УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Милорад Филиповић

УНАПРЕЂЕЊЕ СПЕЦИФИКАЦИЈЕ ЗАХТЕВА ПОСЛОВНИХ АПЛИКАЦИЈА УЗ ОСЛОНАЦ НА ИЗВРШИВЕ МОДЕЛЕ

ДОКТОРСКА ДИСЕРТАЦИЈА

Нови Сад, 2023.

УНИВЕРЗИТЕТ У НОВОМ САДУ **ОБРАЗАЦ – 5а**
НАВЕСТИ НАЗИВ ФАКУЛТЕТА ИЛИ ЦЕНТРА

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА[1]

Врста рада:	Докторска дисертација
Име и презиме аутора:	Милорад Филиповић
Ментор (титула, име, презиме, звање, институција)	проф. др Гордана Милосављевић, ред. проф.
Наслов рада:	Унапређење спецификације захтева пословних апликација уз ослонац на извршиве моделе
Језик публикације (писмо):	Српски (латиница)
Физички опис рада:	Унети број: Страница 143 Поглавља 6 Референци 195 Табела 13 Слика 67 Графикона 0 Прилога 0
Научна област:	Електротехничко и рачунарско инжењерство
Ужа научна област (научна дисциплина):	Софтверско инжењерство
Кључне речи / предметна одредница:	Развој софтвера вођен моделима, софтверски прототип, инжењерство захтева
Резиме на језику рада:	Истраживање представљено у оквиру ове дисертације имало је за циљ унапређење процеса спецификације корисничких захтева пословних апликација на бази детаљних, извршивих прототипова који се могу креирати уз минималан утрошак времена и енергије. Ради постизања овог циља је имплементиран алат отвореног кода под називом Кроки (фр. *croquis* – скица) чија је архитектура пројектована тако да обезбеди: (1) колаборативни развој спецификације пословне апликације са корисницима који немају знање пројектовања и програмирања софтверских система, (2) ефикасно покретање прототипа директно из сопственог развојног окружења, дајући могућност кориснику да испроба прототип током моделовања кад год пожели, (3) поновно коришћење информација добијених приликом развоја прототипова у каснијим фазама развоја, како би се смањило непотребно трошење ресурса. Експеримент за проверу да ли развијени алат задовољава постављене циљеве је дизајниран као серија од десет експлоративних студија случаја

[1] Аутор докторске дисертације потписао је и приложио следеће Обрасце:
5б – Изјава о ауторству;
5в – Изјава о истоветности штампане и електронске верзије и о личним подацима;
5г – Изјава о коришћењу.
Ове Изјаве се чувају на факултету у штампаном и електронском облику и не кориче се са тезом.

	чији је циљ спецификација пословних апликација са учесницима који долазе из различитих пословних домена који нису познати пројектантима. Појединачне студије су обављене са по једним учесником у оквиру двочасовних пројектантских сесија, где су улогу пројектаната имали аутор ове дисертације и његов ментор. Квалитативни и квантитативни подаци прикупљени током сесија и после њих, путем упитника, су искоришћени за извођење позитивних закључака о ефикасности предложеног приступа и алата. Дизајн истраживања је базиран на концептима MEM-а (*Method Evaluation Model*) који дефинише критеријум за успех одређене методологије у пракси. Упитници који евалуирају језик за моделовање и Кроки алат су формулисани тако да одговарају изабраним концептима FQUAD (*Framework for qualitative assessment of domain-specific languages*) оквира за евалуацију језика специфичних за домен.
Датум прихватања теме од стране надлежног већа:	15.7.2021.
Датум одбране: (Попуњава одговарајућа служба)	
Чланови комисије: (титула, име, презиме, звање, институција)	Председник: проф. Др Горан Сладић, ред. проф., Факултет техничких наука, Нови Сад Члан: проф. др Игор Дејановић, ред. проф., Факултет техничких наука, Нови Сад Члан: проф. др Горан Савић, ванр. Проф., Факултет техничких наука, Нови Сад Члан: проф. др Владимир Вујовић, ванр. проф., Електротехнички факултет, Источно Сарајево Ментор: проф. др Гордана Милосављевић, ред. проф., Факултет техничких наука, Нови Сад
Напомена:	

UNIVERSITY OF NOVI SAD
FACULTY OR CENTER

KEY WORD DOCUMENTATION[2]

Document type:	Doctoral dissertation
Author:	Milorad Filipović
Supervisor (title, first name, last name, position, institution)	Gordana Milosavljević, PhD, Full Professor,
Thesis title:	Improving the Specification of Business Application Requirements Based on Executable Models
Language of text (script):	Serbian language (latin)
Physical description:	Number of: Pages 143 Chapters 6 References 195 Tables 13 Illustrations 67 Graphs 0 Appendices 0
Scientific field:	Electrical and computer engineering
Scientific subfield (scientific discipline):	Software Engineering
Subject, Key words:	Model Driven Software Development, software prototype, requirements engineering
Abstract in English language:	The research presented in this dissertation aimed to improve the process of specification of user requirements of business applications based on detailed, executable prototypes that can be created with minimal expenditure of time and energy. To achieve this goal, an open-source tool called Kroki (fr. *croquis* - sketch) was implemented, the architecture of which is designed to provide: (1) Collaborative development of business application specifications with users who do not have knowledge of designing and programming software systems, (2) Efficient prototyping directly from Kroki's development environment, enabling the user to try out the prototype during modeling whenever they want, and (3) Reuse of information obtained during the development of prototypes in later stages of development, to reduce unnecessary consumption of time and energy. The experiment to validate whether the developed Kroki tool meets the set goals was designed as a series of ten exploratory case studies to specify business applications with participants from different business domains unknown to the designers. Individual studies were carried out within two-hour

[2] The author of doctoral dissertation has signed the following Statements:
5б – Statement on the authority,
5в – Statement that the printed and e-version of doctoral dissertation are identical and about personal data,
5г – Statement on copyright licenses.
The paper and e-versions of Statements are held at he faculty and are not included into the printed thesis.

