то ћеш и добити“ (*What You See Is What You Get* – WYSIWYG). Алати за визуелно креирање корисничког интерфејса обично омогућавају „оживљавање“ корисничког интерфејса и прије него што се повеже са апликационом логиком система.

Алати за визуелно креирање корисничког интерфејса најчешће су дио интегрисаних развојних окружења (*Integrated Development Environment – IDE*) попут *NetBeans IDE* [NetBeansIDE], *Eclipse IDE* [EclipseIDE], *Visual Studio* [VisualStudio] Креирање корисничког интерфејса коришћењем ових алата обично резултује генерисањем програмског кода конкретне имплементационе технологије (Java, C#...), али постоје и алати који умјесто генерисања програмског кода имплементационе технологије, генеришу програмски код неког од језика за спецификацију корисничког

интерфејса (нпр. *Extensible Application Markup Language* – *XAML* језик који генерише алат *Visual Studio*), о чему ће бити више ријечи у наставку поглавља.

Иако већина алата омогућава прављење интерфејса у једном смјеру – из графичког алата према програмском коду, постоје и алати који омогућавају да се програмски код корисничког интерфејса који је написан ручно, или добијен процесом генерисања кода „препозна", учита у алат и даље подешава графички (нпр. *Jigloo GUI Builder* [Jigloo]).

Алати за визуелно креирање корисничког интерфејса првенствено се користе у фази имплементације, али по потреби могу бити коришћени и у осталим фазама, нпр. за прављење прототипова или током пројектовања екранских форми. Ако се посматрају ови алати из контекста предмета докторске дисертације, којим се предвиђа аутоматизација процеса генерисања корисничког интерфејса на бази спецификације софтверских захтјева, њихова улога се знатно мијења. Наиме, директно коришћење алата за визуелно креирање корисничког интерфејса не ријешава проблем неконзистентности између спецификације захтјева и резултујућег интерфејса, због непостојања формалних веза. Такође, ако се узме у обзир постојање веза између елемената спецификације софтверских захтјева са резултујућим интерфејсом, може се закључити да се одређени аспекти интерфејса већ дефинишу спцифицирањем захтјева, па би прављење корисничког интерфејса коришћењем ових алата заправо представљало дуплирање посла. Ипак, ако би процес аутоматизације резултовао генерисањем програмског кода корисничког интерфејса, алати за визуелно креирање корисничког интерфејса који омогућавају „препознавање" и учитавање кода могу добро послужити за даља графичка подешавања свих аспеката који нису предвиђени моделом за генерисање корисничког интерфејса.

2.3.3. Језици за спецификацију корисничког интерфејса

Језици за спецификацију корисничког интерфејса омогућавају креирање платформски независног описа корисничког интерфејса. Обично су засновани на XML-у (*Extensible Markup Language*) што кориснички интерфејс специфициран на овај начин чини читљивим од стране рачунарских програма који имају улогу да их интерпретирају и кориснику прикажу кориснички интерфејс. Најкоришћенији језик за спецификацију корисничког интерфејса је HTML (*Hypertext Markup Language*) [HTML]. Програмски код написан овим језиком се интерпретира од стране веб претраживача (*web browser*) и кориснику се приказује кориснички интерфејс. Међутим HTML је првенствено намијењен за коришћење у веб технологији, па се јавила потреба за језиком који ће омогућити виши ниво апстракције, и омогућити интерпретацију и креирање интерфејса за много шири скуп технологија.

Развијено је много различитих језика за спецификацију корисничког интерфејса, а најпознатији међу њима су User Interface Markup Language (UIML), XML User interface Language (XUL) и User Interface Language (UIL) [Phanouriou00]. Међутим, ови језици нису прављени са намјером да се аутоматизује процес генерисања корисничког интерфејса, већ је главни циљ био прављење спецификације корисничког интерфејса која је независна од конкретне технологије којом се конкретан кориснички интерфејс имплементира. Платформска независност спецификације је значајна карактеристика, али није довољна да би ови језици стекли ширу популарност. Језици за спецификацију су много мање интуитивни у односу на алате за визуално креирање корисничког интерфејса. Ипак, платформска независност омогућава да се независно праве графички алати за креирање спецификације у неком од ових језика, као и да се независно праве алати за интерпретацију добијених спецификација у различитим технологијама. Дакле, како би ови језици стекли ширу

популарност, потребно је унаприједити алате како за креирање спецификације, тако и за интерпретирање спецификације.

Можда највећи искорак у коришћењу језика за спецификацију корисничког интерфејса представља *Extensible Application Markup Language* – XAML који је дио *Microsoft Windows Presentation Foundation* – WPF [WPF] пројекта. Овај језик, поред спецификације корисничког интерфејса омогућава и спецификацију пословне логике система која је независна од тога да ли ће бити интерпретирана и извршена у веб или десктоп окружењу. Поред тога, подржана је алатима за визуелно креирање у склопу развојног окрижења *Visual Studio*, али је још увијек углавном ограничена на *.NET* платформу.

У контексту језика за спецификацију корисничког интерфејса свакако треба поменути и *USer Interface eXtensible Markup Language* – UsiXML, али како је овај језик створен са нешто другачијом намјеном у односу на поменуте, о њему ће бити више ријечи у наставку.

И поред добрих карактеристика, прије свега независности од имплементационе технологије, језици за спецификацију корисничког интерфејса, попут алата за визуелно креирање, сами по себи не осигуравају конзистентност између спецификације софтверских захтјева и резултујућег корисничког интерфејса. Ипак, ако би се процесом аутоматизације генерисања корисничког интерфејса, који је заснован на спецификацији софтверских захтјева, генерисала платформски независна спецификација корисничког интерфејса написана неким од наведених језика, омогућила би се интерпретација добијене спецификације и извршавање корисничког интерфејса коришћењем постојећих алата који интерпретирају одабрани језик за спецификацију.

2.3.4. Развој корисничког интерфејса заснован на моделима и моделом вођени развој (MDD)

Овај приступ се значајно разликује у односу на претходне, иако се у много чему може сматрати само допуном неког од претходних приступа. Разлог томе јесте сама природа модела који омогућавају усмјеравање пажње на битне особине система који се моделује и апстрахују све особине које нису од интереса у датом тренутку посматрања. Тако се при моделовању једног софтверског система може правити више различитх модела, при чему сваки модел приказује систем са одређеног аспекта, односно са различим погледом на исти систем. [Ludewig03]. Као такви, иако посматрају различите аспекте система, пошто се ради о једном истом систему, модели би морали да буду конзистентни између себе. Ако се посматра систем као скуп међусобно повезаних ентитета, при чему је стање система одређено вриједностима атрибута који описују ентитет и везама које успоставља са другим ентитетима, може се говорити о моделу који представља структуру система. Са друге стране, ако се посматра систем као процес који одређеним трансформацијама мијења стање система, може се говорити о моделу који представља понашање система. Конзистентност између ова два модела огледа се у томе да се трансформације стања система (модел понашања) односи на промјене вриједности атрибута ентитета или веза између ентитета који су дефинисани у систему моделом структуре система. Ако се ови модели користе да би се олакшао процес разумјевања система од стране различитих заинтересованих страна (нпр. приликом интревјуа са крајњим корисницима), модели могу бити поједностављени, и може се допустити блага неконзистентност односно неподударање различитих модела. Међутим, ако се модели корсте за аутоматско процесирање софтверским алатима који њиховом трансформацијом треба да произведу програмски код, између ових модела је потребно успоставити јасне формалне везе и мора се осигурати њихова

конзистентност. [DaCruz10] Ово је основни принцип на којем почива развој система заснован на моделима.

Интеракција између корисника и система, тј. начин на који ће корисник користити функционалности система се такође може посматрати као посебан аспект софтверског система, па се и за ту сврху могу правити различити модели. Међутим, у литератури се начелно могу препознати два погледа на овај аспект система – „инжењерски" и „дизајнерски" [Belenguer03]. Инжењерски поглед на кориснички интерфејс карактерише моделовање начина на који ће се обезбиједити кориснику приступ до жељених функционалности система. Са друге стране, „дизајнерски" приступ познатији под називом *Human Computer Interaction* – HCI[3] посматра кориснички интерфејс као понашање корисника приликом коришћења система, тј. начин на који ће корисник рјешавати постављене задатке коришћењем система. Иако се може рећи да је разлика између ова два погледа на систем незнатна, временом су приступи дивергирали до те мјере да сваки од њих користи потпуно различите моделе којима се описује кориснички интерфејс. Док се инжењерски приступ претежно ослања на модел случајева коришћења и доменски модел, уз евентуално прављење прототипа корисничког интерфејса, HCI приступ подразумијева дефинисање циљева корисника, и хијерархијског организовања задатака које корисник треба да ријеши коришћењем система, што укључује идентификацију објеката и акција које се повезују са сваким задатком. HCI приступ такође подразумијева и специфицирање карактеристика корисника који ће користити систем и начина његове интеракције са системом.

[3] Често се посматра као преплитање рачунарских наука, бихевиоралистичке науке, графичког дизајна, медија, као и неколико других области. Овај израз су популаризовали Stuart K. Card and Allen Newell са Carnegie Mellon универзитета, као и Thomas P. Moran из истраживачког сектора IBM-а у својој књизи The Psychology of Human-Computer Interaction [Card83] из 1983. године.

Као резултат различитих приступа у пројектовању корисничког интерфејса, развијени су различити модели који у центар стављају различите аспекте овог дијела система. На овом мјесту биће приказани они модели који су у досадашњој пракси коришћени за развој корисничког интерфејса. Насупрот приступима у којима инжењер прецизно дефинише детаље везане за кориснички интерфејс (врсте графичких компоненти, њихов положај, редослијед и сл.) ови модели су осмишљени да дају одговор на питање **шта** се од корисничког интерфејса очекује, не улазећи директно у детаље **како** ће то бити остварено. [Schmidt06] Одговор на питање како, даје се за специфичну имплементациону технологију и платформу за извршење. На тај начин, захтјеви система се изражавају декларативно, а не императивно. У литератури [Schlungbaum96][MBUIUseCases][Meixner11] се ови модели називају декларативни модели, а поред модела случајева коришћења, о којима ће више ријечи бити у следећим поглављима, укратко ће бити описани:

- Модел задатка (Task model);
- Доменски модел (Domain model);
- Кориснички модел (User model);
- Модел дијалога (Dialogue model);
- Презентациони модел (Presentation model).

Модел задатака се може посматрати као скуп сценарија из спецификације случајева коришћења и њиме се описује интеракција између корисника и софтверског система у циљу извршења одређеног задатка. Задаци могу бити специфицирани на различитим нивоима апстракције, па се прије имплементације задаци декомпонују на конкретније задатке који се могу аутоматизовати. Опис једног задатка треба да садржи [Storrs95]:

- један циљ,

- скуп акција или других задатака потребних за достизање постављеног циља,
- начин одабира и редослијед акција или задатака и
- објекат (најчешће један или више доменских објеката) над којим се врши задатак, или је резултат задатка.

Овај модел се најчешће користи у фази пројектовања система и обично се проширује другим моделима, нпр. доменским моделом или моделом дијалога, како би се добила што детаљнија специфицикација система, иако постоји и приступ који директно користи овај модел за генерисање корисничког интерфејса [Rich09]. Предност оваквог приступа јесте у интуитивности корисничког интерфејса, јер директно прати специфициране намјере корисника. Ако корисник може прецизно дефинисати захтјев, моделом задатка се на веома конзистентан начин може извршити спецификација захтјева и генерисати кориснички интерфејс који у потпуности одговара постављеном захтјеву. Међутим, ако корисник не може прецизно да дефинише захтјев и након креирања корисничког интерфејса схвати да такав интерфејс није одговарајући, потребно је поново направити модел задатка и њему одговарајући кориснички интерфејс. Значајан допринос коришћењу овог модела у контексту развоја корисничког интерфејса јесте његова стандардизација од стране Америчког националног института за стандардизацију (*American National Standards Institute – ANSI*) и Асоцијације корисника електронских уређаја (*The Consumer Electronics Association – CEA*). По овом стандарду модел задатка се дефинише као формални опис активности везаних за извршење задатка, укључујући активности које извршава човјек, као и оне које извршава машина. Овај стандард дефинише семантику и *XML* нотацију модела задатка за коришћење електронских уређаја. Стандард је независан у односу на технологије или инфраструктуре [TaskModel].

Доменским моделом, о коме ће више ријечи бити у наредним поглављима, дефинишу се ентитети, њихови атрибути, методе и међусобне везе. Готово сви приступи развоју корисничког интерфејса који су засновани на моделима се у мањој или већој мјери ослањају на овај модел, али најчешће га допуњују информацијама које су специфичне за кориснички интерфејс. Актуелни приступи који су у највећој мјери засновани на овом моделу су *OpenXava* [OpenXava], *JMatter* [JMatter], и *Naked Objects (Apache Isis)* [NakedObjects].

Кориснички модел описује карактеристике крајњих корисника или група крајњих корисника (дефинисаним преко улога) интерактивног система. Главна сврха корисничког модела је да омогући креирање корисничког интерфејса који може да се прилагоди кориснику у зависности од његових карактеристика. Разликују се апликационо-независне карактеристике (нпр. преференције, могућности корисника...) и апликационо-зависне карактеристике (нпр. циљеве, познавање система...). Овај модел се не користи самостално за развој корисничког интерфејса, али у неким приступима као што је *SUPPLE* [Gajos10], допуњује друге моделе.

Моделом дијалога описује се интеракција човјек-рачунар. Овим моделом описује се редослијед уношења улазних података, излазних података и њихово преклапање. Овај модел је директно оријентисан на кориснички интерфејс система, за разлику од модела задатака који се односи на интеракцију између корисника и система независно од корисничког интерфејса. Модел дијалога одређује који ће се елементи наћи на корисничком интерфејсу, под којим условима корисник може да им приступи, као и под којим условима систем извршава одређене операције. Модел дијалога најчешће се користи у комбинацији са моделом задатка.

Презентациони модел описује компоненте које се приказују крајњем кориснику, њихов распоред и међусобне зависности. Такође, овим моделом

успоставља се веза између елемената доменског модела и модела задатака са конкретним графичким компонентама (дугме, падајућа листе, текстуално поље...) који представљају објекте интеракције [Sinnig04]. Презентациони модел, у већини апликација, састоји из статичког и динамичког дијела. Статички дио обухвата графичке компоненте као што су дугмад, менији и листе вриједности и др. Овај дио је најчешће непромјенљив у вријеме извршавања програма. Динамичким дијелом приказују се подаци који су специфични за апликацију, тако да је овај дио најчешће промјенљив у вријеме извршавања програма. Презентација података који су специфични за апликацију није довољно подржана од стране алата за израду корисничког интерфејса – они само обезбјеђују површину за цртање (*drawing area*), док се за приказ и обраду података мора ручно писати програмски код.

Сви ови модели могу да се користе независно један од других, али као што је раније наглашено, сваки од њих представља један поглед на исти систем, и заједно чине комплетну спецификацију система. Зато се намеће логичан закључак да би ови модели по правилу требали да буду конзистентни. Ипак, да би ова конзистентност била осигурана, међу моделима морају да постоје јасна правила пресликавања која би прије свега омогућила валидацију спецификације система. Поред тога, тек након успостављања јасних веза међу моделима, могло би се омогућити трансформационо моделовање система које препоручује приступ моделом вођеног развоја (*Model Driven Development – MDD*), о коме ће више ријечи бити наставку поглавља.

Аутоматизација креирања корисничког интерфејса софтверских система заснована на моделима

Аутоматизација креирања корисничког интерфејса софтверских система заснована на моделима представља тему која је била предмет многих истраживања у претходне двије деценије [Raneburger12]. Резултат ових истраживања представљају

алати чија је основна улога смањење времена и напора потребног за развој корисничког интерфејса. Међутим, до сада се нити један од ових алата није наметнуо на тржишту и око себе окупио шири круг корисника [Kennard10]. Анализом ових алата може се утврдити да међу њима постоје значајне разлике. Ове разлике су присутне како у самим полазним основама тј. улазним спецификацијама па све до разлика у начину реализације генерисања корисничког интерфејса. Што се тиче начина реализације аутоматског генерисања корисничког интерфејса, поједини алати обезбјеђују динамичко генерисање корисничког интерфејса током извршења апликације, док други предвиђају генерисање програмског кода корисничког интерфејса. Такође, поједини алати омогућавају измјене над генерисаним корисничким интерфејсом, као и неке додатне функционалности као што су флексибилност у начину приказа компоненти, избору компоненти које ће се наћи на корисничком интерфејсу, избору између корисничког интерфејса за десктоп и web апликације, избор између различитих циљних архитектура, док је код других читав интерфејс једнообразан и потпуно дефинисан самим алатом.

Међутим, недостаци самих алата се најчешће везују за улазну спецификацију – спецификацију улазних параметара на основу које се врши аутоматско генерисање корисничког интерфејса. Ова спецификација представља модел чијом се трансформацијом генерише кориснички интерфејс. Као полазне основе у постојећим алатима могу се наћи различити модели међу којима су најзаступљенији: модел задатка (*Task model*), доменски модел (*Domain model*), кориснички модел (*User model*), модел дијалога (*Dialogue model*) и презентациони модел (*Presentation model*) [Schlungbaum96]. Основни проблем представља сложеност ових модела и нотација које су тешке за учење и коришћење [Myers95] [Szekely96]. Како би ове моделе приближили ширем кругу корисника, поједини аутори су као улазну спецификацију за своје алате користили већ прихваћене UML дијаграме, али ни ови алати нису

стекли ширу популарност. Разлог томе јесте чињеница да UML дијаграми и CASE алати засновани на UML-у, попут Rational Rose [Quarani98] алата, не узимају у обзир експлицитно спецификацију корисничког интерфејса, односно не предвиђају модел којим би се директно специфицирао овај аспект софтверског система. Због тога је прављење спецификације овим алатима веома сложено и некомплетно [DaSilva00]. Поред тога, један од аутора наводи још три кључна проблема за које алати за аутоматско генерисање корисничког интерфејса још увијек нису дали прихватљиво рјешење: [DaSilva00/1]

- Тешко је доказати корисницима да улазна спецификација на одговарајући начин описује релевантне елементе потребне за генерисање извршивог корисничког интерфејса.
- Проблем интероперабилности генерисаног корисничког интерфејса са осталим дијеловима апликације.
- Проблем дефинисања скупа модела којим ће се описати кориснички интерфејс. Заправо, не постоји консензус око тога које аспекте корисничког интерфејса треба моделовати.

Поред ових недостатака, аутори наводе и недовољно посвећивање пажње такозваним „патернима корисничког интерфејса[4]“ [Molina02], који ће у овом раду, у нешто измијењеном значењу, бити означени као шаблони корисничког интерфејса, о чему ће бити више ријечи у наредним поглављима.

Што се тиче недостатака генерисаног корисничког интерфејса, као најчешћи недостатак у литератури се означава једнообразност корисничког интерфејса

[4] Важно је напоменути да се термин "патерн" у контексту корисничког интерфејса не односи на софтверске патерне [GOF], већ на уобичајене, шаблонске дијелове корисничког интерфејса који се често и на сличан начин понављају у различитим софтверским системима.

[Kennard11]. Ова карактеристика се уједно сматра пожељном у смислу да различити дијелови корисничког интерфејса једне апликације треба да буду што сличнији по начину распоређивања, врсти компоненти и сл. како би крајњи корисник могао да се навикне на одређени изглед и осјећај (*look and feel*) тј. заједничке карактеристике које одликују све дијелове корисничког интерфејса апликације. Ово се односи на врсту компоненти, њихову величину, распоред и сл. Међутим, када се ради о генерисаном корисничком интерфејсу, корисник генератора често не може да утиче на ове заједничке карактеристике, већ су оне обично дефинисане самим алатом. Уколико се генерише програмски код, тада се једино измјенама над генерисаним кодом може утицати на изглед интерфејса и на тај начин направити жељене измјене. Овај задатак није ни мало лак, јер је често структура генерисаног програмског кода сложена па је потребно уложити значајан напор како би се разумио постојећи код прије него што се почне мијењати. Овим се знатно продужава крива учења, па се доводи у питање оправданост коришћења алата. Ако се ради о алату који приликом извршења интерпретира модел и формира кориснички интерфејс (runtime user interface generation) тада постаје готово немогуће извршити било какве измјене, уколико сам модел не предвиђа детаљну спецификацију баш ових карактеристика корисничког интерфејса.

Други недостатак односи се на зависност осталих дијелова апликације од генерисаног корисничког интерфејса. Генерисани кориснички интерфејс захтијева одређену „позадинску“ архитектуру, па се тиме, у зависности од начина пројектовања, у већој или мањој мјери повећава међузависност између ових дијелова апликације. Ова зависност свакако постоји и при директној имплементацији интерфејса, али тада најчешће постоји подударање између логике пројектовања ова два дијела система, што није случај код коришћења алата за генерисање корисничког интерфејса. Зато је потребно уложити напор како би се ове разлике превазишле, што се у случају

генерисања кода корисничког интерфејса може постићи коришћењем софтверских патерна. Међутим, ако се ради о алатима који приликом извршења интерпретирају модел и формирају кориснички интерфејс, тада је ову зависност јако тешко избјећи, нарочито ако алат експлицитно предвиђа читаву позадинску архитектуру апликације. Типичан примјер оваквог алата је Naked Objects (Apache Isis) [NakedObjects] који захтијева посебан начин имплементације доменских објеката система.

Моделом вођени развој (*Model Driven Development* – MDD)

Моделом вођени развој (*Model Driven Development* – MDD) подразумијева развој софтверских система коришћењем више различитих, међусобно повезаних, модела кроз све фазе животног циклуса. Овај приступ користи моделе за приказ различитих аспеката система на различитим нивоима апстракције, и предвиђа прецизна правила трансформације између модела на различитим нивоима апстракције – од апстрактних и платформски независних модела, преко конкретнијих модела до програмског кода за конкретну имплементациону технологију. Иако се приступ може примјењивати без употребе алата за аутоматску трансформацију модела, јасне формалне везе између различитих модела и дефинисана правила трансформације омогућавају да се процес развоја аутоматизује. Зато се, као предуслов за моделом вођени развој софтвера, који ће бити аутоматизован, јавља се потреба за опште прихваћеним стандардима у креирању модела, као и алатима који ће бити засновани на овим стандардима. [Selic03]

Међународни конзорцијум *Object management Group* – OMG је 2001. године представио *Model Driven Architecture* – MDA која је постала стандард за моделом

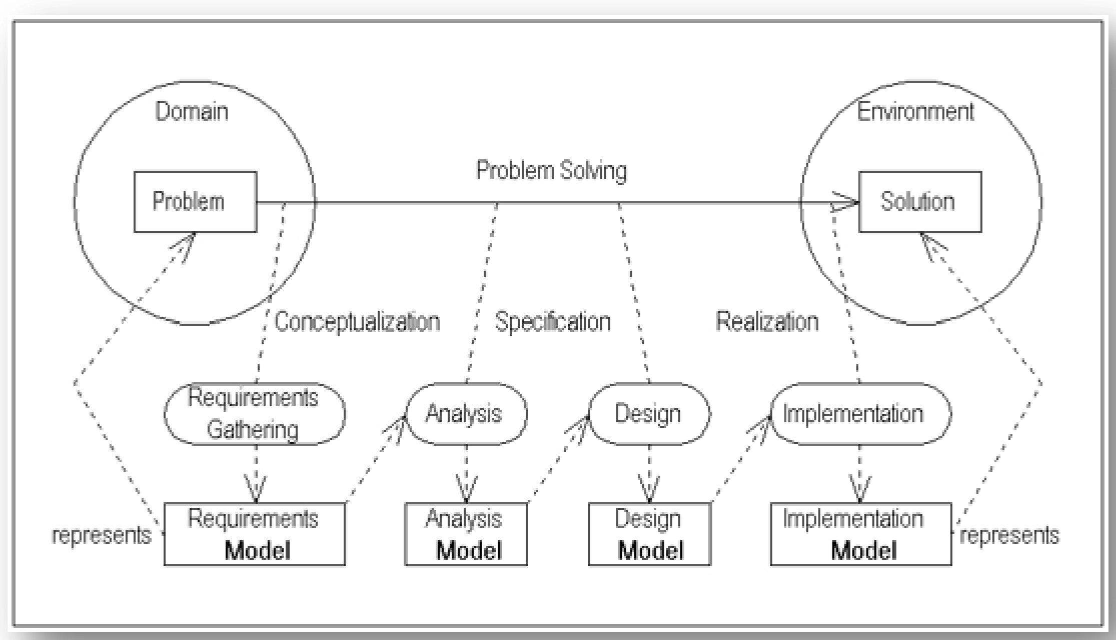
вођени развој софтвера који до тада није био стандардизован. Такође, OMG је дефинисао и стандард за мета-моделовање *Meta-Object Facility* – MOF, који омогућава креирање сопствених мета-модела. Коришћењем MOF-a, омогућено је специфицирање сопствених типова ентитета и дефинисање веза између њих на стандардан начин. Данас постоји више имплементација спецификација дефинисаних од стране OMG-a, а једна од најзначајнијих је *Eclipse Modeling Framework* – EMF [EclipseMF] односно *Graphical Modeling Framework* – GMF [EclipseGMP] развијене од стране *Eclipse Foundation* – EF [EclipseFoundation]. Парадигма моделом вођеног развоја подразумијева дефинисање модела кроз четири нивоа апстракције:

- M3 ниво апстракције – подразумијева мета мета-модел који се користи за спецификацију сопственог мета-модела. Ови модели дефинишу концепте на највишем нивоу апстракције. Сви концепти који се појављују на нижим нивоима заправо представљају инстанце концепата који су дефининисани на овом нивоу. Примјери за коришћење мета мета-модела су прављење мета-модела за UML дијаграм класа, модел објекти – везе, модела случајева коришћења, дијаграма токова података и сл. Ови модели се зову M3 модели и карактерише их платформска независност. Ако се посматра примјена овог нивоа апстракције на примјеру процеса генерисања Јава класа, једини концепт кога је потребно дефинисати на овом нивоу је концепт класе.
- M2 ниво апстракције – подразумијева мета-моделе који се користе за спецификацију модела. Како су ови мета-модели дефинисани концептима на M3 нивоу, концепти у моделима на M2 нивоу ће бити инстанце концепата са M3 нивоа. Ови модели се користе за дефинисање системске платформе. Ако се посматра примјена овог нивоа апстракције на примјеру процеса генерисања Јава класа, и узме у

обзир претходно дефинисани М3 модел, концепти које је потребно дефинисати на М2 нивоу су Јава класа, атрибут и веза. Сви поменути концепти представљају инстанцу концепта класе који је дефинисан на М3 нивоу.

- М1 ниво апстракције – подразумијева моделе који представљају апстрактну представу система који се развија. Нпр. то може бити модел неког конкретног пословног система. Модели на овом нивоу су тзв. М1 модели који су специфицирани за неку од платформи које су креиране на вишем нивоу апстракције. Ако се посматра примјена овог нивоа апстракције на примјеру процеса генерисања Јава класа, и узму у обзир претходно дефинисани М3 и М2 модели, концепти које је потребно дефинисати на М1 нивоу су жељене конкретне класе система који се моделује, са конкретним атрибутима и међусобним везама које класе успостављају. Као примјер на М2 нивоу могу се дефинисати концепти Рачун, са атрибутима шифра и назив, Ставка рачуна са концептима редни број, назив производа, количина и цијена, као и веза између концепата Рачун и Ставка рачуна. Сви поменути концепти представљају инстанце концепата дефинисаних на М2 нивоу.
- М0 ниво апстракције – подразумијева модел система који је у продукцији, тзв. М0 модели који представљају објекте реалног система који су специфицирани на вишем нивоу апстракције. Ако се посматра примјену овог нивоа апстракције на примјеру процеса генерисања Јава класа, и узму у обзир претходно дефинисани М3, М2 и М1 модели, концепти које је потребно дефинисати на М0 нивоу заправо су концепти дефинисани на М1 нивоу представљени Јава изворним кодом.

На слици која слиједи (Слика 4) приказује се концептуални однос између различитих фаза у процесу развоја софтверског система, кроз трансформацију домена проблема, у домен рјешења, при чему се у свакој фази користе одговарајући конкретни модели.



Слика 4: Однос између различитих фаза у процесу развоја софтверског система, кроз трансформацију домена проблема у домен рјешења [Alhir03]

MDA дефинише три врсте односно типа модела *Computation Independent Model* – CIM, *Platform Independent Model* – PIM и *Platform Specific Model* – PSM, којима се може додати и четврти *Implementation Specific Model* – ISM, који ако се говори о процесу генерисања изворног кода заправо представља конкретни изворни код у одабраној имплементационој технологији.

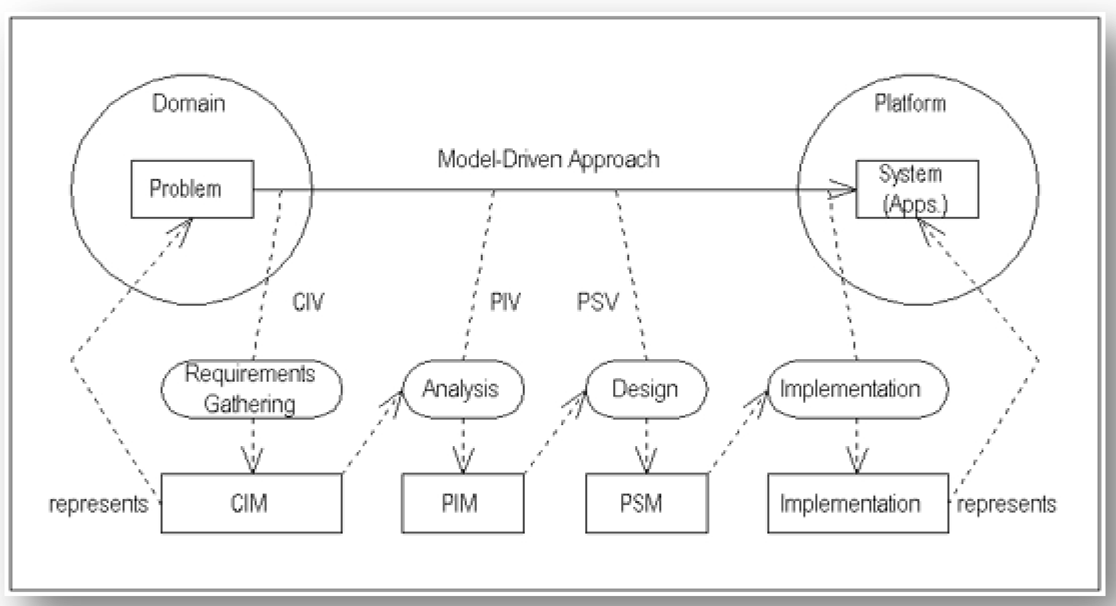
Computation Independent Model – CIM модел система (из рачунарски независне перспективе) описује домен и захтјеве система. Ако се посматра развој софтверског

система који треба да пружи подршку процесима неког пословног система, тада ће CIM модели приказивати концепте пословног система и његовог окружења.

Platform Independent Model – PIM модел система (из платформски независне перспективе) описује систем независно од конкретне платформе. Ако се посматра развој софтверског система који треба да пружи подршку процесима неког пословног система, тада ће PIM модели приказивати концепте софтверског система који се могу односити на структуру или понашање система који се моделује, али на тај начин да су платформски независни, тј. да омогуће да буду реализовани коришћењем различитих платформи.

Platform Specific Model – PSM модел система (из платформски зависне перспективе) описује систем независно од конкретне платформе. Ако се посматра развој софтверског система који треба да пружи подршку процесима неког пословног система, тада ће PSM модели приказивати концепте софтверског система који се могу односити на структуру или понашање система који се моделује, али на тај начин да узимају у обзир особине одабране платформе.

Слика (Слика 5) илуструје процес развоја софтвера приказан на слици (Слика 4) посматран кроз однос различитих погледа на систем, из различитих перспектива, у контексту MDA и дефинисаних врста модела (CIM, PIM, PSM и ISM).



Слика 5: Однос различитих погледа на систем, из различитих перспектива, у контексту MDA и дефинисаних врста модела (CIM, PIM, PSM и ISM) [Alhir03]

Коришћењем различитих нивоа апстракције и модела са прецизно дефинисаним одговорностима (принципом *Separation of Concernes*) омогућава се једноставнија и јаснија спецификација система. Међутим, посљедица оваквог приступа подразумијева коришћење великог броја модела, што може довести до неконзистентности [Wagner03]. Моделом вођени развој треба да омогући развој софтвера коришћењем модела који, у крајњој линији, треба да представе замјену за програмски код. Програмски код није ништа друго до оперативни модел који укључује све потребне детаље повезане са платформом за извршење [Barbier13].

Један од тренутно најзначајнијих пројеката који је везан за моделом вођени развој корисничког интерфејса је *UsiXML* (*USer Interface eXtensible Mark-up Language*) пројекат. [UsiXML] *UsiXML* је прије свега језик за опис различитих врста корисничког интерфејса (текстуалне, графичке, интерфејса засноване на звуку и сл.).

Намијењен је како софтвер инжењерима, тако и крајњим корисницима јер, због добре подржаности алатима за креирање спецификације, не захтијева познавање програмерских вјештина. *UsiXML* је декларативни језик који описује какав је кориснички интерфејс, или какав треба да буде, независно од његових физичких карактеристика [UsiXML/1].

UsiXML предвиђа опис тзв. конкретног корисничког интерфејса (*Concrete User Interface – CUI*) који је независан од конкретне имплементационе технологије. Овај опис представља конкретизацију тзв. апстрактног корисничког интерфејса (*Abstract User Interface – AUI*) за одређену врсту интерфејса, и дефинише распоред графичких компонената и навигацију кроз различите елементе корисничког интерфејса. Конкретни кориснички интерфејс се конкретизује финалним корисничким интерфејсом (*Final User Interface – FUI*). Поменути нивои апстракције – финални кориснички интерфејс, конкретни кориснички интерфејс, апстрактни кориснички интерфејс и такође модел задатака и концептуали модел, представљају структуру *UsiXML* пројекта која је преузета из пројекта *Cameleon* [Cameleon].

Важно је нагласити да *UsiXML* пројекат у фокусу има дефинисање спецификације корисничког интерфејса на различитим нивоима апстракције. Представља покушај групе истраживача да се дефинише стандард [UsiXML/2] прије свега у спецификацији корисничког интерфејса, док је развој алата који би на основу ове спецификације генерисали кориснички интерфејс био у другом плану. Ипак, прихватање оваквог стандарда у софтверској индустрији значио би огроман искорак и подршка развоју алата за аутоматско генерисање корисничког итерфејса заснованих на принципима моделом вођеног развоја.

3. Постојећи приступи у инжењерингу софтверских захтјева

О значају софтверских захтјева у развоју софтвера најбоље говоре истраживања која показују да се половина фактора који утичу на успјех софтверског пројекта везује управо за софтверске захтјеве [Johnson00]. Зато ће у наставку бити представљени кључни концепти везани за процес прикупљања и спецификације софтверских захтјева.

Термин *захтјеви (requirements)* у софтверском инжењерству често се не користи конзистентно, већ се интерпретира различито у зависности од контекста у којем се користи. Тако у литератури се, између осталих, налазе изрази софтверски захтјеви (software requirements), кориснички захтјеви (user requirements), системски захтјеви (system requirements), инжењеринг захтјева (requirements engineering) и спецификација софтверских захтјева (software requirements specification – SRS). Ради прецизности, у наставку ће бити дефинисан сваки од ових израза.

- **Софтверски захтјеви** (*software requirements*) – представљају опис потреба корисника система за одређеним функционалностима које систем треба да пружи, као и опис ограничења која морају бити задовољена приликом функционисања система. При томе корисник не мора увијек бити и крајњи корисник система, већ то може бити наручилац система, који неће директно користити систем. Препорука је да се, кад год је могуће, софтверски захтјеви узимају од крајњих корисника, јер најчешће управо они најбоље познају домен пословног

проблема за који се систем прави. Софтверски захтјеви се могу описивати на различитим нивоима апстракције [Savic12], па се по том критеријуму могу разликовати кориснички захтјеви и системски захтјеви. Софтверски захтјеви морају садржати довољан ниво детаљности како би инжењерима који развијају систем обезбиједили све информације које су потребне за даљи развој.

- **Кориснички захтјеви** (*user requirements*) – представљају опис потреба корисника за функционалностима које систем треба да пружи, најчешће исказан природним језиком у неформалном облику, као и опис ограничења која морају бити задовољена приликом функционисања система. Дакле, кориснички захтјеви се могу сматрати описом софтверских захтјева на вишем нивоу апстракције. Корисничким захтјевима описује се тзв. спољно понашање система, тј. не улази се у детаље о томе како систем треба да изврши тражену функционалност. Такође, ови захтјеви не треба да садрже стручне изразе из области софтверског инжењерства, већ треба да се концентришу на пословни проблем за који се развија систем и обично су формирани из перспективе корисника система.
- **Системски захтјеви** (*system requirements*) – представљају детаљан опис скупа функционалности и ограничења које систем треба да задовољи, и најчешће су разрада корисничких захтјева, тј. на детаљнији начин описују како кориснички захтјеви треба да буду реализовани у софтверском систему. При томе, ни ови захтјеви не би требало да улазе у детаље везане за пројектовање саме структуре и понашања софтверског система, већ треба детаљније да опишу пословни проблем за који се систем пројектује. Међутим, у пракси је тешко раздвојити

спецификацију проблема од предлога рјешења, па се често у опису системских захтјева могу наћи и дијелови који залазе у сферу пројектовања софтверског система. Системски захтјеви се, поред описа природним језиком, представљају и у неком формалном облику (структуриран природни језик, специјални језици за описивање, графичке нотације, математичке спецификације) [Sommerville06].

- **Спецификација софтверских захтјева** (*Software Requirements Specification – SRS*) – документ који садржи опис свих софтверских захтјева и представља званичну изјаву о томе шта чланови развојног тима треба да имплементирају. Овај документ треба да укључи и корисничке и системске захтјеве. У неким случајевима, кориснички и системски захтјеви се заједно представљају, док се најчешће кориснички захтјеви приказују на почетку као нека врста увода, док се системски захтјеви уводе накнадно као разрада корисничких захтјева. Овај документ треба да буде што цјеловитији како би био од користи свима који га користе. Поред крајњих корисника система, и аналитичара (који најчешће састављају овај документ), заинтересоване стране чине и наручиоци система, менаџмент који треба да управља софтверским пројектом, инжењери система који треба да развијају систем, инжењери који треба да тестирају систем, инжењери који треба да одржавају систем итд. Данас постоји много различитих шаблона који дефинишу све дијелове које овај документ треба да садржи. Један од шаблона дефинисан је као стандард *IEEE/ANSI 830-1998* [IEEE98]. Шаблоне и стандарде за спецификацију софтверских захтјева треба схватити више као препоруку и водич кроз спецификацију захтјева, него као правило, јер информације које треба да садржи овај документ

највише зависе од карактеристика самог система који треба да се произведе [Sommerville06]. Спецификација софтверских захтјева може бити посматрана и као процес чији је резултат документ који садржи софтверске захтјеве. Овај процес чине активности које су углавном дефинисане одабраном методом инжењеринга захтјева, а унутар сваке активности могу се користити различите технике и нотације за спецификацију захтјева, о чему ће бити више ријечи у наставку.

- **Инжењеринг захтјева** (*requirements engineering*) – представља процес прикупљања (*elicitation*), анализирања, документовања (спецификације), и провјере (*validation*) софтверских захтјева.

Различите методе пројектовања софтвера (Јединствен процес, Ларман, Екстремно програмирање, Scrum...) на различит начин описују процес прикупљања и спецификације корисничких захтјева. Ове разлике се могу уочити у техникама које се користе (случајеви коришћења, корисничке приче...), моделима којима се захтјеви приказују (различити UML дијаграми, модели објекти-везе...), активностима које се изводе у овом процесу (различите методе подразумијевају различите активности, или различит редослијед сличних активности), а као резултат се добијају различита документа која представљају спецификацију софтверских захтјева. Међутим, сваки документ који представља спецификацију корисничких захтјева, подразумијева бар два стандардна дијела[5]: функционални и нефункционални захтјеви.

[5] Поред функционалних и нефункционалних захтјева, постоје и други који се такође могу наћи као саставни дио спецификације [Sommerville06]:

- *Доменски захтјеви* представљају захтјеве који се тичу специфичности пословног домена система који се пројектује, али не директно везаних за начин интеракције корисника са системом (што је обухваћено функционалним захтјевима). Могу се посредно односити и на функционалне и на нефункционалне захтјеве. Типичан

- **Функционални захтјеви** – опис функционалности које систем треба да пружи, опис начина на који ће реаговати на дефинисане улазе, као и начина понашања система у одређеном ситуацијама. Укратко, функционални захтјеви описују „шта“ систем треба да ради.
- **Нефункционални захтјеви** – најчешће представљају ограничења под којим систем реализује захтијевано понашање. То могу бити временска ограничења, ограничења везана за процес развоја, или стандарди који морају бити задовољени. Најчешће се нефункционални захтјеви односе на систем као цјелину, а не на неки одређени дио система, неку његову посебну функционалност. Такође, дешава се да се нефункционални захтјеви односе само на неки дио одређеног функционалног захтјева. Зато документ који садржи спецификацију софтверских захтјева мора да обезбиједи јасне везе између функционалних и нефункционалних захтјева, како се не би десило да се при имплементацији неког функционалног захтјева превиде нефункционални захтјеви који су за њега везани.

Веома битан чинилац у прикупљању и спецификацији софтверских захтјева јесу технике које се користе у овом процесу. Одабрана метода развоја софтвера најчешће дефинише и која ће техника бити коришћена, и начин на који ће резултати

примјер за доменске захтјеве су формуле за израчунавање појединих вриједности које ће бити коришћене у систему.

- *Захтјеви везани за интерфејс система* представљају захтјеве који се односе на интеракцију система са окружењем, најчешће са другим системима. Тако се дефинишу *процедурални интерфејси* који описују сервисе које систем ставља на располагање окружењу (врста API-a) као и *структура података* који се користе у комуникацији између система и окружења.

примијењене технике бити интерпретирани. Међутим, неке технике се на различите начине користе у различитим методама, па ће у наставку бити приказане суштинске карактеристике најкоришћенијих техника независно од метода у којима се користе.

3.1. Технике за прикупљање захтјева

Прије него што се приступи спецификацији софтверских захтјева, потребно је открити који су то захтјеви које треба специфицирати, што представља један од основних задатака пословне анализе (*business analysis*). При томе, како је наведено у Водичу кроз области знања пословне анализе (*Guide to the Business Analysis Body of Knowledge BABOK*) [BABOK] захтјеви морају бити комплетни, јасни, тачни и конзистентни. Пословна анализа треба да обезбиједи да су сви захтјеви видљиви и схваћени од стране свих заинтересованих страна. Овако прикупљени захтјеви касније се специфицирају неком од техника за спецификацију софтверских захтјева.

Треба напоменути да се израз прикупљање захтјева у пословној анализи сматра непрецизним, па чак и погрешним, јер на почетку овог процеса захтјеви нису довољно видљиви и доступни како би могли бити једноставно прикупљени. Израз који се сматра погоднијим је елицитација захтјева (*requirements elicitation*), откривање захтјева [Elicitation]. Ипак, у наставку ће бити коришћен термин прикупљање захтјева јер се као такав одомаћио у српском језику.

Процес прикупљања захтјева обично подразумјева примјену неколико различитих техника како би се што ефикасније дошло до коначног скупа захтјева које будући систем треба да задовољи. Као технике пословне анализе намијењене прикупљању захтјева наводе се [BABOK]:

- *Brainstorming*
- Анализа докумената
- Фокус групе
- Анализа интерфејса
- Интервју
- Посматрање
- Прототиповање
- Радионице за прикупљање захтјева
- Анкета

Ове технике могу се користити самостално, али се обично користи комбинација неколико техника. Бројни фактори могу утицати на скуп техника које ће се користити, међу којима су искуство аналитичара, конкретан пословни домен, специфичности пословног окружења у којем се захтјеви прикупљају и сл.

Поред поменутих техника истакао бих и двије методе, које се обично користе у неким каснијим фазама животног циклуса, али које могу допринијети ефикаснијем прикупљању захтјева, а то су методе за анализу и моделовање процеса: Моделовање пословних процеса (*Business process modeling – BPM*) и Структурна системска анализа (*Structured system analysis – SSA*). Моје лично искуство говори о неопходности примјена ових метода, прије свега Структурне системске анализе, како би се осигурало да је обухваћен комплетан скуп потребних захтјева. Такође, ова метода се показала као погодна у разговорима са доменским експертима, јер садржи веома једноставну и разумљиву нотацију која се састоји од само четири графичка концепта. Наравно, ова метода је потпуно комплементарна са осталим поменутим техникама за прикупљање захтјева.

Структурна системска анализа посматра пословни систем као процес (функцију) који преко улазних и излазних токова података комуницира са спољним ентитетима – интерфејсима. Структурна системска анализа омогућава декомпозицију пословног система на подпроцесе, чиме се систем посматра као хијерархија процеса у више нивоа, при чему се сваки процес описује посебним дијаграмом тока података. Дијаграм тока података на врху хијерархије се назива дијаграм контекста, на којем је приказан само један процес (читав систем), спољни ентитети (интерфејси) и токови података који повезују систем са интерфејсима. Дијаграми нижих нивоа описују токове података између пословних процеса, интерфејсе, као и одговарајућа складишта података (који су искључиво унутрашње карактеристике система). Декомппозиција се врши на тај начин што се пословни процес са дијаграма на једном нивоу хијерархије декомпонује на подпроцесе који се приказују на новом дијаграму тока података на нижем нивоу хијерархије. Процеси на сваком нивоу хијерархије треба да буду што је могуће међусобно независни (што је могуће постићи успостављањем веза коришћењем складишта података, насупрот прављењу директних веза токовима података), што омогућава даљу независну анализу и декомпозицију појединачних процеса. Декомпозиција се врши док се не дође до примитивних (атомских) процеса, који представљају једну логичку јединицу посла. [Neskovic00]

На овај начин добијен скуп свих примитивних процеса представља коначан скуп захтјева које треба специфицирати коришћењем неке од техника за спецификацију софтверских захтјева. У смислу функционалности које будући систем треба да обезбиједи, овај скуп захтјева дефинише границе система којег треба пројектовати.

3.2. Технике за спецификацију захтјева

Данас, најраспрострањеније технике за прикупљање корисничких захтјева, како је већ раније поменуто, јесу спецификација захтјева помоћу случајева коришћења – *use case specification* [Cockburn00] [Jacobson93], и преко корисничких прича – *user stories* [Cohn04]. Случајеви коришћења и корисничке приче имају доста додирних тачака (засновани су на сценаријима и усмјерени ка циљевима корисника), али и битних разлика (у структурираности, опсегу који покривају, цјеловитости, трајности и детаљности) [Stellman09]. И једна и друга техника имају за циљ да специфицирају намјеру корисника тј. циљ који корисник жели да оствари кроз жељену функционалност.

Случајеви коришћења представљају традиционални приступ у прикупљању и спецификацији корисничких захтјева. У литератури се први пут појављују 1987. године [Jacobson87] као једна од техника за спецификацију понашања система, да би 1993. године били означени као техника за прикупљање и спецификацију корисничких захтјева заснована на сценаријима у оквиру методе за објектно орјентисано пројектовање софтвера [Jacobson93].

Постоји више дефиниција случајева коришћења, али ни једна на цјеловит и недвосмислен начин не даје одговор на питање шта је то случај коришћења. Зато ће у наставку бити приказано неколико дефиниција које се у литератури најчешће цитирају:

- Случај коришћења представља уговор између различитих заинтересованих страна о понашању софтверског система. Случај коришћења описује понашање система под различитим условима при одговору на захтјев корисника система – примарног актора. Примарни

актор започиње интеракцију са системом како би остварио одређени циљ [Cockburn00].

- Опис скупа секвенци акција, укључујући варијанте, које извршава систем пружајући при томе видљиви резултат значајан за корисника [Jacobson00].
- Случајеви коришћења су техника за прикупљање корисничких захтјева заснована на сценаријима [Sommerville06].
- Случај коришћења: Скуп сценарија који остварују исти кориснички циљ [Vliet08].

Случајеви коришћења данас су саставни дио UML (*Unified Modeling Language*) нотације, а техника случајева коришћења полазна је основа за више различитих метода развоја софтвера (*Unified Process, Objectory, Iconix Process*). Треба нагласити да случајевима коришћења није могуће обухватити читав кориснички захтјев, већ само аспект понашања система, тј. његове функције [Cockburn00]. Случајеви коришћења омогућавају да се понашање система описује на једноставан начин који је једнако разумљив како за корисника тако и за инжењера који ће пројектовати систем. Могу бити представљени у текстуалном облику, а могу се и графички представити UML нотацијом[6]. У основи, они представљају кориснички опис жељене функционалности система. Међутим, овај опис је структуриран, и садржи елементе кључне за развој софтверског система. Најважнији елементи који чине случај коришћења су актор и

[6] Случајеви коришћења се најчешће графички представљају прављењем UML дијаграма случајева коришћења, који приказује акторе, случајеве коришћења, затим везе које постоје између актора и случаја коришћења, као и везе између самих случајева коришћења. Треба напоменути да се овим дијаграмом не представљају сценарија случајева коришћења, већ се за сценарија прави текстуална спецификација, а сценарија се графички најчешће приказују коришћењем UML дијаграма секвенци којима се може представити интеракција корисника и система. Овакви дијаграми секвенци се називају системски дијаграми секвенци [Larman98].

сценарио. Актор представља корисника система а може бити особа (одређена својом улогом), компјутерски систем или организација. Сваки случај коришћења може садржати један или више сценарија. Најчешће је то један основни сценарио који доводи до успјешног извршења захтјева, а остало су тзв. алтернативна сценарија која представљају варијације основног сценарија и јављају се у случају када основни сценарио не може да се изврши. Појединачна сценарија једног случаја коришћења често се зову појављивања случаја коришћења (*use case instance*) или нит случаја коришћења. Случај коришћења се може посматрати као скуп повезаних успјешних и неуспјешних сценарија који описују начин на који актор користи систем ради постизања одређеног циља [Larman98]. Скуп случајева коришћења који описују све жељене функционалности система назива се модел случајева коришћења. Под појмом модел случајева коришћења обично се подразумијева UML дијаграм, на коме се приказују сценарија случаја коришћења. У овом раду ће зато моделом случајева коришћења биће сматран скуп текстуалних спецификација случајева коришћења. У зависности од нивоа детаљности, случајеви коришћења могу бити специфицирани на различитим нивоима апстракције [Savic12]

Поред случајева коришћења у пракси се користе и корисничке приче – *user stories*, које су фаворизоване од стране агилних метода развоја софтвера, прије свега методе Екстремног програмирања (*Extreme programming*). Оне могу описивати кориснички захтјев на различитим нивоима апстракције, али обично представљају једну малу функционалност система, значајну за корисника, која може бити развијена у кратком периоду времена (три недеље или мање). Ако корисничке приче описују захтјев на вишем нивоу апстракције, тзв. епске корисничке приче, оне се могу подијелити на конкретиније корисничке приче које ће бити имплементиране од стране развојног тима. [UserStories] Најчешће се представљају у текстуалном облику и пишу се на малим картицама како би се осигурала њихова концизност. Потребно је да

буде што краћа и да садржи само довољан ниво детаљности који ће бити довољан за процјену трајања развоја, као и да буде јасна и кориснику који функционалност захтијева и инжењерима који ће функционалност развијати. Ова техника прикупљања захтјева довољна је у случају када је развојни тим у блиској и константној вези са корисником система како би се све недоумице у развоју отклониле у контакту са корисником [Cockburn00]. Корисничке приче не морају да прате одређену структуру, али се аутори слажу са оцјеном да треба да садрже одговоре на питања: ко извршава тражену функционалност (из чије перспективе), шта жели да изврши (захтјев) и због чега (разлог). У литератури [UserStories/1] [UserStories/2] се може наћи препорука да корисничка прича треба да, у најопштијем смислу, прати следећу структуру:

Као <тип корисника>,

Желим <да извршим одређени задатак>,

Како бих могао да <постигнем неки циљ/корист/вриједност>.[7]

Основне разлике између ова два приступа јесу у њиховом опсегу, комплетности, трајности, као и у нивоу детаља који је могуће укључити. Случајеви коришћења најчешће обухватају много већи опсег корисничког захтјева од корисничке приче. Ако се посматрају корисничке приче и случајеви коришћења специфицирани на приближно истом нивоу апстракције и ако се жели успоставити директна веза између ове двије технике, иако се ради о различитим техникама, једна корисничка прича се може посматрати као један сценарио случаја коришћења. Из тог разлога може се говорити и о разлици у комплетности. Случај коришћења садржи

[7] As a <type of user>,
I want <to perform some task>
so that I can <achieve some goal/benefit/value>.

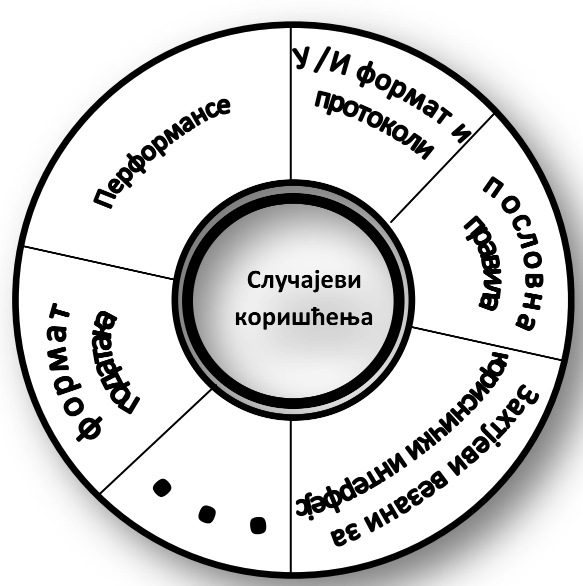
много више информација о жељеној функционалности у односу на корисничку причу, тако да је за опис једног случаја коришћења техником корисничких прича потребно направити више корисничких прича и то највјероватније за сваки алтернативни сценарио по једну. Битна разлика између случаја коришћења и корисничке приче је и у трајности или року важења. Случајеви коришћења обично су актуелни од почетка рада на пројекту па све до краја његовог одржавања, без обзира на број итерација. У свакој итерацији може доћи до њихове допуне или измјене, али најчешће егзистирају током читавог животног циклуса софтвера. Са друге стране корисничке приче егзистирају за вријеме трајања једне итерације у току које се имплементирају и постају саставни дио софтверског система [Cohn04]. Сљедећа важна разлика везана је за ниво детаља које је могуће укључити у спецификацију корисничког захтјева, а које нису искључиво везане за саму намјеру корисника већ укључују информације везане за кориснички интерфејс, заштиту, складиште података и сл. Корисничке приче, прије свега због своје дужине, не дозвољавају укључивање детаља овог типа и усмјерене су искључиво на то шта корисник жели да се оствари, а не и како. Случајеви коришћења, због своје структурираности, омогућавају да се информације овога типа укључе у сам сценарио, али се већина аутора слаже у оцјени да са укључивањем ових детаља треба бити веома опрезан. По једнима, детаље овог типа, никако не треба укључивати у случај коришћења [Cockburn00], док други препоручују да се ове информације изоставе из спецификације случајева коришћења у раним фазама прикупљања корисничких захтјева, док се касније, када се достигне већи ниво стабилности корисничких захтјева, ове информације постепено придружују случајевима коришћења [Larman98]. Овакве препоруке се заснивају на опасности која се огледа у томе да се укључивањем ових информација (које нису директно везане за сам циљ који корисник жели да оствари кроз неки случај коришћења) може скренути пажња са основног циља који случај коришћења треба да оствари, чиме се губи његова вриједност за сам систем.

Као што је раније истакнуто, спецификација корисничког захтјева треба да буде довољно детаљна да пружи потребне информације за пројектовање и имплементацију свих дијелова софтвера – од апликационе логике која учаурује доменску структуру и понашање система па до корисничког интерфејса и складишта података. Посматрајући наведене технике за прикупљање и спецификацију корисничких захтјева може се уочити да се случајеви коришћења намећу као прихватљивије рјешење у односу на корисничке приче, нарочито ако се жели постићи аутоматизација креирања одређених дијелова софтвера. Разлог за то је њихова структурираност, цјеловитост као и могућност уључивања додатних детаља значајних за различите аспекте система. Ипак, укључивање додатних детаља у случајеве коришћења не препоручује се од стране неких аутора, или бар не у почетним фазама спецификације захтјева. Ове препоруке су у литератури веома честе из разлога што је у пракси укључивање поменутих детаља у спецификацију корисничких захтјева честа појава. Разлог за то неки аутори виде у чињеници да је, узимајући у обзир структуру спецификације корисничких захтјева (SRS) као документа, најзгодније мјесто за приказивање захтјева везаних за кориснички интерфејс, управо уз спецификацију случајева коришћења. У овом раду биће представљен модел за спецификацију случајева коришћења који треба да помири ове двије тежње: спецификација случаја коришћења остаће *чиста*, док ће информације везане за кориснички интерфејс моћи да се са њом накнадно повежу, али као засебан документ[8]. У наредним поглављима детаљније ће бити описано које су то информације везане за кориснички интерфејс

[8] У оквиру извођења наставе на предмету Пројектовање софтвера користи се упрошћена Ларманова метода развоја софтвера [Larman98] [Vlajic03]. Ова метода разликује два начина приказивања случајева коришћења. У фази прикупљања захтјева се спецификација приказује независно од могуће реализације, док се у фази пројектовања, у дијелу који се односи на пројектовање корисничког интерфејса, случајеви коришћења приказују заједно са приказом пројектованих екранских форми, гдје се успоставља веза између корака у сценарију случаја коришћења и одређених елемената екранске форме које реализују тај скуп корака.

које ће се повезивати са случајем коришћења, као и начин на који се повезивање врши.

Веза између случајева коришћења и осталих дијелова спецификације корисничких зхтјева може се приказати као точак на коме се налазе захтјеви везани за кориснички интерфејс, пословна правила, формате података, корисничке профиле итд., док осовину точка чине случајеви коришћења (Слика 6).



Слика 6: Веза између случајева коришћења и осталих дијелова спецификације корисничких захтјева [Cockburn00]

Случајеви коришћења користе се и ван опсега самих корисничких захтјева. Тако, на примјер, широко се користе у дефинисању рокова завршетка пројекта, састављања развојних тимова, и праћењу напретка развоја софтверског пројекта. Дакле, иако се не налазе унутар саме спецификације случајева коришћења, све ове информације су у већој или мањој мјери везане за њу. Управо из ових разлога, многи

аутори сматрају спецификацију случајева коришћења централном тачком у развоју софтверског пројекта [Cockburn00].

Случајевима коришћења није могуће обухватити читав кориснички захтјев, већ само аспект понашања система, тј. његове функције, и то прије свега у оним случајевима када се посматра интеракција између корисника и система. Зато спецификација софтверских захтјева техником случајева коришћења није погодна за одређене врсте захтјева односно одређене врсте система. Типичан примјер за то је систем за резервацију и продају авионских карата. Са једне стране случајеви коришћења се показују веома погодном техником за опис интеракције купца са системом, при чему купац треба да одабере дестинацију, датум путовања и слично, као и да позове систем да израчуна цијену карте. Недостатак случајева коришћења огледа се управо у немогућности да се сложена логика рачунања цијене карте (која подразумјева рачунање одговарајуће руте, калкулацију пресиједања, различитих аеродромских такси, различитих тарифа повезаних компанија итд.) специфицира у сценарију случаја коришћења [UseCases]. Међутим, када ово се ограничење стави у контекст прављења корисничког интерфејса, што је у директном фокусу докторске дисертације, овај недостатак постаје потпуно ирелевантан, јер калкулација цијене карте није задатак корисничког интерфејса већ пословне логике система.

Случај коришћења описује једну жељену функционалност система везану за једног или више корисника система (актора). Цјеловит скуп случајева коришћења специфицира све предвиђене начине коришћења система, и на тај начин дефинише захтијевано понашање система, тј. границе система са аспекта његових функционалности [Malan01]. У спецификацији корисничких захтјева за одређени софтверски систем, у зависности од његове сложености, може постојати на стотине случајева коришћења. Што је систем сложенији, то се повећава могућност да дође до непрецизности и тзв. двосмислених (ambiguous) случајева коришћења.

3.3.Најчешће грешке и проблеми везани за захтјеве

Двосмисленост у софтверским захтјевима је веома честа, и јавља се у два облика. Први облик се препознаје у самом случају коришћења, када пројектант на различите начине може да тумачи исти захтјев. Проблем настаје када пројектант не може да уочи који од начина тумачења је исправан, већ само увиђа да постоји двосмисленост. Овај вид двосмислености се лако уочава и отклања у сарадњи са доменским експертима. Други облик двосмислености у захтјевима је много теже уочљив. Ако захтјева има много, они се обично дијеле у различите групе, и свака група захтјева распоређује се различитим члановима тима. Тада може доћи до ситуације у којој сви чланови тима извјештавају да су захтјеви исправни, и да у групи захтјева коју су анализирали нема двосмислености. Међутим, често се дешава да одређена група захтјева није у сагласности са захтјевима из других група, па овакве грешке лако могу да прођу непримијећено у фази прикупљања захтјева, што може постати велики проблем у наредним фазама софтверског пројекта. У литератури се идентификује више различитих извора двосмислености, као и препоруке којих се треба држати како би се смањила могућност њихових појављивања [Wiegers06]:

- Сложена логика (Complex Logic) – Сложеност логике у случају коришћења може бити узрок многим погрешним тумачењима захтјева. Сценарио у оваквим захтјевима је најчешће дугачак, и садржи једну или више разгранатих путања (IF-THEN-ELSE структура). Препорука је да када се уоче овакви захтјеви, да се они подијеле у више независних захтјева.
- Негативни захтјеви (Negative Requirements) – Негативни захтјеви су други извор забуне. Уколико постоји захтјев који описује понашање које не треба да се деси, такав захтјев не би требало специфицирати, јер

је такав захтјев већ задовољен. Уколико систем не ради ништа, нежељено понашање се сигурно неће десити. Препорука је да се захтјеви овог типа поново дефинишу, како би назначили шта систем треба да ради под одређеним условима.

- Изостављање информација (Omissions) – Захтјевима могу недостајати значајне информације, а најчешћи узрок томе јесте претпоставка да се одређено понашање подразумијева. (*Клијент има могућност креирања извјештаја у PDF формату.* Да ли треба развијати опцију штампања генерисаног извјештаја или не?) У том случају пројектанти могу различито разумјети исти захтјев, осим уколико немају исте претпоставке о њему.
- Ограничења (Boundaries) – Граничне бројне вриједности могу довести до двосмислености. Такође, оне могу означавати недостатак информација. (*Студенти који на тесту имају између 80 и 90 поена добијају оцјену 9.* Оно што је нејасно јесте да ли исту оцјену добијају и са тачно 80 или тачно 90 поена)
- Избјегавање двосмислених ријечи (Avoiding Ambiguous Wording) – У наставку ће бити наведене неке смјернице за писање захтјева како би се избјегле могуће замке и двосмислености језика:
    - Синоними: Уколико се у различитим документима описује иста ствар на различите начине, препоручује се смијештање дефиниција у заједнички рјечник, како би чланови тима могли да их досљедно користе кроз пројекат чак и кроз више њих.
    - Скоро-синоними: Понекад се могу користити термини за које се може помислити да су синоними, али они могу означавати суштински различите ствари. Такве термине треба дефинисати у

ријечнику пројекта како би сви читаоци на исти начин разумјели термине.

- Замјенице: Замјенице такође могу бити извор забуне у спецификацији захтјева. Треба се трудити да када се користе замјенице све буде јасно. Уколико се користе замјенице *ово* или *оно*, не би требало да постоји забуна на шта се замјенице односе.
- Скраћенице: Скраћенице такође могу довести до двосмислености. (*Клијент има могућност креирања извјештаја у PDF/XLS формату*. Да ли треба правити извјештај у оба формата, или само у једном од поменутих? Ако се прави у оба формата, да ли се оба генеришу истовремено или клијент треба да одабере жељени формат?) У том случају се предлаже да се експлицитно каже на шта се мисли, чиме се избјегава коришћење скраћеница.
- Непрецизни изрази: Захтјеви често садрже непрецизности које ће различити пројектанти различито протумачити. (*Наслов у извјештају треба да буде написан већим словима од остатка текста.*)

Све двосмислености у софтверским захтјевима морају бити уочене и отклоњене што је могуће раније како би се избјегли додатни напори и направила уштеда у времену. Ако се двосмисленост уочи тек када дође до фазе имплементације, као посљедица настају неопходне измјене које су не ријетко веома сложене и захтијевају много времена. У табели (Табела 1.) су приказани резултати истраживања [Frost07] који говоре о томе колико је времена потребно за исправљање грешака у односу на то у којој су фази пројекта грешке уочене. Тако нпр. ако се грешка уочи у

првој фази и за њено исправљање је потребан један сат, ако се иста грешка уочи тек током рада система, за њено исправљање биће потребно између 40 и 1000 сати.

Табела 1: Вријеме потребно за исправљање грешака по фазама (у сатима)

Прикупљање захтјева	Пројектовање	Имплементација	Тестирање (током развоја)	Тест прихватљивости	Рад система
1	3-6	10	15-40	30-70	40-1000

3.4. Доменски модел у контексту захтјева

Након прављења спецификације случајева коришћења, многе методе развоја софтвера препоручују прављење доменског модела система.

У објектно орјентисаним апликацијама, стања система учаурена су у доменске објекте система који, заједно са везама и ограничењима, чине доменски модел софтверског система. Доменски модел описује статичку структуру софтверског система, тј. основне концепте из домена пословног проблема који се пројектује. Овај модел се још назива и концептуални модел, и прави се у раној фази (фаза анализе) животног циклуса софтвера. Доменски модел представља основу за даље пројектовање модела података који ће бити основа за пројектовање складишта података софтверског система. Како случајеви коришћења описују интеракцију између корисника и система, која се реализује преко корисничког интерфејса и има за циљ промјену стања система, веза између ова три дијела система морала би бити узета у обзир већ у фази прикупљања корисничких захтјева.

Постоје различите технике за прављење доменског модела. Једна од најједноставнијих техника јесте коришћење листе категорија концептуалних класа, а често су у употреби и технике које препоручују тзв. патерне анализе или патерне доменског модела[9]. Међу најчешће коришћеним техникама, вјероватно због једноставности, јесте идентификација концептуалних класа уочавањем именица у корисничком захтјеву [Larman98]. Ова техника предлаже коришћење именица и именичких синтагми као кандидата за концептуалне класе тј. њихове атрибуте. Међутим, ако се узму у обзир узроци двосмислености код случајева коришћења, јасно је да ова техника може довести до грешака које се тада преносе и на структуру система. Грешке у структури које се уоче током имплементације је много теже отклонити, јер ће такве измјене имати утицаја на понашање система, те ће и његови дијелови који реализују понашање тада морати да се мијењају.

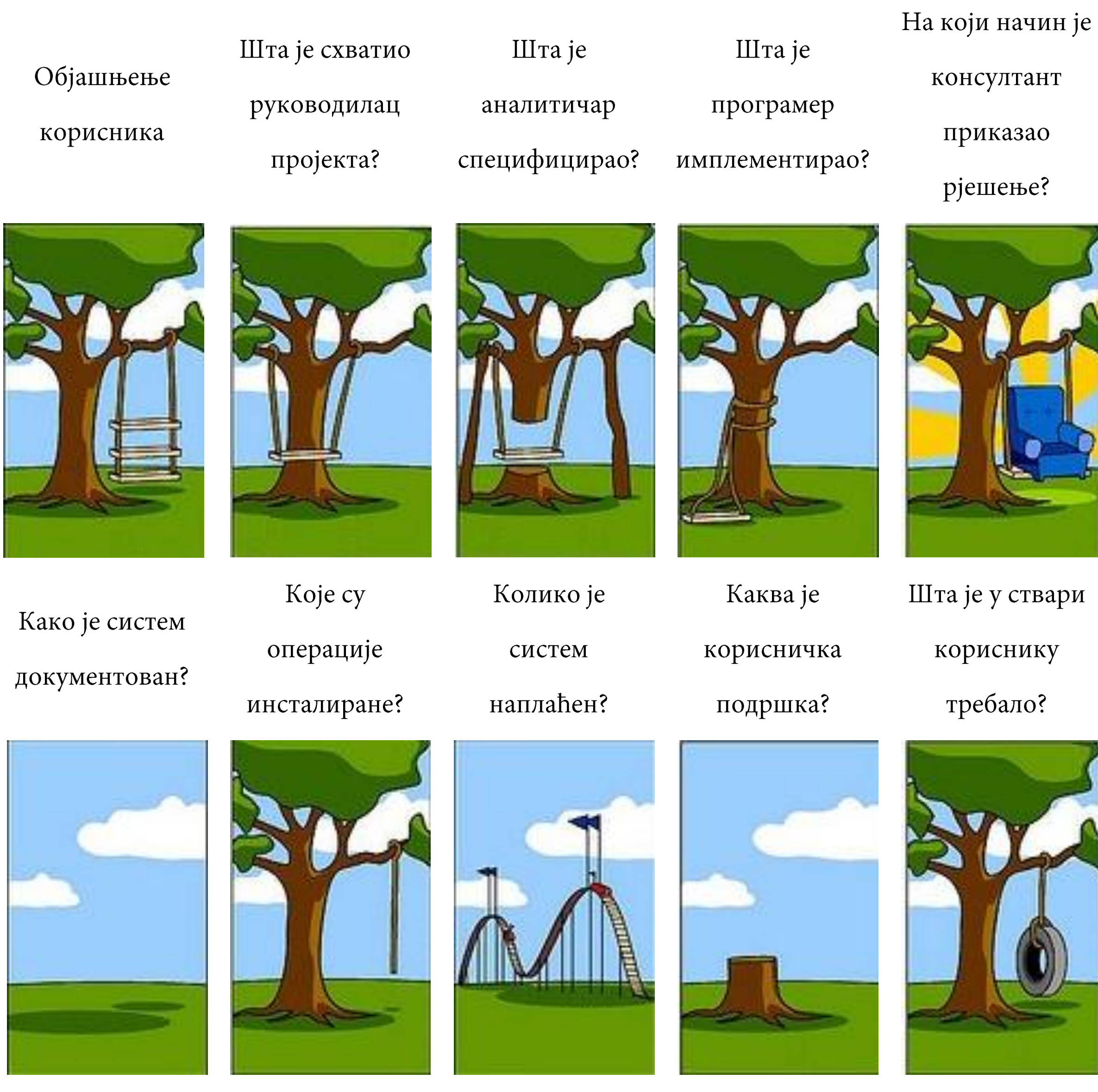
У циљу отклањања двосмислености у корисничким захтјевима, и свих опасности које због њих могу да проистекну, у овом раду биће препоручен нешто другачији приступ спецификацији случајева коришћења и прављења доменског модела. Наиме, умјесто да се доменски модел прави на основу спецификације случајева коришћења, по овом приступу, спецификација случајева коришћења ће бити заснована на претходно креираном доменском моделу. Дакле, прије спецификације случајева коришћења, потребно је направити прву верзију или „скицу“ доменског модела. Доменски модел ће на овај начин постати полазна тачка

[9] Патерни анализе (Martin Fowler) и патерни доменског модела (David C. Hay) представљају готове дијелове доменског (али и других) модела које су сачинили доменски експерти, а које је могуће искористити тј. интегрисати у модел неког новог система везаног за сличан домен. Идеја је заснована на становишту да само доменски експерти, дакле експерти у одређеној области, у довољној мјери познају домен проблема како би за њега могли направити модел. Из ове перспективе, инжењери софтвера у овој фази имају улогу само да едукују доменског експерта како да користи одређену технику моделовања. [Fowler97]

приликом спецификације софтверских захтјева. Ово не значи да ће до краја фазе прикупљања захтјева полазни доменски модел остати непромијењен. Напротив, како се буду уочавали детаљи корисничких захтјева приликом спецификације случајева коришћења, који нису били узети у обзир у прављењу иницијалног доменског модела, ови детаљи ће бити укључени и у доменски модел. Доменски модел ће бити основа за пројектовање статичких дијелова система, док ће спецификација случајева коришћења бити основа за пројектовање динамичких дијелова система. Статички дијелови система представљају његову структуру, док динамички дијелови одређују понашање система. На овај начин биће могуће, од самог почетка прикупљања и спецификације корисничких захтјева, повезати статички (доменски модел) и динамички (случајеви коришћења) дио модела система, и одржати њихову семантичку конзистентност. Ова чињеница је кључна за наредне фазе развоја ако ће бити ослоњене директно на спецификацију случајева коришћења (што је случај са свим методама које су вођене случајевима коришћења – *use case driven*). Доменски модел у овом контексту има улогу рјечника појмова и термина који ће осигурати њихову семантичку конзистентност приликом описивања домена проблема.

У прилог овом приступу иде теза која се може наћи у литератури: „Ако су двосмислени захтјеви непријатељ, доменски модел је прва линија одбране!“ [Rosemberg07]. Из тог разлога једна од агилних метода развоја софтвера – ICONIX Process, чији је творац Doug Rosenberg, као полазну тачку у прикупљању корисничких захтјева поставља управо прављење доменског модела. Традиционални приступ спецификацији случајева коришћења подразумијева спецификацију која треба да буде технолошки независна и апстрактна, у том смислу да не залази у домен рјешења. Међутим, овакав приступ често доводи до двосмислености у захтјевима, чије се разрјешење препушта програмерима који тумаче и имплементирају ове захтјеве. Сљедећа слика (Слика 7) на духовит начин приказује проблем који настаје када

спецификација захтјева није довољно прецизна па на тај начин допушта различита тумачења од стране различитих учесника у пројекту [Oakland89].



Слика 7: Различита тумачења од стране различитих учесника у пројекту [Oakland89]

Базирање спецификације случајева коришћења на доменском моделу у *ICONIX Process* методи оправдавају и чињеницом да ако се жели објектно орјентисана анализа и пројектовање које ће бити засновано на спецификацији случајева коришћења, тада се случајеви коришћења морају везати за објекте, јер случајеви коришћења сами по себи нису објектно орјентисани.

У наставку ће бити размотрени различити модели података, технике за креирање модела као и нотације за њихову спецификацију. Како је предмет рада аутоматизација корисничког интерфејса заснована на моделу случајева коришћења, а претходно су описане везе и значај модела података, тј. доменског модела за спецификацију случајева коришћења, биће приказне технике и нотације које ће на одговарајући начин креирати модел података, како би се он могао користити у процесу генерисања корисничког интерфејса.

Постоји више дефиниција модела података, али се у већини модели података дефинишу као модели којима се реални системи описују преко скупа објеката (ентитета), њихових атрибута и међусобних веза. Такође, модел података се може дефинисати као специфичан теоријски оквир помоћу кога се специфицује, пројектује и имплементира нека конкретна база података [Lazarević03].

Моделовање података представља активност у софтверском процесу у којој се истражују структуре орјентисане на податке [Ambler04]. Модели података могу се правити за различите потребе, од концептуалних – доменских модела високог нивоа, до физичког модела података. Са аспекта објектно орјентисаног развоја софтвера, моделовање података је слично прављењу дијаграма класа, са разликом што се модел података односи искључиво на податке, док се у дијаграму класа, поред атрибута могу приказивати и понашања. Међутим, ако се узме у обзир препорука да концептуални – доменски модел не треба да садржи методе тј. понашање, тада је једина разлика у нотацијама којима ће бити приказивани модели.

У литератури [Ambler04] се може наћи сљедећа подјела модела података према општости (иако се на неким мјестима ова подјела сматра подјелом према стилу):

- Концептуални модел података: Ови модели често се називају и доменски модели, и најчешће се користе у комуникацији са доменским

експертима за упознавање концепата из домена пословног система који се пројектује. Креирају се у почетној фази софтверског процеса, и представљају основу за прављење логичког и физичког модела података.

- Логички модел података (Logical data model – LDM): Описују концепте домена значајне за онај дио домена који се тог тренутка моделује. Опсегом може обухватати појединачне подсистеме, а може приказивати и читав систем. Више се користи у традиционалним методама, него у агилним (јер се сматра да не доприноси значајно у пракси).
- Физички модел података (Physical data model – PDM): Приказује интерну шему базе података, табеле, колоне и везе између табела.

У контексту аутоматизације процеса развоја корисничког интерфејса као предмета овог рада, израз доменски модел ће означавати концептуални и логички модел података, док ће физички модел на апликационом нивоу бити представљен објектним моделом, а на нивоу складишта података релационим моделом.

Најраспрострањенији облик чувања података у данашњим апликацијама јесте коришћење релационих база података. Код перзистенције података у Јава апликацијама, најчешће се подразумијева чување Јава објеката у релационој бази података.

Релационе базе података постале су својеврсни стандард у домену перзистенције података. Разлози за то су бројни, али прије свега то је једноставност њиховог креирања и приступа подацима у њима коришћењем SQL-а, као и једноставност структуре модела података – релационог модела. Релационе базе података пружају структурирану репрезентацију података, у табеларном облику,

омогућавају приступ и ажурирање перзистентних података, и обезбјеђују чување интегритета података. Такође, обезбјеђују конкурентност приступу подацима, као и дијељење података од стране различитих корисника или апликација.

У објектно орјентисаним апликацијама перзистенција треба да омогући чување објеката у релационој бази података. При томе се не мисли само на чување појединачних објеката, већ и цијеле мреже узајамно повезаних објеката, која репрезентује одређени објектни модел. Поред објеката који се трајно чувају, у објектно орјентисаним апликацијама постоји велики број тзв. трансијентних објеката. Најчешће у оваквим апликацијама постоји подсистем који треба да обезбиједи материјализацију и дематеријализацију перзистентних објеката, тј. њихову трансформацију у облик погодан за чување у релационим базама података – релациони модел. Дакле перзистентност у објектно орјентисаним апликацијама које користе релациону базу података може се посматрати као процес трансформације објектног модела у релациони модел и обратно.

Основни проблем у моделовању реалних система је чињеница да су готово сви реални системи сложени, првенствено због тога што у њима постоји велики број објеката, њихових сложених веза и њихових атрибута. Општи методолошки поступак за савладавање сложености у опису система је апстракција. Апстракција је контролисано и постепено укључивање детаља у опис система, „сакривање" детаља у описивању система, односно извлачење и приказивање општих, а одлагање описивања детаљних особина неког система. У моделима података користе се сљедеће апстракције [Lazarević03]:

- Типизација (класификација) објеката, односно података: објекти који имају исти скуп особина и исто динамичко понашање могу се представити типом или класом објеката.

- Генерализација: скуп сличних типова објеката представља се општијим генеричким типом, односно надтипом.
- Агрегација: скуп објеката и њихових међусобних веза се третира као јединствени, агрегирани тип.

У литератури се често на различитим мјестима различито терминолошки одређују модели података, и нема јасне границе између техника за моделирање, нотација за моделирање, и самих модела података. У овом раду:

- техникама ће се сматрати активности које омогућавају идентификацију ентитета (објеката) и њихових веза у реалном систему који се моделује;
- нотације ће бити посматране као начини приказивања идентификованих ентитета;
- модели података означаваће реалан облик у коме се налазе подаци (објектни у објектно орјентисаним апликацијама, и релациони у системима за управљање релационим базама података).

У наставку ће бити приказан објектни и релациони модел података, њихове најважније карактеристике, као и разлике које постоје међу њима. Затим ће се објаснити проблеми у трансформацији једног модела у други, као и начини за превазилажење ових проблема. Биће приказане нотације и технике за моделовање података. Овом анализом олакшаће се избор модела, технике и нотације који ће бити коришћени у формирању модела за генерисање корисничког интерфејса.

Објектни модел

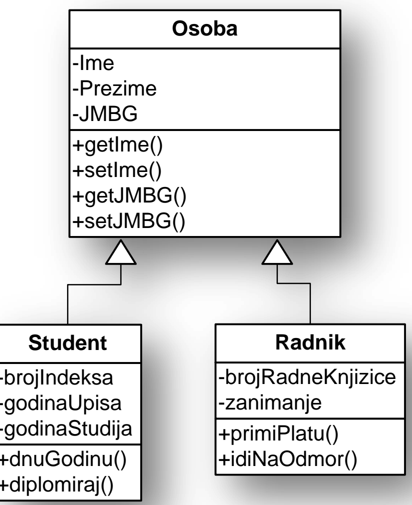
У објектном моделу објекат се дефинише као ентитет који је способан да чува своја стања и који околини ставља на располагање скуп операција преко којих се тим

стањима приступа. Стање објекта представља се вриједностима његових особина: атрибута објеката и његових веза са другим објектима у систему [Lazarević03]. Данас је у развоју апликација преовладао тзв. објектно орјентисани приступ, те се за развој апликација користе програмски језици базирани на коришћењу објектног модела. У основи свих тих програмских језика налази се концепт класе. Класа се дефинише као апстрактна представа скупа објеката који имају исте особине. На овај начин се у објектном моделу постиже типизација тј. класификација, као основна апстракција података. Класа се састоји од атрибута и метода. Атрибутима се дефинише стање, а методама понашање класе. Објекат представља једно конкретно појављивање (примјерак, инстанцу) своје класе.

Osoba
-Ime -Prezime -JMBG
+prikaziPodatke() +provjeriValidnostJMBGa()

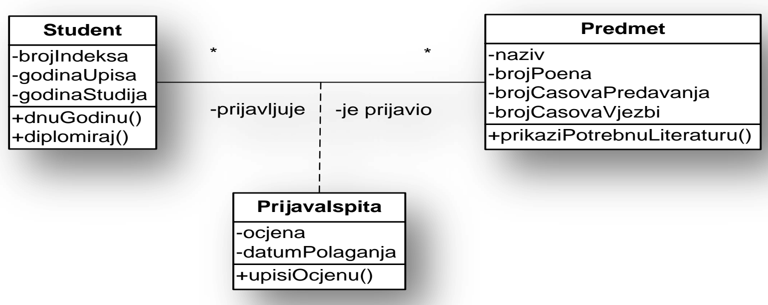
Слика 8: Примјер класе

Поред класификације, објектни модел подржава и генерализацију – апстракцију у којој се скуп сличних типова тј. класа објеката представља општијим генеричким типом тј. надтипом. За различите класе објеката каже се да су слични уколико имају неке заједничке особине или понашање.



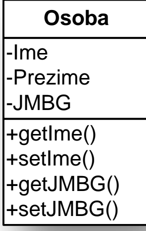
Слика 9: Примјер насљеђивања

Такође, објектни модел подржава и агрегацију – апстракцију података којом се скуп постојећих објеката и њихових међусобних веза представља новим, јединственим – агрегираним типом. Агрегирани објекат као своје компоненте има објекте који чине агрегацију, а може да има, као цјелина и своје сопствене атрибуте, и као цјелина може да ступа у везу са другим објектима у моделу [Lazarević03].



Слика 10: Примјер агрегације

Поред наведених апстракција, објектни модел подржава и концепт учаурења[10] којим се стање објекта учаурује на тај начин што се другим објектима у систему стављa на располагање скуп операција којим се може промијенити стање објекта. На овај начин се сакривају детаљи имплементације стања објеката, и омогућавају промјену стања објекта од стране других објеката у окружењу а да при томе објекти из окружења не познају саму имплементацију стања објекта који мијењају. На овај начин постиже се већи степен независности између различитих компоненти система које су међусобно повезане, јер се измјеном имплементације стања једног објекта не захтијева измјена објеката који су са тим објектом повезани.



Слика 11: Примјер енкапсулације (учаурења)

Комбинацијом наведених особина које подржава објектни модел добија се могућност формирања моћних механизама који објектном моделу дају предност приликом развоја пословних апликација у односу на остале моделе података. То су прије свега концепти: преклапања метода[11], прекривања (редефинисања) метода[12],

[10] учаурење - *encapsulation*

[11] Преклапање метода – *overloading*: могућност да се у истој класи декларишу двије или више метода са истим именом. Ове методе морају се разликовати у броју и/или типу параметара.

[12] Прекривање метода – *overriding*: могућност да се у надкласи и у подкласи декларише метода која се исто зове, има исти потпис као и тип који враћа, а да им се имплементације разликују.

компатибилности објектних типова[13], касног повезивања метода[14], полиморфизма, апстрактних класа, интерфејса и тд.

Релациони модел

У релационом моделу систем се представља преко скупа релација. Релације представљају првенствено типове објеката, а затим и неке везе у моделу[15]. Релација се може представити као табела, гдје су колоне атрибути релације, а врсте N-торке[16] релације. Један или више атрибута релације јединствено идентификују једну N-торку у релацији, и тај скуп атрибута се назива *примарни кључ* релације. Релациони модел је вриједносно орјентисан и везе се у њему остварују преко вриједности атрибута на тај начин што се примарни кључ неке релације појављује као атрибут друге релације који тада постаје тзв. *спољни кључ*. Тада се преко вриједности тог атрибута може успоставити веза између н-торки из те двије релације.

Груписањем атрибута у релацију постиже се типизација тј. класификација као основна апстракција података. Релација се у моделу представља на тај начин што се име релације пише испред заграда у којима се налазе атрибути који чине релацију. Примарни кључ релације обиљежава се подвлачењем атрибута који га чине.

[13] Компатибилност објектних типова: објекат основне класе може добити референцу на било који објекат изведен из те класе.

[14] *Late binding* – касно повезивање метода: својство које имају тзв. виртуалне методе – методе које се повезују са објектом који их је позвао у вријеме извршења програма, за разлику од обичних метода које се повезују са објектом у вријеме компајлирања. Представља један од предуслова за стварање јаког полиморфизма. У програмском језику Јава, све методе су виртуалне.

[15] Веза са кардиналношћу пресликавања *више према више,* реализује се прављењем нове, агрегирајуће релације, која се односи на саму везу између објеката.

[16] N-торке представљају редове у табелама, у којима се налазе конкретне вриједности атрибута из домена дефинисаних релацијом, и представљају конкретно појављивање одређене релације.

Artikal (<u>ArtikalID</u>, Naziv, DatumProizvodnje, Opis, JedinicaMjere, Cijena)

Слика 12: Примјер релације

Генерализација, односно специјализација се у релационом моделу такође постиже преко вриједности атрибута. Примарни кључ надтипа се приказује као примарни кључ у табели подтипа.

Artikal (<u>ArtikalID</u>, Naziv, Opis, JedinicaMjere, Cijena)

PrehrambeniArtikal (<u>ArtikalID</u>, RokTrajanja, Sastav)

AparatZaDomaćinstvo (<u>ArtikalID</u>, TehničkeOsobine, UputstvoZaUpotrebu, Garancija)

Слика 13: Примјер специјализације

Међутим, треба нагласити да је ефекат генерализације тј. специјализације само привидан, јер је релациони модел семантички сиромашан, и посматрајући релацију ни по чему се не може закључити да је она надтип неке друге релације, нити се за неку релацију може тврдити да је она подтип неке друге релације.

Преко вриједности атрибута могуће је постићи и ефекат агрегације. Агрегирана релација ће тада садржати примарне кључеве релација које учествују у агрегацији, а поред тога може садржати и неке сопствене атрибуте, јер се агрегирана релација посматра као и свака друга релација у систему. Ни за агрегирану релацију се у релационом моделу ни по чему не може закључити да представља агрегацију двије или више постојећих релација. Примјер за агрегирану релацију може бити добављање артикала од стране добављача уколико је у систему дозвољено да један артикал добавља више добављача, а да један добављач може да добавља више артикала. Тада би релациони модел приказали на сљедећи начин:

Artikal (<u>ArtikalID</u>, Naziv, Opis, JedinicaMjere, Cijena)

Dobavljač (<u>DobavljačID</u>, Naziv, Adresa, KontaktTelefon)

Dobavlja(<u>DobavljačID, ArtikalID</u>, Marža)

Слика 14: Примјер агрегације

Коришћењем SQL-a[17], стандардног језика релационих система за управљањем базама података (DBMS[18]), на основу релационог модела креирају се табеле, скуп ограничења на вриједности атрибута, као и динамичка правила интегритета[19]. Након тога могуће је извршити SQL упите којима се манипулише над подацима у релационој бази података. Четири основне операције над подацима су **C**reate (insert), **R**etrieve (select), **U**pdate, **D**elete, који се скраћено називају *CRUD* операције.

Проблем трансформације

У објектно орјентисаним апликацијама перзистенција треба да омогући чување објеката у релационој бази података, при чему се подаци трансформишу из објектног у релациони модел. Како су ови модели различити постоји читав низ неподударања које је неопходно превазићи како би процес био могућ. Ова неподударања проистичу из различите намјене ових модела. Релациони модел се

[17] SQL – Structured Query Language

[18] Database Management Systems – Системи за управљање базама података

[19] Под динамичким правилима интегритета подразумијевају се правила којима се одржава интегритет података при извршавању операција одржавања базе података. Када се наруши интегритет података неком од операција (insert, update, delete) извршава се једна од акција: Restrict, Cascade, Set null, Set default.

користи у релационим базама података чија је основна улога трајно чување података, док се објектни модел користи у објектно орјентисаним апликацијама које имају за циљ реализација пословне логике система. Превазилажењу разлика између ова два модела дуго времена се посвећује велика пажња, те се у литератури област која се тиче овог проблема назива *object/relational impedance mismatch* или само *impedance mismatch* [Peak06].

У наставку ће бити приказано неколико кључних разлика између објектног и релационог модела које чине основне проблеме у трансформацији једног модела у други.