	design sessions, where the author of this dissertation and his mentor played the designer role, with a single participant in the user role in each session. Qualitative and quantitative data collected during and after the sessions, through questionnaires, were used to draw positive conclusions about the effectiveness of the proposed approach and tools. The research design is based on the concepts of MEM (Method Evaluation Model), which defines the criteria for the success of a certain methodology in practice. Questionnaires that evaluate the modeling language and the Kroki tool are formulated to correspond to the selected concepts of the FQUAD (Framework for qualitative assessment of domain-specific languages) for evaluating DSLs.
Accepted on Scientific Board on:	15.7.2021.
Defended: (Filled by the faculty service)	
Thesis Defend Board: (title, first name, last name, position, institution)	President: Goran Sladić, PhD, Full Professor, Faculty of Technical Sciences, Novi Sad Member: Igor Dejanović, PhD, Full Professor, Faculty of Technical Sciences, Novi Sad Member: Goran Savić, PhD, Associate Professor, Faculty of Technical Sciences, Novi Sad Member: Vladimir Vujović, PhD, Associate Professor, Faculty of Electrical Engineering, East Sarajevo Mentor: Gordana Milosavljević, PhD, Full Professor, Faculty of Technical Sciences, Novi Sad
Note:	

Sadržaj

REZIME

Istraživanje predstavljeno u okviru ove disertacije imalo je za cilj unapređenje procesa specifikacije korisničkih zahteva poslovnih aplikacija na bazi detaljnih, izvršivih prototipova koji se mogu kreirati uz minimalan utrošak vremena i energije. Radi postizanja ovog cilja je implementiran alat otvorenog koda pod nazivom Kroki (fr. *croquis* – skica) čija je arhitektura projektovana tako da obezbedi:

- kolaborativni razvoj specifikacije poslovne aplikacije sa korisnicima koji nemaju znanje projektovanja i programiranja softverskih sistema,
- efikasno pokretanje prototipa direktno iz sopstvenog razvojnog okruženja, dajući mogućnost korisniku da isproba prototip tokom modelovanja kad god poželi,
- ponovno korišćenje informacija dobijenih prilikom razvoja prototipova u kasnijim fazama razvoja, kako bi se smanjilo nepotrebno trošenje resursa.

Specifikacija poslovnih aplikacija koja se kreira u okviru Kroki alata je bazirana na internom UI standardu i EUIS (*Enterprise User Interface Specification*) DSL-u. Internim UI standardom su definisane funkcionalne i vizuelne karakteristike minimalno potrebnog broja UI komponenti poslovne aplikacije, sa namerom da se obezbedi jednostavno korišćenje aplikacije, brza obuka korisnika i automatizacija izrade korisničkog interfejsa.

Za specifikaciju poslovne aplikacije obezbeđene su tri konkretne sintakse EUIS DSL-a koju implementiraju odgovarajući editori Kroki alata:

- grafička sintaksa u vidu skica korisničkog interfejsa prilagođena radu sa korisnikom, koju implementira editor skica,
- grafička sintaksa koja podseća na UML dijagram klasa sa stereotipima, prilagođena za efikasan rad projektanata, koju implementira pojednostavljeni UML editor (*Lightweight* UML editor) i
- konkretna sintaksa tekstualnog komandnog jezika, namenjena članovima razvojnog tima kojima ne odgovaraju grafičke sintakse.

Podrška za efikasno pokretanje je implementirana na više nivoa:

- Zahvaljujući integrisanom meta-modelu EUIS DSL-a izbegnuto je trošenje vremena na Model-to-Model transformacije.
- Grafički editori Kroki alata omogućavaju specifikaciju poslovnih aplikacija uz unos samo osnovnih obeležja.
- Korišćenjem adaptivnih generičkih aplikacija koje dinamički prilagođavaju svoj izgled i funkcionalnost na osnovu podataka preuzetih iz modela je u velikoj meri skraćeno vreme koje se troši na Model-to-Text transformacije. Adaptivne generičke aplikacije su unapred implementirane, tako da se većinom generišu konfiguracione datoteke koje diktiraju izgled i funkcionalnosti prototipa. Ovi podaci se smeštaju u aplikativni repozitorijum koji je optimizovan za brz pristup. Podrška za unos ručnih izmena je obezbeđena putem aspekt-orijentisanog programiranja.
- H2 sistem za rukovanje bazama podataka koji je uključen u Kroki alat obezbeđuje perzistenciju unetih podataka tokom korišćenja prototipa, bez potrebe za dodatnim

konfigurisanjem. Međutim, podržano je i korišćenje proizvoljnog sistema za rukovanje bazama podataka, ukoliko razvojni tim tako odabere.

Razvijeni model baziran na EUIS DSL-u se može eksportovati u XMI format radi korišćenja od strane UML alata za modelovanje opšte namene. Programski kod razvijenog prototipa se takođe može eksportovati u Eclipse projekat. Na ovaj način se postiže ponovno korišćenje informacija dobijenih prilikom razvoja prototipova za potrebe kasnijih faza razvoja softvera. Dijagram klasa razvijen UML alatima opšteg tipa i snimljen u XMI format se može učitati u Kroki alat i pokrenuti.

Eksperiment za proveru da li razvijeni alat zadovoljava postavljene ciljeve je dizajniran kao serija od deset eksplorativnih studija slučaja čiji je cilj specifikacija poslovnih aplikacija sa učesnicima koji dolaze iz različitih poslovnih domena koji nisu poznati projektantima. Pojedinačne studije su obavljene sa po jednim učesnikom u okviru dvočasovnih projektantskih sesija, gde su ulogu projektanata imali autor ove disertacije i njegov mentor.

Kvalitativni i kvantitativni podaci prikupljeni tokom sesija i posle njih, putem upitnika, su iskorišćeni za izvođenje pozitivnih zaključaka o efikasnosti predloženog pristupa i alata. Dizajn istraživanja je baziran na konceptima MEM-a (*Method Evaluation Model*) koji definiše kriterijum za uspeh određene metodologije u praksi. Upitnici koji evaluiraju jezik za modelovanje i Kroki alat su formulisani tako da odgovaraju izabranim konceptima FQUAD (*Framework for qualitative assessment of domain-specific languages*) okvira za evaluaciju DSL-ova.