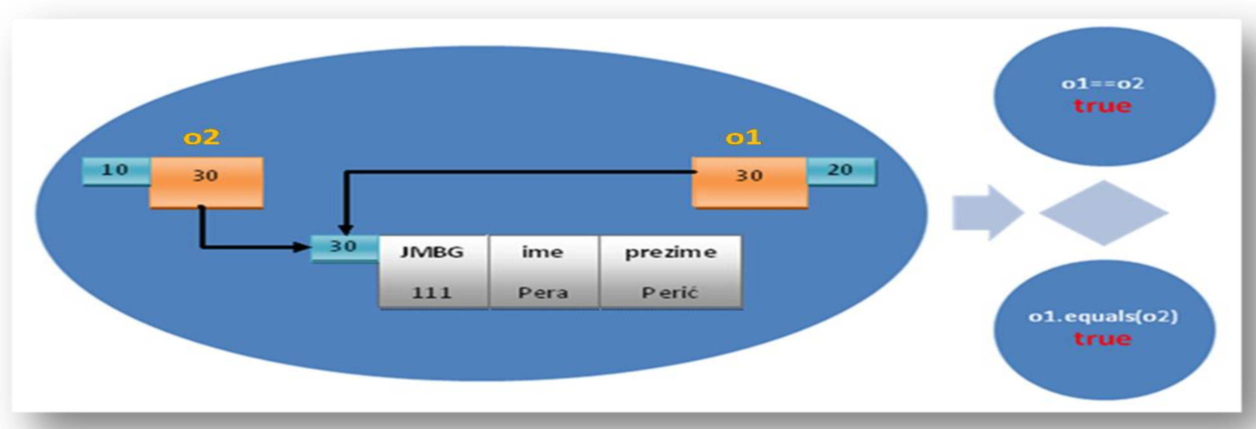
Концепт насљеђивања

Један од најважнијих концепата објектног модела јесте могућност да један објекат наслиједи и прошири особине неког другог објекта. Већ је речено да релациони модел не подржава концепт насљеђивања, те се проблем своди на трансформацију хијерархије објеката у низ релација. Дакле, проблема не би ни било када не би било хијерархије у објектном моделу, али како објектно орјентисане апликације моделују реалне пословне процесе, тада се концепт насљеђивања намеће као најбољи начин за прецизно дефинисање објектног модела, који пружа довољно флексибилности софтверском систему како би могао да испрати промјене у пословном систему. Ових предности се не треба одрећи приликом пројектовања објектног модела, што значи да је потребно имплементирати механизам за трансформацију хијерархије објеката у релације и то тако да се постигне ефекат специјализације тј. генерализације у релационом моделу. Међутим, у пракси се поред специјализације често прибјегава неким другим стратегијама. Разлог лежи у томе што

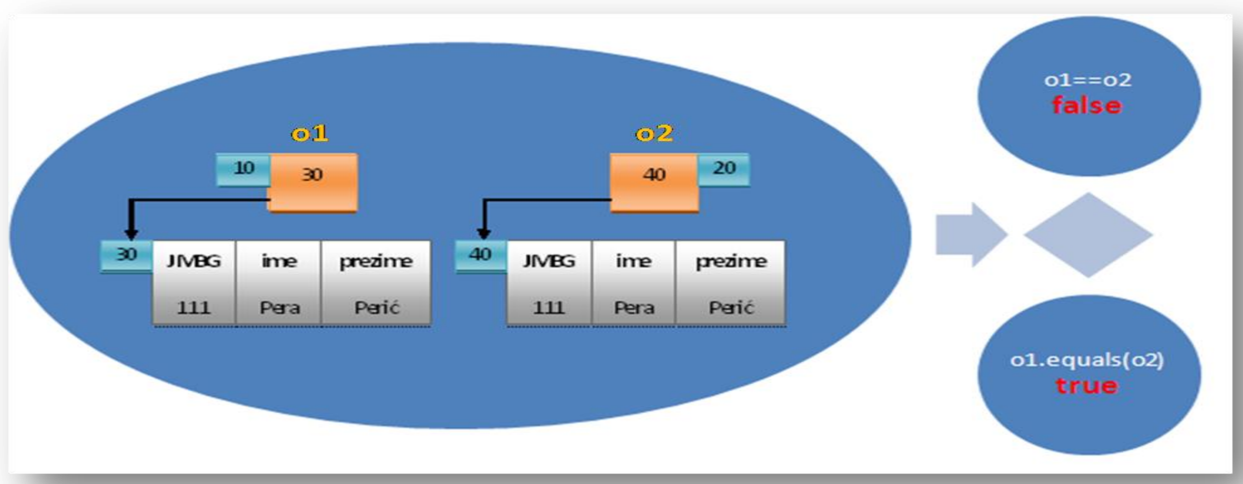
се приликом манипулисања подацима, који се налазе у табелама које чине специјализацију, јавља потреба за великим бројем спајања табела кроз SQL упите, а то у многоме смањује перформансе извршења. Тако су у употреби и стратегија прављења посебних табела за сваки изведени објекат са свим пољима које објекат насљеђује у хијерархији. Мана ове стратегије је немогућност контроле јединствености примарног кључа у различитим табелама. Такође, у употреби је и стратегија која предвиђа постојање једне табеле за све објекте у хијерархији. Тада се уводи посебна колона у табели која представља тип објекта из хијерархије, па се приликом манипулације подацима прво сазнаје о ком се објекту ради, па се онда извлаче подаци потребни том типу објекта. Мана ове стратегије лежи у чињеници да ће свака н-торка имати поља која су празна и која се не користе.

Концепт идентитета

Када се говори о идентитету објеката треба раздвојити постојање објекта као *референце* и *вриједности* објеката. Објекат постоји независно од своје вриједности. У Јава програмском језику поређење објеката врши се на два начина:



Слика 15: Објекти са референцама на исту меморијску локацију



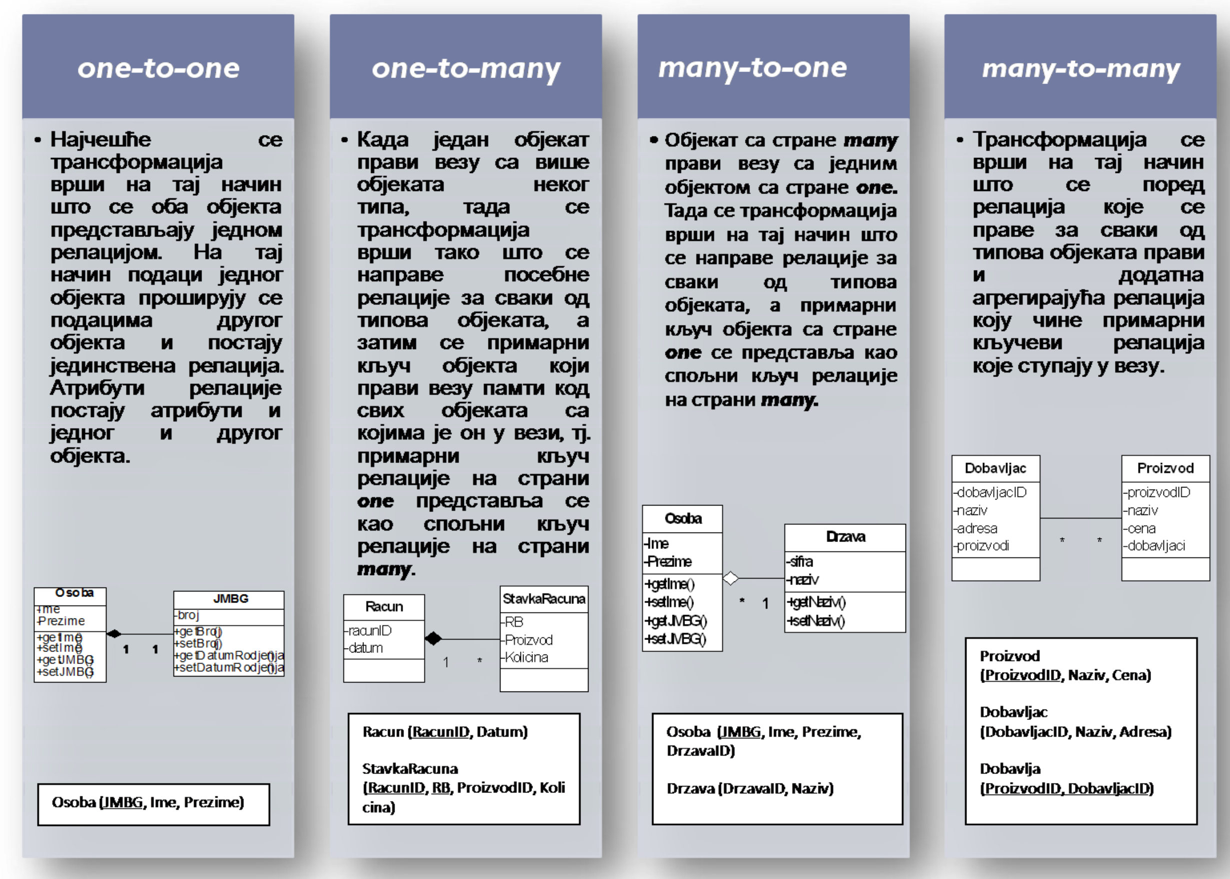
Слика 16: Објекти са референцама на различите меморијске локације које садрже исте вриједности

Први начин пореди само референце на објекат, и уколико су једнаки то значи да се ради о истом објекту, док други начин пореди саме вриједности које објекти садрже, те могу бити једнаки, иако су референце различите. Такође, референце на објекат идентификују и тип којем објекти припадају, јер објекат може добити референцу на објекат исте класе или класе која је изведена из класе референтног објекта. Везе између објеката успостављају се на тај начин што један објекат садржи референцу на други објекат. Са једне стране, ова референца говори о томе који је тип објекта са којим се прави веза, а са друге стране преко ње се приступа вриједностима атрибута другог објекта.

У релационом моделу једини идентификатор сваке н-торке јесте вриједност која се налази у ћелији која представља примарни кључ релације. Зато се за сваки објекат из објектног модела, при трансформацији у релациони модел одређује атрибут који ће га јединствено идентификовати (примарни кључ релације), и који ће приликом повезивања релација играти улогу референце у релацији са којом се врши повезивање (спољни кључ).

Асоцијације

Проблем асоцијација односи се на трансформације веза које се успостављају између објеката у објектном моделу у везе између релација у релационом моделу. У релационом моделу везе се успостављају коришћењем спољних кључева, тј. атрибут који представља примарни кључ у првој релацији представља се као атрибут у релацији која је повезана са првом релацијом. У објектном моделу постоји неколико врста асоцијација: 1-1 (one-to-one), 1-* (one-to-many), *-* (many-to-many), и њих је у релационом моделу потребно обезбиједити коришћењем спољних кључева.



Слика 17: Асоцијације

Релациони модел користи податке као интерфејс према окружењу, дакле интерфејс чини структура релације, док интерфејс објектног модела чини понашање објеката, тј. јавне методе које представљају сервисе које неки објекат пружа. Концептом учаурења постиже се потпуно сакривање имплементације структуре објекта.

Процес трансформације објектног у релациони модел и обратно, треба да омогући пресликавање атрибута објеката који чине његову структуру у атрибуте релације. Како концепт интерфејса у објектном моделу не подразумијева директан приступ његовим атрибутима, потребно је повезати сваки атрибут релације са одговарајућом методом, тј. методом која обезбјеђује приступ траженом атрибуту објекта.

Нотације за приказивање модела података

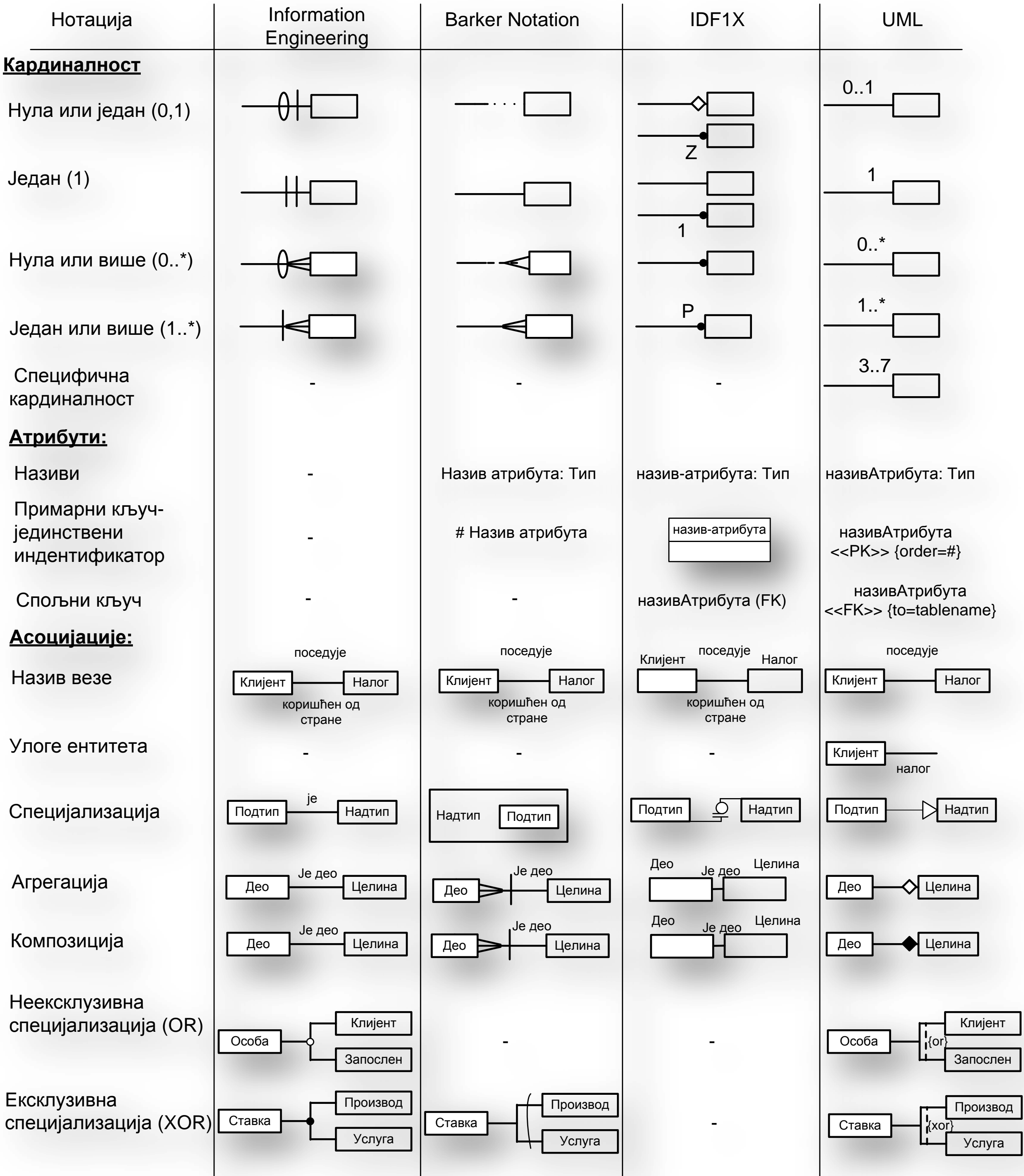
Постоји више нотација за приказивање модела података. Најчешће су ове нотације везане или за чисти објектни модел (UML – дијаграм класа), или су замишљени као визуелни приказ објеката и њихових веза који омогућава једноставну (или чак аутоматску) трансформацију у релациони модел.

Данас, најкоришћенија нотација за моделирање података јесте Модел објекти-везе (*Entity relationship modelling*). Ова нотација чији је аутор Peter Chen [Chen76], развијена је у другој половини седамдесетих година прошлог вијека, а основни циљ је био приказивање логичке – семантичке структуре домена (посматраног реалног система), а не реализацију у одређеној бази података. За приказ се користе дијаграми

објекти-везе. На основу ове нотације настало је јако много различитих варијанти, а данас је највише у употреби верзија која је уведена као стандард у америчкој државној администрацији, и зове се IDEF1X нотација.

Модел објекти-везе неки аутори сматрају засебним моделом података, други техником за креирање модела података, а у овом раду модел објекти-везе биће сматран нотацијом за приказивање модела података, као и нпр. UML дијаграм класа.

У литератури се могу наћи детаљна поређења различитих нотација за креирање модела података [Hay99], а у наставку ће бити приказани различити начини графичког приказивања елемената модела података и то за *Information Engineering, Barker, IDF1X* и *UML* нотацију (Слика 18).



Слика 18: Графички приказ елемената различитих нотација за креирање модела података [Ambler04]

На претходној слици приказане су најчешће коришћене нотације. Међу њима, највећу популарност имају UML и IDF1X нотације. UML нотација је данас постала

стандард у објектно орјентисаној анализи и пројектовању, док се IDF1X нотација паралелно са UML-ом користи за физичко моделирање података, јер је развијен велики број CASE[20] алата, међу којима је најпознатији *ERwin*[21], који омогућавају аутоматску трансформацију ове нотације у релациони модел, и генерисање шеме базе података за велики скуп постојећих система за управљање базама података. Нотација коју је развио Richard Barker такође се доста користи јер је прихваћена као стандард за моделовање од стране Oracle корпорације.

Технике за креирање модела података

Како је раније поменуто, техникама за креирање модела података се сматрају активности које омогућавају идентификацију ентитета (објеката) и њихових веза у реалном систему који се моделује. Идентификовани ентитети и везе међу њима се потом одабраном нотацијом могу графички приказати, а након тога и трансформисати у жељени модел података. Постоје различите технике за прављење модела података. Једна од најједноставнијих јесте коришћење листе категорија концептуалних класа, а често су у употреби и технике које препоручују тзв. патерне анализе или патерне доменског модела. Међу најчешће коришћеним техникама, вјероватно због једноставности, јесте и идентификација концептуалних класа уочавањем именица у корисничком захтјеву. Овдје се може закључити да су технике за прављење модела података у ствари технике за прављење доменског модела. То и јесте случај ако се посматра објектни модел као модел података, као најчешће

[20] CASE – Computer Added Software Enginnering – софтверски алати који се користе као подршка одређеним активностима у животном циклусу софтвера.

[21] CA ERwin Data Modeler – Софтверски алат за моделирање података, који омогућава визуелизацију сложених структура података, као и синхронизацију визуелног модела са шемом базе података у различитим системима за управљање базама података.

коришћен приступ у објектно орјентисаном пројектовању софтвера. Зато ће идентификовани концепти и везе међу њима касније моћи да се прикажу нотацијом која је одговарајућа жељеном моделу података (нпр. UML дијаграмом класа за објектни модел, или IDEF1X нотацијом за релациони модел). У наставку ће детаљније бити описане наведене технике.

Техника за прављење доменског модела коришћењем листе категорија концептуалних класа

Једна од најједноставнијих техника за прављење доменског модела јесте коришћење листе категорија концептуалних класа. Ова техника препоручује идентификацију појмова у корисничком захтјеву независно од технике којом је специфициран. Сваки појам се сврстава у одређену категорију из унапред дефинисане листе. У наставку ће бити приказана једна могућа листа категорија на основу које је могуће идентификовати концептуалне класе у посматраном систему [Larman98]. Пракса је да се направи табела са двије колоне, гдје ће се у првој колони наћи категорије, а у другу колону ће се уписивати идентификовани концепти за одређену категорију.

Табела 2: Листа категорија за идентификкацију концептуалних класа

Категорија	Идентификовани концепти
Физички или опипљиви објекти	Каса, производ...
Спецификације, пројекти или описи ствари	Начин означавања производа бар кодом.
Мјеста	Адресе продајних објеката са географским подацима.
Трансакције	Издавање рачуна, сторнирање рачуна,

	плаћање кешом, плаћање картицом, наручивање, пријем робе...
Објекти коришћени у трансакцијама	Производ, рачун, картица...
Улоге људи	Купац, продавац...
Објекти који могу да садрже ствари	Рачун садржи више ставки.
Ствари садржане у објектима који могу да садрже ствари	Ставка садржи производ, уз количину производа који се купује.
Други компјутерски или електромеханички системи који окружују посматрани систем	Информациони системи који врше верификацију кредитних картица и омогућавају плаћање.
Организације	Добављачи, банке...
Догађаји	Отварање продавнице, затварање продавнице, попис, куповина производа, набавка производа.
Закони и правила	Законска регулатива.
Каталози	Каталози производа од добављача, лагер листа...
Документа и записи везани за финансије, посао, уговори, правне ствари	Уговори са добављачима, издати рачуни...
Финансијски инструменти и услуге	Куповина на кредит.
Упутства, документа, објашњења, књиге	Документа стандарда квалитета

Главна мана овог приступа је у томе што осим што се једноставно уочавају ентитети, ова техника не омогућава идентификацију атрибута и веза, тј. тешко се уочава разлика између ентитета, атрибута и веза, па се ова техника често комбинује са другим техникама.

Патерни анализе и патерни доменског модела

Патерни анализе (Martin Fowler) [Fowler97] и патерни доменског модела (David C. Hay) [Hay95] представљају готове дијелове доменског (али и других) модела које су сачинили доменски експерти, а које је могуће искористити тј. интегрисати у модел неког новог система везаног за сличан домен. Идеја је заснована на становишту да само доменски експерти, дакле експерти у одређеној области, у довољној мјери познају домен проблема како би за њега могли направити модел. Из ове перспективе, инжењери софтвера у овој фази имају улогу само да едукују доменског експерта како да користи одређену технику моделовања [Fowler97].

Овај приступ не полази од претпоставке да ће домен проблема који се моделује увијек да се подудара са моделима који су изложени у литератури [Fowler97] [Hay95], већ се дати модели могу или дјелимично искористити, или се на основу ентитета и веза између ентитета у овим моделима може наћи сличност са неким новим доменом који се моделује. Поред великог броја примјера који су изложени у овим радовима, дати су и неки генерални принципи којих би се требало придржавати приликом моделовања.

Идентификација концептуалних класа уочавањем именица у корисничком захтјеву

Међу најчешће коришћеним техникама, вјероватно због једноставности, јесте и идентификација концептуалних класа уочавањем именица у корисничком захтјеву. Ову технику представио је Russel Abbott [Abbott83], и предвиђа лингвистичку акализу софтверских захтјева. У свакој реченици уочавају се именице и именичке синтагме и оне постају кандидати за концептуалне класе тј. њихове атрибуте. Међутим, ако се

узму у обзир узроци двосмислености код случајева коришћења, јасно је да ова техника може довести до грешака које се тада преносе и на структуру система. Наиме, двије различите именице могу указивати на исти концепт. Зато аутоматско пресликавање именица у класе или атрибуте није могуће. Ипак, развијени су софтверски алати који аутоматски препознају концепте из захтјева на енглеском језику, њихове атрибуте и везе, на основу чега је могуће направити модел. Ова техника се најчешће не користи самостално, већ се обично комбинује са листом категорија концептуалних класа, како би се избјегли потенцијални проблеми.

3.5. Шаблони за спецификацију случајева коришћења

Раније је напоменуто да је једна од предности спецификације случајева коришћења у односу на остале технике прикупљања и спецификације софтверских захтјева у њиховој структурираности. Ова структура није строго дефинисана, па различити аутори у различитом облику представљају случајеве коришћења. Оно што се подразумијева да ће спецификација случаја коришћења садржати јесте један или више актора који ће користити случај коришћења, назив случаја коришћења, као и један или више сценарија за случај коришћења (основни и алтернативна сценарија).

Данас се у литератури могу наћи на стотине различитих формата, тј. шаблона (*formats* или *templates*) за спецификацију случајева коришћења. Ови шаблони су настали као потреба стандардизовања спецификације случајева коришћења како би се осигурала комплетност спецификације и олакшала комуникација између чланова тима који врши спецификацију. Сваки од ових шаблона прављен је за одређену намјену, па се тешко може говорити о шаблону који је генерално прихватљив. Ипак, показало се да шаблон који се користи за спецификацију има велики утицај на

употребљивост тако написаног случаја коришћења у фази пројектовања, као и на квалитет пројектованог софтвера. Везано за пројектовање корисничког интерфејса, коришћење неких шаблона олакшава процес прављења доброг корисничког интерфејса, док други шаблони или не пружају корисне информације за одвијање овог процеса, или га чак отежавају [Constantine03].

У својој књизи *Writing Effective Use Case* [Cockburn00] Alistair Cockburn приказује неколико репрезентативних шаблона за писање случајева коришћења. Већина представљених шаблона је у тзв. потпуно сређеном облику (*fully dressed form*).

У наставку ће бити приказана четири најпотпунија шаблона за спецификацију случајева коришћења:

- шаблон за случај коришћења у потпуно сређеном облику;
- шаблон за случај коришћења у табеларном облику са једном колоном;
- шаблон за случај коришћења у табеларном облику са двије колоне;
- шаблон за случај коришћења препоручен од стране RUP (*Rational Unified Process*) методе.

3.5.1. Шаблон за случај коришћења у потпуно сређеном облику

Шаблон за случај коришћења у потпуно сређеном облику подразумијева [Cockburn00]:

- спецификације случајева коришћења писањем у једној колони без употреба табела;
- акције сценарија су нумерисане;
- не користе се разгранате (*IF-THEN-ELSE*) структуре у сценаријима;
- алтернативна сценарија су такође нумерисана на тај начин да се може пратити када настају у основном сценарију.

Име: <треба да означи циљ случаја коришћења у форми кратког глаголског израза у активу >

Контекст коришћења: <детаљније објашњени циљеви, и, по потреби, ситуације у којима се случај појављује>

Опсег: <опсег пројектовања, систем који се пројектује>

Ниво: <one of: Summary, User-goal, Subfunction>

Примарни актор: <назив улоге коју има примарни актор или опис>

Заинтересоване стране: <листа заинтересованих страна и значај случаја коришћења за сваку од њих>

Предуслов: <стање система које се очекује прије покретања случаја коришћења>

Минимални постуслови: <минимални скуп задовољених услова за прекид извршења>

Постуслови успјешног извршења: <стање система које се очекује након успјешног извршења>

Догађај окидач (*Trigger*): <догађај који покреће извршење случаја коришћења>

Основни сценарио: <кораци сценарија од догађаја окидача до испуњења циља>

<корак #> <опис акције>

Алтернативна сценарија: <један за другим алтернативни сценарио, гдје ће сваки да референцира корак из основног сценарија у коме се појављује>

<корак основног сценарија у коме се појављује> <услов кад се појављује>: <акција или скуп акција>

Листа варијација у односу на технологију или податке: <варијације које могу довести до бифуркационих тачака у сценарију>

<варијација # > <објашњење варијације>

Додатне информације: <Било шта што је потребно да се дода у спецификацији

Слика 19: Шаблон за случај коришћења у потпуно сређеном облику

3.5.2. Шаблон за случај коришћења у табеларном облику са једном колоном

У неким пројектима случајеви коришћења се приказују у табеларном облику. На овај начин се постиже већа прегледност спецификације, а са друге стране, то може бити сметња приликом писања саме спецификације [Cockburn00].

Случај коришћења #	<треба да означи циљ случаја коришћења у форми кратког глаголског израза у активу>	
Контекст коришћења	<опис контекста коришћења, уколико је потребно>	
Опсег	<опсег пројектовања, систем који се пројектује>	
Ниво	<једно од : Резиме, Основни задатак, Подфункција>	
Примарни актор	<назив улоге коју има примарни актор, или опис>	
Заинтересоване стране и значај	Заинтересоване стране	Значај
	<Заинтересована страна>	<Значај>
	<Заинтересована страна>	<Значај>
Предуслови	<стање система које се очекује прије покретања случаја коришћења>	
Минимални постуслови	<минимални скуп задовољених услова за прекид извршења>	
Постуслови успјешног извршења	<стање система које се очекује након успјешног извршења>	
Догађај окидач (Trigger)	<догађај који покреће извршење случаја коришћења>	
Опис	Корак	Акција
	1	<...>
	2	<...>
	3	<...>
Алтернативна сценарија	Корак	Акција гранања
	1а	<услов који је довео до гранања>: <акција или скуп акција>
Листа варијација у односу на технологију или податке		
	1	<листа варијација>

Слика 20: Шаблон за случај коришћења у табеларном облику са једном колоном

3.5.3. Шаблон за случај коришћења у табеларном облику са двије колоне

Овај шаблон за спецификацију случајева коришћења препоручује коришћење двије колоне за приказивање акција сценарија. У лијевој колони се уписују акције актора, док се десна колона користи за приказ акција система. Шаблон са двије колоне препоручује се од стране неких аутора [Constantine03] [Wirfs-Brock93] из разлога што се у њему јасно могу раздвојити акције актора и акције система, чиме се наглашава дијалог између актора и система, што може бити веома корисно, прије свега за пројектовање корисничког интерфејса. Ако се узме у обзир чињеница да се случај коришћења прави за пројектанте са једне, и наручиоце са друге стране, раздвајање дијела који се тиче акција корисника од акција које извршава систем помаже наручиоцима да се фокусирају на дефинисање својих намјера приликом коришћења одређене функционалности система, а да их при томе не збуњују акције система, које они и не треба да разумију. Са друге стране, у разговору са наручиоцима, аналитичар сазнаје детаље о доменском моделу и осталим информацијама везаним за функционисање система, и допуњава десну колону – акције система. У наставку ће бити представљен шаблон којега су аутори [Constantine03] назвали структурирани суштински случај коришћења (*Structured essential use case*), а који приказује акције сценарија у двије колоне. Аутори овог шаблона, поред циља (*goal*) корисника који случај коришћења мора да оствари, нагласак стављају и на намјере (*intentions*) корисника. Наиме, разлика између циља и намјере је врло мала, али веома значајна, нарочито за пројектовање корисничког интерфејса. Циљ представља крајње жељено стање система, дакле статичке особине система након завршетка случаја коришћења, док се намјере односе на динамички аспект, и представљају пут доласка до тог циља. Овај пут управо се специфицира кроз интеракцију корисника и система, што је најдиректније повезано са корисничким интерфејсом који ће посредовати у тој

комуникацији. Дакле, намјере се могу посматрати као кораке на путу до одређене дестинације, која представља циљ [Constantine03].

Као једна од мана овог шаблона наводи се отежан унос корака сценарија, јер се кораци уносе час са једне, час са друге стране табеле, што може представљати сметњу и скренути пажњу аналитичара приликом писања. Аутори овог шаблона зато не инсистирају на коришћењу табела, већ само предлажу да се акције система и актора јасно могу разликовати, па наводе могућност увлачења параграфа који се односе на акције које извршава систем у односу на оне које извршава актор, или коришћење различитих боја, или различита сјенчења.

Оно што, такође разликује овај шаблон од осталих јесте увођење појма *асинхрони изузетак* (*asynchronous extension*). Ови изузеци представљају она алтернативна сценарија која се могу догодити у било ком тренутку извршења случаја коришћења, за разлику од осталих алтернативних сценарија за које се тачно зна послије ког корака у основном сценарију случаја коришћења могу да наступе. Асинхрони изузеци се, по овом шаблону, специфицирају прије спецификације основног сценарија. Такође се препоручује да се спецификација ове врсте изузетака изостави приликом разговора са корисницима, из разлога што корисници најчешће нису у могућности да разумију њихову намјену.

Идентификација

Шифра: Сврха:

Везе са другим случајевима коришћења

Процес

Предуслови:	
Намјере корисника	**Одговорности система**
асинхрони изузеци	асинхрони изузеци
(кораци)	(кораци)

Слика 21: Шаблон за случај коришћења у табеларном облику са двије колоне

3.5.4. Шаблон за случај коришћења препоручен од стране RUP (Rational Unified Process) методе

Metoda *Rational Unified Process* – RUP препоручује шаблон сличан шаблону у *потпуно сређеном облику,* са разликом да нумерисање корака у сценарију није обавезно. У наставку је приказан овај облик представљања случаја коришћења.

1. Назив случаја коришћења
    1.1. Кратак опис
        ...текст...
    1.2. Актори
        ...текст...
    1.3. Догађаји окидачи (Triggers)
        ...текст...
2. Ток догађаја
    2.1. Основни ток
        ...текст...
    2.2. Алтернативни токови
        2.2.1. Услов 1
            ...текст...
        2.2.2. Услов 2
            ...текст...
        2.2.3. ...
3. Посебни захтјеви
    3.1. Платформа
        ...текст...
    3.2. ...
4. Предуслови
    ...текст...
5. Постуслови
    ...текст...
6. Тачке проширења

Слика 22: Шаблон за случај коришћења препоручен од стране RUP (Rational Unified Process) методе

3.6. Препоруке за спецификацију корака у сценарију случаја коришћења

Шаблони за спецификацију случајева коришћења описују елементе структуре које сваки случај коришћења треба да садржи. Међутим, нити један од приказаних шаблона не описује детаљније структуру основног сценарија, тј. структуру корака у

сценарију, осим што се по неким шаблонима препоручује нумерација корака, или раздвајање акција које извршава актор од оних које извршава систем.

У литератури [Cockburn00] [Rosemberg07] се може наћи препорука да се кораци у сценарију пишу у активу, и то реченицама које треба да имају тзв. именица-глагол-именица структуру (*Noun-Verb-Noun*) односно субјекат-предикат-објекат. При томе прва именица означава актора или систем као субјекат у реченици (у зависности од тога да ли се ради о акцији коју извршава актор или систем), глагол представља предикат у реченици и означава радњу тј. акцију која се извршава, док друга именица означава објекат над којим се врши радња. Поред субјекта, предиката и објекта, корак у случају коришћења може да садржи и додатне информације или објашњења везана за акцију сценарија. Примјер који слиједи приказује дио основног сценарија [22]случаја коришћења „Наручивање производа“:

1. Клијент бира производ који жели да наручи.
2. Клијент уноси количину производа коју жели да наручи.
3. Клијент позива систем да сачува наруџбину.
    4. Систем провјерава унесене податке.
    5. Систем чува наруџбину.
    6. Систем приказује поруку о успјешно сачуваној наруџбини.

Слика 23: Дио основног сценарија случаја коришћења „Наручивање производа“

Структура субјекат-предикат-објекат у корацима сценарија случаја коришћења

Раније у овом раду објашњена је потреба за повезивањем доменског модела и спецификације случајева коришћења, па је закључено да ће спецификација случајева коришћења бити заснована на претходно креираном доменском моделу. Ако се

[22] Како би се направила јасна разлика између корака које извршава актор и корака које извршава систем, у приказу сценарија користе се двије колоне, тј. направљено је увлачење параграфа који садрже кораке које извршава систем.

детаљније проуче кораци у сценарију случаја коришћења, може се уочити да се као објекат над којим се врши радња појављује ентитет у доменском моделу, или атрибут неког ентитета[23], у зависности од нивоа детаљности који се користи при спецификацији случаја коришћења[24]. Управо на овај начин биће прављена веза између случајева коришћења и доменског модела, тј. модела података који ће бити креиран на основу доменског модела. Тако се за сваки случај коришћења јасно може уочити скуп ентитета на које се односи случај коришћења, који представља подскуп скупа ентитета из доменског модела, чиме се јасно одређују границе случаја коришћења са аспекта структуре система.

У литератури [Jacobson87] [Jacobson93] [Ochodek08] се често акција која се извршава у кораку сценарија случаја коришћења описује као трансакција у комуникацији између актора и система. У даљем тексту умјесто термина трансакција користиће се термин акција. Ivar Jacobson уочава четири типа акција[25] којима се описује корак у сценарију:

1. Актор шаље захтјев и податке систему.

---

[23] Ентитет у објектном моделу података представља класу док у релационом моделу представља релацију.

[24] Сценарио приказан у примјеру је могао бити приказан и са мање детаља. Наиме, прва два корака сценарија могу бити сажета у један, и гласити: Клијент уноси ставку наруџбине, при чему је ставка наруџбине ентитет који садржи атрибуте производ и количина.

[25] У оквиру извођења наставе на предмету Пројектовање софтвера [Vlajic03], ради лакше анализе захтјева дефинисане су следеће врсте акција, у зависности од тога да ли их извршава актор или систем. Актор изводи једну од три врсте акција:

а) Актор Припрема Улаз (Улазне Аргументе) за Системску Операцију (АПУСО).

б) Актор Позива систем да изврши Системску Операцију (АПСО).

ц) Актор извршава НеСистемску Операцију (АНСО).

Систем изводи две акције у континуитету:

а) Систем извршава Системску Операцију(СО):

б) Резултат системске операције (Излазни аргументи (IA)) се прослеђује до актора.

2. Систем врши валидацију података и извршава операцију којом мијења унутрашње стање.
3. Систем приказује резултат извршења операције као одговор актору.

Ако се покушају уочити ови типови акција у примјеру сценарија случаја коришћења „Наручивање производа“, могуће је примијетити да постоје кораци у сценарију који не припадају нити једном од поменута четири типа:

Табела 3: Типови акција које одговарају одређеним корацима у сценарију

Кораци сценарија	Тип акције
1. Клијент бира производ који жели да наручи.	X
2. Клијент уноси количину производа коју жели да наручи.	X
3. Клијент позива систем да сачува наруџбину.	**Актор шаље захтјев и податке систему.**
4. Систем провјерава унесене податке.	**Систем врши валидацију података.**
5. Систем чува наруџбину.	**Систем извршава операцију којом мијења унутрашње стање.**
6. Систем приказује поруку о успјешно сачуваној наруџбини.	**Систем приказује резултат извршења операције као одговор актору.**

Кораци сценарија чије акције не припадају нити једном од поменута четири типа, управо су кораци које извршава актор припремајући улазне податке прије него што се позове систем да изврши одређену операцију. Ове акције актор најчешће врши над компонентама корисничког интерфејса намијењеним за унос података. Како су ове акције веома важне за пројектовање корисничког интерфејса апликације, што је предмет овог рада, биће дефинисани додатни типови акција које ће обухватити припрему улазних података неопходних за извршење сценарија. За припрему улазних података биће користишћена два додатна типа акција: унос и одабир. Разлика између ова два типа се огледа у томе што је за одабир потребно актору унапријед припремити скуп дозвољених улазних података, док при уносу актор сам задаје улазни податак. Ова разлика има директне посљедице на пројектовање корисничког интерфејса апликације.

Ако се узму у обзир додатни типови акција, акције се могу груписати на сљедећи начин (типови акција су подвучени):

- **Акције актора**
  - Припрема улазних података
    - Унос
    - Одабир
  - Слање захтјева и података систему
- **Акције система**
  - Извршење операције која мијења унутрашње стање система, чији је предуслов извршавање валидације података
  - Приказивање резултата извршења операције актору

3.7. Коришћење прототипова у инжењерингу захтјева

Како је раније поменуто, најчешћи узрок неуспјеха софтверских пројеката јесу недостаци и грешке у корисничким захтјевима [Frost07] [Johnson00]. Један од начина за њихово отклањање у раним фазама јесте прављење прототипова корисничког интерфејса. Корисник преко корисничког интерфејса ступа у интеракцију са системом, и управља његовим стањима. У објектно орјентисаним апликацијама, стања система учаурују се у доменске објекте система који, заједно са везама и ограничењима, чине доменски модел софтверског система. Како случајеви коришћења описују интеракцију између корисника и система, која се реализује преко корисничког интерфејса и има за циљ промјену стања система, веза између ова три дијела система морала би бити узета у обзир већ у фази прикупљања корисничких захтјева. Када би се узела у обзир ова чињеница приликом креирања прототипа корисничког интерфејса, такав интерфејс би се могао користити за валидацију како случајева коришћења, тако и модела података [Fitzgerald05]. Под валидацијом се подразумијева провјера усаглашености корисничких захтјева са пројектованим доменским моделом.

Са друге стране, прављење прототипова, или само њихово скицирање (*storyboard*) значајно може помоћи приликом дефинисања случајева коришћења [Rosemberg07]. Често, више заинтересованих страна у софтверском пројекту има потешкоћа да софтверске захтјеве представи на тај начин да буду јасни и недвосмислени свим учесницима. Све док у корисничком захтјеву постоје елементи који нису потпуно једнозначни и јасни свим учесницима у пројекту, они представљају велики ризик за даљи развој софтверског производа. Док пројектанти и програмери фаворизују коришћење модела у представљању корисничких захтјева, а аналитичари текстуалну репрезентацију захтјева, корисници најчешће желе да виде спецификацију

захтјева која је ближа реализацији производа. Један од начина да се овај проблем превазиђе јесте да се корисници обуче да користе и читају моделе који се користе у спецификацији корисничких захтјева. Међутим, овај приступ није лако изводљив. Прије свега, учење неке од техника моделовања изискује вријеме, а тиме се повећавају и трошкови пројекта, могући су отпори од стране корисника, а проблем представља и стање постојећих алата за моделовање. Ако функционалности алата за моделовање нису комплетне, и ако корисник не може у модел укључити све што мисли да је значајно, тада ови алати умјесто да помогну, могу представљати сметњу у комуникацији између корисника и пројектаната. Пракса је показала да прављење прототипова може знатно убрзати процес прикупљања корисничких захтјева и отклонити многе потенцијалне нејасноће.

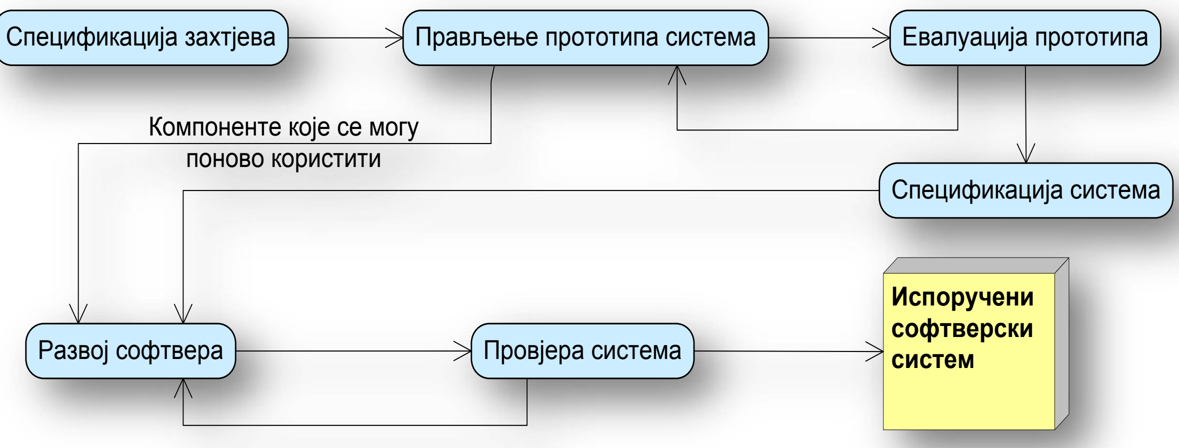
Прототипови су много лакше схватљиви како за кориснике, тако и за пројектанте и програмере, и не изискују додатну обуку корисника. Брзи развој прототипа дозвољава прављење представе циљаног система која помаже да се уклоне нејасноће и уоче недостаци у корисничким захтјевима. Прототипови не морају увијек бити везани за кориснички интерфејс, већ се такође могу користити у раним фазама развоја софтверског производа и за друге аспекте система. Прототип система може послужити за прављење вертикалног пресјека архитектуре система, како би се унапред предвидјели могући проблеми везани за перформансе система [Berenbach09].

Прототипови се могу подијелити у двије групе:

- **Прототипови који се бацају** (*throw-away prototype*)
- **Еволутивни прототипови** *(evolutionary prototype*)

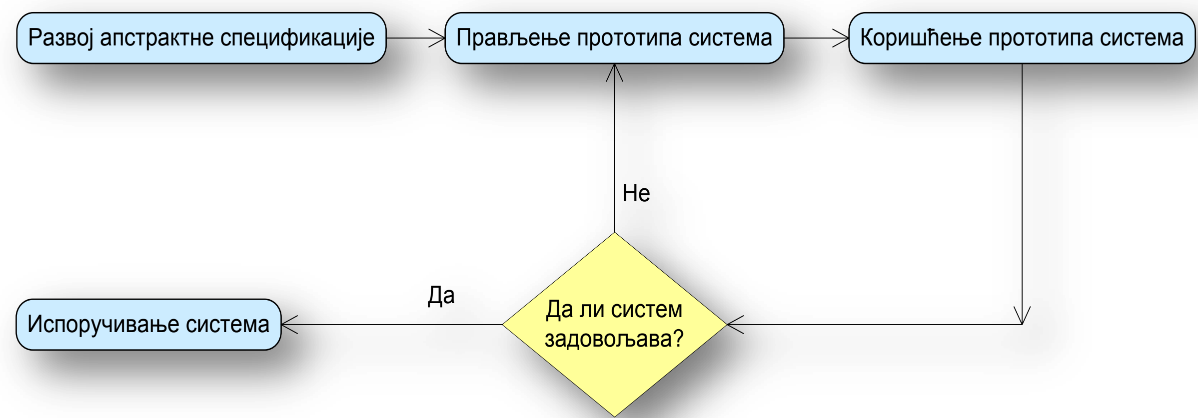
Прототипови који се бацају (throw-away prototype) су прототипови који се обично имплементирају како би се што боље открили захтјеви и могући проблеми у њима[26], па се затим одбацују. Систем се након тога развија од почетка, најчешће коришћењем неке друге технологије. Прво се праве за оне захтјеве који су најмање познати.

[26] Као члан Лабораторије за софтверско инжењерство учествовао сам у развоју неколико софтверских система које је подразумијевало коришћење *прототипова који се бацају* у циљу детаљне анализе и валидације захтјева. Један од најсложенијих система у чијем развоју је коришћен овај приступ је Костмод, верзија 4.0, за потребе Министарства одбране Краљевине Норвешке. И поред тога што је наручиоц направио веома детаљну спецификацију захтјева, прототип система, који је направљен коришћењем Microsoft Access алата, помогао је да се уоче многе непрецизности, па чак и грешке и контрадикторности у захтјевима. Добијени прототип је био функционалан, а поред информација о жељеном корисничком интерфејсу, пружио је много корисних информација које су везане за пословну логику система. Развој прототипа трајао је око 2 мјесеца. Након завршене фазе анализе у првој итерацији развоја, овај прототип је напуштен, а систем је имплементиран Јава технологијом. Оваквим приступом направљена је својеврсна превенција настанка бројних проблема за чије би исправљање било потребно много више времена, да су били уочени касније у току имплементације. Са друге стране, вријеме утрошено за развој прототипа није било занемарљиво, а након његове израде није постојала ни једна линија кода крајњег рјешења. Управо ове чињенице за мене су представљале снажан мотив да покушам да направим алат који ће пружати све предности које пружа прототип, а са друге стране смањити потребно вријеме и напор потребан за његов развој, са крајњим циљем да добијени прототип буде искоришћен као полазна верзија система, која ће се касније надограђивати како би се добило крајње рјешење. На тај начин умјесто прототипа који се баца добио би се еволутивни прототип који је извршив и чији је развој аутоматизован. Управо током развоја система Костмод направљена је прва верзија генератора који представља основу за алат и приступ који је предмет овог рада.



Слика 24: Процес развоја коришћењем прототипова који се бацају

Еволутивни прототипови *(evolutionary prototype)* се развијају у раним фазама софтверског процеса, а затим се кроз више итерација дограђују, све до завршетка развоја система. Прво се праве за захтјеве који су најбоље познати.



Слика 25: Процес развоја коришћењем еволутивних прототипова

Прављење прототипа мора бити брзо како би били доступни што раније у процесу развоја, и морају бити направљени на тај начин да се могу лако модификовати у зависности од измјена и допуна у захтјевима корисника. Из ових

разлога, често се дешава да програмери током развоја прототипова често не воде рачуна о поновном коришћењу (*Reuse*) развијених компоненти. У тим случајевима, прототипови се посматрају као "код за бацање", и неће бити саставни дио коначног система. Са друге стране, менаџери подстичу поновно коришћење развијених компоненти за прототип, иако то изискује много више напора у раним фазама развоја. Ипак, овај приступ омогућава да неки прототипови у одређеном тренутку постану дио система који се испоручује. Такође, већ у процесу прављења прототипова развојни тим се упознаје са циљном технологијом, што касније у фази имплементације система повећава ефикасност развојног тима. Овај приступ на крају резултира смањењем трошкова извођења пројекта.

Агилне методе развоја софтвера, као нпр. Scrum, препоручују тзв. Истраживачке прототипове у оним ситуацијама када све функционалности система нису унапред познате, већ се мијењају и откривају током развоја [Berenbach09].

Одређени аспекти апликације могу бити моделовани коришћењем неке описне нотације, која се, коришћењем специјализованог алата који интерпретира дату нотацију, може извршити. На овај начин добија се извршиви прототип апликације који пружа могућност за верификацију предложеног рјешења. Овај приступ се може користити како за прављење "еволутивних" прототипова који ће се даље развијати све до коначне верзије, тако и за прављење прототипова који ће бити "бачени".

У овом раду предлаже се прављење семантички богатог модела корисничких захтјева на основу кога ће се, коришћењем софтверског алата имплементираног за ту намјену, генерисати извршиви програмски код корисничког интерфејса који се може посматрати као извршиви еволутивни прототип који се може допуњавати и прилагођавати специфичним потребама корисника.

4. Упоредна анализа алата за аутоматизацију развоја корисничког интерфејса

У претходне двије деценије, аутоматизација креирања корисничког интерфејса софтверских система представља тему која је била предмет многих истраживања. [Raneburger12] Резултат ових истраживања представљају алати чија је основна улога смањење времена и напора потребног за развој корисничког интерфејса. Међутим, до сада се нити један од ових алата није наметнуо на тржишту и око себе окупио шири круг корисника [Kennard10]. Разлоге за неприхватање постојећих алата од стране корисника различити аутори виде како у недостацима самих алата, тако и у недостацима везаним за кориснички интерфејс апликација креиран коришћењем ових алата, о чему је више ријечи било у поглављу 2.3.4.

Циљ овог поглавља је да се дође до одговора на неколико питања:

1. Који су разлози за чињеницу да ни један од постојећих алата није успио да око себе окупи широк круг корисника, односно да задовољи потребе привреде.
2. Који су то захтјеви које алати за генерисање корисничког интерфејса морају да испуне како би били прихваћени од стране потенцијалних корисника.
3. Да ли данас актуелни алати испуњавају ове захтјеве.

Како би се добили одговори на ова питања, спроведено је истраживање које се састојало из двије фазе.

Прва фаза истраживања подразумијевала је методу научног испитивања и то техником спровођења анкете. Циљ анкете био је утврђивање преовлађујућег става носилаца софтверске индустрије – искусних софтверских инжењера, по питању елемената које алат за генерисање корисничког интерфејса мора да посједује како би га прихватили и користили у процесу развоја софтвера. Ови елементи биће посматрани као захтјеви које алати за генерисање корисничког интерфејса морају да испуне како би били прихваћени од стране потенцијалних корисника. На основу уочених захтјева биће дефинисани критеријуми за поређење постојећих алата за генерисање корисничког интерфејса.

Друга фаза истраживања односи се прво на одабир актуелних алата за генерисање корисничког интерфејса, затим преглед њихових основних карактеристика и на крају испитивање да ли и у којој мјери задовољавају уочене критеријуме. На крају ће бити представљени резултати упоредне анализе ових алата.

На основу ових резултата биће могуће прецизније утврдити разлоге неприхватања ових алата од стране софтверске индустрије, и дефинисати скуп карактеристика које ће бити основ развоја алата за генерисање корисничког интерфејса као крајњег резултата докторске дисертације.

4.1. Дефинисање критеријума за поређење алата за аутоматизацију развоја корисничког интерфејса

Циљ овог дијела истраживања јесте утврђивање преовлађујућег става носилаца софтверске индустрије – искусних софтверских инжењера, по питању елемената које алат за генерисање корисничког интерфејса мора да посједује како би га прихватили и користили у процесу развоја софтвера. Ови елементи биће посматрани као захтјеви које алати за генерисање корисничког интерфејса морају да испуне како би били прихваћени од стране потенцијалних корисника. На основу уочених захтјева биће дефинисани критеријуми за поређење постојећих алата за генерисање корисничког интерфејса.

Након дефинисања циља, формиран је упитник који треба да садржи питања која ће омогућити закључивање првенствено о критеријумима које алат за генерисање корисничког интерфејса треба да задовољи, али и о карактеристикама које испитаници доминантно препознају као пожељне по сваком критеријуму. Упитник је формиран на основу предистраживања које се састојало у прегледу референтне литературе, анализи постојећих приступа рјешавању овог проблема, разговорима са искусним програмерима и анализи њихових искустава, као и у директним разговорима са ауторима неких од постојећих алата.

До сада није постојало много научних испитивања (интервјуа или анкета) који су се бавили овом облашћу. Значајно истраживање [Myers], које се често цитира у литератури, је спроведено давне 1992. године, а и данас се резултати овог истраживања сматрају релевантним. Зато је дио упитника формиран по узору на ово истраживање. Популацију, на коју се односи испитивање, чине искусни софтверски инжењери који се у свом послу претежно баве развојем пословних апликација. Како нема прецизних података о величини ове популације, циљ је био да узорак буде на

приближном нивоу узорка коришћеног у истраживању [Myers] који је бројао 74 испитаника.

Анкета је спроведена електронским питем (*online* анкета) при чему је за формирање упитника коришћен *Google Forms* [GoogleForms].

О спровођењу анкете обавијештене су вође пројеката из неколико компанија[27] за које постоје сазнања да се баве развојем пословних апликација, са упутством о профилу запослених на софтверским пројектима или пословним партнерима којима треба омогућити попуњавање упитника. Поред тога, обавијештене су и колеге са неколико домаћих и иностраних факултета који се баве облашћу софтверског инжењерства, такође уз упутство о томе који је профил запослених (да раде на развоју пословних апликација), односно сарадника и пословних партнера, који треба да попуне упитник.

За статистичку обраду података коришћен је *IBM SPSS Statistics 20* [SPSS]. Како детаљан приказ свих резултата истраживања изискује много простора, у наставку ће бити приказани само најрелевантнији резултати у односу на предмет докторске дисертације, а они најинтересантнији ће бити приказани графички.

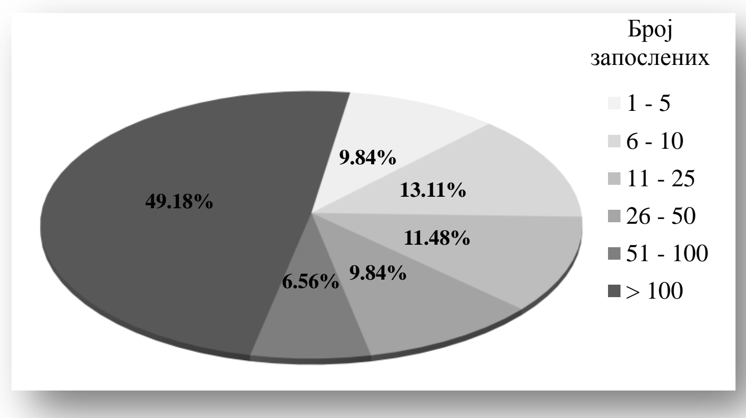
Укупно је прикупљено 70 попуњених упитника, али је из анализе искључено 9, јер нису били комплетни (садржали су одговоре само на пар почетних питања), па је укупан број анализираних испитаника био 61.

Међу испитаницима је било 88,52% мушкараца и 11,48% жена.

[27] У истраживању су учествовали запослени следећих компанија: Microsoft, Комтрејд, Сага, Digit, E-Smart Systems, Fusion, Теленор, Vip, Телеком Србије, Asseco, S&T и др.

Половина испитаника је из Србије, док су остали из Аргентине, Аустралије, Канаде, Колумбије, Чешке, Њемачке, Француске, Индије, Ирана, Италије, Холандије, Португалије, Русије, Хрватске, Сингапура, Швајцарске, Велике Британије и Сједињених Америчких Држава.

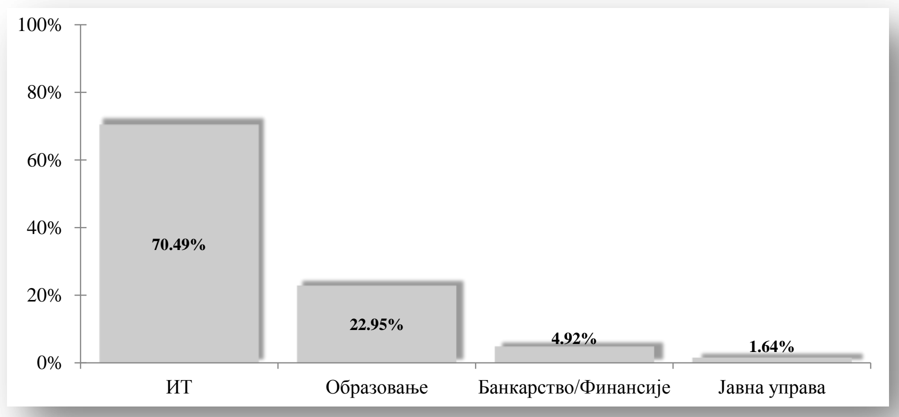
Скоро половина испитаних (49.18%) ради у компанијама са преко 100 запослених, 13.11% у компанијама од 6-10 запослених, 11,48% од 11-25 запослених, по 9,84% испитаних ради у компанијама од 1-5 односно од 26-50 запослених, док 6,56% испитаних ради у компанијама које броје између 51 и 100 запослених (Слика 26).



Слика 26: Број запослених у компанији у којој раде

Просјек година искуства у области софтверског инжењерства је 11,47%.

На питање којој области индустрије односно ком пољу припада њихова компанија, 70,49% испитаних је одговорило да припада области информационих технологија, 22,95% припада образовним институцијама, док остали припадају области банкарства и финансија, као и јавне управе (Слика 27).



Слика 27: Област којој припада компанија у којој раде

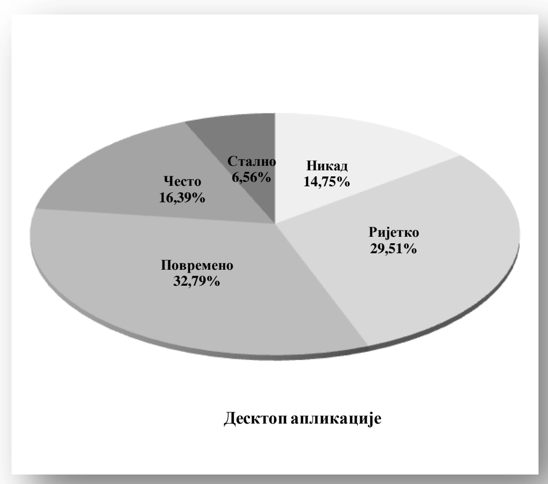
How often do you develop different application types?*

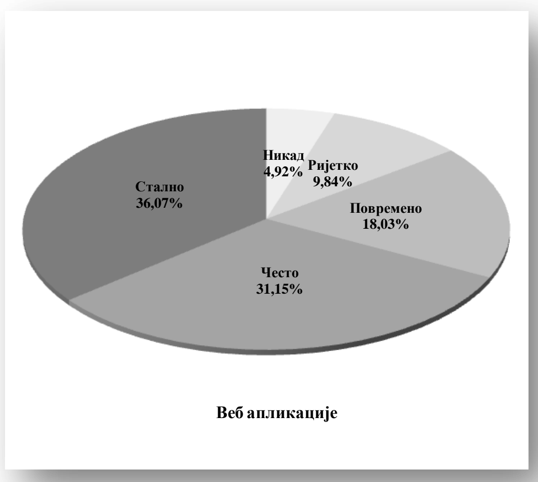
	Never	Rarely	From time to time	Often	Constantly
Desktop applications					
Web applications					
Mobile applications					

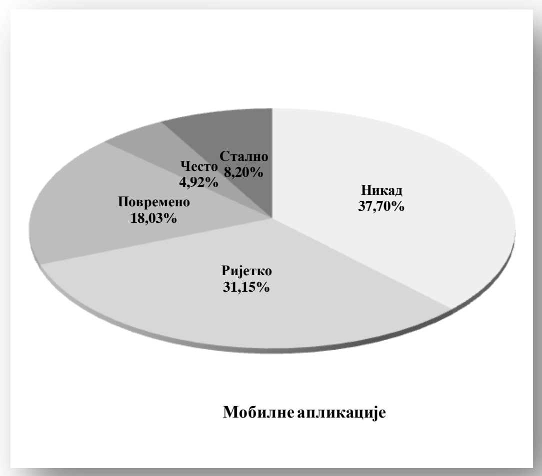
Слика 28: Питање – Учесталост развоја различитих врста апликација

На питање колико често развијају различите врсте апликација (десктоп, веб и мобилне апликације), одговори су следећи (Слика 29):

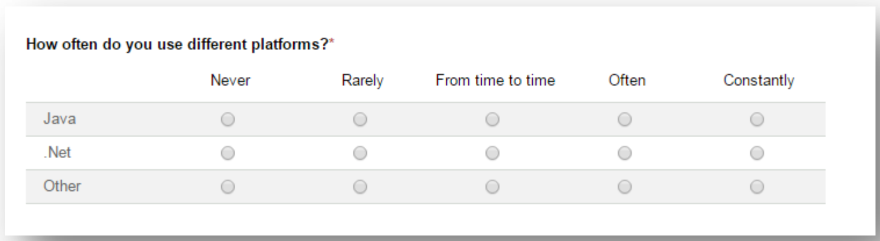
- Десктоп апликације стално развија 6,56% испитаних, често 16,39%, повремено 32,79%, ријетко 29,51%, док их никада не развија 14,75% испитаних,
- веб апликације стално развија 36,07% испитаних, често 31,15%, повремено 18,03%, ријетко 9,84%, док их никада не развија 4,92% испитаних,
- мобилне апликације стално развија 8,20% испитаних, често 4,92%, повремено 18,03%, ријетко 31,15%, док их никада не развија 37,70% испитаних.







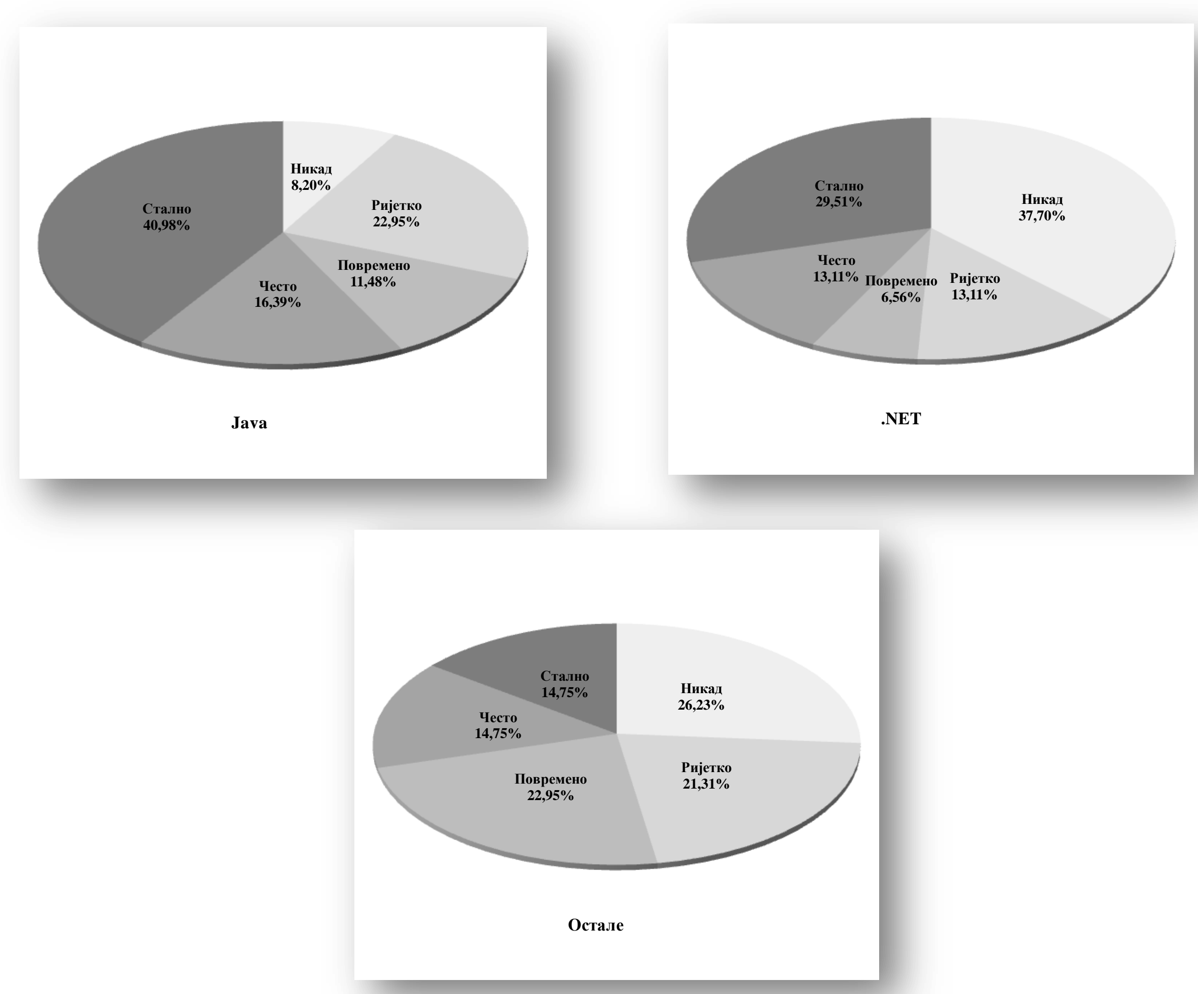
Слика 29: Учесталост развоја различитих врста апликација



Слика 30: Питање – Учесталост коришћења различитих платформи

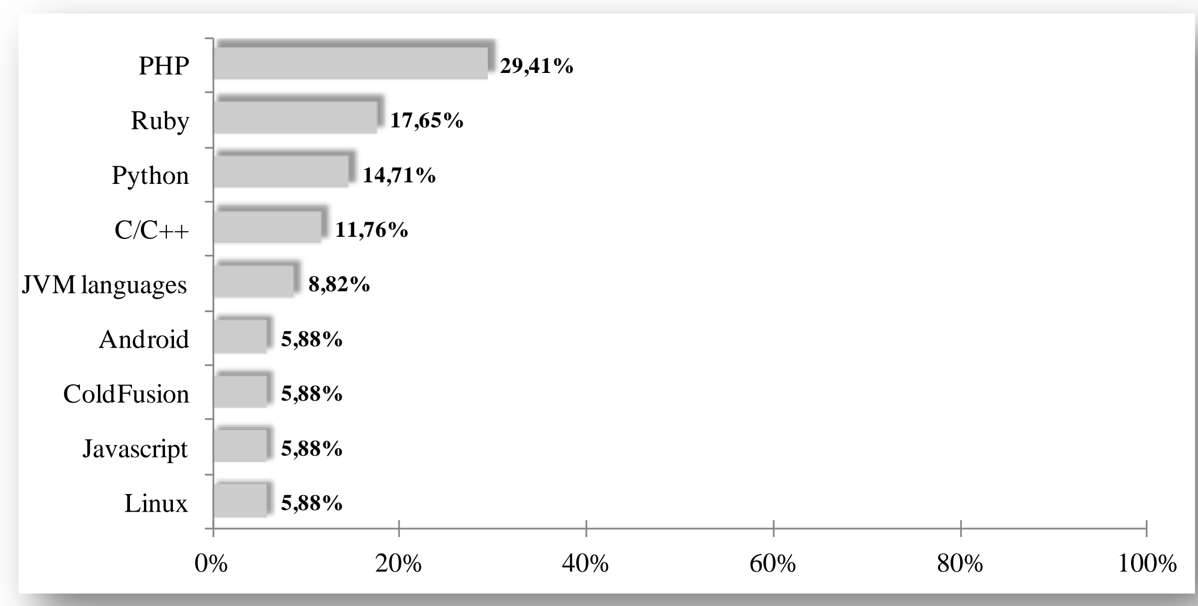
На питање колико често испитаници за развој користе различите врсте платформе (Јава, .NET, остале), одговори су следећи (Слика 31):

- Јава платформу стално користи 49,98% испитаних, често 16,39%, повремено 11,48%, ријетко 22,95%, док је никада не користи 8,20% испитаних,
- .NET платформу стално користи 29,51% испитаних, често 13,11%, повремено 6,56%, ријетко 13,11%, док их никада не користи 37,70% испитаних,
- Остале платформе стално користи 14,75% испитаних, често 14,75%, повремено 22,95%, ријетко 21,31%, док их никада не користи 26,23% испитаних.



Слика 31: Учесталост коришћења различитих платформи

Ни једну другу платформу не користи 4 испитаника, док је 31 навело да користи неку другу платформу. Испитаници су имали могућност да сами унесу врсту платформе коју користе, а њихови одговори су приказани у следећем графикону (Слика 32), уз напомену да је приказан проценат коришћења платформе у односу на испитанике који су дали одговор на ово питање:



Слика 32: Остале коришћене платформе

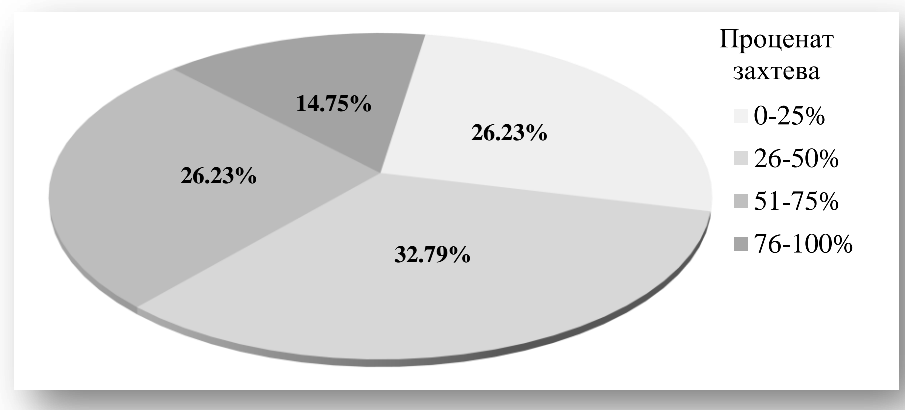
Поред приказаних, испитаници су наводили и следеће платформе (нису приказане на дијаграму јер су се појављивале само по једном):

- PL-SQL/T-SQL/ANSI-SQL
- Delphi
- iOS
- LAMP
- M2T QVT
- Metada Metarepository (own platform)
- Netwaver, HANA for SAP
- PERL
- R/T
- Solaris, aix
- UML/XML/XMI transformation, knowledge representation tools

Based on your experience, what percent of the total number of requirements in a business application can be mostly implemented by the CRUD operations, including validation, without more complex business logic:*
In business applications some requirements can be implemented mostly by the CRUD operations, including validation, but other requirements, besades the CRUD operations and validation, requires implementation of more complex business logic, eg. complex calculations, simulations, or complex database transactions including not only entities manipulated on user interface, but other entities defined by specific business rules.

Слика 33: Питање – Број захтјева који се углавном могу имплементирати коришћењем *CRUD* операција

Следеће питање је од испитаника захтијевало да, према сопственом искуству, одговори колики проценат захтјева у односу на укупан број захтјева, приликом развоја пословне апликације, се скоро потпуно може реализовати само коришћењем *CRUD* операција, укључујући и валидацију, без имплементације сложене пословне логике. 26,23% испитаних сматра да је оваквих захтјева до 25% у развоју пословне апликације, 32,79% сматра да је оваквих захтјева између 26 и 50%, 26,23% сматра да је оваквих захтјева између 51 и 75%, док 14.75% испитаних сматра да број оваквих захтјева превазилази 76% (Слика 34).



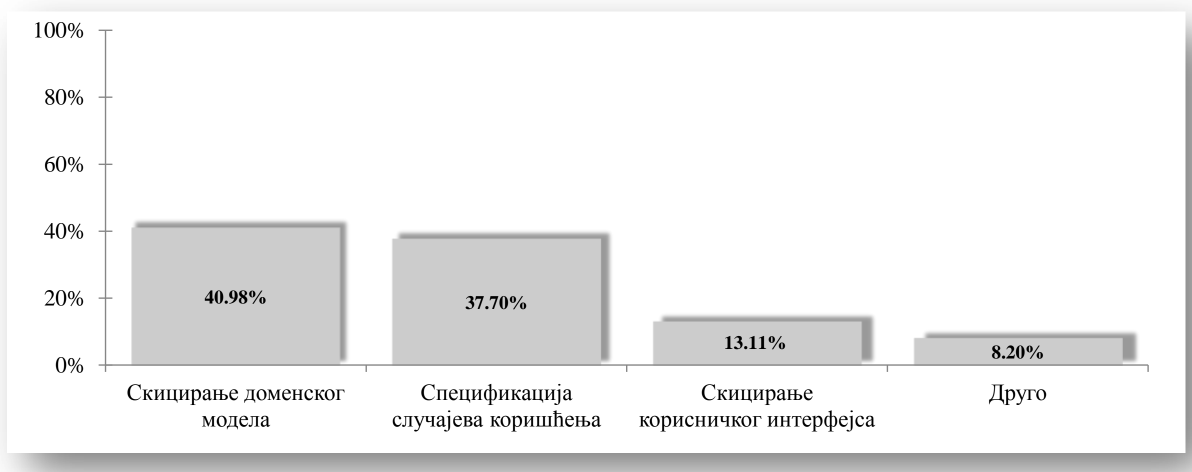
Слика 34: Број захтјева који се углавном могу имплементирати коришћењем *CRUD* операција

After interviewing end users during requirements elicitation, the first step you make in a project is:*

- Sketching of domain model
- Creating the use case specification
- Sketching of user interface
- Other:

Слика 35: Питање – Први корак након прикупљања захтјева од корисника

Након прикупљања захтјева, први следећи корак у процесу развоја је у 40.98% случајева скицирање доменског модела, 37.70% је одговорило да тада врши спецификацију случајева коришћења, 13,11% прави скицу корисничког интерфејса, док је 8.20% испитаних одговорило другачије (Слика 36).



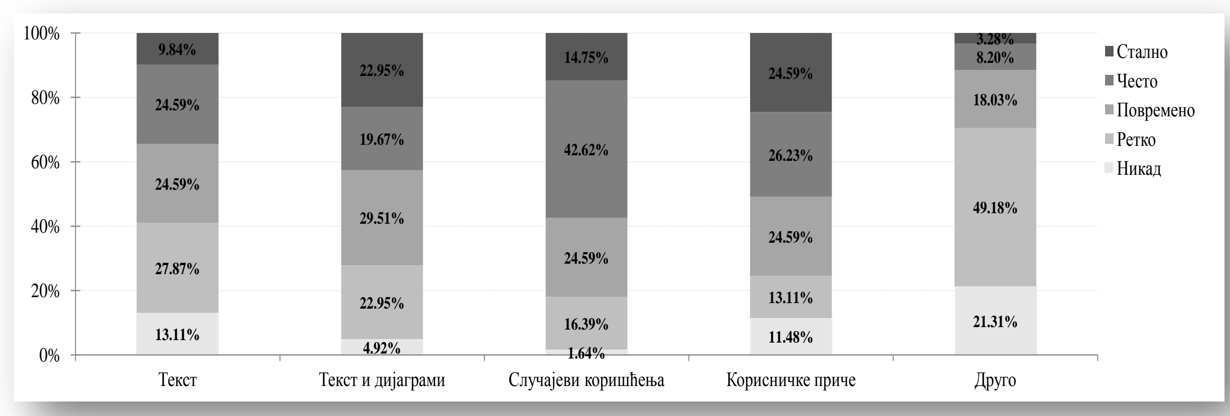
Слика 36: Први корак након прикупљања захтјева од корисника

Your requirement specification is based on:*

	Never	Rarely	From time to time	Often	Constantly
Plain text	○	○	○	○	○
Plain text and diagrams	○	○	○	○	○
Use cases	○	○	○	○	○
User stories	○	○	○	○	○
Something else	○	○	○	○	○

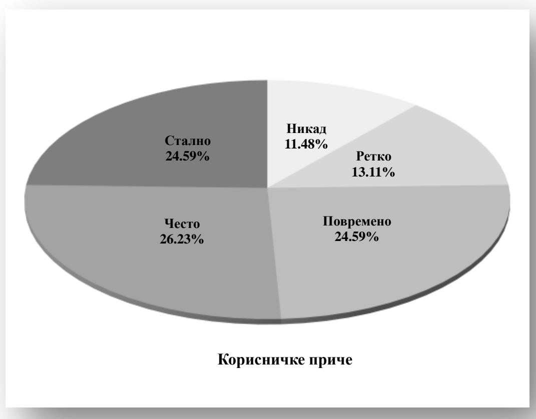
Слика 37: Питање – Учесталост коришћења различитих начина и формата спецификације захтјева

На следећем дијаграму (Слика 38) је приказана учесталост коришћења различитих начина и формата спецификације захтјева (текст, текст и дијаграми, случајеви коришћења, корисничке приче, нешто друго):



Слика 38: Учесталост коришћења различитих начина и формата спецификације захтјева

На следећој слици (Слика 39) су детаљно приказани подаци за случајеве коришћења и корисничке приче, који су издвојени јер представљају најчешће коришћене технике за спецификацију захтјева:



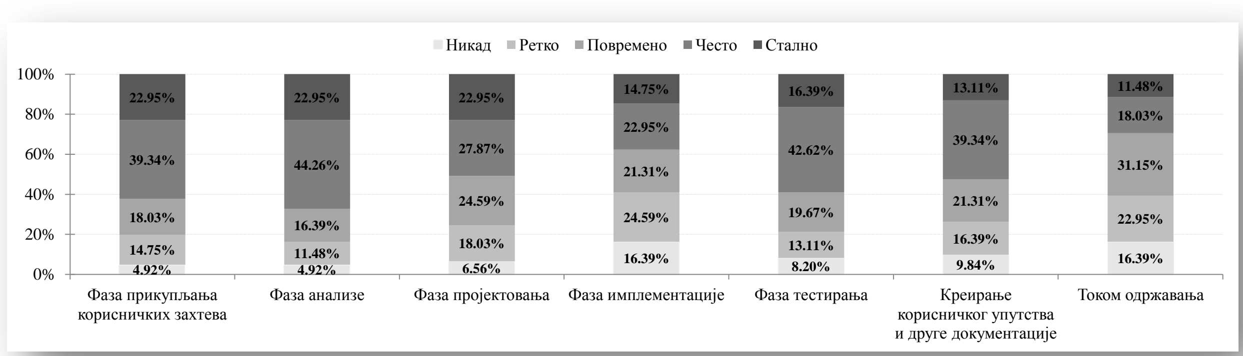
Слика 39: Учесталост коришћења случајева коришћења и корисничких прича приликом спецификације захтјева

How often do you use the use cases in:*

	Never	Rarely	From time to time	Often	Constantly
Requirement phase	○	○	○	○	○
Analysis phase	○	○	○	○	○
Design phase	○	○	○	○	○
Implementation phase	○	○	○	○	○
Testing phase	○	○	○	○	○
Creating the user manual and other user documentation	○	○	○	○	○
Maintenance	○	○	○	○	○

Слика 40: Питање – Када и колико често се користе случајеви коришћења

Одговори на питање када се све током трајања пројекта (фаза прикупљања корисничких захтјева, фаза анализе, фаза пројектовања, фаза имплементације, фаза тестирања, при креирању корисничког упутства и друге документације, током одржавања) и колико често користе случајеви коришћења, приказани су на следећем дијаграму (Слика 41):



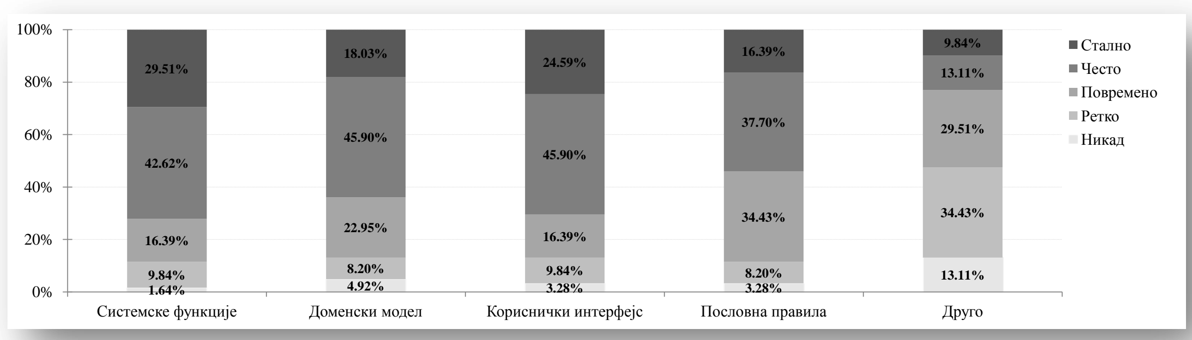
Слика 41: Када и колико често се користе случајеви коришћења

How often the use case specification enables you to discover:*

	Never	Rarely	From time to time	Often	Constantly
System functions					
Domain model					
User interface					
Business rules					
Something else					

Слика 42: Питање – Колико често се случајеви коришћења користе за откривање различитих елемената система

Одговоре на питање колико често случајеве коришћења користе за откривање различитих елемената система (системске функције, доменски модел, кориснички интерфејс, пословна правила, нешто друго), испитаници су дали на следећи начин (Слика 43):



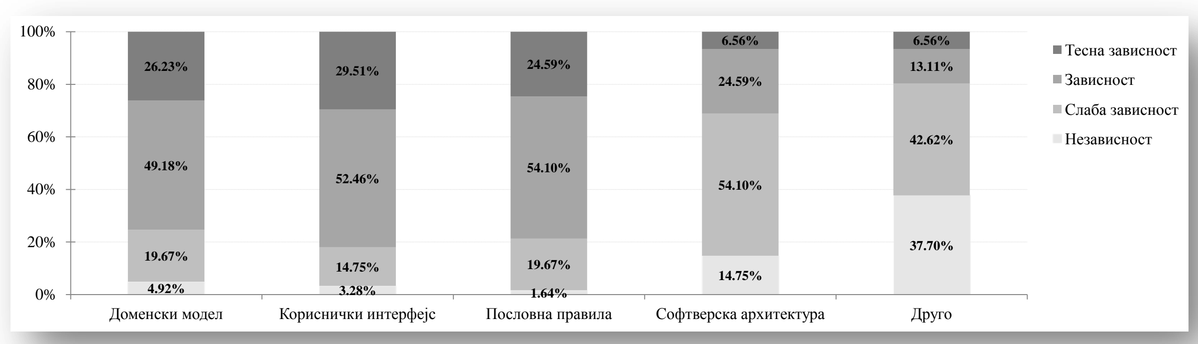
Слика 43: Колико често се случајеви коришћења користе за откривање различитих елемената система

Based on your experience, a use case specification should be closely related to: *

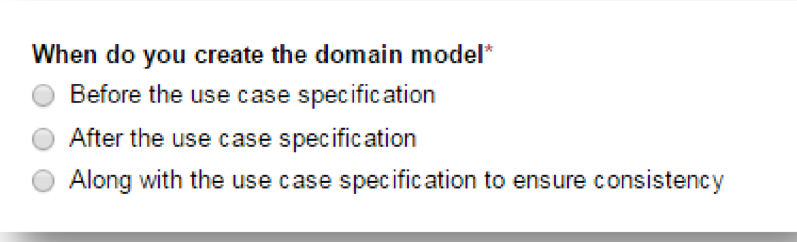
	Not related	Weakly related	Related	Closely related
Domain model	○	○	○	○
User interface	○	○	○	○
Business rules	○	○	○	○
Software architecture	○	○	○	○
Something else	○	○	○	○

Слика 44: Питање – Колико тијесно случајеви коришћења треба да буду повезани са различитим елементима система

На питање колико тијесно случајеви коришћења треба да буду повезани са различитим елементима система (доменски модел, кориснички интерфејс, пословна правила, софтверска архитектура, нешто друго), испитаници су одговорили на следећи начин (Слика 45):

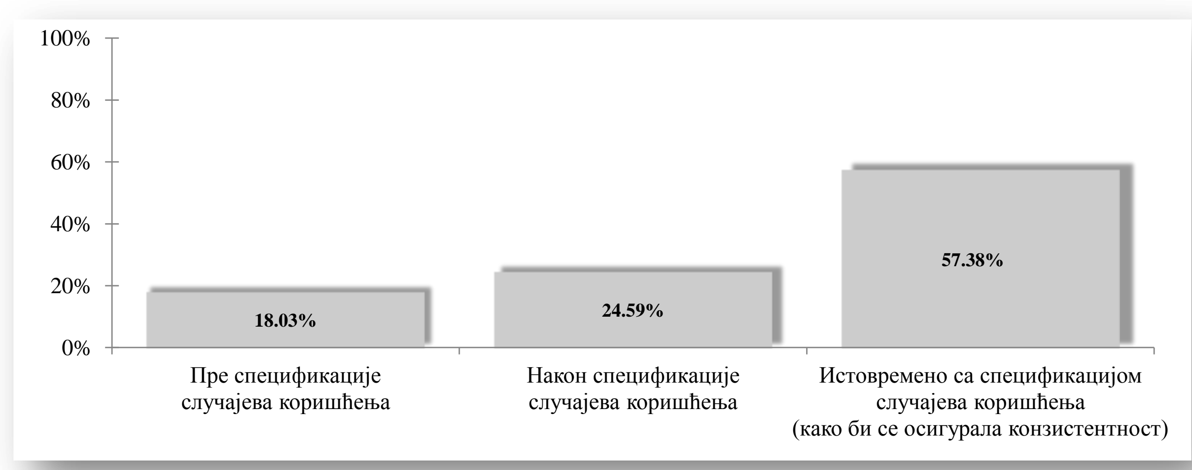


Слика 45: Колико тијесно случајеви коришћења треба да буду повезани са различитим елементима система



Слика 46: Питање – Када се креира доменски модел

На питање када креирају доменски модел (прије спецификације случајева коришћења, након спецификације случајева коришћења, истовремено са спецификацијом случајева коришћења), добијени су следећи резултати (Слика 47):



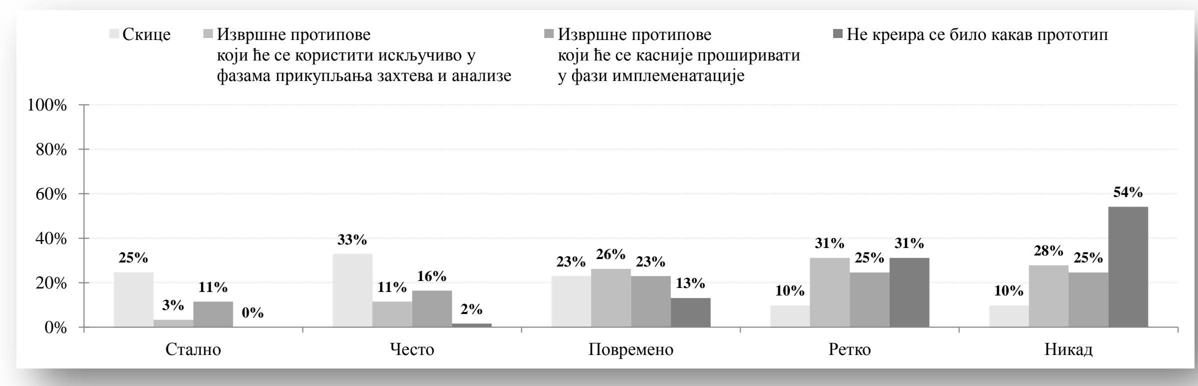
Слика 47: Када се креира доменски модел

How often do you create different kinds of user interface prototypes?*
If you create prototypes that can not be executed (drawings by hand or drawing tools), select first option. If the prototype can be executed (eg. created with technology that enables rapid solutions), but when it comes to implementation, you start implementing your final application from scratch, that means that you throw away the prototype, and you should select second option. If you create prototype that can be executed, and then continue to implement it to the final application, you shoould select third option.

	Never	Rarely	From time to time	Often	Constantly
Sketching	○	○	○	○	○
Executable throwaway prototype (not for use in implementation phase, but only for requirements and analysis)	○	○	○	○	○
Executable evolutive prototype (for use in implementation as a starting point, then refined and extended to the final application)	○	○	○	○	○
I do not create user interface prototypes	○	○	○	○	○

Слика 48: Питање – Колико често се креирају различите врсте прототипова корисничког интерфејса

На питање колико често испитаници креирају различите врсте прототипова корисничког интерфејса (скице корисничког интерфејса, извршиви прототипови који се бацају, извршиви еволутивни прототипови, не правим прототипове), испитаници су одговорили на следећи начин (Слика 49):



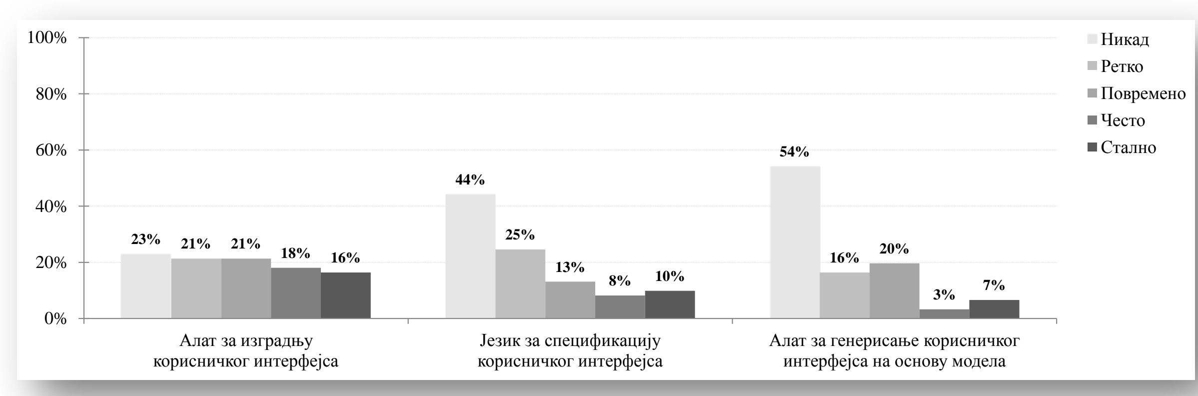
Слика 49: Колико често се креирају различите врсте прототипова корисничког интерфејса

How often do you implement the user interface by:*

	Never	Rarely	From time to time	Often	Constantly
User interface builder tool (often part of an IDE, with palette of UI widgets, and drawing area)	○	○	○	○	○
Language for user interface specification	○	○	○	○	○
Model based user interface generation tool	○	○	○	○	○

Слика 50: Питање – Учесталост различитих начина креирања корисничког интерфејса

Следеће питање односило се на то на који начин (алати за визуелно креирање корисничког интерфејса, језици за спецификацију корисничког интерфејса, алати за креирање корисничког интерфејса засновани на моделима) и колико често испитаници праве кориснички интерфејс. Одговори су приказани на следећем дијаграму (Слика 51):



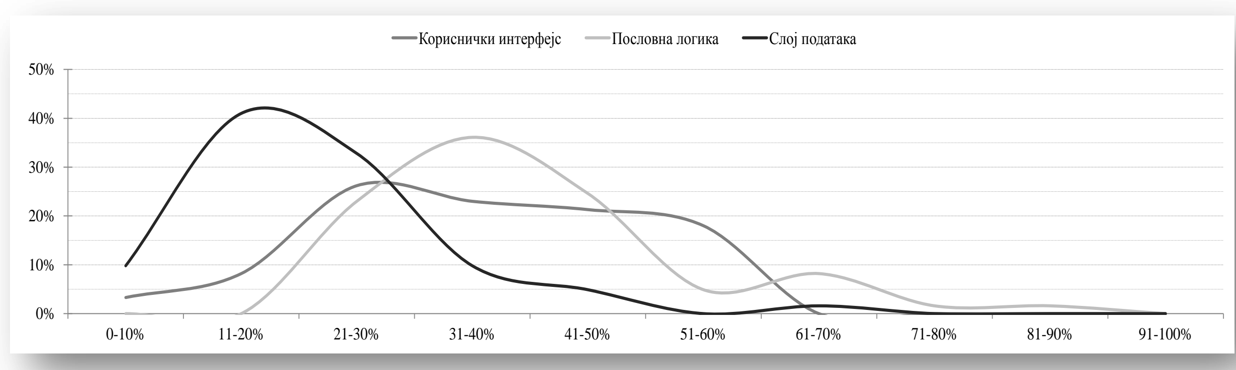
Слика 51: Учесталост различитих начина креирања корисничког интерфејса

Based on your working experience, what percent of the time and effort in a software project (all phases) for a three-tier business application goes to: *
(%)

	0-10	11-20	21-30	31-40	41-50	51-60	61-70	71-80	81-90	91-100
User interface	○	○	○	○	○	○	○	○	○	○
Business logic	○	○	○	○	○	○	○	○	○	○
Persistence	○	○	○	○	○	○	○	○	○	○

Слика 52: Питање – Колико времена и напора, на основу искуства испитаника, је потребно у софтверском пројекту издвојити на развој корисничког интерфејса, пословне логике и нивоа перзистенције

Одговоре питање колико времена и напора, на основу искуства испитаника, је потребно у софтверском пројекту издвојити на развој корисничког интерфејса, пословне логике и нивоа перзистенције, биће приказани графички (Слика 53):

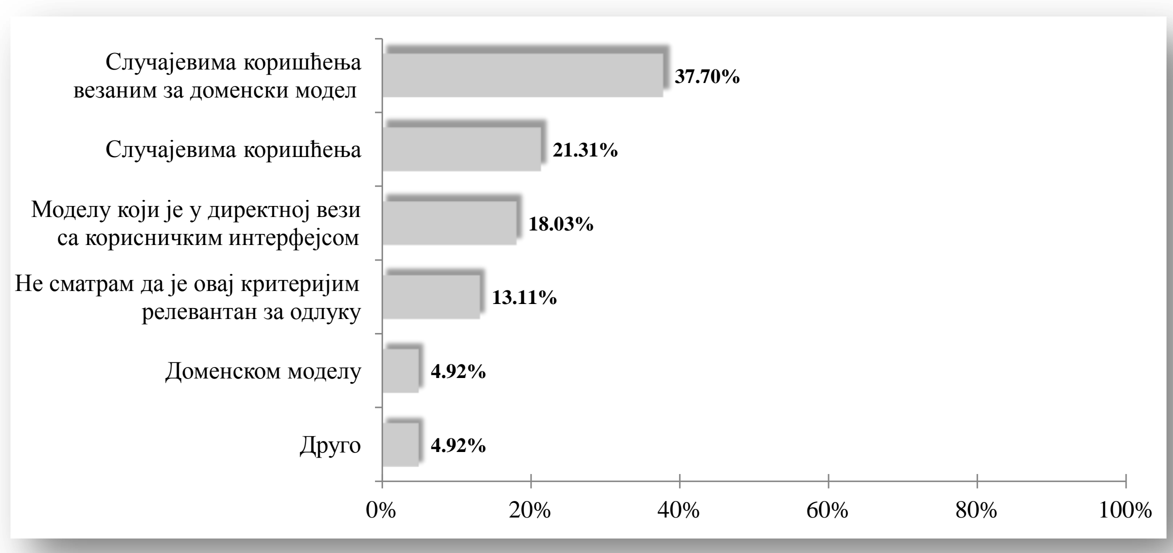


Слика 53: Колико времена и напора, на основу искуства испитаника, је потребно у софтверском пројекту издвојити на развој корисничког интерфејса, пословне логике и нивоа перзистенције

The input specification should be based on:*
The input specification is the set of input parametars needed by the tool to generate final user interface
- Use cases
- Domain model
- Use cases bounded to domain model
- A model directly related to user interface
- I don't consider this criteria relevant for descision
- Other:

Слика 54: Питање – На чему треба да буде заснована улазна спецификација коју користе алати за аутоматизацију развоја корисничког интерфејса

Када је ријеч о ставу испитаника према томе на чему треба да буде заснована улазна спецификација коју користе алати за аутоматизацију развоја корисничког интерфејса, резултати испитивања су следећи (Слика 55):



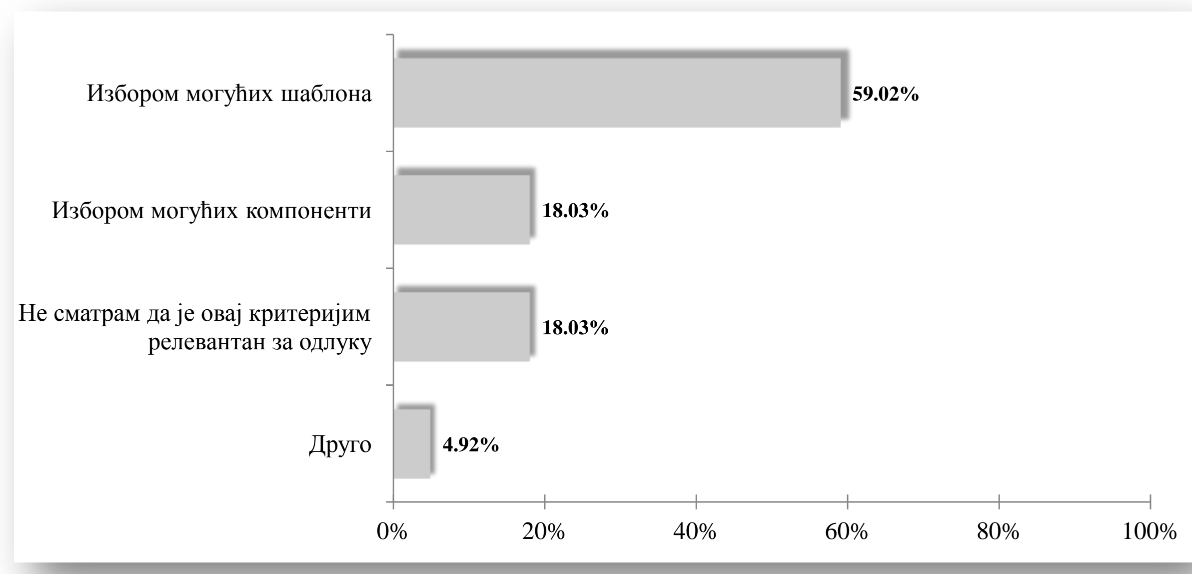
Слика 55: На чему треба да буде заснована улазна спецификација коју користе алати за аутоматизацију развоја корисничког интерфејса

The input specification should be extended with:*
The input specification often lacks some information related to the user interface. These extensions are optional, but can still be used to enrich the initial input specification so as to enable generating more apropriate user interfaces

- Choosing among different user interface templates created based on good practice in user interface development, defining set of widgets, layout, etc.
- Choosing a particular user interface components
- I don't consider this criteria relevant for descision
- Other:

Слика 56: Питање – Чиме основна улазна спецификација треба да буде проширена

На питање чиме основна улазна спецификација треба да буде проширена (избором између различитих шаблона корисничких интерфејса, избором између понуђених графичких компоненти, не сматрам овај критеријум значајним, остало), испитаници су одговорили на следећи начин (Слика 57):

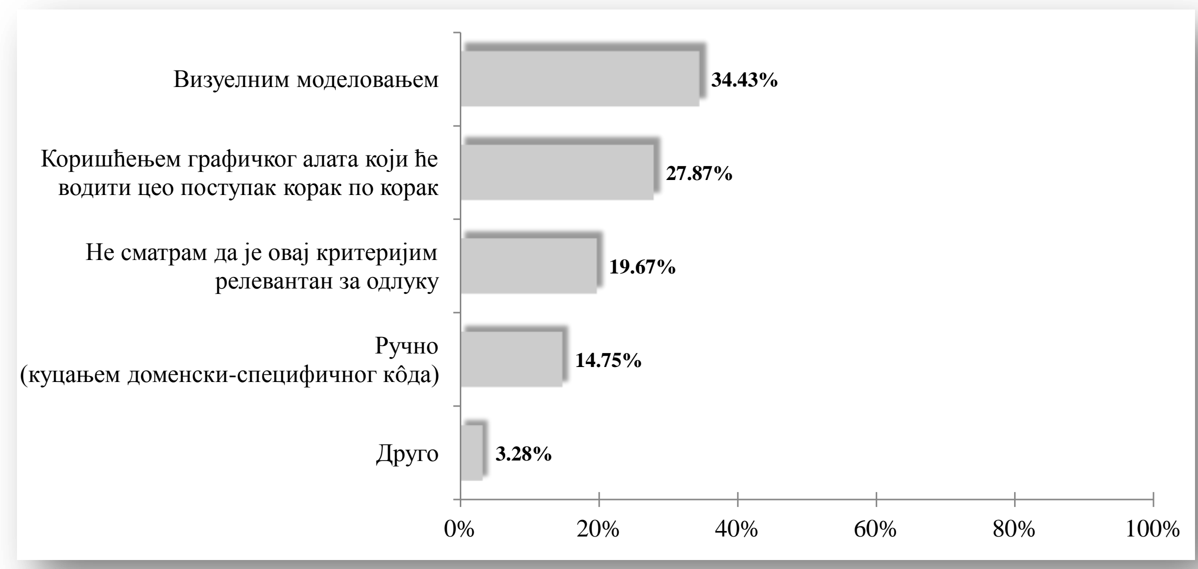


Слика 57: Чиме основна улазна спецификација треба да буде проширена

The input specification should be created:*
What kind of method for creating input specification would you prefer
- Manuely by typing in the domain specific language code
- By visual modelling of input specification
- Using wizard - a graphical tool that leeds you through the process of creating input specification
- I don't consider this criteria relevant for descision
- Other:

Слика 58: Питање – Пожељан начину креирања улазне спецификације

На питање о пожељном начину креирања улазне спецификације (визуелним моделовањем, коришћењем графичког алата који ће водити корисника корак по корак кроз поступак, ручно, не сматрам овај критеријум релевантним за одлуку), испитаници су се изјаснили на следећи начин (Слика 59):



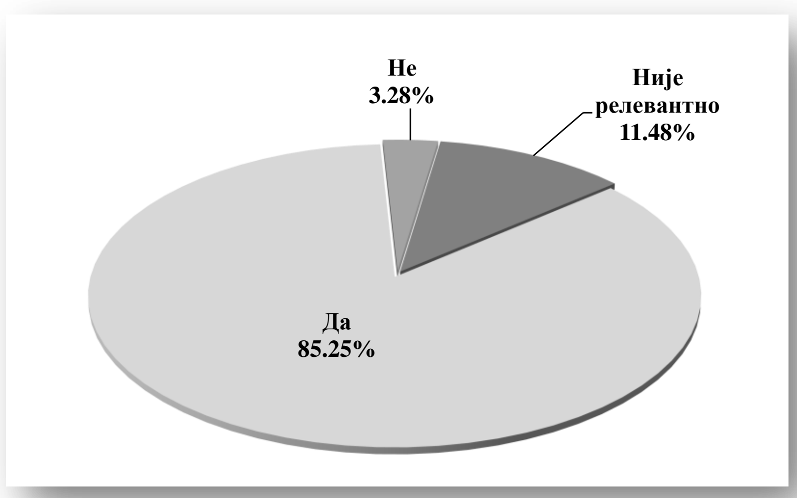
Слика 59: Пожељан начину креирања улазне спецификације

Would it be valuable to have the option to reuse other parts of the system e.g. database scheme, domain classes (if they exsist) for simplifying input specification*
The tool for input specification can simplify creating the input specification by analysing existing parts of the system, and using collected information.

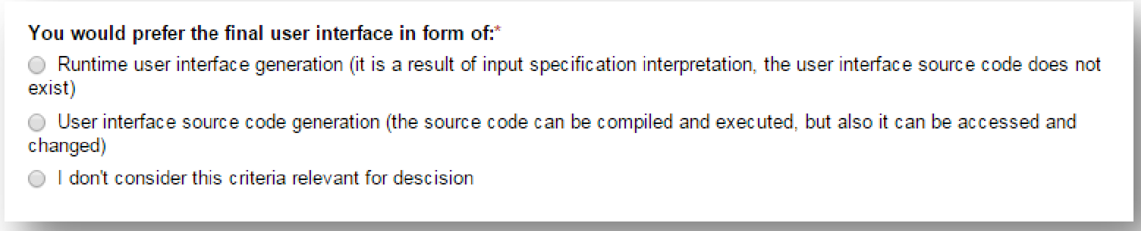
- Yes
- No
- I don't consider this criteria relevant for descision

Слика 60: Питање – Да ли би било корисно да постоји могућност поновног коришћења других дијелова система

На питање да ли би било корисно да постоји могућност поновног коришћења других дијелова система (нрп. шеме базе података или доменског модела уколико постоји) како би се поједноставио процес креирања улазне спецификација, одговори су приказани процентуално (Слика 61):



Слика 61: Да ли би било корисно да постоји могућност поновног коришћења других дијелова система


You would prefer the final user interface in form of:*

- Runtime user interface generation (it is a result of input specification interpretation, the user interface source code does not exist)
- User interface source code generation (the source code can be compiled and executed, but also it can be accessed and changed)
- I don't consider this criteria relevant for descision

Слика 62: Питање – У којој форми треба генерисати кориснички интерфејс

По питању форме у којој треба генерисати кориснички интерфејс (у вријеме извршења, генерисање програмског кода, није релевантно), резултати су следећи (Слика 63):

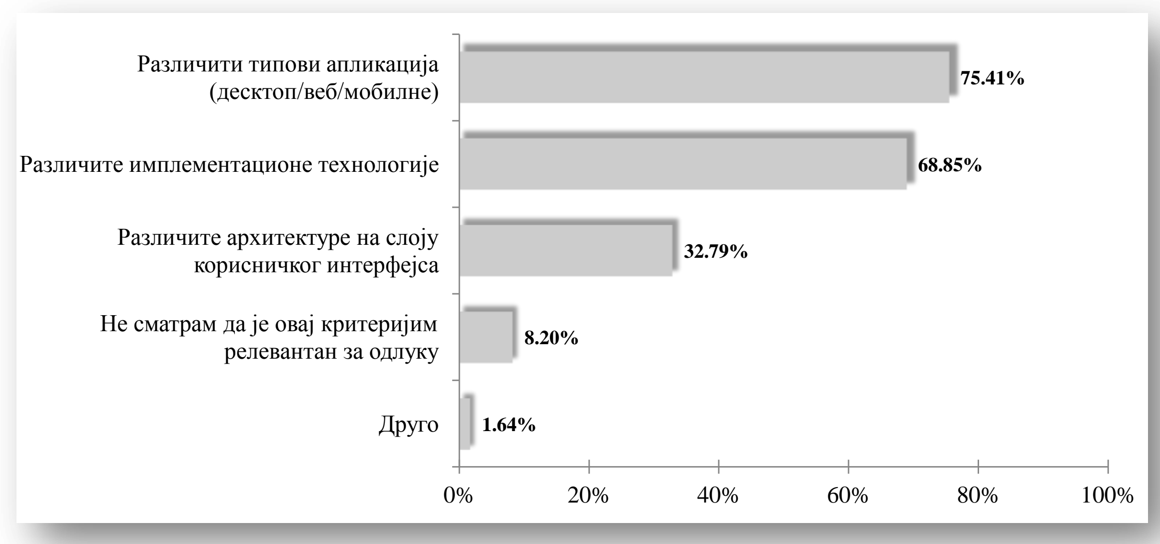
Слика 63: У којој форми треба генерисати кориснички интерфејс

The tool must enable generating user interface for:*
Select the TWO most important options in your opinion!

- Different implementation technologies
- Different application types (desktop/web/mobile)
- Different architectures at user interfaces layer
- I don't consider this criteria relevant for descision
- Other:

Слика 64: Питање – Значај различитих могућности које треба да обезбиједи алат за генерисање корисничког интерфејса

За питање шта сматрају значајним приликом генерисања корисничког интерфејса (тражено је да се одаберу два одговора која се сматрају најзначајнијим, а понуђени су одговори: различити типови апликација, различите имплементационе технологије, различите архитектуре на презентационом нивоу, није релевантно, нешто друго), резултати су следећи (Слика 65):



Слика 65: Значај различитих могућности које треба да обезбиједи алат за генерисање корисничког интерфејса

На основу анализе резултата истраживања, искристалисала се слика о бројним карактеристикама које будући алат за генерисање корисничког интерфејса треба да посједује. Ове карактеристике су груписане у седам категорија које ће у наставку бити коришћене као критеријуми за поређење постојећих алата. У наставку се налази листа идентификованих критеријума са пожељним карактеристикама по мишљењу испитаника:

- **Модели на којима је заснована улазна спецификација**
  - Улазна спецификација треба да буде заснована на случајевима коришћења, али на тај начин да јој се могу придружити информације које су везане за жељени кориснички интерфејс (избор шаблона корисничког интерфејса, избор графичких компонената);
- **Зависност улазне спецификације од модела података**
  - улазна спецификација мора да буде заснована на моделу података, али овај модел не треба посматрати одвојено од спецификације захтјева, већ их треба специфицирати паралелно, како би се осигурала њихова међусобна конзистентност;
- **Начин дефинисања улазне спецификације**
  - Потребно је омогућити коришћење сва три начина за генерисање корисничког интерфејса: визуелно моделовање, коришћење графичког алата који корисника води корак по корак кроз процес креирања спецификације, ручно писање кода доменски специфичног језика за креирање улазне спецификације;
- **Начин креирања корисничког интерфејса**
  - Алат треба да обезбиједи генерисање програмског кода корисничког интерфејса;
- **Измјењивост генерисаног програмског кода**

- Алат мора да омогући слободан приступ генерисаном програмском коду, и да пружи могућност његове измјене и надоградње. Како би ово било могуће, генерисани код треба да буде прегледан, а његова архитектура пројектована у складу са добром праксом прије свега у виду коришћења софтверских патерна.

- **Подржана могућност избора шаблона корисничког интерфејса**
  - Алат за генерисање корисничког интерфејса треба да омогући корисницима избор између различитих шаблона корисничког интерфејса. Зато је претходно потребно идентификовати скуп шаблона корисничког интерфејса који морају поштовати добру праксу и фундаменталне принципе у пројектовању корисничког интерфејса, која се првенствено односи на примјену патерна пројектовања корисничког интерфејса;
- **Подршка за различите типове апликација**
  - Алат за генерисање корисничког интерфејса треба да омогући корисницима генерисање корисничког интерфејса за различите типове апликација (десктоп апликације, веб апликације и мобилне апликације);

4.2. Преглед карактеристика постојећих алата за аутоматизацију развоја корисничког интерфејса

У овој фази истраживања прво ће бити објашњени критеријуми и разлози за одабир актуелних алата за генерисање корисничког интерфејса, затим ће бити направљен преглед њихових основних карактеристика и на крају ће бити испитано да ли и у којој мјери задовољавају дефинисане критеријуме. На крају ће бити представљени резултати упоредне анализе ових алата.

Већ је раније поменуто да се ни један алат од бројних алата за генерисање корисничког интерфејса није по броју корисника значајно издвојио од осталих па је посебан проблем био одабрати оне алате који ће бити анализирани. Критеријуме који су дефинисани били су да алати користе неки од познатих модела за које се може рећи да се користе у фази анализе корисничких захтјева, као и да пружају флексибилност у избору резултујуће врсте апликације и имплементационе технологије. Такође, фокус је био на алатима који се користе за генерисање корисничког интерфејса пословних апликација и омогућавају у потпуности, или дјелимично извршење *CRUD* операција над моделом података. На тај начин избор алата је сужен, а коначан скуп од шест алата који ће бити анализирани добијен је укључивањем још једног критеријума који се односи на број научних радова објављених у посљедњих пет година који анализирају неки до ових алата.[28]

У наставку ће бити дат преглед основних карактеристика шест одабраних алата: *ApacheIsis framework (Naked Objects)* [ApacheIsis], *Metawidget* [Metawidget],

[28] Препорука за овај додатни критеријум добијена је од рецензената и уредника научних часописа [IETSoftware] [SPE] који су својим коментарима помогли да се повећа објективност приликом одабира алата.

Webratio [WebRatio], *AlphaSimple* [AlphaSimple], *BizAgi BPM suite* [BizAgi] и Netbeans [NetBeansIDE] да би након тога били приказани резултати упоредне анализе по претходно дефинисаним критеријумима. Ради постизања што веће објективности, анализа је извршена на тај начин што је, поред прегледа научних радова који су за предмет имали ове алате, коришћен и метод студије случаја. Наиме, дефинисн је кориснички захтјев који није био превише обиман, али је осмишљен тако да обухвати најчешће ситуације које се могу срести у реалним пословним апликацијама. Ово се прије свега односи на богат доменски модел који обухвата најразличитије врсте типова атрибута и веза између објеката, као и случајеве коришћења који дефинишу жељена сценарија коришћења система. Затим је, за овакав кориснички захтјев имплементирно пет различитих апликација – коришћењем сваког од одабраних алата. На основу уочених карактеристика, биће представљене могућности примјене сваког од одабраних алата, а нарочито ће бити апострофиране разлике међу њима у односу на дефинисане критеријуме. Такође њихове заједничке карактеристике представљаће показатељ доминантних концепата савремених алата и приступа. У дисертацији неће бити приказана читава методологија и реализација истраживања, већ само најважније уочене карактеристике алата и резултати поређења.[29] Читаво истраживање доступно је у радовима [Buzejic11] [Cirovic11].

4.2.1. ApacheIsis framework (Naked Objects)

Apache Isis представља оквир који је дизајниран тако да омогућава брз развој апликација. Овај оквир слиједи Доменом вођени развој *(Doman Driven Design* у даљем

[29] Истраживање је спроведено у Лабораторији за софтверско инжењерство Факултета организационих наука током 2011. и 2012. године. Предмет истраживања привукао је пажњу студената студијског програма Софтверско инжењерство на Мастер академским студијама, па су се колеге дипл. инг. мастер Стеван Бузејић и дипл. инг. мастер Игор Ћировић активно укључили у реализацију истраживања, што је резултовало одбраном два Мастер рада [Buzejic11] [Cirovic11] који детаљно приказују методологију и резултате истраживања.

тексту DDD). Суштину овог приступа чини потпуна орјентисаност на доменски модел. Посредством домена се врши комуникација између свих чланова тима који учествују у развоју апликативног софтвера [Haywood09]. Заговорници DDD концепта сматрају да домен олакшава комуникацију између доменских експерата са једне стране и софтверских инжењера са друге стране. Уколико не би постојала ваљана комуникација ове двије групе приликом развоја неког софтвера, као резултат рада цијелог тима не би било могуће развити комплетно софтверско рјешење које задовољава критеријуме које је поставио крајњи корисник.

Оно што *Apache Isis* чини другачијим од осталих оквира је чињеница да он имплеменитра *NakedObjects* софтверски патерн. *NakedObjects* патерн је патерн архитектуре који омогућава аутоматско приказивање објектно орјентисаног корисничког интерфејса на основу структуре доменског модела [Haywood09]. Поред приказивања стања домена (простих атрибута, колекција и сл.), приказује се и њихово понашање. Преставља неку врсту пандана ORM (*Object-Relational Mapping* – алати за објектно-релационо пресликавање[30]) алатима. ORM алати врше пресликавање домена

[30] На основу функционалности које пружају, ORM алати могу бити сврстани у једну од четири категорије, тј. нивоа квалитета [Fussell97]:

- **Pure relational** (Потпуно релационо оријентисани алати) Цијела апликација, укључујући и кориснички интерфејс базирана је на релационом моделу и директно се ослања на SQL операције. У оваквим апликацијама се најчешће дио апликационе логике пребацује у базу података коришћењем ускладиштених процедура. Овај приступ има значајних недостатака, прије свега везаних за одржавање и портабилност, као и слабе могућности за поновно коришчење, али и поред тога може бити право рјешење за једноставне апликације.
- **Light object mapping** (Алати са ниским нивоом објектно-релационог пресликавања) Ентитети су представљени класама које се ручно пресликавају у табеле базе података. SQL и JDBC (Java DataBase Connectivity) код се такође пише ручно, али се од слоја пословне логике сакрива коришћењем софтверских патерна. И код овог приступа често се среће коришћење ускладиштених процедура. Употребљава се код апликација са мањим бројем ентитета.
- **Medium object mapping** (Алати са средњим нивоом објектно-релационог пресликавања) Апликација се пројектује на основу објектног модела. SQL се

ка перзистенционом слоју и обрнуто, док у случају *Apache Isis* оквира, пресликавање се врши од домена ка корисничком интерфејсу.

Претеча *Apache Isis* оквира представља *NakedObjects* оквир [NakedObjects], пројекат који се поделио на два одвојена пројекта:

- *Apache Isis* (који представља Јава верзију овог оквира) [ApacheIsis]
- *CodePlex* (.NET верзија)

Основна идеја *Apache Isis* оквира је да се у почетним фазама софтверског пројекта што пре формира доменски модел како би се доменским експертима представио кориснички интерфејс који је изграђен аутоматски, коришћењем функција овкира, а све у циљу њиховог оцјењивања структуре домена и каснијег унапређења. Овим би се избјегле грешке при дефинисању основне структуре система, чиме би се уштедјело вријеме евентуалног ревидирања програмског кода у касним фазама развоја. Апликација развијена коришћењем функција оквира је спремна за коришћење, али како је систем у потпуности заснован на доменском моделу, од

аутоматски генерише коришћењем код генератора или га генерише перзистентни оквир током извршења. Перзистентни оквир подржава асоцијације између објеката, а упити најчешће нису креирани коришћењем SQL-а, већ неког објектно-орјентисаног језика. Код овог приступа избјегава се коришћење ускладиштених процедура. Најчешће се користи у апликацијама средње величине. Већина постојећих ORM алата обезбјеђује овај ниво функционалности.

- **Full object mapping** (Алати са највишим нивоом објектно-релационог пресликавања) Овакви перзистентни оквири подржавају све могућности објектог модела: композицију, насљеђивање, полиморфизам, и тиме пружају могућност креирања богатијег доменског модела. Овакав приступ реализује тзв. транспарентну перзистентност, тј. класе из доменског модела не морају да насљеђују неку одређену основну класу нити да имплементирају одређени интерфејс чиме се постиже независност доменског модела од перзистентног оквира. Такође, механизми кеширања и пуњења података из базе у меморију, који могу у знатној мјери побољшати перформансе перзистентног слоја, потпуно су ”невидљиви” за апликацију. Постоје бројни комерцијални и *open source* (оквири засновани на концепту отвореног кода) перзистентни оквири који обезбјеђују овај ниво функционалности.

корисника се захтијева добро познавање пословног домена. Такође, апликација може бити коришћена и као прототип. Алат омогућава креирање *desktop* і *web* Јава апликација што посебно погодује крајњем кориснику али и инжењерима јер се за иста уложена средства добијају двије апликације засноване на истом доменском моделу.

Предности овог оквира првенствено се односе на брзину развоја и јасној могућности валидације доменског модела. Када крајњи корисник има интеракцију са системом посредством корисничког интерфејса, он заправо подацима које уноси, мења или брише вредности над објектима доменског модела. Карактеристика добијеног корисничког интерфејса је да свака графичка форма омогућава интеракцију са једним доменским објектом, односно омогућава промјену стања објекта. Захваљујући могућностима брзог развоја апликације, могуће је на основу интеракције корисника са системом и његовог запажања прикупити сугестије и прилагодити изглед и функционалност апликације жељама корисника.

Једно од значајних ограничења јесте то што се све функционалности апликације налазе у оквиру доменских објеката па самим тим и слој апликационе логике. Постоје бројни противници оваквог става (потпуно учаурење понашања) чиме се додаје посебна примједба на то да домен више не представља такозване *POJO* (Plain Old Java Object) објекте. Противници Naked Objects патерна као главну ману наводе успостављање зависности између домена и конкретног оквира [DDDNakedObjects]. Домену се додјељује превелика одговорност, иако би она требала да буде у неком међуслоју који би се налазио између корисничког интерфејса са једне и доменског модела са друге стране.

Први и основни корак у развоју апликације коришћењем Apache Isis оквира је креирање објектно орјентисаног доменског модела. Сам поступак креирања доменског модела је тривијалан. Потребно је направити одговарајуће класе које ће репрезентовати ентитете, а њима додати одговарајуће атрибуте.

Програмски модел оквира чини скуп анотација и програмских конвенција које оквир препознаје. *Apache Isis* подржава велики број анотација, од којих свака обезбјеђује одређену функционалност.

Најбржи начин за израду комплетне апликације јесте такозвано прототипизирање – поступак у коме ће се користити динамичке структуре као вид перзистенције података [Haywood09].

Apache Isis омогућава да се доменски модел у презентационом слоју представи на више различитих начина, а у зависности од потреба и захтјева корисника. Аутоматски изгенерисана апликација нуди следеће презентационе технологије и алате:

- **DnD** (Drag-n-Drop) – обезбјеђује презентацију софтверског система као *desktop* апликације са којом се врши интеракција *drag-n-drop* принципом.
- **HTML** – основна web апликација.
- **Scimpi** – омогућава отпремање апликације у виду web пројекта [Scimpy/1] [Scimpy/2].
- **WicketViewer** – користи Apache Wicket за приказ генеричког изгледа доменских објеката у web апликацији [Wicket].
- **RESTful *browser*** – приказују доменске објекте кроз RESTful интерфејс [RestfulObjects].

Заједничку карактеристику свих наведених алата за преглед представља могућност приступа доменском моделу. Доменски објекти писани коришћењем *Apache Isis* оквира морају да наслиједе *org.apache.isis.applib.AbstractDomainObjectklasu* која реализује *org.apache.isis.applib.DomainObjectContainer* интерфејс. Након тога, врши се дефиниција атрибута класе. Сви атрибути би требали да буду приватни уз одговарајуће *get()* и *set()* методе.

Када се програм покрене помоћу DnD алата за преглед, додавање референцираног објекта се врши једноставним превлачењем. Међутим, уколико се апликација изврши на *web* серверу, њено покретање у *web* browser-у не даје могућност повезивања објеката превлачењем. Прво је потребно "открити" објекат који се жели повезати како би он био доступан оквиру. Након тога, прелази се у фазу повезивања.

Поред везе један према један, у којој једна класа прихвата објекат друге као референцу, постоји могућност дефинисања везе један према више у коме би једна класа садржала листу објеката друге класе. Да би се реализовао овакав вид везе потребно је додати одређен број метода у одговарајуће класе доменских ентитета. Уколико је над колекцијом дефинисана *addLIST()* и *removeLIST()* метода, *Apache Isis* ће позивати ове методе аутоматски при додавању односно избацивању елемента из колекције, гдје *LIST* представља назив колекције у конкретној класи.

Након дефинисања ентитета и успостављања међусобних веза, следећи корак у развоју апликације коришћењем *Apache Isis* оквира представља дефинисање операција које ће се извршавати над доменским ентитетима, односно дефинисање акција. Да би нека метода била видљива на корисничком интерфејсу и самим тим изазивала неки догађај, довољно је да она буде декларисана са *public* модификатором приступа. Акције могу да буду дефинисане у оквиру доменских ентитета или чак у оквиру такозваних репозиторијума. Репозиторијум представља механизам за добијање референце постојећег објекта. Поред репозиторијума уводи се и појам *factory*-ја који дефинише механизам учаурења (енкапсулације) креирања објеката. Дакле, следећи корак у дефинисању и развоју апликације представљао би дефинисање репозиторијума. Акције се поред репозиторијума могу додавати у оквиру доменских ентитета. Најчешће, акције које се додају у оквиру домена представљају акције које омогућавају креирање објекта референцираног ентитета.

При креирању корисничког интерфејса не нуди се могућност избора одређеног темплејта по ком ће страница бити организована. Подразумијевани изглед се поставља када су у питању и DnD и HTML начин приказа, при чему се кориснику оставља могућност измјене *css*-а кад је у питању HTML. Програмском коду корисничког интерфејса није могуће приступити, прије свега зато што програмски код корисничког интерфејса и не постоји. Цио код је директно, као што је већ поменуто, везан за доменски модел. Креирање компоненти корисничког интерфејса *Apache Isis* aplikacije vrši se u вријеме извршења. Измјена домена доводи до измјене корисничког интерфејса, тако да уколико се жели утицати на структуру и понашање корисничког интерфејса то се мора урадити посредством домена или репозиторијума. Корисничке улоге је могуће дефинисати и одредити обичног корисника и администратора, а самим тим и делегирати одговорности сходно њиховим улогама.

Пословну логику система контролише програмер. Као што је речено, она се налази у склопу доменског модела и репозиторијумима. Уколико корисник жели да измијени начин на који су повезане класе, или да уведе неку нову методу која ће бити видљива на корисничком интерфејсу као компонента интерфејса (нпр. дугме), то мора да уради директно над доменом односно над репозиторијумима. Међутим значајније промјене над архитектуром система нису могуће јер оквир захтијева имплементацију пословне логуке директно у оквиру доменског модела.

Са становишта функционалности које оквир обезбјеђује, а које се односе на *CRUD* операције, могло би се рећи да оквир обезбјеђује основу за извршење ових операција (позив са корисничког интерфејса коришћењем дугмета и догађаја), али је програмер дужан да имплементира читаву операцију уколико се подаци чувају у релационом систему за управљање базама података.

У примјеру софтверског пројекта из студије случаја нашег истраживања, израда апликације коришћењем *Apache Isis* оквира захтијевала је ручно писање око 650 линија програмског кода.

4.2.2. Metawidget

Metawidget [Metawidget] представља објектно орјентисан алат који се користи у изради корисничког интерфејса (Object/User Interface mapping – OIM – пресликавање између објеката и корисничког интерфејса). Заснован је на принципу провјеравања карактеристика објеката у тренутку извршавања програма и коришћења резултата провјере у изради компоненти корисничког интерфејса.

Цио процес *Metawidget* обавља без увођења нових технологија. Први корак у аутоматизацији израде корисничког интерфејса представља испитивање постојеће такозване *back-end* односно позадинске технологије постојеће апликације (као што су *Java Bean*-ovi, *XML* конфигурациони фајлови, анотације и сл.), након чега слиједи креирање основних компоненти корисничког интерфејса на већ постојећим технолошким рјешењима (као што је *Swing, JSF, Struts, Android* и сл.).

Једна од основних идеја *Metawidget* оквира је да се анализом постојеће *front-end* и *back-end* технологије, односно технологија којима је имплементиран постојећи кориснички интерфејс и технологија које су коришћене на нивоу пословне логике и перзистенционом нивоу система, добије кориснички интерфејс који се уклапа у постојећу архитектуру софтверског система.

Творац *Metawidget* оквира је др Richard Kennard[31], а сам оквир се развија у оквиру Kennard Consulting предузећа [KennardConsulting].

Metawidget оквир се састоји од пет нивоа архитектуре (Слика 66). Први ниво чини слој Инспектора (Inspector). Инспектори су задужени за провјеру, тј. анализу постојеће *back-end* архитектуре система. Инспектор може бити сложен: сачињен од једног конкретног инспектора или листе више различитих.

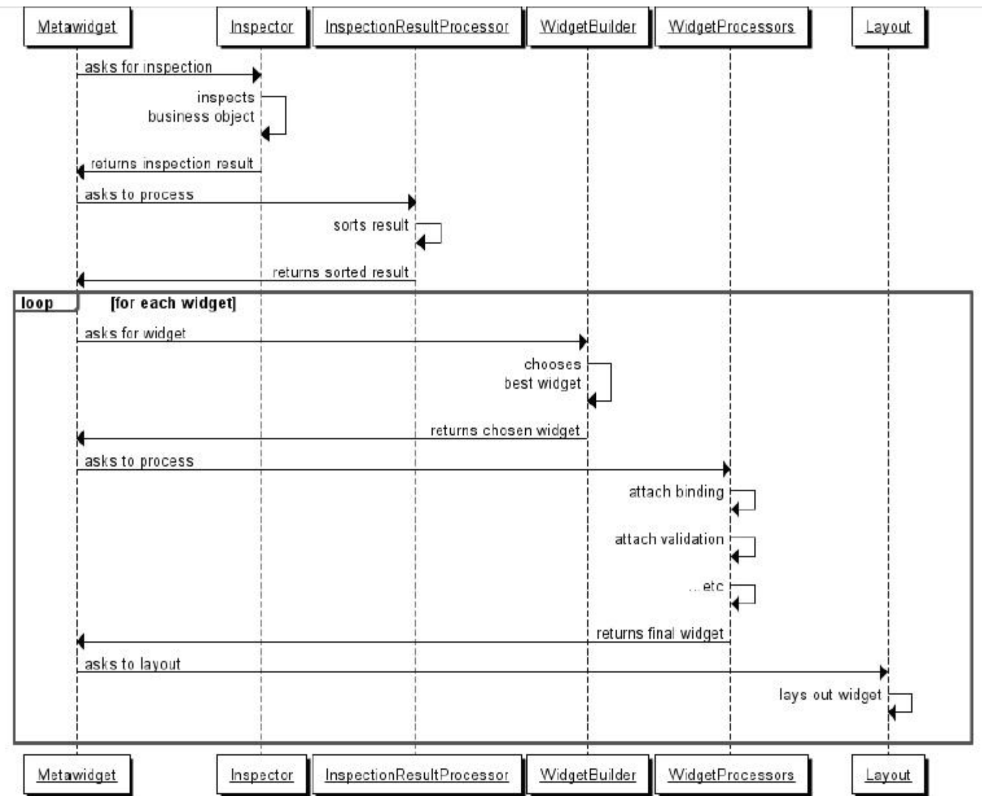
Други ниво оквира чини листа *InspectionResultProcessors* која је задужена за обраду и измјену резултата инспекције.

Следећи ниво чине *WidgetBuilder*-i који су задужени за креирање *widget*-a односно компоненти корисничког интефејса за конкретну *front-end* технологију.

WidgetProcessors представљају листу *WidgetProcessor*-a која је задужена за обраду и мијењање конкретне компоненте корисничког интерфејса.

Layout-и представљају посљедњи ниво који приказује компоненте на екрану, организује их у колоне или их међусобно групише.

[31] Осјећам обавезу да поменем изузетну сарадњу коју сам током истраживања остварио са др Кенардом током које смо размјењивали искуства и на конструктиван начин допринијели развоју оба приступа (Metawidget и *SilabUI* приступа) иако су се у неким тачкама наша мишљења значајно разликовала, прије свега у погледу начина креирања интерфејса, тј. да ли треба интерфејс да буде генерисан током извршења програма (*runtime user interface development* – не генерише се програмски код корисничког интерфејса, већ се интерфејс добија интерпретаијом улазне спецификације током извршења програма) што је теза коју заступа др. Кенард, или је боље генерисати програмски код корисничког интерфејса, што је једна од карактеристика мог приступа.



Слика 66: Петослојна архитектура Metawidget оквира [Kennard11]

Основу апликације која користи *Metawidget* оквир представља доменски модел. Доменски модел се креира ручно или коришћењем неког од перзистентних механизама. Може бити потпуно изолован од утицаја *Metawidget* оквира, а класе доменског модела могу представљати *POJO* класе. На основу резултата анализе *back-end* архитектуре система врши се креирање корисничког интерфејса, односно *widget*-а за конкретне атрибуте домена, који се уклапају у постојећу *front-end* архитектуру.

Програмски код интерфејса у потпуности је под контролом програмера. Одговорност програмера је да креира основу интерфејса а затим ће сам *Metawidget* да се постара око креирања компоненти интерфејса.

Оно што аутори посебно истичу као предност оквира, јесте креирање свих *widget*-а тј. графичких компоненти у вријеме извршења програма. Колико креирање

компоненти у време извршења представља предност толико представља и недостатак. Недостатак је прије свега због тога што програмском коду креираног *widget*-а није могуће директно приступити и тиме модификовати начин представљања конкретног атрибута. Али као што је помињано, програмеру се оставља могућност редефиниције начина приказа.

Аутоматско дефинисање корисничких улога, у смислу одређивања администратора и обичног корисника, а самим тим и делегирање одговорности у зависности од улоге, није подржано оквиром и не постоји механизам оквира којим је могуће одредити улоге корисника на нивоу апликације.

Пословну логику оквир не генерише. Све што је специфично за апликацију, па самим тим и пословну логику, дефинише програмер. Такође, на програмеру је да изабере архитектуру апликације и да је реализује на начин на који он мисли да је најбољи.

Не постоји могућност аутоматског дефинисања *CRUD* операција, прије свега из разлога непостојања перзистентног механизма у оквиру.

Као недостатак у коришћењу оквира могла би се навести чињеница претјераног оптерећивања корисничког интерфејса логиком апликације. Наиме, уколико се жели нека акција извести то није могуће без додавања логике на кориснички интерфејс. У апликацији студијског примера, коришћена је *@UIAction* анотација, која креира дугме на корисничком интерфејсу уз одговарајућу подршку *EventHandler*-а и аутоматизацијом генерисања догађаја чиме је успостављена веза између оквира и апликације, али на презентационом нивоу.

Први корак у реализацији апликације која је заснована на *Metawidget* оквиру представља дефинисање доменског модела. Доменски модел у суштини могу чинити

POJO класе, чиме се избјегава диреткно повезивање домена и оквира. Међутим, над атрибутима домена могуће је постављати анотације, као што је нпр. *@UISection* анотација која означава да на корисничком интерфејсу за задати атрибут *Metawidget* треба да креира посебну секцију. Тиме се успоставља зависност између пословних објеката домена и самог оквира. Међутим, оквир пружа могућност издвајања ових зависности у посебан XML документ, који се по програмерским стандардима *Metawidget* заједнице назива *metawidget.xml*, чиме је омогућено коришћење доменског модела од стране других апликација.

Након дефинисања доменског модела, следећи корак у реализацији апликације јесте креирање корисничког интерфејса. Процесу креирања корисничког интерфејса претходи инспекција доменског модела. Након извршене инспекције, оквир препознаје атрибуте домена као и њихов тип. Међутим кориснички интерфејс се неће аутоматски креирати. Ово је једна од суштинских карактеристика *Metawidget* оквира. Резултати инспекције биће примијењени на постојеће *front-end* рјешење, на постојећу структуру интерфејса апликације. Кориснику се препушта одговорност реализовања основног корисничког интерфејса и његовог повезивања са *Metawidget* оквиром, па самим тим и резултатима спроведене иснпекције над доменским моделом. Након дефинисања основе, врши се подешавање параметара инспекције за конкретан доменски објекат чија се форма жели приказати.

У конфигурационој *metawidget.xml* датотеци наводи се путања ка другим датотекама без којих није могуће извршити инспекцију домена, а самим тим и креирати апликацију. У датотеци *metawidget-metadata.xml* наведени су ентитети на основу којих треба градити компоненте интерфејса.

Након спецификације *metawidget.xml* датотеке потребно је додати контејнеру вишег нивоа *Metawidget* који је претходно дефинисан. У студијском примјеру коришћен је *SwingMetawidget.*

Metawidget сваком атрибуту испитиване класе, на основу типа атрибута, додјељује одређену компоненту. Тако на примјер параметри типа *int* реализују се коришћењем *JSpinner* компоненте, параметри типа *String* коришћењем *JTextField*, енумерације коришћењем *JComboBox* и слично, кад је у питању *Swing* апликација. Софтверском инжењеру, који представља корисника овог оквира, остављена је могућност дефиниције сопственог начина приказа конкретног атрибута, тако да програмер може изабрати компоненту којом ће приказати изабрани атрибут.

Да би се додало дугме на екранску форму и тиме омогућила интеракција корисника са системом, потребно је у класи, која је одговорна за приказ конкретног доменског објекта на форми, дефинисати методу која ће изнад свог потписа имати *@UIAction* анотацију. Такве методе се називају акцијске методе. Као једна од предности коришћења *Metawidget* оквира би се могла навести управо способност веома једноставног начина дефинисања акције.

Као један од недостатака *SwingMetawidget*-а, могло би се навести непостојање аутоматизације у генерисању механизма за повезивање форме са објектима доменског модела на основу којих се врши генерисање форми корисничког интерфејса.

Коришћење *Metawidget*-а при дефинисању корисничког интерфејса јесте убрзало сам процес развоја презентационог слоја, али не у мјери у којој се то очекивало. У примјеру софтверског пројекта из студије случаја нашег истраживања, израда апликације коришћењем *Metawidget* оквира захтијевала је ручно писање око 2000 линија програмског кода. Општи утисак је да се *Metawidget* може користити у

апликативном развоју али искључиво као нека врста помоћног алата, а не као оквир који би доминирао процесом израде апликације.

4.2.3. WebRatio

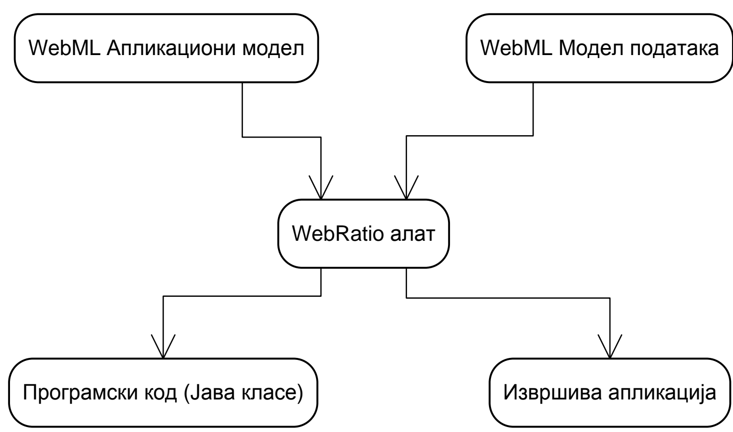
WebRatio је алат за заснован на идеји моделом вођеног развоја пословних апликација прилагођених потребама корисника. Алат је фокусиран прије свега на израду апликација са *Web/SOA* (*Service Oriented Architecture*) архитектуром. Овај алат омогућава представљање корисничких захтјева помоћу апстрактних модела и аутоматско генерисање потпуне пословне апликације. Модели су засновани на *BPMN (Business Process Model and Notation)* стандарду и *WebML (Web Modeling Language)* језику за моделовање. Као резултат се добија стандардна *Јава* веб апликација која се може покренути на било ком *Јава* апликационом серверу. Алат користи развојно окружење отвореног типа – правила генерисања се могу проширивати и прилагођавати по потреби [WebRatio].

WebML представља формалну графичку спецификацију која се користи у процесу пројектовања web апликација. *WebML* има подршку у алатима који олакшавају процес пројектовања. Главни циљеви *WebML* спецификације су: [Ceri00]

- Дефинисање структуре web апликације на високом нивоу апстракције.
- Обезбјеђивање више различитих погледа за исти садржај.
- Одвајање садржајних информација од структуре странице, линкова, навигације и презентације.
- Чување мета-података прикупљених у процесу пројектовања у оквиру репозиторијума, које се могу користити и позивати током животног циклуса апликације при динамичком генерисању web страна.

- Експлицитно моделовање корисника и заједница у репозиторијумима.
- Спецификација операција које се користе у управљању подацима и ажурирању садржаја странице.

WebRatio је алат који користи *Eclipse* развојно окружење за израду пројеката. Израда пројекта у *WebRatio* алату врши се израдом *WebML* модела података, који представља доменски модел. Затим се врши израда модела апликације, или модела процеса. У процесу израде студијског примјера коришћен је модел апликације представљен *WebML* нотацијом. На основу описаних модела добија се цјелокупан програмски код (*Јава* класе) и извршива web апликација. Описани процес приказан је на слици:



Слика 67: Процес израде апликације у *WebRatio* алату

На основу дефинисаног модела података креира се база података, која ће имати идентичну структуру као и сам модел, уз све енитете и њихове међусобне везе. Алат аутоматски генерише *Јава* класе за креиране ентитете. Међутим, генерисаним доменским класама није могуће приступити и мијењати их из алата.

Када се заврши са поступком дефинисања модела података (доменског модела) и креирања базе података на основу тог модела, може се прећи на развој апликације помоћу изабране нотације. Треба напоменути да свака измјена структуре модела

након генерисања базе података захтијева додатну синхронизацију базе са моделом, што у ствари значи понављање поступка креирања базе података.

Правила генерисања која *WebRatio* користи за израду коначне web апликације се могу у потпуности прилагођавати и проширивати. На тај начин се развојно окружење прилагођава потребама, стандардима и навикама различитих програмера и програмерских тимова.

Конкретно, развојно окружење се може проширити на следеће начине:

- дефинисањем распоредних шаблона
- додавањем нових компоненти модела.

Визуелни идентитет web апликације се може прецизно подесити. Ово је могуће захваљујући специфичним правилима генерисања, која може дефинисати графички дизајнер и која ће *WebRatio* користити да би изгенерисао распоредне (шаблонске) стране веб апликације.

Презентациона правила се дефинишу у презентационом стилу – посебној врсти *WebRatio* пројекта – и могу се написати у било ком описном језику, најчешће HTML.

Презентациони стил и апликациони модел су потпуно независни један од другог. Према томе, могуће је на основу истог апликацијског модела генерисати апликацију са потпуно другачијим визуелним идентитетом. Такође је могуће користити један презентациони стил за давање истог визуелног идентитета више различитих web апликација.

Пословна логика апликације се дефинише компонентама које чине апликациони модел и начином на који су те компоненте међусобно повезане. У

стандардном развојном окружењу доступан је предефинисан скуп компоненти, али корисник може такође дефинисати и интегрисати нове компоненте по жељи.

WebRatio обезбјеђује формирање улазне спецификације коришћењем графичког алата – *wizard*-а који значајно убрзава поступак развоја апликације. Цјелокупан програмски код апликације се аутоматски генерише, међутим програмском коду није могуће директно приступити и мијењати га. Изузетак су кориснички дефинисане јединице које се је могуће креирати у одређеним ситуацијама.

Презентациони ниво се дефинише на тај начин што је страница замишљена као табела чије ћелије програмер може испунити жељеним садржајем (обично графичким компонентама које су повезане са атрибутима одређеног доменског објекта). На тај начин програмеру је омогућена флексибилност при распоређивању компоненти корисничког интерфејса, иако се овдје не може говорити предефинисаним шаблонима – темплејтима корисничког интерфејса који се аутоматски могу примијенити. Ипак, ова могућност представља значајан искорак у односу на остале анализиране приступе.

Програм који се добија као резултат коришћења *WebRatio* алата представља web апликацију написану у *Јава* програмском језику. Презентациони слој је реализован помоћу *JSP (Java Server Pages)* страна, а перзистентни слој *Hibernate*[32] оквиром [Acerbis07].

---

[32] Hibernate представља комплетно рјешење за објектно релационо пресликавање, а поред тога омогућава и управљање перзистентношћу. Hibernate је оквир (framework) који обезбјеђује подршку за пресликавање свих типова веза из објектног модела, као и напреднијих концепата објектно орјентисаног приступа као што су асоцијације, насљеђивање, полиморфизам, као и рад са колекцијама објеката. Осим тога, садржи и механизме који обезбјеђују *connection pooling*, управљање трансакцијама, различите механизме кеширања података, моћан објектни упитни језик, са подршком за SQL, *lazy*

4.2.4. AlphaSimple

AlphaSimple је web алат који омогућава кориснику да, на основу креираног модела апликације, пише тестове, генерише прототип апликације и генерише код за циљану платформу (*model-centric tool*). Помоћу доменског модела представљају се подаци са којима ће апликација радити. Извршиви прототипови[33] засновани на доменском моделу омогућавају учесницима да у раним фазама пројекта врше валидацију начина на који ће захтјеви бити реализовани у апликацији коју треба израдити.

AlphaSimple је алат за израду детаљних модела чијом се интерпретацијом добија извршива апликација. За опис модела *AlphaSimple* користи *TextUML* нотацију. Аутори алата су одабрали текстуалну нотацију, јер сматрају да се њоме боље описују и приказују потребни детаљи, док је њихово мишљење да је графичка нотација погоднија за општи преглед система [AlphaSimpleDoc].

TextUML је текстуална нотација за *UML* (*Unified Modeling Language*) језик. Много популарнија је графичка нотација за *UML*, али сама семантика *UML* језика, дакле начин описивања модела и његових елемената, је дефинисана потпуно засебно од графичке нотације. Према томе, могуће су алтернативне *UML* нотације, укључујући

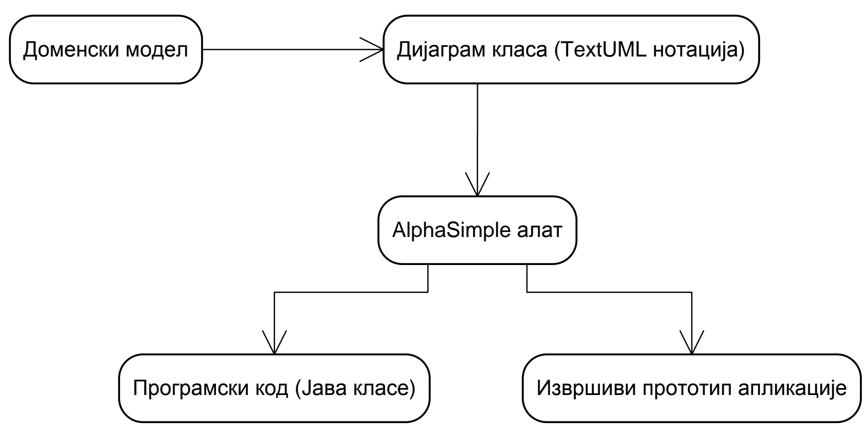
---

loading итд. Уз све ове особине, Hibernate не намеће посебна правила које доменски модел треба да задовољи, нити имплементацију одређених интерфејса, чиме је омогућено да доменски модел апликације садржи *POJO* објекте и да буде потпуно независан од перзистенционог слоја апликације. Hibernate се може користити како у десктоп (swing) апликацијама, тако и у JEE (Java Enterprise Edition) апликацијама коришћењем јава servlet или EJB session beans компоненти.

[33] Под извршивим прототипом подразумијева се рана верзија апликације, која имплементира функционалности идентификоване из прикупљених корисничких захтјева. Његова сврха је утврђивање и даље прецизирање корисничких захтјева.

и текстуалне попут *TextUML*-а. Сви концепти и механизми изложени у *TextUML*-у су дефинисани *UML* спецификацијом. [TextUML]

У *AlphaSimple* алату израда апликације врши се описом доменског модела над којим ће та апликација радити. Доменски модел представљен је дијаграмом класа. На основу описаног доменског модела добијају се одговарајуће *Јава* класе и извршиви прототип апликације. Описани процес израде пројекта приказан је на слици:



Слика 68: Процес израде апликације у AlphaSimple алату

Генерисање кода је једна од кључних могућности *AlphaSimple* алата. *AlphaSimple* користи *StringTemplate 3*, једноставан механизам за генерисање кода [StringTemplate]. Познавање начина писања шаблона у *StringTemplate*-у *3* омогућава креирање сопствених шаблона за *AlphaSimple*. *StringTemplate* језик се не разликује много од других језика за шаблоне и једноставан је за учење. Поред ове могућности, постоје и готови шаблони за *AlphaSimple* алат, који су доступни на његовом

репозиторијуму [AlphaSimpleDoc]. *AlphaSimple* тренутно подржава Јава платформу, прецизније генерисање *POJO* и *JPA*[34] *DAO* [35] класа.

Друга кључна особина *AlphaSimple* алата је обезбјеђивање извршивог прототипа апликације. Довољно је описати доменски модел са својствима и везама да би се добио потпуно функционалан прототип који обезбјеђује основне *CRUD* операције за сваки ентитет.

У зависности од природе веза између ентитета (нпр. један према више) постоји могућност да подразумијеване операције додавања, измјене, или брисања не одговарају појединим ентитетима. У оваквим случајевима, као и онда када је потребна нека додатна операција, могуће је дефинисати сопствене операције и користити их по

---

[34] Java Persistence API представља стандард за објектно-релационо пресликавање и управљање перзистенцијом за JEE (Java Enterprise Edition) платформу. Развијен је од стране Sun Microsystems компаније из два основна разлога:

1. Поједностављивање израде EE и SE апликација коришћењем једноставнијег перзистенционог механизма.
2. Окупљање цјелокупне Јава заједнице око јединственог – стандардног перзистенционог API-a.

JPA је заснован на доброј пракси преузетој из постојећих пројеката (прије свега на Hibernate, али и остале као на пр. JDO, TopLink...), и укључен је у састав EJB 3.0, као замјена за Entity Bean концепт из претходних верзија EJB спецификације. Основна разлика у односу на Entity Bean компоненте јесте у томе што JPA ради са *POJO* објектима, а опис пресликавања врши се коришћењем анотација и/или XML датотекама.

JPA је само спецификација, тј. скуп интерфејса и као такав није употребљив за перзистенцију података. Да би перзистенција била могућа, потребна је имплементација ових интерфејса. Постоји много opensource, али и комерцијалних имплементација JPA. Најзначајније су Hibernate (кога је подржао JBoss), TopLink (кога је подржао Oracle), и JDO (подржан од стране Oracle-a и BEA). JPA дефинише мањи скуп функционалности него нпр. Hibernate, али то не значи да оне нису доступне. Управо ове „додатне" функционалности утичу на одлуку о томекоју имплементацију JPA одабрати. Ако се одабере Hibernate, могуће је искористити и функционалности које не дефинише JPA, а које Hibernate посједује (нпр. Неке стратегије кеширања).

[35] DAO – Data Access Object представља софтверски патерн који омогућава раздвајање нивоа пословне логике софтверског система од складишта података, обично релационог система за управљање базама података. Може се посматрати као реализација перзистентног оквира који нивоу пословне логике система ставља на располагање скуп функционалности којима се обезбјеђује приступ и манипулација ентитетима базе података.

потреби, одабиром из падајућег менија на одговарајућој форми прототипа корисничког интерфејса.

Важно је напоменути да прототип који *AlphaSimple* алат генерише у тренутној верзији алата не обезбјеђује перзистентност података. Као такав, намијењен је искључиво за потребе демонстрације и утврђивања онога што корисник жели по питању функционалности будуће апликације. Такође, прототип нема могућности подешавања изгледа графичког корисничког интерфејса.

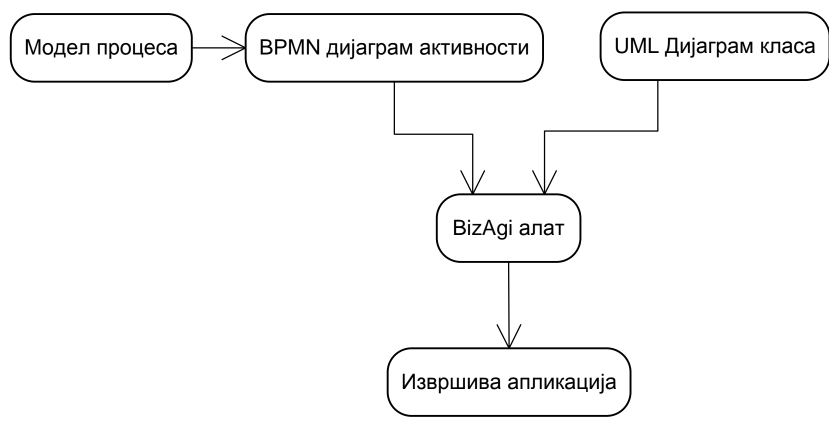
4.2.5. BizAgi BPM suite

BizAgi нуди комплетну платформу за аутоматизацију процеса. Ова платформа је пројектована тако да подржава унапређење процеса организације. Основни концепт *BizAgi* алата је аутоматско генерисање web апликације на основу модела представљеног дијаграмом активности и покретање генерисане апликације интерпретацијом тог модела. Аутори *BizAgi* алата за њега везују аксиом "процес је апликација", а у томе се огледа особина алата да генерише апликацију без потребе програмирања. Да би се апликација генерисала на овакав начин *BizAgi BPM Suite* алат управља комплетним животним циклусом пословног процеса.

Животни циклус пословног процеса подијељен је у три фазе: моделовање, извршавање и побољшавање. *BizAgi* алат садржи компоненте за сваку од ових фаза. Те компоненте омогућавају изградњу рјешења заснованог на процесу коришћењем графичког и динамичког окружења.

BizAgi BPM Suite [BizAgi] је самосталан алат који се инсталира и покреће на локалном рачунару. По креирању новог пројекта потребно је креирати нови процес и представити га *BPMN* дијаграмом активности. По изради новог, или импортовању

постојећег дијаграма активности, прелази се на повезивање процеса са моделом података и израду форми графичког корисничког интерфејса. На основу тога алат генерише web апликацију која се креира у времену извршења. Описани процес приказан је на слици:



Слика 69: Процес израде апликације у *BizAgi* алату

BPMN (*The Business Processs Modelling Notation*) [BPMN] је стандард који се користи у индустрији јер пружа нотацију која омогућава људима који раде на дефинисању процеса да те процесе изразе графички, на јасан, стандардизован и потпун начин. *BPMN* представља графички оријентисан језик подржан од стране многих алата. Дизајниран је тако да буде лак за коришћење и разумљив за тумачење. Главни циљ ове нотације је олакшавање моделовања пословних процеса и олакшавање комуникације између доменских експерата из пословног окрижења и техничких инжењера.

Израда пројекта у *BizAgi* алату има неколико фаза:

- Моделовање процеса.
- Моделовање података.
- Израда форми.
- Дефинисање пословних правила.

- Додјељивање ресурса.
- Извршавање.

Најважнији корак у изради рјешења помоћу *BizAgi* алата огледа се у утврђивању процеса. Ово се обавља помоћу *Process Modeler* компоненте *BizAgi* алата. Моделовање процеса је полазна фаза која је кључна за израду животног циклуса у *BizAgi* алату. Главни задатак у овој фази је да се на дијаграму јасно прикажу све активности које чине одређени процес.

Затим се врши креирање структурираног модела података који ће на одговарајући начин обухватати пословне податке неопходне за сваки процес. Креирање модела података се у потпуности обавља у одговарајућој компоненти *BizAgi* алата.

Када се дефинишу подаци потребни за процес, потребно је дефинисати форме које ће бити приказане крајњем кориснику. Ово се дефинише помоћу *Form generator* компоненте *BizAgi BPM Suite* алата. Свакој активности процеса се додељује форма. Треба нагласити да се за сваки процес одређује један и само један процесни ентитет из модела података. Ентитет који је у једном процесу означен као процесни, не може бити означен као процесни у неком другом процесу. Ово је важно јер се форме за један процес односе прије свега на означени процесни ентитет, уз могућност рада са ентитетима који су са њим у вези (нпр. веза један према више).

BizAgi алат садржи *Business Rule Engine* компоненту која омогућава флексибилан начин представљања различитих ситуација, као што су:

- транзиције у току процеса,
- валидације у активностима,
- генерисање узастопних бројева,

- дефинисање корисничких група,
- дефинисање услова који чине поље у форми измјењивим, видљивим и/или захтијеваним.

BizAgi алат има за циљ побољшање ефикасности организације пружањем метода за распоређивање запослених ради извршавања сваке активности у процесу. Алати за додјељивање помажу организацији да прилагоди своје приоритете и адекватно додијели задатак одговарајућем ресурсу.

BizAgi има могућност позива других апликација током процеса, осим координисања људских процеса. *BizAgi* не замјењује постојеће трансакционе апликације, умјесто тога оне се користе током процеса када год је то потребно. Слој интеграције *BizAgi* алата омогућава интеракцију са постојећим системима у организацији. Ти системи могу користити различите платформе ради заједничког рада у оквиру моделираних процеса.

BizAgi BPM Server извршава и контролише пословне процесе изграђене у *BizAgi* студију. Сервер је заснован на скупу компоненти, које нуде све функционалности неопходне за ефективно управљање процесима у организацији (радни портали, пословна правила, интеграција, итд). *BizAgi BPM Server*, заснован на претходно изграђеном моделу, прати исправност и адекватност извршавања различитих задатака и активности пословног процеса. Контролише се и верификује да ли су задаци извршени у одговарајућем тренутку, од стране одговарајуће особе или ресурса и у складу са смијерницама, циљевима и осталим правилима компаније. Резултат претходно описаних фаза је то да ће *BPM Server* бити одговоран за тумачење и извршење модела, јер је он тај који крајњем кориснику представља *BizAgi* радни портал. Треба напоменути да у претходно описаним фазама није било потребе за било каквим генерисањем програмског кода.

Крајњи корисници свој посао обављају кроз радни портал. Доступан им је списак неизвршених активности, којима се на основу различитих критеријума могу одређивати нивои приоритета. Корисници такође могу да прегледају информације о перформансама њихових процеса. Такође, *BizAgi* нуди могућност одређивања приоритета активности коришћењем шеме семафора (црвено, жуто, зелено), тако да крајњи корисник има видљив индикатор приоритета према коме ће свој посао обављати.

4.2.6. NetBeans

Иако се NetBeans IDE као развојно окружење опште намјене разликује од претходно описаних алата који су специјализовани за генерисање корисничког интерфејса, или комплетних софтверских система, уврштен је у анализу јер омогућава аутоматско креирање корисничког интерфејса. Иако су могућности које пружа у контексту аутоматизације развоја корисничког интерфејса скромне, основна предност је у чињеници да се ово развојно окружење широко користи у развоју софтверских система, а функционалност генерисања корисничког интерфејса се испоручује у стандардном пакету, што потенцијално може допринијети популарности овог концепта и за посљедицу имати унапријеђење постојеће функционалности. NetBeans IDE омогућава аутоматизацију креирања програмског кода JSF веб апликације на основу JPA Entity класа чији се код такође може генерисати из истог окружења на основу шеме базе података. У неколико корака могуће је добити функционалну веб апликацију која омогућава извршење *CRUD* операција над одабраним доменским ентитетима и то на следећи начин:

1. Креирати базу података.
2. Креирати пројекат за веб апликацију.

3. Генерисати Entity класе на основу базе података (Entity Classes from Database)
4. На основу Entity класа генерисати JSF странице (JSF Pages from Entity Classes).

Добијена апликација може се извршити и постављањем на веб сервер, а омогућено је извршење основних *CRUD* операција. Међутим, није могућ одабир различитих графичких компоненти као ни избор шаблона – темплејта корисничког интерфејса. Веза више према један (many-to-one) је подржана на тај начин што се креира компонента за избор из листе постојећих инстанци повезаног ентитета. Међутим, веза један према више (one-to-many) није подржана, па се повезани ентитети морају независно креирати. Нпр. ако ентитет *Invoice* формира везу један према више са ентитетом *InvoiceItem,* креирање инстанце објекта *Invoice* се врши на једној, а креирање објекта *InvoiceItem* се врши на другој страници.

Предност коришћења ове функционалности је што је генерисани код директно доступан и може се даље прилагођавати, као и то што се од програмера као корисника алата не захтијева додатно познавање правила и дефинисање сложених модела како би се добила потпуно функционална апликација. Улазна спецификација је потпуно дефинисана доменским моделом, који се коришћењем wizard-а може аутоматски генерисати.

4.3. Резултати упоредне анализе постојећих алата на основу дефинисаних критеријума

Преглед карактеристика одабраних шест алата показује да сваки алат на себи својствен начин третира проблем аутоматизације креирања корисничког интерфејса система. Разлике су примјетне по сваком од дефинисаних критеријума поређења, осим у зависности улазне спецификације од модела података односно доменске структуре система, с тим што неки алати овај модел проширују и моделима који описују понашање софтверског система (модел процеса) као и моделима који су директно повезани са пројектовањем корисничког интерфејса апликације. Начин креирања улазне спецификације се такође разликује од случаја до случаја, али заједничка карактеристика свих посматраних алата је непостојање могућности избора између различитих начина креирања улазне спецификације. Наиме, чак и алати који предвиђају коришћење два различита начина креирања улазне спецификације захтијевају коришћење оба начина, сваки у различитим активностима процеса. Такође је интересантно да поједини алати и поред генерисања изворног кода корисничког интерфејса не предвиђају могућност његове директне манипулације – измјене или проширења. Када се погледа веома ограничена подржаност или потпуно одсуство избора између различитих шаблона корисничког интерфејса, као и ограничења везана за реализацију различитих врста веза између ентитета доменског модела на корисничком интерфејсу, може се закључити да приказани алати у великој мјери сужавају простор програмеру као кориснику алата да одговори на специфичне захтјеве корисника. Такође, и поред тога што поједини алати међу посматраним подржавају развој различитих типова апликација (десктоп, веб или мобилне апликације), чињеница да алати не пружају могућност одабира између различитих шаблона корисничког интерфејса већ доминантно нуде један предефинисани шаблон,

намеће питање да ли предефинисани шаблон узима у обзир специфичне карактеристике различитих типова апликације. Наиме, одређена добра пракса која се примјењује приликом пројектовања корисничког интерфејса за десктоп апликације се обично не може директно примијенити на мобилне или веб апликације, најчешће из разлога који се тичу технолошких ограничења, али и стечених навика корисника, о чему ће више ријечи бити у наредном поглављу.

Ипак, сваки од посматраних алата садржи неке специфичне карактеристике које могу бити веома интересантне и примјењиве приликом дефинисања новог приступа аутоматизацији креирања корисничког интерфејса. Биће поменут концепт инспекције (анализе) различитих елемената постојећег система како би се убрзао развој и омогућила лакша интеграција добијеног интерфејса са постојећим дијеловима система коју у први план истиче *Metawidget.* Такође, интересантна је и могућност директног распоређивања елемената корисничког интерфејса коју пружа *WebRatio,* иако се на овај начин отвара могућност нарушавања конзистентности између корисничког захтјева (нпр. редоследа корака у сценарију случаја коришћења) и распореда графичких компоненти (које можда више неће пратити ток процеса који је дефинисан сценариом). Треба поменути и предности коришћења алата директно из развојног окружења које омогућавају *WebRatio (*који се интегрише са Eclipse развојним окружењем*)* и *NetBeans* IDE, уз напомену да се алат за аутоматизацију креирања корисничког интерфејса испоручује уз основни пакет *NetBeans* развојног окружења, што га чини директно доступним широкој заједници корисника. Треба напоменути и то да и поред скромних могућности добијеног интерфејса, *NetBeans* не захтијева учење додатних правила за креирање улазне спецификације.

Утисак који се стиче након дубоке анализе карактеристика посматраних алата, са изузетком *WebRatio* алата*,* је да се могу углавном користити као помоћни алати приликом израде појединих једноставнијих дијелова система који су намијењени

корисницима који су добро упознати са пословним доменом, или дијеловима система за које није пресудна интуитивност и конфорност у коришћењу корисничког интерфејса (нпр. функционалности које служе за подешавања система које обично извршавају администратори система и сл.) или за креирање прототипова система који би се користили за валидацију корисничких захтјева. Са друге стране *WebRatio* алат обезбјеђује услове за развој комплетних пословних апликација, уз ограничења у избору шаблона корисничког интерфејса, као и у приступу и манипулацији генерисаним програмским кодом, што представља значајно ограничење приликом одговора на специфичне захтјеве корисника који нису предвиђени моделима на основу којих се врши генерисање корисничког интерфејса.

У наставку (Табела 4) су табеларно приказане уочене карактеристике посматраних алата на основу дефинисаних критеријума за поређење:

Табела 4: Карактеристике посматраних алата на основу дефинисаних критеријума за поређење

	Apache Isis	Metawidget	WebRatio	AlphaSimple	BizAgi BPM	NetBeans IDE
Улазна спецификација	Доменски модел	Доменски модел	WebML/BPMN модел	textUML	BPMN модел	Доменски модел
Начин дефинисања улазне спецификације	Ручни унос	Ручни унос	Wizard и визуелно моделовање	Ручни унос	Wizard и визуелно моделовање	Wizard
Зависност улазне спецификације од модела података	Да	Да	Да	Да	Да	Да
Начин креирања корисничког интерфејса	При извршењу	При извршењу	Изворни код	Изворни код	Изворни код	Изворни код
Подржана могућност избора шаблона корисничког интерфејса	Не	Не	Да (ограничено)	Не	Не	Не
Измјењивост генерисаног програмског кода	Не	Не	Да (ограничено)	Да	Не	Да
Подршка за различите типове апликација	Да	Да	Да	Да	Не	Не

5. Анализа корисничког интерфејса различитих типова апликација у различитим имплементационим технологијама

Предмет дисертације јесте аутоматизација процеса пројектовања и имплементације корисничког интерфејса, као и веза између софтверских захтјева и будућег корисничког интерфејса апликације узимајући у обзир карактеристике циљаних технологија и типова софтверских система. Резултати истраживања приказани у претходном поглављу недвосмислено указују на потребу да алат за аутоматизацију креирања корисничког интерфејса мора да омогући кориснику избор између различитих типова софтверских система, као и различитих имплементационих технологија. Типове софтверских система у контексту пословних апликација на које се фокусира ово истраживање, најгрубље се могу класификовати на десктоп, веб и мобилне апликације, при чему се за израду сваког од поменутих типова софтверских система могу користити различите имплементационе технологије, или комбинације имплементационих технологија.

У претходном поглављу анализирани су неки од постојећих алата за генерисање корисничког интерфејса. Уочено је да већина алата омогућава избор између различитих циљних имплементационих технологија и типова софтверских система, али чињеница да алати доминантно нуде један предефинисани шаблон корисничког интерфејса, навела је на размишљања о томе да ли предефинисани шаблон узима у обзир специфичне карактеристике различитих типова апликација. Искуство стечено при развоју бројних софтверских система различитих типова

говори о томе да одређена добра пракса која се примјењује приликом пројектовања корисничког интерфејса за десктоп апликације обично не може да се директно примијени на мобилне или веб апликације. Преовлађујући разлог за другачији начин пројектовања корисничког интерфејса различитих типова софтверских система тиче се технолошких ограничења. Брз развој информационих технологија омогућио је да се ова ограничења временом превазиђу, али разлике у пројектовању корисничког интерфејса и даље постоје, и то не као посљедица технолошких ограничења, већ превасходно због стечених навика корисника, које се мијењају много теже и спорије од развоја технологије.

У овом поглављу биће разматране специфичности корисничких интерфејса различитих типова софтверских система (десктоп, веб и софтверске системе за мобилне уређаје) и имплементационих технологија које се користе за њихов развој. Циљ ове анализе је идентификација карактеристика корисничког интерфејса различитих типова софтверских система. Будући алат за аутоматизацију генерисања корисничког интерфејса мора узети у обзир идентификоване карактеристике како би на одговарајући начин одговорио на захтјеве корисника који су јасно артикулисани резултатима истраживања који су изложени у претходном поглављу. У литератури [WebOrDesktop] [Alor-Hernandez14] постоје бројна поређења различитих типова софтверских система са циљем да се утврде предности и мане сваког од њих. Оваква анализа је са становишта нашег истраживања ирелевантна, већ ће фокус бити на разликама које постоје у пројектовању корисничког интерфејса у односу на одабрани тип софтверског система.

Прије анализе карактеристика корисничког интерфејса десктоп, веб и мобилних апликација, биће приказан осврт се на принципе пројектовања корисничког интерфејса који су примјењиви на било који од поменутих типова софтверских система, као и на препоруке за рјешавање уобичајених проблема при

пројектовању које се често у литератури [Tidwell05] [UIPatterns] називају патернима корисничког интерфејса (*User Interface Patterns*).

5.1. Принципи пројектовања корисничког интерфејса

У овом дијелу рада биће дат преглед основних принципа пројектовања корисничког интерфејса независно од типа софтверског система. Најчешће цитирана листа принципа за пројектовање корисничког интерфејса [Constantine99] по ријечима аутора[36] намијењена је побољшању квалитета пројектовања корисничког интерфејса:

- **Принцип структуре** – Пројектовани кориснички интерфејс треба да буде организован намјенски, на смислен и употребљив начин заснован на јасним моделима који су препознатљиви кориснику. Препорука је да се групишу елементи који су на одређени начин повезани, а да се раздвоје елементи који нису повезани, а такође да се јасно прикажу слични елементи као и да се направи јасна разлика између различитих елемената. Принцип структуре се тиче како појединачног корисничког интерфејса тако и комплетне пројектоване архитектуре корисничког интерфејса.

[36] Аутори Larry Constantine и Lucy Lockwood у својим радовима [Constantine03] [Constantine00] [Constantine02] дали су огроман допринос моделом вођеном развоју корисничког интерфејса, и то прије свега вези између модела који се користе у раним фазама софтверског пројекта са финалним корисничким интерфејсом. Као резултат аутори су дефинисали методу развоја под називом *Usage-Centered Design* [Constantine99]. Радови ових аутора имали су значајан утицај на израду ове дисертације. Поред тога, интересантно је поменути и да је Larry Constantine заслужан и за увођење концепата кохезије (cohesion) – степена повезаности интерних елемената модула, и купловања (coupling) – степена зависности одређеног модула од осталих модула. Ови концепти представљају фундаменталне принципе пројектовања софтвера.

- **Принцип једноставности** – Пројектовани кориснички интерфејс треба да омогући што једноставније извршње задатака који се често понављају, али и да обезбиједи што смисленије пречице до компликованијих процеса и процедура. Интеракција треба да буде потпуно прилагођена кориснику и језику[37] који корисник разумије.
- **Принцип видљивости** – Пројектовани кориснички интерфејс треба да обезбеди да су све опције и потребни параметри за извршење одређеног задатка видљиви. Такође, кориснику не треба скретати пажњу са основног задатка неким мање релевантним или непотребним информацијама. Добро пројектовани интерфејс не „затрпава“ корисника са великим бројем могућих алтернатива.
- **Принцип повратних информација** – Пројектовани кориснички интерфејс треба константно да обавјештава корисника о извршеним акцијама или промјенама стања или услова рада, као и о грешкама или изузецима које су релевантне и од интереса за корисника на јасан, концизан, недвосмислен начин, језиком који познаје корисник.
- **Принцип толеранције** – Пројектовани кориснички интерфејс треба да омогући флексибилност и толеранцију на грешке које прави корисник. Треба омогућити поништавање одређених акција корисника, али и, кад год је могуће, вршити превенцију настајања грешака.
- **Принцип поновног коришћења** – Пројектовани кориснички интерфејс треба да користи унутрашње и спољашње компоненте и понашања, намјерно задржавајући досљедност како би се смањила потреба да корисник у новим ситуацијама изнова размишља о акцијама које треба да изврши.

[37] Овдје се под језиком не подразумијева само коришћење језика који говори корисник, већ се мисли и на саме изразе и начин формулације порука који треба да буде у складу са доменом који познаје корисник.

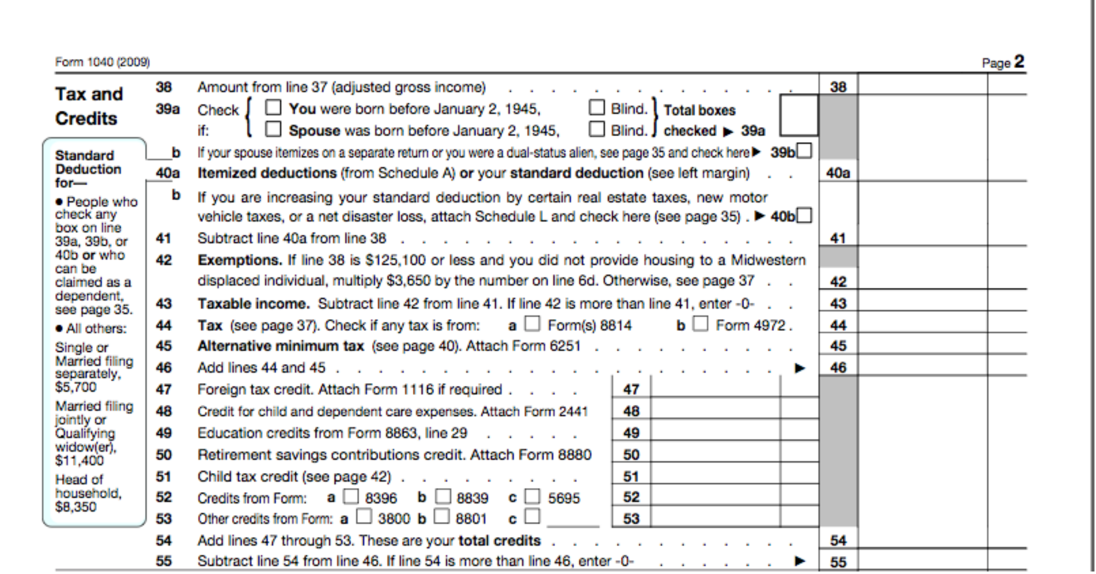
Поред ових основних принципа, у литератури [Usability] се још могу наћи и принцип корисности[38] и ефикасности[39], али се већина њих своди на већ поменуте принципе. У погледу пројектовања корисничког интерфејса, значајно је размотрити и концепте дедуктивног и индуктивног корисничког интерфејса [UIDesign] и при пројектовању тежити равнотежи између једноставности и функционалности корисничког интерфејса.

Дедуктивни кориснички интерфејс (*Deductive User Interface*) – Овај концепт подразумијева презентовање кориснику великог броја могућих акција на једној екранској форми (Слика 70). Овакав интерфејс је погодан за искусне кориснике који могу да закључе на шта се односе сви дијелови корисничког интерфејса, као и које акције треба да позове. За просјечног корисника ово може представљати проблем.

---

[38] Принцип корисности (*usefulness*) предвиђа да пројектовани кориснички интерфејс пружи кориснику информације и функционалности које су релевантне за контекст и задатак који корисник тренутно обавља. Овај принцип је сличан принципу видљивости.

[39] Принцип ефикасности препоручује пројектовање корисничког интерфејса на тај начин да убрза рад корисника омогућавањем пречица и функцијских тастера за често коришћене акције. Овај принцип је у сагласју са принципом једноставности.



Form 1040 (2009) Page 2

Tax and Credits

38 Amount from line 37 (adjusted gross income) . . . 38

39a Check if: { ☐ **You** were born before January 2, 1945, ☐ Blind. } **Total boxes checked ▶ 39a**
☐ **Spouse** was born before January 2, 1945, ☐ Blind.

b If your spouse itemizes on a separate return or you were a dual-status alien, see page 35 and check here ▶ 39b ☐

40a **Itemized deductions** (from Schedule A) **or** your **standard deduction** (see left margin) . . 40a

b If you are increasing your standard deduction by certain real estate taxes, new motor vehicle taxes, or a net disaster loss, attach Schedule L and check here (see page 35) . ▶ 40b ☐

41 Subtract line 40a from line 38 . . . 41

42 **Exemptions.** If line 38 is $125,100 or less and you did not provide housing to a Midwestern displaced individual, multiply $3,650 by the number on line 6d. Otherwise, see page 37 . . 42

43 **Taxable income.** Subtract line 42 from line 41. If line 42 is more than line 41, enter -0- . . 43

44 **Tax** (see page 37). Check if any tax is from: a ☐ Form(s) 8814 b ☐ Form 4972 . 44

45 **Alternative minimum tax** (see page 40). Attach Form 6251 . . . 45

46 Add lines 44 and 45 . . . ▶ 46

47 Foreign tax credit. Attach Form 1116 if required . . . 47

48 Credit for child and dependent care expenses. Attach Form 2441 48

49 Education credits from Form 8863, line 29 . . . 49

50 Retirement savings contributions credit. Attach Form 8880 50

51 Child tax credit (see page 42) . . . 51

52 Credits from Form: a ☐ 8396 b ☐ 8839 c ☐ 5695 52

53 Other credits from Form: a ☐ 3800 b ☐ 8801 c ☐ 53

54 Add lines 47 through 53. These are your **total credits** . . . 54

55 Subtract line 54 from line 46. If line 54 is more than line 46, enter -0- . . . ▶ 55

Standard Deduction for—
• People who check any box on line 39a, 39b, or 40b **or** who can be claimed as a dependent, see page 35.
• All others:
Single or Married filing separately, $5,700
Married filing jointly or Qualifying widow(er), $11,400
Head of household, $8,350

Слика 70: Дедуктивни кориснички интерфејс

На основу прегледа основних принципа пројектовања корисничког интерфејса може се закључити да корисника не треба доводити у ситуацију да се двоуми и размишља о акцијама које треба да позове на корисничком интерфејсу, већ је кориснички интерфејс тај који треба да обезбиједи одговоре на та питања и да води корисника кроз процес.

Индуктивни кориснички интерфејс (*Inductive User Interface*) – Дефинише тип интерфејса који би требао да ријеши проблеме који настају пројектовањем дедуктивног корисничког интерфејса. У сагласности је са природом веб апликација и захтјев – одговор (*request – response*) парадигмом која је њима својствена[40]. У основи овог концепта је да извршење једног појединачног задатка одговара једној екранској форми корисничког интерфејса (Слика 71). Дакле, свака екранска форма треба да

[40]Ова парадигма је била доминантна све до појаве AJAX (*Asynchronous JavaScript and XML*) технологије, када је постало могуће асинхроно упућивање захтјева са клијентске странице ка серверу, као и асинхрони пријем и обраду одговора сервера. До појаве AJAX технологије, свака клијентска веб страница је била повезана са са једним захтјевом према серверу, који као одговор шаље клијенту нову веб страницу.

омогући извршавање једног, прецизно дефинисаног задатка. Сви елементи на екранској форми треба да буду у функцији извршавања тог задатка. Једини изузетак може бити мени за избор блиско повезаних задатака са задатком коме је екранска форма намијењена. Може се рећи да је на овај начин корисник вођен кроз процес, што омогућава једноставно коришћење апликације и од стране мање искусних корисника.

Tell Us Why You Moved

Choose the situation that applies to you, and we'll determine if you can **deduct any expenses** associated with your move to chicago.

Note: Do not select any if you are a retiree or surviving spouse (or dependent) returning to the United States. Explain This

- My move to a new location was work-related.
- I am a member of the armed forces and moved due to a Permanent Change of Station (PCS).
- I paid only storage fees related to my move to a location outside the United States in an earlier year.
- None of the above

Back Continue

Слика 71: Индуктивни кориснички интерфејс

Овај концепт погодан је и за развој десктоп апликација, јер олакшава кориснику извршење неког сложеног задатка на тај начин што га води кроз комплетан процес.

Када су у питању мобилне апликације, овај концепт је посебно значајан узевши у обзир ограничења која проистичу из величине екрана мобилног уређаја. Сврха сваке екранске форме треба да буде јасно дефинисана.

Недостатак овог концета је то што може да успори извршење процеса код искусних корисника, па је потребно је наћи компромис између ова два описана концепта.

Пројектовање корисничког интерфејса подразумијева прављење компромиса између потреба корисника и осталих заинтересованих страна, нових и корисника са искуством, функционалности и једноставности корисничког интерфејса. Да би се успјешно креирао кориснички интерфејс потребно је пажњу усмјерити ка корисницима и задацима које треба да ријеше, односно основним функционалностима апликације. У наставку ће бити приказани тзв. патерни корисничког интерфејса који имају циљ да дају рјешење за уобичајене проблеме у развоју корисничког интерфејса [UIPatterns].

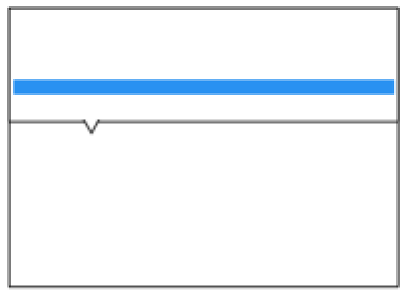
5.1.1. Патерни пројектовања корисничког интерфејса

Постоје бројне препоруке за рјешавање уобичајених проблема у пројектовању корисничког интерфејса које се у литератури [Scott09] [UIPatterns] [Tidwell05] често називају патернима корисничког интерфејса. Иако се у литератури може наћи на стотине патерна корисничких интерфејса, у наставку ће бити приказана листа од 13 патерна корисничког интерфејса, који су представљени у књизи [Scott09], а који су по нашем мишљењу најрелевантнији за развој пословних апликација. Примјена патерна корисничког интерфејса може смањити криву учења корисника приликом упознавања са корисничким интерфејсом.

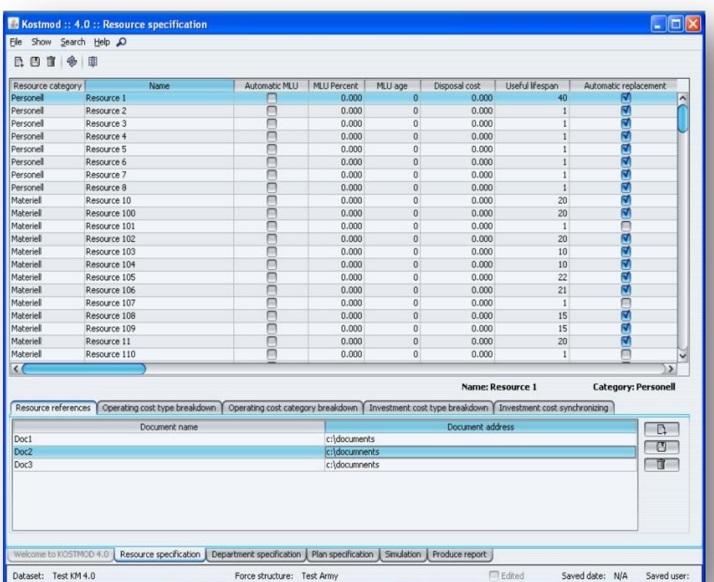
1. Master/Detail патерн

Master/Detail патерн је један од најчешће коришћених у развоју пословних апликација. Омогућава навигацију кроз елементе листе објеката и детаљан приказ одабраног елемента.