ABSTRACT

The research presented in this dissertation aimed to improve the process of specification of user requirements of business applications based on detailed, executable prototypes that can be created with minimal expenditure of time and energy. In order to achieve this goal, an open-source tool called Kroki (fr. *croquis* - sketch) was implemented, the architecture of which is designed to provide:

- collaborative development of business application specifications with users who do not have knowledge of designing and programming software systems,
- efficient prototyping directly from Kroki's development environment, enabling the user to try out the prototype during modeling whenever they want, and
- reuse of information obtained during the development of prototypes in later stages of development, to reduce unnecessary consumption of time and energy.

The specification of business applications created within the Kroki tool is based on the internal UI (user interface) standard and the EUIS (Enterprise User Interface Specification) DSL. The internal UI standard defines the functional and visual characteristics of the minimally required number of UI components of a business application, with the intention of ensuring simple use of the application, rapid user training and automation of user interface development.

Three concrete EUIS DSL syntaxes are provided for the specification of a business application, which are implemented by the corresponding Kroki tool editors:

- Graphic syntax in the form of user interface sketches intended for collaborative work with the user, implemented by the Mockup editor,
- A graphic syntax reminiscent of the UML class diagram with stereotypes, intended for efficient modeling, which is implemented by the Lightweight UML editor, and
- A textual command language, intended for development team members who are not comfortable with graphical syntaxes.

Support for efficient prototype activation is implemented at multiple levels:

- Thanks to the integrated meta-model of EUIS DSL, spending time on Model-to-Model transformations is avoided.
- Graphical editors of Kroki tools enable the specification of business applications by entering only basic properties.
- By using adaptive generic applications that dynamically adjust their appearance and functionality based on data retrieved from the model, the time spent on Model-to-Text transformations is greatly reduced. Adaptive generic applications are pre-implemented, so mostly configuration files are generated that specify the look and functionality of the prototype. These files are placed in an application repository that is optimized for fast access. Support for making manual changes is provided through aspect-oriented programming.
- The H2 database management system that is included in the Kroki tool ensures the persistence of the entered data during the use of the prototype, without the need for additional configuration. However, using an arbitrary database management system is also supported, depending on the choice of the development team.

The developed specification based on the EUIS DSL can be exported in XMI format for use by general-purpose UML modeling tools. The code of the developed prototype can also be exported to an Eclipse project. In this way, the reuse of information obtained during the development of prototypes for the needs of later stages of software development is achieved. A class diagram developed with generic UML tools and saved in XMI format can be loaded into the Kroki tool and run.

The experiment to validate whether the developed Kroki tool meets the set goals was designed as a series of ten exploratory case studies to specify business applications with participants from different business domains unknown to the designers. Individual studies were carried out within two-hour design sessions, where the author of this dissertation and his mentor played the designer role, with a single participant in the user role in each session.

Qualitative and quantitative data collected during and after the sessions, through questionnaires, were used to draw positive conclusions about the effectiveness of the proposed approach and tools. The research design is based on the concepts of MEM (Method Evaluation Model), which defines the criteria for the success of a certain methodology in practice. Questionnaires that evaluate the modeling language and the Kroki tool are formulated to correspond to the selected concepts of the FQUAD (Framework for qualitative assessment of domain-specific languages) for evaluating DSLs.

POGLAVLJE 1

UVODNA RAZMATRANJA

Tradicionalni pristupi izradi softverskih sistema uključivali su rad sa korisnikom softvera dominantno u fazi prikupljanja korisničkih zahteva, nakon čega je prvi sledeći susret organizovan obično kada je jedan, često veći deo, bio implementiran. To je bio trenutak kada su otkrivani mnogi aspekti u kojima razvijeni proizvod nije odgovarao potrebama korisnika, što je dovodilo do neprihvatanja projekta ili dodatnih troškova i odlaganja rokova, kako bi se uočeni nedostaci otklonili [77]. Pod korisnikom se u nastavku teksta podrazumeva osoba koja je zainteresovana za softver koji se razvija i potiče iz redova naručioca, krajnjih korisnika ili njihovih predstavnika i kao takva ima pravo da definiše zahteve.

Istraživanje u kojem je učestvovalo preko 350 američkih softverskih kompanija je pokazalo da su glavni uzroci neuspeha trećine od preko 8000 posmatranih projekata bili: nedovoljno učešće korisnika, nepotpuni korisnički zahtevi i promenljivost korisničkih zahteva [19]. Problemi koji mogu da ugroze projekat dolaze u velikoj meri iz faze specifikacije zahteva, često usled uzajamnog nerazumevanja korisnika i razvojnog tima. Prema [3], 64% grešaka su poticale iz ove faze kao i faze dizajna, iako su korisnici potpisali svu dokumentaciju proisteklu iz tih faza.

Nedovoljna ili neodgovarajuća komunikacija između razvojnog tima i korisnika predstavlja stalni izazov, tako da softverska zajednica ulaže kontinuirane napore za stvaranje kulture razvoja softverskih proizvoda koja bi omogućila njeno poboljšanje. Radi premošćivanja komunikacijskog jaza teži se što češćem i aktivnijem uključivanju korisnika u proces razvoja, kao i kontinuiranom isporučivanju razvijenih delova proizvoda u što kraćim iteracijama. Na ovim principima nastale su metodologije agilnog razvoja [4-6], čije su tehnike danas uveliko našle svoje mesto u industriji razvoja softvera. Kontinuirana isporuka (*continuous delivery*), iterativni i inkrementalni razvoj, kao i direktno uključivanje korisnika u razvojni tim (*user on site*) su značajno skratili vreme razvoja i omogućili uvid u proces svim zainteresovanim stranama. Uprkos ovome, i dalje postoji potreba za unapređenjem komunikacije u fazi specifikacije zahteva, pogotovo kada je reč o softverskim proizvodima kod kojih korisnički interfejs igra značajnu ulogu [1].