Слика 72: Хоризонтални Master/Detail патерн



Слика 73: Вертикални Master/Detail патерн

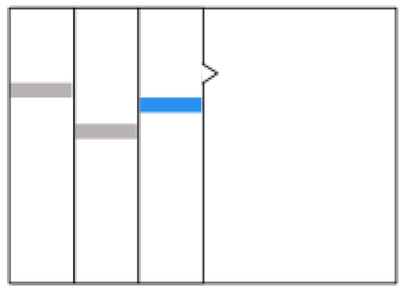


Слика 74: Примјер коришћења Master/Detail патерна у десктоп апликацији[41]

[41] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Костмод 4.0, случај коришћења Resource specification.

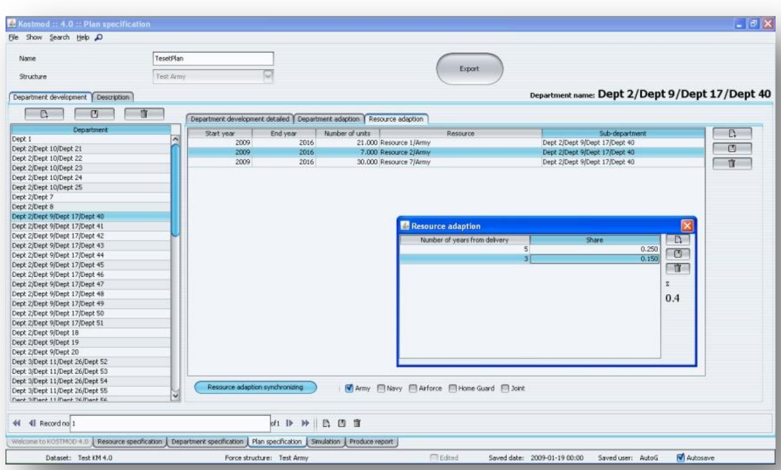
2. Column Browse патерн

Column Browse патерн омогућава навигацију кроз елементе листе објеката у више нивоа хијерархије и детаљан приказ одабраног елемента. Заправо се може посматрати као Master/Detail патерн уколико структура која се приказује има више нивоа хијерархије.



Слика 75: Хоризонтални Column Browse патерн

Слика 76: Вертикални Column Browse патерн

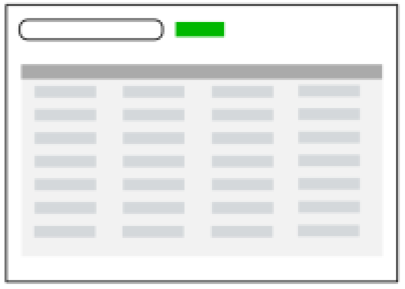


Слика 77: Примјер коришћења Column Browse патерна у десктоп апликацији[42]

[42] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Костмод 4.0, случај коришћења Plan specification.

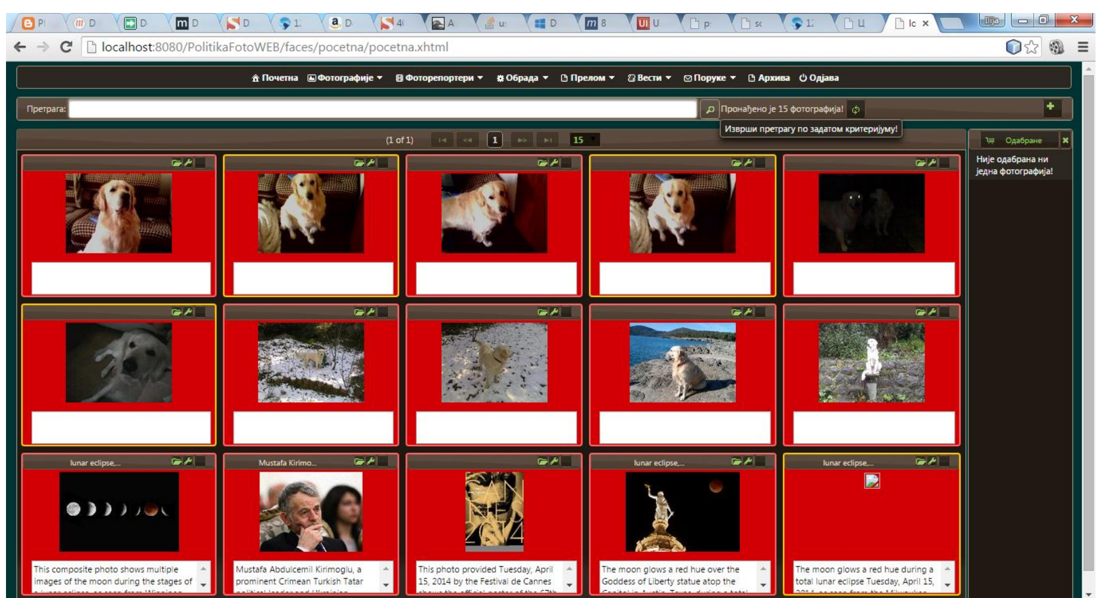
3. Search/Results патерн

Search/Results патерн користи се за пројектовање корисничког интерфејса чија је основна улога дефинисање критеријума за претрагу и навигацију кроз објекте који задовољавају дефинисане критеријуме.



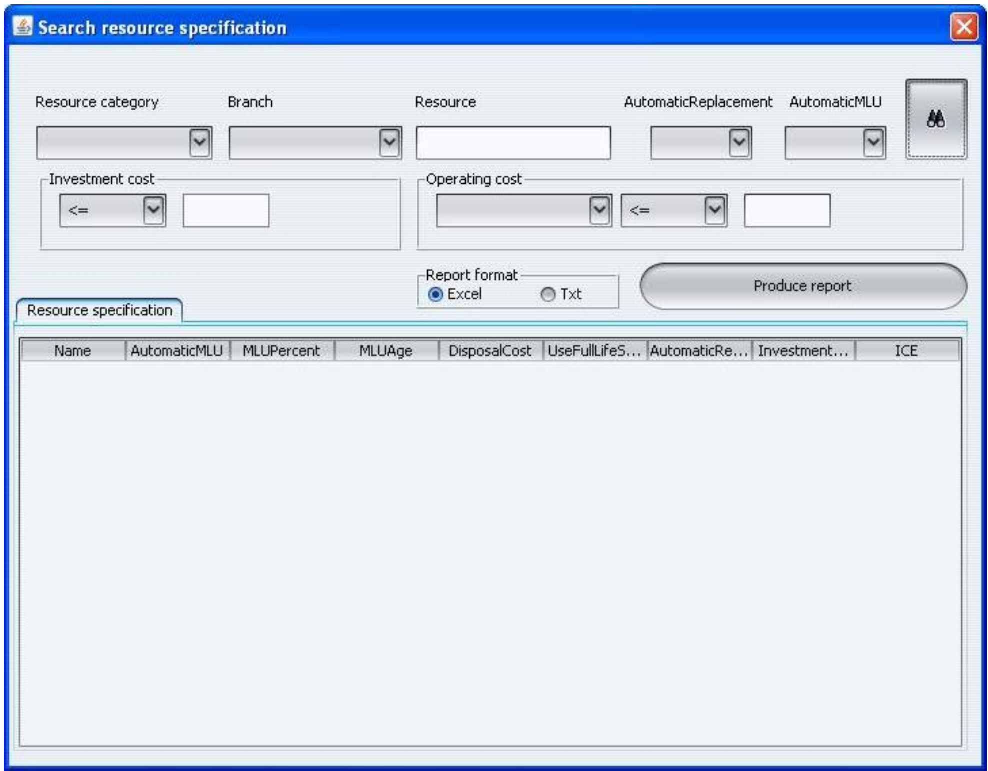
Слика 78: Search/Results патерн са простим критеријумом

Слика 79: Search/Results патерн са сложеним критеријумима



Слика 80: Примјер коришћења Search/Results патерна у веб апликацији[43]

[43]Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Политика – Фото модул, случај коришћења Преглед свих фотографија.

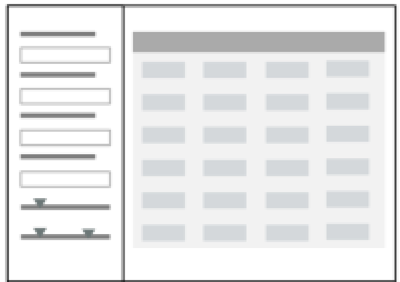


Слика 81: Примјер коришћења Search/Results патерна у десктоп апликацији[44]

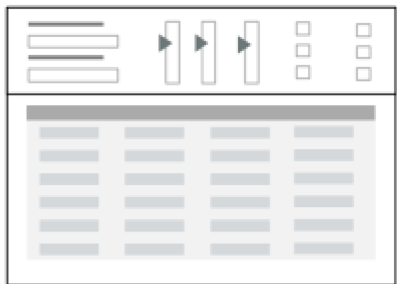
[44] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Костмод 4.0, случај коришћења Search Resource specification.

4. Filter Dataset патерн

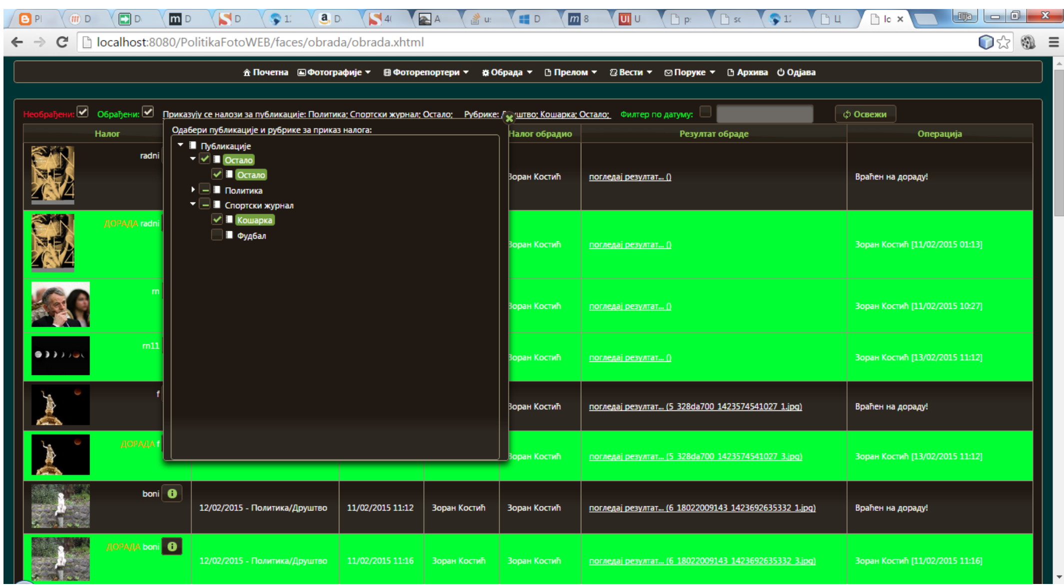
Filter Dataset патерн се користи за додатно филтрирање приказаних објеката, најчешће добијених претходно извршеном претрагом. Може се посматрати и као специјалан случај Search/Results патерна.



Слика 82: Хоризонтални Filter Dataset патерн



Слика 83: Вертикални Filter Dataset патерн



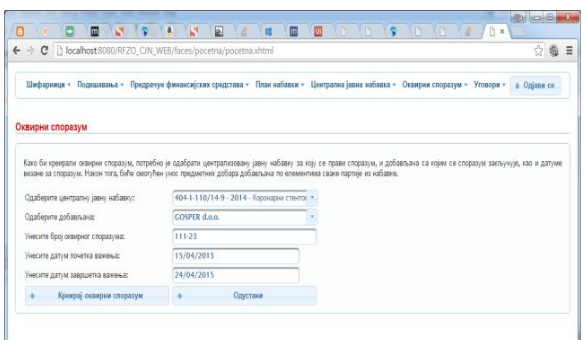
Слика 84: Примјер коришћења Filter Dataset патерна у веб апликацији[45]

[45]Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Политика – Фото модул, случај коришћења Преглед налога за прелом.

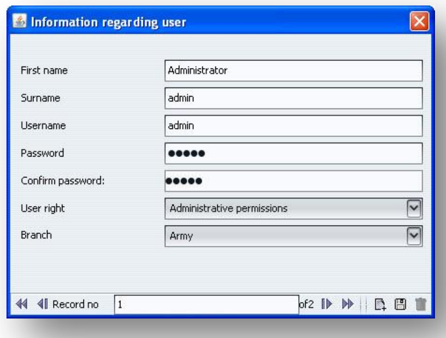
5. Form патерн

Form патерн се користи како би се кориснику омогућио унос података и позив одређене системске операције. Све компоненте намијењене уносу потребних података за извршење једног задатка организоване су у оквиру једне екранске форме.

Слика 85: Form патерн



Слика 86: Примјер коришћења Form патерна у веб апликацији[46]



Слика 87: Примјер коришћења Form патерна у десктоп апликацији[47]

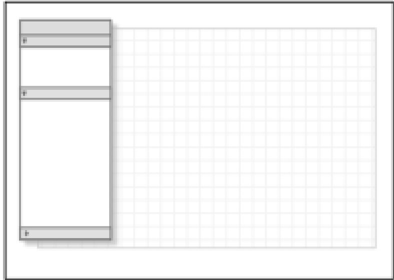
---

[46]Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Менаџмент јавним набавкама, случај коришћења Креирање оквирног споразума.

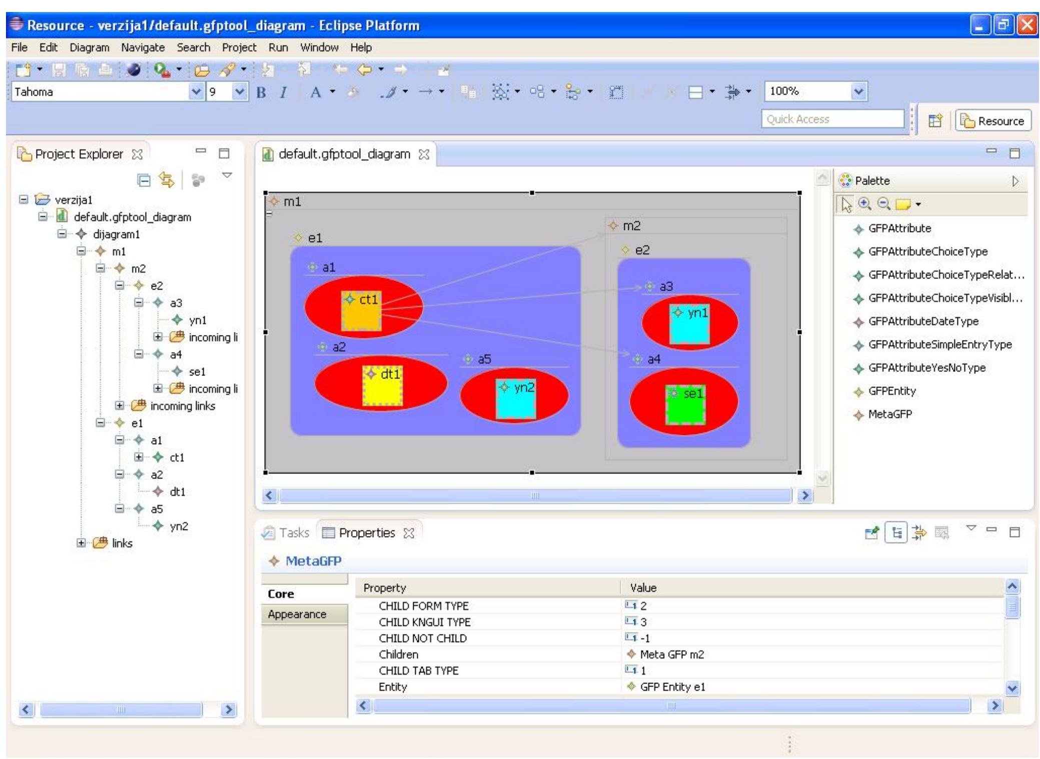
[47] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Костмод 4.0, случај коришћења Information regarding user.

6. Palette/Canvas патерн

Palette/Canvas патерн се обично користи код апликација које обезбјеђују визуелно моделовање, нпр. за креирање дијаграма, графова и сл.



Слика 88: Palette/Canvas патерн

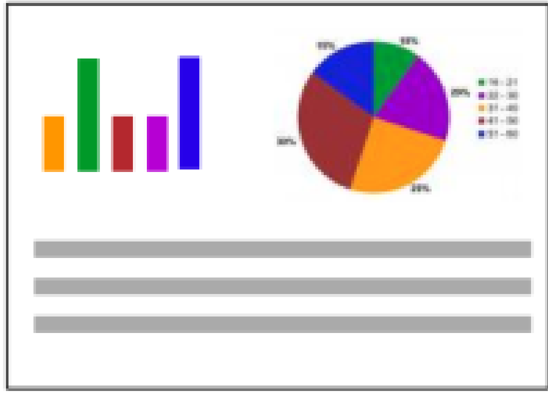


Слика 89: Примјер коришћења Palette/Canvas патерна за Eclipse Modeling Framework[48]

[48] Приказани примјер је дио корисничког интерфејса алата за креирање улазне спецификације у оквиру *SilabUI* приступа, који је пројектован за потребе овог истраживања.

7. Dashboard патерн

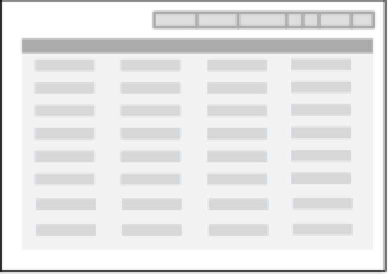
Dashboard патерн се обично користи за приказ сумарних извјештаја који омогућава једноставно тумачење кључних информација.



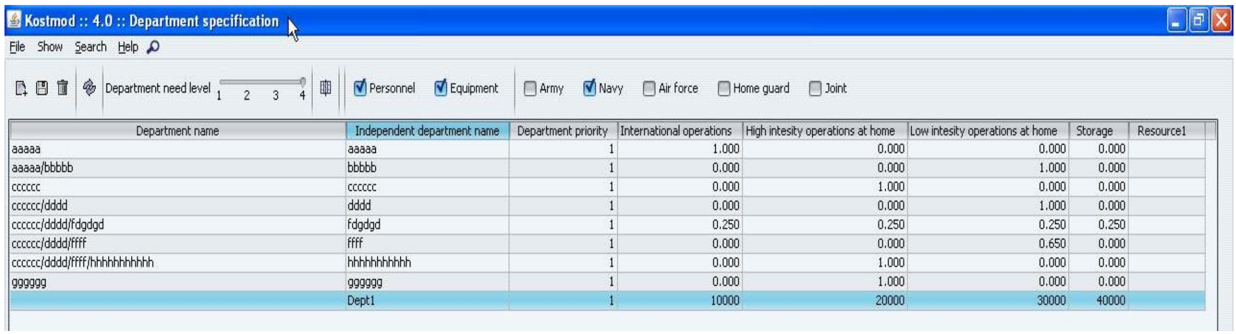
Слика 90: Dashboard патерн

8. Spreadsheet патерн

Spreadsheet патерн је још један од често коришћених патерна при пројектовању корисничких интерфејса за пословне апликације. Омогућава навигацију, претраживање, филтрирање, сортирање, унос и обраду табеларно приказаних података. Унос и измјена података се може вршити директно у оквиру табеле, а може се вршити и у комбинацији са Form патерном, најчешће у зависности од типа софтверског система, о чему ће више ријечи бити у наставку.



Слика 91: Spreadsheet патерн

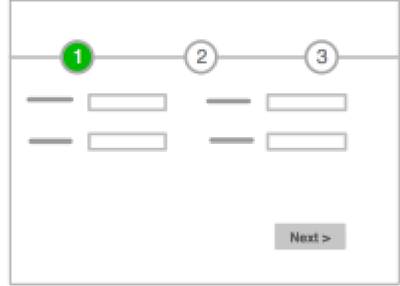


Слика 92: Примјер коришћења Spreadsheet патерна у десктоп апликацији[49]

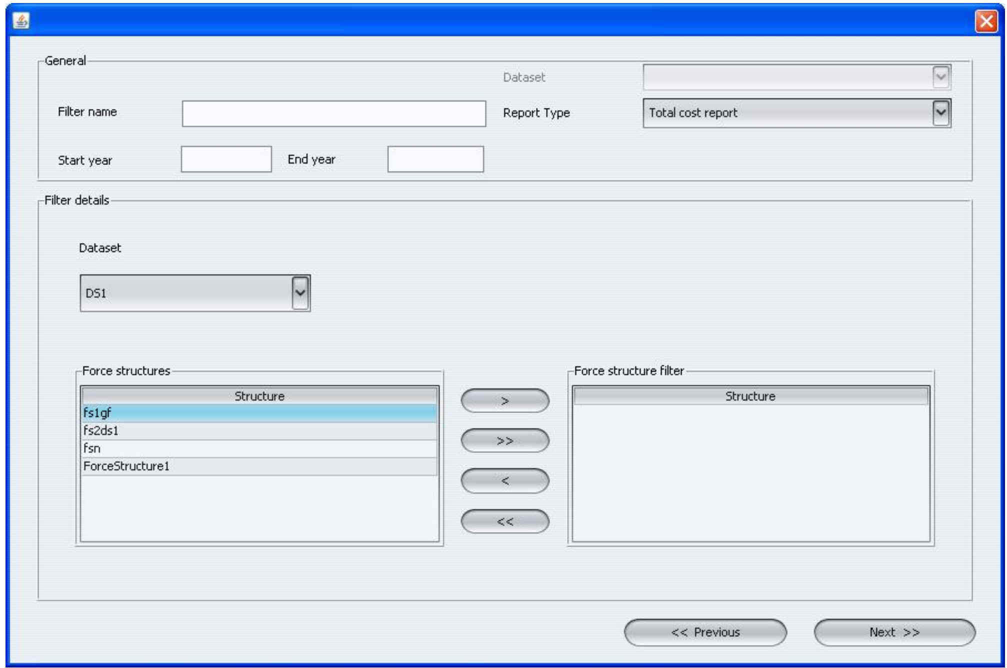
[49] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Костмод 4.0, случај коришћења Department specification.

9. Wizard патерн

Wizard патерн је намијењен за пројектовање корисничког интерфејса који олакшава кориснику извршење сложеног задатка на тај начин што га води кроз поступак корак по корак, све док се не заврши комплетан процес.



Слика 93: Wizard патерн



Слика 94: Примјер коришћења Wizard патерна у десктоп апликацији[50]

[50] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Костмод 4.0, случај коришћења Create report and export results.

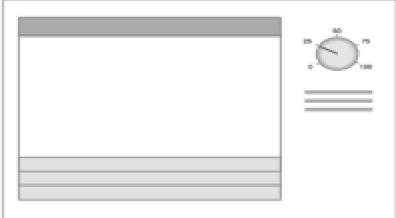
10. Question/Answer патерн

Question/Answer патерн омогућава кориснику да на бази унесених познатих информација добија једно или више могућих рјешења или препорука. Сличан је Search/Results патерну, са том разликом што су код претраге могући критеријуми унапред дефинисани, док се у овом случају на основу корисниковог уноса закључује о могућим рјешењима и препорукама.

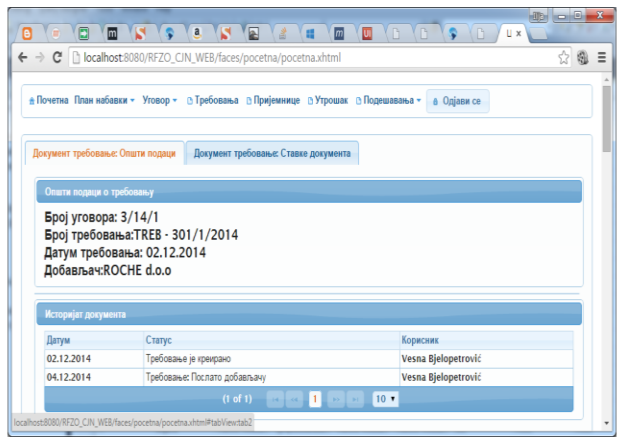
Слика 95: Question/Answer патерн

11. Parallel Panels патерн

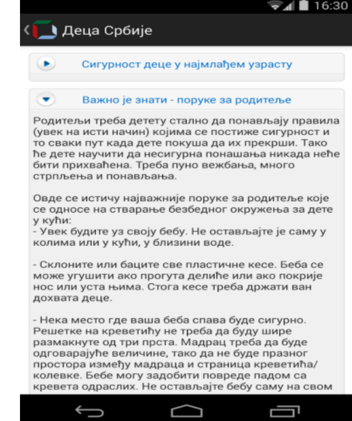
Parallel Panels патерн се користи када се подаци на једној екранској форми организују у специфичне категорије, при чему је свака категорија приказана у засебном панелу, а корисник има могућност да приступи панелу који му је потребан, као и да се креће из једног у други панел, не напуштајући екранску форму. Панели се обично распоређују по картицама (табовима), при чему је једна картица активна и припадајући панел је приказан кориснику, док су други панели сакривени, све док се не одабере одговарајућа картица.



Слика 96: Parallel Panels патерн



Слика 97: Примјер коришћења Parallel Panels патерна у веб апликацији[51]



Слика 98: Примјер коришћења Parallel Panels патерна у мобилној апликацији[52]

---

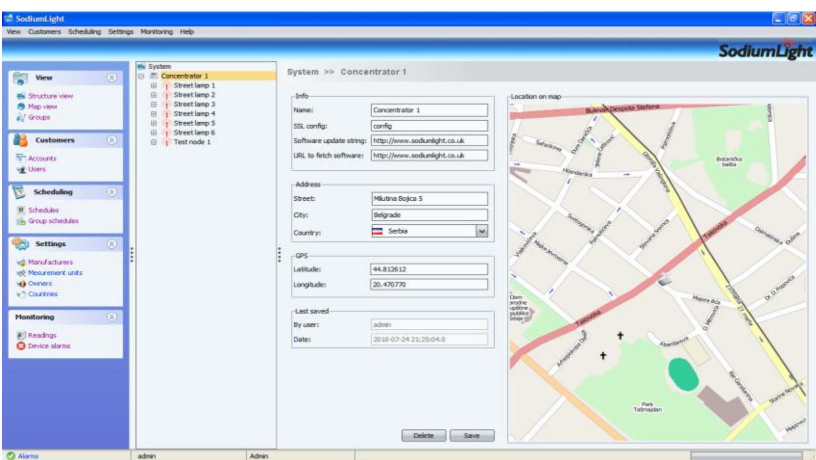
[51] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Менаџмент јавним набавкама, случај коришћења Обрада требовања.

[52] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем РФЗО – Деца Србије, случај коришћења Поруке за родитеље.

12. Interactive Model патерн

Interactive Model патерн се обично користи за графички приказ и обраду специфичних података на начин који је логички близак кориснику. Нпр. за рад са географским локацијама, кориснику је много ближи начин приказа локације на мапи, него података о географској ширини и дужини. Такође, за рад са датумима, обично је много једноставније кориснику да ради са компонентама које приказују календар, него тражити да уноси или мијења датум у текстуалном формату.

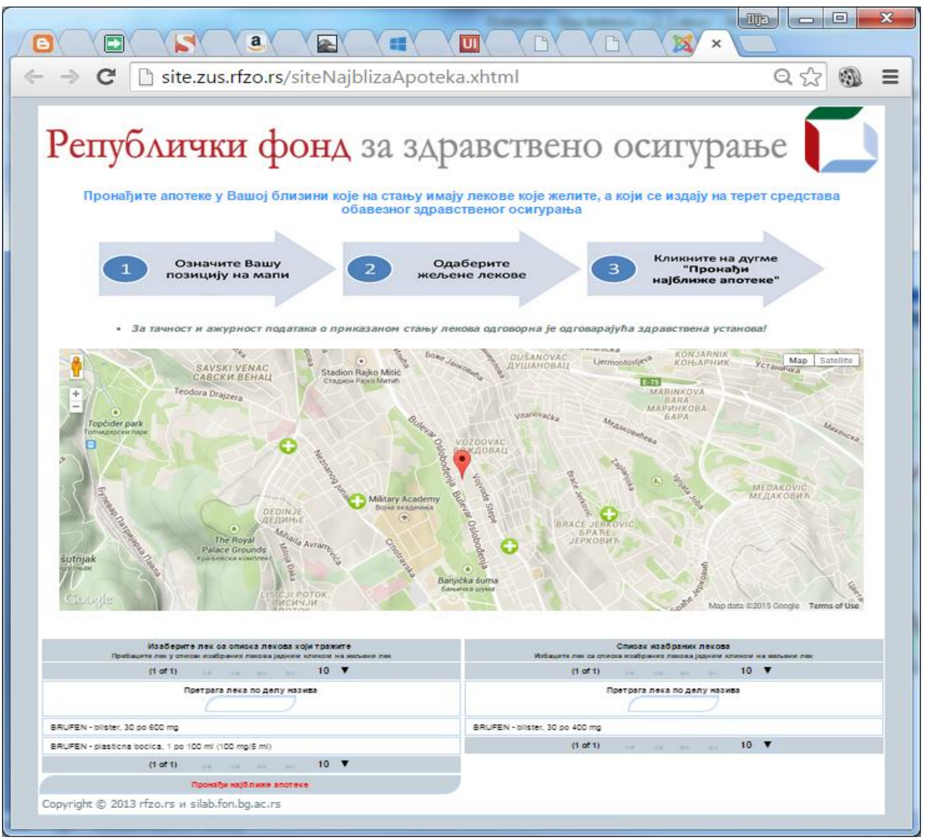
Слика 99: Interactive Model патерн



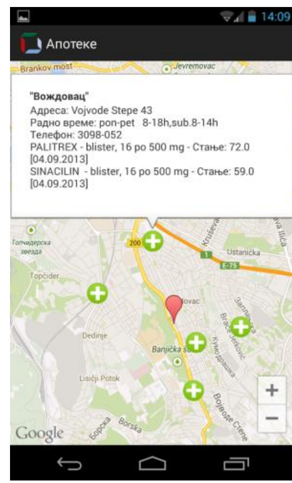
Слика 100: Примјер коришћења Interactive Model патерна у десктоп апликацији[53]

---

[53]Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Менаџмент јавним осветљењем - SodiumLight, случај коришћења Edit concentrator settings.



Слика 101: Примјер коришћења Interactive Model патерна у веб апликацији[54]



Слика 102: Примјер коришћења Interactive Model патерна у мобилног апликацији[55]

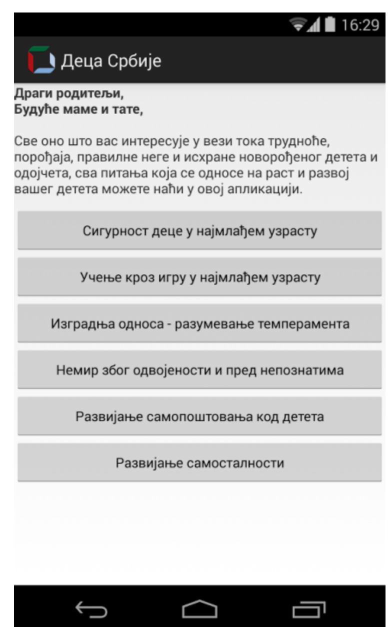
[54] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем Нађи лек, случај коришћења Пронађи лек у најближој апотеци помоћу мапе.

[55] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем РФЗО Апотеке, случај коришћења Пронађи лек у најближој апотеци помоћу тренутне GPS локације.

13. Blank State патерн

Blank State патерн се користи за приказ одређених информација кориснику прије него што се учитају специфични подаци или се позове одређена акција. Може садржати видео записе, објашњења везана за коришћење система или неке друге корисне информације.

Слика 103: Blank State патерн



Слика 104: Примјер коришћења Blank State патерна у мобилној апликацији[56]

[56] Приказани примјер је дио корисничког интерфејса пројектованог за софтверски систем РФЗО – Деца Србије.

5.2. Анализа карактеристика корисничког интерфејса десктоп, веб и мобилних пословних апликација

Претходно описани принципи пројектовања корисничког интерфејса, концепти који се користе приликом пројектовања, као и препоруке односно патерни корисничког интерфејса се могу сматрати универзалним, обзиром да су примјењиви у пројектовању десктоп, веб и мобилних пословних апликација, што се може закључити и увидом у приказане примјере коришћења различитих патерна корисничког интерфејса. Ипак, како је раније истакнуто, искуство стечено при развоју бројних софтверских система различитих типова говори о томе да одређена добра пракса која се примјењује приликом пројектовања корисничког интерфејса за десктоп апликације обично не може да се директно примијени на мобилне или веб апликације. Типичан примјер је коришћење десног тастера миша за приказ додатних функционалности који је уобичајен код десктоп апликација, али се и поред тога што га је могуће имплементирати и у веб и мобилним апликацијама, у развоју ових типова софтверских система се веома ријетко појављује. Преовлађујући разлог за другачији начин пројектовања корисничког интерфејса различитих типова софтверских система тиче се технолошких ограничења. Ако се посматра претходно наведени примјер коришћења десног тастера миша, у веб апликацијама се додатне опције такође приказују, али обично нису везане за саму веб апликацију која се користи, већ на веб претраживач (*web browser*) у коме се апликација приказује. Брз развој информационих технологија омогућио је да се оваква ограничења временом превазиђу, али разлике у пројектовању корисничког интерфејса и даље постоје, и то

не као посљедица технолошких ограничења, већ превасходно због стечених навика корисника, које се мијењају много теже и спорије од развоја технологије.

У наставку ће бити дат преглед специфичности корисничких интерфејса различитих типова софтверских система (десктоп, веб и софтверске системе за мобилне уређаје) за које се сматра да морају бити узете у обзир приликом пројектовања корисничког интерфејса, како би се на одговарајући начин одговорило на захтјеве корисника, смањила крива учења и повећала његова ефикасност. Уочена су три основна узрока постојања специфичности корисничких интерфејса различитих типова софтверских система: димензије екранске форме различити распоред компоненти корисничког интерфејса, различити начини коришћења улазних уређаја (миш, тастатура, екран осјетљив на додир) и различити начин коришћења компоненти корисничког интерфејса.

1. Димензије екранске форме и распоред компоненти корисничког интерфејса

Пројектанти корисничког интерфејса десктоп апликација теже томе да обезбиједе да се кориснички интерфејс приказује на идентичан начин на свим рачунарима на којима је апликација инсталирана. Као посљедица тога, интерфејс се најчешће пројектује тако да величина екранских форми не превазилази стандардну величину екрана, тј. стандардну резолуцију, па је доста дуго времена ова величина износила максимално 800 x 600 пиксела. Поред тога, пракса код пројектовања корисничког интерфејса за десктоп апликације је да се све графичке компоненте распореде у оквиру екранске форме на тај начин да буду видљиве и доступне кориснику без потребе за коришћењем вертикалних или хоризонталних клизача (*scroll bar*). Насупрот оваквом начину пројектовања, код корисничког интерфејса веб апликација оваква пракса није уобичајена. Наиме, појавом Интернета, веб странице

су доминантно биле презентационо оријентисане, тј. садржале су статички садржај који се на захтјев корисника приказивао у оквиру веб претраживача. Садржај ових страница великим дијелом чинио је текст, чија је дужина обично превазилазила висину видљивог дијела странице, па се подразумијевала употреба вертикалних клизача. На тај начин је створена навика код корисника који приступају веб садржају, па се приликом пројектовања корисничког интерфејса веб апликација препоручује ослањање на ове стечене навике корисника, тј. допушта се распоређивање графичких компоненти по вертикали без обзира на величину видљивог дијела веб претраживача. Што се тиче ширине пројектованог веб интерфејса, препоручује се да се она ограничи на одређену стандардну димензију, умјесто да се допушта да се екранска форма распростире од једне до друге ивице претраживача по хоризонтали. Разлог томе је опет стечена навика корисника који су навикли на преовлађујући текстуални садржај. Наиме, ова пракса је преузета из штампарства, које је своју добру праксу градило вјековима, а која код слагања и преламања текста предност даје организовању текста у више мањих колона насупрот једне широке колоне. Читаоцу је много теже да се фокусира на следећи ред уколико је колона прешироко [Readability]. Поред саме навике корисника, велико ограничење код веб корисничког интерфејса је зависност приказа странице од конкретног веб претраживача којег користи корисник. Исти веб садржај различити веб претраживачи приказују на различити начин, па се пројектант не може ослонити на апсолутне димензије и распоред компоненти корисничког интерфејса, већ је пракса да се компоненте пројектују релативно у односу на остали садржај.

Ограничење у димензији екранске форме код корисничког интерфејса десктоп апликација, као и пракса да се све графичке компоненте распореде у оквиру екранске форме на тај начин да буду видљиве и доступне кориснику без потребе за коришћењем вертикалних или хоризонталних клизача, за посљедицу има често

коришћење панела у облику картица (*tabbed panes*) односно коришћење Parallel Panels патерна, као и приступ одређеним функционалностима коришћењем дијалога, што се много ређе користи у веб апликацијама.

Што се тиче мобилних апликација, може се рећи да су, у контексту ограничења величине екранске форме, много сличније десктоп апликацијама. Наиме, сама величина мобилног уређаја одређује и величину екранских форми. Димензија уређаја утиче и на распоређивање графичких компоненти, па се нпр. Master/Detail патерн обично реализује кроз више екранских форми или у комбинацији са Parallel Panels патерном.

2. Коришћење улазних уређаја (миш, тастатура, екран осјетљив на додир)

Устаљена пракса приликом развоја корисничког интерфејса десктоп апликација је омогућавање пречица за позив често коришћених функционалности комбинацијом различитих тастера на тастатури. Ако се говори о *CRUD* операцијама, корисници су стекли навику да операцију чувања измјена позивају комбинацијом тастера *CTRL* и *S,* операцију креирања комбинацијом *CTRL* и *N*, претраживања *CTRL* и *F*, операцију брисања притиском на тастер *DELETE.* Такође, кориснику се омогућава и да приступа ставкама главног менија апликације на сличан начин. Тастер табулатор се обично користи за пребацивање фокуса са једне на другу компоненту унутар екранске форме. Главни разлог за прављење пречица је смањење потребе за коришћењем миша, већ се омогућава кориснику да, уколико врши унос података преко тастатуре, читав процес заврши коришћењем тастатуре. Иако је у веб апликацијама могуће имплементирати коришћење пречица, њихово коришћење се не препоручује, прије свега из разлога што корисници нису навикнути да их користе. Наиме, уобичајено понашање претходно наведених пречица се односи на веб

претраживач, а не на конкретну веб апликацију. Примјер за то је комбинација тастера *CTRL* и *N* која, без обзира на то што је веб апликација активна у претраживачу, по правилу отвара нови прозор претраживача. Управо из разлога што имплементација пречица у веб апликацији захтијева редефинисање и прекривање понашања веб претраживача, то представља веома захтјеван задатак, нарочито када се на све поменуто дода чињеница да су ове операције директно зависне од самог претраживача који се користи, па није извјесно да ли ће имплементиране пречице функционисати у новијим верзијама истог, или у другим претраживачима који су на располагању кориснику. Ако се посматрају мобилне апликације, обзиром да данас доминирају уређаји који користе екране осјетљиве на додир, па се тастатура у класичном облику практично и не користи, поменуте пречице нису пракса приликом пројектовања корисничког интерфејса мобилних апликација.

Слично пречицама, код корисничког интерфејса десктоп апликација могуће је имплементирати различито понашање за тастере *ENTER* и *BACKSPACE.* Међутим, у контексту развоја корисничког интерфејса веб апликација, ова два тастера су такође веома специфична. Наиме, тастер *ENTER* се у веб апликацијама подразумијевано користи за слање захтјева веб серверу. Ово понашање је до појаве *AJAX* технологије било прихватљиво, јер су форме пројектоване тако да шаљу тачно одређени захтјев серверу [57], али данашње веб апликације, које са једне екранске форме могу позивати различите функционалности сервера, доводе у питање употребу *ENTER* тастера, јер корисник не зна која ће од постојећих функционалности бити позвана. Тастер *BACKSPACE* је, слично проблему са пречицама, у веб апликацијама подразумијевано везан за дугме *BACK* веб претраживача, које враћа корисника на претходно учитану страницу. Како је коришћење овог тастера директно зависно од веб претраживача,

[57] Овакво понашање је било могуће промијенити коришћењем *Java script* језика, али је подразумијевано веб форма била пројектована за извршење једног, унапред одређеног задатка.

препорука је да се избјегава његово коришћење у веб апликацијама (осим за едитовање текста унутар одређене графичке компоненте).

Поред поменутих разлика у коришћењу улазних уређаја, прије свега тастатуре, потребно је поменути и разлике које постоје у коришћењу миша. Десни тастер миша се у десктоп апликацијама користи за приказ додатних функционалности које се не користе често, па директно видљиве кориснику на екранској форми. Коришћење десног тастера је уобичајено код десктоп апликација, али се и поред тога што га је могуће имплементирати и у веб и мобилним апликацијама, у развоју ових типова софтверских система се веома ријетко појављује. Наиме, коришћењем десног тастера миша у веб апликацијама се додатне функционалности такође приказују, али обично нису везане за саму веб апликацију која се користи, већ на веб претраживач у коме се апликација приказује. Неки оквири за развој веб апликација садрже библиотеке графичких компоненти, нпр. *PrimeFaces* [PrimeFaces] библиотека која важи за најпопуларнију *JSF 2.0* [JSF] библиотеку компоненти, омогућава коришћење десног тастера миша за приказ додатних функционалности и у веб апликацијама. Ипак, коришћење десног тастера миша за приказ додатних функционалности и у веб апликацијама се не препоручује, прије свега због стечених навика корисника. Изузетак могу бити веб апликације које су пројектоване као замјена постојећих десктоп апликација на које су корисници навикнути, а које су користиле поменуте функционалности. Што се тиче коришћења лијевог тастера миша, такође се може говорити о различитој интерпретацији. Наиме, код веб апликација један клик лијевог тастера миша подразумијевано се користи за позив операције, док се у десктоп апликацијама за позив операције често користи дупли клик лијевим тастером миша.

Веома честа пракса у десктоп апликацијама је коришћење миша за превлачење, односно *drag and drop.* Овај начин се ријетко користи у веб

апликацијама, иако га је коришћењем савремених оквира за развој веб апликација могуће имплементирати.

У погледу коришћења улазних уређаја и њиховог утицаја на развој различитих типова софтверских система, значајно је размотрити и екране осјетљиве на додир. Екрани осјетљиви на додир се превасходно користе на мобилним уређајима, па су апликације развијене за ове уређаје у потпуности подређене њиховом коришћењу у интеракцији са системом. Међутим, данас се екрани осјетљиви на додир инсталирају и на десктоп и лаптоп – преносним рачунарима, који се обично користе за извршавање и приступ десктоп и веб апликацијама. Ипак, и поред постојања препорука (*guidelines*) [TouchDesign] за развој десктоп и веб апликација за уређаје који посједују екран осјетљив на додир, још увијек се не могу примијетити значајнији утицаји на промјену устаљене праксе пројектовања корисничког интерфејса за ове типове апликација. Ипак, у будућности се могу очекивати промјене и у овом домену, па се неки утицаји на будући развој већ могу наслутити, а који су данас доминантни у изради мобилних апликација. Типичан примјер је коришћење поља за унос података. Наиме, компоненте за унос нумеричних података у мобилним апликацијама се, за разлику од десктоп и веб апликација код којих се очекује унос са тастатуре, омогућавају одабир бројева (*spinner* компоненте). Такође, навигација кроз елементе који су приказани у листи или табеларно, која се у десктоп и веб апликацијама врши коришћењем стрелица на тастатури, или коришћењем клизача и одабира лијевим тастером миша, у мобилним апликацијама се врши покретом прста који подсјећа на листање књиге (*sliding* компоненте). Алат за аутоматизацију процеса креирања корисничког интерфејса за десктоп и веб апликације треба да узме у обзир и омогући коришћење и ових компоненти.

3. Компоненте корисничког интерфејса

Сваки тип софтверског система, у зависности од одабране имплементационе технологије, омогућава избор различитих компоненти корисничког интерфејса. Бројне графичке компоненте могу се користити за развој корисничког интерфејса било ког типа софтверског система, али су неке од њих у већој или мањој мјери заступљене у развоју, обично у зависности од навика корисника или технолошких ограничења. Типичан примјер су линије са алатима (*toolbar* компоненте) и линије за приказ статуса (*status bar* компоненте) које се веома често користе у изради корисничког интерфејса десктоп апликација, а и поред тога што постоје у библиотекама графичких компоненти за развој мобилних и веб апликација, за развој ових типова апликација се ријетко користе.

Неизоставна компоненте приликом развоја корисничког интерфејса било ког типа софтверског система је табела. Међутим, начин на који се обично користи у различитим типовима софтверских система се разликује. Наиме, код десктоп апликација табеле се користе како за приказ података, тако и за унос и измјену података. У веб и мобилним апликацијама се табеле користе превасходно за приказ података, а много ређе за директан унос и измјену података, иако саме графичке компоненте обично омогућавају и ове операције[58]. Много чешћа пракса је да се за унос и измјену података пројектује засебна екранска форма са пољима за унос (Form патерн) и операцијама, па се након уноса или измјене поново приказује табела са ажурираним садржајем.

[58] Типичан примјер је Google Spreadsheets [GoogleSheets] веб апликација која омогућава коришћење основних функционалности програма MS Excel [Excel].

Као што је већ поменуто, због технолошких ограничења се у веб и мобилним апликацијама употреба дијалога за извршавање одређених функционалности своди на најмању мјеру, већ се дијалози претежно користе за приказ порука о статусу извршених операција кориснику. Са друге стране, у десктоп апликацијама се дијалози често користе како за приказ порука, тако и за извршавање одређених функционалности.

Поред поменутих компоненти, занимљиво је осврнути се и на компоненте које служе за позив извршења функционалности као што су дугмад (*button* компонента) и линкови (*hyperlink* компонента). Наиме, раније су се линкови искључиво користили у контексту веб апликација, али су касније почели да се користе и у десктоп апликацијама. Дугме је коришћено и у десктоп и у веб апликацијама. И једна и друга компонента могу да се користе за позив извршења одрешене функционалности, али се препоручује да се линкови користе за навигацију кроз систем (прелазак са једне на другу екранску форму или веб страницу), док би дугме требало да буде коришћено за позив системских операција.

6. Шаблони корисничког интерфејса

У претходном поглављу направљен је преглед основних принципа пројектовања корисничког интерфејса, концепата који се користе у пројектовању као и препорука и добре праксе, у виду патерна корисничког интерфејса, којих се треба придржавати приликом пројектовања корисничког интерфејса независно од типа софтверског система. Такође, направљен је преглед специфичности различитих типова софтверских система у контексту пројектовања корисничког интерфејса. Све уочене карактеристике треба узети у обзир приликом пројектовања корисничког интерфејса, како би се на одговарајући начин одговорило на захтјеве корисника, смањила крива учења и повећала његова ефикасност. Међутим, важно је примијетити да коришћење основних принципа за пројектовање корисничког интерфејса и патерна корисничког интерфејса не осигурава аутоматски сагласност добијеног корисничког интерфејса са постављеним софтверским захтјевима. Алат за аутоматизацију процеса креирања корисничког интерфејса треба да обезбиједи директно пресликавање дефинисане улазне спецификације, засноване на случајевима коришћења који представљају захтјеве корисника, са једне стране и будућег корисничког интерфејса са друге стране, који мора уважити основне принципе пројектовања укључујући и патерне корисничког интерфејса, као и специфичности одабране технологије, односно типа софтверског система. Зато моделом за спецификацију софтверских захтјева, који представља улаз у процес генерисања корисничког интерфејса, треба омогућити успостављање формалних веза са

корисничким интерфејсом. Како се за један софтверски захтјев кориснички интефрејс може реализовати на више начина, при томе задржавајући специфициране функционалности, у наставку ће бити размотрене везе које постоје између случајева коришћења и елемената корисничког интерфејса, са циљем да се дође до скупа шаблона корисничког интерфејса који се аутоматски могу повазати са одређеним корисничким захтјевом[59]. Ове везе ће бити формализоване у моделу за спецификацију, па ће тиме бити омогучено да се једноставном и брзом промјеном шаблона у моделу изгенерише потпуно другачији кориснички интерфејс који задовољава специфициране функционалности. Сви дефинисани шаблони морају уважити принципе, препоруке и уочене специфичности у пројектовању корисничког интерфејса које су описане у претходном поглављу. Дефинисани шаблони прије свега треба да обухвате опсег који је дефинисан овим истраживањем, а то је кориснички интерфејс пословних апликација који обезбјеђује извршење *CRUD* операција, а које је могуће примијенити за различите типове софтверских система.

Елементи из модела случајева коришћења значајни за пројектовање корисничког интерфејса

Разматрајући проблем пројектовања корисничког интерфејса, уочени су различити извори зависности између спецификације случајева коришћења и корисничког интерфејса који се пројектује [Antovic10] [Antovic12]. Како је више пута истакнута потреба за конзистентношћу случајева коришћења са моделом података,

[59] Потреба за дефинисањем шаблона корисничког интерфејса, као и основни принципи за њихово пројектовање изложени су у оквиру мог Магистарског рада [Antovic10], али ће због значаја и утицаја који имају на резултате истраживања које представљам у Докторској дисертацији на неколико мијеста у овом поглављу бити поново приказани, уз све измјене и допуне које су у међувремену настале.

важан утицај на пројектовани кориснички интерфејс имаће и различите компоненте модела података. Заправо, сваки случај коришћења односи се на један или више ентитета из модела података. У наставку ће бити приказане везе које су уочене између елемената спецификације случајева коришћења и елемената корисничког интерфејса пројектованог на основу одређене спецификације. Такође, биће приказане и везе између елемената модела података (ентитети, атрибути и везе) са пројектованим корисничким интерфејсом.

У литератури се често може наћи препорука да информације везане за кориснички интерфејс треба изоставити из спецификације случајева коришћења. Као једну од најчешћих грешака у спецификацији случајева коришћења аутори управо наводе постојање детаља везаних за кориснички интерфејс. Ова препорука долази из оправданог страха да детаљи везани за кориснички интерфејс могу одвући пажњу заинтересованих страна приликом специфицирања софтверских захтјева са суштине проблема који се анализира.

Међу информацијама везаним за кориснички интерфејс унутар спецификације случајева коришћења често се наводе графичке компоненте које треба да буду приказане приликом извршавања случаја коришћења. То се односи на прозоре, тј. форме, текстуална поља, табеле, падајуће листе, меније, дугмад, дакле врсте компоненти које ће бити коришћене, затим распоред самих компоненти на екрану, као и динамичка својства корисничког интерфејса. Што се више оваквих информација укључи у спецификацију случајева коришћења, то спецификација престаје да има функцију везану за прикупљање и анализу захтјева и постаје један облик корисничког упутства [Cockburn00].

Информације везане за кориснички интерфејс нису од значаја када се описује циљ и намјера корисника, док су циљ и намјера корисника од суштинске важности за

случајеве коришћења. Међутим, сама чињеница да се информације о корисничком интерфесју тако често јављају унутар спецификације случајева коришћења, свједочи о нужности успостављања јасних релација између пројектовања корисничког интерфејса и интеракције корисника и система описане случајевима коришћења [Sinnig05].

Анализирајући спецификације случајева коришћења, као и кориснички интерфејс који је пројектован за ове спецификације, уочене су одређене везе које се између њих могу успоставити [Antovic10].

Прва веза која се може уочити односи се на редослијед извршења корака у основном сценарију случаја коришћења, и то оних корака које извршава актор. Наиме, кориснички интерфејс би требало да омогући интеракцију између корисника и система, и то на начин описан спецификацијом случајева коришћења. Када се то узме у обзир, природно је да ће распоред компоненти на корисничком интерфејсу бити пројектован на тај начин да кориснику омогући логички слијед извршења операција који одговара жељеном опису исказаном кроз спецификацију случаја коришћења.

Друга веза између елемената корисничког интерфејса и спецификације случајева коришћења односи се на сам тип акције у кораку основног сценарија који извршава актор. У претходном тексту утврђени су различити типови акција које извршава актор као и акција које извршава систем. Акције актора се односе на припрему улазних података (гдје су уочена два типа акција: унос и одабир), као и на слање захтјева и података систему (трећи тип акције актора).

У односу на тип акције у кораку сценарија, може се направити разлика у типу компоненти које ће бити коришћене на корисничком интерфејсу. У наредној табели

биће приказане неке могуће графичке компоненте које одговарају различитим типовима акција актора.

Табела 5: Графичке компоненте које одговарају различитим типовима акција сценарија

<table>
<tr><th>Тип акције</th><th>Врста графичких компоненти</th><th>Пратеће компоненте</th></tr>
<tr><td rowspan="4">Унос</td><td>текстуално поље</td><td rowspan="4">лабела, tool-tip...</td></tr>
<tr><td>поље за чекирање</td></tr>
<tr><td>поље за унос датума</td></tr>
<tr><td>табела</td></tr>
<tr><td rowspan="5">Одабир</td><td>падајућа листа</td><td rowspan="5">лабела, tool-tip...</td></tr>
<tr><td>табела</td></tr>
<tr><td>радио дугмад</td></tr>
<tr><td>листа</td></tr>
<tr><td>стабло</td></tr>
<tr><td rowspan="3">Слање захтјева и података систему</td><td>дугме</td><td rowspan="3">догађаји и ослушкивачи догађаја</td></tr>
<tr><td>мени или ставка менија</td></tr>
<tr><td>догађај дефинисан над неком компонентом</td></tr>
</table>

Тип акције одабир специфичан је и по додатној семантици која се односи на податке који се морају наћи, тј. који ће бити учитани на корисничком интерфејсу приликом извршења случаја коришћења. Наиме, ако се оваква акција реализује коришћењем падајуће листе, најчешће ће ова листа бити напуњена подацима који се учитавају прије него што актор почне извршење корака који садржи ову акцију, обично приликом отварања прозора намијењеног за реализацију тог случаја коришћења.

Акције актора које се односе на припрему улазних података (унос и одабир), поред графичких компоненти које ће прихватати улазне податке, подразумијева и

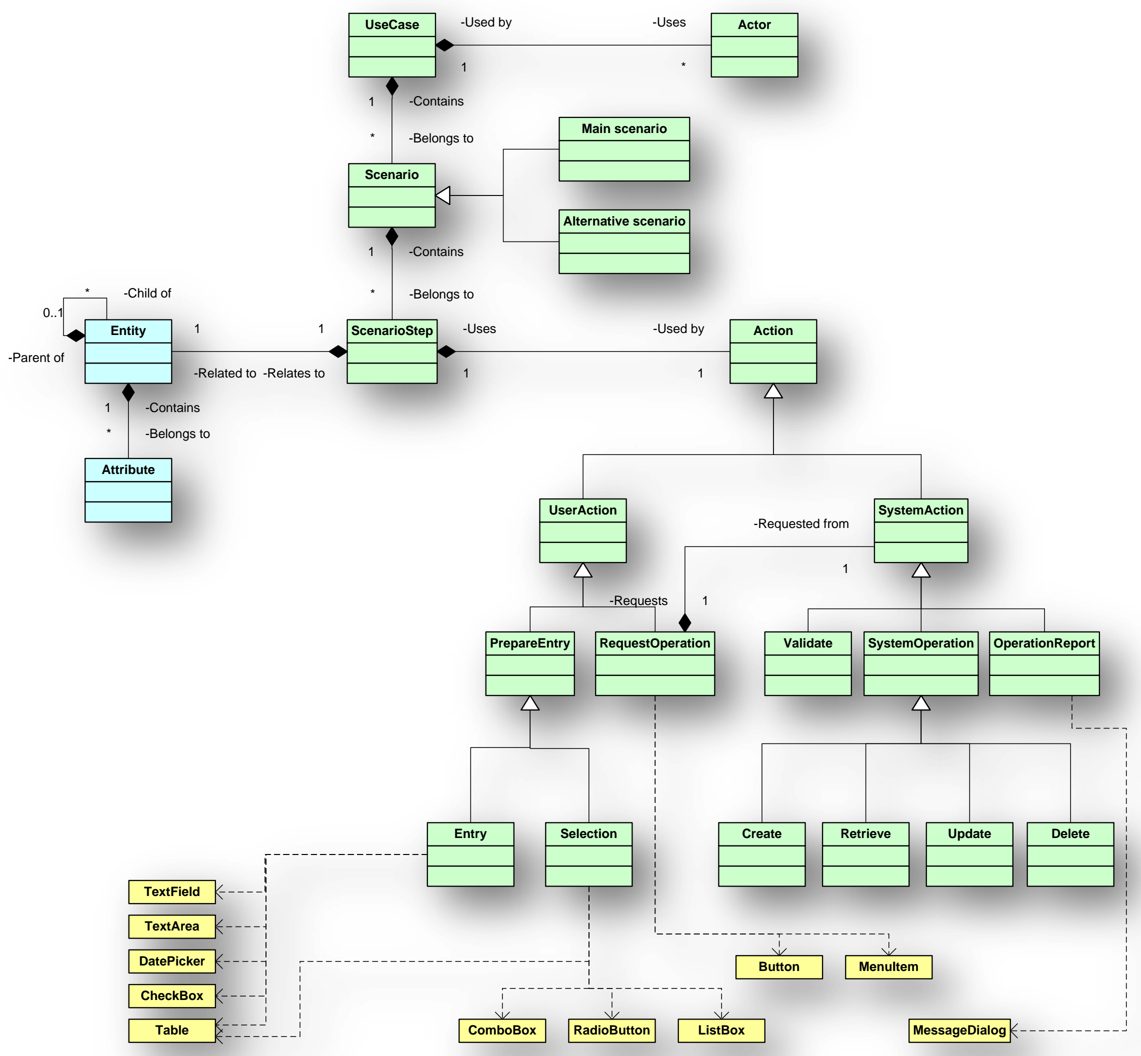
графичке компоненте које ће кориснику описати значење сваке компоненте. Најчешће се за ову намјену користе тзв. лабеле, а уз њих могу се користити и неке друге компоненте (као нпр. *tool tip* компонента).

Тип акције слање захтјева и података систему поред компоненте којом ће се омогућити извршење, подразумијева и дефинисање догађаја над одабраном компонентом, услијед којег акција треба да се изврши. Тако нпр. ако се за реализацију ове акције користи дугме, над дугметом мора бити дефинисан одговарајући догађај (клик мишем, дупли клик, притисак одређеног тастера на тастатури и сл.), и за догађај треба имплементирати позив тражене функционалности система (најчешће извршење одређене системске операције). Натпис на дугмету треба да одговара, тј. да одражава логику операције која се захтијева од система.

Што се тиче акција система, од три типа акција које извршава систем (валидација података, извршење операције која мијења унутрашње стање система, приказивање резултата извршења операције актору), акција типа приказивање резултата извршења операције актору, има директне везе са корисничким интерфејсом апликације. У зависности од врсте системске операције о чијим се резултатима извршења извјештава актор, зависиће и приказ извјештаја актору. Наиме, ако се узму у обзир *CRUD* операције, *Retreave* (*read*) операција ће, ако је успјешно извршена, подразумијевати попуњавање компоненти на корисничком интерфејсу апликације подацима који су резултат извршене операције. Ако операција није извршена успјешно, корисник треба да буде обавијештен о грешци која је наступила, и разлозима због којих је до грешке дошло. Ово обавјештење се најчешће приказује коришћењем тзв. дијалога за поруке (*message dialog*), дијалога који су посебно направљени за ову намјену и обично су поред текстуалне поруке и описа праћени визуелним и аудитивним сигналима који указују на врсту поруке (грешка, информација, питање итд.). Овакви дијалози користе се и код осталих типова

операција (*Create, Update, Delete*), и то тако што корисника обавјештавају и о успјешном и о неуспјешном извршењу операције.

Веза елемената из спецификације случајева коришћења са елементима корисничког интерфејса графички је приказана на дијаграму који слиједи.



Слика 105: Веза између елемената спецификације случајева коришћења и корисничког интерфејса

У трећем поглављу објашњен је значај коришћења тзв. <u>именица-глагол-именица</u> структуре (*Noun-Verb-Noun*) односно <u>субјекат-предикат-објекат</u> приликом

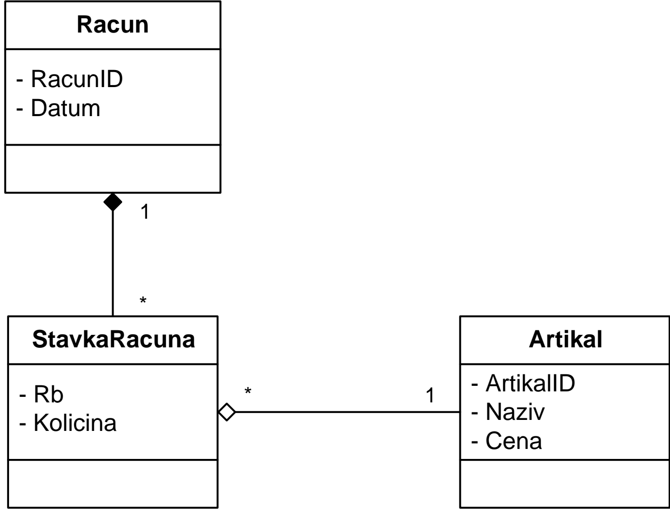
специфицирања корака у сценарију случаја коришћења. При томе прва именица означава актора или систем као субјекат у реченици (у зависности од тога да ли се ради о акцији коју извршава актор или систем), глагол представља предикат у реченици и означава радњу тј. акцију која се извршава, док друга именица означава објекат над којим се врши радња а који представља ентитет или атрибут ентитета из модела података. Ова веза се може уочити на претходном дијаграму. У наставку ће бити приказане везе између елемената модела података (ентитети, атрибути и везе) са пројектованим корисничким интерфејсом.

Анализирајући моделе података за које су везани случајеви коришћења, као и кориснички интерфејс који је пројектован за ове моделе, уочене су одређене везе које се могу успоставити између њих. Приликом посматрања модела података, разматран је утицај сваког од основних елемената модела података (ентитети, њихови атрибути и везе између ентитета) на особине пројектованог корисничког интерфејса.

Особине елемената модела података у контексту пројектовања корисничког интерфејса не треба посматрати независно од спецификације случајева коришћења. Наиме, модел података се пројектује са циљем да се обезбиједи трајно чување података, па поред информација које су од значаја за корисника система, могу да садрже и додатне информације које су укључене у модел како би се осигурала конзистентност података, или испратиле промјене које су се над подацима десиле. Тако сваки ентитет у моделу може да садржи информације о посљедњој направљеној измјени над тим ентитетом, датуму креирања ентитета и сл. Све информације које нису обрађене у спецификацији случајева коришћења треба да буду сакривене од корисника, што је у складу са основним принципима пројектовања корисничког интерфејса, јер осим што не представљају корист за корисника приликом извршења случаја коришћења, могу бити узрок забуне.

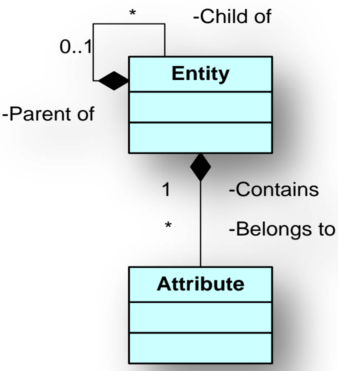
У односу на тип атрибута неког ентитета, такође је могуће успоставити релације са одређеним компонентама корисничког интерфејса. Неке компоненте су у већој или мањој мјери одговарајуће за приказ или унос података одређеног типа. Тако, на примјер, ако се ради о логичком типу податама (*boolean*) ријетко ће се кориснику приказивати текстуално поље у које корисник треба да уноси вриједност (*true/false*, *yes/no*, 0/1 и сл.), већ ће најчешће бити приказана нека компонента која је пројектована за ову намјену (поље за чекирање, група радио дугмади, падајућа листа са понуђеним вриједностима и сл.). За текстуалне податке, најчешће се користи текстуално поље (са једном или више линија текста, у зависности од дужине податка), док за атрибуте који представљају датум постоје компоненте развијене управо за ту намјену (различите врсте компоненти за унос или одабир датума (*date-picker*)).

Везе у моделу података такође имају значајну улогу у пројектовању корисничког интерфејса. Веза 1-* (један прма више) подразумијева да ће за појављивање једног типа ентитета постојати више повезаних појављивања ентитета са стране *. Ако се као примјер узме однос између ентитета рачун и ставка рачуна, и ако се у случају коришћења за унос рачуна подразумијева и унос ставки рачуна, најчешће ће кориснички интерфејс за унос ентитета са стране *, у овом случају ставке рачуна, бити одвојен и приказан као под-форма у односу на основну форму која се креира за унос података о рачуну, као што предвиђа Master/Detail патерн корисничког интерфејса. Подформа може бити приказана у истом прозору, а може се приказати и у одвојеном прозору, с тим што се мора направити механизам синхронизације података са основним прозором. О врстама и распореду компоненти на екрану биће више ријечи у наставку, приликом дефинисања различитих шаблона корисничког интерфејса.



Слика 106: Примјер везе 1-* између ентитета рачун и ставка рачуна

Ова подформа треба да омогући унос више појављивања ставки за један рачун, па се за ту намјену може користити графичка компонента која приказује податке у табеларном облику (слично *Spreadsheet* патерну), при чему сваки ред у табели представља различито појављивање ставке рачуна за тренутно активни рачун приказан на главној форми. Овај однос често се назива однос отац-дијете (*parent-child*) при чему отац може имати више дјеце док свако дијете има једног оца.

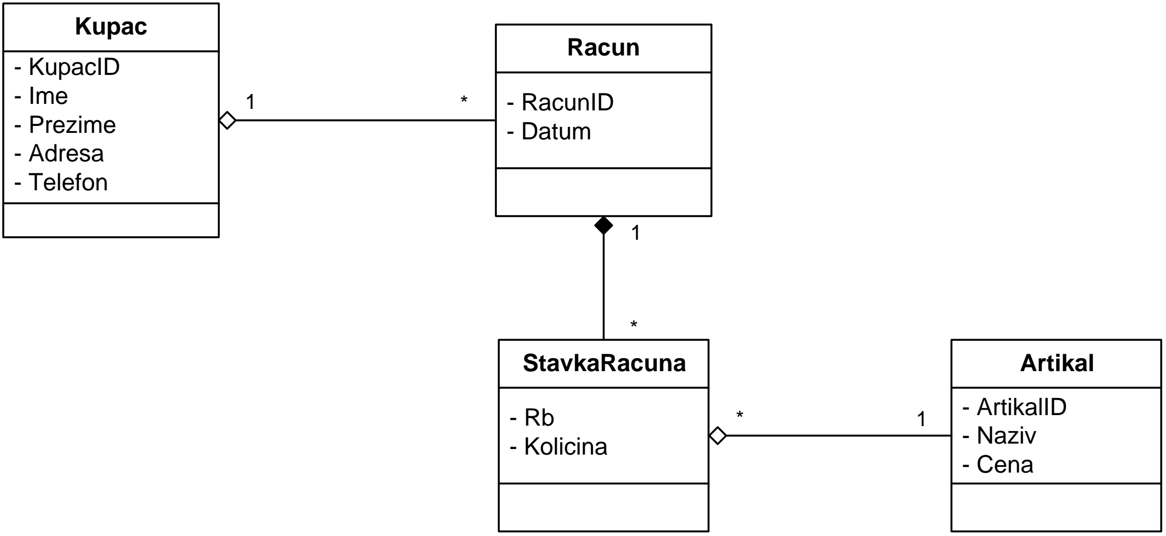


Слика 107: Графички приказ везе *parent-child* између ентитета

Међутим дијете не мора увијек бити приказано у табеларном облику, већ се однос један-више може реализовати и на тај начин да се у под-форми за унос

података о дјетету умјесто табеле могу појавити и текстуална поља, поља са датумом, падајуће листе, које служе за унос података за једно дијете (слично *Form* патерну). Ако се подформа реализује на овај начин, онда је поред графичких компоненти за припрему уноса података за једно дијете, потребно реализовати и неки механизам навигације, како би корисник могао да се креће кроз постојећу листу дјеце, тј. у примјеру рачун-ставка рачуна, да приступа свим ставкама рачуна за активни рачун. Ова навигација најчешће подразумијева дугмад за прелазак на сљедеће, претходно, прво и посљедње појављивање ставке, а могуће је обезбиједити и директан приступ тачно одређеној ставци.

Када се у моделу појављује веза *-1 (више према један, у релационом моделу са стране 1 атрибут се третира као спољни кључ релације са стране *), тада се ентитет са стране 1 најчешће приказује у некој листи (падајућа листа, табела, група радио дугмади). Ако се узме претходни примјер, и допуни на тај начин што се сваки рачун издаје одређеном купцу, а једном купцу може бити издато више рачуна, тада је веза између рачуна и купца веза *-1. Приликом уношења података о рачуну, потребно је одабрати једног од постојећих купаца коме се издаје рачун. Тада ће листа из које се бира купац бити напуњена подацима који припадају ентитету купац, при чему је свака ставка у листи једно конкретно појављивање ентитета купац. Поред овога, често се у спецификацији случаја коришћења предвиђа ситуација када у листи купаца не постоји купац коме се жели издати рачун, или се жели омогућити кориснику да види детаље везане за купца који најчешће нису видљиви приликом приказивања купаца у листи. Зато је корисно омогућити кориснику да на овом мјесту има могућност измјене (додавања нових, измјена постојећих или брисања купаца) или прегледа података о одабраном купцу из листе. На овај начин прави се веза између различитих случајева коришћења (јер је манипулација подацима о купцу најчешће представљена независним случајевима коришћења).



Слика 108: Примјер везе *-1 између ентитета рачун и купац

На основу уочених веза између елемената спецификације случајева коришћења и корисничког интерфејса, као и основних принципа, концепата и патерна корисничког интерфејса, а узимајући у обзир специфичности различитих типова софтверских система, дефинисано је неколико шаблона корисничког интерфејса који могу бити повезани са одређеним случајем коришћења у моделу на основу кога ће бити генерисан кориснички интерфејс. На дефинисање шаблона утицала је и анализа корисничког интерфејса неколико различитих софтверских система[60], код којих су уочене одређене карактеристике које се понављају независно од конкретног домена за који су ови системи пројектовани. Ове карактеристике нису везане само за типове графичких компоненти које се користе (тип компоненти који ће бити коришћен прије свега ће бити одређен типом акције из спецификације случаја коришћења и елементима модела података, на начин објашњен у овом поглављу), већ на начин организовања ових компоненти у корисничком интерфејсу апликације.

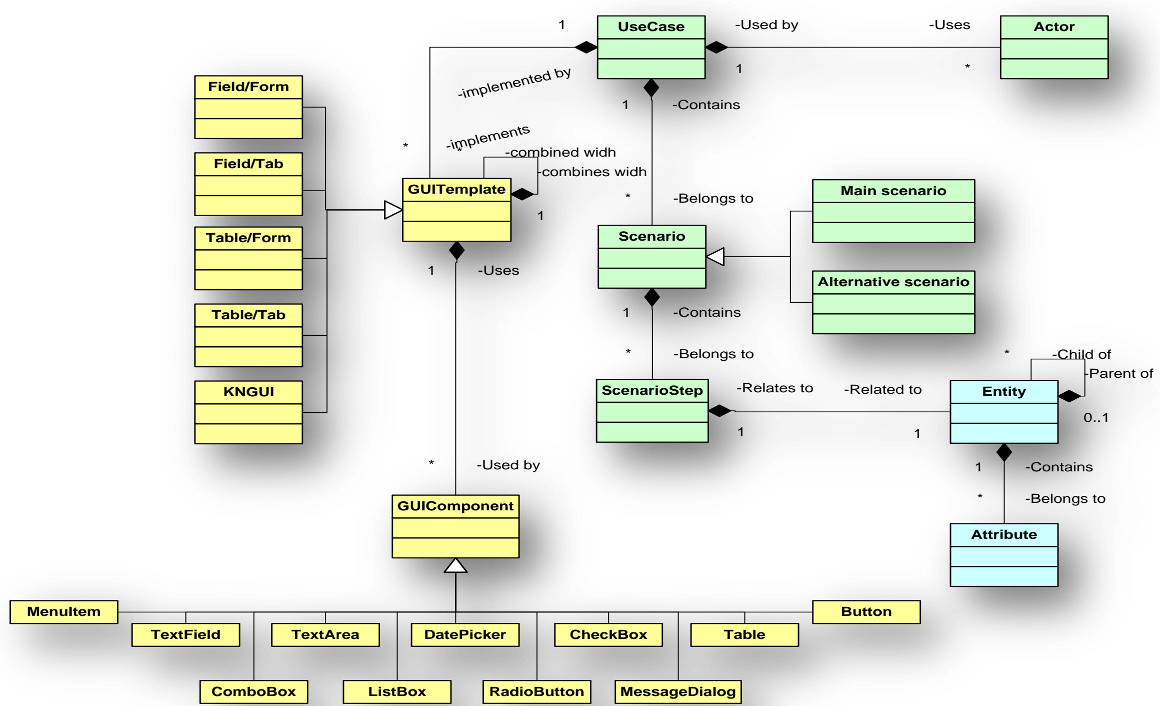
[60] Kostmod 4.0, Менаџмент јавним набавкама, Менаџмент јавним осветљењем – SodiumLight, РФЗО - Листе чекања, Сајтови здравствених установа, Апотеке, Политика – фото модул...

Дефинисани су следећи шаблони корисничког интерфејса:

- ***Field-form* шаблон;**
- ***Field-tab* шаблон;**
- ***Table-form* шаблон;**
- ***Table-tab* шаблон;**
- ***KNGUI* шаблон.**

Један случај коришћења може бити реализован различитим шаблонима, па је коришћењем алата за аутоматско генерисање корисничког интерфејса могуће веома брзо и једноставно добити потпуно другачији кориснички интерфејс за исти случај коришћења простом замјеном одабраног шаблона у моделу. На овај начин, заинтересоване стране приликом спецификације корисничких захтјева могу да испробају различите шаблоне и одлуче се за онај који највише одговара одређеном случају коришћења.

Шаблони су пројектовани тако да се могу комбиновати, па се за један случај коришћења за његове различите дијелове могу користити различити шаблони. У претходном поглављу објашњено је како се веза 1-* између ентитета у моделу података приказује коришћењем једне основне родитељ форме за ентитет на страни 1, док се ентитети са стране * приказују подформама. Свака подформа може бити реализована коришћењем различитог шаблона. На сљедећем дијаграму приказан је однос између различитих елемената случаја коришћења, модела података, шаблона и графичких компоненти које шаблони користе (Слика 109).



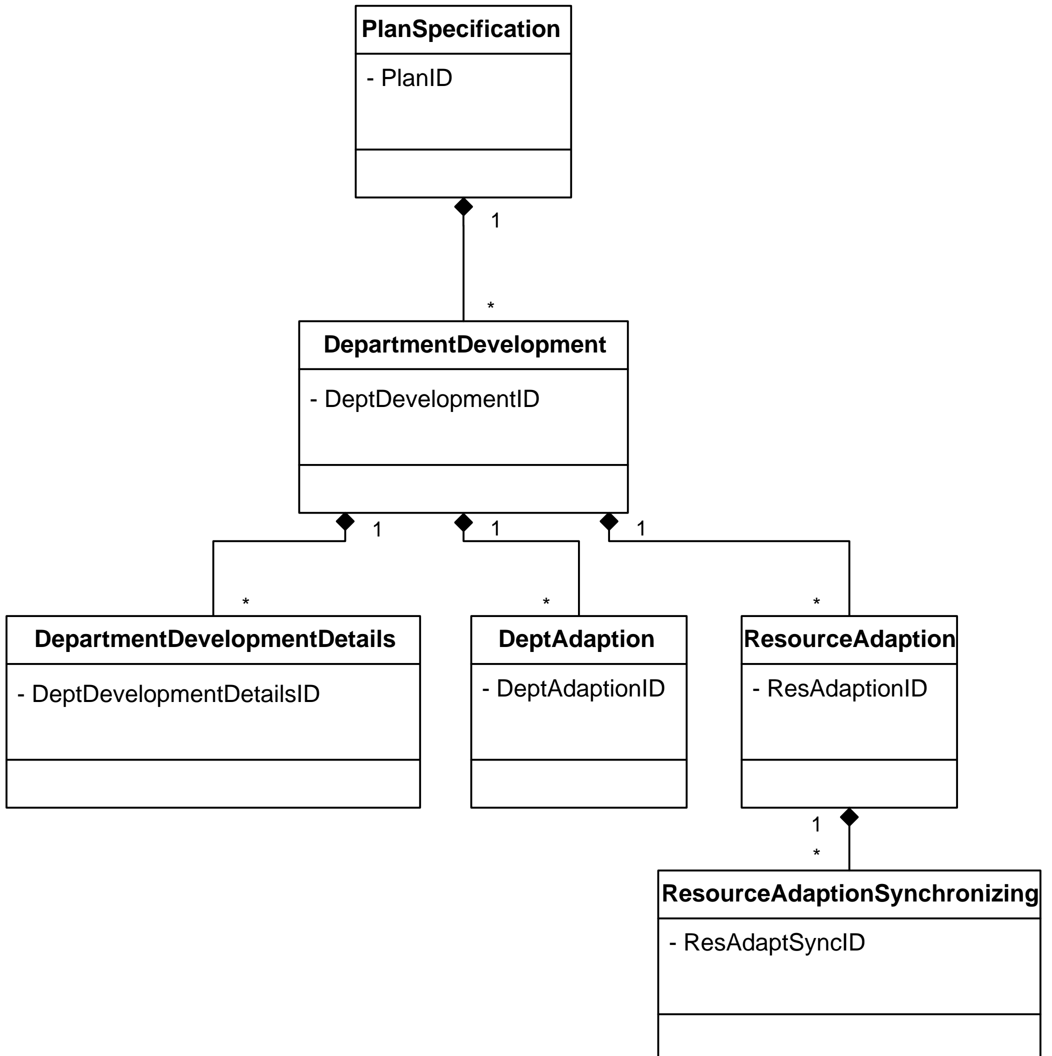
Слика 109: Однос између различитих елемената случаја коришћења, модела података, шаблона и графичких компоненти које шаблони користе

Ако случај коришћења обрађује ентитет у више нивоа дубине (родитељ који има дјецу, а дјеца имају своју дјецу итд.), на сваком нивоу за свако дијете могуће је искористити другачији шаблон. Овдје треба напоменути да се препоручује опрез приликом увођења нових нивоа дубине, јер тада кориснички интерфејс може постати превише компликован и „пренатрпан“, и може довести до забуне код корисника. Наравно, ова препорука се, заправо, не тиче шаблона корисничког интерфејса, већ спецификације случаја коришћења, док је проблем са „пренатрпаношћу“ корисничког интерфејса само посљедица лоше спецификације случаја коришћења. Узрок лоше спецификације не мора бити пројектант, већ је то најчешће нека друга заинтересована страна која захтијева одређену функционалност и инсистира на одређеном начину реализације. Тада пројектант може да укаже на могући проблем, али и да препоручи

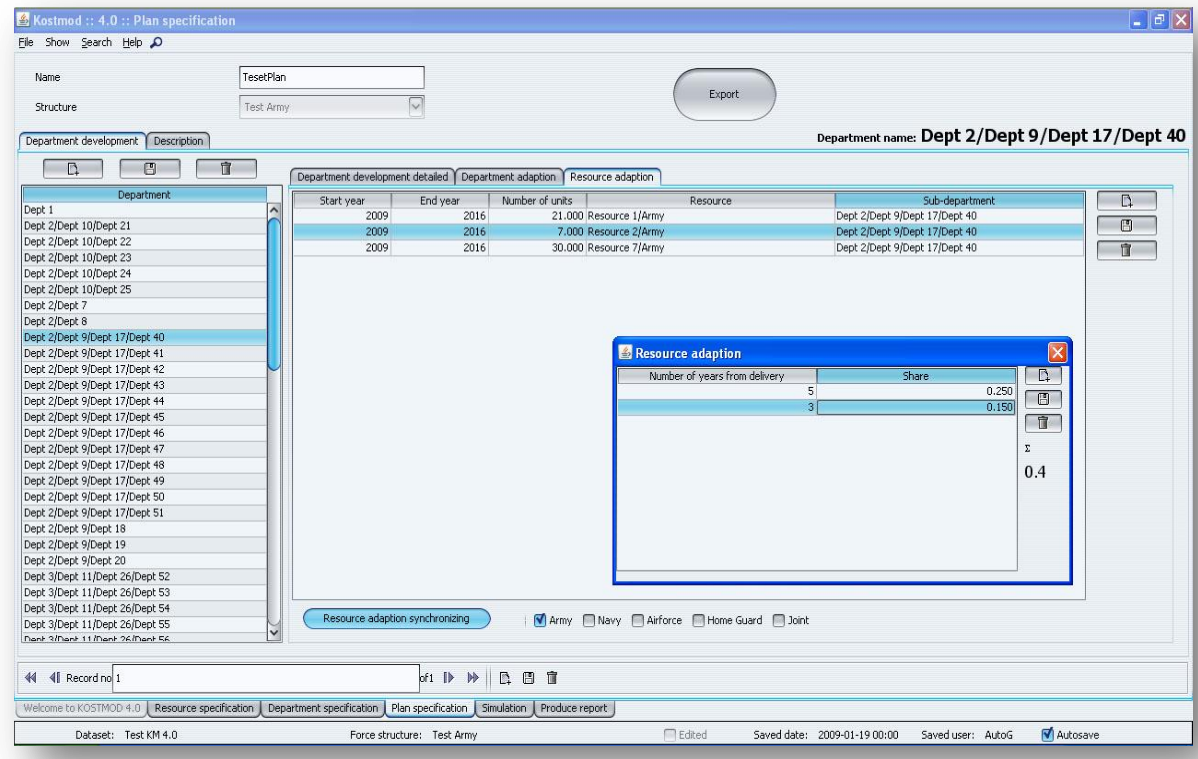
коришћење одређене комбинације шаблона која може да смањи сложеност у корисничком интерфејсу.

У наставку ће бити приказан примјер комбиновања различитих шаблона у више нивоа дубине. На овом примјеру се може видјети једно од рјешења за веома компликован случај коришћења који обрађује ентитет у четири нивоа дубине. Примјер који је приказан дио је корисничког интерфејса развијеног за пројекат Kostmod 4.0, који је имплементиран за потребе Министарства одбране Краљевине Норвешке. Овај дио корисничког интерфејса је комплетно изгенерсан коришћењем *SilabUI* алата. И након упознавања наручиоца са потенцијалним проблемима, наручиоци су остали при своме захтјеву, па је проблем ријешен комбинацијом различитих шаблона корисничког интерфејса (Слика 111). Након неколико година коришћења, наручиоци и корисници система нису имали примједбе на имплементирано рјешење.

На дијаграму је приказан упрошћени модел података, тј. онај дио модела података на који се односи поменути случај коришћења (Слика 110).



Слика 110: Дио модела података за примјер комбиновања различитих шаблона корисничког интерфејса



Слика 111: Примјер комбиновања различитих шаблона корисничког интерфејса са 4 нивоа дубине

У наставку ће детаљније бити објашњене карактеристике сваког уоченог шаблона, њихове везе са спецификацијом случаја коришћења и ентитетима модела података на које се случајеви коришћења односе.

6.1. FIELD-FORM ШАБЛОН

Field-form шаблон замишљен је тако да обезбиједи унос и приказ информација везаних за одговарајући ентитет[61] коришћењем поља (по узору на *Form* патерн), тј. графичких компоненти које служе за унос или одабир појединачних података везаних за атрибуте ентитета који се обрађује, док се дјеца ентитета који се обрађује приказују у засебним прозорима. Свако дијете може бити приказано другачијим шаблоном. У овом шаблону могуће је појављивање следећих компоненти:

- **Лабела**
- **Текстуално поље** (са једном (*text-field*) или више линија за унос (*text-area*))
- **Падајућа листа** (*combo-box*)
- **Дугмад за навигацију** (приказани у облику линије са алатима (*toolbar*) или као обична дугмад)
- **Дугмад за опције** (*insert/update/delete*)
- **Поље за унос датума** (*date-picker*)
- **Поље за чекирање** (*check-box*)
- **Опциона дугмад** (*option button*) која служе за приказивање прозора у којима је садржај везан за дјецу

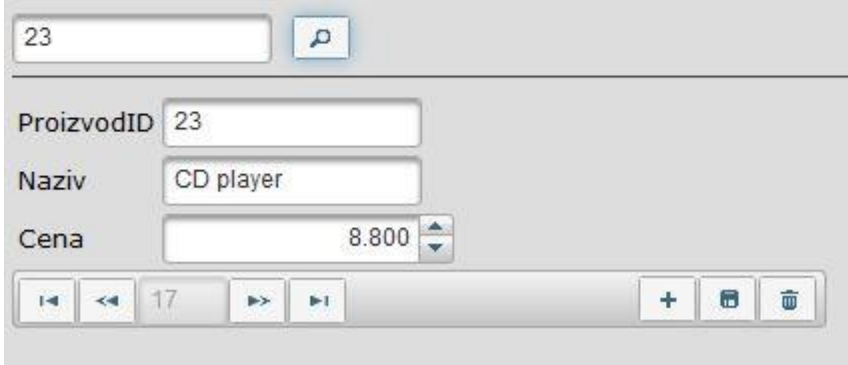
Које од ових компоненти ће бити приказане, зависи од типа акција специфицираних у сценарију случаја коришћења, као и од самог модела података, о чему је било више ријечи у претходном дијелу текста. На следећим сликама се виде

[61] У претходном дијелу текста објашњено је да је могуће комбиновање различитих шаблона на различитим нивоима. Одговарајући ентитет представља онај ентитет који се приказује одређеним шаблоном.

кориснички интерфејси за једноставан случај коришћења који укључује информације везане за један ентитет (без дјеце), а који су реализовани коришћењем *field-form* шаблона и то коришћењем алата за аутоматско генерисање корисничког интерфејса за десктоп апликације у Јава *Swing* технологији (Слика 112) и за веб апликације [62] у *Java Server Faces* (JSF) (библиотека *Prime Faces*) технологији (Слика 113). На слици се може видјети један од начина приказивања навигационих дугмади, и опција за унос, измјену и брисање (линија са алатима у дну прозора).

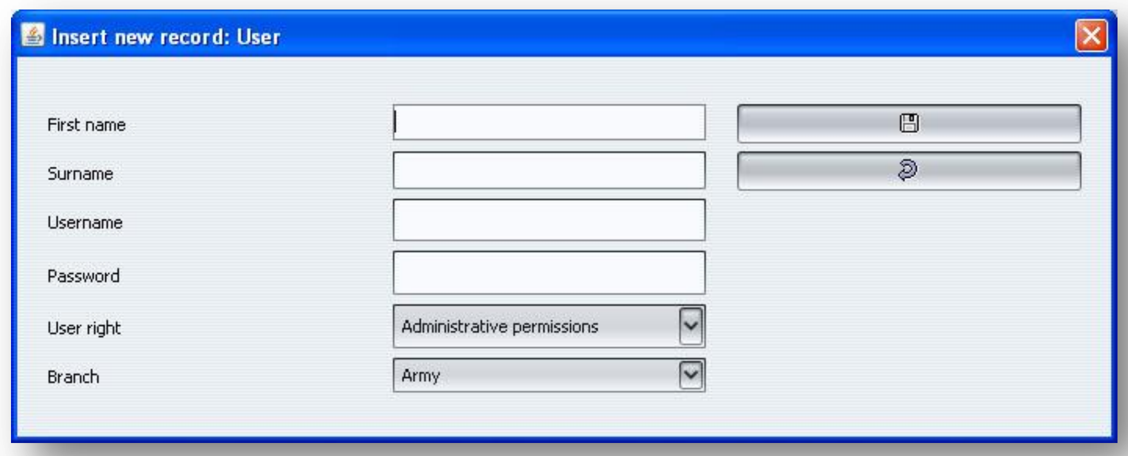
Слика 112: Field-form шаблон: један ентитет без дјеце (*Java Swing* технологија)

[62] За развој генератора корисничког интерфејса веб апликација на основу *SilabUI* модела спроведено је истраживање чији су детаљни резултати изложени у оквиру Мастер рада [Karic14].



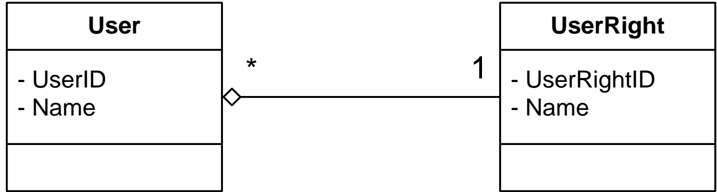
Слика 113: Field-form шаблон: један ентитет без дјеце (Java Server Faces – Prime Faces)

Операција *insert* (унос новог слога) реализује се приказивањем новог прозора у коме је кориснику омогућен унос података и опција за чување односно одустајање (Слика 114).



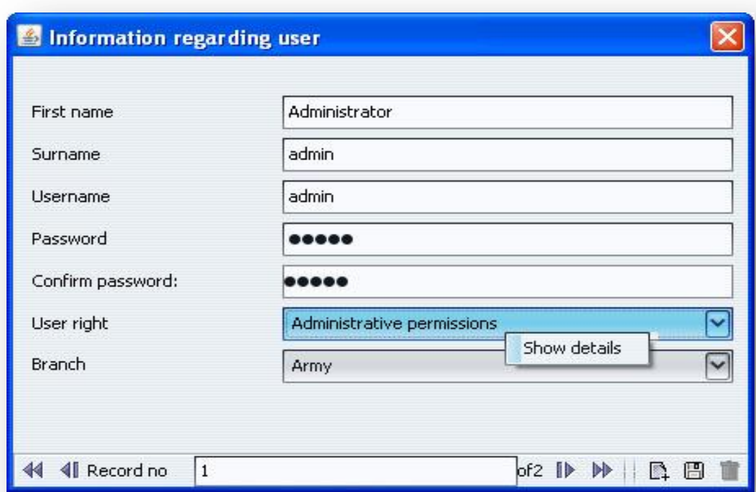
Слика 114: Реализација операције Insert у Field-form шаблону

На овом једноставном примјеру може се уочити да је податак *User right* приказан падајућом листом. Ово је посљедица типа акције одабир из спецификације случаја коришћења, која је везана за везу *-1 из модела података (тј. спољни кључ у релационом моделу). Упрошћен модел података који приказује везу између поменутих ентитета приказан је на сљедећој слици (Слика 115).



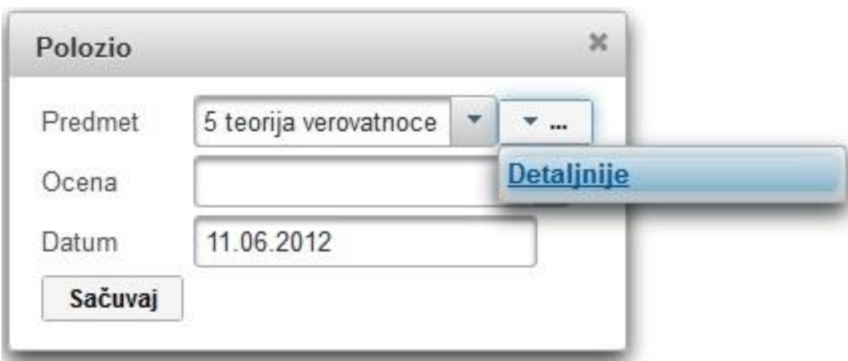
Слика 115: Веза између ентитета која је један од предуслова за коришћење падајуће листе

Ако корисник приликом уноса новог слога или измјене постојећег примијети да у листи не постоји податак који њему треба (или жели да види детаљније информације о неком елементу листе), могуће је омогућити кориснику да на лицу мјеста може да унесе нови елемент у листу, или види детаље везане за одабрани елемент. Ова функционалност позива се кликом десног тастера миша над падајућом листом и одабиром опције *Show details* или коришћењем дугмета који се налази уз падајућу листу. Наравно, претходно је потребно у моделу за генерисање корисничког интерфејса специфицирати жељено понашање. На следећим сликама приказана је реализација ове функционалности у десктоп (Слика 116, Слика 117) и веб апликацијама (Слика 118, Слика 119).

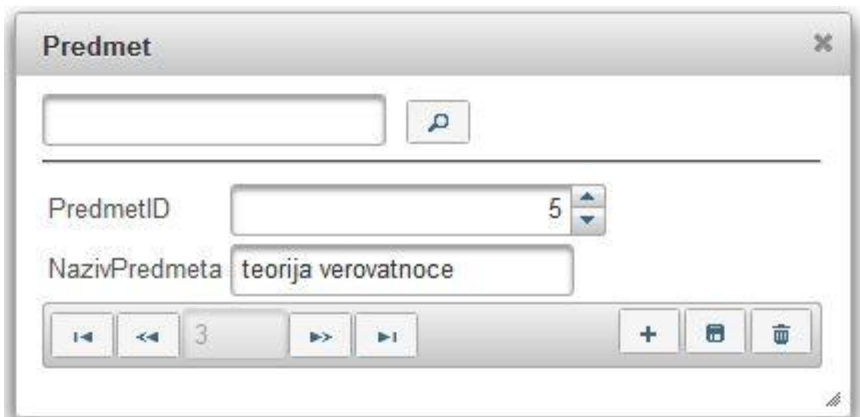


Слика 116: Позив функционалности *Show details* на корисничком интерфејсу десктоп апликације

Слика 117: Реализација функционалности *Show details* над падајућом листом на корисничком интерфејсу десктоп апликације

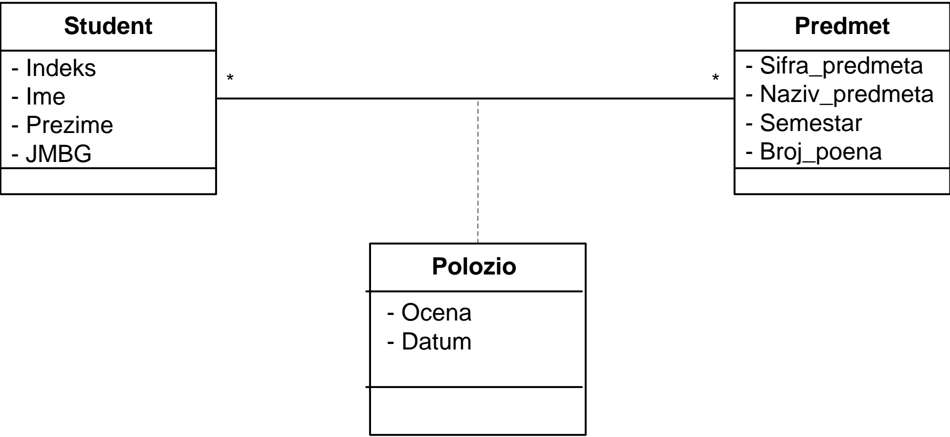


Слика 118: Позив функционалности Show details на корисничком интерфејсу веб апликације



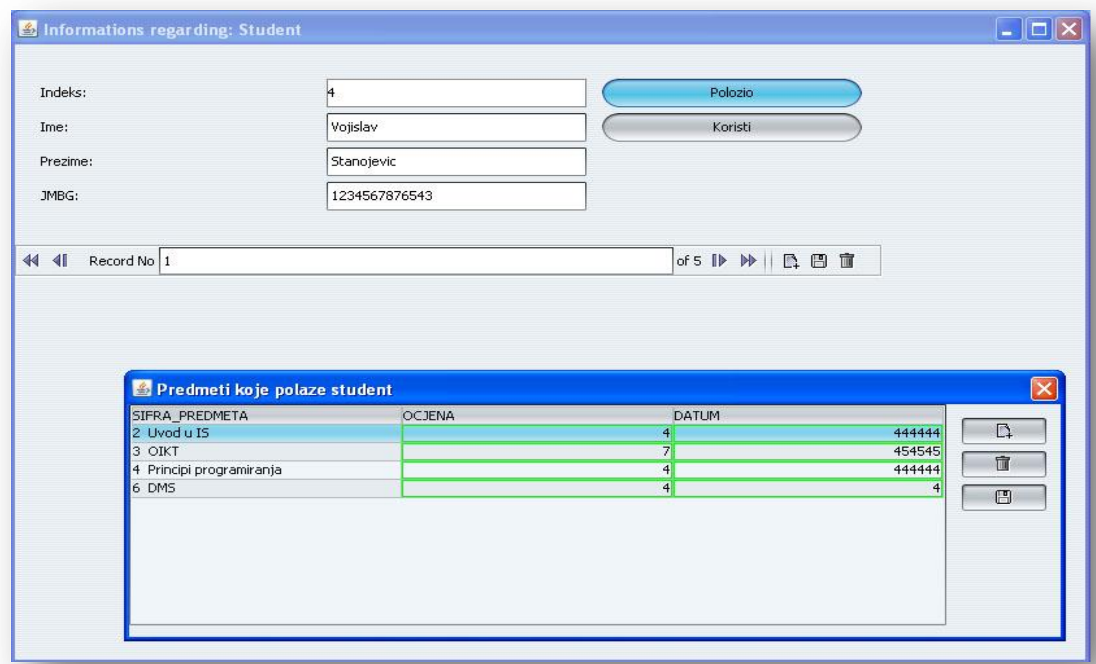
Слика 119: Реализација функционалности *Show details* над падајућом листом на корисничком интерфејсу веб апликације

Приказ, унос измјена или брисање података везаних за дјецу у овом шаблону омогућено је у одвојеним прозорима за свако дијете. Ови прозори се отварају означавањем одређеног опционог дугмета[63] . Све док је одређено дугме означено, приказиваће се прозор са подацима везаним за дијете на које се ово дугме односи. На следећим сликама (Слика 120, Слика 121, Слика 122) биће приказан упрошћени модел података и кориснички интерфејси десктоп и веб апликација, који поред података о основном ентитету садрже и податке о дјеци.

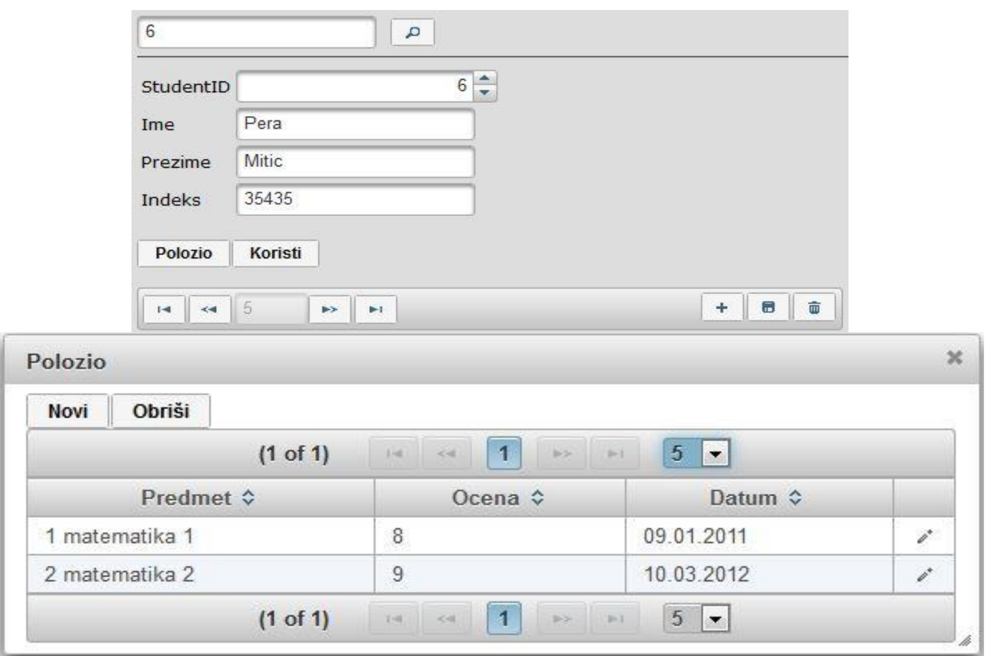


Слика 120: Дио модела података за приказ *field-form* шаблона који укључује информације о дјеци

[63] Опциона дугмад (*Option buttons*) су компоненте које имају изглед дугмета, али слично пољима за чекирање могу да буду у једном од два стања - означени или не.



Слика 121: Кориснички интерфејс реализован коришћењем Field-Form шаблона на корисничком интерфејсу десктоп апликација са приказом информација везаних за дјецу



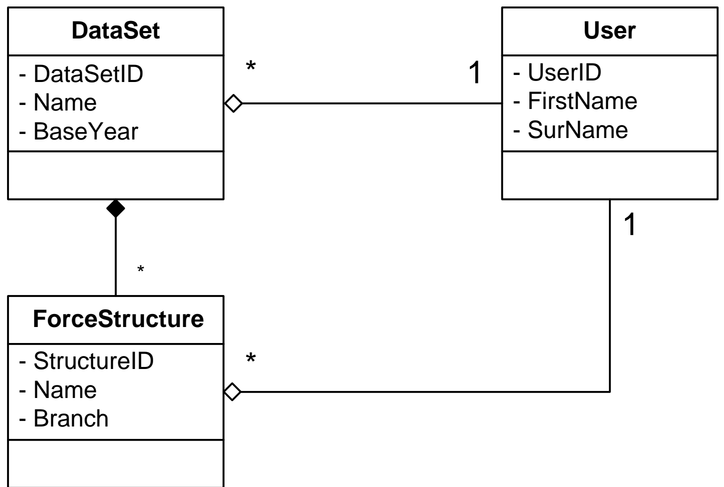
Слика 122: Кориснички интерфејс реализован коришћењем Field-Form шаблона на корисничком интерфејсу веб апликација са приказом информација везаних за дјецу

6.2. FIELD-TAB ШАБЛОН

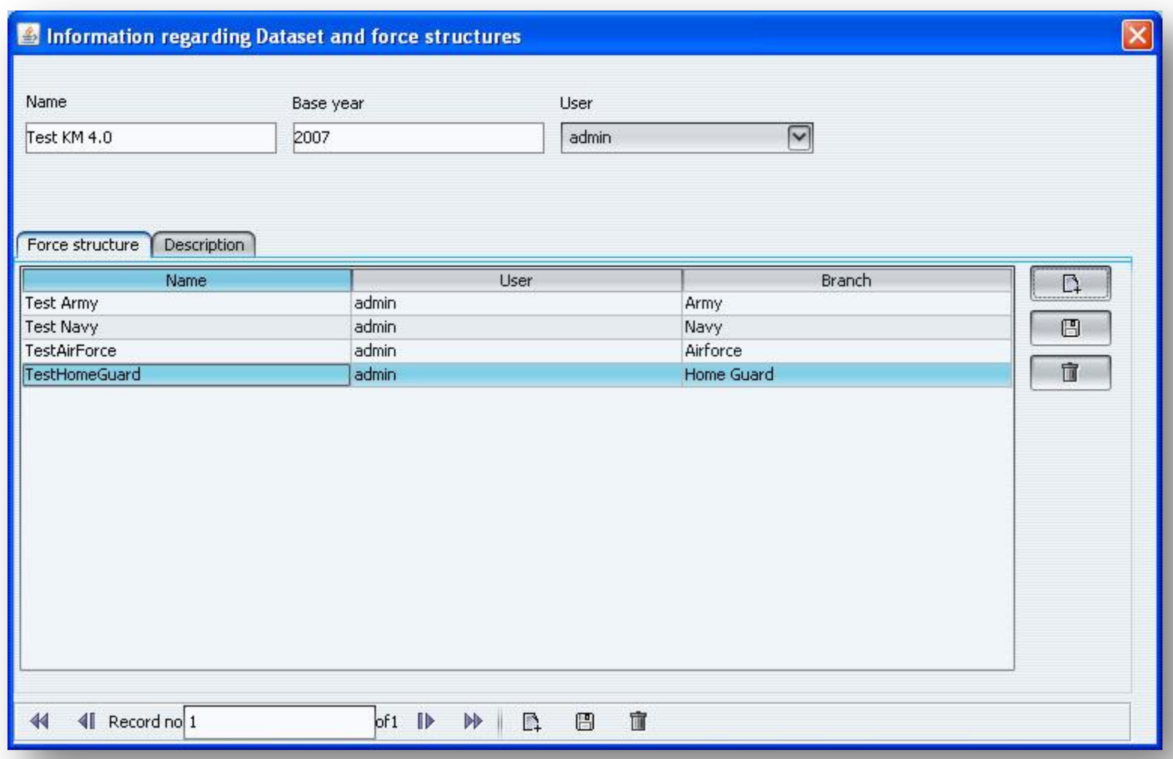
Field-tab шаблон, слично *field-form* шаблону, замишљен је тако да обезбиједи унос и приказ информација везаних за одговарајући ентитет коришћењем поља, тј. графичких компоненти које служе за унос или одабир појединачних података везаних за атрибуте ентитета који се обрађује, док се код њега дјеца ентитета који се обрађује приказују у истом прозору, у различитим табовима (картицама), по узору на *Paralell panels* патерну пројектовања корисничког интерфејса. Свако дијете може бити приказано другачијим шаблоном, што се специфицира у моделу на основу кога се генерише кориснички интерфејс.

Графичке компоненте које се користе у овом шаблону поред оних које се користе за *field-form* шаблон подразумијевају и тзв. *Tabbed-Pane* компоненту која се користи за приказивање панела у којима је садржај везан за дјецу. Приликом навигације кроз основни ентитет коришћењем навигационих дугмади, ажурира се приказ података у табовима на тај начин да се приказују подаци о дјеци тренутно приказаног отац ентитета.

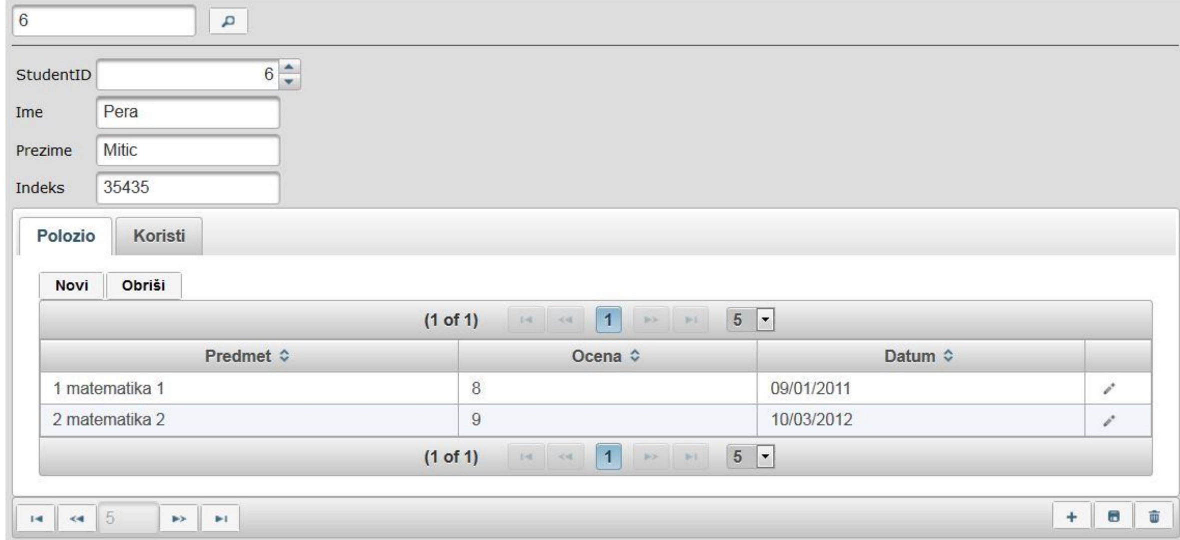
На сљедећој слици биће приказани упрошћени модели података (Слика 123, Слика 126) а након њих и примјери корисничког интерфејса који су реализовани коришћењем *Field-tab* шаблона за приказане моделе података и то редом за десктоп (Слика 124) и веб апликације (Слика 125). На овом примјеру (Слика 125) може се уочити још једна функционалност, а то је да се примарни кључеви, ако се додјељују аутоматски (*autoincrement*), могу сакрити од корисника. Потребно је само да се ова опција назначи у моделу на основу кога се врши генерисање корисничког интерфејса.



Слика 123: Дио модела података за приказ *field-tab* шаблона



Слика 124: Примјер корисничког интерфејса који користи *Field-tab* шаблон у десктоп апликацији



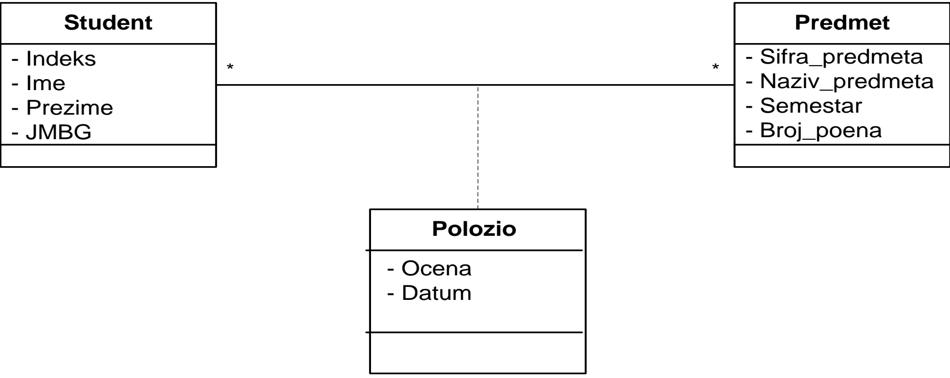
Слика 125: Примјер корисничког интерфејса који користи *Field-tab* шаблон у веб апликацији

Оно што разликује *Field-Tab* од *Field-form* шаблона јесте начин приказивања информација везаних за дјецу одређеног ентитета. Ако случај коришћења обухвата само један ентитет, без дјеце, свеједно је који ће од ова два шаблона бити одабран у моделу на основу кога се врши генерисање корисничког интерфејса.

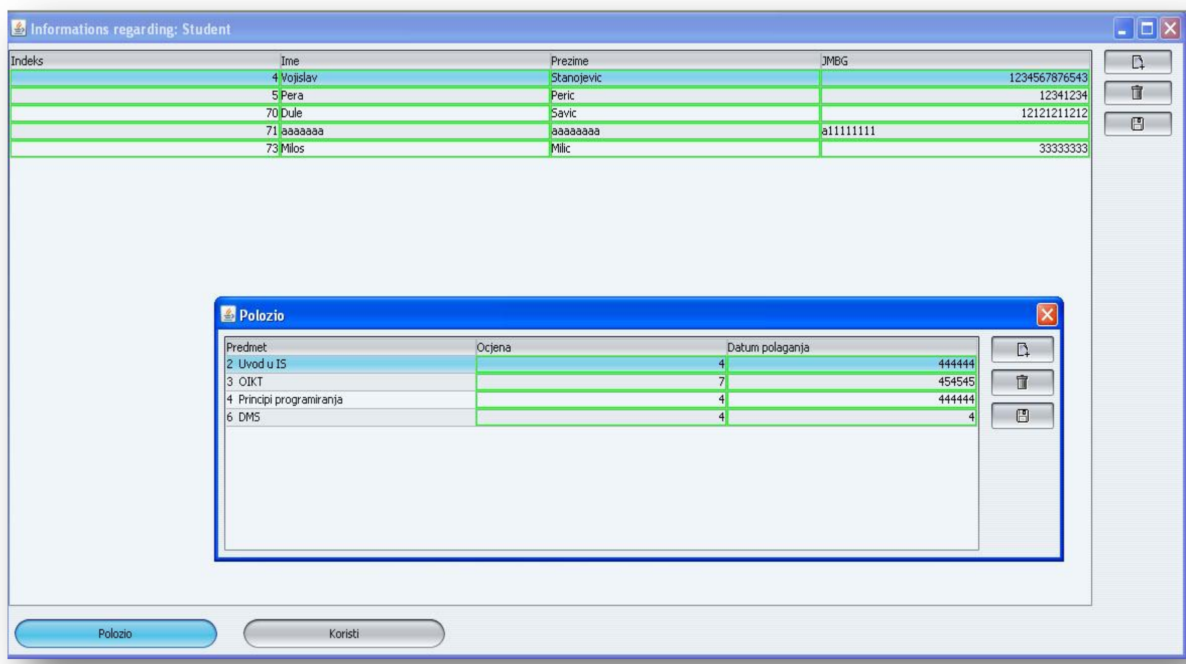
6.3. TABLE-FORM ШАБЛОН

Table-form шаблон замишљен је тако да обезбиједи унос и приказ информација везаних за одговарајући ентитет у табеларном облику, док се дјеца ентитета који се обрађује приказују у засебним прозорима на тај начин што се означи одговарајуће опционо дугме. Свако дијете може бити приказано другачијим шаблоном.

Овај шаблон као основну графичку компоненту користи табелу, по узору на *Spreadsheet* патерн пројектовања корисничког интерфејса, и погодан је у ситуацијама када корисник жели да има преглед информација везаних за више појављивања једног ентитета. *Field-form* и *field-tab* шаблони у једном тренутку могу да прикажу информације о једном појављивању неког ентитета, док је за приступ осталим појављивањима тог ентитета било потребно коришћење навигационих дугмади. У *table-form* шаблону подаци су приказани табеларно, па је омогућена лакша навигација једноставном селекцијом одређеног реда у табели. Приликом селекције одређеног реда, ажурира се приказ у прозорима који приказују информације о дјеци за одабрани отац ентитет. На сљедећој слици биће приказан упрошћени модел података (Слика 126) и кориснички интерфејс који илуструје коришћење Table-form шаблона у десктоп апликацији (Слика 127).



Слика 126: Дио модела података за приказ *Table-form* шаблона у десктоп апликацији

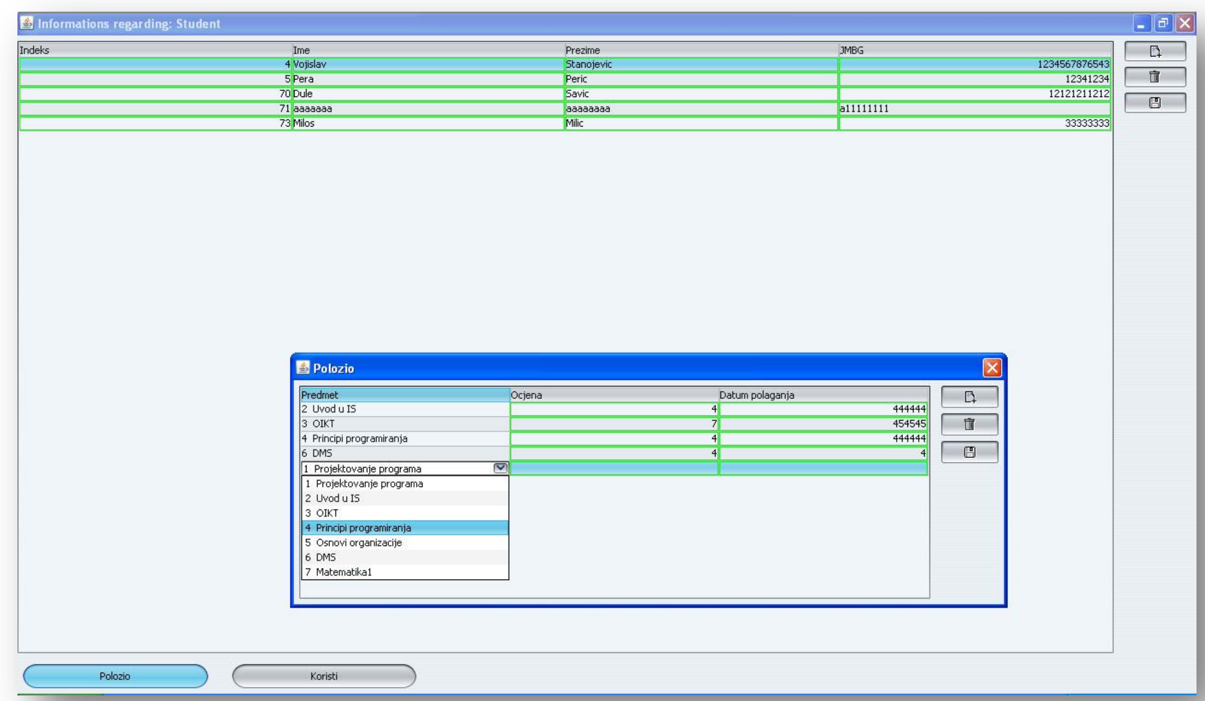


Слика 127: Примјер корисничког интерфејса развијеног коришћењем *table-form* шаблона у десктоп апликацији

Поред приказа података у табеларном облику, омогућена је и измјена и унос података, такође коришћењем табеле. Измјене се врше директно у одабраном реду у табели, а подаци се чувају или позивом опције за чување помоћу дугмета или преласком на неки други ред у табели. Тада ће корисник бити обавијештен о томе да су подаци измијењени, и биће понуђена могућност да се измјене сачувају, или да се одустане од измјена. Уколико подаци нису валидни, систем ће спријечити прелазак на други слог све док се подаци не исправе, или не одустане од чувања измјена.

Приликом уноса података или измјене, у зависности од типа акције специфициране у кораку случаја коришћења, или типа податка који се уноси, кориснику ће бити приказана одговарајућа графичка компонента. Ово је реализовано коришћењем различитих *cell editora* који су прилагођени за коришћење различитих графичких компоненти приликом уноса. На сљедећој слици биће приказан примјер

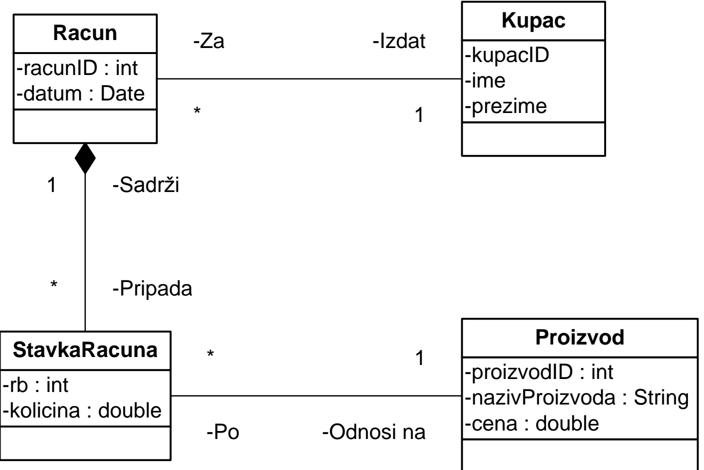
коришћења *cell editor-a* који је прилагођен за приказ падајуће листе у ћелији табеле (Слика 128).



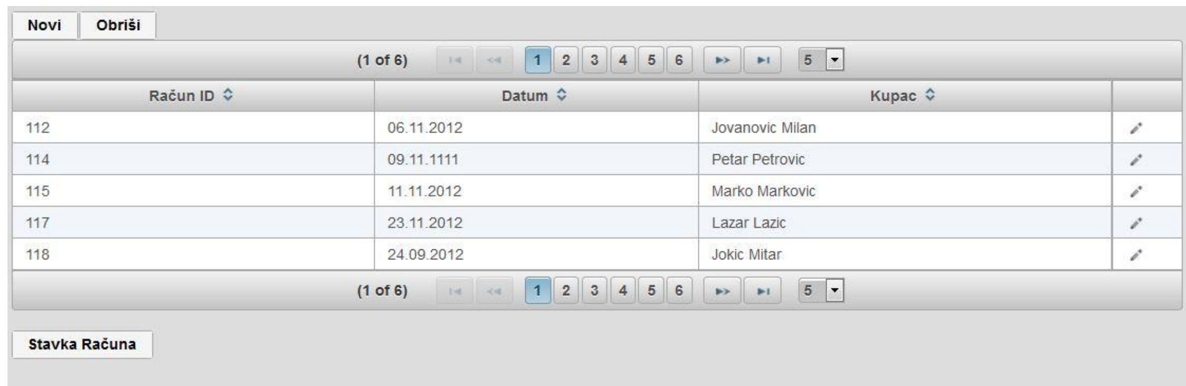
Слика 128: Примјер коришћења *cell editor-a* приликом уноса и измјене података у табели за кориснички интерфејс десктоп апликације

Функционалност којом се приказују детаљније информације о елементу падајуће листе и омогућава њихова измјена (*show details*), такође је обезбијеђена и у овом шаблону и позива се десним тастером миша над ћелијом која садржи падајућу листу.

Table-form шаблон за веб апликације узима у обзир специфичне карактеристике веб апликација описане у претходном поглављу. На сљедећој слици биће приказан упрошћени модел података (Слика 129) и кориснички интерфејс који илуструје коришћење Table-form шаблона у веб апликацији (Слика 130).

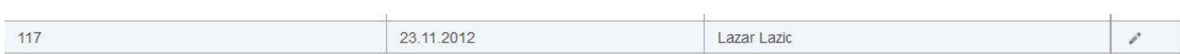


Слика 129: Дио модела података за приказ *Table-form* шаблона у веб апликацији



Слика 130: Примјер корисничког интерфејса развијеног коришћењем *table-form* шаблона у веб апликацији

Одабир реда се врши коришћењем иконице ✐ на десном крају реда (Слика 131), чиме се освјежава приказ података у дијалозима који приказују податке о ентитетима везаним за дјецу.



Слика 131: Одабир реда табеле у *Table-form* шаблону

Поред приказа, омогућена је измјена података директно у табели (Слика 132), али на тај начин што се одабрани ред трансформише тако да се у свакој ћелији приказују одговарајуће компоненте које омогућавају измјену (слично *cell editor*

компонентама у десктоп апликацијама), а поред тога појављују се и иконица за потврду инзмјене података ✓ и иконица за одустајање од измјене ×. Након завршетка измјене, корисник се обавјештава о успјешности извршења операције, а у случају неуспјеле валидације, корисник има могућност да исправи погрешно унесене податке.

Слика 132: Измјена одабраног појављивања ентитета у *Table-form* шаблону за веб апликације

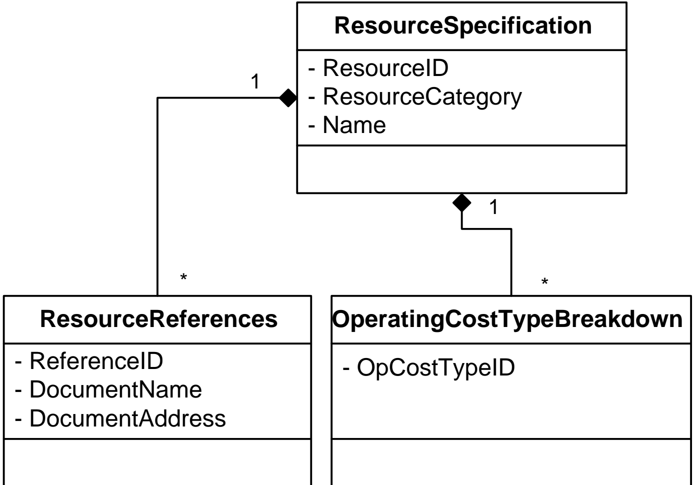
На овај начин је постигнуто да се за измјену, без обзира што се врши директно у оквиру табеле, користе одговарајуће графичке компоненте, са свим додатним функционалностима, попут *Show details* дугмета који се налази уз падајућу листу.

6.4. TABLE-TAB ШАБЛОН

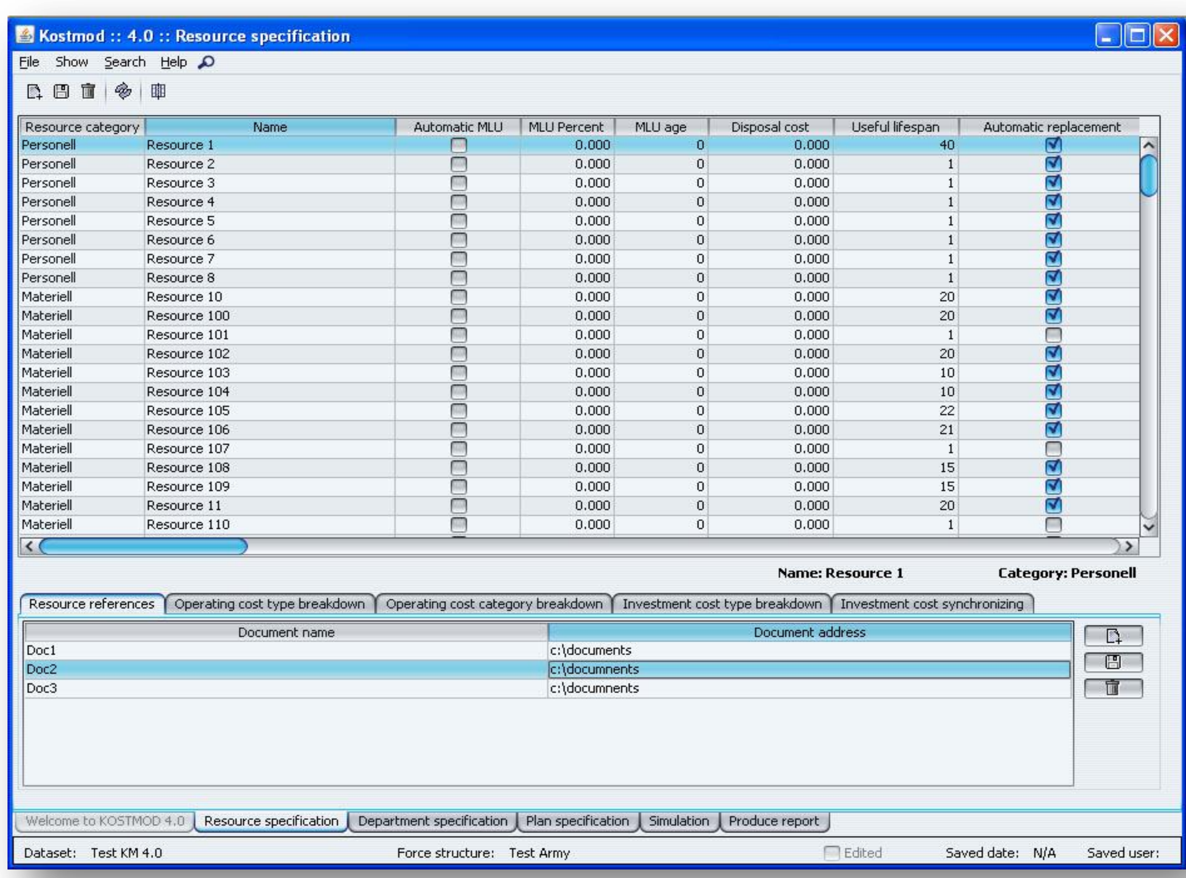
Table-tab шаблон, слично *table-form* шаблону, замишљен је тако да обезбиједи унос и приказ информација везаних за одговарајући ентитет у табеларном облику, док се код њега дјеца ентитета који се обрађује приказују у истом прозору, у различитим табовима (картицама) по узору на *Paralell panels* патерн пројектовања корисничког интерфејса. Свако дијете може бити приказано другачијим шаблоном, што се специфицира у моделу на основу кога се генерише кориснички интерфејс.

Графичке компоненте које се користе у овом шаблону поред оних које се користе за table-form шаблон подразумијевају и тзв. *Tabbed-Pane* компоненту која се користи за приказивање панела у којима је садржај везан за дјецу. Приликом навигације кроз основни ентитет селекцијом одређеног реда у табели, ажурира се приказ података у табовима на тај начин да се приказују подаци о дјеци тренутно одабраног отац ентитета.

На сљедећој слици биће приказан упрошћени модел података (Слика 133) а након њега и примјер корисничког интерфејса (Слика 134) који је реализован коришћењем *table-tab* шаблона за приказани модел података у десктоп апликацији. На овом примјеру може се уочити коришћење *cell editor-a* који је прилагођен за податке логичког типа.



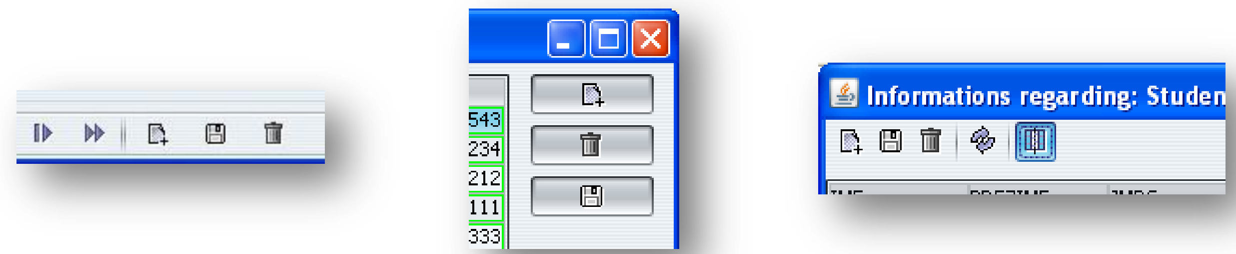
Слика 133: Дио модела података за приказ *table-tab* шаблона за кориснички интерфејс десктоп апликације



Слика 134: Примјер корисничког интерфејса развијеног коришћењем *table-tab* шаблона у десктоп апликацији

Оно што разликује *table-tab* од *table-form* шаблона јесте начин приказивања информација везаних за дјецу одређеног ентитета. Ако случај коришћења обухвата само један ентитет, без дјеце, свеједно је који ће од ова два шаблона бити одабран у моделу на основу кога се врши генерисање корисничког интерфејса.

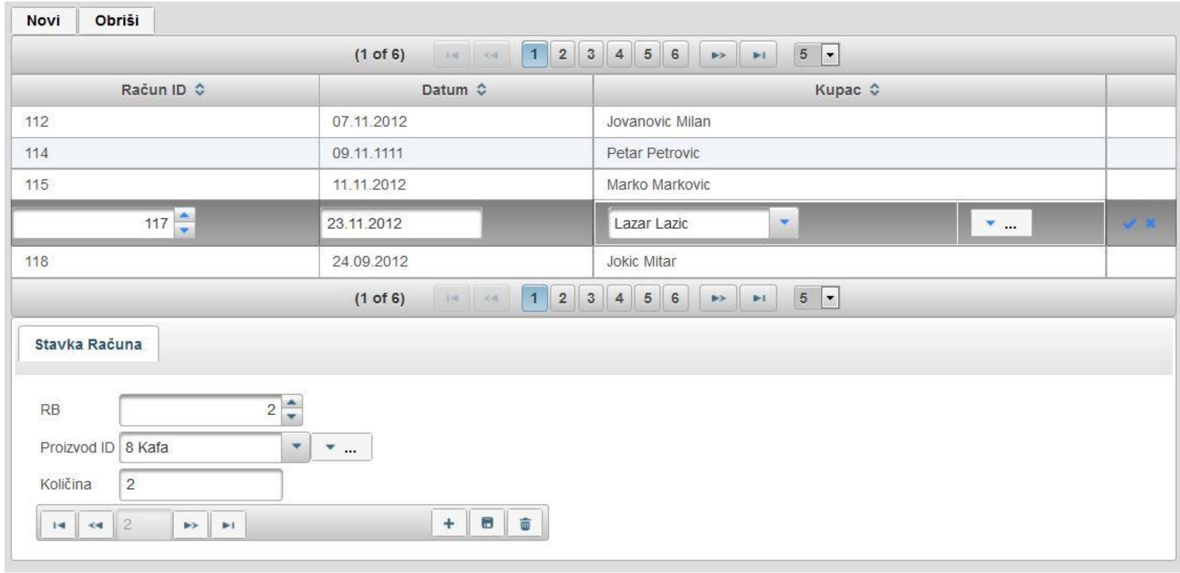
На претходној слици (Слика 134) може се уочити другачији начин приказа опција за креирање, измјену и брисање, и то коришћењем *tool-bar* компоненте. Овакав начин приказа обезбјеђује се специфицирањем одговарајућег параметра у моделу на основу кога се генерише кориснички интерфејс. Погодан је у оним ситуацијама када компоненте за приказ података (у овом случају табела) заузимају много простора на корисничком интерфејсу. Могућност оваквог начина приказа опција није ограничен само на овај шаблон, већ се може користити и у свим осталим шаблонима и у зависности од сваког појединачног случаја може се одабрати одговарајући (Слика 135).



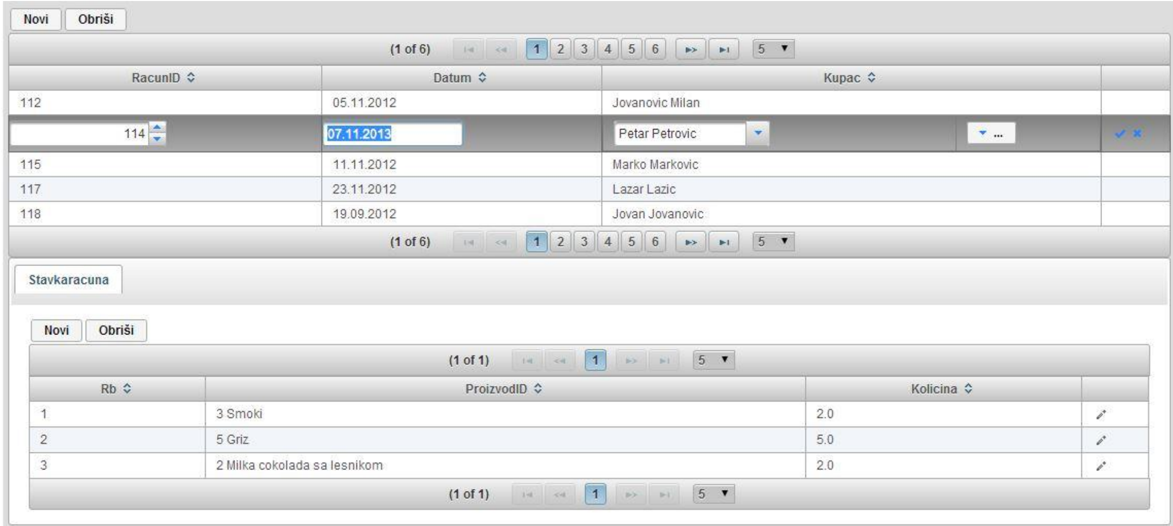
Слика 135: Различити начини приказивања опција за креирање, ажурирање и брисање података

У наставку су приказани примјери коришћења *Table-tab* шаблона на корисничком интерфејсу веб апликације. Први примјер показује примјену шаблона у којем је дијете ентитет приказан коришћењем *Field-form* шаблона (Слика 136), а у другом примјеру (Слика 137) дијете је приказано коришћењем *Table-form* шаблона. Ови примјери показују примјену основног *Table-tab* шаблона, али и начине

комбновања са другим шаблонима. Оба примјера направљена су на основу модела података приказана у дијаграму (Слика 129).



Слика 136: Примјер корисничког интерфејса развијеног коришћењем *Table-tab* шаблона у веб апликацији, са приказом дјетета коришћењем *Field-form* шаблона



Слика 137: Примјер корисничког интерфејса развијеног коришћењем *Table-tab* шаблона у веб апликацији, са приказом дјетета коришћењем *Table-form* шаблона

6.5. KNGUI ШАБЛОН

KNGUI[64] шаблон замишљен је тако да обезбиједи унос и приказ информација везаних за одговарајући ентитет у табеларном облику, док се моделом одабрана дјеца ентитета који се обрађује приказују у истој табели. Остала дјеца могу бити приказана или у табовима или у одвојеним прозорима, коришћењем различитих, претходно описаних шаблона.

Овај шаблон погодан је у ситуацијама када постоје дјеца ентитета која представљају агрегацију ентитета који се обрађује са неким другим ентитетом. Тада се на основну табелу, која у колонама приказује атрибуте ентитета који се обрађује, динамички додају колоне које представљају сва појављивања ентитета са којим се врши агрегација. Ћелија која настаје оваквим укрштањем ће приказивати вриједност атрибута у агрегирајућем ентитету, и то оног атрибута који је специфициран у моделу. На овај начин, табеларни приказ који се добија на корисничком интерфејсу комплетно трансформише модел података који постоји у бази, и у ствари представља укрштање података из најмање три[65] табеле релационе базе података.

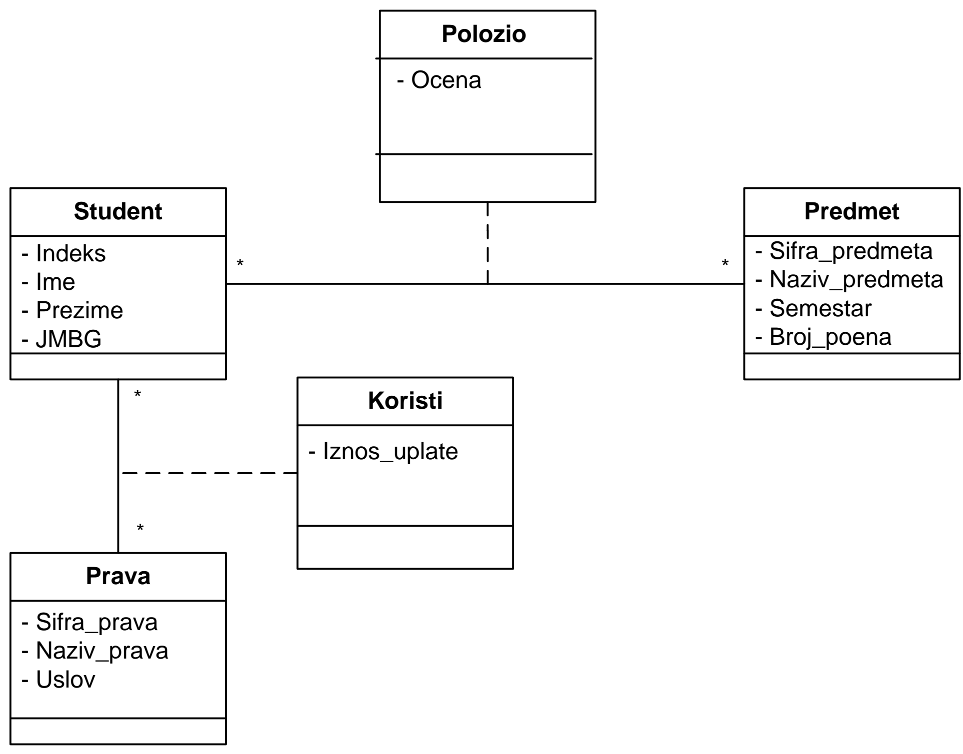
Овај шаблон као основну графичку компоненту користи табелу, и погодан је у ситуацијама када корисник жели да има преглед информација везаних за више појављивања једног ентитета, као и могућност једноставног уноса података везаних за његову дјецу, без заузимања додатног простора на корисничком интерфејсу. Table-

[64] Назив KNGUI представља радни назив овог шаблона који се као такав одомаћио у Лабораторији за софтверско инжењерство на ФОН-у. Наиме, то је скраћеница од квази-нормализовани GUI, назив који је коришћен зато што када се овакав приказ посматра у статичком облику, може се закључити да се ради о нормализованим подацима. Међутим, оно што се на приказаним подацима не види, то је да је број колона које ће се појавити на корисничком интерфејсу промјењив, и зависи од броја слогова у табели која представља агрегацију двије независне релације.

[65] Свако ново дијете које се приказује овим шаблоном укрштаће податке из двије табеле са подацима из основне табеле, која представља отац ентитет који се обрађује.

form и table-tab шаблони за приказ података о сваком дјетету захтијевају приказ панела било у истом, било у одвојеним прозорима. Такође, корисник не мора да се креће из панела у панел како би уписивао различите податке, већ се сви ови подаци уносе у оквиру исте табеле, чиме се значајно убрзава унос. На сљедећој слици биће приказан упрошћени модел података (Слика 138) и кориснички интерфејс (Слика 139) који илуструје коришћење *KNGUI* шаблона у десктоп апликацији. Биће приказан и тренутни садржај табела на основу кога ће динамички бити креиране колоне у табели (

Табела 6, Табела 7, Табела 8, Табела 9).



Слика 138: Дио модела података за приказ KNGUI шаблона

Табела 6: Садржај табеле Студент

Индекс	Име	Презиме	ЈМБГ
4	Vojislav	Stanojevic	1234567876543
5	Pera	Peric	12341234
70	Dule	Savic	12121211212
71	Marko	Markovic	11111111
73	Milos	Milic	33333333
74	Zika	Zivkovic	1234567

Табела 7: Садржај табеле Предмет

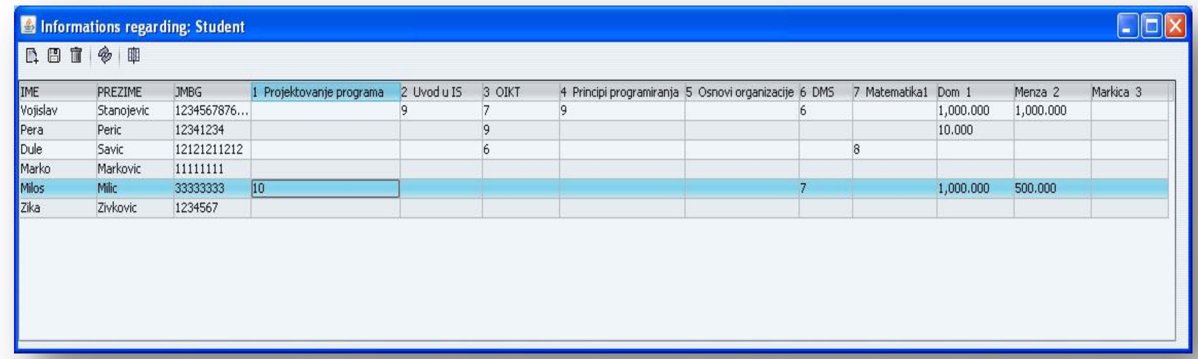
Шифра предмета	Назив предмета	Семестар	Број бодова
1	Projektovanje programa	7	60
2	Uvod u IS	2	30
3	OIKT	1	30
4	Principi programiranja	3	40
5	Osnovi organizacije	1	30
6	DMS	3	30
7	Matematika1	1	35

Табела 8: Садржај табеле Положио

Индекс	Шифра предмета	Оцјена
4	2	9
4	3	7
4	4	9
4	6	6
5	3	9
70	3	6
70	7	8
73	1	10
73	6	7

Табела 9: Садржај табеле Користи

Индекс	Шифра права	Износ уплате
4	1	1000
4	2	1000
5	1	10
73	1	1000
73	2	500



Informations regarding: Student

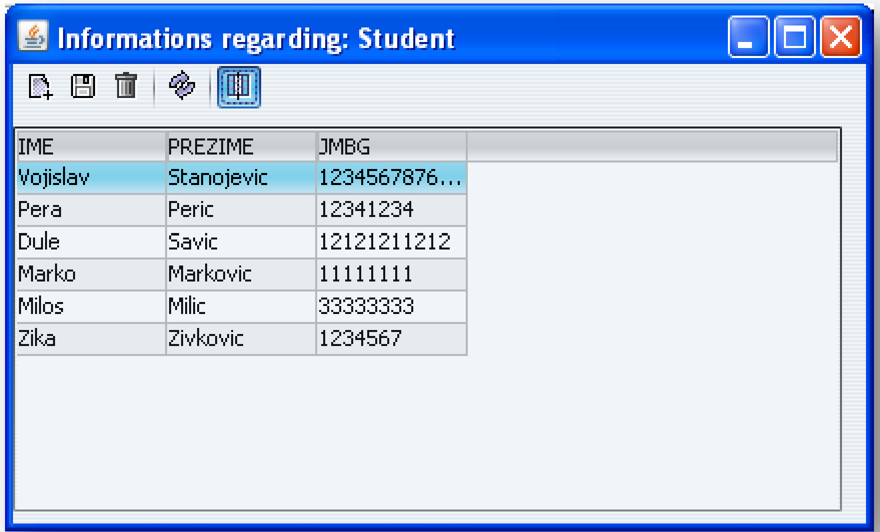
IME	PREZIME	JMBG	1 Projektovanje programa	2 Uvod u IS	3 OIKT	4 Principi programiranja	5 Osnovi organizacije	6 DMS	7 Matematika1	Dom 1	Menza 2	Markica 3
Vojislav	Stanojevic	1234567876...		9	7	9		6		1,000.000	1,000.000	
Pera	Peric	12341234			9					10.000		
Dule	Savic	12121211212			6				8			
Marko	Markovic	11111111										
Milos	Milic	33333333	10					7		1,000.000	500.000	
Zika	Zivkovic	1234567										

Слика 139: Примјер корисничког интерфејса развијеног коришћењем KNGUI шаблона у десктоп апликацији

Трансформација модела из базе података која обезбјеђује овакав приказ података има за посљедицу и другачију интерпретацију измјена над подацима у приказаној табели. Узрок за то је чињеница да све ћелије које немају своју вриједност, а налазе се на пресјеку неког реда, и неке колоне која припада неком *дјетету*, заправо не постоје као слогови у бази података. Наиме, ако се пође од приказаног примјера,

унос оцјене за неког студента из неког предмета захтијеваће убацивање новог слога у агрегирајућој табели у бази података, и то тако што ће примарни кључ овог слога бити састављен од примарних кључева оног студента и оног предмета који се налазе на реду односно колони којој припада ћелија у којој је извршен унос. Брисање оцјене за неког студента из неког предмета захтијева брисање читавог одговарајућег слога из агрегирајуће табеле, док измјена оцјене подразумијева ажурирање слога у агрегирајућој табели, и то само у колони оцјена.

На претходном примјеру показано је да овај шаблон није ограничен само на обрађивање једног *дјетета*, већ се може укључити произвољан број ентитета који представљају *дјецу* обрађиваног основног ентитета (поред уношења података о положеним испитима, обезбијеђено је и уношење података о правима које користи студент). Треба бити опрезан у броју *дјеце* која ће бити укључена у приказ због тога што на тај начин табела може постати предугачка по хоризонтали, и тиме се може смањити прегледност. Такође, ако табела са којом се врши агрегација садржи велики број слогова, приказана табела такође може постати непрегледна. Зато је поред основних операција за креирање, измјену и брисање, уведена и нова, којом се по потреби приказују или сакривају подаци о *дјеци* ентитета који се обрађује. На сљедећој слици приказан је кориснички интерфејс из претходног примјера са укљученом опцијом којом се скривају подаци о *дјеци*. (Слика 140)



Слика 140: Илустрација коришћења операције којом се сакривају подаци о дјеци на *KNGUI* шаблону

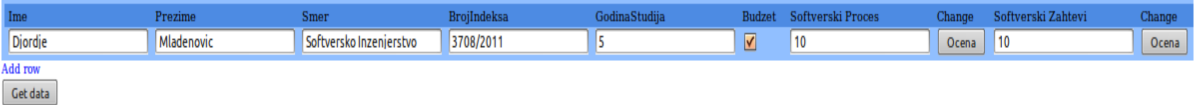
Сличан начин приказивања података омогућен је коришћењем *cross-tab* тј. *pivot* инструкције унутар самог *SQL* упита у *Microsoft SQL Server* [SQLServer] систему за управљање базама података. *Pivot* омогућава приказивање вриједности редова неке табеле као колона резултујуће табеле *SQL* упита. [Cunningham04] Општи облик овог упита може се приказати на следећи начин (Слика 141):

```
SELECT columns
FROM table
PIVOT
( Aggregate Function(Measure Column)
  FOR Pivot Column IN ([Pivot Column Values])
)
```

Слика 141: Општи облик коришћења *PIVOT* наредбе

Оно што разликује *KNGUI* шаблон од претходно описаног приступа јесте што се њиме омогућава не само трансформација података из табела у релациононој бази у циљу другачијег начина приказивања ових података, већ се омогућава и коришћење операција које врше измјене над приказаним подацима које се затим рефлектују и на податке у бази података. Свака операција тада трансформише приказане податке у низ објеката чије се измијењено стање чува у бази коришћењем различитих *SQL* упита, у зависности од насталих измјена.

Што се тиче реализације овог шаблона за веб апликације (Слика 142), а због сложености имплементације и специфичних карактеристика веб апликација, посвећено је много времена истраживању начина имплементације кроз развој нове веб графичке компоненте, а детаљни резултати истраживања приказани су у оквиру Мастер рада [Mladenovic13].



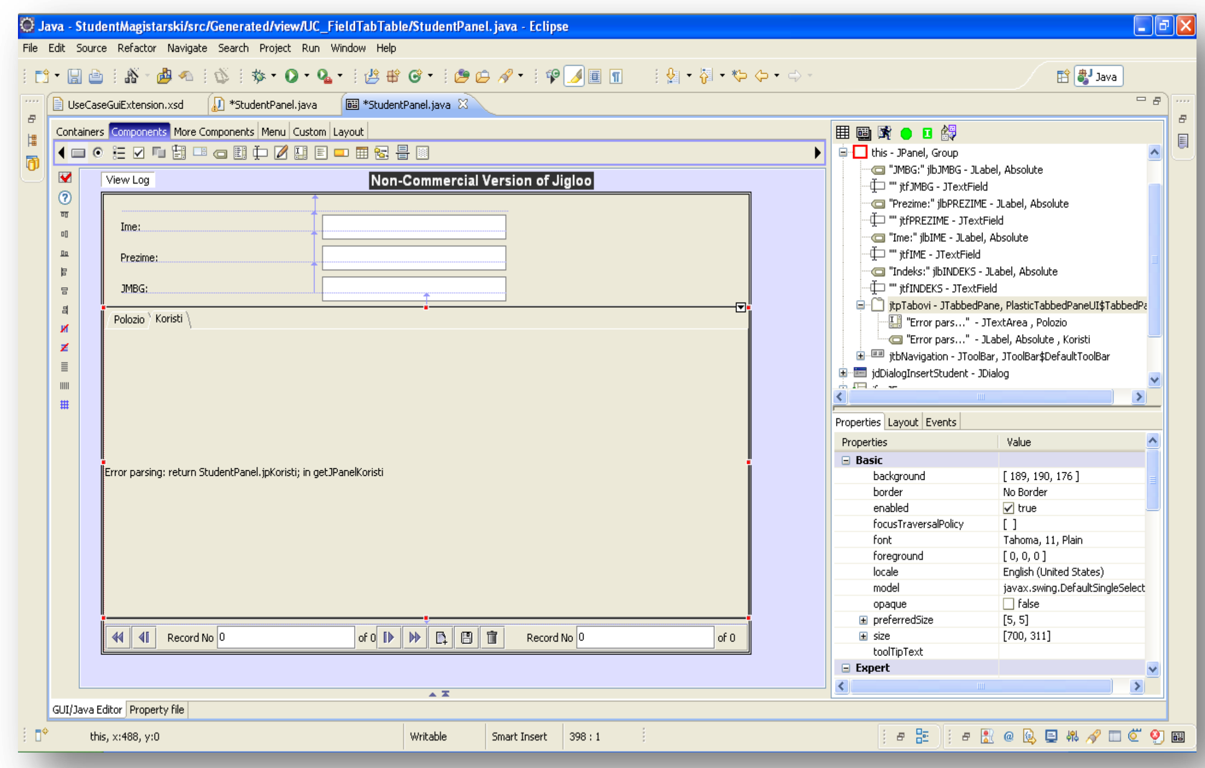
Слика 142: Примјер корисничког интерфејса развијеног коришћењем KNGUI шаблона у веб апликацији

7. Предлог новог приступа – *SilabUI*

У овом поглављу биће приказане карактеристике новог приступа који представља резултат истраживања спроведеног у оквиру израде докторске дисертације. Нови приступ је назван *SilabUI*, а име чини скраћени назив Лабораторије за софтверско инжењерство Факултета организационих наука (*SILAB*) [SILAB], у оквиру које је спроведено истраживање, и суфикс који означава да је пројекат усмјерен на развој корисничког интерфејса (*User Interface – UI*). *SilabUI* приступ има за циљ да обједини сва сазнања до којих се дошло током истраживања, а која су приказана у претходним поглављима.

У другом поглављу су разматрани различити приступи у развоју корисничког интерфејса (скицирање, визуелно креирање, коришћење језика за спецификацију, као и коришћење модела и моделом вођени развој), а за сваки приступ су приказане предности и недостаци. У овом контексту *SilabUI* приступ се може посматрати као приступ који је заснован на моделима, и који користи предности моделом вођеног развоја, али исто тако комбинује добре карактеристике осталих приступа. У наставку ће бити приказани начини за спецификацију модела, на основу кога ће се вршити генерисање корисничког интерфејса, међу којима је изузетно значајна могућност директног креирања улазне спецификације која је заснована на *XML*-у, као једном виду језика за спецификацију будућег корисничког интерфејса. Поред тога, могућност коришћења шаблона корисничког интерфејса (поглавље 6.) и њиховог комбиновања, уз једноставност промјене одабраних шаблона кроз промјену улазне

спецификације, омогућава прављење корисничког интерфејса који се може користити као прототип приликом прикупљања и валидације корисничких захтјева, па се на овај начин постижу све предности које пружају алати за скицирање корисничког интерфејса. *SilabUI* приступ предвиђа генерисање програмског кода корисничког интерфејса који се касније може учитати и бити „препознат“ од стране алата за визуелно креирање корисничког интерфејса (Слика 143), и на тај начин искористити предности алата за даљу обраду генерисаног корисничког интерфејса, уколико за тиме постоји потреба.



Слика 143: Обрада генерисаног корисничког интерфејса коришћењем *Jigloo* [Jigloo] алата за визуелно креирање корисничког интерфејса који се интегрише са *Eclipse* развојним окружењем

У трећем поглављу разматрани су постојећи приступи инжењерингу софтверских захтјева, техника за прикупљање захтјева, дат је преглед карактеристика

двије доминантне технике за спецификацију захтјева (случајеви коришћења и корисничке приче), и идентификоване су предности коришћења случајева коришћења (прије свега структурираност, цјеловитост и ниво детаљности) у контексту аутоматизације процеса креирања корисничког интерфејса засноване на захтјевима. Поред тога објашњена је и улога доменског модела и његова веза са случајевима коришћења. Такође, приказано је и неколико најчешће коришћених шаблона за спецификацију случајева коришћења. На крају су дате препоруке за спецификацију корака у сценарију случаја коришћења и њихов утицај на будући кориснички интерфејс. Узимајући у обзир изведене закључке формиран је мета-модел за креирање улазне спецификације која је у потпуности заснована на случајевима коришћења. Поглавље 3 завршава објашњењем улоге прототипова у инжењерингу захтјева, гдје се истичу предности коришћења еволутивних прототипова. *SilabUI* приступ омогућава генерисање програмског кода корисничког интерфејса који се може извршити, а поред тога омогућена је и његова измјена, па се добијени кориснички интерфејс може посматрати као извршиви еволутивни прототип.

Поглавље 4 представља покушај да се дође одговора на питања који су разлози за чињеницу да ни један од постојећих алата није успио да око себе окупи широк круг корисника, односно да задовољи потребе привреде, који су то захтјеви које алати за генерисање корисничког интерфејса морају да испуне како би били прихваћени од стране потенцијалних корисника и да ли данас актуелни алати испуњавају ове захтјеве. Зато је прво спроведено испитивање у форми анкете чији је циљ био утврђивање преовлађујућег става носилаца софтверске индустрије – искусних софтверских инжењера, по питању елемената тј. карактеристика које алат за генерисање корисничког интерфејса мора да посједује како би га прихватили и користили у процесу развоја софтвера. Уочене карактеристике ће у овом поглављу бити посматратне као захтјеви које нови приступ генерисању корисничког

интерфејса мора да задовољи како би био прихваћен од стране потенцијалних корисника. На основу уочених захтјева дефинисани су критеријуми на основу којих је извршено поређење постојећих алата за генерисање корисничког интерфејса. У наставку ће бити приказана листа критеријума, са карактеристикама које ће бити дио новог приступа, а које су од стране испитаника препознате као потребне, уз карактеристике које ће бити прихваћене као добра пракса из постојећих приступа.

- **Модели на којима је заснована улазна спецификација**

  Анализирајући изнесене ставове испитаника по неколико различитих питања везаних за моделе који се користе у раним фазама софтверског пројекта може се закључити да случајеви коришћења тијесно повезани са доменским моделом доминирају као полазни модел у развоју софтвера. Уз случајеве коришћења и доменски модел, испитаници су у значајној мјери истакли и потребу прављења прототипова корисничког интерфејса. Овакви резултати само потврђују закључке добијене у трећем поглављу разматрањем постојећих приступа инжењерингу софтверских захтјева и техника за прикупљање захтјева. Зато ће мета-модел за креирање улазне спецификације омогућити директно повезивање елемената спецификације случајева коришћења и елемената доменског модела, уз могућност придруживања информација које одређују жељене специфичности будућег корисничког интерфејса. Иако су овакви резултати на линији досадашњих личних искустава које сам имао у развоју софтвера, па су за мене били очекивани, поставља се питање зашто нити један од разматраних алата (осим само дјелимично *WebRatio* и *BizAgi BMP*) за аутоматизацију развоја корисничког интерфејса није на овај начин пројектовао улазну спецификацију. Сматрам да је ово један од значајнијих разлога слабог прихватања посматраних приступа од стране ширег круга корисника.

- **Зависност улазне спецификације од модела података**

  Сви посматрани алати су у великој мјери зависни од модела података, а потребу за успостављањем ове зависности приликом дефинисања улазне спецификације потврђују и изнесени ставови испитаника. Оно што треба нагласити јесте да улазна спецификација не би требала да буде искључиво заснована на моделу података, као што је случај са већином посматраних алата. Наиме, као што резултати истраживања потврђују, а што је дијелом изнесено у образложењу претходног критеријума, модел података, прије свега доменски модел (који описује структуру софтверског система) треба да прати спецификацију случајева коришћења, која поред структуре описује и понашање корисника приликом интеракције са системом, као и спецификацију тога шта систем треба да ради (понашање софтверског система). Сматрам да улазна спецификација тек у том случају постаје семантички довољно богата за аутоматизацију развоја интерфејса.

- **Начин дефинисања улазне спецификације**

  Ставови испитаника по питању начина дефинисања улазне спецификације су подијељени, прије свега између визуелног моделовања и графичког алата који би водио корисника кроз процес дефинисања спецификације, али значајан број испитаних сматра да је пожељни начин дефинисања улазне спецификације ручним куцањем кода доменски специфичног језика развијеног за ту намјену. Са друге стране, посматрани алати за аутоматизацију развоја корисничког интерфејса углавном омогућавају само један начин дефинисања улазне спецификације. *SilabUI* приступ омогућава сва три начина дефинисања улазне спецификације, о чему ће бити више ријечи у наставку поглавља.

- **Начин креирања корисничког интерфејса**

  У погледу начина креирања корисничког интерфејса, односно да ли алат за аутоматизацију развоја корисничког интерфејса треба да као резултат обезбиједи програмски код корисничког интефејса (*user interface source code generation*) или кориснички интерфејс који ће бити резултат интерпретације улазне спецификације у вријеме извршења (*runtime user interface generation*), испитаници су убједљиво опредијељени за генерисање програмског кода корисничког интерфејса. *SilabUI* приступ омогућава управо овакав начин креирања корисничког интерфејса[66], уз омогућавање кориснику потпуне контроле (измјене, допуне и сл.) над генерисаним програмским кодом. По овом критеријуму се разликују и посматрани алати, једни подржавају један, а други подржавају други приступ.

- **Измјењивост генерисаног програмског кода**

  Као што је претходно наглашено, *SilabUI* приступ омогућава кориснику потпуну контролу над генерисаним програмским кодом. Иако неки од посматраних алата такође обезбјеђују генерисање програмског кода корисничког интерфејса, ови алати углавном не дозвољавају кориснику да врши измјене у генерисаном програмском коду.

[66] *SilabUI* пројекат је у почетку обезбјеђивао оба начина креирања корисничкког интерфејса. Развој једног и другог начина је вршен паралелно, међутим ова пракса је са једне стране изискивала много ресурса, док реална потреба за оба приступа није постојала, јер су наручиоци софтверских пројеката искључиво захтијевали генерисање програмског кода корисничког интерфејса. Коришћење другог начина генерисања, односно интерпретације улазне спецификације и генерисање корисничког интерфејса у вријеме извршења, свело се на валидацију корисничких захтјева односно прављење прототипова који се бацају (*throwaway prototype*), али се за ову намјену може користити и генерисање програмског кода корисничког интерфејса. Након извршеног испитивања и анализе резултата, који показују убједљиву опредијељеност испитаних за генерисање програмског кода корисничког интерфејса, одлучилено је да се одустане од даљег паралелног развоја оба приступа и пребаци фокус на генерисање програмског кода корисничког интерфејса.

- **Подржана могућност избора шаблона корисничког интерфејса**

Већински став испитаника о информацијама везаним за кориснички интерфејс којим је потребно проширити улазну спецификацију је да спецификација треба да омогући избор између различитих шаблона корисничког интерфејса. Поред тога, много мањи, али ипак значајан проценат испитаних сматра да корисник треба да има могућност избора специфичних графичких компонената. Посматрани алати за аутоматизацију развоја корисничког интерфејса не пружају могућност избора шаблона корисничког интерфејса (изузев ограничене подршке алата *WebRatio*), док поједини омогућавају избор специфичних графичких компонената. *SilabUI* приступ омогућава избор између различитих шаблона корисничкког интерфејса (о којима је било ријечи у шестом поглављу), а такође кориснику пружа флексибилност при избору графичких компоненти. Уз то, ова подешавања нису обавезна, већ је омогућено да улазна спецификација не садржи нити једну информацију о корисничком интерфејсу. Интерпретацијом овакве спецификације алат за генерисање програмског кода поставља подразумијевана подешавања везана за кориснички интерфејс. На овај начин је обезбијеђено да корисник може дефинисати улазну спецификацију и генерисати кориснички интерфејс без улажења у детаље везане за кориснички интерфејс, што знатно убрзава процес развоја. Након тога, анализом добијеног корисничког интерфејса, уколико се покаже потреба за другачијим подешавањима, корисник може допунити спецификацију потребним подешавањима. О начину избора шаблона корисничког интерфејса и специфичних графичких компоненти ће бити више ријечи у наставку поглавља.

- **Подршка за различите типове апликација**

  Као и већина посматраних алата за аутоматизацију развоја корисничког интерфејса, *SilabUI* приступ такође омогућава развој корисничког интерфејса за различите типове софтверских система, узимајући у обзир све специфичности различитих типова софтверских система (о чему је било ријечи у петом поглављу), коришћењем различитих имплементационих технологија[67]. Резултати испитивања такође показују да испитаници ова питања сматрају веома значајним и потребним дијеловима алата за аутоматизацију развоја корисничког интерфејса.

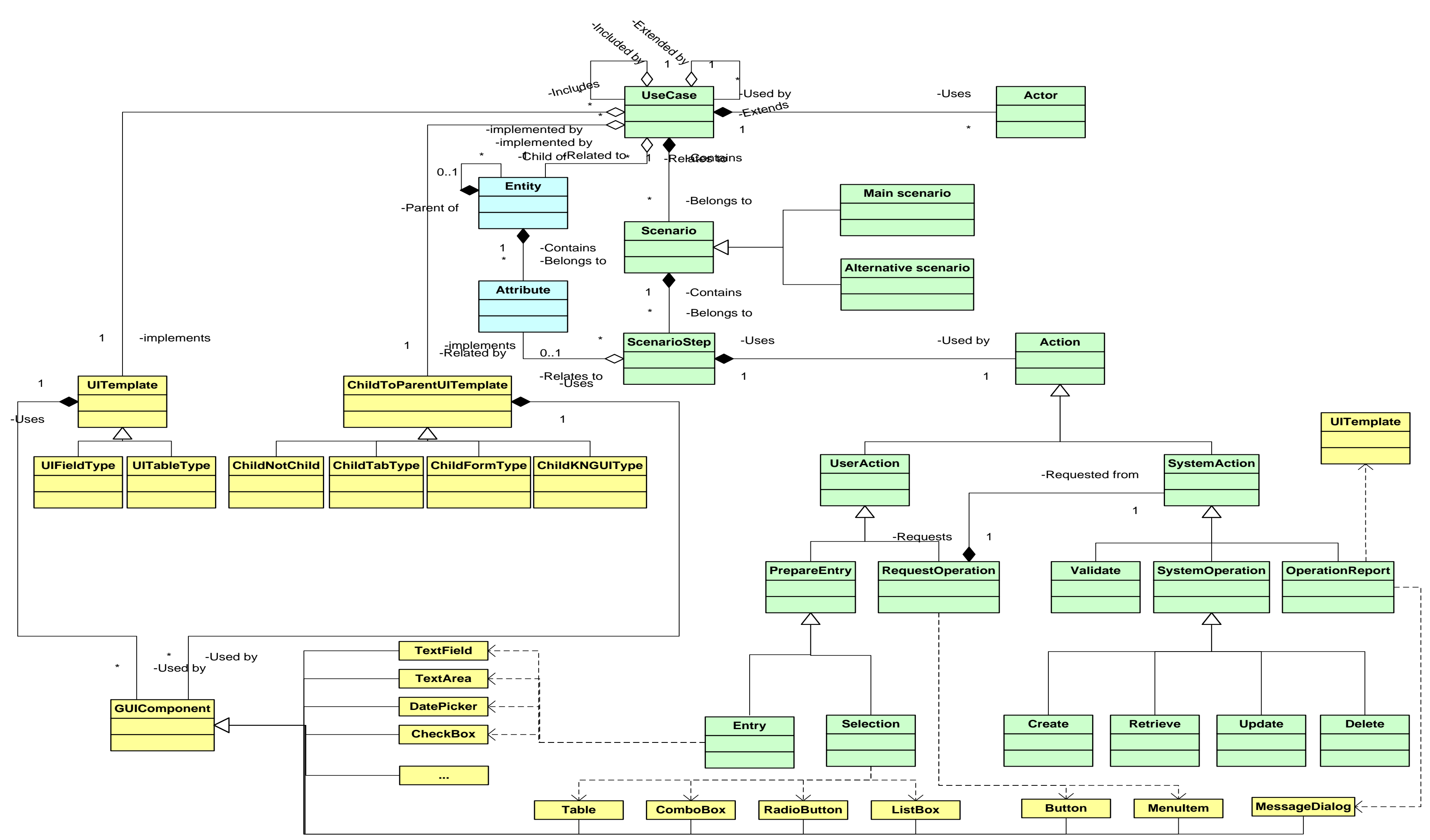
У наставку ће детаљно бити приказане све наведене карактеристике *SilabUI* приступа.

7.1. Мета-модел софтверских захтјева заснован на моделу случајева коришћења и информацијама везаним за кориснички интерфејс

Централна тачка *SilabUI* приступа је мета-модел софтверских захтјева заснован на моделу случајева коришћења и информацијама везаним за кориснички интерфејс. Мета-модел представља синтезу свих сазнања и закључака који су представљени у претходним поглављима. На основу мета-модела конструишу се

[67] У тренутној фази истраживања *SilabUI* у потпуности подржава развој десктоп и веб апликација и то коришћењем Swing и JSF имплементационих технологија. Поред тога, направљени су и алати који подржавају развој коришћењем Flex и GWT имплементационих технологија за веб апликације, али ови алати још увијек нису достигли потребну зрелост, па неће бити представљени у дисертацији.

конкретни модели који чине улазну спецификацију коју користи софтверски алат за генерисање програмског кода корисничког интерфејса. Мета-модел је прије свега заснован на случајевима коришћења и омогућава креирање једноставних и интуитивних конкретних модела на основу којих ће се вршити генерисање корисничког интерфејса, допуштајући истовремено флексибилност при избору различитих шаблона корисничког интерфејса и конкретних графичких компоненти. У наставку ће у форми UML дијаграма класа бити приказана структура мета-модела



Слика 144: *SilabUI* мета-модел

Основни ентитет мета-модела је класа *UseCase* која садржи податке о случају коришћења за који се генерише кориснички интерфејс. Случај коришћења садржи један или више сценарија који су дефинисани класом *Scenario* који може бити основни (*MainScenario*) или алтернативни (*AlternativeScenario*), а сваки сценарио садржи више корака сценарија (*ScenarioStep*).

Са друге стране, случај коришћења је мета-моделом директно повезан са одређеним ентитетом доменског модела (класа *Entity*), који садржи више атрибута (класа *Attribute*). Ентитети доменског модела између себе могу успостављати везу родитељ – дијете, на начин који је описан у шестом поглављу. На овај начин је мета-моделом успостављена директна веза између случајева коришћења и доменског модела. У складу са препорукама за спецификацију корака у сценарију случаја коришћења, корак сценарија је директно повезан са моделом података, односно атрибутом ентитета, као и са врстом акције (класа *Action*), на начин који је описан у трећем поглављу. Акција може да буде системска (*SystemAction*) или корисничка (*UserAction*). Системска акција може бити акција валидације (*Validate*), системска операција (*SystemOperation*) или приказивања резултата извршења операције актору (*OperationReport*). Што се тиче корисничких акција, како је објашњено у трећем поглављу, могу се разликовати акције припреме улазних података (*PrepareEntry*) или позива акције система (*RequestOperation*). Припрема улазних података подразумијева или акцију уноса података (*Entry*), или одабир понуђених података (*Selection*).

Сваки случај коришћења може успоставити релацију *Include* или *Extend* са другим случајевима коришћења. Иако се коришћење ових веза у литератури [Johnston] различито тумачи, могу се сматрати веома корисним приликом формирања улазне спецификације, јер омогућавају већу „грануларност“ случајева коришћења. Наиме, коришћењем ових релација у мета-моделу је омогућено да се један случај коришћења специфицира на тај начин да се односи на само један ентитет

из модела података, а да се успостављањем релација *Include* и *Extend* праве зависности које постоје између различитих случајева коришћења, који у том смислу постају само дијелови неког сложенијег случаја коришћења. Због различитог тумачења ових веза које се могу наћи у литератури, имам потребу да нагласим да ће се у контексту ове дисертације релација *Include* између два случаја коришћења користити у ситуацији када се приликом извршења основног сценарија једног случаја коришћења подразумијева и извршење основног сценарија укљученог случаја коришћења. Ако се посматрају случајеви коришћења Унос рачуна и Унос ставке рачуна, релација *Include* би била дефинисана тако да случај коришћења Унос рачуна (који је спецификацијом повезан са ентитетом Рачун из доменског модела) укључује случај коришћења Унос ставке рачуна (који је спецификацијом повезан са ентитетом Ставка рачуна из доменског модела). На овај начин се специфицира да се приликом уноса рачуна подразумијева бар један унос ставке рачуна. Релација *Extend* ће бити коришћена у ситуацијама када се приликом извршења основног сценарија једног случаја коришћења може (а не мора) десити потреба за извршењем основног сценарија другог случаја коришћења. На овај начин полазни случај коришћења се проширује другим случајем коришћења. Ако се посматра случај коришћења Унос рачуна (који је спецификацијом повезан са ентитетом Рачун из доменског модела који укључује информацију о купцу којем се издаје рачун) и случај коришћења Унос купца (који је спецификацијом повезан са ентитетом Купац из доменског модела), приликом уноса података о Рачуну подразумијева се одабир Купца коме се рачун издаје. Ако се деси да Купац коме се издаје рачун не постоји у евиденцији, релацијом *Extend* се омогућава да се извршење основног сценарија случаја коришћења Унос купца покрене приликом извршења основног сценарија случаја коришћења Унос рачуна. Коришћењем релација *Include* и *Extend* омогућава се комбиновање различитих спецификација случајева коришћења, али и њихово поновно коришћење. На овај начин могуће је

формирати сложене случајеве коришћења који се односе на ентитете у више различитих нивоа хијерархије повезаних везом родитељ-дијете, а да се при томе задржи једноставност и прегледност спецификације.

Што се тиче информација везаних за кориснички интерфејс, оне се дефинишу одабиром шаблона корисничког интерфејса са једне стране, односно одабиром конкретних графичких компоненти са друге стране. Спецификација шаблона корисничког интерфејса који ће бити придружен одређеном случају коришћења се врши кроз успостављање везе између класе која представља основу шаблона корисничког интерфејса (*UITemplate*) и класе која дефинише начин приказа дијете-ентитета у односу на приказ ентитета родитељ (*ChildToParentUITemplate*). [68] Основа шаблона корисничког интерфејса се односи на начин приказа информација о ентитету текућег случаја коришћења. Како је приказано у шестом поглављу, информације се могу приказати на два основна начина и то коришћењем поља (*UIFieldType*) или коришћењем табеле (*UITableType*). Поред основе шаблона корисничког интерфејса специфицира се и начин приказа дијете-ентитета у односу на приказ ентитета родитељ и ту се разликује неколико могућих ситуација:

- када се спецификација односи на ентитет који није дијете, односно ентитет на врху хијерархије (*ChildNotChild*),
- када је ентитет дијете које се жели приказати у оквиру истог прозора ентитета родитељ, али у засебном панелу у форми таба-картице

[68] Претходне верзије мета-модела су омогућавале једноставнији начин спецификације шаблона корисничког интерфејса простим одабиром једног од понуђених шаблона. Међутим, иако нешто простији, овакав мета-модел је имао велико ограничење које се огледа у томе да на корисничком интерфејсу који је везан за један родитељ-ентитет, сва дјеца-ентитети морају бити приказана или у истом прозору у оквиру табова (картица), или да се сва дјеца-ентитети приказују у оквиру засебних прозора (дијалога). Нови начин реализације потпуно уклања ово ограничење и омогућава да се за сваки дијете-ентитет слободно пројектује кориснички интефејс независно од корисничког интерфејса остале дјеце-ентитета.

(*ChildTabType*) слично *Paralell panels* патерну пројектовања корисничког интерфејса приказаног у петом поглављу,

- када је ентитет дијете које се жели прикати у оквиру засебног прозора у односу на ентитет родитељ у оквиру дијалога (*ChildFormType*),
- када је ентитет дијете које се жели приказати у оквиру истог прозора ентитета родитељ, у оквиру исте табеле у којој се приказују информације везане за ентитет родитељ, на начин који је приказан у шестом поглављу везано за KNGUI шаблон (*ChildKNGUIType*).

Комбинацијом спецификације основног шаблона и начина приказа дијете-ентитета у односу на приказ ентитета родитељ омогућава се формирање свих шаблона корисничког интерфејса приказаних у шестом поглављу.

Поред спецификације шаблона корисничког интерфејса, мета-модел омогућава и флексибилност приликом избора конкретних графичких компоненти. Класа *GUIComponent* и класе које је насљеђују омогућавају спецификацију графичких компоненти. На слици (Слика 144), због прегледности је приказан само дио могућих графичких компоненти које су подржане мета-моделом, а комплетан скуп графичких компоненти ће бити приказан у наставку поглавља. Спецификација конкретних графичких компоненти складу је са везама између спецификације случајева коришћења и корисничког интерфејса које су идентификоване у шестом поглављу. Као што је утврђено, врста акције у сценарију случаја коришћења, као и особине атрибута ентитета на који се акција односи, утичу на одабир графичке компоненте. Ипак, и поред тога што се на овај начин олакшава одабир компоненте и сужава скуп компоненти које у датом контексту могу бити коришћене, корисник и даље има могућност избора између више компоненти које могу бити одговарајуће. Типичан примјер је корак у сценарију у којем је врста корисничке акције унос, а атрибут који се уноси је типа датум. Овај корак се на корисничком интерфејсу може реализовати

коришћењем текстуалног поља, али и коришћењем специјализованих графичких компоненти за одабир датума из приказаног календара, или коришћењем три текстуална поља (за дан, мјесец и годину). Коришћењем мета-модела, кориснику је остављена могућност избора између различитих графичких компоненти које се у одређеном контексту могу користити.

7.1.1. XML дефиниција модела

Како би се дефинисани мета-модел могао користити за формирање конкретних модела који представљају улазну спецификацију коју користе алати за аутоматско генерисање програмског кода корисничког интерфејса, потребно је да модел буде дефинисан у формату који је погодан за манипулацију електронским путем, а који у исто вријеме могу читати и разумјети све заинтересоване стране у процесу спецификације корисничких захтјева. Као најпогоднији формат за ову намјену намеће се XML (*Extensible Markup Language*), као језик са једноставном синтаксом, који је истовремено читљив људима и рачунарским програмима.

Како спецификација софтверских захтјева треба да садржи информације о циљу и намјерама корисника приликом коришћења одређеног дијела система, а не имплементационе детаље (који се најчешће тичу корисничког интерфејса, али и реализације других дијелова система), дакле како би спецификација потпуно остала у домену проблема, а не у домену рјешења, моделом је потребно омогућити раздвајање ова два домена и постићи независност домена проблема од једног или више начина реализација које припадају домену рјешења. Како би се раздвојила спецификација софтверских захтјева од информација везаних за кориснички интерфејс, модел на основу кога ће бити вршено аутоматско генерисање корисничког интерфејса ће бити специфициран коришћењем два одвојена XML документа. Први XML документ

представља саме случајеве коришћења, и дио модела података на који се случај коришћења односи, и ови подаци се смијештају у датотеку *useCase.xml*. Други *XML* документ представља информације које се односе на елементе графичког корисничког интерфејса, и сваки елемент овог документа референцира одређени елемент из дефинисаног *useCase.xml* документа. Садржај другог документа специфицира се у датотеци *useCase_gui_extension.xml.* Постојање другог документа није обавезано, јер се у случају да се други документ не креира, за описану спецификацију случајева коришћења користе подразумијевана подешавања свих елемената специфицираних у *useCase.xml* датотеци (шаблони корисничког интерфејса који се користе за приказ, графичке компоненте, и сл.). Ако се жели креирање корисничког интерфејса који се разликује од подразумијевано коришћених подешавања, ова подешавања се специфицирају у *useCase_gui_extension.xml* датотеци.

XML документ који је креиран у складу са синтаксним правилима је формално исправно креиран (*Well Formed*). Осим формалне исправности, исправност *XML* документа се може контролисати у односу на специфицирану шему [Vlajić08]. Шема одређује које елементе и атрибуте одређени *XML* документ смије да садржи, као и редослијед и број тих елемената. Шема се дефинише као засебан документ, а сваки *XML* документ који је базиран на одређеној шеми у заглављу треба да специфицира о којој се шеми ради.