Kreiranje ranih prototipova, na osnovu kojih bi korisnici mogli da steknu realniju sliku o razvijanom sistemu i na taj način provere da li su njihovi zahtevi pravilno shvaćeni, se brzo nametnulo kao odgovor na ove probleme. Prototip predstavlja apstrakciju finalnog proizvoda ili nekog njegovog dela kreiranu u cilju testiranja nekog koncepta ili procesa [90]. Korišćenje prototipova je korisno u situacijama kada domen razvijanog sistema nije dovoljno definisan ili nije poznat razvojnom timu, u projektima koji su podložni promenama zahteva (inovativni ili visoko rizični projekti) ali i kao tehnika podsticanja korisničkog učestvovanja zbog bliskosti i lake razumljivosti nekih tipova prototipova krajnjim korisnicima.

Prototip može biti u bilo kojoj formi, od grubih skica nacrtanih ručno ili alatima za crtanje, do izvršive aplikacije kreirane specijalizovanim alatima koju korisnici mogu da isprobaju [3, 37, 40, 42]. Kreiranje različitih vrsta prototipova unutar softverskih timova je u velikoj meri (prema nekim autorima i do 70% vremena [91]) zastupljeno u toku procesa razvoja. Po načinu korišćenja, mogu biti jednokratni, odbacivi (*throwaway*) ili evolutivni. Jednokratni se koriste u svrhu testiranja pretpostavki, nakon čega se odbacuju i implementacija kreće od nule, na bazi prikupljenih

informacija. Zbog toga se preporučuje da razvoj ovakvih prototipova bude brz i jeftin, što obično podrazumeva kreiranje skica, crtanje različitih vrsta dijagrama i sl.

Evolutivni prototipovi, sa druge strane, su okosnica razvoja i oni se kontinuiranom nadogradnjom, kroz više iteracija, prevode u konačni proizvod. Iz tog razloga razvojni tim ulaže značajne resurse kako bi evolutivni prototip sadržavao sve tehnološke elemente finalnog proizvoda i zadovoljavao nefunkcionalne zahteve.

Odluka o tome koji tip prototipa će se kreirati vrlo često varira od projekta do projekta, a moguće su i situacije gde jednokratni prototip prerasta u evolutivni nakon obavljene specifikacije zahteva. U zavisnosti od prirode i kompleksnosti sistema, za jedan sistem se često kreira i više vrsta prototipova od kojih svaki pokriva određeni aspekt (dizajn korisničkog interfejsa, upravljanje pravima korisnika, projektovanje poslovnih procesa i dr.).

Softverski prototip se većinom smatra dobrom tehnikom za validaciju korisničkih zahteva jer predstavlja lako razumljiv i intuitivan model razvijanog softvera putem kojeg korisnici i softverski tim mogu naći zajednički jezik [40, 90]. Međutim, postoje i određene negativne posledice ukoliko se stavi prevelik naglasak na njih [91, 77]. Jeftini prototipovi često ne uspevaju da pobude dovoljno pažnje korisnika i pomognu da se otkriju omaške u specifikaciji projektovanog sistema. Detaljniji prototipovi mogu zahtevati previše vremena i energije tima i dovesti do iscrpljivanja, posebno ako se na kraju odbace [76]. U slučaju izvršivih prototipova implementiranih specijalizovanim alatima, može se desiti da korisnik stekne pogrešnu sliku o tome koliko je vremena potrebno za razvoj sistema jer mu je pružena prilika da isproba „sistem koji radi“ u kratkom roku. Ovo može dovesti do sumnje klijenta u troškove projekta, što rezultuje nepoverenjem između uključenih strana [77]. Kako bi se prevazišli ovi problemi potrebna je konstantna i transparentna komunikacija između njih kako bi se efektivno upravljalo očekivanjima.

Radi smanjivanja „šuma“ u komunikaciji, bitno je da prezentacioni detalji prototipa (boje, dizajn i sl.) ne skreću pažnju sa važnijih elemenata koji direktno utiču na funkcionisanje sistema, poput tipova podataka i poslovne logike. Neki autori iz ovog razloga preporučuju izbegavanje softverskih prototipova i korišćenje skica nacrtanih na papiru ili alatima za crtanje [76]. Skice predstavljaju prirodan i razumljiv način vizualizacije kompleksnih koncepata [31, 58] i kao takve su pogodne za kreiranje brzih prototipova razvijanog softverskog sistema. U istraživanju prezentovanom u [1] pokazuje se da je ovaj način kreiranja prototipova najčešće korišćena tehnika u fazi specifikacije zahteva (68.48% učesnika ankete). Isto istraživanje, kao i [20], pokazuje i da se nakon specifikacije zahteva, one uglavnom direktno ne koriste u daljem razvoju.

Nažalost, grube skice kreirane na papiru ili uz pomoć alata za skiciranje opšte namene često ne uspevaju da pobude dovoljno pažnje korisnika i pomognu da se otkriju omaške u specifikaciji projektovanog sistema. Najkorisnije povratne informacije možemo dobiti od korisnika koji je u prilici da rukuje izvršivim prototipom, u koji može da unosi podatke iz svojih realnih dokumenata i procesa, u okruženju koje je slično njegovom svakodnevnom radnom okruženju. Međutim, izvršivi prototipovi ovakve vrste se smatraju skupim sa stanovišta razvoja i zbog toga se tradicionalno ne upotrebljavaju u inicijalnim fazama projekta, već se koriste jeftinija, iako manje pouzdana sredstva [3].

Optimalan pristup dobijanju pouzdane specifikacije zahteva bilo bi jeftino i efikasno kreiranje izvršivih skica u saradnji sa korisnicima još u inicijalnim fazama projekta, kako bi se što pre moglo proveriti uzajamno razumevanje. Potrebna brzina i ekonomičnost razvoja, mogli bi se postići primenom inženjerstva vođenog modelima (*Model-Driven Engineering* – MDE) [8]. MDE ove zahteve obezbeđuje korišćenjem alata koji model na visokom nivou apstrakcije, kroz jednu ili više automatizovanih transformacija, prevode na izvršivi programski kod na ciljnoj platformi.