За прављење и контролу исправности модела на основу кога ће бити вршено аутоматско генерисање програмског кода корисничког интерфејса, који су специфицирани у *XML* формату, направљене су двије *XML* шеме: једна за креирање *useCase.xml* (*UseCaseModel.xsd*), а друга за *useCase_gui_extension.xml* датотеку (*UseCaseGUIExtension.xsd*). Заједно, ове двије шеме представљају ентитете и везе између ентитета садржаних у приказаном мета-моделу. У наставку ће бити приказана дефиниција поменутих *XML* шема са описом појединачних елемената и атрибута.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="specification" type="specification"/>

 <xs:complexType name="specification">
  <xs:sequence>
    <xs:element name="useCase" type="useCase" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="useCase">
  <xs:sequence>
    <xs:element name="actor" type="actor" maxOccurs="unbounded" minOccurs="0"/>
    <xs:element name="entity" type="entity" />
    <xs:element name="mainScenario" type="mainScenario" minOccurs="1"/>
    <xs:element name="alternativeScenarios" type="alternativeScenario" minOccurs="0"
                maxOccurs="unbounded"/>
    <xs:element name="include" type="include" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="usecaseID" type="xs:ID" use="required"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="precondition" type="xs:string"/>
  <xs:attribute name="postcondition" type="xs:string"/>
 </xs:complexType>

 <xs:complexType name="actor">
  <xs:attribute name="name" type="xs:string"/>
 </xs:complexType>

 <xs:complexType name="scenarioStep">
  <xs:sequence>
    <xs:element name="extends" type="extends" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="stepNumber" type="xs:int" use="required"/>
  <xs:attribute name="action" type="action" use="required"/>
  <xs:attribute name="entity" type="xs:IDREF" use="required" />
 </xs:complexType>

 <xs:complexType name="entity">
  <xs:sequence>
   <xs:element name="attribute" type="attribute" minOccurs="0" maxOccurs="unbounded"/
  </xs:sequence>
  <xs:attribute name="id" type="xs:ID" use="required"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
 </xs:complexType>

 <xs:complexType name="attribute">
  <xs:complexContent>
    <xs:extension base="entity">
     <xs:attribute name="type" type="attributeType" default="STRING" use="optional"/
     <xs:attribute name="identity" type="xs:boolean" default="false" use="optional"/
     <xs:attribute name="required" type="xs:boolean" default="true" use="optional"/>
    </xs:extension>
  </xs:complexContent>
 </xs:complexType>
```

```
<xs:complexType name="mainScenario">
  <xs:sequence>
   <xs:element name="step" type="scenarioStep" maxOccurs="unbounded"
minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="alternativeScenario">
  <xs:sequence>
   <xs:element name="steps" type="scenarioStep" maxOccurs="unbounded"
minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="referenceMainScenarioNumber" type="xs:int" use="required"/>
  <xs:attribute name="precondition" type="xs:string"/>
  <xs:attribute name="errorMessage" type="xs:string"/>
 </xs:complexType>

 <xs:simpleType name="action">
  <xs:restriction base="xs:string">
   <xs:enumeration value="ENTRY"/>
   <xs:enumeration value="SELECTION"/>
   <xs:enumeration value="REQUEST_OPERATION"/>
   <xs:enumeration value="SYSTEM_ACTION"/>
   <xs:enumeration value="OPERATION_REPORT"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:simpleType name="attributeType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="NUMBER"/>
   <xs:enumeration value="STRING"/>
   <xs:enumeration value="DATE"/>
   <xs:enumeration value="LOGICAL"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="include">
  <xs:attribute name="child" type="xs:IDREF" use="required"/>
  <xs:attribute name="joinedByAttribute" type="xs:IDREF" use="required"/>
 </xs:complexType>

 <xs:complexType name="extends">
  <xs:attribute name="useCase" type="xs:IDREF"/>
  <xs:attribute name="joinedByAttribute" type="xs:IDREF" use="required"/>
 </xs:complexType>

</xs:schema>
```

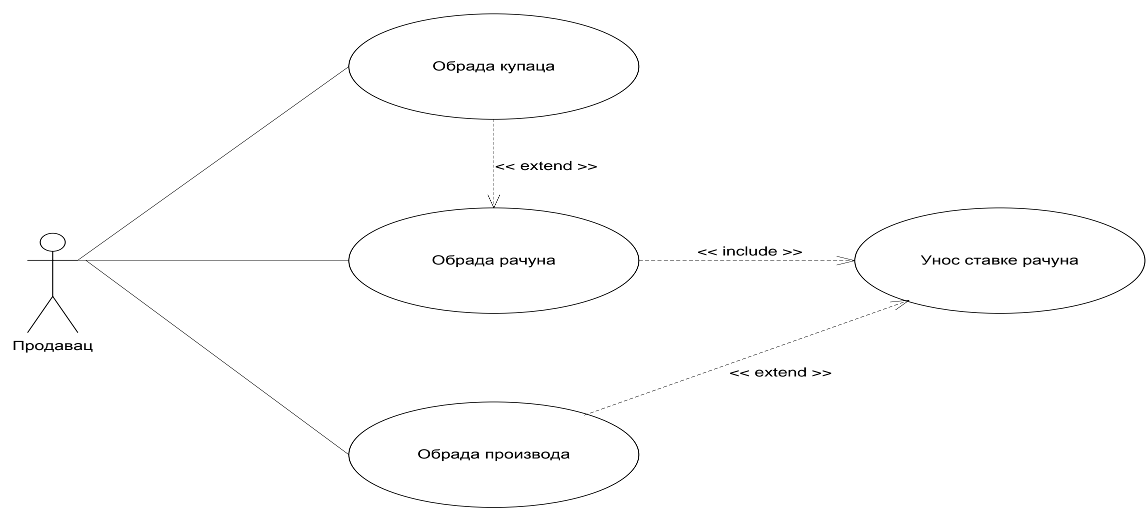
Слика 145: Садржај датотеке *useCaseModel.xsd*

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="useCaseGuiExtensions" type="useCaseGuiExtensions"/>

 <xs:complexType name="useCaseGuiExtensions">
  <xs:sequence>
   <xs:element name="useCaseGuiExtension" type="useCaseGuiExtension"
               minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="useCaseGuiExtension">
  <xs:sequence>
   <xs:element name="scenarioStepGUIExtension" type="scenarioStepGUIExtension"
               maxOccurs="unbounded" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="useCaseID" type="xs:string" use="required"/>
  <xs:attribute name="uiTemplate" type="guiTemplate"/>
  <xs:attribute name="childToParentTemplate" type="childToParentTemplate"/>
  <xs:attribute name="optionButtonsTemplate" type="optionButtonsTemplate"/>
  <xs:attribute name="optionInsert" type="xs:boolean" default="true"/>
  <xs:attribute name="optionUpdate" type="xs:boolean" default="true"/>
  <xs:attribute name="optionDelete" type="xs:boolean" default="true"/>
  <xs:attribute name="optionSearch" type="xs:boolean" default="true"/>
  <xs:attribute name="entityDisplayName" type="xs:string"/>
  <xs:attribute name="entityCodeName" type="xs:string"/>
  <xs:attribute name="ignoreValidation" type="xs:boolean"/>
 </xs:complexType>

 <xs:complexType name="scenarioStepGUIExtension">
  <xs:attribute name="attributeCodeName" type="xs:string"/>
  <xs:attribute name="attributeDisplayName" type="xs:string"/>
  <xs:attribute name="mainScenarioStepNumber" type="xs:int" use="required"/>
  <xs:attribute name="component" type="guiComponent" />
  <xs:attribute  name="format" type="xs:string" use="optional"/>
  <xs:attribute name="visibleAttributes">
   <xs:simpleType>
    <xs:list itemType="xs:string"/>
   </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="showDetails" type="xs:boolean" default="true"/>
  <xs:attribute name="showDetailsTemplate" type="showDetailsTemplate"
                default="BUTTON"/>
  <xs:attribute name="ignoreValidation" type="xs:boolean"/>
 </xs:complexType>

 <xs:simpleType name="guiTemplate">
  <xs:restriction base="xs:string">
   <xs:enumeration value="UI_FIELD_TYPE"/>
   <xs:enumeration value="UI_TABLE_TYPE"/>
  </xs:restriction>
 </xs:simpleType>
```

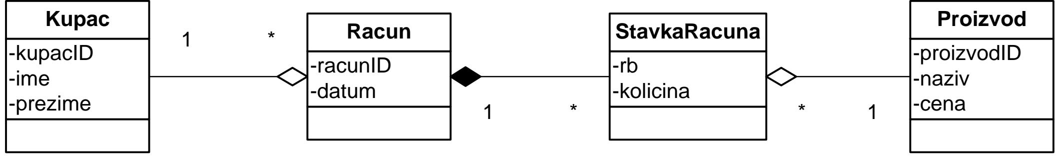
```xml
    <xs:simpleType name="childToParentTemplate">
        <xs:restriction base="xs:string">
            <xs:enumeration value="CHILD_NOT_CHILD"/>
            <xs:enumeration value="CHILD_TAB_TYPE"/>
            <xs:enumeration value="CHILD_FORM_TYPE"/>
            <xs:enumeration value="CHILD_KNGUI_TYPE"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="optionButtonsTemplate">
        <xs:restriction base="xs:string">
            <xs:enumeration value="OPTION_BUTTONS"/>
            <xs:enumeration value="TOOLBAR"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="guiComponent">
        <xs:restriction base="xs:string">
            <!--YesNoType-->
            <xs:enumeration value="CheckBox"/>
            <xs:enumeration value="YesNoButton"/>
            <xs:enumeration value="YesNoRadioButtons"/>
            <xs:enumeration value="YesNoComboBox"/>
            <!--Choice-->
            <xs:enumeration value="ComboBox"/>
            <xs:enumeration value="ListBox"/>
            <xs:enumeration value="RadioButtons"/>
            <xs:enumeration value="Table"/>
            <xs:enumeration value="AutoCompleteComboBox"/>
            <!--SimpleEntry-->
            <xs:enumeration value="TextArea"/>
            <xs:enumeration value="TextField"/>
            <xs:enumeration value="Password"/>
            <xs:enumeration value="InputMask"/>
            <!--Number-->
            <xs:enumeration value="Spinner"/>
            <xs:enumeration value="KeyPadNumber"/>
            <!--date-->
            <xs:enumeration value="Calendar"/>
            <xs:enumeration value="DateTextField"/>
            <xs:enumeration value="DateTextFieldComposition"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="showDetailsTemplate">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RIGHT_CLICK"/>
            <xs:enumeration value="BUTTON"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Слика 146: Садржај датотеке *useCaseGuiExtension.xsd*

У наставку ће бити приказан примјер конкретне спецификације која је формирана у складу са приказаним *XML* шемама, односно на основу *SilabUI* мета-модела (Слика 149, Слика 150, Слика 151). Примјер који слиједи односи се на случај коришћења Обрада рачуна, који је повезан са случајевима коришћења Унос ставке рачуна, Обрада купаца и Обрада производа. Због лакшег праћења улазне спецификације на сљедећем УМЛ дијаграму случајева коришћења (Слика 147) приказан је однос између поменутих случајева коришћења, а након тога и дио концептуалног модела (Слика 148) на који се приказани случајеви коришћења односе.



Слика 147: УМЛ дијаграм случајева коришћења за примјер конкретне улазне спецификације



Слика 148: Дио концептуалног дијаграма за примјер конкретне улазне спецификације

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<specification
 xsi:noNamespaceSchemaLocation="http://silab.fon.rs/silabui/useCaseModel.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <useCase usecaseID="UCRacun0" name="Racun" precondition="Nema preduslova"
         postcondition="Nema postuslova">
  <actor name="Prodavac"/>
  <entity id="Racun1" name="Racun">
   <attribute type="NUMBER" identity="true" id="racunID2" name="racunID"/>
   <attribute type="DATE" id="datum3" name="datum"/>
   <attribute type="NUMBER" id="kupacID4" name="kupacID"/>
  </entity>
  <mainScenario>
   <step stepNumber="1" action="ENTRY" entity="racunID2"/>
   <step stepNumber="2" action="ENTRY" entity="datum3"/>
   <step stepNumber="3" action="SELECTION" entity="kupacID4">
    <extends useCase="UCKupac11" joinedByAttribute="kupacID13"/>
   </step>
   <step stepNumber="4" action="REQUEST_OPERATION" entity="Racun1"/>
   <step stepNumber="5" action="SYSTEM_ACTION" entity="Racun1"/>
   <step stepNumber="6" action="OPERATION_REPORT" entity="Racun1"/>
  </mainScenario>
  <include child="UCStavka Racuna5" joinedByAttribute="racunID7"/>
 </useCase>

 <useCase usecaseID="UCStavka Racuna5" name="Stavka Racuna" precondition="Nema
         preduslova" postcondition="Nema postuslova">
  <entity id="StavkaRacuna6" name="StavkaRacuna">
   <attribute type="NUMBER" id="racunID7" name="racunID"/>
   <attribute type="NUMBER" identity="true" id="rb8" name="rb"/>
   <attribute type="NUMBER" id="proizvodID9" name="proizvodID"/>
   <attribute type="NUMBER" id="kolicina10" name="kolicina"/>
  </entity>
  <mainScenario>
   <step stepNumber="1" action="ENTRY" entity="rb8"/>
   <step stepNumber="2" action="SELECTION" entity="proizvodID9">
    <extends useCase="UCProizvod16" joinedByAttribute="proizvodID18"/>
   </step>
   <step stepNumber="3" action="ENTRY" entity="kolicina10"/>
   <step stepNumber="4" action="REQUEST_OPERATION" entity="StavkaRacuna6"/>
   <step stepNumber="5" action="SYSTEM_ACTION" entity="StavkaRacuna6"/>
   <step stepNumber="6" action="OPERATION_REPORT" entity="StavkaRacuna6"/>
  </mainScenario>
 </useCase>
```

```
  <useCase usecaseID="UCKupac11" name="Kupac" precondition="Nema preduslova"
           postcondition="Nema postuslova">
   <entity id="Kupac12" name="Kupac">
    <attribute type="NUMBER" identity="true" id="kupacID13" name="kupacID"/>
    <attribute id="ime14" name="ime"/>
    <attribute id="prezime15" name="prezime"/>
   </entity>
   <mainScenario>
    <step stepNumber="1" action="ENTRY" entity="kupacID13"/>
    <step stepNumber="2" action="ENTRY" entity="ime14"/>
    <step stepNumber="3" action="ENTRY" entity="prezime15"/>
    <step stepNumber="4" action="REQUEST_OPERATION" entity="Kupac12"/>
    <step stepNumber="5" action="SYSTEM_ACTION" entity="Kupac12"/>
    <step stepNumber="6" action="OPERATION_REPORT" entity="Kupac12"/>
   </mainScenario>
  </useCase>

  <useCase usecaseID="UCProizvod16" name="Proizvod" precondition="Nema
           preduslova" postcondition="Nema postuslova">
   <entity id="Proizvod17" name="Proizvod">
    <attribute type="NUMBER" identity="true" id="proizvodID18"
               name="proizvodID"/>
    <attribute id="naziv19" name="naziv"/>
    <attribute type="NUMBER" id="cena20" name="cena"/>
   </entity>
   <mainScenario>
    <step stepNumber="1" action="ENTRY" entity="proizvodID18"/>
    <step stepNumber="2" action="ENTRY" entity="naziv19"/>
    <step stepNumber="3" action="ENTRY" entity="cena20"/>
    <step stepNumber="4" action="REQUEST_OPERATION" entity="Proizvod17"/>
    <step stepNumber="5" action="SYSTEM_ACTION" entity="Proizvod17"/>
    <step stepNumber="6" action="OPERATION_REPORT" entity="Proizvod17"/>
   </mainScenario>
  </useCase>
</specification>
```

Слика 149: Примјер улазне спецификације (*useCase_ObradaRcuna.xml*) за случај коришћења Обрада рачуна

```
<?xml version="1.0" encoding="UTF-8"?>
<useCaseGuiExtensions
 xsi:noNamespaceSchemaLocation="http://silab.fon.rs/silabui/useCaseGuiExtension.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <useCaseGuiExtension
        useCaseID="UCRacun"
        uiTemplate="UI_FIELD_TYPE"
        childToParentTemplate="CHILD_NOT_CHILD">
       <scenarioStepGUIExtension mainScenarioStepNumber="3"
                                 visibleAttributes="ime prezime"/>
    </useCaseGuiExtension>
    <useCaseGuiExtension
        useCaseID="UCStavkaRacuna"
        uiTemplate="UI_TABLE_TYPE"
        childToParentTemplate="CHILD_TAB_TYPE">
       <scenarioStepGUIExtension mainScenarioStepNumber="2"
                                 visibleAttributes="naziv"/>
    </useCaseGuiExtension>
</useCaseGuiExtensions>
```

Слика 150: Примјер проширења улазне спецификације (*useCaseGuiExtension_ObradaRcuna.xml*) за случај коришћења Обрада рачуна

На примјеру улазне спецификације за случај коришћења Обрада рачуна (Слика 149), као и проширења која се тичу информација везаних за кориснички интерфејс (Слика 150), може се уочити коришћење елемената дефинисаних *XML* шемама, а које су у сагласју са приказаним мета-моделом. Како је раније наглашено, проширење улазне спецификације није неопходно за генерисање програмског кода, али без овог дијела спецификације кориснички интерфејс би се генерисао на основу подразумијеваних вриједности елемената дефинисаних *useCaseGuiExtension.xsd* датотеком. Такође, чак и у случају потребе за специфицирањем информација које се односе на кориснички интерфејс, могуће је специфицирати само оне елементе за које се жели да садрже специфична подешавања у односу на подразумијевана. То се може видјети на приказаном проширењу спецификације (Слика 151), гдје се специфичности тичу само одређених елемената везаних за два случаја коришћења и то само за по један корак у основном сценарију, док основна спецификација (Слика 149) садржи четири случаја коришћења са комплетним основним сценаријима. На овај начин је знатно олакшано формирање улазне спецификације, а у наставку (Слика

151) ће бити приказано комплетно проширење улазне спецификације са свим елементима које је могуће специфицирати којима су додијељене подразумијеване вриједности. Сва проширења се референцирају на основу спецификације коришћењем `useCaseID` односно `mainScenarioStepNumber` атрибута.

```
<?xml version="1.0" encoding="UTF-8"?>

<useCaseGuiExtensions
 xsi:noNamespaceSchemaLocation="http://silab.fon.rs/silabui/useCaseGuiExtension.xsd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
>
<extensions>
 <useCaseGuiExtension useCaseID="UCRacun0"
                      uiTemplate="UI_FIELD_TYPE"
                      childToParentTemplate="CHILD_NOT_CHILD"
                      optionButtonsTemplate="OPTION_BUTTONS"
                      optionInsert="true"
                      optionUpdate="true"
                      optionDelete="true"
                      optionSearch="true"
                      entityDisplayName="Racun"
                      entityCodeName="Racun"
                      ignoreValidation="false">
    <scenarioStepGUIExtension attributeCodeName="racunID"
                              attributeDisplayName="racunID"
                              mainScenarioStepNumber="1"
                              component="TextField"/>
    <scenarioStepGUIExtension attributeCodeName="datum"
                              attributeDisplayName="datum"
                              mainScenarioStepNumber="2"
                              component="TextField"/>
    <scenarioStepGUIExtension attributeCodeName="kupacID"
                              attributeDisplayName="kupacID"
                              mainScenarioStepNumber="3"
                              component="ComboBox"
                              visibleAttributes="ime prezime"
                              showDetails="true"
                              showDetailsTemplate="BUTTON"/>
    <scenarioStepGUIExtension attributeCodeName="Racun"
                              attributeDisplayName="Racun"
                              mainScenarioStepNumber="4"/>
    <scenarioStepGUIExtension attributeCodeName="Racun"
                              attributeDisplayName="Racun"
                              mainScenarioStepNumber="5"/>
    <scenarioStepGUIExtension attributeCodeName="Racun"
                              attributeDisplayName="Racun"
                              mainScenarioStepNumber="6"/>
 </useCaseGuiExtension>
```

```
<useCaseGuiExtension useCaseID="UCStavka Racuna5"
                     uiTemplate="UI_TABLE_TYPE"
                     childToParentTemplate="CHILD_TAB_TYPE"
                     optionButtonsTemplate="OPTION_BUTTONS"
                     optionInsert="true"
                     optionUpdate="true"
                     optionDelete="true"
                     optionSearch="true"
                     entityDisplayName="StavkaRacuna"
                     entityCodeName="StavkaRacuna"
                     ignoreValidation="false">
   <scenarioStepGUIExtension attributeCodeName="rb"
                             attributeDisplayName="rb"
                             mainScenarioStepNumber="1"
                             component="TextField"/>
   <scenarioStepGUIExtension attributeCodeName="proizvodID"
                             attributeDisplayName="proizvodID"
                             mainScenarioStepNumber="2"
                             component="ComboBox"
                             visibleAttributes="naziv"
                             showDetails="true"
                             showDetailsTemplate="BUTTON"/>
   <scenarioStepGUIExtension attributeCodeName="kolicina"
                             attributeDisplayName="kolicina"
                             mainScenarioStepNumber="3"
                             component="TextField"/>
   <scenarioStepGUIExtension attributeCodeName="StavkaRacuna"
                             attributeDisplayName="StavkaRacuna"
                             mainScenarioStepNumber="4"/>
   <scenarioStepGUIExtension attributeCodeName="StavkaRacuna"
                             attributeDisplayName="StavkaRacuna"
                             mainScenarioStepNumber="5"/>
   <scenarioStepGUIExtension attributeCodeName="StavkaRacuna"
                             attributeDisplayName="StavkaRacuna"
                             mainScenarioStepNumber="6"/>
</useCaseGuiExtension>
<useCaseGuiExtension useCaseID="UCKupac11"
                     uiTemplate="UI_FIELD_TYPE"
                     childToParentTemplate="CHILD_NOT_CHILD"
                     optionButtonsTemplate="OPTION_BUTTONS"
                     entityDisplayName="Kupac"
                     entityCodeName="Kupac">
  <scenarioStepGUIExtension attributeCodeName="kupacID"
                            attributeDisplayName="kupacID"
                            mainScenarioStepNumber="1"
                            component="TextField"/>
  <scenarioStepGUIExtension attributeCodeName="ime"
                            attributeDisplayName="ime"
                            mainScenarioStepNumber="2"
                            component="TextField"/>
  <scenarioStepGUIExtension attributeCodeName="prezime"
                            attributeDisplayName="prezime"
                            mainScenarioStepNumber="3"
                            component="TextField"/>
```

```
    <scenarioStepGUIExtension attributeCodeName="Kupac"
                              attributeDisplayName="Kupac"
                              mainScenarioStepNumber="4"/>
    <scenarioStepGUIExtension attributeCodeName="Kupac"
                              attributeDisplayName="Kupac"
                              mainScenarioStepNumber="5"/>
    <scenarioStepGUIExtension attributeCodeName="Kupac"
                              attributeDisplayName="Kupac"
                              mainScenarioStepNumber="6"/>
  </useCaseGuiExtension>

  <useCaseGuiExtension useCaseID="UCProizvod16"
                       uiTemplate="UI_FIELD_TYPE"
                       childToParentTemplate="CHILD_NOT_CHILD"
                       optionButtonsTemplate="OPTION_BUTTONS"
                       entityDisplayName="Proizvod"
                       entityCodeName="Proizvod">
    <scenarioStepGUIExtension attributeCodeName="proizvodID"
                              attributeDisplayName="proizvodID"
                              mainScenarioStepNumber="1"
                              component="TextField"/>
    <scenarioStepGUIExtension attributeCodeName="naziv"
                              attributeDisplayName="naziv"
                              mainScenarioStepNumber="2"
                              component="TextField"/>
    <scenarioStepGUIExtension attributeCodeName="cena"
                              attributeDisplayName="cena"
                              mainScenarioStepNumber="3"
                              component="TextField"/>
    <scenarioStepGUIExtension attributeCodeName="Proizvod"
                              attributeDisplayName="Proizvod"
                              mainScenarioStepNumber="4"/>
    <scenarioStepGUIExtension attributeCodeName="Proizvod"
                              attributeDisplayName="Proizvod"
                              mainScenarioStepNumber="5"/>
    <scenarioStepGUIExtension attributeCodeName="Proizvod"
                              attributeDisplayName="Proizvod"
                              mainScenarioStepNumber="6"/>
  </useCaseGuiExtension>
 </extensions>
```

Слика 151: Примјер проширења улазне спецификације (*useCaseGuiExtension_ObradaRcuna.xml*) за случај коришћења Обрада рачуна са спецификацијом свих могућих елемената дефинисаних шемом

Слиједи приказ свих елемената дефинисаних *XML* шемама са објашњењем улоге сваког елемента у процесу генерисања програмског кода корисничког интерфејса.

***useCaseModel.xsd* датотека**

`specification`

Елемемент `specification` представља основни елемент *XML* датотека формираних на основу *useCaseModel.xsd* шеме. Елемент `specification` може садржати више елемената `useCase` који могу бити међусобно повезани.

`useCase`

Елемент *useCase* садржи информације о једном конкретном случају коришћења. Садржи сљедеће атрибуте:

`usecaseID` – који је веома важан јер се коришћењем овог атрибута успоставља веза између различитих случајева коришћења приликом дефинисања *include* или *extend* релације, а такође се користи и приликом прављења проширења улазне спецификације за референцирање случаја коришћења на који се проширење односи. Овај атрибут не утиче на карактеристике генерисаног корисничког интерфејса. Специфицирање овог атрибута је обавезно;

`name` – назив случаја коришћења. На корисничком интерфејсу ће назив бити видљив у насловној линији екранске форме односно дијалога или таба (картице), у зависности од одабира шаблона као и врсте апликације. Специфицирање овог атрибута је обавезно;

`precondition` – предуслов случаја коришћења. У тренутној верзији *SilabUI* пројекта, овај атрибут се не користи за генерисање програмског кода корисничког интерфејса, већ се у текстуалном облику уноси у оквиру спецификације сваког случаја

коришћења и информативног је карактера. Планирано је да се у наредним фазама пројекта овај атрибут уноси у облику формалног језика за опис ограничења (нпр. *OCL – Object Constraint Language* [OCL]), како би се на основу овакве формалне спецификације могла генерисати правила која би омогућавала извршење основног сценарија случаја коришћења. Овај атрибут не утиче на карактеристике генерисаног корисничког интерфејса. Специфицирање овог атрибута није обавезно;

`postcondition` – постуслов случаја коришћења. У тренутној верзији *SilabUI* пројекта, слично `precondition` атрибуту, овај атрибут се не користи за генерисање програмског кода корисничког интерфејса, већ се у текстуалном облику уноси у оквиру спецификације сваког случаја коришћења и информативног је карактера. Овај атрибут не утиче на карактеристике генерисаног корисничког интерфејса. Специфицирање овог атрибута није обавезно.

Поред приказаних атрибута, елемент `useCase` садржи по један елемент `entity` и `mainScenario`, а може садржати више елемената типа `actor,` `alternativeScenarios` и `include`, који су описани у наставку;

`entity`

Елемент `entity` успоставља везу са моделом података односно са ентитетом модела података на који се односи случај коришћења. Садржи два атрибута:

`id –` који служи за референцирање ентитета приликом спецификације корака сценарија случаја коришћења. Овај атрибут не утиче на карактеристике генерисаног корисничког интерфејса. Специфицирање овог атрибута је обавезно;

`name –` назив ентитета. Овај атрибут не утиче директно на карактеристике генерисаног корисничког интерфејса, али уколико се не направи проширење спецификације вриједност овог атрибута се подразумијевано користи као назив ентитета приликом генерисања кода корисничког интерфејса, углавном за

приказивање порука о успјешности извршења операције над ентитетом. Специфицирање овог атрибута је обавезно.

Поред ова два атрибута, елемент `entity` може садржати више елемената типа `attribute` којим се дефинишу атрибути који припадају ентитету;

`attribute`

Елемент `attribute` омогућава спецификацију информација везаних за одређени атрибут ентитета. Овај елемент је XML шемом дефинисан као елемент који насљеђује елемент `entity`[69] и на тај начин преузима све карактеристике ентитета које проширује са три додатна атрибута:

`type` – тип атрибута. Овај атрибут утиче на генерисање корисничког интерфејса јер, у складу са разматрањима и закључцима шестог поглавља, од њега може зависити одабир графичке компоненте којом ће бити омогућена манипулација подацима везаним за одабрани атрибут. Тип атрибута може бити: `STRING`, `NUMBER`, `DATE` или `LOGICAL`. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `STRING`;

`identity` – коришћењем овог атрибута специфицира се да ли је атрибут идентификатор ентитета. Овај атрибут може утицати на генерисање корисничког интерфејса уколико идентификатор није потребно приказивати кориснику на корисничком интерфејсу (најчешће ако се вриједност овог атрибута аутоматски

[69] Одлука да у шеми елемент `attribute` насљеђује елемент `entity` је произишла из чињенице да се корак сценарија може односити на одређени атрибут ентитета, али се може односити и на читав ентитет. Претходне верзије ове шеме нису на овај начин дефинисале однос између ентитета и атрибута, али је због тога процес креирања конкретне улазне спецификације на основу тих шема био знатно сложенији. Дакле, и поред тога што се оваква одлука може сматрти дискутабилном, она се показала оправданом због лакоће креирања и боље прегледности улазне спецификације.

генерише у бази података или на неки други начин). Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `false`;

`required` – коришћењем овог атрибута специфицира се да ли атрибут мора да буде попуњен приликом извршења системских операција. Овај атрибут утиче на генерисање корисничког интерфејса јер се у зависности од њега генерише програмски код за валидацију улазних података, као и пратеће поруке о грешкама. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `true`.

`mainScenario`

Елемент `mainScenario` омогућава спецификацију секвенце корака основног сценарија који се дефинишу елементом `scenarioStep;`

`scenarioStep`

Елемент `scenarioStep` омогућава спецификацију информација везаних за корак сценарија случаја коришћења. Овај елемент се специфицира коришћењем три атрибута:

`stepNumber` – број корака у сценарију. Овај атрибут утиче на генерисање корисничког интерфејса јер, у складу са разматрањима и закључцима шестог поглавља, од њега ће зависити редослијед графичких компоненти на екранској форми. Такође, важно је напоменути да се коришћењем вриједности овог атрибута референцира евентуално проширење спецификације. Специфицирање овог атрибута је обавезно;

`action` – тип акције корака сценарија случаја коришћења. Овај атрибут утиче на генерисање корисничког интерфејса јер, у складу са разматрањима и закључцима трећег поглавља, од њега може зависити одабир графичке компоненте којом ће бити омогућена манипулација подацима везаним за одабрани атрибут. Тип акције може

бити: `ENTRY`, `SELECTION`, `REQUEST_OPERATION`, `SYSTEM_ACTION` или `OPERATION_REPORT`. Специфицирање овог атрибута је обавезно;

`entity` – референца на ентитет, односно атрибут ентитета, на који се односи корак сценарија. Коришћењем ове референце преко корака у сценарију може се приступити свим вриједностима које су специфициране за одређени ентитет, односно атрибут ентитета из модела, што може бити од помоћи приликом одабира графичке компоненте. Специфицирање овог атрибута је обавезно.

Поред ова три атрибута, елемент `scenarioStep` омогућава успостављање везе *extends* између различитих случајева коришћења. Ова веза се специфицира коришћењем елемента `extends`;

`extends`

Елемент `extends` омогућава успостављање релације *extends* између текућег случаја коришћења и неког другог случаја коришћења који је специфициран у оквиру исте спецификације (нпр. Обрада рачуна и Обрада купаца, или Унос ставке рачуна и Обрада производа). Коришћењем само два атрибута омогућено је да се повежу два постојећа случаја коришћења. Потребно је специфицирати:

`useCase` – референца (`useCaseID`) на случај коришћења чији се сценарио може (а не мора) покренути током извршења текућег корака сценарија текућег случаја коришћења. Овај атрибут утиче на генерисање корисничког интерфејса јер је на тај начин омогућено генерисање програмског кода за функционалност *Show details* која је објашњена у шестом поглављу. Специфицирање овог атрибута је обавезно;

`joinedByAttribute` – референца на идентификатор атрибута који успоставља везу између одговарајућих ентитета модела података на које се односе повезани случајеви коришћења. Овај атрибут има улогу сличну спољном кључу у релационом моделу. Специфицирање овог атрибута је обавезно.

`actor`

Елемент `actor` се више пута може наћи у оквиру једног случаја коришћења (елемента `useCase`). Садржи само један атрибут:

`name –` назив актора. Специфицирање овог атрибута је обавезно.

`alternativeScenarios`

Елемент `alternativeScenarios` се више пута може наћи у склопу једног случаја коришћења (елемента `useCase`). Дефинисан је типом `alternativeScenario` који слично елементу `mainScenario` садржи више појављивања елемента `scenarioStep`, али поред тога, могуће је специфицирати три додатна атрибута:

`referencedMainScenarioNumber –` редни број корака основног сценарија на који се односи алтернативни сценарио. Специфицирање овог атрибута је обавезно;

`precondition –` предуслов за покретање алтернативног сценарија. У тренутној верзији *SilabUI* пројекта, слично `precondition` атрибуту елемента `useCase`, овај атрибут се не користи за генерисање програмског кода корисничког интерфејса, већ се у текстуалном облику уноси у оквиру спецификације сваког алтернативног сценарија и информативног је карактера. Овај атрибут не утиче на карактеристике генерисаног корисничког интерфејса. Специфицирање овог атрибута није обавезно.

`errorMessage –` Текст поруке који ће бити приказан кориснику уколико дође до покретања алтернативног сценарија. Специфицирање овог атрибута није обавезно.

`include`

Елемент `include` омогућава успостављање релације *include* између текућег случаја коришћења и неког другог случаја коришћења који је специфициран у оквиру исте спецификације (нпр. Обрада рачуна и Унос ставке рачуна). Може се јавити више пута у оквиру једног случаја коришћења (елемента `useCase`). Коришћењем само два

атрибута омогућено је да се повежу два постојећа случаја коришћења везом родитељ-дијете. Потребно је специфицирати:

`child` – референца (`useCaseID`) на случај коришћења чији се сценарио бар једном покреће (веза *include*) током извршења основног сценарија текућег случаја коришћења. Овај атрибут утиче на генерисање корисничког интерфејса јер је на тај начин омогућено генерисање програмског кода за корисничког интерфејса за дјецу ентитете. Специфицирање овог атрибута је обавезно;

`joinedByAttribute` – референца на идентификатор атрибута који успоставља везу између одговарајућих ентитета модела података на које се односе повезани случајеви коришћења. Овај атрибут има улогу сличну спољном кључу у релационом моделу. Специфицирање овог атрибута је обавезно.

useCaseGuiExtension.xsd датотека

`useCaseGuiExtensions`

Елемемент `useCaseGuiExtensions` представља основни елемент *XML* датотека формираних на основу *useCaseGuiExtension.xsd* шеме. Елемент `useCaseGuiExtensions` може садржати више елемената `useCaseGuiExtension` који могу бити међусобно повезани.

`useCaseGuiExtension`

Елемент `useCaseGuiExtension` садржи проширење основне спецификације информацијама које су везане за кориснички интерфејс једног конкретног случаја коришћења. Садржи сљедеће атрибуте:

`usecaseID` – користи се за референцирање случаја коришћења на који се проширење односи. Специфицирање овог атрибута је обавезно;

`uiTemplate` – основни шаблон корисничког интерфејса за референцирани случај коришћења. Могуће је специфицирати двије вриједности: `UI_FIELD_TYPE` и `UI_TABLE_TYPE`. У зависности од одабраног шаблона, кориснички интерфејс ће бити формиран коришћењем поља односно у табеларном облику. Специфицирање овог атрибута није обавезно, а подразумијеване вриједности се одређују приликом генерисања програмског кода корисничког интерфејса у односу на то да ли је референцирани случај коришћења родитељ на врху хијерархије, када се подразумијевано приказује коришћењем поља, односно ако је дијете, подразумијевани начин приказа је коришћењем табеле;

`childToParentTemplate` – начин приказа дијете-ентитета у односу на приказ ентитета родитељ. Могуће је специфицирати једну од следећих вриједности: `CHILD_NOT_CHILD`, `CHILD_FORM_TYPE`, `CHILD_TAB_TYPE` или `CHILD_KNGUI_TYPE`. У зависностти од спецниициране вриједности, овим атрибутом се може специфицирати да се случај коришћења не односи на дијете ентитет (`CHILD_NOT_CHILD`), а уколико се ради о дијете ентитету, кориснички интерфејс за референцирани случај коришћења може бити креиран у засебном прозору у односу на случај коришћења који се односи на ентитет родитељ (`CHILD_FORM_TYPE`), у истом прозору у оквиру таба (картице) (`CHILD_TAB_TYPE`), или унутар исте табеле у којој се обрађују информације о родитељу, како је дефинисано KNGUI шаблоном корисничког интерфејса (`CHILD_KNGUI_TYPE`). Специфицирање овог атрибута није обавезно, а подразумијеване вриједности се одређују приликом генерисања програмског кода корисничког интерфејса у односу на то да ли је референцирани случај коришћења дијете родитеља који је на врху хијерархије, када се подразумијевано приказује коришћењем таба (картице), односно, ако се ради о родитељу на нижем нивоу хијерархије подразумијевани начин приказа је у засебном прозору;

`optionsButtonsTemplate` – начин приказа опција за извршење одабраних CRUD операција, односно организација графичких компоненти које омогућавају позив операција на корисничком интерфејсу. Могуће је специфицирати једну од двије вриједности: `OPTIONS_BUTTONS,` или `TOOLBAR`, у зависности од тога да ли се жели да се ове опције приказују као скуп независних компоненти (обично компоненте дугме – *button*), или груписано у склопу линије са алатима – *toolbar*. Различити начини груписања опција за креирање, ажурирање и брисање података приказани су у шестом поглављу (Слика 135: Различити начини приказивања опција за креирање, ажурирање и брисање података) Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `OPTIONS_BUTTONS;`

`optionInsert` – атрибут је логичког типа и њиме се специфицира да ли ће генерисани кориснички интерфејс садржати функционалност која омогућава креирање нове инстанце ентитета референцираног случаја коришћења. Уколико је назначено да се ова опција генерише, на корисничком интерфејсу ће бити приказана на начин специфициран атрибутом `optionsButtonsTemplate.` Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `true;`

`optionUpdate` – атрибут је логичког типа и њиме се специфицира да ли ће генерисани кориснички интерфејс садржати функционалност која омогућава измјену постојеће инстанце ентитета референцираног случаја коришћења. Уколико је назначено да се ова опција генерише, на корисничком интерфејсу ће бити приказана на начин специфициран атрибутом `optionsButtonsTemplate.` Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `true;`

`optionDelete` – атрибут је логичког типа и њиме се специфицира да ли ће генерисани кориснички интерфејс садржати функционалност која омогућава брисање постојеће инстанце ентитета референцираног случаја коришћења. Уколико

је назначено да се ова опција генерише, на корисничком интерфејсу ће бити приказана на начин специфициран атрибутом `optionsButtonsTemplate.` Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `true;`

`optionSearch` – атрибут је логичког типа и њиме се специфицира да ли ће генерисани кориснички интерфејс садржати функционалност која омогућава претраживање инстанци ентитета референцираног случаја коришћења. Уколико је назначено да се ова опција генерише, на корисничком интерфејсу ће бити приказана на начин специфициран атрибутом `optionsButtonsTemplate.` Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `true;`

`entityDisplayName` – атрибут омогућава специфицирање назива ентитета који се може разликовати од назива специфицираног у основној спецификацији, а за који се жели да буде приказиван кориснику на корисничком интерфејсу. Типичан примјер је ентитет *StavkaRacuna* за који би се дефинисала вриједност атрибута `entityDisplayName` Ставка рачуна. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је вриједност дефинисана као назив ентитета у основној спецификацији`;`

`entityCodeName` – атрибут омогућава специфицирање назива ентитета који се може разликовати од назива специфицираног у основној спецификацији, а за који се жели да буде генерисан и коришћен у програмском коду. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је вриједност дефинисана као назив ентитета у основној спецификацији`;`

`ignoreValidation` – атрибут је логичког типа и њиме се специфицира да ли ће генерисани кориснички интерфејс садржати програмски код који је одговоран за валидацију података на корисничком интерфејсу. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `false.`

Поред наведених атрибута, елемент `useCaseGuiExtension` може садржати више елемената типа `scenarioStepGuiExtension` којим се врши спецификација проширених подешавања за кораке сценарија референцираног случаја коришћења.

`scenarioStepGuiExtension`

Елемент `scenarioStepGuiExtension` садржи проширење основне спецификације информацијама које су везане за кориснички интерфејс једног конкретног корака сценарија случаја коришћења. Садржи сљедеће атрибуте:

`mainScenarioStepNumber` – користи се за референцирање корака сценарија случаја коришћења на који се проширење односи. Специфицирање овог атрибута је обавезно;

`attributeDisplayName` – атрибут омогућава специфицирање назива атрибута атрибута из референцираног корака сценарија случаја коришћења који се може разликовати од назива специфицираног у основној спецификацији, а за који се жели да буде приказиван кориснику на корисничком интерфејсу. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је вриједност дефинисана као назив атрибута у основној спецификацији;

`attributeCodeName` – атрибут омогућава специфицирање назива атрибута из референцираног корака сценарија случаја коришћења који се може разликовати од назива специфицираног у основној спецификацији, а за који се жели да буде генерисан и коришћен у програмском коду. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је вриједност дефинисана као назив атрибута у основној спецификацији;

`component` – атрибут омогућава одабир између више компонената корисничког интерфејса: `CheckBox`, `YesNoButton`, `YesNoRadioButtons`, `YesNoComboBox`, `ComboBox`, `ListBox`, `RadioButtons`, `Table`, `AutoCompleteComboBox`, `TextArea`, `TextField`, `Password`, `InputMask`,

`Spinner`, `KeyPadNumber`, `Calendar`, `DateTextField`, `DateTextFieldComposition`. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност се дефинише приликом генерисања програмског кода, на тај начин што се анализира тип атрибута и врста акције у кораку сценарија, у складу са разматрањима изнесеним у шестом поглављу;

`ignoreValidation` – атрибут је логичког типа и њиме се специфицира да ли ће генерисани кориснички интерфејс садржати програмски код који је одговоран за валидацију унесене вриједности на корисничком интерфејсу која се односи на атрибут из референцираног корака сценарија. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `false;`

`format` – коришћењем овог атрибута специфицира се формат података који се уносе односно приказују на корисничком интерфејсу. Типичан примјер је форматирање приказа датума за који се може специфицирати вриједност „`dd/MM/yyyy`“. Специфицирање овог атрибута није обавезно`;`

`visibleAttributes` – коришћењем овог атрибута специфицира се који ће атрибути бити видљиви приликом приказа листе за одабир између различитих инстанци повезаног ентитета. Овај атрибут се користи искључиво у ситуацији када је тип акције у кораку сценарија `SELECTION`. Типичан примјер је корак сценарија у коме се бира Купац у основном сценарију случаја коришћења Обрада рачуна, би се умјесто шифре купца која представља идентификатор ентитета Купац, за атрибут `visibleAttributes` специфицирали атрибути име и презиме купца. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је атрибут идентификатор референцираног ентитета`;`

`showDetails` – коришћењем овог атрибута специфицира се да ли ће се приликом генерисања кода корисничког интерфејса генерисати код одговоран за

извршење *ShowDetails* функционалности која је описана у шестом поглављу (Слика 116, Слика 117, Слика 118, Слика 119), а која омогућава успостављање релације *Extend* између различитих случајева коришћења. Овај атрибут се користи искључиво у ситуацији када је тип акције у кораку сценарија `SELECTION`. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `true;`

`showDetailsTemplate` – коришћењем овог атрибута специфицира се начин приказа опције која омогућава извршење *ShowDetails* функционалности на корисничком интерфејсу. Могуће је одабрати једну од двије вриједности: `BUTTON` која подразумијева да се уз листу из које се бирају вриједности креира графичка компонента дугме, или `RIGHT_CLICK` која омогућава позив операције коришћењем десног клика миша над компонентом која приказује листу. Овај атрибут се користи искључиво у ситуацији када је тип акције у кораку сценарија `SELECTION`. Специфицирање овог атрибута није обавезно, а подразумијевана вриједност је `BUTTON.`

7.2. Алати за формирање улазне спецификације

У претходном дијелу дата је детаљна спецификација свих елемената који чине улазну спецификацију. Као што је раније напоменуто, улазна спецификација се формира кроз два одвојена *XML* документа, а која морају бити у складу са *XML* шемама, док шеме представљају реализацију *SilabUI* мета-модела. Ставови испитаника по питању начина дефинисања улазне спецификације су подијељени, прије свега између визуелног моделовања и графичког алата који би водио корисника кроз процес дефинисања спецификације, али значајан број испитаних сматра да је

пожељни начин дефинисања улазне спецификације писањем кода доменски специфичног језика развијеног за ту намјену. Са друге стране, посматрани алати за аутоматизацију развоја корисничког интерфејса углавном омогућавају само један начин дефинисања улазне спецификације. *SilabUI* приступ омогућава сва три начина дефинисања улазне спецификације. У наставку ће бити приказани различити начини формирања улазне спецификације.

7.2.1. Директно писање кода улазне спецификације

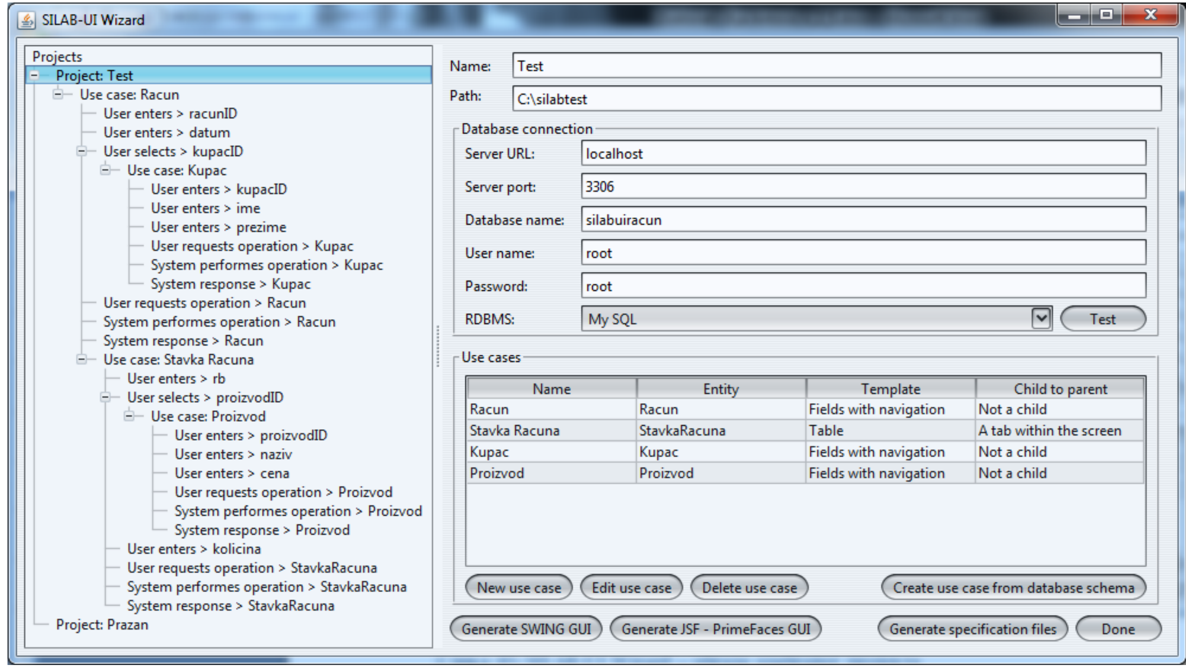
Ако се говори о алату који би омогућио директно писање кода улазне спецификације, може се рећи да је довољан алат најпростији текст едитор, а пожељно је користити неки од алата за обраду *XML* докумената, који често чине саставни дио развојних окружења. Сам процес формирања улазне спецификације на овај начин, своди се на специфицирање конкретних вриједности за елементе и атрибуте елемената који су описани у поглављу 7.1.1. Довољно је формирати основну спецификацију у форми *XML* документа на основу *useCaseModel.xsd* шеме, а по потреби се може формирати и проширење спецификације, такође у форми *XML* документа на основу *useCaseGuiExtension.xsd* шеме. Тако формирана (Слика 149, Слика 150) спецификација може бити обрађена од стране алата за генерисање програмског кода корисничког интерфејса – генератора. Коришћењем дефинисаних *XML* шема добијен је доменски специфични језик за формирање улазне спецификације који је заснован на *XML*-у. Поред тога, у току је израда текстуалног доменски специфичног језика [Savic11] као алтернатива *XML* спецификацији.

7.2.2. Формирање улазне спецификације коришћењем графичког алата (wizard)

Како су резултати испитивања показали да скоро 30% испитаних сматра неопходним постојање графичког алата за формирање улазне спецификације који би

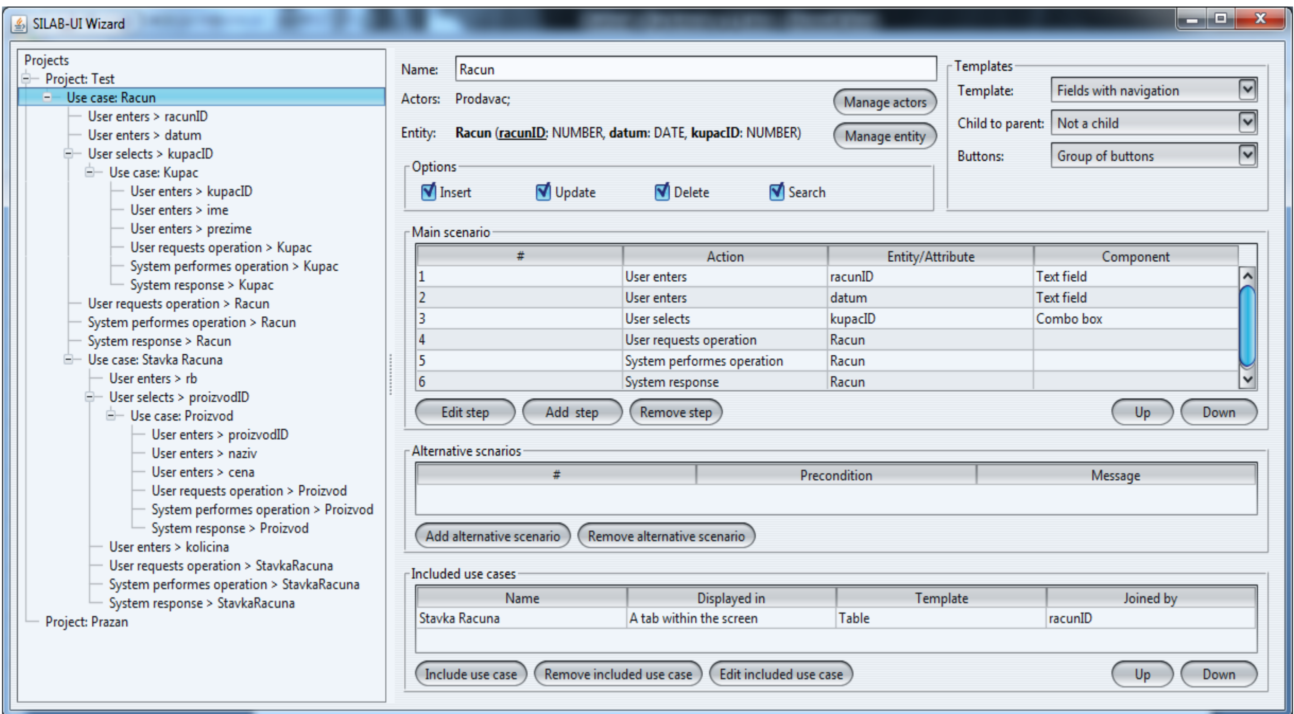
водио корисника кроз процес формирања улазне спецификације, пројектован је и имплементиран алат који од корисника не захтијева добро познавање мета-модела и правила формирања *XML* докумената, а поред тога, смањује могућност прављења грешака.

Алат омогућава спецификацију случајева коришћења, уз дефинисање пратећег модела података и информација које се могу придружити спецификацији случајева коришћења, а које су везане за кориснички интефрејс. Случајеви коришћења су организовани у пројекте, а између себе могу а не морају бити повезани релацијама *include* и *extend.* Сваки пројекат је у позадини повезан са једним паром *XML* докумената (основном и проширеном спецификацијом) који се могу учитати и мијењати уколико су претходно креирани ручно или на неки други начин, а могу се и креирати директно из алата и по потреби даље обрађивати. Корисници алата не морају бити свјесни постојања *XML* докумената јер је из самог алата омогућено директно генерисање програмског кода корисничког интефрејса. Такође, сваком пројекту се може придружити конекција на релациону базу података која се може искористити за аутоматско прављење спецификације на основу мета-података релационе шеме постојеће базе података, о чему ће бити више ријечи касније. У наставку је приказан изглед корисничког интерфејса алата који омогућава обраду одабраног пројекта (Слика 152).



Слика 152: *SilabUI* Wizard – обрада одабраног пројекта

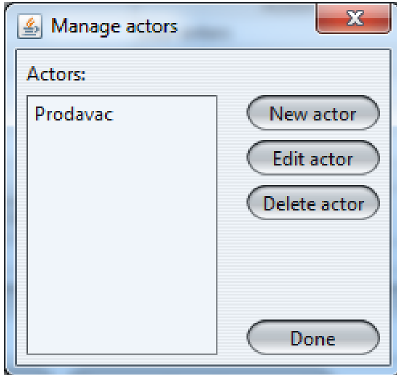
Као што се може видјети са слике (Слика 152) на лијевој страни се налази стабло у којем су приказани активни пројекти, а сваки пројекат може садржати више случајева коришћења. За сваки случај коришћења се у стаблу приказују кораци основног сценарија. Случајеви коришћења који су повезани са текућим случајем коришћења релацијама *extend* или *include* налазе се унутар самог корака сценарија уколико се ради о релацији *extend* или на крају ако се ради о релацији *include*. На овај начин се добија графички приказ који хијерархијски представља везане случајеве коришћења. Одабиром неког случаја коришћења, са десне стране ће се приказати детаљна спецификација са свим функционалностима за обраду случаја коришћења (Слика 153). Свака промјена над случајем коришћења освјежава податке на приказаном стаблу.



Слика 153: *SilabUI* Wizard – обрада одабраног случаја коришћења

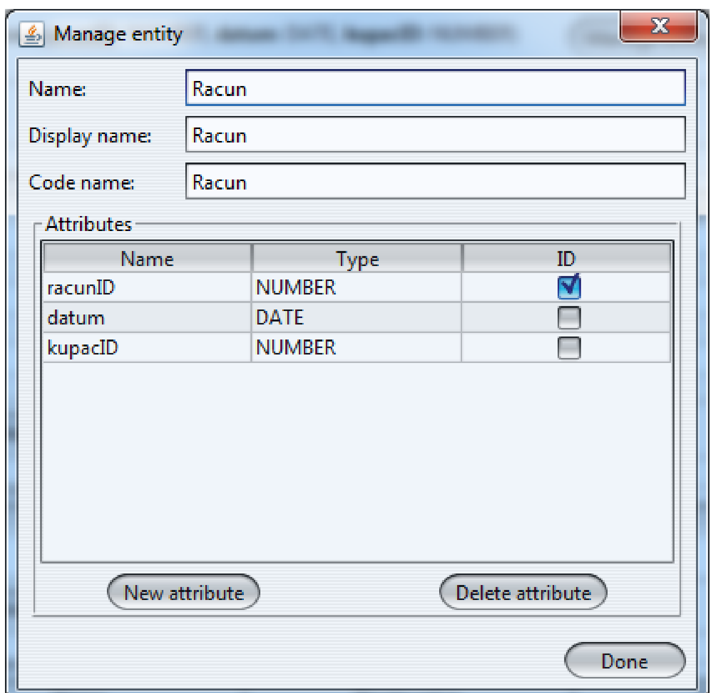
Приказани кориснички интерфејс омогућава дефинисање:

- назива случаја коришћења,
- актора који користе случај коришћења (кликом на дугме *Manage actors,* Слика 154),



Слика 154: *SilabUI* Wizard – обрада Актора

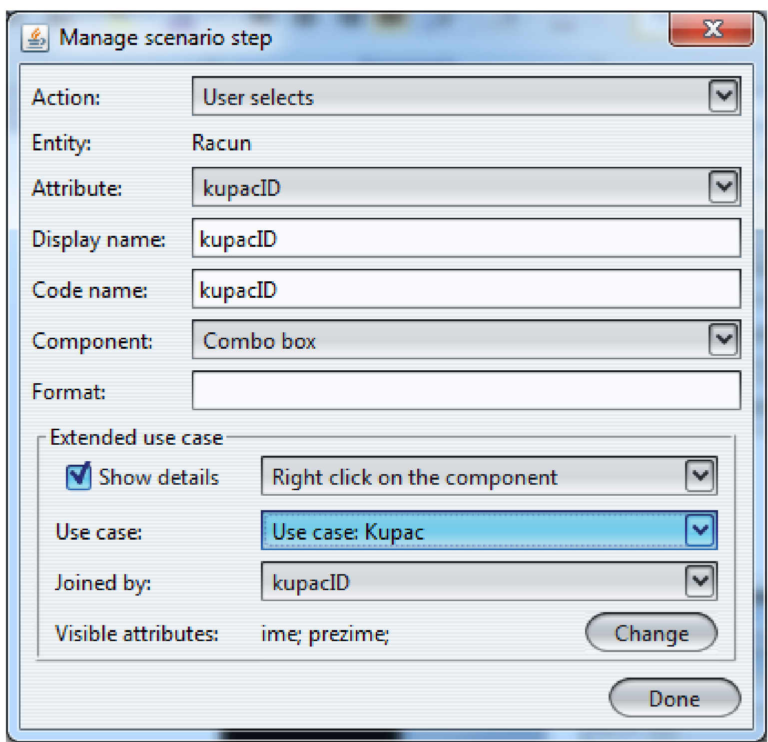
- ентитета који је повезан са случајем коришћења (кликом на дугме *Manage entity,* Слика 155),



Слика 155: *SilabUI* Wizard – обрада ентитета повезаног са случајем коришћења

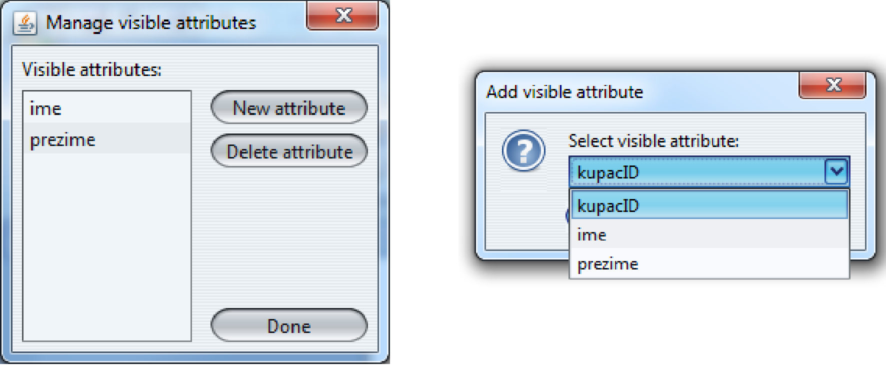
- одабир *CRUD* операција које ће се генерисати,
- одабир шаблона корисничког интерфејса (основног шаблона)
- одабир начина приказа случаја коришћења везаног за ентитет који представља дијете у релацији родитељ-дијете о чему је било ријечи у шестом поглављу),
- одабир шаблона којим ће се приказивати опције за извршење *CRUD* операција
- спецификацију корака основног сценарија случаја коришћења
- спецификацију алтернативних сценарија случаја коришћења
- спецификацију повезивања са другим случајевима коришћења релацијом *include.*

Спецификација корака основног сценарија случаја коришћења врши се у засебном дијалогу који се приказује кликом на дугме *Add step* или *Edit step* (Слика 156). Могуће је одабрати врсту акције, атрибут, назив атрибута који ће се користити за приказ, назив атрибута који ће се користити у програмском коду, графичку компоненту и формат података. У зависности од одабране врсте акције и типа атрибута на које се односи корак основног сценарија, кориснику се приказују различите графичке компоненте које може одабрати.



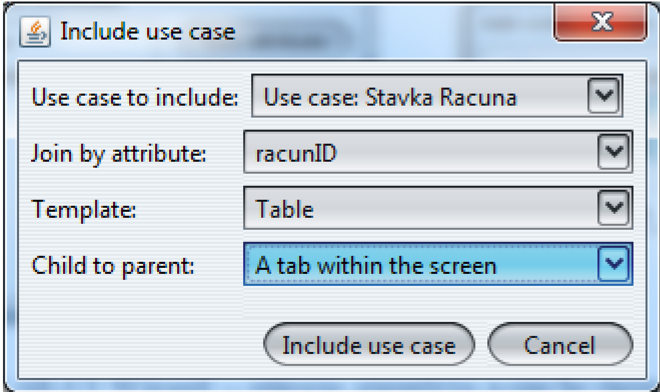
Слика 156: *SilabUI* Wizard – обрада корака основног сценарија случаја коришћења

Поред тога, уколико се одабере акција у којој корисник треба да бира инстанцу ентитета из понуђеног скупа инстанци ентитета, на овом мјесту је могуће омогућити опцију *Show details* и специфицирати на који начин ће се овој опцији приступати на генерисаном корисничком интерфејсу, на који се случај коришћења односи, преко ког атрибута се успоставља веза, а може се дефинисати и листа атрибута који ће бити приказани за сваку инстанцу ентитета у листи из које се врши одабир (Слика 157).



Слика 157: *SilabUI* Wizard – обрада атрибута који ће бити приказивани кориснику за сваку инстанцу ентитета у листи из које се врши одабир

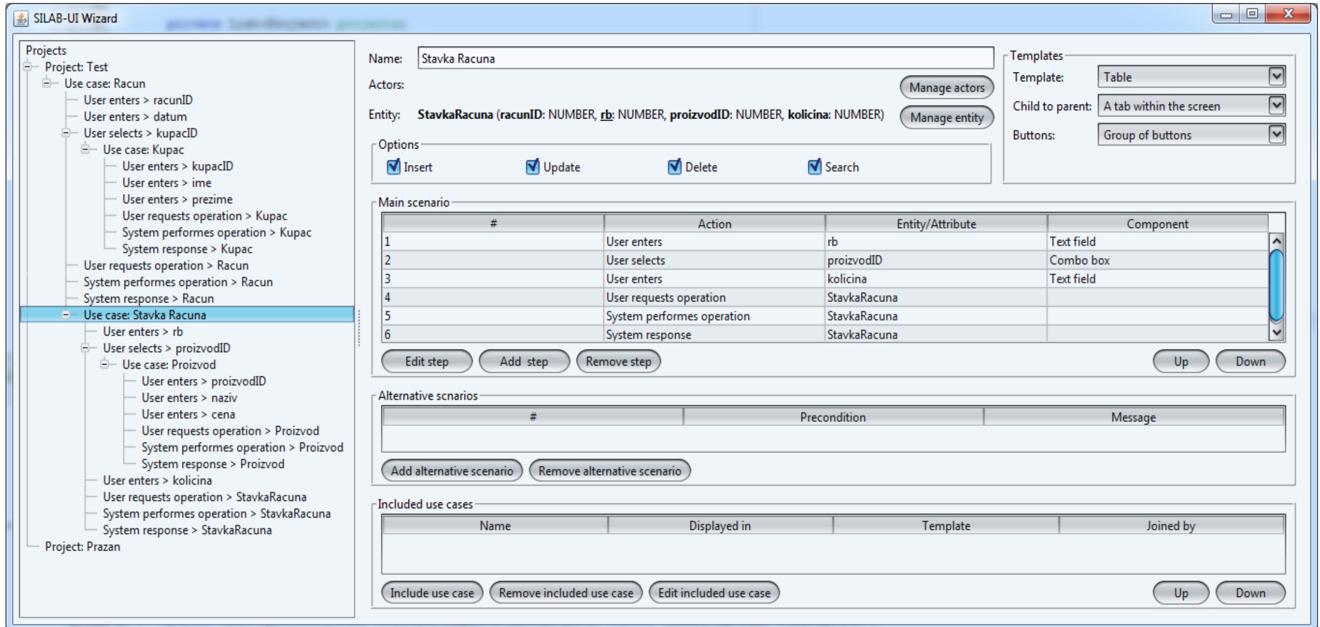
Текући случај коришћења се може повезати са другим случајевима коришћења релацијом *Include* кликом на дугме *Include use case* након чега се отвара дијалог (Слика 158) који омогућава избор случаја коришћења, атрибут преко кога се врши повезивање, основни шаблон корисничког интерфејса, као и начин на који ће се одабрани случај коришћења приказивати у односу на текући случај коришћења.



Слика 158: *SilabUI* Wizard – повезивање текућег случаја коришћења са другим случајем коришћења релацијом *Include*

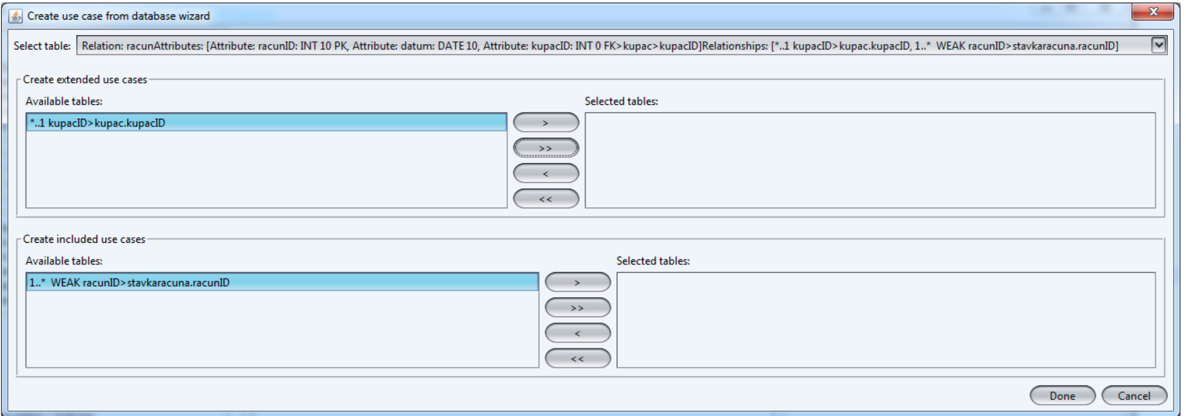
Повезани случај коришћења ће се појавити у стаблу са лијеве стране, а приликом одабира повезаног случаја коришћења могуће га је обрађивати на исти начин, на који је вршена обрада текућег случаја коришћења (Слика 159). Поред тога, приликом одабира постојећег случаја коришћења за повезивање са текућим, алат омогућава да корисник одабере да ли ће случај коришћења бити доступан кориснику

директно или само посредно, кроз текући случај коришћења. Ако се остави могућност да корисник и директно може приступати одабраном случају коришћења, у стаблу ће се одабрани случај коришћења појавити и и склопу текућег случаја коришћења и независно. Оба случаја коришћења се тада могу посебно обрађивати и сваки независно детаљно специфицирати. Приликом генерисања корисничког интерфејса, за оба случаја коришћења ће бити генерисан посебан програмски код.



Слика 159: *SilabUI* Wizard – Обрада случаја коришћења који је повезан релацијом *Include*

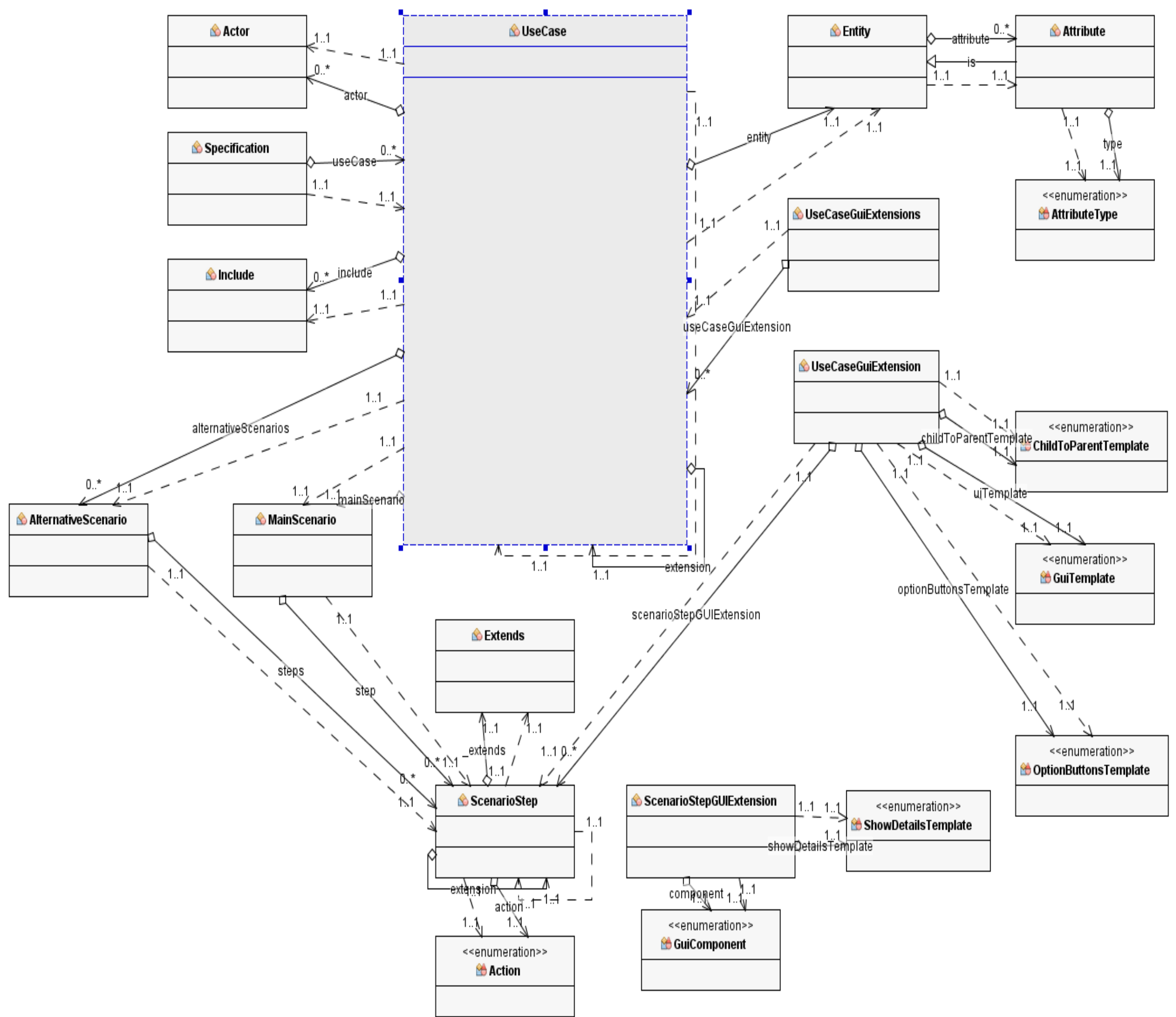
Како је раније поменуто, сваки пројекат може садржати конекцију на постојећу релациону базу података. Овај дио алата је пројектован како би се омогућило коришћење евентуалних већ постојећих дијелова система, у овом случају релационе базе података, приликом формирања улазне спецификације, односно генерисања програмског кода корисничког интерфејса. На овај начин се омогућава кориснику да одабиром релација из шеме базе података (Слика 160), која је учитана коришћењем мета-података, у неколико тренутака креира спецификацију и изгенерише програмски код корисничког интерфејса.



Слика 160: *SilabUI* Wizard – коришћење постојеће шеме базе података за формирање улазне спецификације

На основу одабраних релација (табела) и веза са другим релацијама (*include* или *extend*), формира се спецификација са подразумијеваним вриједностима. Основни сценарио се формира тако што се прави посебан корак за сваки атрибут, а на основу типа податка или везе са другим релацијама (ако је атрибут спољни кључ, а релација је одабрана у листи за *extend*) дефинише се акција сценарија и формирају се и остали потребни случајеви коришћења. Креирани случајеви коришћења се приказују у стаблу текућег пројекта а по потреби се могу обрађивати, некон чега се може генерисати програмски код корисничког интерфејса, или само *XML* документа основне и проширене улазне спецификације. На сличан начин се могу користити и други постојећи дијелови система нпр. ако већ постоје доменски објекти коришћењем рефлексије, или ако постоје пресликавања за одређени алат за објектно релационо пресликавање. Коришћењем ове функционалности се драстично смањује вријеме за израду спецификације, укида редуданса која би постојала уколико би се независно правила шема базе података и улазна спецификација која би засигурно могла довести и до грешака које би се огледале у неподударању модела података који се налази у спецификацији са физичким моделом у релационој бази података.

У наставку је приказан мета-модел који је прилагођен за коришћење у графичком алату за креирање улазне спецификације (Слика 161).

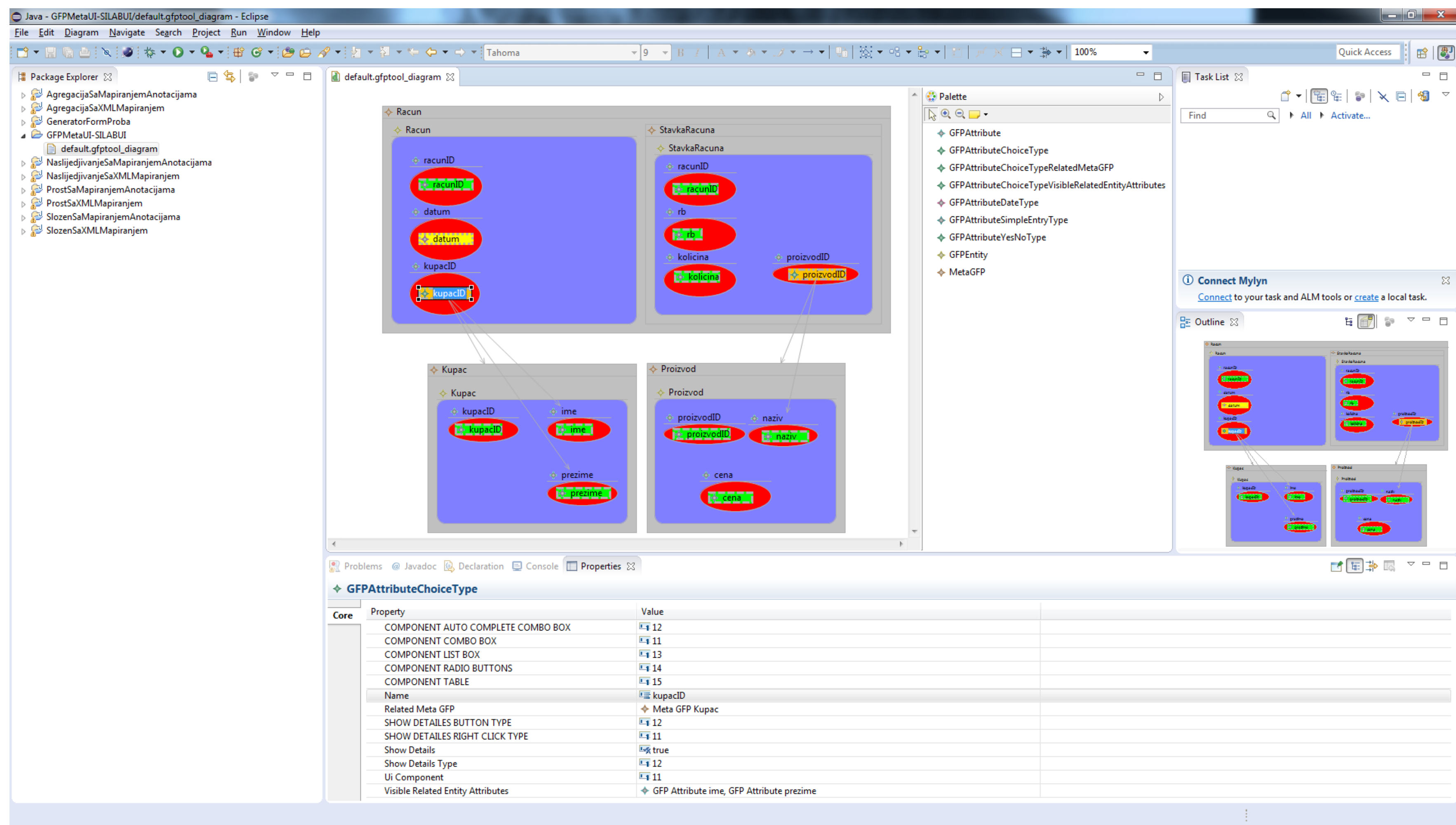


Слика 161: *SilabUI* Wizard – прилагођени мета-модел за коришћење у графичком алату за креирање улазне спецификације

7.2.3. Формирање улазне спецификације визуелним моделовањем

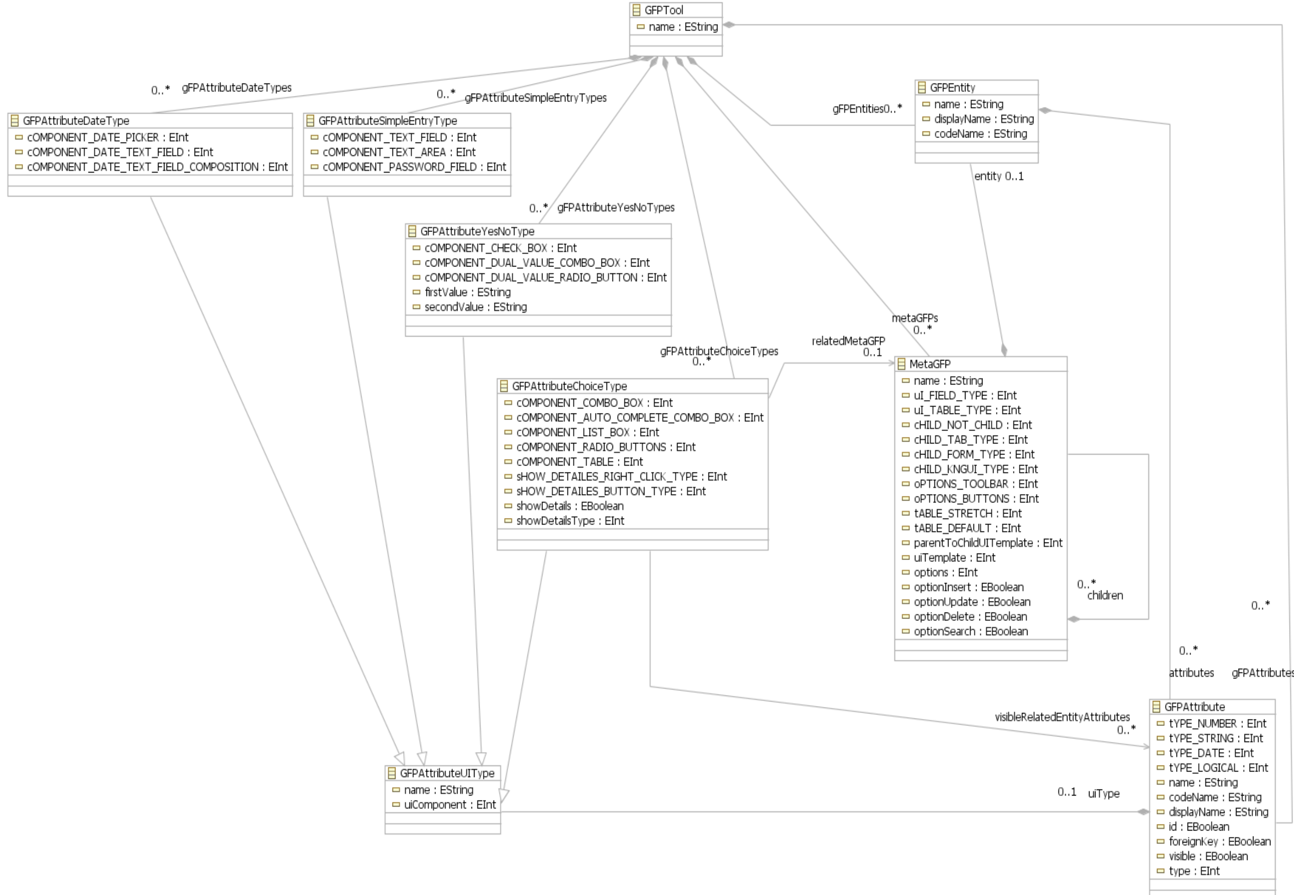
Како би одговорили на јасно артикулисан захтјев испитаника за постојањем алата за визуелно моделовање, који би се користио за формирање улазне спецификације, искоришћен је *Eclipse Modeling Framework* (*EMF*). Овај оквир је развијен од стране *Eclipse Tools* пројекта, а користи се за израду алата и чини значајан елемент *Model Driven Architecture (MDA)*. *Eclipse Modeling Framework* укључује *Essential Meta-Object Facility* (*EMOF*) стандард групе *OMG* који је имплементиран у склопу *Eclipse* платформе као *Ecore* модел, заправо мета-модел који се користи за опис других модела. На основу *Eclipse Modeling Framework* оквира настао је и *Graphical Modeling Framework* (*GMF*) који се користи за развој едитора за визуелно моделовање.

Алат за визуелно моделовање омогућава прављење модела заснованих на *SilabUI* мета-моделу, који је прилагођен за потребе креирања алата (Слика 163). Сви кључни елементи мета-модела имају своју графичку репрезентацију. Алат је креиран тако да омогући одабир елемената из палете, и постављање одабраних елемената на радну површину. У зависности од случајева коришћења који се желе представити моделом, ентитета који су везани за случајеве коришћења, типова њихових атрибута и одабира графичких компоненти, могуће је формирати визуелни графички модел конкретне улазне спецификације (Слика 162). Сваком елементу је могуће мијењати особине специфицирајући различите вриједности атрибута у *Properties* табу (картици).



Слика 162: Алат за визуелно моделовање улазне спецификације

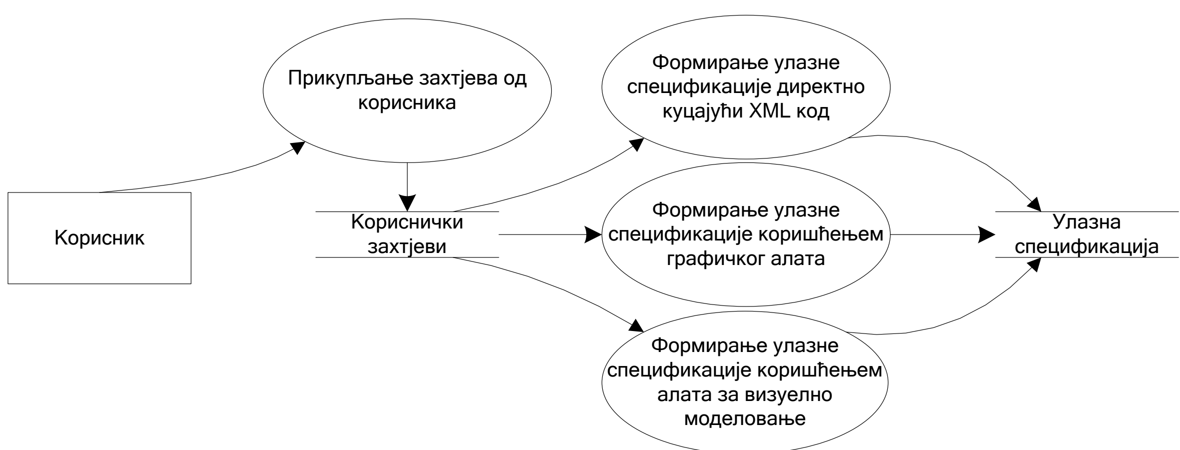
Након цртања модела и спецификације вриједности атрибута, добијени дијаграм се може отворити у текстуалном облику у форми *XMI* (*XML Metadata Interchange*) документа који представља стандардан формат за размјену модела заснован на *XML*-у. Спецификација у оваквом облику се може трансформисати у *XML* документа за основну и проширену спецификацију, која се даље може обрађивати графичким алатом, или се може директно учитати од стране алата за генерисање програмског кода корисничког интерфејса.



Слика 163: Мета-модел прилагођен за коришћење у алату за визуелно моделовање

7.3. Аутоматизација процеса трансформације предложеног модела у извршиви програмски код корисничког интерфејса

Циљ овог рада је да се развије модел за спецификацију корисничког захтјева који ће бити довољно семантички богат како би на основу њега било могуће не само пројектовање и имплементација корисничког интерфејса, већ и аутоматизација овог процеса. Циљ је да модел на основу којег ће се вршити генерисање програмског кода корисничког интерфејса садржи у себи све потребне информације како би било могуће задовољење свих постављених критеријума. Како би се олакшало креирање овог модела развијени су софтверски алати који имају улогу да воде корисника кроз процес спецификације корисничких захтјева и да конструише семантички богат модел који је потом спреман за трансформацију у извршиви програмски код корисничког интерфејса за различите типове софтверских система. Структура развијеног мета-модела, као и алати за формирање улазне спецификације описани су у поглављу 7.2. На сљедећем дијаграму виде се основни процеси и артефакти који су претходно описани (Слика 164).

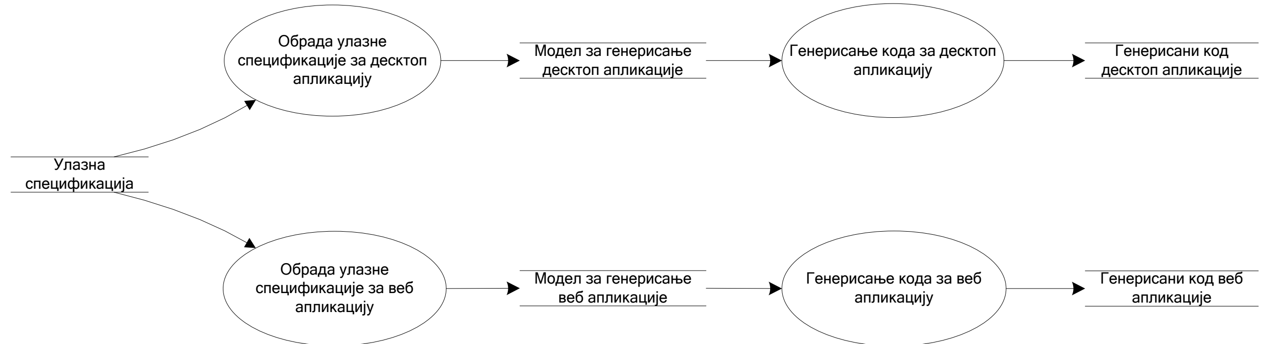


Слика 164: Формирање улазне спецификације

Да би се доказала одрживост овог приступа било је потребно развити софтверски алат који ће вршити трансформацију модела дефинисаног улазном спецификацијом у извршиви програмски код. На тај начин биће омогућено креирање корисничког интерфејса у раној фази животног циклуса софтвера, што пружа могућност валидације и верификације корисничких захтјева, али и значајно смањење потребног времена и напора за имплементацију корисничког интерфејса. Развијена су два алата – генератора програмског кода корисничког интерфејса и то:

- алат који омогућава генерисање извршивог програмског кода корисничког интерфејса **Јава десктоп апликација** и
- алат који омогућава генерисање извршивог програмског кода корисничког интерфејса **Јава веб апликација**.

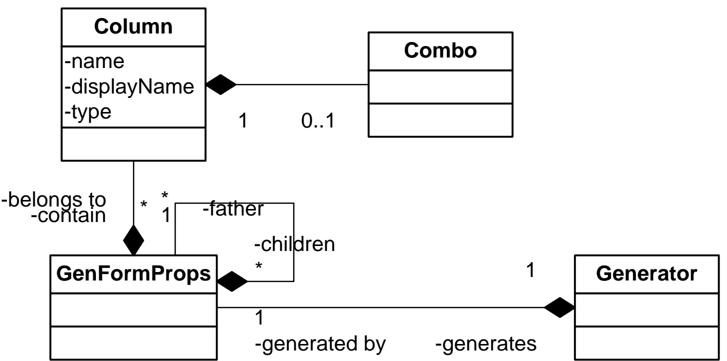
У општем случају генератори кода за дати улаз, примјеном одређених трансформационих правила генеришу одговарајући излаз. Зато оба развијена генератора прво врше обраду улазне спецификације и трансформишу је у структуру која је погодна за примјену правила трансформације односно генерисање кода (Слика 165).



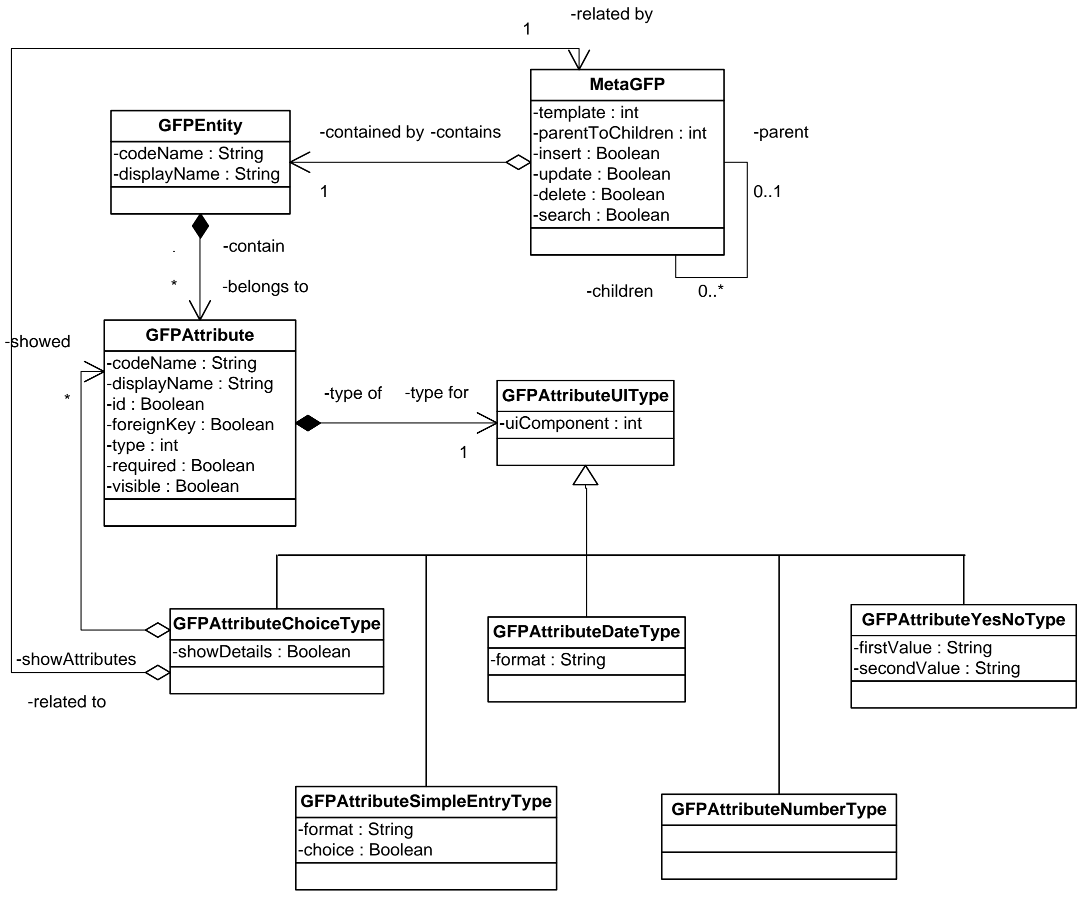
Слика 165: Обрада улазне спецификације и генерисање програмског кода

7.3.1. Структура генератора

У наставку ће бити приказане структуре модела које се користе као улаз код оба алата, као и структура самих генератора програмског кода. Треба истаћи да се ове структуре разликују једна од друге из чисто практичних разлога. Наиме, ако се погледа структура модела за генерисање корисничког интерфејса десктоп апликације (Слика 166) на први поглед се може примијетити да знатно одудара од предложеног мета-модела, док је код модела за генерисање корисничког интерфејса веб апликације (Слика 167) сличност много већа. Ове разлике заправо представљају посљедицу тога што се сам мета-модел током истраживања мијењао, о чему је и раније говорено, а приликом пројектовања алата, који нису пројектовани у исто вријеме, основ су биле актуелне верзије мета-модела. Намеће се питање да ли алат за генерисање корисничког интерфејса десктоп апликација, који је пројектован на основу мета-модела који је у међувремену знатно промијењен, може да омогући све оне карактеристике које омогућава алат за генерисање корисничког интерфејса веб апликација, ако се зна да се за генерисање веб апликација користи измијењени мета-модел? Одговор на ово питање је потврдан, јер се сама семантика модела временом незнатно мијењала, док структура мета-модела константно усавршавана са циљем да се олакша процес формирања улазне спецификацје, што не утиче на семантику потребну за генерисање програмског кода. Свакако, и у једном и у другом случају је неопходно улазну спецификацију прво трансформисати у модел који користе генератори кода, па свака промјена структуре улазне спецификације изискује промјену овог дијела генератора кода.