MDE ima više pravaca, a za ovu disertaciju je od najvećeg interesa razvoj softvera vođen modelima (*Model-Driven Software Development* – MDSD) koji kreira modele na bazi jezika specifičnih za domen (*Domain Specific Language* – DSL). Jezici specifični za domen su optimizovani za efikasno modelovanje u određenom, usko definisanom domenu primene [8]. Oni su ekspresivniji i lakši za razumevanje od jezika opšte namene jer koriste notacije i koncepte vezane za domen određenog problema. Koncepti iz domena dobijaju svoju reprezentaciju u okviru apstraktne sintakse jezika (meta-modela), a kreiranje modela na datom jeziku se vrši korišćenjem konkretnih sintaksi (notacija) jezika. Jezik specifičan za domen može imati proizvoljan broj konkretnih sintaksi (tekstualna, grafička, bazirana na dijalozima, itd.) što nam omogućava da se prilagodimo različitim tipovima učesnika u razvojnom procesu.

Glavni predmet istraživanja ove disertacije je unapređenje procesa specifikacije zahteva poslovnih aplikacija uz oslonac na izvršive prototipove, čiji razvoj treba da je dovoljno brz i jeftin tako da se mogu koristiti i u najranijim fazama razvoja softverskog proizvoda. Pod pojmom „poslovna aplikacija“ podrazumeva se podsistem informacionog sistema (IS) ili manji informacioni sistem namenjen za podršku jedne grupe poslova. Informacioni sistem u širem smislu predstavlja podršku poslovanju određene organizacije (preduzeća) čiji je zadatak prikupljanje i obrada podataka na osnovu kojih se dobijaju informacije neophodne za njeno funkcionisanje [77, 92]. Istraživanje ove disertacije se odnosi na informacione sisteme podržane računarima (*Computer-based* IS [77]), tako da će izraz „informacioni sistem“ ubuduće označavati ovaj tip IS.

Unapređenje procesa specifikacije zahteva poslovnih aplikacija je planirano da se sprovode kroz dve faze:

1. Razvoj softverskog alata baziranog na MDSD principima koji omogućava efikasno kreiranje izvršivih prototipova u saradnji sa korisnicima.
2. Sprovođenje istraživanja radi ispitivanja mogućnosti korišćenja takvog alata u inicijalnim razvojnim sesijama.

Radi evaluacije pomenutih ciljeva postavljene su sledeće hipoteze:

- **H0**: Korišćenje izvršivih modela u saradnji sa korisnicima u najranijim fazama razvoja dovodi do boljeg razumevanja između razvojnog tima i korisnika i skraćuje vreme specifikacije zahteva.
- **H1:** Jezik specifičan za domen modelovanja poslovnih aplikacija čija konkretna sintaksa izgledom oponaša skice korisničkog interfejsa je pogodan kao sredstvo komunikacije između projektanata i korisnika.
- **H2:** Moguće je razviti alat koji omogućava modelovanje na jeziku iz hipoteze H1 i direktno izvršavanje modela tako da korisnici i članovi razvojnog tima, u okviru sesija zajedničkog razvoja (kolaborativnih sesija), mogu da efikasno testiraju kvalitet specifikacije i stepen uzajamnog razumevanja.

1.1 Razvoj softverskog alata

Istraživanje u ovoj disertaciji je nastavak istraživanja iz [27], gde je prezentovan jezik za specifikaciju korisničkog interfejsa poslovnih aplikacija pod nazivom EUIS (*Enterprise User Interface Specification*) UML (*Unified Modeling Language*) profil. UML profil je proširenje UML-a specifičnim konceptima koji su potrebni za modelovanje u nekom domenu primene koji nije predviđen postojećim UML dijagramima. EUIS UML profil je kreiran tako da podrži brzo

modelovanje korisničkog interfejsa i laku integraciju sa postojećim alatima i razvojnim procesima [27].

Podloga za razvoj UML profila je interni standard korisničkog interfejsa (*User-interface* – UI) kojim su definisane funkcionalne i vizuelne karakteristike gradivnih elemenata poslovnih aplikacija na visokom nivou apstrakcije. UI standard je projektovan kao podloga za automatizaciju razvoja, radi postizanja što je moguće većeg procenta generisanog koda, uz što je moguće manji broj potrebnih gradivnih elemenata. Mali broj standardizovanih gradivnih elemenata doprinosi lakšoj obuci korisnika [27]. Interni UI standard je, sa manjim unapređenjima, primenjen od 1996. godine u preko 70 projekata informacionih sistema iz različitih oblasti, a za njegovo korišćenje je obučeno više od sedam hiljada korisnika (ne računajući *web* korisnike) [27].

Modelovanje ciljnog rešenja korišćenjem EUIS UML profila podrazumeva: (1) kreiranje modela podataka nezavisnog od platforme (*Platform Independent Model* – PIM) korišćenjem dijagrama klasa, (2) njegovu automatsku transformaciju u model korisničkog interfejsa, (3) ručnu popravku i dopunu modela korisničkog interfejsa i (4) generisanje koda za ciljnu platformu. Iako se na ovaj način može relativno brzo doći do izvršivog prototipa, korisnici se u evaluaciju mogu uključiti tek na kraju, kada je prototip završen. Modelovanje korišćenjem dijagrama klasa nije pogodno za zajednički rad sa korisnicima, tako da razvojni tim samostalno prevodi specifikaciju zahteva na model, što obično rezultuje potrebom za više iteracija dok se postigne uzajamno razumevanje [25].

Cilj razvoja novog alata je zadržati dobre strane pristupa iz [27] koje su proverene u praksi (UI standard, koncepte jezika, integraciju sa postojećim alatima za modelovanje), uz dobijanje mogućnosti:

- zajedničkog modelovanja sa korisnicima koji nemaju znanje projektovanja i programiranja softverskih sistema, korišćenjem prigodne konkretne sintakse,
- efikasnog pokretanja direktno iz razvojnog okruženja alata (bez potrebe za transformacijama, izvozom u programsko okruženje, konfigurisanjem okruženja za testiranje), dajući mogućnost korisniku da često isproba prototip tokom modelovanja,
- ponovnog korišćenja informacija dobijenih prilikom razvoja prototipova u kasnijim fazama razvoja, kako bi se smanjilo nepotrebno trošenje resursa na razvoj „od nule“.