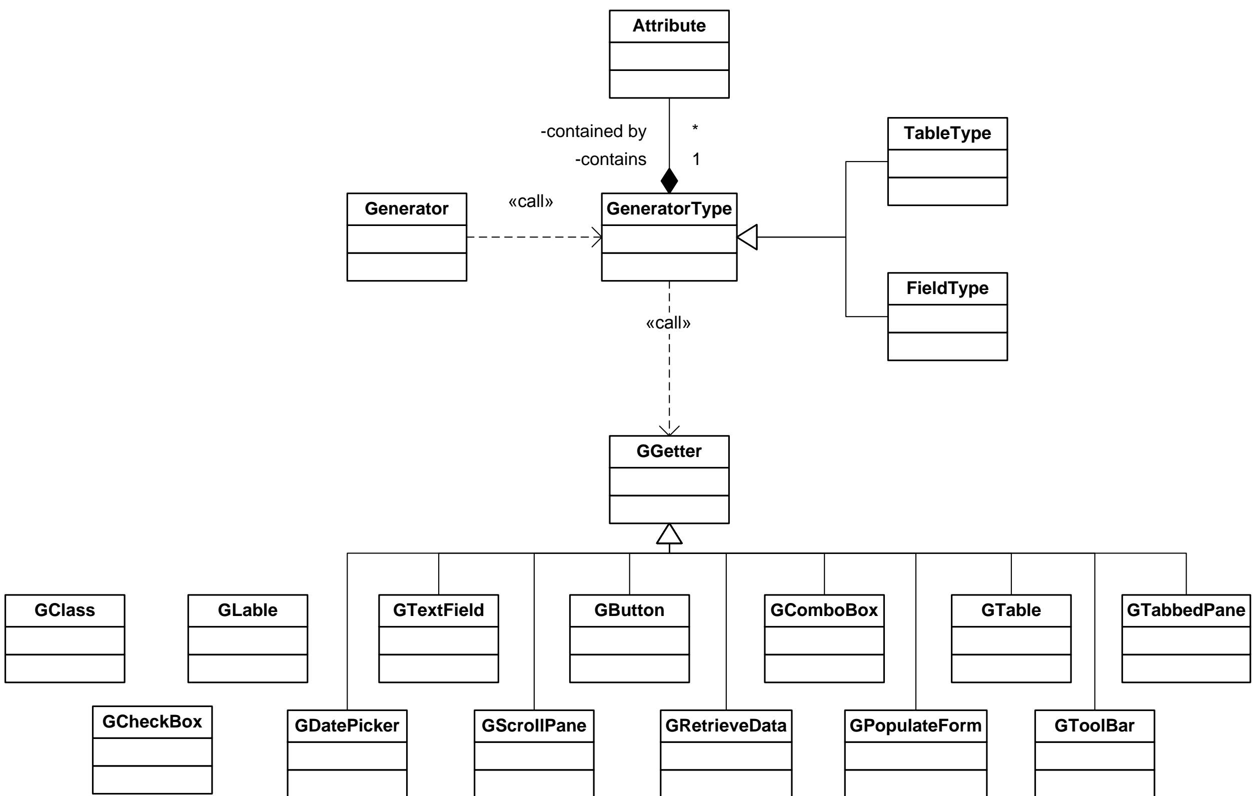


Слика 166: Структура модела за генерисање корисничког интерфејса десктоп апликације

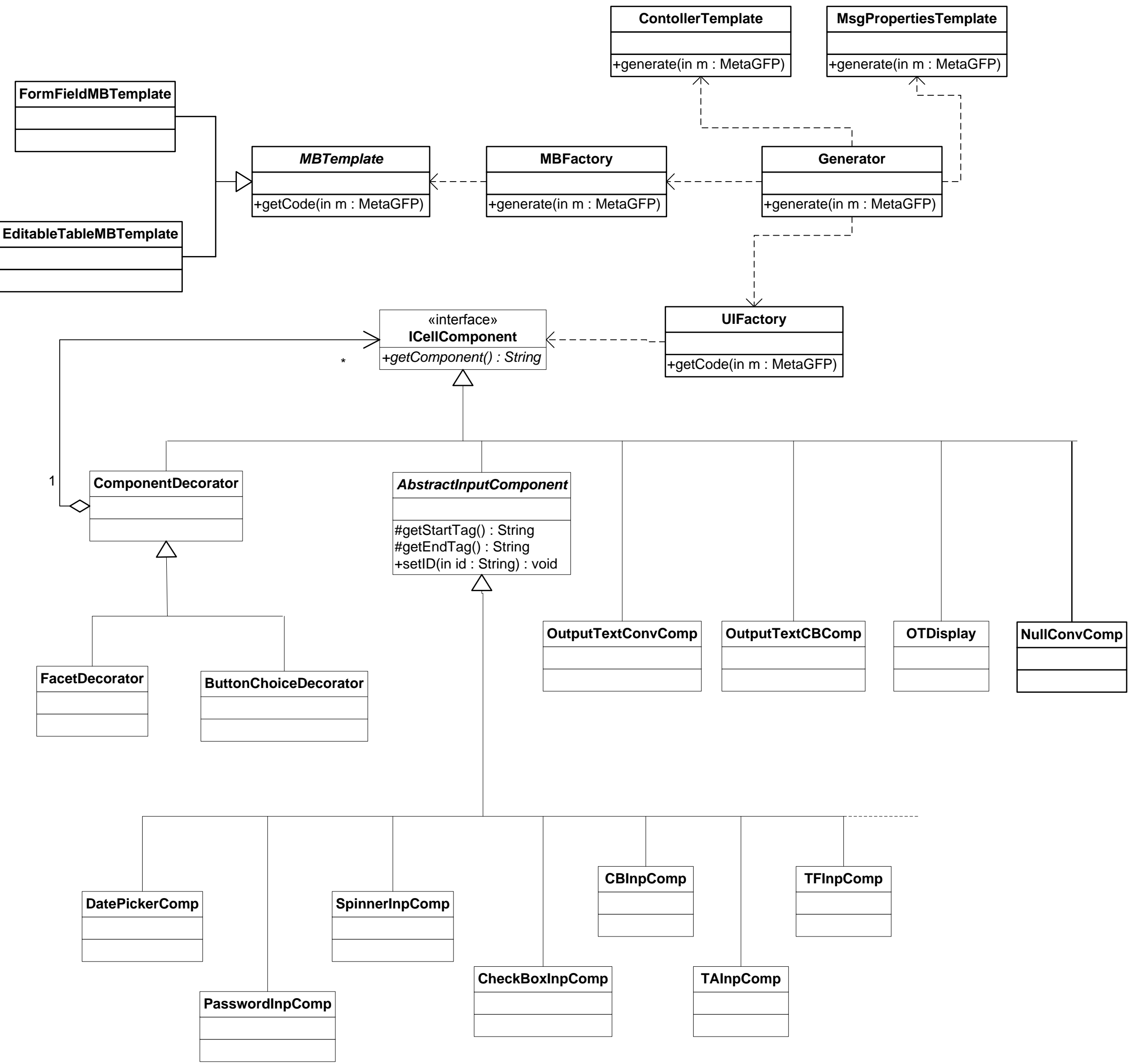


Слика 167: Структура модела за генерисање корисничког интерфејса веб апликације

У наставку ће бити приказане структуре генератора програмског кода за десктоп (Слика 168) и веб (Слика 169) апликације.



Слика 168: Структура генератора програмског кода корисничког интерфејса десктоп апликација



Слика 169: Структура генератора програмског кода корисничког интерфејса веб апликација

Сваки генератор је условљен финалном архитектуром генерисаног корисничког интерфејса која се знатно разликује код Јава десктоп апликација у односу на Јава веб апликације. У конкретном случају, имплементационе технологије – *Java Swing* која се користи за десктоп апликације и оквир за развој веб апликација *JavaServer Faces-Primefaces* изискивале су другачију структуру генератора, а у наставку

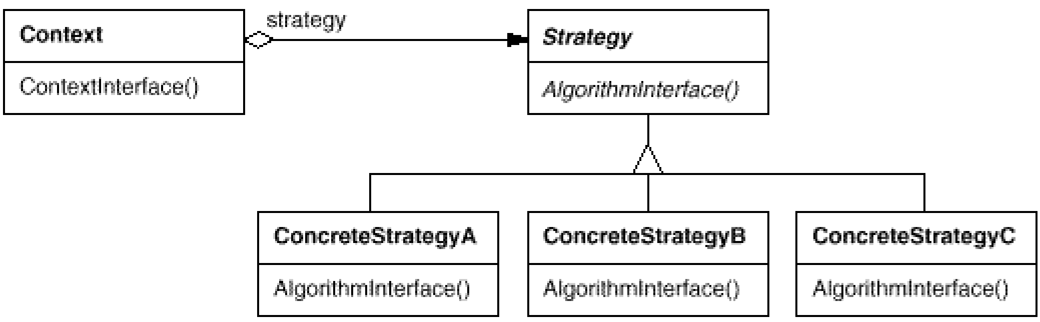
ће бити приказани софтверски патерни који су коришћени приликом пројектовања генератора.

7.3.2. Софтверски патерни коришћени у пројектовању генератора

У циљу лакшег одржавања генератора програмског кода корисничког интерфејса, приликом пројектовања су коришћени софтверски патерни. Одржавање генератора је неопходно из разлога што се циљане технологије често мијењају, али и како би се одговорило на специфичне захтјеве наручиоца, прије свега у смислу додавања нових графичких компонената. У наставку ће бити приказани коришћени софтверски патерни, као и разлози за њихову примјену.

Strategy **патерн**

Структура генерисаног кода јава десктоп апликације се значајно разликује у односу на одабране шаблоне корисничког интерфејса. Зато генератор мора обезбиједити различито понашање у зависности од вриједности дефинисаних у спецификацији. Како би програмски код генератора био што прегледнији, и како би се избјегла потреба за имплементацијом сложене логике извршења за оба случаја, која би се јављала на многим мјестима у програмском коду, као добро рјешење се наметнуо ***Strategy*** патерн [GOF]. Овај патерн омогућава формирање измјењиве фамилије алгоритама од којих ће бити извршен онај алгоритам који је одабран приликом извршења програма (Слика 170).

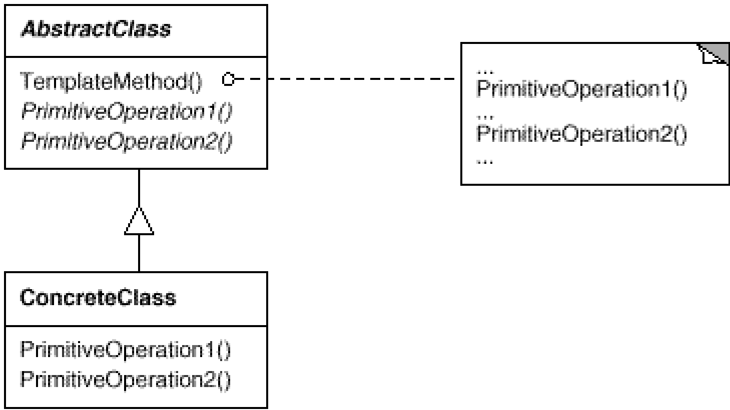


Слика 170: *Strategy* патерн

У структури генератора корисничког интерфејса десктоп апликације (Слика 168) може се видјети имплементација *Strategy* патерна гдје класа *Generator* игра улогу *Context* класе *Strategy* патерна, класа *GeneratorType* има улогу *Strategy*класе, док класе *FieldType* и *TableType* имају улогу *ConcreteStrategy* класа.

Template Method патерн

Коришћењем JSF технологије графичке компоненте се дефинишу коришћењем тагова, при чему се за зваку компоненту мора дефинисати почетни таг, крајњи таг, а између њих по потреби се дефинишу тагови задужени за конверзију и валидацију. Сваки од ових корака се, у зависности од врсте компоненте, различито извршава, па је за рјешење овог проблема примијењен *Template Method* патерн [GOF] који омогућава прављење шаблонске методе – методе која дефинише редослијед корака, при чему је могуће да се сваки корак имплементира на другачији начин (Слика 171).



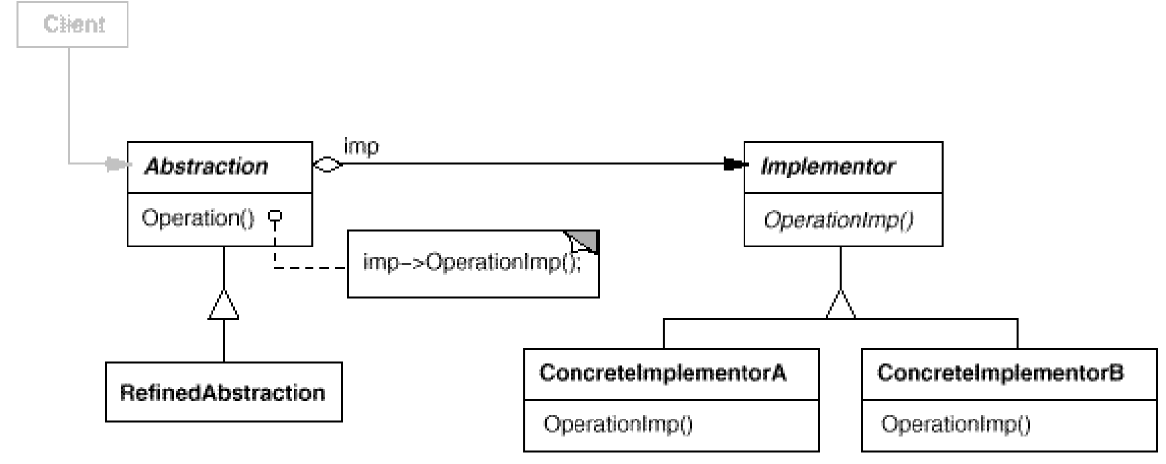
Слика 171: *Template Method* патерн

У структури генератора корисничког интерфејса веб апликације (Слика 168) може се видјети имплементација *Template Method* патерна гдје класа *AbstractInputComponent* игра улогу *AbstractClass* класе *Template Method* патерна, док класе *DatePickerComp, PasswordImpComp* итд. имају улогу *ConcreteClass* класа.

Bridge патерн

Постојање два основна шаблона корисничког интерфејса, као што је раније речено, захтијева различиту имплементацију корисничког интерфејса, односно различито извршење процеса генерисања кода. При томе, и за један и за други шаблон морају се генерисати исте графичке компоненте, са разликом у томе што се у једном случају оне директно постављају на екранску форму, а у другом се постављају у ћелијама табеле. Такође, поменуто је да се свакој компоненти може придружити програмски код задужен за конверзију и валидацију вриједности. Међутим свака компонента, у зависности од типа податка који се обрађује, може бити повезана са различитим врстама конверзије. Тако текстуално поље може бити генерисано за унос цјелобројних вриједности, али исто тако и за унос датума, када се компоненти

придружују различите врсте конвертера. Уз то, треба додати да се обрада вриједности садржаних у графичким компонентама разликује у односу на то да ли се компонента налази директно на екранској форми, или у ћелији табеле, па се на различит начин мора генерисати код. За рјешење овог проблема примијењен је *Bridge* патерн [GOF] који омогућава раздвајање апстракција од конкретних имплементација, како би се могле независно мијењати (Слика 172).



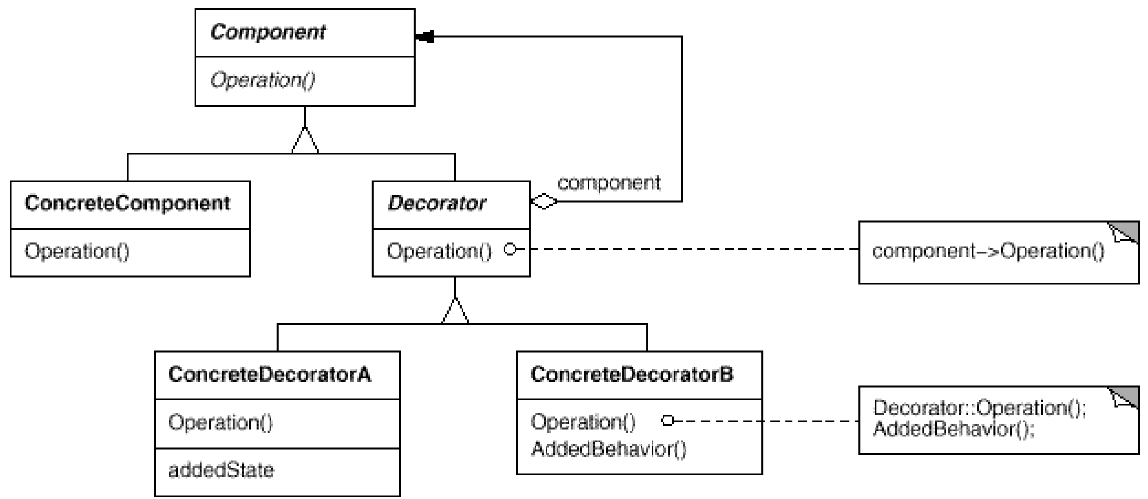
Слика 172: *Bridge* патерн

У структури генератора корисничког интерфејса веб апликације (Слика 168) може се видјети имплементација *Bridge* патерна гдје класа *AbstractInputComponent* игра улогу *Abstraction* класе *Bridge* патерна, док класе *IConverterComponent* и *ICompValue* итд. имају улогу *Implementor* класа. За обје *Implementor* класе постоје конкретне реализације.

Decorator патерн

У зависности од тога да ли је у улазној спецификацији одабрано генерисање опције *Show details* и у зависности од тога који је шаблон приказа ове опције одабран,

поред компоненте за избор вриједности је потребно генерисати и дугме за позив ове функционалности. Поред тога, приликом рада са табелама и директним уносом података у табелу, потребно је за сваку ћелију табеле обезбиједити тагове за унос и приказ података За рјешење овог проблема примијењен је *Decorator* патерн [GOF] који омогућава додавање додатних одговорности објекту динамички. (Слика 173).



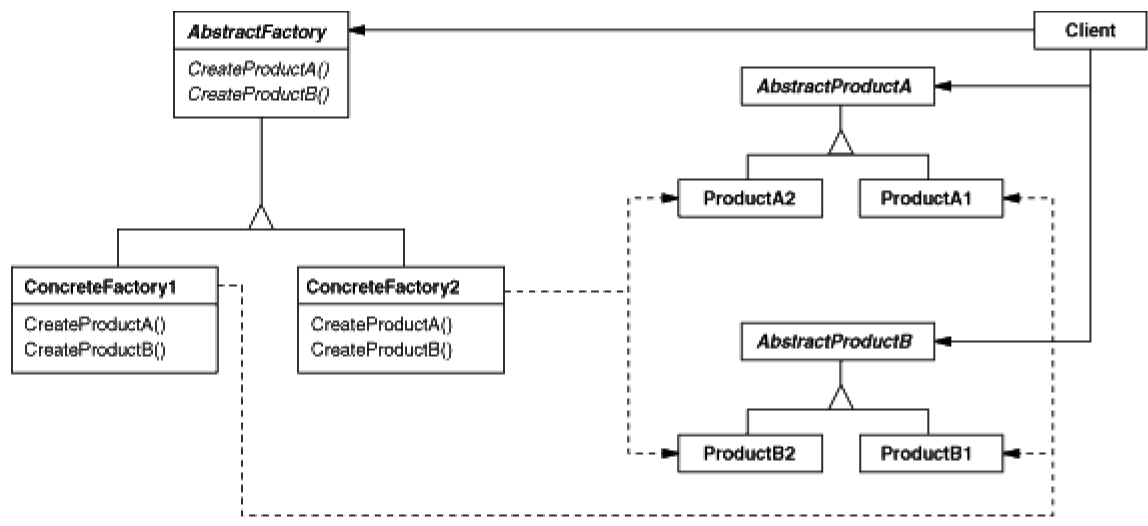
Слика 173: *Decorator* патерн

У структури генератора корисничког интерфејса веб апликације (Слика 168) може се видјети имплементација *Decorator* патерна гдје класа *ICellComponent* игра улогу *Component* класе *Decorator* патерна, класа *ComponentDecorator* игра улогу *Decorator* класе, док класе *FacetDecorator* и *ButtonChoiceDecorator* имају улогу *ConcreteDecorator* класа.

***Factory* патерн**

Самим увидом у мета-модел може се уочити велики број различитих графичких компоненанта које се могу користити у процесу генерисања кода.

Поједностављивање креирања компоненената је поједностављено коришћењем *Factory* патерна [GOF] (Слика 173).



Слика 174: *Factory* патерн

У структури генератора корисничког интерфејса веб апликације (Слика 168) може се видјети имплементација *Abstract Factory* патерна гдје класа *ComponentFactory* игра улогу *AbstractFactory (*у исто вријеме и *ConcreteFactory* јер је одмах дата конкретна имплементација класе*)* класе *Abstract Factory* патерна, док су различите класе које представљају графичке компоненте заправо *Product* класе *Abstract Factory* патерна.

7.3.3. Динамика процеса генерисања кода корисничког интерфејса

У случају оба алата за генерисање програмског кода корисничког интерфејса, без обзира да ли се ради о *html* или Јава програмском коду, уочено је да је већина програмског кода који је потребно изгенерисати непромјењива. Ако се као примјер узме графичка компоненту дугме у јава десктоп апликацији, потребно је извршити

иницијализацију односно креирање инстанце објекта, затим додијелити назив који ће бити видљив кориснику (а који се разликује за свако дугме), придружити дугмету ослушкивач догађаја (који такође треба да се разликује у односу на осталу дугмад) и конкретну имплементацију акције за одговарајући догађај (Слика 175). Програмски код за два различита дугмета ће по структури бити исти, али ће се разликовати по називима промјењивих и називима метода које се позивају. Како би се поједноставило генерисање програмског кода, направљени су текстуални шаблони који садрже стандардну структуру кода, али су унутар тог текста на посебан начин означена мјеста која се разликују од једног до другог случаја.

```
public $type$ get$name$(){\n

\tif ($name$ == null) {\n

\t\t$name$ = new $type$();\n

\t\t$name$.setText(\"$text$\");\n"

\t\t$name$.setIcon(new ImageIcon(getClass().getResource(\"$iconPath$\")));\n

\t\t$name$.addActionListener(new java.awt.event.ActionListener() {\n

\t\t\tpublic void actionPerformed(java.awt.event.ActionEvent e) {\n

\t\t\t\t$name$ActionPerformed(e);\n

\t\t\t}\n

\t\t});\n
```

Слика 175: Примјер текстуалног шаблона за компоненту дугме

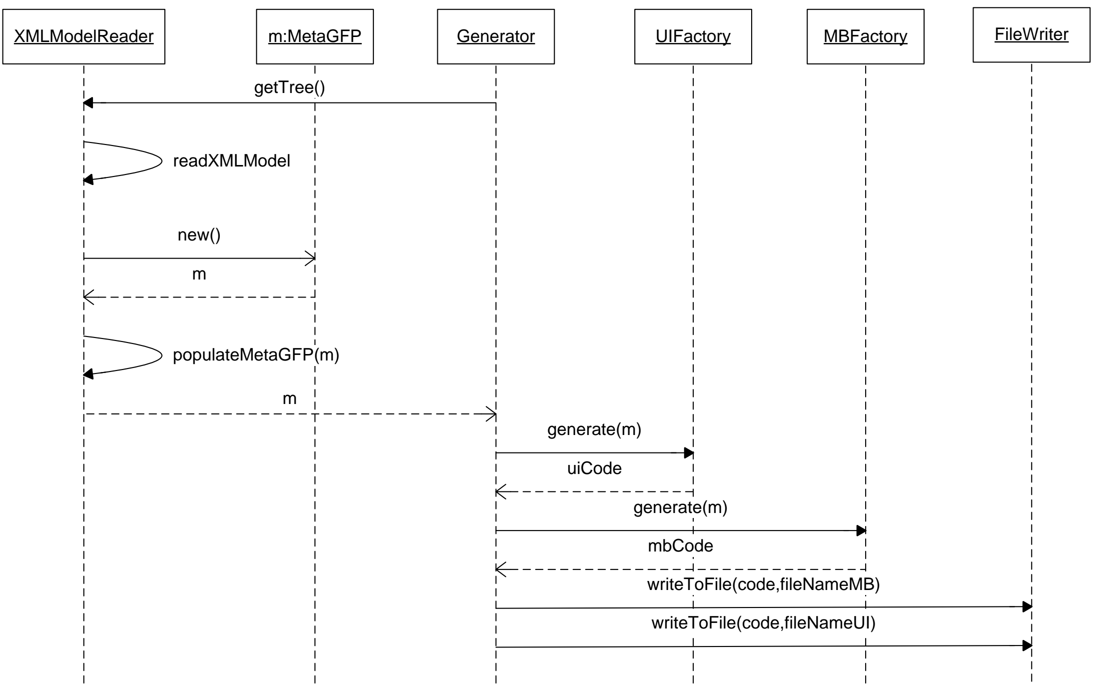
Генератори програмског кода тада имају улогу да припреме одговарајуће вриједности и поставе их у одговарајућа мјеста унутар текстуалног шаблона (Слика 176), након чега се овако конкретизован шаблон чува у одговарајућој датотеци (са

екстензијом *.java* ако се ради о Јава програмском коду, *.xhtml* у случају веб страница или *.properties* у случају датотека које се користе за интернационализацију).

```
public JButton getjbtInsert(){
    if (jbtInsert == null) {
        jbtInsert = new JButton();
        jbtInsert.setText("");
        jbtInsert.setIcon(new ImageIcon(getClass().getResource("/icon/Add16.gif")));
         jbtInsert.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                jbtInsertActionPerformed(e);
            }
        });
}
```

Слика 176: Примјер конкретизованог текстуалног шаблона за компоненту дугме

учитати улазну спецификацију из *useCase.xml* односно *useCaseGuiExtension.xml* датотека и трансформисати податке у облик погодан за генерисање кода. За оба генератора имплементиране су тзв. *ModelLoader* компоненте које парсирају улазне датотеке и припремају улазни модел за генерисање кода. Након тога се позива метода *generate()* која прво генерише програмски код корисничког интерфејса са случај коришћења на највишем нивоу, а затим рекурзивно и за остале случајеве коришћења, све до случајева коришћења који немају своју дјецу, када се процес генерисања завршава. Динамика процеса генерисања програмског кода биће приказана УМЛ дијаграмом секвенци на примјеру алата за генерисање програмског кода веб апликација (Слика 177).



Слика 177: *UML* дијаграм секвенци који описује генерисање програмског кода веб апликације

7.3.4. Генерисани програмски код

У наставку ће бити описане карактеристике генерисаног програмског кода оба генератора, који представља резултат, односно излаз, из процеса генерисања програмског кода корисничког интерфејса.

Генерисани програмски код генератора корисничког интерфејса десктоп апликација

Након примјене трансформације улазног модела генератора генерише се програмски код, чиме се добија резултујућа апликација. Добијена апликација је у

потпуности специфична домену проблема. У том смислу, централна графичка компонента која се користи као контејнер компонента за остале графичке компоненте је класа *JPanel*. На тај начин резултујућа апликација није чврсто повезана са начином приказа графичких компоненти. Наиме, најчешће се за приказ форми користе компоненте које представљају појављивања класе *JFrame* и *JDialog*. Међутим, компонента класе *JPanel* се може поставити у *JFrame* и *JDialog* компоненте, тако да се може рећи да компонента *JPanel* представља генеричку, лагану (*Lightweight*) контејнер компоненту, лако преносиву из једног у други контејнер.

Као што је већ напоменуто, дефинисано је неколико шаблона, односно неколико начина за приказ података на форми. Приликом генерисања програмског кода користе се предефинисани шаблони у оквиру којих се постављају графичке компоненте. На основу тога, може се закључити да се исти подаци могу приказати на различите начине (у облику форме са текстуалним пољима, табеларно, у засебним прозорима...), при чему се графичке компоненте налазе на *JPanel* компоненти. Добијене панеле могуће је отворити из неког графичког алата који подржава уређивање форми, о чему је више ријечи било у уводу овог поглавља (Слика 143).

Поред тога, за поједине типове форми дефинисани су и панели за унос нових података. Уколико се изабере операција за додавање новог елемента, отвара се екранска форма за унос. Ове екранске форме такође су реализоване преко *JPanel* компоненти, а њихов приказ врши се у оквиру модалне *JDialog* компоненте. Након уноса и чувања података, затварањем екранске форме за унос, унесени подаци приказују се на форми која приказује *отац* ентитет.

Такође, за компоненте које су реализоване као листе за избор (на примјер компонента *JComboBox*) постоји могућност генерисања панела за детаљнији приказ података (*Show details* функционалност). Наиме, обично се у листи за избор приказује

само један атрибут посматраног слога (најчешће назив, опис или нешто слично). Такође, у листи за избор приказују се вриједности које није могуће мијењати, већ се из листе бира одређена вриједност. Уколико је укључена могућност за генерисање ове функционалности, десним кликом на листу за избор, или кликом на дугме, приказује се падајући мени који омогућава отварање екранске форме за детаљнији приказ података. Овај панел се такође приказује у оквиру модалне *JDialog* компоненте. На тај начин се омогућава детаљнији преглед референцираних података. Такође, уколико је листа празна, могуће је додати нове вриједности коришћењем панела за детаљнији приказ. Након избора/чувања података, затварањем екранске форме за детаљнији приказ, изабрани/унесени подаци приказују се у оквиру *JComboBox* компоненте полазне форме.

Генерисани кориснички интерфејс омогућава извршавање основних операција над базом података (додавање новог слога, промјену слогова, брисање слогова, селекцију слогова). Међутим, у највећем броју случајева мора се дефинисати пословна логика која ће извршавати неке операције (нпр. израчунавање неких међурезултата, сложена правила валидације и сл.). Такође, временом се кориснички интерфејс и пословна логика могу промијенити, што ће довести до потребе да се промијени генерисани програмски код. У том случају се препоручује да се промјене не уносе директно у генерисану класу (тј. генерисани панел), већ да се генерисана класа наслиједи, при чему је потребно да се све специфичности смијештају у оквиру насљеђене класе. На тај начин могуће је извршити поновно генерисање екранских форми, при чему ће специфична логика остати сачувана.

За сваки дефинисани модел случајева коришћења креира се засебан пакет (*Java package*), у који се смијештају све генерисане *.java* датотеке.

Генерисане екранске форме немају информације о доменским објектима, чиме је обезбијеђена независност презентационог дијела апликације од доменског модела. Комплетна структура која представља модел података који се приказује на екранској форми чува се у тзв. *Data Transfer Object* (*DTO*) објекту који је дефинисан на форми која приказује податке о ентитету родитељ.

Генерисани програмски код генератора корисничког интерфејса веб апликације

Приликом генерисања програмског кода корисничког интерфејса веб апликације као резултат се добијају сљедеће датотеке:

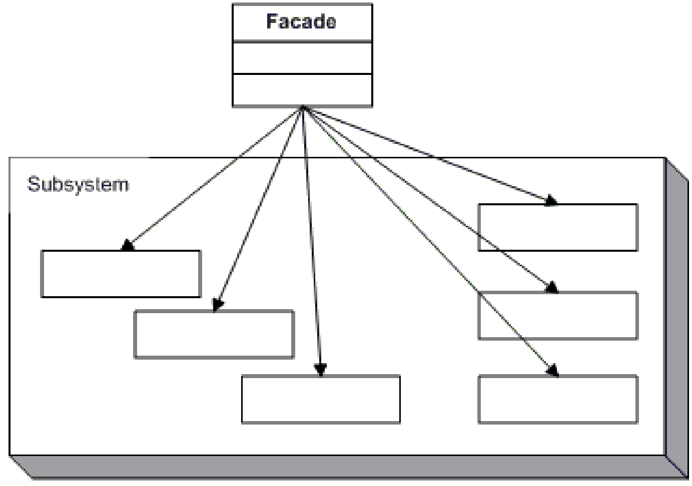
- *include.xhtml* (веб странице које садрже графичке компоненте),
- *ManagedBean* (Јава класе које играју улогу контролера корисничког интерфејса),
- *Controller* (Јава класа којом се успоставља веза са пословном логиком система),
- *Converter* (класе које омогућавају конверзију података у графичким компонентама),
- *message.properties* (датотеке које омогућавају интернационализацију података).

XHTML датотеке представљају презентациони ниво веб апликације који садржи графичке компоненте преко којиг се врши приказ и обрада података. Компоненте су директно везане за атрибуте специфициране у *Managed bean* класама. *Include.xhtml* се може посматрати као један фрагмент шире веб странице у коју се може укључити коришћењем *ui:include* тага који се поставља унутар *form* тага на веб страници. Изгенерисане компоненте корисничког интерфејса се смијештају у *panelGroup*

компоненту која се у зависности од тога да ли се ради о ентитету родитељ или ентитету дијете смијешта или у оквиру *form* тага, или у оквиру таб компоненте (картице) или унутар посебног дијалога. Унос нове инстанце ентитета као и обрада постојеће инстанце ентитета коришћењем *Show details* опције се врши у оквиру засебног дијалога.

Managed bean има улогу контролера корисничког интерфејса који садржи све податке који се обрађују или приказују на веб страници. Поред тога, садржи и методе које омогућавају комуникацију са пословном логиком система. Сама структура *Managed Bean* класе се разликује прије свега у зависности од специфицираних шаблона корисничког интерфејса.

Класа *Controller* садржи методе које позива *Managed Bean* у циљу извршења системских операција апликационе логике система. Ова класа представља имплементацију *Facade* патерна пројектовања [GOF] (Слика 178), јер представља интерфејс према подсистему пословне логике. На овај начин *Managed Bean* класе нису директно зависне од конкретних системских операције, већ само од класе *Controller*.



Слика 178: *Facade* патерн

Конверзија података на корисничком интерфејсу врши се коришћењем класа које имплементирају интерфејс `javax.faces.convert.Converter`.

Генерисан и кориснички интерфејс нема информације о доменским објектима чиме је постигнута независност од доменског модела. Модел података се, као и у случају генерисаног корисничког интерфејса десктоп апликација, чува у оквиру *Data Transfer Object* (*DTO*) објекта, који се декларише у оквиру *Managed bean*.

7.3.5. Комуникација генерисаног корисничког интерфејса са апликационом логиком система

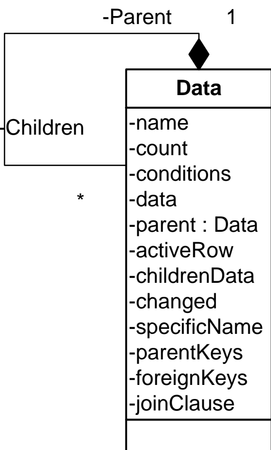
Један од циљева овог рада јесте да се омогући аутоматска трансформација спецификације, коју чине случајеви коришћења обогаћени информацијама везаним за кориснички интерфејс, у извршиви програмски код. Међутим, поред презентационог нивоа који се генерише коришћењем представљених алата, софтверски систем мора да садржи и компоненте које чине ниво апликационе логике која учаурује структуру и понашање софтверског система. Како се овај алат односи

искључиво за генерисање корисничког интерфејса, било је потребно обезбиједити механизам комуникације са апликационом логиком система, без везивања за конкретан начин реализације апликационе логике.

Зависност између ових дијелова система значајно је смањена дефинисањем интерфејса према подсистему који представља апликациону логику система (Слика 178). Овим интерфејсом дефинише се начин комуникације, која мора да постоји између корисничког интерфејса и апликационе логике система. Поред скупа метода које се дефинишу интерфејсом, а које се тичу захтијеваног понашања система, које могу бити реализоване на различите начине, без утицаја на клијента – тј. кориснички интерфејс апликације, потребно је направити и независност од доменске структуре система која се користи на нивоу апликационе логике, и која такође на различите начине може бити реализована.

Како би се извршило декупловање – тј. смањила зависност између структуре података која се користи на нивоу корисничког интерфејса и оне структуре која се користи на нивоу апликационе логике, дефинисана је класа *DataTransferObject*.

Data Transfer Object (DTO), за који се често користи израз *Value Object* тј. VO. представља објекат који служи као медијум за пренос података између различитих нивоа софтверског система. [Alur03] [Fowler03] [Marinilli06] Трансферни објекти обично имају јаван приступ до својих атрибута због чега није непходно правити *get()* и *set()* методе. Наравно, у случају да је потребан одређени степен контроле приступа атрибутима, за такве атрибуте неопходно је промијенити право приступа и направити одговарајуће методе за приступ. Зависно од захтјева апликације постоје варијације у имплементацији трансферних објеката. Трансферни објекти најчешће имплементирају интерфејс *Serializable* како би било омогућено њихово коришћење у дистрибуираном окружењу.



Слика 179: Класа *DataTransferObject*

DataTransferObject класа је пројектована тако да буде генерична и омогући велику флексибилност у односу на различите структуре података које треба да садржи. Приликом генерисања корисничког интерфејса водило се рачуна да се за спецификацију у *useCase.xml* датотеци креира структура трансферног објекта која је потпуно независна у односу на различите спецификације у *useCaseGuiExtensionl.xml* датотеци. Сваки шаблон посједује сопствени механизам који ће вршити пресликавања вриједности између трансферног објекта и графичких компоненти. То значи да ако се захтијева промјена шаблона корисничког интерфејса за одређени, већ генерисани случај коришћења, за који је имплементирана апликациона логика, потребно је само поновно генерисање корисничког интерфејса, на основу измијењене *useCaseGuiExtensionl.xml* датотеке, које не захтијева никакве измјене над имплементираном апликационом логиком.

7.3.6. Валидација података на презентационом нивоу

Коришћењем *useCaseGuiExtensionl.xml* датотеке омогућено је и генерисање кода одговорног за извршење валидације на страни корисничког интерфејса апликације. Без обзира на извршење валидације на страни корисничког интерфејса,

валидациона правила најчешће се провјеравају и на страни апликационе логике система, док се на перзистенционом нивоу такође врши провјера конзистентности и интегритета података. На тај начин, на неколико мијеста могуће је блокирати обраду података који не задовољавају дефинисана валидациона правила. Употреба валидације на страни корисничког интерфејса корисна је из разлога што се смањује радно оптерећење апликационе логике система, која се позива само у случају да су сви подаци валидни [Antovic10].

Валидациона правила односе се на податке које обрађује систем. Када се говори о релационом моделу података валидација се може односити на атрибуте релације, табеле, као и на више повезаних табела. Тако је могуће идентификовати пет врста валидационих правила тј. правила која се односе на валидацију:

- типа атрибута;
- вриједности атрибута;
- међузависности атрибута једне релације;
- међузависности атрибута различитих релација;
- структурних ограничења.

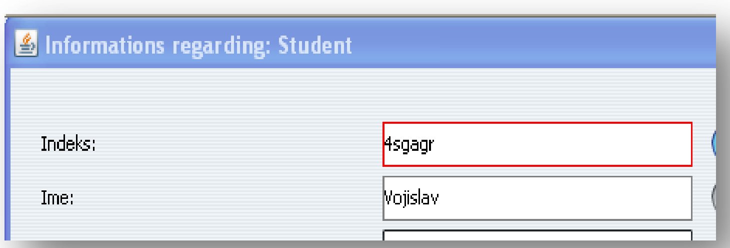
Приликом генерисања корисничког интерфејса могуће је генерисати код одговоран за валидацију, при чему се на основу предложеног модела могу аутоматски примијенити правила која се односе на тип атрибута релације. Сва остала валидациона правила могуће је накнадно додати проширивањем метода које се односе на валидацију.

Оно о чему треба водити рачуна јесте да валидациона правила на страни корисничког интерфејса не треба да буду превише наметљива, и ограничавајућа приликом уноса података, већ корисника треба упознати са постојањем нарушавања

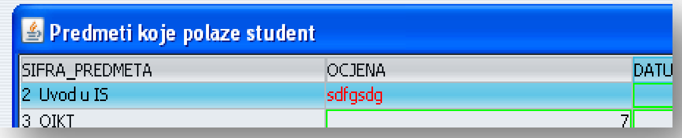
одређеног правила, како би се, прије позива системске операције, дала могућност кориснику да исправи уочене грешке. Ако грешке нису отклоњене, а дође до позива системске операције, биће приказана порука о грешци, али ће и тада корисник имати прилику да прије извршења операције одустане од позива и исправи грешке. Ако се ипак позове системска операција, тада ће систем приказати поруку о неуспјешном извршењу операције.

У случају веб апликација, поруке које се приказују корисницима, а које су везане за валидацију смјештене су у оквиру *validation-msg.properties* датотеке [Karic14].

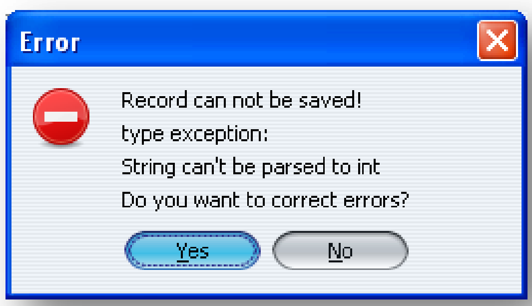
На следећим сликама приказани су ефекти генерисаних валидационих правила на понашање корисничког интерфејса апликације за десктоп (Слика 180, Слика 181, Слика 182) и веб апликације (Слика 183, Слика 184, Слика 185, Слика 186).



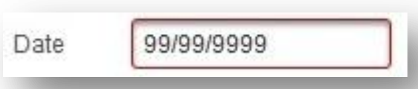
Слика 180: Валидација – приказивање грешке приликом уноса у текстуално поље у десктоп апликацији



Слика 181: Валидација – приказивање грешке приликом уноса у ћелију табеле у десктоп апликацији

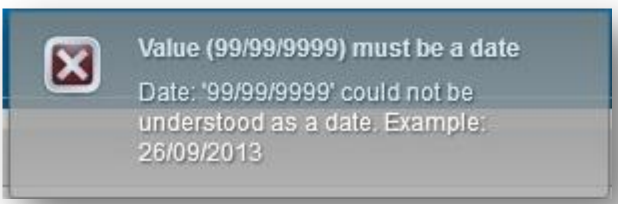


Слика 182: Валидација – приказивање грешке приликом позива системске операције у десктоп апликацији

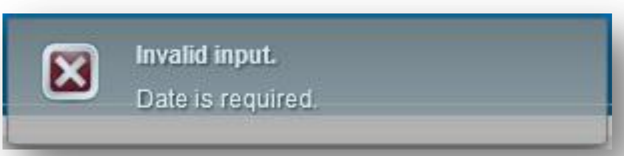


Слика 183: Валидација – приказивање грешке приликом уноса вриједности за датум у веб апликацији

Слика 184: Валидација – приказивање грешака приликом уноса у ћелију табеле у веб апликацији



Слика 185: Валидација – приказивање грешке приликом уноса неодговарајућег типа податка у веб апликацији

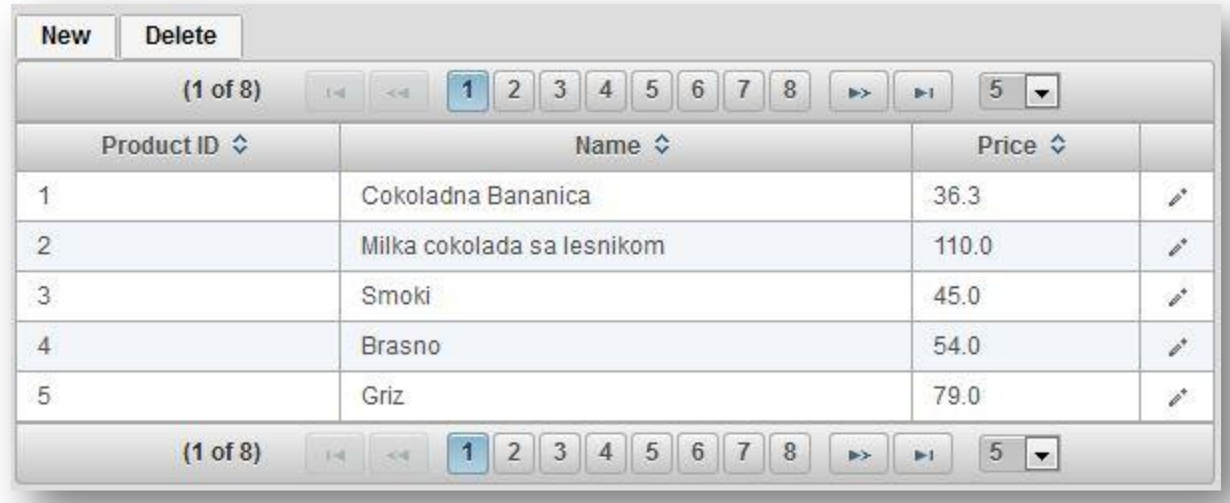


Слика 186: Валидација – приказивање грешке у случају непопуњавања обавезног поља у веб апликацији

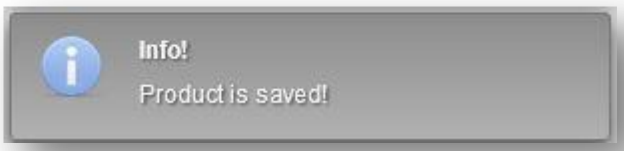
7.3.7. Локалишација и интернационализација

Генерисани програмски код корисничког интерфејса садржи подршку за интернационализацију и локализацију података који се приказују на корисничком интерфејсу. Међу овим подацима се налазе називи ентитета, називи атрибута ентитета, као и називи опција који се приказују на дугмадима и поруке које се приказују корисницима (Слика 187, Слика 188, Слика 189, Слика 190). Сви наведени подаци се приликом генерисања смијештају у *messages.properties* датотеку. На овај начин, без улажења у програмски код, ове податке је могуће мијењати. Уз

коришћење механизма локализације, који омогућава прилагођавање приказа података локалним подешавањима, могуће је дефинисати и посебне *.properties* датотеке које ће садржати наведене податке преведене на жељени језик. Како би се омогућио приказ података на српском језику, потребно је дефинисати *messages_sr.properties* датотеку, и у њој унијети податке на српском. Уколико је омогућено да се аутоматски препознају локална подешавања корисника приликом приступа систему, аутоматски ће се користити одговарајућа *.properties* датотека, уколико постоји, а ако не постоји, учитаће се подаци у облику који се специфициран у *messages.properties* датотеци [Karic14].



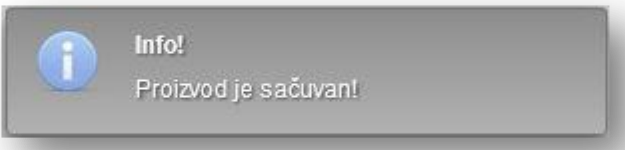
Слика 187: Приказ корисничког интерфејса прилагођен енглеском језику



Слика 188: Приказ поруке која се приказује кориснику на енглеском језику

ProizvodID	Naziv	Cena	
6	Sporet	7800.0	
7	Pegla	4110.0	
8	Kafa	120.0	
9	Kesa za vino	120.0	
12	Fen	6700.0	

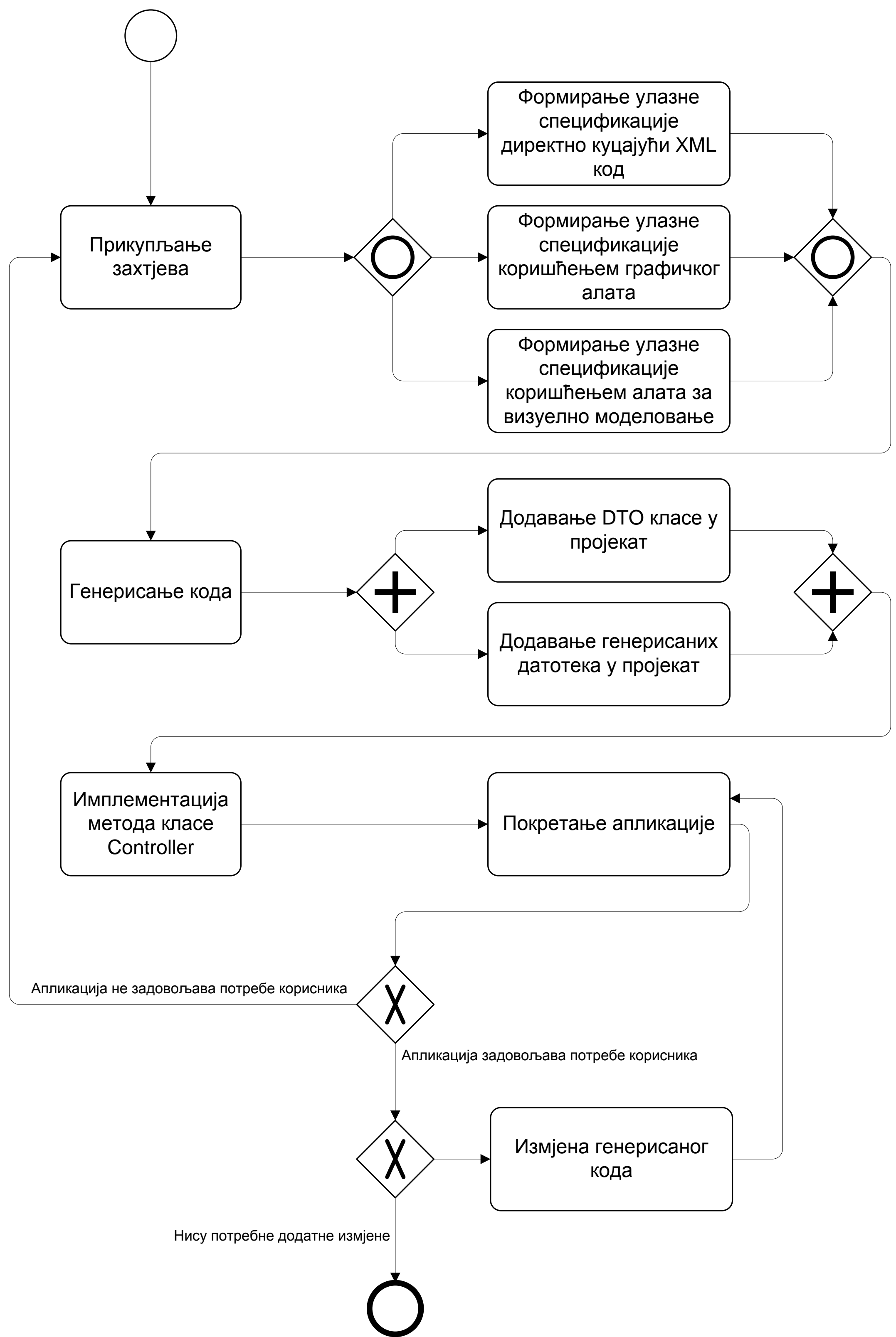
Слика 189: Приказ корисничког интерфејса прилагођен српском језику



Слика 190: Приказ поруке која се приказује кориснику на српском језику

7.3.8. Процес развоја корисничког интерфејса SilabUI приступом

У наставку ће графички бити приказан ток процеса развоја корисничког интерфејса софтверског система коришћењем предложеног *SilabUI* приступа (Слика 191).



Слика 191: Процес развоја корисничког интерфејса *SilabUI* приступом

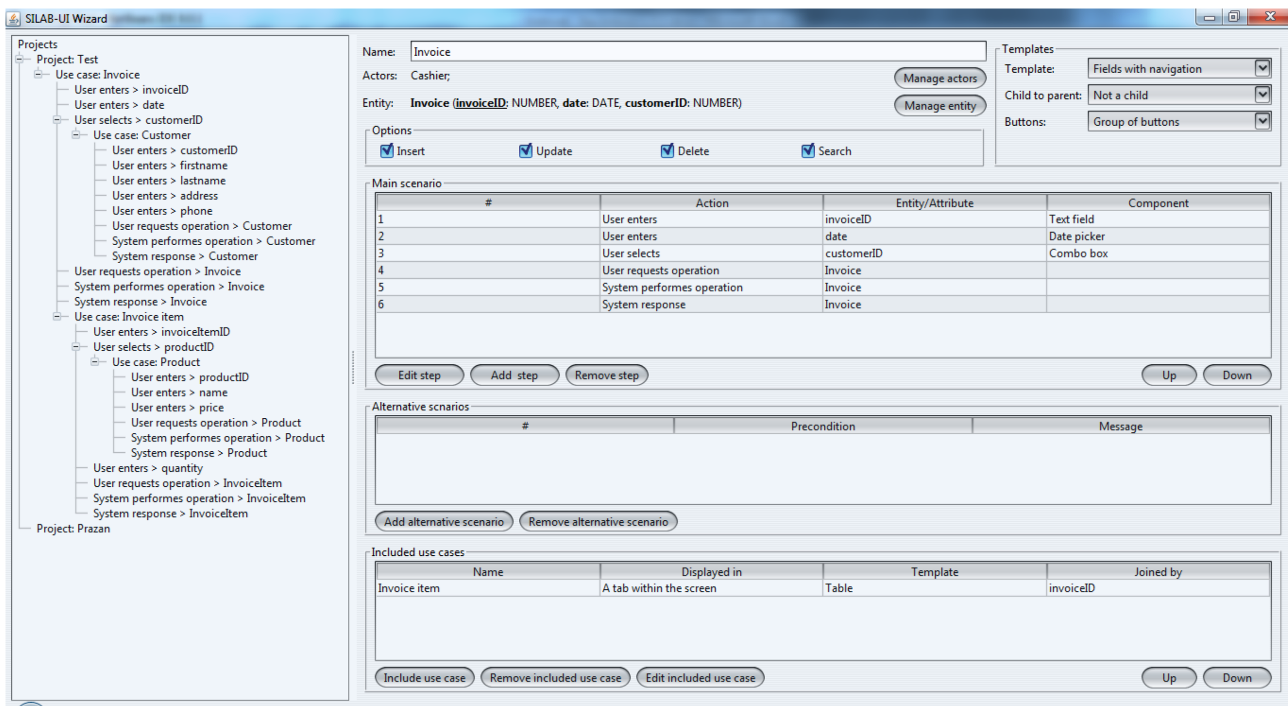
8. Евалуација предложеног SilabUI приступа

У овом поглављу биће извршена евалуација предложеног *SilabUI* приступа кроз два студијска примјера чиме ће бити приказан развој десктоп и веб апликације и то кроз активности приказане дијаграмом (Слика 191). У плану је да се спроведе експеримент који ће подразумијевати учешће искусних програмера и мјерење времена развоја и измјене софтверског система са и без коришћења генератора, као и анкета у којој ће програмери моћи да дају своју оцјену о предложеном приступу и вриједности оваквог алата у свакодневном коришћењу за потребе привреде, као и евентуалне предлоге за његово побољшање, као једног од начина евалуације резултата истраживања. Иако је овај експеримент дио истог истраживања, неће бити обухваћен докторском дисертацијом, прије свега због времена и других ресурса које је потребно обезбиједити за спровођење експеримента. Међутим, како је за потребе дисертације спроведено испитивање – анкета, чији су резултати посматрани као захтјеви које треба да испуни нови приступ генерисању програмског кода корисничког интерфејса, сматрам да само испуњење сваког од уочених захтјева, на начин описан у претходном поглављу, као и у дијелу који слиједи, представља довољно квалитетну евалуацију *SilabUI* приступа.

Прва и друга активност у процесу развоја (прикупљање захтјева и формирање улазне спецификације) су идентичне за оба примјера. Кориснички захтјев подразумијева прављење апликације за евидентирање рачуна (*Invoice*) који се издају купцима (*Customer*) као и производима (*Product*) које су купили, а који се

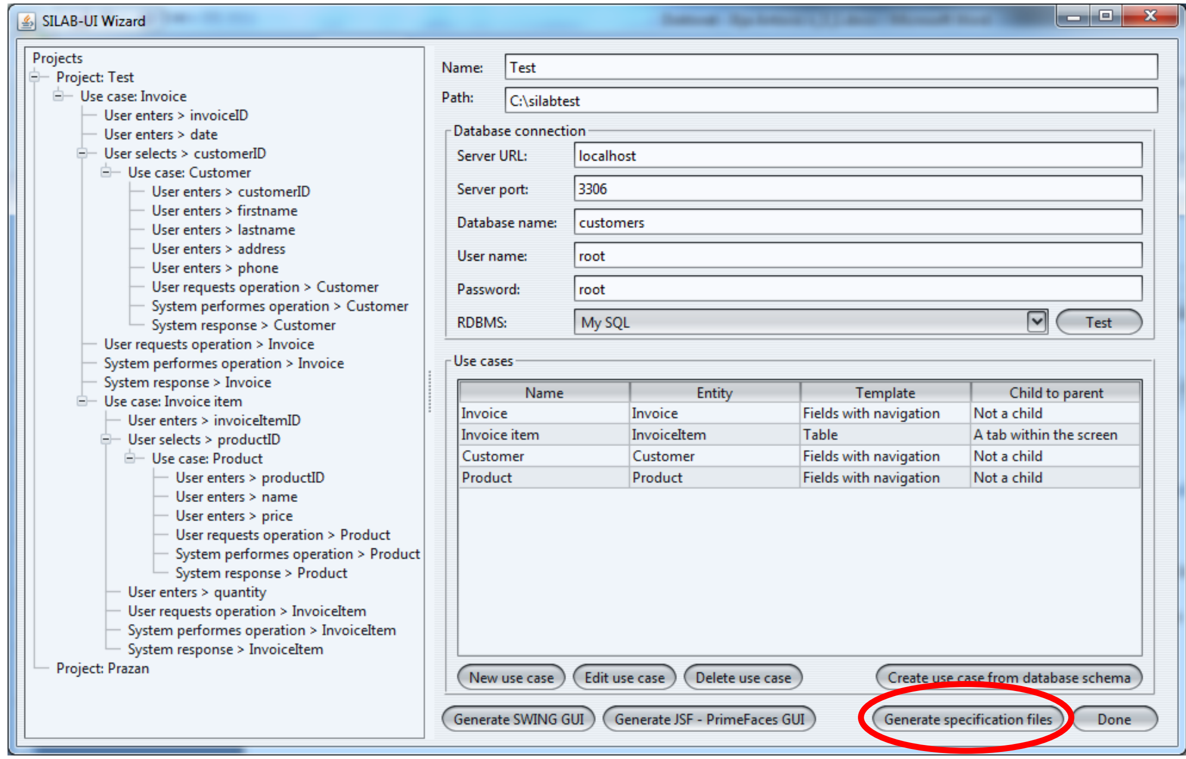
евидентирају кроз ставке рачуна (*InvoiceItem*). Потребно је обезбиједити унос, измјену, брисање и претрагу рачуна, производа и купаца.

Улазна спецификација је формирана коришћењем графичког алата. У наставку је приказан изглед графичке форме за спецификацију случаја коришћења *Invoice* (Слика 192).



Слика 192: Формирање улазне спецификације за случај коришћења *Invoice* коришћењем графичког алата

На основу спецификације се аутоматски коришћењем графичког алата може генерисати програмски код десктоп и веб апликација, али се исто тако може генерисати спецификација у текстуалном (*XML*) облику (Слика 193).



Слика 193: Генерисање улазне спецификације у текстуалном облику

У наставку ће бити приказана добијена улазна спецификација (основа и проширење) у текстуалном (*XML*) облику (Слика 194, Слика 195).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<specification>
 <useCase usecaseID="UCInvoice0" name="Invoice" precondition="Nema preduslova"
postcondition="Nema postuslova">
  <actor name="Cashier"/>
  <entity id="Invoice1" name="Invoice">
    <attribute type="NUMBER" identity="true" id="invoiceID2" name="invoiceID"/>
    <attribute type="DATE" id="date3" name="date"/>
    <attribute type="NUMBER" id="customerID4" name="customerID"/>
  </entity>
  <mainScenario>
    <step stepNumber="1" action="ENTRY" entity="invoiceID2"/>
    <step stepNumber="2" action="ENTRY" entity="date3"/>
    <step stepNumber="3" action="SELECTION" entity="customerID4">
        <extends useCase="UCCustomer11" joinedByAttribute="customerID13"/>
    </step>
    <step stepNumber="4" action="REQUEST_OPERATION" entity="Invoice1"/>
    <step stepNumber="5" action="SYSTEM_ACTION" entity="Invoice1"/>
    <step stepNumber="6" action="OPERATION_REPORT" entity="Invoice1"/>
  </mainScenario>
  <include child="UCInvoice item5" joinedByAttribute="invoiceID7"/>
 </useCase>
```

```xml
<useCase usecaseID="UCInvoice item5" name="Invoice item" precondition="Nema preduslova"
postcondition="Nema postuslova">
  <entity id="InvoiceItem6" name="InvoiceItem">
    <attribute type="NUMBER" id="invoiceID7" name="invoiceID"/>
    <attribute type="NUMBER" identity="true" id="invoiceItemID8" name="invoiceItemID"/>
    <attribute type="NUMBER" id="productID9" name="productID"/>
    <attribute type="NUMBER" id="quantity10" name="quantity"/>
  </entity>
  <mainScenario>
    <step stepNumber="1" action="ENTRY" entity="invoiceItemID8"/>
    <step stepNumber="2" action="SELECTION" entity="productID9">
        <extends useCase="UCProduct18" joinedByAttribute="productID20"/>
    </step>
    <step stepNumber="3" action="ENTRY" entity="quantity10"/>
    <step stepNumber="4" action="REQUEST_OPERATION" entity="InvoiceItem6"/>
    <step stepNumber="5" action="SYSTEM_ACTION" entity="InvoiceItem6"/>
    <step stepNumber="6" action="OPERATION_REPORT" entity="InvoiceItem6"/>
  </mainScenario>
 </useCase>
 <useCase usecaseID="UCCustomer11" name="Customer" precondition="Nema preduslova"
postcondition="Nema postuslova">
  <entity id="Customer12" name="Customer">
    <attribute type="NUMBER" identity="true" id="customerID13" name="customerID"/>
    <attribute id="firstname14" name="firstname"/>
    <attribute id="lastname15" name="lastname"/>
    <attribute id="address16" name="address"/>
    <attribute id="phone17" name="phone"/>
  </entity>
  <mainScenario>
    <step stepNumber="1" action="ENTRY" entity="customerID13"/>
    <step stepNumber="2" action="ENTRY" entity="firstname14"/>
    <step stepNumber="3" action="ENTRY" entity="lastname15"/>
    <step stepNumber="4" action="ENTRY" entity="address16"/>
    <step stepNumber="5" action="ENTRY" entity="phone17"/>
    <step stepNumber="6" action="REQUEST_OPERATION" entity="Customer12"/>
    <step stepNumber="7" action="SYSTEM_ACTION" entity="Customer12"/>
    <step stepNumber="8" action="OPERATION_REPORT" entity="Customer12"/>
  </mainScenario>
 </useCase>
 <useCase usecaseID="UCProduct18" name="Product" precondition="Nema preduslova"
postcondition="Nema postuslova">
  <entity id="Product19" name="Product">
    <attribute type="NUMBER" identity="true" id="productID20" name="productID"/>
    <attribute id="name21" name="name"/>
    <attribute type="NUMBER" id="price22" name="price"/>
  </entity>
  <mainScenario>
    <step stepNumber="1" action="ENTRY" entity="productID20"/>
    <step stepNumber="2" action="ENTRY" entity="name21"/>
    <step stepNumber="3" action="ENTRY" entity="price22"/>
    <step stepNumber="4" action="REQUEST_OPERATION" entity="Product19"/>
    <step stepNumber="5" action="SYSTEM_ACTION" entity="Product19"/>
    <step stepNumber="6" action="OPERATION_REPORT" entity="Product19"/>
  </mainScenario>
 </useCase>
</specification>
```

Слика 194: Основа улазне спецификације у текстуалном (XML) облику генерисана коришћењем графичког алата

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<extensions>
    <useCaseGuiExtension useCaseID="UCInvoice0"
                         uiTemplate="UI_FIELD_TYPE"
                         childToParentTemplate="CHILD_NOT_CHILD"
                         optionButtonsTemplate="OPTION_BUTTONS"
                         optionInsert="true"
                         optionUpdate="true"
                         optionDelete="true"
                         optionSearch="true"
                         entityDisplayName="Invoice"
                         entityCodeName="Invoice"
                         ignoreValidation="false">
        <scenarioStepGUIExtension attributeCodeName="invoiceID"
                                  attributeDisplayName="invoiceID"
                                  mainScenarioStepNumber="1"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="date"
                                  attributeDisplayName="date"
                                  mainScenarioStepNumber="2"
                                  component="Calendar"
                                  format=""
                                  visibleAttributes=""/>
        <scenarioStepGUIExtension attributeCodeName="customerID"
                                  attributeDisplayName="customerID"
                                  mainScenarioStepNumber="3"
                                  component="ComboBox"
                                  visibleAttributes="firstname lastname"
                                  showDetails="true"
                                  showDetailsTemplate="RIGHT_CLICK"/>
        <scenarioStepGUIExtension attributeCodeName="Invoice"
                                  attributeDisplayName="Invoice"
                                  mainScenarioStepNumber="4"/>
        <scenarioStepGUIExtension attributeCodeName="Invoice"
                                  attributeDisplayName="Invoice"
                                  mainScenarioStepNumber="5"/>
        <scenarioStepGUIExtension attributeCodeName="Invoice"
                                  attributeDisplayName="Invoice"
                                  mainScenarioStepNumber="6"/>
    </useCaseGuiExtension>
    <useCaseGuiExtension useCaseID="UCInvoice item5"
                         uiTemplate="UI_TABLE_TYPE"
                         childToParentTemplate="CHILD_TAB_TYPE"
                         optionButtonsTemplate="OPTION_BUTTONS"
                         optionInsert="true"
                         optionUpdate="true"
                         optionDelete="true"
                         optionSearch="true"
                         entityDisplayName="Invoice item"
                         entityCodeName="InvoiceItem"
                         ignoreValidation="false">
        <scenarioStepGUIExtension attributeCodeName="invoiceItemID"
                                  attributeDisplayName="invoiceItemID"
                                  mainScenarioStepNumber="1"
                                  component="TextField"/>
```

```
        <scenarioStepGUIExtension attributeCodeName="productID"
                                  attributeDisplayName="productID"
                                  mainScenarioStepNumber="2"
                                  component="ComboBox"
                                  visibleAttributes="name"
                                  showDetails="true"
                                  showDetailsTemplate="RIGHT_CLICK"/>
        <scenarioStepGUIExtension attributeCodeName="quantity"
                                  attributeDisplayName="quantity"
                                  mainScenarioStepNumber="3"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="InvoiceItem"
                                  attributeDisplayName="InvoiceItem"
                                  mainScenarioStepNumber="4"/>
        <scenarioStepGUIExtension attributeCodeName="InvoiceItem"
                                  attributeDisplayName="InvoiceItem"
                                  mainScenarioStepNumber="5"/>
        <scenarioStepGUIExtension attributeCodeName="InvoiceItem"
                                  attributeDisplayName="InvoiceItem"
                                  mainScenarioStepNumber="6"/>
    </useCaseGuiExtension>
    <useCaseGuiExtension useCaseID="UCCustomer11"
                         uiTemplate="UI_FIELD_TYPE"
                         childToParentTemplate="CHILD_NOT_CHILD"
                         optionButtonsTemplate="OPTION_BUTTONS"
                         entityDisplayName="Customer"
                         entityCodeName="Customer">
        <scenarioStepGUIExtension attributeCodeName="customerID"
                                  attributeDisplayName="customerID"
                                  mainScenarioStepNumber="1"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="firstname"
                                  attributeDisplayName="firstname"
                                  mainScenarioStepNumber="2"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="lastname"
                                  attributeDisplayName="lastname"
                                  mainScenarioStepNumber="3"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="address"
                                  attributeDisplayName="address"
                                  mainScenarioStepNumber="4"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="phone"
                                  attributeDisplayName="phone"
                                  mainScenarioStepNumber="5"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="Customer"
                                  attributeDisplayName="Customer"
                                  mainScenarioStepNumber="6"/>
        <scenarioStepGUIExtension attributeCodeName="Customer"
                                  attributeDisplayName="Customer"
                                  mainScenarioStepNumber="7"/>
        <scenarioStepGUIExtension attributeCodeName="Customer"
                                  attributeDisplayName="Customer"
                                  mainScenarioStepNumber="8"/>
    </useCaseGuiExtension>
```

```
    <useCaseGuiExtension useCaseID="UCProduct18"
                         uiTemplate="UI_FIELD_TYPE"
                         childToParentTemplate="CHILD_NOT_CHILD"
                         optionButtonsTemplate="OPTION_BUTTONS"
                         entityDisplayName="Product"
                         entityCodeName="Product">
        <scenarioStepGUIExtension attributeCodeName="productID"
                                  attributeDisplayName="productID"
                                  mainScenarioStepNumber="1"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="name"
                                  attributeDisplayName="name"
                                  mainScenarioStepNumber="2"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="price"
                                  attributeDisplayName="price"
                                  mainScenarioStepNumber="3"
                                  component="TextField"/>
        <scenarioStepGUIExtension attributeCodeName="Product"
                                  attributeDisplayName="Product"
                                  mainScenarioStepNumber="4"/>
        <scenarioStepGUIExtension attributeCodeName="Product"
                                  attributeDisplayName="Product"
                                  mainScenarioStepNumber="5"/>
        <scenarioStepGUIExtension attributeCodeName="Product"
                                  attributeDisplayName="Product"
                                  mainScenarioStepNumber="6"/>
    </useCaseGuiExtension>
</extensions>
```

Слика 195: Проширење улазне спецификације у текстуалном (XML) облику генерисано коришћењем графичког алата

8.1. Студијски примјер – развој десктоп апликације коришћењем *SilabUI* приступа

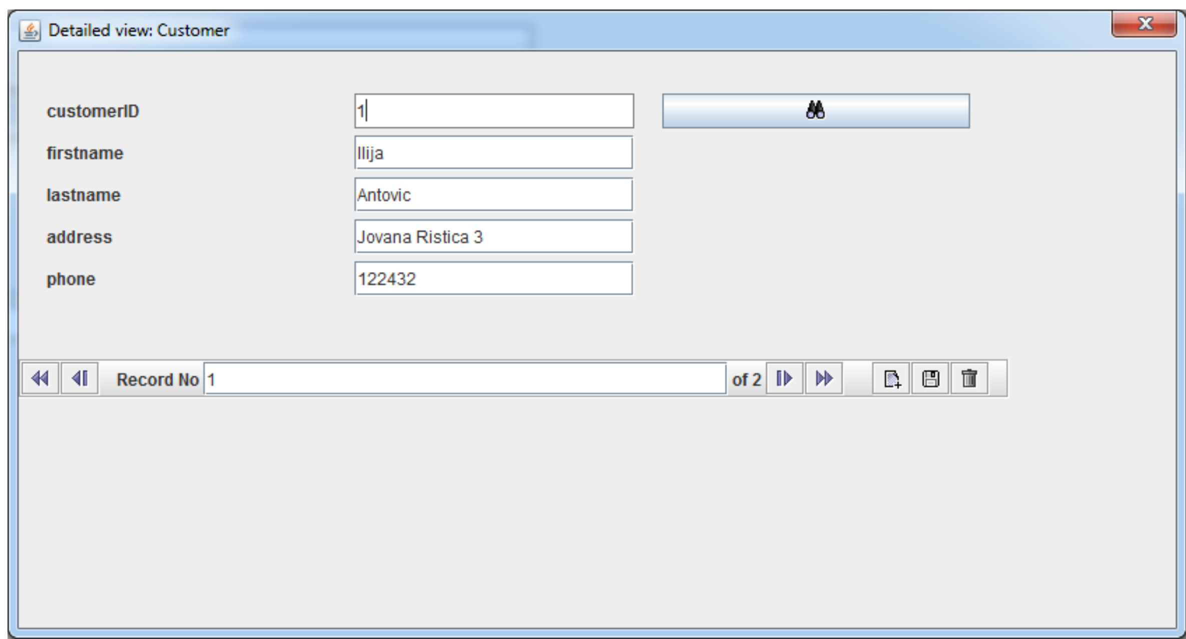
Коришћењем графичког алата за креирање улазне спецификације могуће је погренути генерисање програмског кода корисничког интерфејса десктоп апликација кликом на дугме *Generate SWING GUI* (Слика 193). Након генерисања програмског кода, добијено је осам *.java* датотека:

- *InvoicePanel.java*
- *InvoiceItemPanel.java*
- *InvoiceInsertPanel.java*

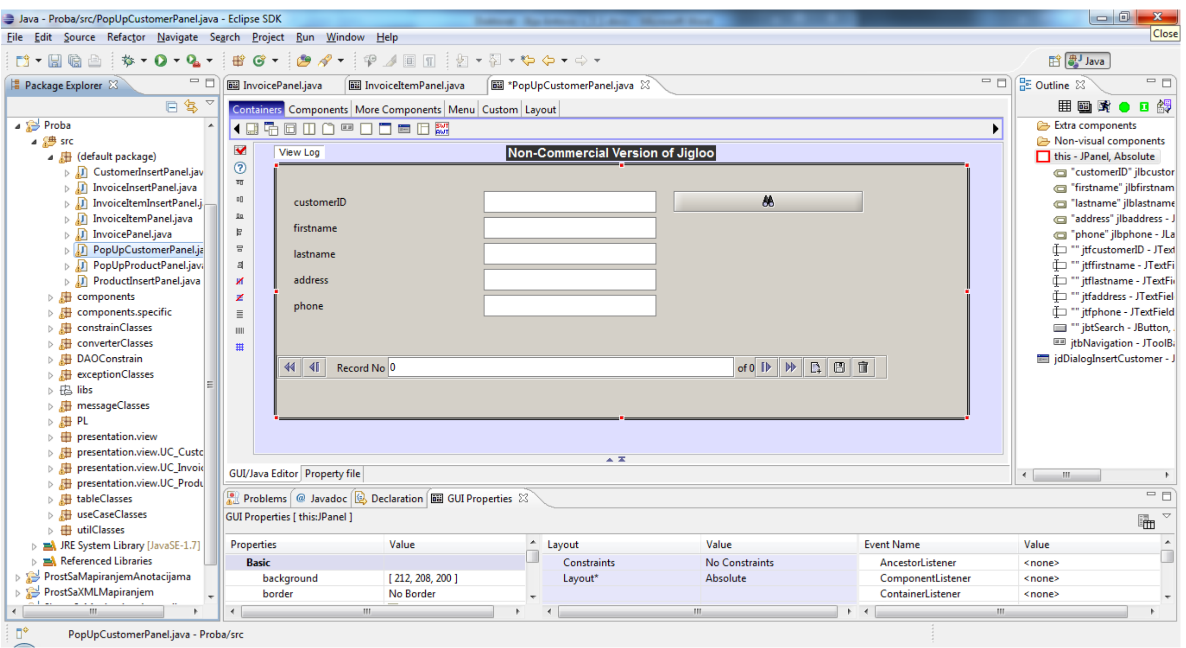
- *InvoiceItemInsertPanel.java*
- *PopUpCustomerPanel.java*
- *CustomerInsertPanel.java*
- *PopUpProductPanel.java*
- *ProductInsertPanel.java*

Следећи корак у креирању апликације је прављење јава пројекта у одабраном развојном окружењу и копирање генерисаних датотека у пројекат, као и копирање класе која има улогу трансферног објекта, након чега је потребно имплементирати методе класе *Controller*. За потребе истраживања, направљен је генератор који на основу улазне спецификације генерише програмски код класе *Controller* као и класа које су одговорне за перзистенцију података коришћењем система за управљање релационим базама података, али како је овај генератор ван опсега докторске дисертације и дио је једног ширег истраживања, даље неће бити разматран.

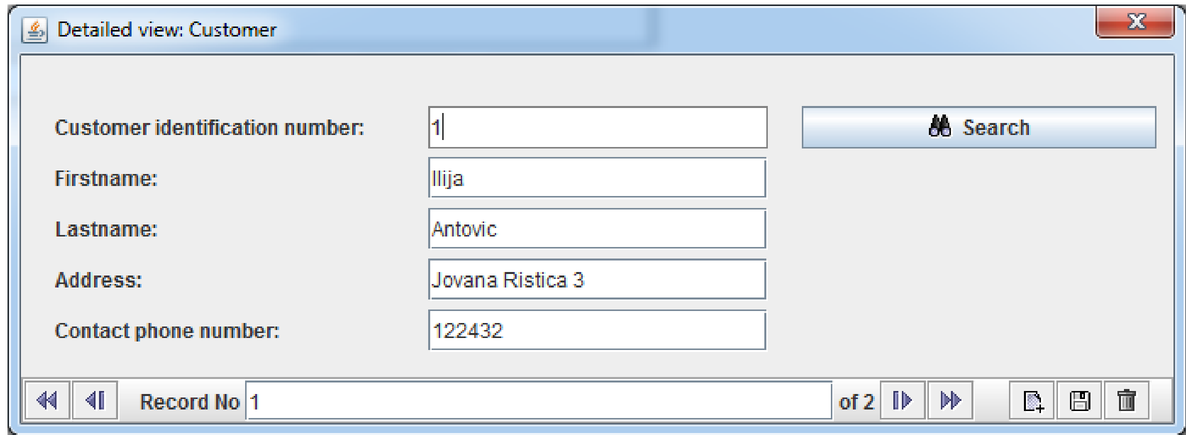
Апликацију је могуће покренути директно извршењем *main* методе класе *InvoicePanel,* након чега ће бити приказан генерисани кориснички интерфејс. Ако добијени кориснички интерфејс захтијева одређене измјене, могуће га је мијењати или директно приступајући генерисаном програмском коду, или коришћењем алата за визуелно креирање корисничког интерфејса. На примјеру панела за обраду података о купцима (*PopUpCustomerPanel.java*) биће приказан изглед корисничког интерфејса прије измјене (Слика 196), обраду коришћењем алата за визуелно креирање корисничког интерфејса (Слика 197), као и послије измјена (Слика 198).



Слика 196: Приказ генерисаног корисничког интерфејса за обраду података о купцима прије измјена

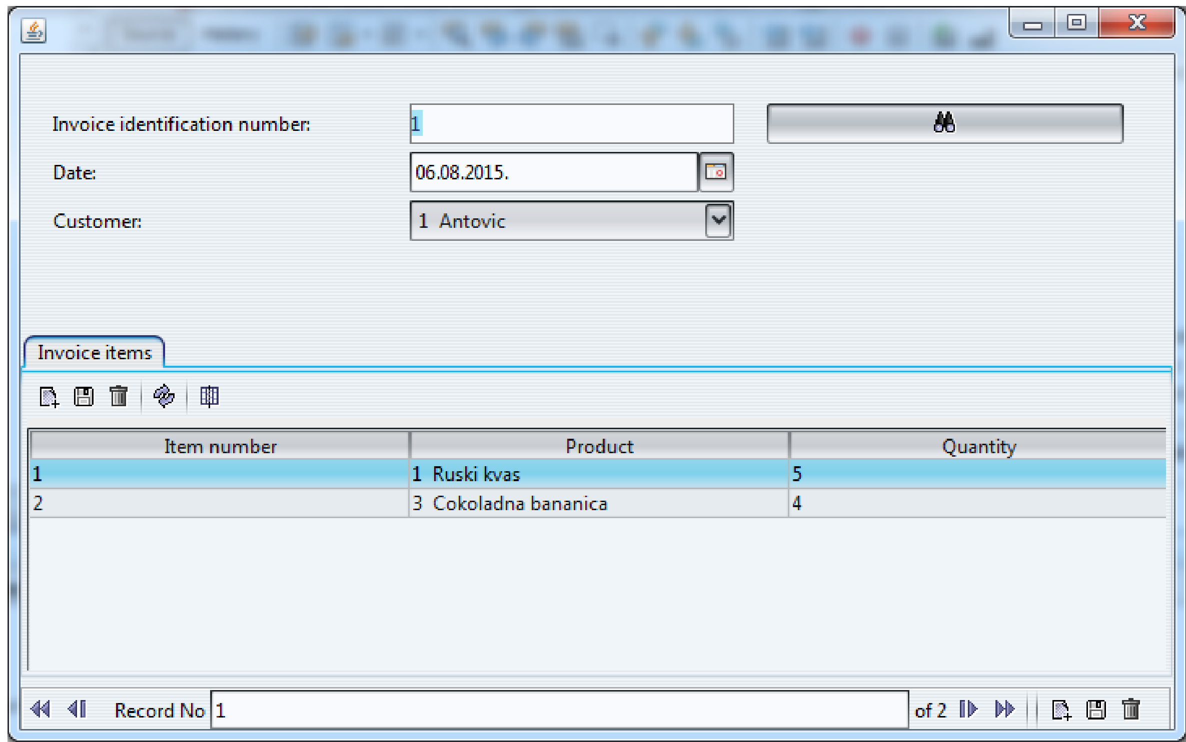


Слика 197: Обрада генерисаног корисничког интерфејса алатом за визуелно креирање корисничког интерфејса

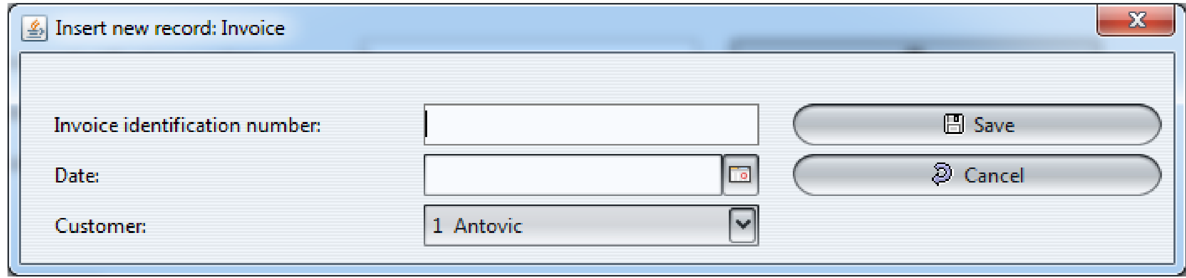


Слика 198: Приказ генерисаног корисничког интерфејса за обраду података о купцима послије измјена

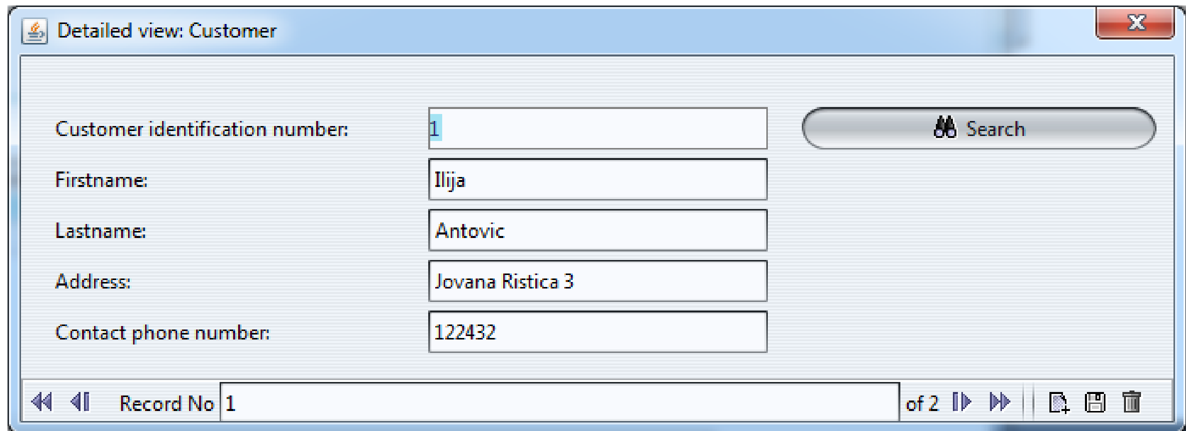
На сличан начин могуће је извршити обраду свих осталих генерисаних панела. У наставку ће бити приказан изглед резултујуће апликације.