U ovoj disertaciji je planirano da se:

- EUIS UML profil prevede u samostalan jezik pod nazivom EUIS DSL.
- Specificira grafička konkretna sintaksa EUIS DSL-a čiji elementi treba da podsećaju na elemente skica korisničkog interfejsa, radi podrške zajedničkog modelovanja sa korisnikom.
- Obezbedi grafički editor za datu sintaksu.
- Implementira generički razvojni okvir (*engine*, generička aplikacija) za direktno izvršavanje modela specificiranog EUIS DSL-om.
- Razvije podrška za izvoz programskog koda prototipa u programska okruženja opšte namene, radi mogućnosti nastavka razvoja, uz podršku za integraciju ručno pisanog i generisanog koda.
- Obezbedi podrška za razmenu UML dijagrama klasa između Kroki alata i alata za modelovanje opšte namene
- Razvije tekstualna sintaksa komandnog jezika sa pripadajućim parserom i komandnom konzolom, za podršku projektanata kojima ne odgovaraju grafičke sintakse.

Deo ovog alata koji je dobio naziv Kroki (od francuske reči *croquis* - skica) je razvijen u okviru disertacije [93]. U [93] je implementiran editor za konkretnu grafičku sintaksu EUIS DSL-a čiji izgled podseća na UML dijagram klasa. Grafički editor podržava automatsko raspoređivanje elemenata modela, u cilju dobijanja optimalnog rasporeda sa stanovišta čitljivosti i estetike.

1.2 Ispitivanje mogućnosti korišćenja razvijenog alata u inicijalnim razvojnim sesijama

Evaluacioni eksperiment je planiran da se dizajnira kao serija od deset eksplorativnih studija slučaja čiji je cilj specifikacija poslovnih aplikacija sa učesnicima koji dolaze iz različitih poslovnih domena koji nisu poznati projektantima. Specifikacija će se obaviti u okviru pojedinačnih dvočasovnih projektantskih sesija, gde ulogu projektanata igraju autor ove disertacije i njegov mentor.

Kvalitativni i kvantitativni podaci prikupljeni tokom sesija i posle njih, putem upitnika, biće korišćeni za izvođenje zaključaka o efikasnosti predloženog pristupa i alata i pravcima daljih unapređenja. Dizajn istraživanja će biti baziran na konceptima MEM-a (*Method Evaluation Model*) [47] koji definiše kriterijum za uspeh određene metodologije u praksi. Upitnici koji evaluiraju jezik za modelovanje i Kroki alat će biti formulisani tako da odgovaraju izabranim konceptima FQUAD (*Framework for qualitative assessment of domain-specific languages*) okvira za evaluaciju DSL-ova [53].

1.3 Struktura disertacije

Tekst ove disertacije se sastoji od ukupno šest poglavlja.

U drugom poglavlju su predstavljene osnovne oblasti, metodologije i tehnike razvoja softvera, kao i softverska rešenja na koje se oslanja istraživanje opisano u ovoj disertaciji. Prezentovane su oblasti inženjerstva zahteva, agilne metodologije, razvoj softvera vođen modelima i različite tehnike za sprovođenje istraživanja i dizajn eksperimenta. Analizirana su postojeća softverska rešenja za brz razvoj prototipova poslovnih aplikacija na bazi inženjerstva vođenog modelima, kao i eksperimenti za evaluaciju alata i metoda od strane korisnika.

U trećem poglavlju je prezentovano projektovanje i implementacija Kroki alata. Radi kompletnosti teksta, u prvom delu poglavlja je predstavljen interni UI standard i EUIS UML profil na koje se alat oslanja. Zatim je prikazana implementacija EUIS DSL-a kao samostalnog jezika i njegovih grafičkih i tekstualnih sintaksi od strane pripadajućih editora, implementacija generičke web aplikacije koja obezbeđuje efikasno pokretanje razvijene specifikacije i podsistemi za saradnju sa alatima za modelovanje i programiranje opšteg tipa. Na kraju poglavlja se nalazi poređenje Kroki alata sa postojećim rešenjima analiziranim u sekciji 2.2.4.

Četvrto poglavlje sadrži uputstvo za modelovanje korišćenjem implementiranih grafičkih editora, radi sticanja uvida u funkcionalnost alata i procene pogodnosti za realan razvoj.

Peto poglavlje detaljno opisuje eksperiment sproveden u cilju evaluacije opisanog alata i njegove mogućnosti korišćenja u inicijalnim fazama razvoja. U prvom delu poglavlja je objašnjen kriterijum izbora učesnika i dati su upitnici koji su korišćeni za dobijanje kvantitativnih podataka o studijama. U drugom delu su opisane pojedinačne studije. Prezentovani su podaci o učesniku, domenu aplikacije na čijoj specifikaciji zahteva je rađeno i zapažanja članova Kroki tima i učesnika koja predstavljaju izvor kvalitativnih informacija. Poslednji deo poglavlja sadrži diskusiju o rezultatima i o validnosti studije.

Šesto poglavlje zaključuje disertaciju i daje pravce daljeg razvoja.

POGLAVLJE *2*

TEORIJSKE OSNOVE I PRIKAZ STANJA U OBLASTI

U ovom poglavlju će biti predstavljene oblasti, metodologije i tehnike razvoja softvera, kao i softverska rešenja na koje se oslanja istraživanje opisano u ovoj disertaciji. Prezentovane su oblasti inženjerstva zahteva, agilne metodologije, razvoj softvera vođen modelima i različite tehnike za sprovođenje istraživanja i dizajn eksperimenta. Analizirana su postojeća softverska rešenja za brz razvoj prototipova poslovnih aplikacija na bazi inženjerstva vođenog modelima, kao i eksperimenti za evaluaciju alata i metoda od strane korisnika koji su poslužili kao inspiracija za eksperiment prezentovan u četvrtom poglavlju.

2.1 Metodologije i tehnike razvoja softvera

2.1.1 Inženjerstvo zahteva

Inženjerstvo zahteva (*requirements engineering*, RE) je disciplina koja se bavi obradom korisničkih (*user requirements*) ili softverskih (*software requirements*) zahteva [66, 84] i uključuje prikupljanje, analizu, specifikaciju i validaciju zahteva za jedan softverski proizvod [31, 66, 76, 79, 81, 86]. Softverski zahtev predstavlja osobinu ili uslov koji softverski sistem ili neki njegov deo treba da zadovolji kako bi omogućio korisniku postizanje željenog cilja u skladu sa njegovom specifikacijom [76, 79, 87]. Mogu se podeliti u dve velike grupe [76, 79, 84, 86] :

1. Funkcionalni – predstavljaju zahteve koje sistem treba da zadovolji kako bi ispunio jednu, jasno definisanu, funkciju. Definišu potrebne ulazne podatke, način na koji sistem funkcioniše u odnosu na zadani cilj i očekivane rezultate.
2. Nefunkcionalni – određuju na koji način bi neka funkcija trebalo da se izvrši kako bi se sistem smatrao upotrebljivim. Oni najčešće predstavljaju dopunu funkcionalnih zahteva.

U širem smislu, specifikacija softverskih zahteva predstavlja dokumentovani skup zahteva koji jedan softverski proizvod treba da ispunjava, kao i odgovore na pitanja vezana za širi kontekst projekta (potrebe za datim sistemom, čemu služe komponente sistema, na koji način će sistem biti implementiran, upravljanje promenama u toku projekta i dr.) [83, 86]. U tradicionalnim metodologijama rad sa zahtevima se odvijao u inicijalnim fazama razvoja, dok se u novijim metodologijama sprovodi tokom celokupnog životnog ciklusa.

RE predstavlja jednu od najstarijih disciplina u softverskom inženjerstvu, što se može videti po godinama izdanja publikacija koje su utemeljile oblast [66, 79, 86]. Prisutna literatura iz različitih decenija ukazuju na konstantnu aktuelnost teme [81, 82, 83, 84].

Osnovni učesnici u ovom procesu su krajnji korisnici i stručnjak za korisničke zahteve iz softverskog tima koji je zadužen za razvoj datog proizvoda. Često u ovom procesu učestvuju i druge zainteresovane strane kao što su: naručioci softvera, marketing stručnjaci i različiti tipovi softverskih inženjera [76]. Postoji i veliki broj softverskih alata čija je jedina namena podrška inženjerstvu zahteva [88, 89].

Sam proces prikupljanja i dokumentovanja softverskih zahteva nije strogo standardizovan, tako da njegov opis varira u zavisnosti od korišćene metodologije i navika softverskog tima. Ipak,

postoji određeni skup aktivnosti koji se može smatrati standardnim i koji može poslužiti za stvaranje šire slike o ovom procesu:

1. Prikupljanje zahteva (*requirements elicitation*) – predstavlja inicijalnu fazu specifikacije zahteva u kojoj je potrebno prikupiti što je moguće više informacija o razvijanom sistemu. Da bi se to postiglo, ova aktivnost se u velikoj meri oslanja na komunikaciju i saradnju svih uključenih strana i na primenu različitih tehnika kao što su: intervjuisanje različitih nivoa korisnika, kreiranje scenarija korišćenja, kreiranje prototipova različitog stepena detaljnosti, praćenje rada korisnika i druge.
2. Analiza zahteva (*requirements analysis*) – bavi se obradom prikupljenih zahteva sa ciljem njihove klasifikacije, razrešavanja potencijalnih konflikata između zahteva kao i inicijalnog modelovanja kako bi se stekla slika o obimu i prirodi projekta.
3. Specifikacija zahteva (*requirements specification*) – ima za cilj kreiranje formalne specifikacije zahteva na osnovu koje se može izvršiti njihova sistematična evaluacija. Ova specifikacija, u zavisnosti od obima i prirode projekta, ne mora nužno da bude jedan dokument.
4. Validacija zahteva (*requirements valdation*) – uključuje nekoliko pod-aktivnosti čiji je cilj evaluacija zahteva specificiranih u prethodno opisanoj fazi. Evaluacija se može izvršiti upotrebom nekoliko tehnika kao što su: inspekcija zahteva, kreiranje prototipa softvera na osnovu specifikacije ili sprovođenje različitih testova prihvatanja. Nakon ove faze obično se može početi sa razvojem opisanog softverskog proizvoda.
5. Upravljanje zahtevima u toku životnog ciklusa projekta – skup aktivnosti vezanih za softverske zahteve koji se sprovodi nakon što je razvoj softvera započeo i ima za cilj odgovaranje na promene zahteva. Upravljanje zahtevima nakon njihove formalne specifikacije je neophodno zbog promenljive i nelinearne prirode gotovo svih softverskih projekata. Neki od faktora koji mogu da utiču na promenu zahteva su dobijanje novih informacija nakon što je jedan deo sistema implementiran, tržišne promene, organizacione promene unutar kompanije naručioca ili softverskog tima, pogrešna specifikacija nastala usled međusobnog nerazumevanja uključenih strana i drugi.

Svaka od predstavljenih aktivnosti uključuje i ceo niz pod-aktivnosti čiji opis je ovde većinski izostavljen, tako da se čitaoci upućuju na priloženu literaturu i reference ukoliko žele da se detaljnije upoznaju sa ovim procesom.

2.1.2. Rapid Application Development

Metodologija rapidnog razvoja softvera (*Rapid Application Development* – RAD) prvi put se pojavila u istoimenoj knjizi Džejmsa Martina iz 1991. godine [94], ali je najveću popularnost stekla deceniju kasnije, kada je počela da se prihvata kao odgovor na probleme tradicionalnih metodologija razvoja informacionih sistema [77]. U svom osnovnom obliku RAD predstavlja metodologiju koja se bazira na inkrementalnom razvoju, jakom naglasku na korisničkom učešću, favorizovanju efikasnosti u odnosu na ispunjavanje svih korisničkih zahteva i oslanjanjem na softverske alate radi automatizacije razvoja [77, 95, 96].