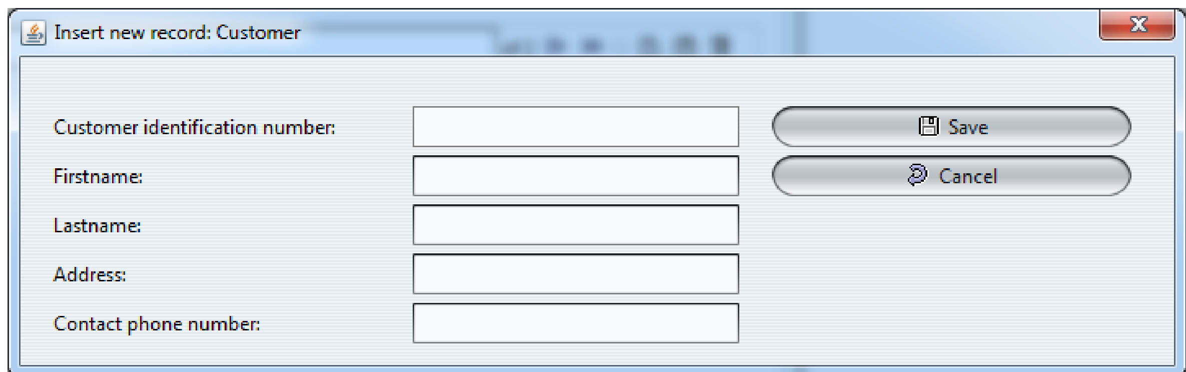
Слика 199: Резултујући кориснички интерфејс – *InvoicePanel*



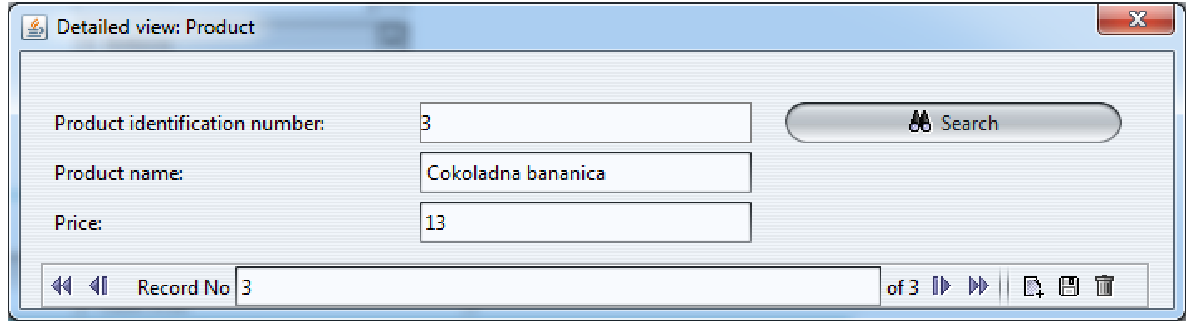
Слика 200: Резултујући кориснички интерфејс – *InvoiceInsertPanel*



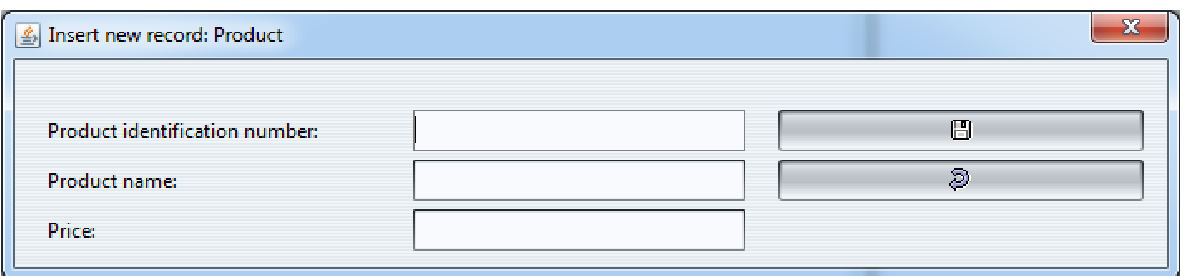
Слика 201: Резултујући кориснички интерфејс – *PopUpCustomerPanel*



Слика 202: Резултујући кориснички интерфејс – *CustomerInsertPanel*



Слика 203: Резултујући кориснички интерфејс – *PopUpProductPanel*



Слика 204: Резултујући кориснички интерфејс – *ProductInsertPanel*

8.2. Студијски примјер – развој веб апликације коришћењем *SilabUI* приступа

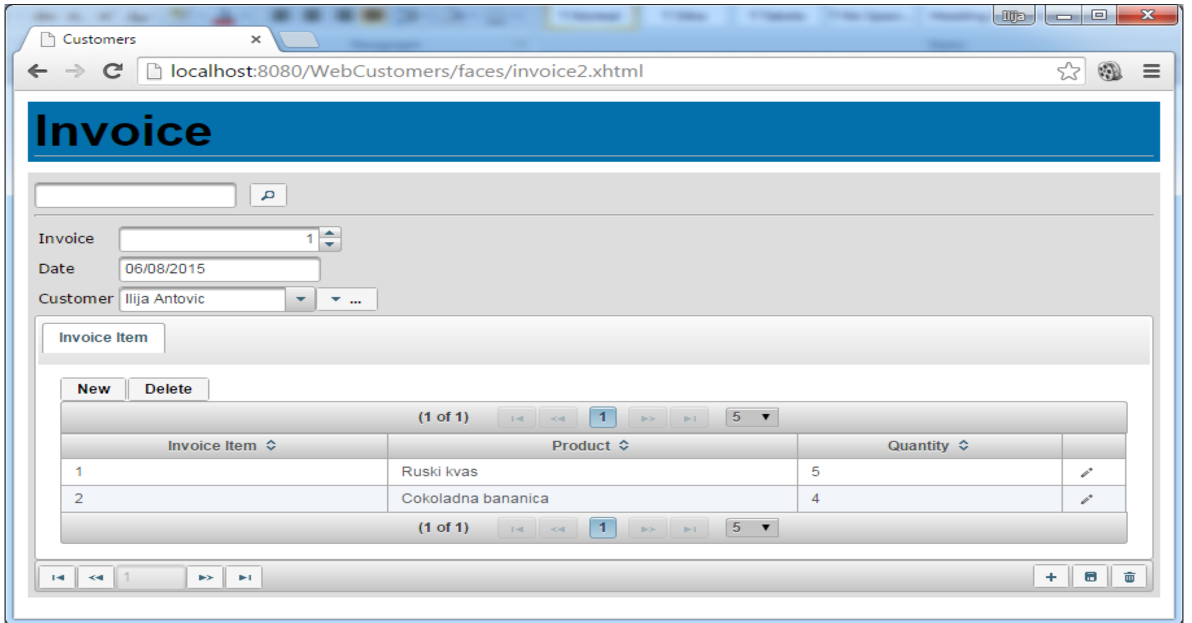
Коришћењем графичког алата за креирање улазне спецификације могуће је погренути генерисање програмског кода корисничког интерфејса веб апликација кликом на дугме *Generate JSF – PrimeFaces GUI* (Слика 193). Након генерисања програмског кода, добијено је седам датотека:

- *IncludeInvoice.xhtml*

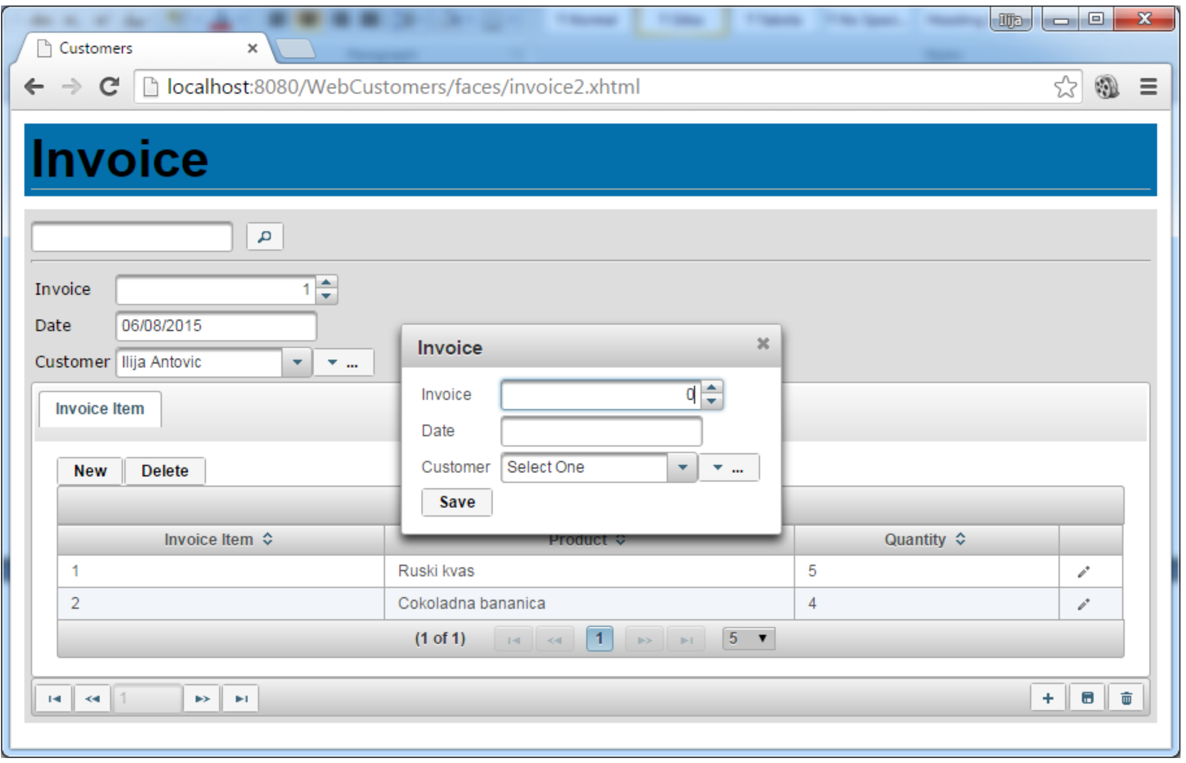
- *InvoiceMB.java*
- *CustomerConverter.java*
- *ProductConverter.java*
- *Controller.java*
- *messages.properties*
- *validation-msg. properties*

Следећи корак у креирању апликације је прављење јава веб пројекта у одабраном развојном окружењу и копирање генерисаних датотека у пројекат, као и копирање класе која има улогу трансферног објекта, након чега је потребно имплементирати методе класе *Controller*. Поред тога, потребно је у веб страницу укључити *IncludeInvoice.xhtml* страницу коришћењем <ui:include> тага.

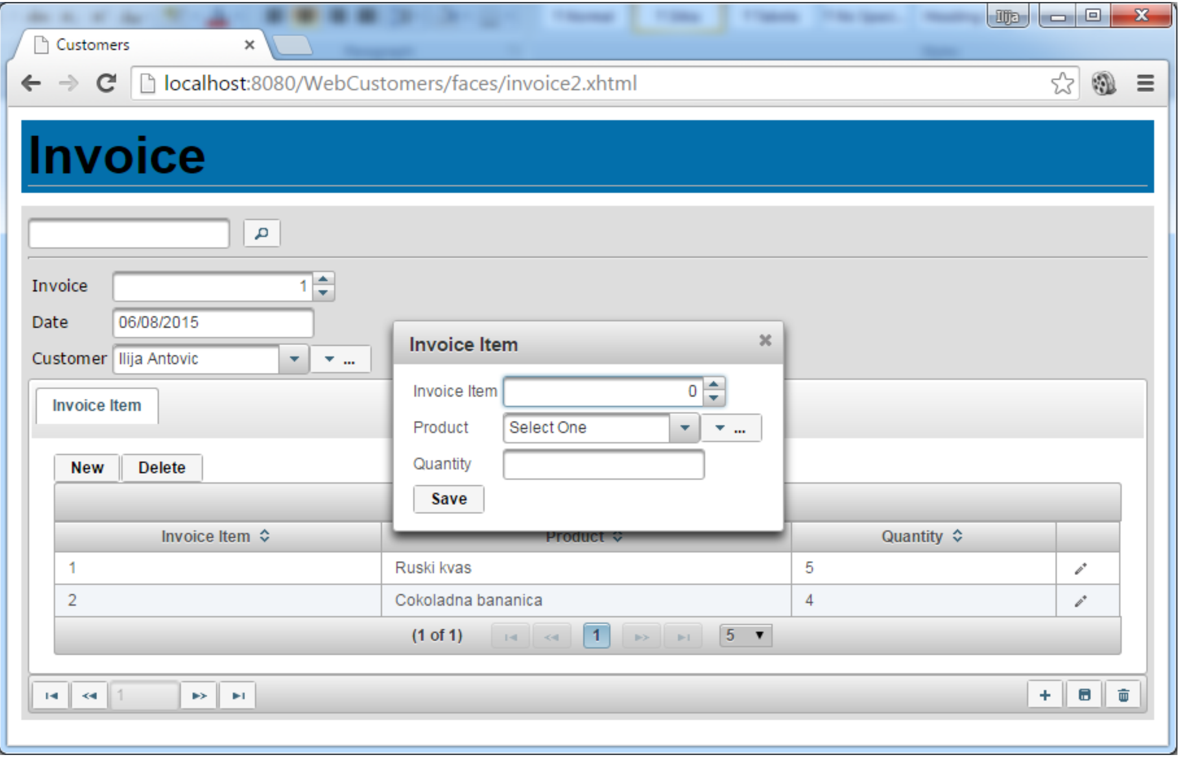
Након постављања апликације на сервер, могуће јој је приступити из веб клијента (*browser*). У наставку ће бити приказан резултујући кориснички интерфејс апликације.



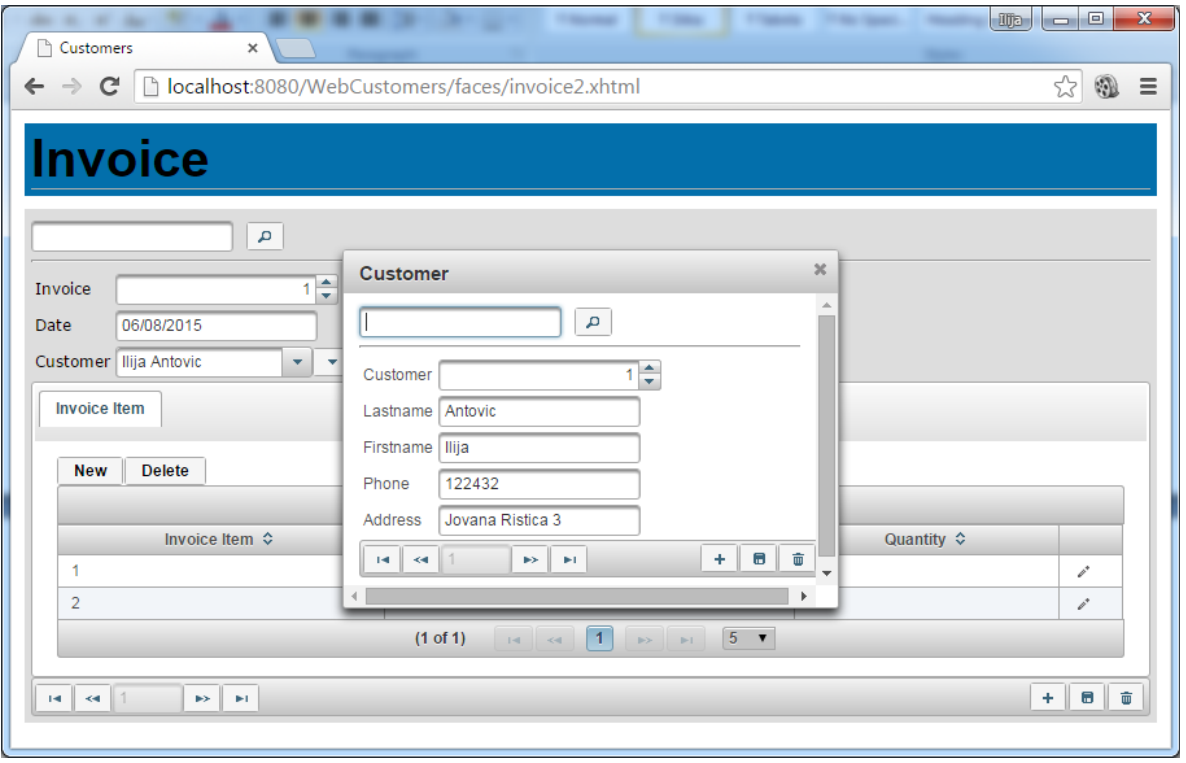
Слика 205: Резултујући кориснички интерфејс – *Invoice*



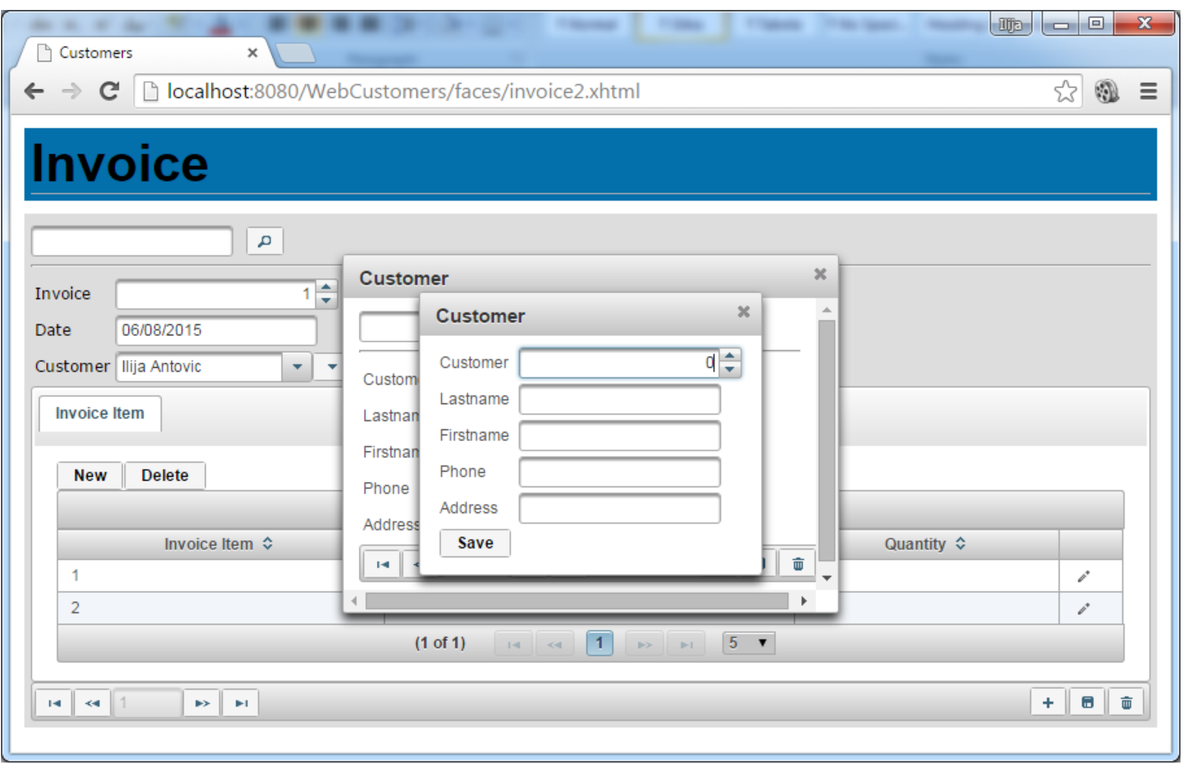
Слика 206: Резултујући кориснички интерфејс – *Invoice insert dialog*



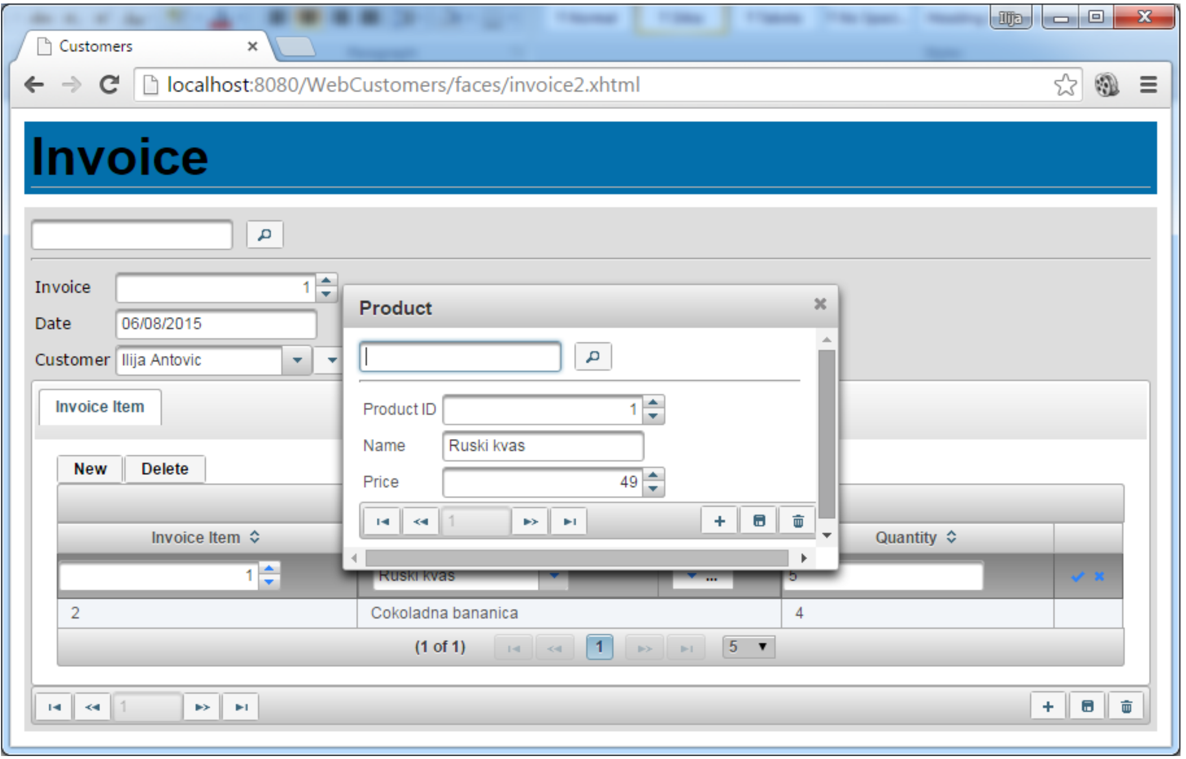
Слика 207: Резултујући кориснички интерфејс – *InvoiceItem insert dialog*



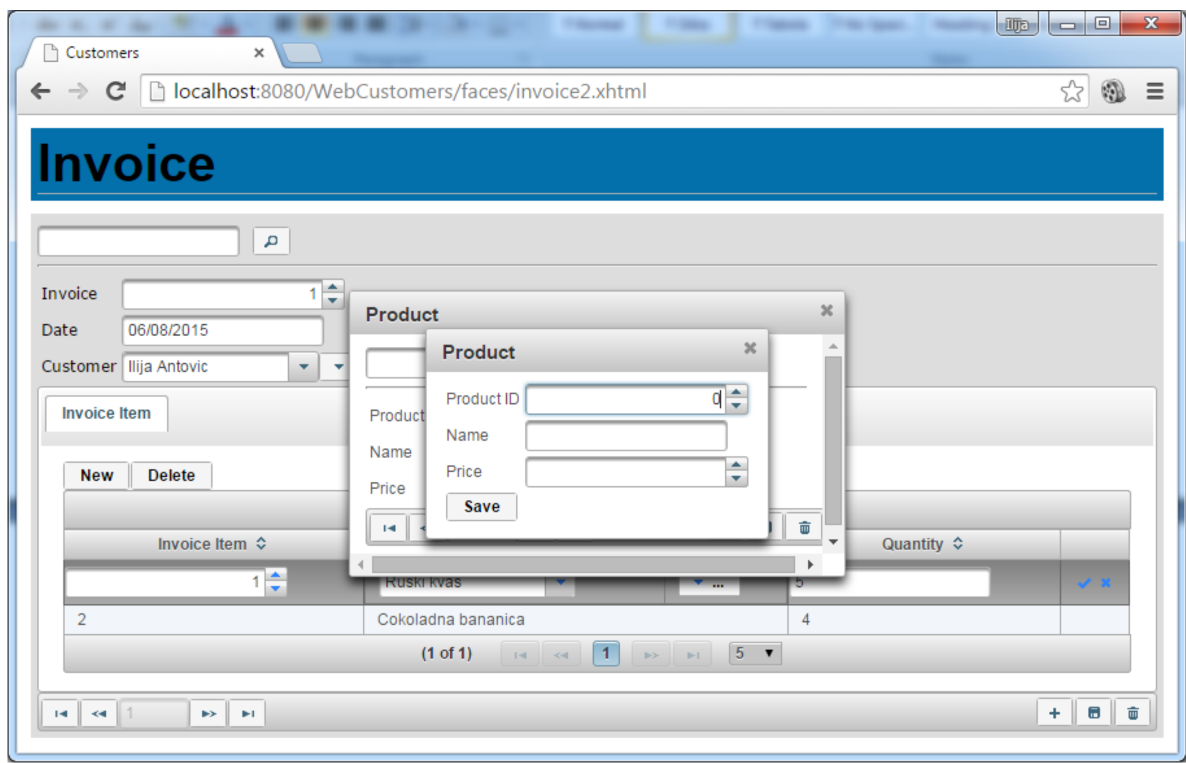
Слика 208: Резултујући кориснички интерфејс – *Customer dialog*



Слика 209: Резултујући кориснички интерфејс – *Customer insert dialog*



Слика 210: Резултујући кориснички интерфејс – *Product dialog*



Слика 211: Резултујући кориснички интерфејс – *Product insert dialog*

9. Закључак

Од својих најранијих искустава у развоју софтверских система, увиђао сам проблем трајања развоја које је праћено бројним понављајућим активностима, које не захтијевају велико интелектуално ангажовање, али захтијевају значајно вријеме при чему повећавају монотоност посла, а у исто вријеме су извор грешака. Ове активности се јављају приликом развоја свих дијелова система, а најочигледније су приликом формирања модела података (објектног и релационог), пресликавања модела података из релационог у објектни и обратно, али и приликом прављења корисничког интерфејса који је великим дијелом завистан од модела података. У то вријеме, постојало је много различитих алата који су омогућавали превазилажење проблема објектно-релационог пресликавања, а поред ових алата, који тада још увијек нису достигли ниво зрелости и распрострањености коришћења, развијао сам и своја рјешења која су пружала генеричке механизме којим сам постизао аутоматизацију објектно-релационог пресликавања. Временом су се у домену објектно-релационог пресликавања издвојили алати који су својим квалитетом и једноставношћу употребе стекли значајан број корисника и као посљедицу постали стандардни дијелови развојног процеса. Веома брзо, ако се говори о Јава платформи, на основу добре праксе која је преузета од најзначајнијих алата за објетно-релационо пресликавање, створена је стандардна спецификација (*Java Persistence API*) пружајући корисницима јединствени, стандардни интерфејс за обезбјеђивање објектно-

релационог пресликавања, а у исто вријеме дозвољавајући флексибилност у избору конкретног алата који ће се приликом извршења користити за пресликавање.

Са друге стране, веома сличан проблем постојао је и приликом прављења корисничког интерфејса софтверског система. Наиме, број потребних графичких компонената, врста компонената, њихово именовање у програмском коду, називи које корисник треба да види на корисничком интерфејсу, такође су у великој мјери били условљени моделом података. У односу на проблем објектно-релационог пресликавања, који дефинише начин комуникације између два рачунарска система (софтверског система са системом за управљање базом података), овај проблем је утолико сложенији јер се њиме дефинише начин комуникације између човјека и софтверског система, па алат који би омогућио аутоматизацију пројектовања и имплементације корисничког интерфејса мора омогућити много већу флексибилност и испуњење различитих захтјева корисника. 2006. године, када сам почео да се бавим ријешавањем овог проблема, постојало је веома мало алата за превазилажење овог проблема, са прилично скромним могућностима, а нити један од њих око себе није окупио значајнији број корисника. Визија коју сам имао тада, заједно са својим колегама из Лабораторије за софтверско инжењерство, била је да се направи алат који би омогућио аутоматско генерисање корисничког интерфејса на бази модела који постоје у раним фазама софтверског пројекта, и који би пружио рјешење на презентационом нивоу са једнаким успјехом и популарношћу која је постигнута на перзистенционом нивоу коришћењем алата за објектно релационо пресликавање.[70]

[70] Током израде дисертације, често сам размјењивао ставове о алатима за генерисање корисничког интерфејса са творцем Metawidget оквира (Richard Kennard), који је описан у четвртом поглављу. Наши ставови су се по одређеним питањима поклапали, док су се по неким значајно разликовали, али је интерасантна чињеница да нас је обојицу, да започнемо истраживање на овом пољу, инспирисала иста идеја – направити пробој на овом пољу сличан пробоју који се догодио на пољу перзистентних оквира.

Иако су се у међувремену појавили алати са много бољим могућностима, и до данас се није појавио алат који је стекао значајнију популарност међу корисницима, нити стандард око којег би се окупила стручна и научна заједница.

Анализирајући постојеће приступе и алате који су развијени за ову намјену, уз моје искуство, интуицију и идеје које сам имао о будућем алату и карактеристикама које ће га одликовати, примијетио сам да су поједине карактеристике заједничке и прихваћене код готово свих постојећих алата, нпр. заснованост на доменском моделу и идеја да се креира функционалан интерфејс у раним фазама софтверског пројекта. Међутим, по осталим карактеристикама се постојећи приступи значајно разликују нпр. поред доменског модела, приступи се разликују у односу на друге моделе на којима су засновани, разликују се у начину генерисања корисничког интерфејса, подршци за коришћење различитих шаблона корисничког интерфејса, постојању различитих алата за креирање улазне спецификације и сл.

Сматрао сам да у свом истраживању треба да се ослоним на постојећу добру праксу у генерисању корисничког интерфејса, али како су се моје идеје значајно разликовале од постојећих приступа, и како ни међу њима није постојала сагласност по бројним питањима, сматрао сам да бих направио грешку уколико истраживање темељим на приступима који нису имали значајнијег успјеха у пракси. Мислио сам да би на такав начин био креиран само још један алат без веће шансе да заживи у пракси. Зато сам ријешио да, прије свега, свој приступ темељим на теорији и пракси у развоју софтверских система, у други план стављајући постојеће приступе који овај процес аутоматизују. Основни принцип који сам поставио био је да нови приступ треба да буде еволуција процеса развоја софтвера, а не револуција! Као директна посљедица тога, нови приступ је заснован на случајевима коришћења који су уско везани за доменски модел, јер се у мојој досадашњој пракси спецификација софтверских

захтјева, која је полазна тачка у пројектовању софтверског система, темељи баш на ова два артифакта.

Како бих доказао одрживост оваквог приступа, за потребе свог магистарског рада [Antovic10] сам развио алат који на бази спецификације случајева коришћења, доменског модела и информација везаних за кориснички интерфејс аутоматски генерише функционални кориснички интерфејс Јава десктоп пословних апликација.

Иако сам био задовољан резултатима истраживања објављеним у магистарском раду, а касније и у значајним научним часописима и на конференцијама [Savic11] [Antovic12] [Savic12] [Antović14] [DaSilva15], сматрао сам да истраживање треба наставити и проширити како би се обухватиле теме које су биле изван опсега магистарског рада, а све у циљу приближавања оваквог приступа корисницима, односно директној примјени у софтверској индустрији.

Како бих што боље сагледао потребе потенцијалних корисника оваквог алата, поред добрих предлога добијених од стране рецензената радова које сам објављивао, спровео сам научно испитивање у форми анкете. Циљ анкете био је утврђивање преовлађујућег става носилаца софтверске индустрије – искусних софтверских инжењера, по питању елемената које алат за генерисање корисничког интерфејса мора да посједује, како би га прихватили и користили у процесу развоја софтвера. Ови елементи су посматрани као захтјеви које будући алат за генерисање корисничког интерфејса мора да испуни како би био прихваћен од стране потенцијалних корисника. Поред тога, како би се обезбиједило теоријско утемељење новог приступа, извршена је анализа различитих приступа развоју корисничког интерфејса, разматране су различите методе и технике које се користе у инжењерингу софтверских захтјева и направљен је преглед добре праксе у пројектовању корисничког интерфејса, уз анализу карактеристика корисничког интерфејса

различитих типова апликација. У наставку ћу дати кратак преглед карактеристика које чине *SilabUI* приступ.

Након увода, у другом поглављу дисертације су разматрани различити приступи у развоју корисничког интерфејса (скицирање, визуелно креирање, коришћење језика за спецификацију, као и коришћење модела и моделом вођени развој), а за сваки приступ су приказане предности и недостаци. У овом контексту *SilabUI* приступ се може посматрати као приступ који је заснован на моделима, и који користи предности моделом вођеног развоја, али исто тако комбинује добре карактеристике осталих приступа. У седмом поглављу су приказани начини за спецификацију модела, на основу кога ће се вршити генерисање корисничког интерфејса, међу којима је изузетно значајна могућност директног креирања улазне спецификације која је заснована на *XML*-у, као једном виду језика за спецификацију будућег корисничког интерфејса. Поред тога, могућност коришћења шаблона корисничког интерфејса (поглавље 6.) и њиховог комбиновања, уз једноставност промјене одабраних шаблона кроз промјену улазне спецификације, омогућава прављење корисничког интерфејса који се може користити као прототип приликом прикупљања и валидације корисничких захтјева, па се на овај начин постижу све предности које пружају алати за скицирање корисничког интерфејса. *SilabUI* приступ предвиђа генерисање програмског кода корисничког интерфејса који се касније може учитати и бити „препознат“ од стране алата за визуелно креирање корисничког интерфејса, и на тај начин се могу искористити предности алата за даљу обраду генерисаног корисничког интерфејса, уколико за тиме постоји потреба.

У трећем поглављу разматрани су постојећи приступи инжењерингу софтверских захтјева, техника за прикупљање захтјева, дат је преглед карактеристика двије доминантне технике за спецификацију захтјева (случајеви коришћења и

корисничке приче), и идентификоване су предности коришћења случајева коришћења (прије свега структурираност, цјеловитост и ниво детаљности) у контексту аутоматизације процеса креирања корисничког интерфејса засноване на захтјевима. Поред тога објашњена је и улога доменског модела и његова веза са случајевима коришћења. На крају су дате препоруке зе спецификацију корака у сценарију случаја коришћења и њихов утицај на будући кориснички интерфејс. Узимајући у обзир изведене закључке формиран је мета-модел за креирање улазне спецификације која је у потпуности заснована на случајевима коришћења. Поглавље 3 завршава објашњењем улоге прототипова у инжењерингу захтјева, гдје се истичу предности коришћења еволутивних прототипова. *SilabUI* приступ омогућава генерисање програмског кода корисничког интерфејса који се може извршити, а поред тога омогућена је и његова измјена, па добијени кориснички интерфејс се може посматрати као извршиви еволутивни прототип.

У поглављу 4 покушано је да се дође до одговора на питања који су разлози за чињеницу да ни један од постојећих алата није успио да око себе окупи широк круг корисника, односно да задовољи потребе привреде, који су то захтјеви које алати за генерисање корисничког интерфејса морају да испуне како би били прихваћени од стране потенцијалних корисника и да ли данас актуелни алати испуњавају ове захтјеве. Зато је прво спроведено поменуто испитивање – анкета. У наставку ће бити приказана листа дефинисаних критеријума, са карактеристикама које су имплементиране у *SilabUI* приступ, а које су од стране испитаника препознате као потребне, уз карактеристике које су прихваћене као добра пракса из постојећих приступа.

- **Модели на којима је заснована улазна спецификација**

Анализирајући изнесене ставове испитаника по неколико различитих питања везаних за моделе који се користе у раним фазама софтверског пројекта може се закључити да случајеви коришћења тијесно повезани са доменским моделом доминирају као полазни модел у развоју софтвера. Уз случајеве коришћења и доменски модел, испитаници су у значајној мјери истакли и потребу прављења прототипова корисничког интерфејса. Овакви резултати само потврђују закључке добијене у трећем поглављу разматрањем постојећих приступа инжењерингу софтверских захтјева и техника за прикупљање захтјева. Зато мета-модел за креирање улазне спецификације омогућава директно повезивање елемената спецификације случајева коришћења и елемената доменског модела, уз могућност придруживања информација које одређују жељене специфичности будућег корисничког интерфејса. Иако су овакви резултати на линији досадашњих личних искустава које сам имао у развоју софтвера, па су за мене били очекивани, поставља се питање зашто нити један од разматраних алата (осим само дјелимично *WebRatio* и *BizAgi BMP*) за аутоматизацију развоја корисничког интерфејса није на овај начин пројектовао улазну спецификацију. Сматрам да је ово један од значајнијих разлога слабог прихватања посматраних приступа од стране ширег круга корисника.

- **Зависност улазне спецификације од модела података**

Сви посматрани алати су у великој мјери зависни од модела података, а потребу за успостављањем ове зависности приликом дефинисања улазне спецификације потврђују и изнесени ставови испитаника. Оно што треба нагласити јесте да улазна спецификација не би требала да буде искључиво заснована на моделу података, као што је случај са већином посматраних

алата. Наиме, као што резултати истраживања потврђују, а што је дијелом изнесено у образложењу претходног критеријума, модел података, прије свега доменски модел (који описује структуру софтверског система) треба да прати спецификацију случајева коришћења, која поред структуре описује и понашање корисника приликом интеракције са системом, као и спецификацију тога шта систем треба да ради (понашање софтверског система). Сматрам да улазна спецификација тек у том случају постаје семантички довољно богата за аутоматизацију развоја корисничког интерфејса.

- **Начин дефинисања улазне спецификације**

Ставови испитаника по питању начина дефинисања улазне спецификације су подијељени, прије свега између визуелног моделовања и графичког алата који би водио корисника кроз процес дефинисања спецификације, али значајан број испитаних сматра да је пожељни начин дефинисања улазне спецификације ручним куцањем кода доменски специфичног језика развијеног за ту намјену. Са друге стране, посматрани алати за аутоматизацију развоја корисничког интерфејса углавном омогућавају само један начин дефинисања улазне спецификације. *SilabUI* приступ омогућава сва три начина дефинисања улазне спецификације.

- **Начин креирања корисничког интерфејса**

У погледу начина креирања корисничког интерфејса, односно да ли алат за аутоматизацију развоја корисничког интерфејса треба да као резултат обезбиједи програмски код корисничког интефејса, или кориснички интерфејс који ће бити резултет интерпретације улазне спецификације у

вријеме извршења, испитаници су убједљиво опредијељени за генерисање програмског кода корисничког интерфејса. *SilabUI* приступ омогућава управо овакав начин креирања корисничког интерфејса, уз омогућавање кориснику потпуне контроле над генерисаним програмским кодом. По овом критеријуму се разликују и посматрани алати, једни подржавају један, а други подржавају други приступ.

- **Измјењивост генерисаног програмског кода**

Као што је претходно наглашено, *SilabUI* приступ омогућава кориснику потпуну контролу над генерисаним програмским кодом. Иако неки од посматраних алата такође обезбјеђују генерисање програмског кода корисничког интерфејса, ови алати углавном не дозвољавају кориснику да врши измјене у генерисаном програмском коду.

- **Подржана могућност избора шаблона корисничког интерфејса**

Већински став испитаника о информацијама везаним за кориснички интерфејс којим је потребно проширити улазну спецификацију је да спецификација треба да омогући избор између различитих шаблона корисничког интерфејса. Поред тога, много мањи, али ипак значајан проценат испитаних сматра да корисник треба да има могућност избора специфичних графичких компонената. Посматрани алати за аутоматизацију развоја корисничког интерфејса не пружају могућност избора шаблона корисничког интерфејса (изузев ограничене подршке алата *WebRatio*), док поједини омогућавају избор специфичних графичких компонената. *SilabUI* приступ омогућава избор између различитих шаблона корисничкког интерфејса (о којима је било ријечи у шестом

поглављу), а такође кориснику пружа флексибилност при избору графичких компоненти.

- **Подршка за различите типове апликација**

Као и већина посматраних алата за аутоматизацију развоја корисничког интерфејса, *SilabUI* приступ такође омогућава развој корисничког интерфејса за различите типове софтверских система, узимајући у обзир све специфичности различитих типова софтверских система (о чему је било ријечи у петом поглављу), коришћењем различитих имплементационих технологија. Резултати испитивања такође показују да испитаници ова питања сматрају веома значајним и потребним дијеловима алата за аутоматизацију развоја корисничког интерфејса.

У уводу ове дисертације постављено је неколико циљева које истраживање треба да достигне:

- Циљ овог рада је да се развије модел за спецификацију корисничког захтјева који ће бити довољно семантички богат како би на основу њега било могуће не само пројектовање и имплементација корисничког интерфејса, већ и аутоматизација овог процеса.
- Циљ је да модел на основу којег ће се вршити генерисање програмског кода корисничког интерфејса садржи у себи све потребне информације како би било могуће задовољење свих постављених критеријума.
- Поред тога, на основу анализе начина интеракције корисника са системом, циљ је да се дође до скупа могућих шаблона корисничког интерфејса којим се различити кориснички захтјеви могу приказати.
- Крајњи циљ овог рада је да се у што већој мјери смање вријеме и напор, а тиме и трошкови процеса имплементације корисничког интерфејса

апликације као најзахтјевнијег дијела процеса имплементације читавог софтверског система.

На основу постављених циљева истраживања, постављене су и хипотезе истраживања:

- Полазна хипотеза истраживања је да је могуће успоставити везу између модела случајева коришћења и жељеног корисничког интерфејса. Доказивање постојања јасних веза између ових елемената омогућава формирање семантички богатог мета-модела заснованог на моделу случајева коришћења и информацијама везаним за кориснички интерфејс који омогућава аутоматизацију процеса генерисања извршивог програмског кода корисничког интерфејса.
- Модел једног корисничког захтјева формиран коришћењм поменутог мета-модела може се аутоматски трансформисати у више шаблона корисничког интерфејса, задржавајући жељену функционалност.
- Овај модел се, такође, може аутоматски трансформисати у програмски код корисничког интерфејса на различитим имплементационим технологијама за различите типове апликација
- Апликација развијена током овог истраживања – генератор – треба да покаже да је могућа аутоматизација процеса креирања корисничког интерфејса на основу предложеног модела у складу са дефинисаним критеријумума, како би се добио извршиви програмски код у различитим имплементационим технологијама.

На основу претходно изнесеног може се закључити да су достигнути циљеви дисертације и потврђене постављене хипотезе. У наставку ће бити приказани научни доприноси спроведеног истраживања:

- Преглед различитих техника спецификације корисничких захтјева, техника за креирање модела података и значај прототиповања
- Преглед и упоредна анализа постојећих приступа и алата који аутоматизују процес имплементације корисничког интерфејса
- Идентификација и образложење критеријума и карактеристика које генератор корисничког интерфејса мора да задовољава како би могао бити прихваћен од стране ширег круга корисника у односу на постојеће генераторе
- Откривање веза између модела случајева коришћења и будућег корисничког интерфејса
- Откривање правила пресликавања између корисничког интерфејса развијеног за различите типове апликација (десктоп, веб и мобилне апликације)
- Дефинисање семантички богатог модела корисничких захтјева заснованог на моделу случајева коришћења и информацијама везаним за кориснички интерфејс како би на основу њега било могуће не само пројектовање и имплементација корисничког интерфејса, већ и аутоматизација овог процеса.
- Развој алата (алат за визуелно моделовање и алат у виду графичке апликације која води корисника корак по корак кроз процес креирања улазне спецификације) за креирање улазне спецификације која се користи у процесу аутоматског генерисања програмског кода корисничког интерфејса

- Аутоматизација процеса креирања корисничког интерфејса апликације у различитим имплементационим технологијама уз задовољење дефинисаних критеријума
- Идентификација скупа могућих шаблона корисничког интерфејса којим се различити кориснички захтјеви могу приказати задржавајући жељену функционалност.

Евентуални даљи правци истраживања могли би се односити на:

- Развој алата за аутоматско генерисање корисничког интерфејса за друге популарне платформе и имплементационе технологије (*.NET*, *PHP*, *Android*, *iOS*…)
- Обогаћивање мета-модела могућношћу спецификације сложенијих валидационих правила и то прије свега за елементе:
  - предуслови случаја коришћења;
  - постустуслови случаја коришчења;
  - алтернативна сценарија:
    - ограничења на вриједности атрибута;
    - ограничења на међузависност атрибута;
- Обезбјеђивање прилагодљивости (адаптабилности) корисничког интерфејса, односно имплементације концепта који омогућава промјену корисничког интерфејса приликом промјене у окружењу, задржавајући жељене функционалности:
  - адаптабилност на различите кориснике;

  - адаптабилност на различите технологије;

- Интеграција са постојећим алатима за генерисање корисничког интерфејса, прије свега трансформацијом модела које користе различити приступи. Оваква интеграција омогућила би корисницима да користе предности различитих алата на тај начин што би нпр. улазну спецификацију могли креирати коришћењем једног алата, а затим је трансформисали и генерисање корисничког интерфејса вршили коришћењем другог алата.

- Интеграција алата са развојним окружењима. Различити алати за формирање улазне спецификације, као и различити алати за генерисање корисничког интерфејса у *SilabUI* приступу, представљају софтверске системе између којих не постоји директна зависност, већ се она успоставља посредно преко улазне спецификације. Такође, поменути алати нису везани ни за једно конкретно развојно окружење, већ корисник може генерисани програмски код по потреби обрађивати у било ком развојном окружењу које подржава рад са одређеном имплементационом технологијом. Иако оваква врста независности алата доприноси флексибилности и независном развоју нових и измјени постојећих алата, из угла корисника алата може представљати својеврсно ограничење. Овај правац истраживања сам поставио као један од главних приоритета у наредном периоду, а одређени кораци у том правцу су већ начињени. Идеја је да корисник који је навикнут на коришћење једног развојног окружења, приликом инсталације може да изабере које алате, који припадају *SilabUI* приступу, жели да интегрише са окружењем (нпр. само један начин креирања улазне спецификације,

и само алат за генерисање Јава веб апликација), и да комплетан процес развоја, укључујући и евентуалне измене генерисаног програмског кода, може да изврши „на једном мјесту“.

- Аутоматизација генерисања тест случајева и корисничке документације. Поред генерисања програмског кода корисничког интерфејса, модел улазне спецификације садржи довољно информација и за генерисање тест случајева који би пратили генерисани програмски код корисничког интерфејса, као и информације за генерисање пратеће документације (документовање програмског кода, дијела пројектне документације, корисничка упутства и сл.). Ове особине сам до сада сматрао као нешто што превазилази опсег истраживања, и само дијелом су разматране у досадашњем истраживању, али ће у наредном периоду постати један од значајнијих предмета истраживања.

Сматрам да резултати истраживања представљени у овој дисертацији представљају стабилну полазну основу за даља истраживања усмјерена ка дефинисању стандарда, који могу бити прихваћени од стране шире научне и стручне заједнице.

10. Литература

[Abbott83] R. Abbott, *Program Design by Informal English Descriptions*, Communications of the ACM, vol. 26, 1983.

[Acerbis07] R. Acerbis, A. Bongio, M. Brambilla, S. Butti, *WebRatio 5: An Eclipse-based CASE tool for engineering Web applications*, Web Engineering, Proceedings of 7th International Conference, ICWE 2007 Como, Italy, 2007.

[Alhir03] S. Si Alhir, *Understanding the Model Driven Architecture (MDA),* Methods & Tools, 2003.

[Alor-Hernandez14] G. Alor-Hernández, V. Y. Rosales-Morales, L. O. Colombo-Mendoza, *Frameworks, Methodologies, and Tools for Developing Rich Internet Applications (Advances in Web Technologies and Engineering (Awte))*, IGI Global, 2014.

[AlphaSimple] *AlphaSimple*, http://www.alphasimple.com, preuzeto 30.03.2013.

[AlphaSimpleDoc] *AlphaSimple*, http://www.alphasimple.com, preuzeto 30.03.2013.

[Alur03] D. Alur, J. Crupi, D. Malks, *Core J2EE™ Patterns: Best Practices and Design Strategies*, Second Edition, Prentice Hall, 2003.

[Ambler04] S. W. Ambler, *The Object Primer – Agile Model Driven Development with UML 2*, Cambridge University Press, 2004.

[Antovic10] I. Antović, *Razvoj modela i alata za generisanje korisničkog interfejsa na osnovu modela slučajeva korišćenja i modela podataka*, Magistarski rad, Fakultet organizacionih nauka, Univerzitet u Beogradu, 2010.

[Antovic12] I. Antović, S. Vlajić, M. Milić, D. Savić, V. Stanojević, *Model and software tool for automatic generation of user interface based on use case and data model*, IET Software, vol. 6, issue 6, 2012.

[Antović14] I. Antović, D. Savić, V. Stanojević, Automatic Generation of Executable UI Prototypes Using SilabREQ Language, ICT and Management, SymOrg, 2014.

[ApacheIsis] *Apache Isis*, http://isis.apache.org/, preuzeto 30.03.2013.

[BABOK] *A Guide to the Business Analysis Body of Knowledge® (BABOK® Guide)*, Version 2.0, International Institute of Business Analysis, Toronto, Ontario, Canada, 2009.

[Barbier13] M. F. Barbier, *Enterprise Model-Driven Development with BLU AGE – A Netfective Technology White Paper,* NETFECTIVE TECHNOLOGY, http://www.omg.org/mda/mda_files/White_paper_Netfective_technology.pdf, 2013.

[Belenguer03] J. Belenguer , J. Parra , I. Torres , P. J. Molina, *HCI Designers and Engineers: It is possible to work together?*, INTERACT 2003 Workshop on Bridging the Gap Between Software Engineering and Human-Computer Interaction, 2003.

[Berenbach09] B. Berenbach, D. J. Paulish, J. Kazmeier, A. Rudorfer, *Software & Systems Requirements Engineering In Practice*, The McGraw-Hill Companies, USA, 2009.

[BizAgi] *BizAgi*, http://www.bizagi.com/, preuzeto 30.03.2013.

[BPMN] *Object Management Group – Business Process Model and Notation*, http://www.bpmn.org/, preuzeto 10.04.2015.

[Buzejic11] S. Buzejić, *Komparativna analiza alata za generisanje Java korisničkog interfejsa*, Master rad, Fakultet organizacionih nauka, Univerzitet u Beogradu, 2011.

[Cameleon] *Cameleon reference framework*, http://www.w3.org/2005/Incubator/model-based-ui/wiki/Cameleon_reference_framework, preuzeto: 28.08.2014.

[Card83] S. K. Card, T. P. Moran, A. Newell, The *Psychology of Human-Computer Interaction*, CRC Press, 1983.

[Ceri00] S. Ceri, P. Fraternali, A. Bongio, *Web Modeling Language (WebML): A Modeling Language for Designing Web Sites*, The international journal of computer and telecommunications networking, vol. 33, issue 1-6, 2000.

[Chen76] P. Chen, *The entity-relationship model: Towards a unified view of data*, ACM Transactions on Database Systems 1, 1976.

[Cirovic11] I. Ćirović, *Komparativna analiza CASE alata za generisanje aplikacija na osnovu modela*, Master rad, Fakultet organizacionih nauka, Univerzitet u Beogradu, 2011.

[Cockburn00] A. Cockburn, *Writing Effective Use Cases*, Addison Wesley Longman Publishing Co. Inc, Boston, 2000.

[Cohn04] M. Cohn, *User Stories Applied: For Agile Software Development*, Addison Wesley Longman Publishing Co. Inc, Boston, 2004.

[Constantine00] L. L. Constantine, H. Windl, J. Noble, L.A.D. Lockwood, *From abstraction torealization in user interface designs: Abstract prototypes based on canonical abstract components*, Working Paper, http://paginas.fe.up.pt/ipc/suporte/praticas/Constantine2003FromAbstractionToRealizationCanonicalAbstractComponents.pdf, 2000.

[Constantine02] L. L. Constantine, L. A. D. Lockwood, *Usage-Centered Engineering for Web Applications*, IEEE Software, vol. 19, issue 2, 2002.

[Constantine03] L. Constantine, L. Lockwood, *Structure and Style in Use Cases for User Interface Design,* 2nd International Conference on Usage-Centered Design, New Hampshire, 2003.

[Constantine99] L. L. Constantine, L. A. D. Lockwood, Software for use: a practical guide to the models and methods of usage-centered design, ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1999.

[Creately] Creately, http://creately.com/, preuzeto: 15.08.2015.

[Cunningham04] C. Cunningham, C. A. Galindo-Legaria, G. Graefe, *PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS,* Proceedings of the Thirtieth international conference on Very large data bases – Volume 30, 2004.

[DaCruz10] A. M. Rosado da Cruz, *Automatic Generation of User Interfaces from Rigorous Domain and Use Case Models*, PhD. Thesis, University of Porto, Portugal, 2010.

[DaSilva00/1] P. P. Da Silva, *User Interface Declarative Models and Development Environments: A Survay,* Proceedings of the 7th international conference on Design, specification, and verification of interactive systems, DSV-IS'00, 2000.

[DaSilva00] P. P. Da Silva, N. W. Paton, *User Interface Modeling With UML*, Proceedings of the 10th European-Japanese Conference on Information Modeling and Knowledge Representation, 2000.

[DaSilva15] A. R. da Silva, D. Savic, S. Vlajic, I. Antovic, S. Lazarevic, V. Stanojevic, M. Milic, *Patterns for Better Use Cases Specification*, 20th European Conference on Pattern Languages of Programs – EuroPLoP, 2015.

[DDDNakedObjects] *Interview and Book Excerpt: Dan Haywood's Domain-Driven Design Using Naked Objects*, http://www.infoq.com/articles/haywood-ddd-no, preuzeto 30.03.2013.

[DENIM] *DENIM: An Informal Tool For Early Stage Web Site and UI Design*, http://dub.washington.edu:2007/denim/, preuzeto: 15.08.2015.

[EclipseFoundation] Eclipse Foundation, https://eclipse.org/org/foundation/, preuzeto: 10.02.2014.

[EclipseGMP] *Graphical Modeling Project (GMP)*, http://www.eclipse.org/modeling/gmp/, preuzeto: 10.02.2014.

[EclipseIDE] *Eclipse IDE*, www.eclipse.org, preuzeto: 15.08.2015.

[EclipseMF] *Eclipse Modeling Framework (EMF),* http://eclipse.org/modeling/emf/, preuzeto: 10.02.2014.

[Elicitation] *10 requirement elicitation techniques*, http://www.slideshare.net/PeteFrey1/10-requirement-elicitation-techniques-for-b-as, preuzeto 15.12.2014.

[Excel] *Microsoft Office Excel*, https://products.office.com/en-us/excel, preuzeto 15.04.2015.

[Fitzgerald05] J. Fitzgerald, P. Larsen, P. Mukherjee, N. Plat, M. Verhoef, *Validated Designs for Object-oriented Systems*, Springer-Verlag, London, 2005.

[Fowler03] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.

[Fowler97] M. Fowler, *Analysis Patterns, Reusable Objects Models*, Addison Wesley Longman Publishing Co. Inc, Indianapolis, 1997.

[Freeman12] A. Freeman, *Pro ASP. NET MVC 4*, Apress, 2012.

[Frost07] A. Frost, M. Campo, *Advancing Defect Containment to Quantitative Defect Management*, The Journal of Defense Software Engineering, Dec. 2007. Issue, 2007.

[Fussell97] M. L. Fussell, *Foundations of Object-Relational Mapping*, ChiMu Corp, 1997.

[Gajos10] K. Z. Gajos, D. S. Weld, J. O. Wobbrock, *Automatically generating personalized user interfaces with Supple*, Artificial Intelligence, 2010.

[GOF] E. Gamma, R. Helm, R. Jonson, J. Vilissides, *Design patterns*, Addison Wesley, 18th Printing, 1999.

[GoogleForms] *Google upitnici*, https://www.google.com/forms/about/, preuzeto 1.02.2013.

[GoogleSheets] *Google Sheets*, https://docs.google.com/spreadsheets, preuzeto 15.04.2015.

[Harmelen01] V.M. Harmelen, *Object modeling and user interface design: designing interactive systems.* Addison Wesley Longman Publishing Co. Inc, Boston, 2001.

[Hay95] D. C. Hay, *Data Model Patterns – Conventions of Thought*, Dorset House Publishing, 1995.

[Hay99] D. C. Hay, *A comparison of data modeling techniques*, Essential Strategies, Inc, 1999.

[Haywood09] D. Haywood, *Domain driven design using Naked Objects*, The Pragmatic Bookshelf, North Carolina, Texas, 2009.

[HTML] *HTML, The Web's Core Language,* http://www.w3.org/html/, preuzeto: 15.08.2015.

[IEEE98] *830-1998 – IEEE Recommended Practice for Software Requirements Specifications*,

http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=720574&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D720574, preuzeto: 5.04.2012.

[IETSoftware] *Institution of Engineering and Technology*, http://www.theiet.org/resources/journals/, preuzeto 10.04.2012.

[Jacobson00] G. Booch, J. Rumbaugh, I. Jacobson, *UML Vodič za korisnike,* Addison – Wesley, CET Computer Equipment and Trade, Beograd, 2000.

[Jacobson87] I. Jacobson, *Object-oriented development in an industrial environment*, ACM SIGPLAN Notices 22(12), 1987.

[Jacobson93] I. Jacobson, M. Christerson, *Object-Oriented Software Engineering*, Addison Wesley Longman Publishing Co. Inc, Boston, 1993.

[Jendrock08] E. Jendrock, J. Ball, D. Carson, I. Evans, S. Fordin, K. Haase, *The Java EE 5 Tutorial,* Sun Microsystems, Santa Clara, CA95054, U.S.A. 2008.

[Jigloo] *Jigloo SWT/Swing GUI Builder for Eclipse and WebSphere*, http://www.cloudgarden.org/jigloo/, preuzeto: 15.08.2015.

[JMatter] *JMatter*, http://jmatter.org/, preuzeto: 10.04.2012.

[Johnson00] J. Johnson, *Turning Chaos into Success*, Software Magazine, Vol. 19, No. 3, 2000.

[Johnston] R. Johnston, C. Moran, *The «include» and «extend» Relationships in Use Case Models*, Working paper (http://www.karonaconsulting.com/downloads/UseCases_IncludesAndExtends.pdf), Karona Consulting Ltd, preuzeto: 10.02.2015.

[JSF] *JavaServer Faces (JSF)*, http://www.javaserverfaces.org/, preuzeto 05.02.2013.

[Karic14] M. Karić, *Razvoj generatora korisničkog interfejsa web aplikacija zasnovanog na sopstvenom SilabUI modelu*, Master rad, Fakultet organizacionih nauka, Univerzitet u Beogradu, 2014.

[Kennard10] R. Kennard, J. Leaney: *Towards a general purpose architecture for UI generation*, Journal of Systems and Software, vol. 83, issue 10, Elsevier, 2010.

[Kennard11] R. Kennard, Derivation of a *General Purpose Architecture for Automatic User Interface Generation*, PhD. Thesis, University of Technology, Sydney, Australia, 2011.

[KennardConsulting] *Kennard Consulting*, http://www.kennardconsulting.com/, preuzeto 30.03.2013.

[Kivisto00] K. Kivistö, *A third generation object-oriented process model, Roles and architectures in focus*, Academic Dissertation, Faculty of Science, University of Oulu, Oulu, 2000.

[Larman98] C. Larman, *Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Second edition, Prentice Hall, 1998.

[Lazarević03] B. Lazarević, Z. Marjanović, N. Aničić, S. Babarogić, *Baze Podataka,* Fakultet organizacionih nauka, Beograd, 2003.

[Ludewig03] J. Ludewig, *Models in software engineering – an introduction*, Software and Systems Modeling, Springer-Verlag, 2003.

[Malan01] R. Malan, D. Bredemeyer, *Functional Requirements and Use Cases,* Bredemeyer Consulting, Bloomington, 2001.

[Marinilli06] M. Marinilli, *Professional Java User Interfaces*, Wiley, 2006.

[MBUIUseCases] *MBUI Use Cases*, http://www.w3.org/wiki/MBUI_Use_Cases, preuzeto: 15.08.2015.

[Meixner11] G. Meixner, F. Paternó, J. Vanderdonckt, *Past, Present, and Future of Model-Based User Interface Development,* i-com, vol. 10, issue 3, De Gruzter, 2011.

[Metawidget] *Metawidget*, http://metawidget.org/, preuzeto 30.03.2013.

[Mladenovic13] Đ. Mladenović, *Razvoj sopstvene Google Web Toolkit komponente za kreiranje slozenog korisnickog interfejsa*, Master rad, Fakultet organizacionih nauka, Univerzitet u Beogradu, 2013.

[Molina02] P. J. Molina, S. Meliá, O. Pastor, *JUST-UI: A User Interface Specification Model*, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, Computer-Aided Design of User Interfaces III, Valenciennes, France, 2002.

[MSVisio] Microsoft Visio, http://products.office.com/en-US/visio/flowchart-software, preuzeto: 15.08.2015.

[Myers] B. Myers, M. Rosson, *Survey on user interface programming,* ACM: Human Factors in Computing Systems, Proceedings SIGHI, 1992

[Myers95] B. Myers, *User Interface Software Tools*, ACM Transactions on Computer-Human Interaction, vol. 2, issue 1, 1995.

[Myers98] B. A. Myers, *A Brief History of Human Computer Interaction Technology*, ACM interactions, vol 5, issue 2, 1998.

[NakedObjects] *Naked Objects*, http://www.nakedobjects.org/, preuzeto: 25.02.2012.

[Neskovic00] S. Nešković, B. Lazarević, *A Methodology for Business Process Modelling*, Yugoslav Journal of Operations Research, vol. 10, issue 1, 2000.

[NetBeansIDE] *NetBeans IDE*, www.netbeans.org, preuzeto: 15.08.2015.

[Nielsen12] J. Nielsen, *Usability 101: Introduction to Usability*, Nielsen Norman Group, 2012.

[Oakland89] J. S. Oakland, *Total Quality Management*, Heinemann Professional, 1989.

[Ochodek08] M. Ochodek, J. Nawrocki, *Automatic Transactions Identification in Use Cases*, Balancing Agility and Formalism in Software Engineering: 2nd IFIP Central and East European Conference on Software Engineering Techniques CEE-SET 2007, vol. 5082, Springer Verlag. LNCS. 2008.

[OCL] *Object Constraint Language (OCL)*, http://www.omg.org/spec/OCL/, preuzeto: 12.08.2013.

[OpenXava] *OpenXava*, http://www.openxava.org/, preuzeto: 15.08.2015.

[Peak06] P. Peak, N. Heudecker, *Hibernate quickly,* Manning Publications Co, Greenwich, 2006**.**

[Pfleeger06] S. L. Pfleeger, J. M. Atlee, *Software Engineering Theory and Practice*, Third edition, Prentice Hall, 2006.

[Phanouriou00] C. Phanouriou*, A Device-Independent User Interface Markup Language*, faculty of the Virginia Polytechnic Institute and State University, PhD theses, 2000.

[PrimeFaces] *PrimeFaces – Ultimate JSF Framework*, http://www.primefaces.org/, preuzeto 05.02.2013.

[Quarani98] T. Quatrani, *Visual Modeling with Rational Rose and UML*, Addison-Wesley, 1998.

[Raneburger12] D. Raneburger, R.Popp, J. Vanderdonckt, *An automated layout approach for model-driven WIMP-UI generation*, Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12), ACM, New York, 2012.

[Readability] *Readability: the Optimal Line Length*, http://baymard.com/blog/line-length-readability, preuzeto 08.04.2015.

[RestfulObjects] *The Restful Objects*, http://restfulobjects.org/, preuzeto 30.03.2013.

[Rich09] C. Rich, *Building task based user interfaces with ANSI/CEA 2018*, IEEE Computer, vol. 42, issue 8, 2009.

[Rosemberg07] D. Rosemberg, M. Stewphens, *Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, Berkeley, 2007.

[Savic11] D. Savić, I. Antović, S. Vlajić, V. Stanojević, M. Milić, *Language for Use Case Specification*, Proceedings of the 2011 IEEE 34th Software Engineering Workshop, 2011.

[Savic12] D. Savic, A. R. Da Silva, S. Vlajic, S. Lazarevic, V. Stanojevic, I. Antovic, M. Milic, *Use Case Specification at Different Levels of Abstraction*, QUATIC, IEEE Computer Society, 2012.

[Schlungbaum96] E. Schlungbaum, *Model-based User Interface Software tools – Current state of declarative models*, Graphics, Visualization and Usability Center, Georgia Institute of Technology, GVU Tech Report, Atlanta, 1996.

[Schmidt06] D.C. Shmidt, *Model-Driven Engineering*, IEEE Computer, vol. 39, issue 2, 2006.

[Scimpy/1] *Scimpi*, http://sourceforge.net/projects/scimpi/, preuzeto 30.03.2013.

[Scimpy/2] *The Scimpy Framework*, http://blog.nakedobjects.org/2008/05/08/the-scimpi-framework/, preuzeto 30.03.2013.

[Scott09] B. Scott, T. Neil, *Designing Web Interfaces: Principles and Patterns for Rich Interactions*, O'Reilly Media, 2009.

[Selic03] B. Selic, *The Pragmatics of Model-Driven Development*, IEEE Software, vol. 20, issue 5, 2003.

[SILAB] *Laboratorija za softversko inženjerstvo*, http://silab.fon.rs/, preuzeto 10.05.2015.

[SilkDesigner] *Silk Designer Authoring Tool,* http://mentiscorp.com/silk_designer.aspx, preuzeto: 15.08.2015.

[Sinnig05] D. Sinnig, P. Chalin, F. Rioux, *Use Cases in Practice: A Survey*, Proceedings of CUSEC 05, Ottawa, Canada, 2005.

[Sommerville06] I. Sommerville, *Software Engineering*, Eight edition, Addison Wesley Longman Publishing Co. Inc, Boston, 2006.

[SPE] *Software: Practice and Experience*, http://eu.wiley.com/WileyCDA/WileyTitle/productCd-SPE.html, preuzeto 01.10.2011.

[SPSS] *SPSS Statistics*, http://www-01.ibm.com/software/analytics/spss/products/statistics/, preuzeto 15.11.2014.

[SQLServer] *SQL Server*, http://www.microsoft.com/en-us/server-cloud/products/sql-server/, preuzeto 10.05.2015.

[Stellman09] A. Stellman, *Requirements 101: User Stories vs. Use Cases,* http://www.stellman-greene.com/2009/05/03/requirements-101-user-stories-vs-use-cases/,

Building Better Software: Jennifer Greene and Andrew Stellman weblog, postavljeno: 3. 05. 2009, preuzeto: 1. 07. 2012.

[Storrs95] G. Storrs, *The Notion of Task in Human-Computer Interaction*, People and Computers X. Proceedings British HCI'95, Cambridge University Press, 1995.

[StringTemplate] *StringTemplate*, http://www.stringtemplate.org/, preuzeto 30.03.2013.

[Szekely96] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, E. Salcher, *Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach*, Engineering for Human-Computer Interaction, Chapman & Hall, 1996.

[TaskModel] *ANSI/CEA-2018 Task Model Description standard*, http://www.w3.org/2005/Incubator/model-based-ui/wiki/ANSI/CEA-2018, preuzeto: 15.08.2015.

[TextUML] *TextUML Toolkit*, http://sourceforge.net/projects/textuml/, preuzeto 30.03.2013.

[Tidwell05] J. Tidwell, *Designing Interfaces: Patterns for Effective Interaction Design*, O'Reilly, 2005.

[TouchDesign] *Touch design guidelines*, https://msdn.microsoft.com/en-us/library/windows/apps/hh465370.aspx, preuzeto 22.10.2014.

[UIDesign] *Who nailed the principles of great UI design? Microsoft, that's who,* http://www.infoworld.com/article/2614315/application-development/who-nailed-the-principles-of-great-ui-design--microsoft--that-s-who.html, preuzeto 18.12.2013.

[UIPatterns] *User Interface Design patterns*, http://ui-patterns.com/, preuzeto: 15.08.2015.

[Usability] *Principles for Usable Design*, http://www.usabilitybok.org/principles-for-usable-design, preuzeto 06.02.2014.

[UseCases] *When Use Cases Aren't Enough, Part 2*, http://www.modernanalyst.com/Resources/Articles/tabid/115/ID/2925/When-Use-Cases-Arent-Enough-Part-2.aspx, preuzeto 10.11.2014.

[UserStories/1] *Agile User Stories – The Building Blocks for Software Project Development Success*, https://www.scrumalliance.org/community/articles/2013/september/agile-user-stories, preuzeto 10.11.2014.

[UserStories/2] *Old Favourite: Feature Injection User Stories On A Business Value Theme*, http://antonymarcano.com/blog/2011/03/fi_stories/, preuzeto 10.11.2014.

[UserStories] *User Stories*, http://www.mountaingoatsoftware.com/agile/user-stories, preuzeto 10.11.2014.

[UsiXML/1] *UsiXML – Key goals*, http://www.usixml.eu/key-goals, preuzeto: 28.08.2014.

[UsiXML/2] *USer Interface eXtensible Markup Language (UsiXML) – W3C Working Group Submission 1 February 2012,* http://www.w3.org/wiki/images/5/5d/UsiXML_submission_to_W3C.pdf, preuzeto: 28.08.2014.

[UsiXML] *UsiXML – About the project*, http://www.usixml.eu/about-the-project, preuzeto: 28.08.2014.

[VisualStudio] *Visual Studio*, http://www.visualstudio.com/, preuzeto: 15.08.2015.

[Vlajic03] S. Vlajić, *Projektovanje programa (skripta)*, Dr Siniša Vlajić, Beograd, 2003.

[Vlajić08] S. Vlajić, D. Savić, V. Stanojević, I. Antović, M. Milić, *Projektovanje Softvera – Napredne Java tehnologije,* Zlatni Presek, Beograd, 2008.

[Vliet08] H. V. Vliet, Software engineering: principles and practice, John Wiley & Sons Ltd, 3rd edition, Chichester, West Sussex, England, 2008.

[Wagner03] R. Wagner, H. Giese, U. Nickel, *A plug-in for flexible and incremental consistency management*, Proceedings of the Workshop on Consistency Problems in UML-based Software Development II, UML 2003 Workshop 7, Blekinge Institute of Technology, 2003.

[WebOrDesktop] *Designing for Web or Desktop?*, https://msdn.microsoft.com/en-us/library/ms973831.aspx, preuzeto 15.04.2015.

[WebRatio] *WebRatio*, http://www.webratio.com/, preuzeto 30.03.2013.

[Wicket] *Apache Wicket*, http://wicket.apache.org/, preuzeto 30.03.2013.

[Wiegers06] K. E. Wiegers, *More About Software Requirements: Thorny Issues and Practical Advice*, Microsoft Press, 2006.

[Wirfs-Brock93] R. J. Wirfs-Brock, *Designing Scenarios: Making the Case for a Use Case Framework,* The Smalltalk Report, Vol.3, No. 3, 1993.

[WPF] *Windows Presentation Foundation*, https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx, preuzeto: 15.08.2015.

[Zukowski06] J. Zukowski, *Java 6 Platform Revealed*, Apress, 2006, Berkeley, CA, USA, 2006.

11. Списак слика

Слика 1: Однос између алата, приступа и карактеристика различитих приступа за генерисање корисничког интерфејса ____ *7*

Слика 2: Однос између генератора програма, случајева коришћења и корисничког интерфејса ____ *10*

Слика 3: Упрошћен приказ трослојне архитектуре ____ *20*

Слика 4: Однос између различитих фаза у процесу развоја софтверског система, кроз трансформацију домена проблема у домен рјешења [Alhir03] ____ *45*

Слика 5: Однос различитих погледа на систем, из различитих перспектива, у контексту MDA и дефинисаних врста модела (CIM, PIM, PSM и ISM) [Alhir03] ____ *47*

Слика 6: Веза између случајева коришћења и осталих дијелова спецификације корисничких захтјева [Cockburn00] ____ *63*

Слика 7: Различита тумачења од стране различитих учесника у пројекту [Oakland89] ____ *71*

Слика 8: Примјер класе ______ *76*

Слика 9: Примјер насљеђивања ______ *77*

Слика 10: Примјер агрегације ______ *77*

Слика 11: Примјер енкапсулације (учаурења) ______ *78*

Слика 12: Примјер релације ______ *80*

Слика 13: Примјер специјализације ______ *80*

Слика 14: Примјер агрегације ______ *81*

Слика 15: Објекти са референцама на исту меморијску локацију ______ *83*

Слика 16: Објекти са референцама на различите меморијске локације које садрже исте вриједности ______ *84*

Слика 17: Асоцијације ______ *85*

Слика 18: Графички приказ елемената различитих нотација за креирање модела података [Ambler04] ______ *88*

Слика 19: Шаблон за случај коришћења у потпуно сређеном облику ______ *95*

Слика 20: Шаблон за случај коришћења у табеларном облику са једном колоном ___ *97*

Слика 21: Шаблон за случај коришћења у табеларном облику са двије колоне ___ *100*

Слика 22: Шаблон за случај коришћења препоручен од стране RUP (Rational Unified Process) методе ___ *101*

Слика 23: Дио основног сценарија случаја коришћења „Наручивање производа“ ___ *102*

Слика 24: Процес развоја коришћењем прототипова који се бацају ___ *109*

Слика 25: Процес развоја коришћењем еволутивних прототипова ___ *109*

Слика 26: Број запослених у компанији у којој раде ___ *115*

Слика 27: Област којој припада компанија у којој раде ___ *116*

Слика 28: Питање – Учесталост развоја различитих врста апликација ___ *117*

Слика 29: Учесталост развоја различитих врста апликација ___ *118*

Слика 30: Питање – Учесталост коришћења различитих платформи ___ *119*

Слика 31: Учесталост коришћења различитих платформи ___ *120*

Слика 32: Остале коришћене платформе ___ *121*

Слика 33: Питање – Број захтјева који се углавном могу имплементирати коришћењем CRUD операција 122

Слика 34: Број захтјева који се углавном могу имплементирати коришћењем CRUD операција 122

Слика 35: Питање – Први корак након прикупљања захтјева од корисника 123

Слика 36: Први корак након прикупљања захтјева од корисника 123

Слика 37: Питање – Учесталост коришћења различитих начина и формата спецификације захтјева 124

Слика 38: Учесталост коришћења различитих начина и формата спецификације захтјева 124

Слика 39: Учесталост коришћења случајева коришћења и корисничких прича приликом спецификације захтјева 125

Слика 40: Питање – Када и колико често се користе случајеви коришћења 126

Слика 41: Када и колико често се користе случајеви коришћења 126

Слика 42: Питање – Колико често се случајеви коришћења користе за откривање различитих елемената система 127

Слика 43: Колико често се случајеви коришћења користе за откривање различитих елемената система ________ *127*

Слика 44: Питање – Колико тијесно случајеви коришћења треба да буду повезани са различитим елементима система ________ *128*

Слика 45: Колико тијесно случајеви коришћења треба да буду повезани са различитим елементима система ________ *128*

Слика 46: Питање – Када се креира доменски модел ________ *129*

Слика 47: Када се креира доменски модел ________ *129*

Слика 48: Питање – Колико често се креирају различите врсте прототипова корисничког интерфејса ________ *130*

Слика 49: Колико често се креирају различите врсте прототипова корисничког интерфејса ________ *130*

Слика 50: Питање – Учесталост различитих начина креирања корисничког интерфејса ________ *131*

Слика 51: Учесталост различитих начина креирања корисничког интерфејса ___ *131*

Слика 52: Питање – Колико времена и напора, на основу искуства испитаника, је потребно у софтверском пројекту издвојити на развој корисничког интерфејса, пословне логике и нивоа перзистенције ____ *132*

Слика 53: Колико времена и напора, на основу искуства испитаника, је потребно у софтверском пројекту издвојити на развој корисничког интерфејса, пословне логике и нивоа перзистенције ____ *132*

Слика 54: Питање – На чему треба да буде заснована улазна спецификација коју користе алати за аутоматизацију развоја корисничког интерфејса ____ *133*

Слика 55: На чему треба да буде заснована улазна спецификација коју користе алати за аутоматизацију развоја корисничког интерфејса ____ *133*

Слика 56: Питање – Чиме основна улазна спецификација треба да буде проширена ____ *134*

Слика 57: Чиме основна улазна спецификација треба да буде проширена ____ *134*

Слика 58: Питање – Пожељан начину креирања улазне спецификације ____ *135*

Слика 59: Пожељан начину креирања улазне спецификације ____ *135*

Слика 60: Питање – Да ли би било корисно да постоји могућност поновног коришћења других дијелова система ____ *136*

Слика 61: Да ли би било корисно да постоји могућност поновног коришћења других дијелова система ____ 136

Слика 62: Питање – У којој форми треба генерисати кориснички интерфејс ____ 137

Слика 63: У којој форми треба генерисати кориснички интерфејс ____ 137

Слика 64: Питање – Значај различитих могућности које треба да обезбиједи алат за генерисање корисничког интерфејса ____ 138

Слика 65: Значај различитих могућности које треба да обезбиједи алат за генерисање корисничког интерфејса ____ 138

Слика 66: Петослојна архитектура Metawidget оквира [Kennard11] ____ 151

Слика 67: Процес израде апликације у WebRatio алату ____ 156

Слика 68: Процес израде апликације у AlphaSimple алату ____ 160

Слика 69: Процес израде апликације у BizAgi алату ____ 163

Слика 70: Дедуктивни кориснички интерфејс ____ 177

Слика 71: Индуктивни кориснички интерфејс ____ 178

Слика 72: Хоризонтални Master/Detail патерн ____ 180

Слика 73: Вертикални Master/Detail патерн _____ *180*

Слика 74: Примјер коришћења Master/Detail патерна у десктоп апликацији _____ *180*

Слика 75: Хоризонтални Column Browse патерн _____ *181*

Слика 76: Вертикални Column Browse патерн _____ *181*

Слика 77: Примјер коришћења Column Browse патерна у десктоп апликацији _____ *181*

Слика 78: Search/Results патерн са простим критеријумом _____ *182*

Слика 79: Search/Results патерн са сложеним критеријумима _____ *182*

Слика 80: Примјер коришћења Search/Results патерна у веб апликацији _____ *182*

Слика 81: Примјер коришћења Search/Results патерна у десктоп апликацији _____ *183*

Слика 82: Хоризонтални Filter Dataset патерн _____ *184*

Слика 83: Вертикални Filter Dataset патерн _____ *184*

Слика 84: Примјер коришћења Filter Dataset патерна у веб апликацији _____ *184*

Слика 85: Form патерн _____ *185*

Слика 86: Примјер коришћења Form патерна у веб апликацији 185

Слика 87: Примјер коришћења Form патерна у десктоп апликацији 185

Слика 88: Palette/Canvas патерн 186

Слика 89: Примјер коришћења Palette/Canvas патерна за Eclipse Modeling Framework 186

Слика 90: Dashboard патерн 187

Слика 91: Spreadsheet патерн 188

Слика 92: Примјер коришћења Spreadsheet патерна у десктоп апликацији 188

Слика 93: Wizard патерн 189

Слика 94: Примјер коришћења Wizard патерна у десктоп апликацији 189

Слика 95: Question/Answer патерн 190

Слика 96: Parallel Panels патерн 191

Слика 97: Примјер коришћења Parallel Panels патерна у веб апликацији 191

Слика 98: Примјер коришћења Parallel Panels патерна у мобилној апликацији 191

Слика 99: Interactive Model патерн 192

Слика 100: Примјер коришћења Interactive Model патерна у десктоп апликацији 192

Слика 101: Примјер коришћења Interactive Model патерна у веб апликацији 193

Слика 102: Примјер коришћења Interactive Model патерна у мобилног апликацији 193

Слика 103: Blank State патерн 194

Слика 104: Примјер коришћења Blank State патерна у мобилној апликацији 194

Слика 105: Веза између елемената спецификације случајева коришћења и корисничког интерфејса 210

Слика 106: Примјер везе 1- између ентитета рачун и ставка рачуна* 213

Слика 107: Графички приказ везе parent-child између ентитета 213

*Слика 108: Примјер везе *-1 између ентитета рачун и купац* 215

Слика 109: Однос између различитих елемената случаја коришћења, модела података, шаблона и графичких компоненти које шаблони користе 217

Слика 110: Дио модела података за примјер комбиновања различитих шаблона корисничког интерфејса 219

Слика 111: Примјер комбиновања различитих шаблона корисничког интерфејса са 4 нивоа дубине ___ *220*

Слика 112: Field-form шаблон: један ентитет без дјеце (Java Swing технологија) ___ *222*

Слика 113: Field-form шаблон: један ентитет без дјеце (Java Server Faces – Prime Faces) ___ *223*

Слика 114: Реализација операције Insert у Field-form шаблону ___ *223*

Слика 115: Веза између ентитета која је један од предуслова за коришћење падајуће листе ___ *224*

Слика 116: Позив функционалности Show details на корисничком интерфејсу десктоп апликације ___ *224*

Слика 117: Реализација функционалности Show details над падајућом листом на корисничком интерфејсу десктоп апликације ___ *225*

Слика 118: Позив функционалности Show details на корисничком интерфејсу веб апликације ___ *225*

Слика 119: Реализација функционалности Show details над падајућом листом на корисничком интерфејсу веб апликације ___ *225*

Слика 120: Дио модела података за приказ field-form шаблона који укључује информације о дјеци ____ 226

Слика 121: Кориснички интерфејс реализован коришћењем Field-Form шаблона на корисничком интерфејсу десктоп апликација са приказом информација везаних за дјецу ____ 227

Слика 122: Кориснички интерфејс реализован коришћењем Field-Form шаблона на корисничком интерфејсу веб апликација са приказом информација везаних за дјецу227

Слика 123: Дио модела података за приказ field-tab шаблона____ 229

Слика 124: Примјер корисничког интерфејса који користи Field-tab шаблон у десктоп апликацији____ 229

Слика 125: Примјер корисничког интерфејса који користи Field-tab шаблон у веб апликацији____ 230

Слика 126: Дио модела података за приказ Table-form шаблона у десктоп апликацији ____ 231

Слика 127: Примјер корисничког интерфејса развијеног коришћењем table-form шаблона у десктоп апликацији ____ 232

Слика 128: Примјер коришћења cell editor-а приликом уноса и измјене података у табели за кориснички интерфејс десктоп апликације____ 233

Слика 129: Дио модела података за приказ Table-form шаблона у веб апликацији 234

Слика 130: Примјер корисничког интерфејса развијеног коришћењем table-form шаблона у веб апликацији 234

Слика 131: Одабир реда табеле у Table-form шаблону 234

Слика 132: Измјена одабраног појављивања ентитета у Table-form шаблону за веб апликације 235

Слика 133: Дио модела података за приказ table-tab шаблона за кориснички интерфејс десктоп апликације 237

Слика 134: Примјер корисничког интерфејса развијеног коришћењем table-tab шаблона у десктоп апликацији 237

Слика 135: Различити начини приказивања опција за креирање, ажурирање и брисање података 238

Слика 136: Примјер корисничког интерфејса развијеног коришћењем Table-tab шаблона у веб апликацији, са приказом дјетета коришћењем Field-form шаблона 239

Слика 137: Примјер корисничког интерфејса развијеног коришћењем Table-tab шаблона у веб апликацији, са приказом дјетета коришћењем Table-form шаблона 239

Слика 138: Дио модела података за приказ KNGUI шаблона 241

Слика 139: Примјер корисничког интерфејса развијеног коришћењем KNGUI шаблона у десктоп апликацији ______ *243*

Слика 140: Илустрација коришћења операције којом се сакривају подаци о дјеци на KNGUI шаблону ______ *245*

Слика 141: Општи облик коришћења PIVOT наредбе ______ *245*

Слика 142: Примјер корисничког интерфејса развијеног коришћењем KNGUI шаблона у веб апликацији ______ *246*

Слика 143: Обрада генерисаног корисничког интерфејса коришћењем Jigloo [Jigloo] алата за визуелно креирање корисничког интерфејса који се интегрише са Eclipse развојним окружењем ______ *248*

Слика 144: SilabUI мета-модел ______ *256*

Слика 145: Садржај датотеке useCaseModel.xsd ______ *264*

Слика 146: Садржај датотеке useCaseGuiExtension.xsd ______ *266*

Слика 147: УМЛ дијаграм случајева коришћења за примјер конкретне улазне спецификације ______ *267*

Слика 148: Дио концептуалног дијаграма за примјер конкретне улазне спецификације ______ *267*

Слика 149: Примјер улазне спецификације (useCase_ObradaRcuna.xml) за случај коришћења Обрада рачуна ______ *269*

Слика 150: Примјер проширења улазне спецификације (useCaseGuiExtension_ObradaRcuna.xml) за случај коришћења Обрада рачуна ______ *270*

Слика 151: Примјер проширења улазне спецификације (useCaseGuiExtension_ObradaRcuna.xml) за случај коришћења Обрада рачуна са спецификацијом свих могућих елемената дефинисаних шемом ______ *273*

Слика 152: SilabUI Wizard – обрада одабраног пројекта ______ *289*

Слика 153: SilabUI Wizard – обрада одабраног случаја коришћења ______ *290*

Слика 154: SilabUI Wizard – обрада Актора ______ *290*

Слика 155: SilabUI Wizard – обрада ентитета повезаног са случајем коришћења *291*

Слика 156: SilabUI Wizard – обрада корака основног сценарија случаја коришћења *292*

Слика 157: SilabUI Wizard – обрада атрибута који ће бити приказивани кориснику за сваку инстанцу ентитета у листи из које се врши одабир ______ *293*

Слика 158: SilabUI Wizard – повезивање текућег случаја коришћења са другим случајем коришћења релацијом Include ______ *293*

Слика 159: SilabUI Wizard – Обрада случаја коришћења који је повезан релацијом Include 294

Слика 160: SilabUI Wizard – коришћење постојеће шеме базе података за формирање улазне спецификације 295

Слика 161: SilabUI Wizard – прилагођени мета-модел за коришћење у графичком алату за креирање улазне спецификације 296

Слика 162: Алат за визуелно моделовање улазне спецификације 298

Слика 163: Мета-модел прилагођен за коришћење у алату за визуелно моделовање 299

Слика 164: Формирање улазне спецификације 300

Слика 165: Обрада улазне спецификације и генерисање програмског кода 301

Слика 166: Структура модела за генерисање корисничког интерфејса десктоп апликације 303

Слика 167: Структура модела за генерисање корисничког интерфејса веб апликације 303

Слика 168: Структура генератора програмског кода корисничког интерфејса десктоп апликација 304

Слика 169: Структура генератора програмског кода корисничког интерфејса веб апликација ___ *305*

Слика 170: Strategy патерн ___ *307*

Слика 171: Template Method патерн ___ *308*

Слика 172: Bridge патерн ___ *309*

Слика 173: Decorator патерн ___ *310*

Слика 174: Factory патерн ___ *311*

Слика 175: Примјер текстуалног шаблона за компоненту дугме ___ *312*

Слика 176: Примјер конкретизованог текстуалног шаблона за компоненту дугме *313*

Слика 177: UML дијаграм секвенци који описује генерисање програмског кода веб апликације ___ *314*

Слика 178: Facade патерн ___ *319*

Слика 179: Класа DataTransferObject ___ *321*

Слика 180: Валидација – приказивање грешке приликом уноса у текстуално поље у десктоп апликацији ___ *323*

Слика 181: Валидација – приказивање грешке приликом уноса у ћелију табеле у десктоп апликацији ____ *324*

Слика 182: Валидација – приказивање грешке приликом позива системске операције у десктоп апликацији ____ *324*

Слика 183: Валидација – приказивање грешке приликом уноса вриједности за датум у веб апликацији ____ *324*

Слика 184: Валидација – приказивање грешака приликом уноса у ћелију табеле у веб апликацији ____ *324*

Слика 185: Валидација – приказивање грешке приликом уноса неодговарајућег типа податка у веб апликацији ____ *325*

Слика 186: Валидација – приказивање грешке у случају непопуњавања обавезног поља у веб апликацији ____ *325*

Слика 187: Приказ корисничког интерфејса прилагођен енглеском језику ____ *326*

Слика 188: Приказ поруке која се приказује кориснику на енглеском језику ____ *326*

Слика 189: Приказ корисничког интерфејса прилагођен српском језику ____ *327*

Слика 190: Приказ поруке која се приказује кориснику на српском језику ____ *327*

Слика 191: Процес развоја корисничког интерфејса SilabUI приступом ________ *328*

Слика 192: Формирање улазне спецификације за случај коришћења Invoice коришћењем графичког алата ________ *330*

Слика 193: Генерисање улазне спецификације у текстуалном облику ________ *331*

Слика 194: Основа улазне спецификације у текстуалном (XML) облику генерисана коришћењем графичког алата ________ *332*

Слика 195: Проширење улазне спецификације у текстуалном (XML) облику генерисано коришћењем графичког алата ________ *335*

Слика 196: Приказ генерисаног корисничког интерфејса за обраду података о купцима прије измјена ________ *337*

Слика 197: Обрада генерисаног корисничког интерфејса алатом за визуелно креирање корисничког интерфејса ________ *337*

Слика 198: Приказ генерисаног корисничког интерфејса за обраду података о купцима послије измјена ________ *338*

Слика 199: Резултујући кориснички интерфејс – InvoicePanel ________ *338*

Слика 200: Резултујући кориснички интерфејс – InvoiceInsertPanel ________ *339*

Слика 201: Резултујући кориснички интерфејс – PopUpCustomerPanel ____________ *339*

Слика 202: Резултујући кориснички интерфејс – CustomerInsertPanel ____________ *339*

Слика 203: Резултујући кориснички интерфејс – PopUpProductPanel ____________ *340*

Слика 204: Резултујући кориснички интерфејс – ProductInsertPanel ____________ *340*

Слика 205: Резултујући кориснички интерфејс – Invoice ________________________ *341*

Слика 206: Резултујући кориснички интерфејс – Invoice insert dialog ____________ *342*

Слика 207: Резултујући кориснички интерфејс – InvoiceItem insert dialog _________ *342*

Слика 208: Резултујући кориснички интерфејс – Customer dialog _______________ *343*

Слика 209: Резултујући кориснички интерфејс – Customer insert dialog ___________ *343*

Слика 210: Резултујући кориснички интерфејс – Product dialog _________________ *344*

Слика 211: Резултујући кориснички интерфејс – Product insert dialog ____________ *344*

12. Списак табела

Табела 1: Вријеме потребно за исправљање грешака по фазама (у сатима) 68

Табела 2: Листа категорија за идентификкацију концептуалних класа 90

Табела 3: Типови акција које одговарају одређеним корацима у сценарију 104

Табела 4: Карактеристике посматраних алата на основу дефинисаних критеријума за поређење 171

Табела 5: Графичке компоненте које одговарају различитим типовима акција сценарија 208

Табела 6: Садржај табеле Студент 242

Табела 7: Садржај табеле Предмет 242

Табела 8: Садржај табеле Положио 243

Табела 9: Садржај табеле Користи 243

Биографија

Илија Антовић је рођен 8. 7. 1980. године у Котору, република Црна Гора.

Основну школу завршио је у Котору као и нижу музичку школу – инструмент клавир и гимназију – општи смјер.

Године 1999. уписује Факултет организационих наука Универзитета у Београду на коме је дипломирао 2004. године. Након тога уписује постдипломске студије. Магистрирао је у области софтверског инжењерства 2010. године на Факултету организационих наука, а докторску тезу је пријавио 2012. године на истом Факултету. Од 2013. до 2015. године ангажован је као консултант Републичког фонда за здравствено осигурање. Од 2000. године ангажован је на извођењу лабораторијских вјежби на Факултету организационих наука, а од 2006. године запослен је у Лабораторији за софтверско инжењерство на ФОН-у. Као сарадник Лабораторије учествовао је у извођењу значајних пројеката међу којима су:

- Сајтови здравствених установа, Републички фонд за здравствено осигурање, Београд, 2013.
- Листе чекања, Републички фонд за здравствено осигурање, Београд, 2013.
- Идејни пројекат информационог система е-аукцијске јавне набавке (за потребе Министарства за телекомуникације и информатичко друштво Републике Србије),
- Пројекат KOSTMOD (Forsvarets forskningsinstitutt Ministarstva одбране Краљевине Норвешке и Министарства одбране Републике Србије).

- Модернизација информационог система за здравствене установе, IQ-net, Београд 2006.

Поред тога учествовао је у извођењу пројеката:

- Менаџмент јавних набавки, Републички фонд за здравствено осигурање, Београд, 2014.
- Фото модул, вести и архива, Политика новине и магазини, Београд, 2014.
- RFZO Apoteke, Републички фонд за здравствено осигурање, Београд, 2013.
- Sodiumlight, Streetlight Management System (SMCS), a pilot project for the Qatar Public Works Authorities in Doha, 2010.
- IT strategy support to the modernization of the Ministry of finance and economy of Serbia, Европска Агенција за Реконструкцију, Београд 2006.
- Development of health information system for basic health and pharmaceutical services, republic of Serbia, Euro Health Group, Denmark, 2007.

Истакао се и у научно-истраживачком раду. Коаутор је књиге Пројектовање софтвера: Напредне Јава технологије, а аутор је више научних радова објављених на симпозијумима и у научним часописима.

2014. године учествује у формирању хора састављеног од запослених на Факултету организационих наука ПолиФОН. Био је члан хора Обилић при АКУД Бранко Крсмановић.

Ужива у одрастању свог сина Ива!

Прилог 1.

Изјава о ауторству

Потписани-а Илија Антовић

број индекса 523/12

Изјављујем

да је докторска дисертација под насловом

Аутоматско генерисање корисничког интерфејса апликације засновано на случајевима коришћења

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 23. 09. 2015. год.

Илија Антовић

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора Илија Антовић

Број индекса 523/12

Студијски програм

Наслов рада Аутоматско генерисање корисничког интерфејса апликације засновано на случајевима коришћења

Ментор проф. Др Владан Девеџић, редовни професор Факултета организационих наука

Потписани/а Илија Антовић

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, 23. 09. 2015. год.

Илија Антовић

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић" да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

Аутоматско генерисање корисничког интерфејса апликације засновано на случајевима коришћења

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство
2. Ауторство - некомерцијално
3. Ауторство – некомерцијално – без прераде
4. Ауторство – некомерцијално – делити под истим условима
5. Ауторство – без прераде
6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 23. 09. 2015. год.

Илија Антовић