2.1.2.1 Inkrementalni razvoj softverskog proizvoda i inkrementalni razvoj prototipova

Osnovna teza RAD metodologije je da se svi zahtevi jednog informacionog sistema ne mogu predvideti i definisati unapred, već se većina može jasno i precizno odrediti tek kada korisnici dođu

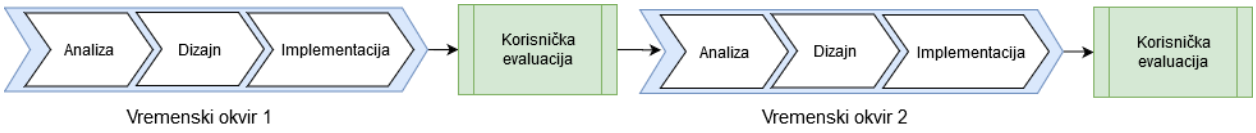
u dodir sa razvijenim sistemom. Uzimajući ovo u obzir, razvoj softvera vođen RAD-om počinje od malog skupa često površno definisanih zahteva koji se dorađuju i usavršavaju u toku procesa razvoja [77].

Inkrementalni razvoj se u velikoj meri oslanja na prototipove koji u ovoj metodologiji služe kao sredstvo provere razumevanja sa korisnicima, ali i kao osnova za razvoj samog proizvoda (oslonac na evolutivne prototipove). Veliki naglasak na upotrebu prototipova se ogleda u činjenici da se oni mogu koristiti u bilo kojoj fazi razvoja (dizajn, implementacija, testiranje, isporuka) i da se faze evaluacije i rafiniranja korisničkih zahteva mogu obavljati u nekoliko iteracija, uglavnom zahvaljujući lakoći razumevanja i korišćenja i maloj ceni izrade i dorade prototipova (u odnosu na tradicionalne metodologije), zahvaljujući snažnom osloncu na alate za automatizaciju.

2.1.2.2. Vremenske etape

RAD razvoj softvera se odvija u unapred definisanim vremenskim etapama (*timeboxing*) između kojih se obavlja korisnička evaluacija (slika 1). S obzirom da je naglasak na što kraćem vremenu isporuke, ispunjenje svih korisničkih zahteva se stavlja u drugi plan. Vremenski okviri uvek ostaju fiksni, dok se ciljevi vezani za ispunjenje korisničkih zahteva mogu smanjivati ukoliko razvojni tim proceni da ih ne može ispuniti. To je u suprotnosti sa tradicionalnim metodologijama razvoja gde se vreme i resursi smatraju promenljivim - česta praksa je pomeranje rokova ili alokacija većeg broja ljudi kako bi se odgovorilo na korisničke zahteve [77].

Broj i trajanje pojedinačnih etapa se određuju posebno za svaki projekat, pri čemu je preporuka da projekat ne treba da ima više od 90 etapa, dok se za trajanje svake od njih preporučuje od 90 dana do šest meseci [77]. Neki od autora tvrde da su projekti čije ukupno trajanje prelazi šest meseci u opasnosti da budu poslovno i tržišno prevaziđeni u trenutku isporuke [95].



Slika 1 Primer vremenskih okvira u RAD metodologiji.

Ovakav pristup razvoju ima više prednosti. Prva je brzina, odnosno skraćeno vreme isporuke finalnog proizvoda. Druga je podizanje poverenja i entuzijazma kod korisnika, zahvaljujući čestim evaluacijama koje se vrše na „živom“ sistemu, što omogućavaju lako i brzo otkrivanje grešaka i nepotpunosti u inicijalnoj specifikaciji zahteva koje se mogu mnogo lakše popraviti na sistemu koji je još uvek u razvoju. Kako bi se ovo pravilno izvelo, bitno je da se na korisničke zahteve odgovara prema njihovom prioritetu, pri čemu se oni najvažniji određuju i zadovoljavaju u prvoj vremenskoj etapi projekta, kako bi se korisnicima što pre isporučio sistem koji u svojoj osnovi ima potrebne funkcije.

2.1.2.3. JAD sesije

S obzirom da RAD metodologija stavlja veliki naglasak na uključivanje korisnika u proces razvoja, inicijalno planiranje, ali i svako dalje donošenje odluka, se obavlja učešćem svih zainteresovanih strana. Ovo se sprovodi organizovanjem sesija objedinjenog modelovanja (Joint Application Design - JAD) koje uključuju mali razvojni tim (obično četiri do osam [95] osoba), krajnje korisnike i naručioce proizvoda. Ove sesije služe kao zamena za tradicionalne korisničke intervjue koji se često sprovode izolovano sa različitim tipovima korisnika, tako da je posle teško usaglasiti konfliktne zahteve i suprotstavljena mišljenja.

Cilj JAD sesija je dovođenje malog broja relevantnih ljudi koji će u kratkom vremenskom roku doneti najbitnije odluke vezane za razvoj sistema. U inicijalnoj JAD sesiji se određuje broj i trajanja pojedinačnih etapa, kao i šta se očekuje da bude dostavljeno korisnicima na kraju svake etape. Dodatne JAD sesije se organizuju kako bi se evaluirao dosadašnji razvoj u odnosu na inicijalno zacrtan cilj. Važan element procesa je stvaranje atmosfere da sve uključene strane pripadaju jednom timu, tako da se ove sesije često organizuju u nekom neformalnom okruženju kao što je restoran.

2.1.2.4. DSDM

DSDM (*Dynamic Systems Development Method*) [77, 97] predstavlja jednu konkretnu implementaciju RAD metodologije predstavljenu od strane DSDM konzorcijuma [97] 1995. godine sa ciljem kreiranja RAD standarda na području Velike Britanije [95]. DSMD se zasniva na 9 osnovnih principa:

1. Obavezno aktivno učestvovanje korisnika,
2. Svi učesnici u projektu treba da budu u poziciji da donose odluke,
3. Fokus razvoja je na čestom isporučivanju rezultata,
4. Unapređenje poslovanja ima prioritet u odnosu na slepo ispunjavanje korisničkih zahteva prilikom testa prihvatanja određenog dela projekta,
5. Iterativni i inkrementalni način razvoja su neophodni kako bi projekat konvergirao,
6. Sve izmene u toku razvoja treba da mogu da se ponište ukoliko je potrebno,
7. Specifikacija zahteva služi kao osnova za definisanje generalne svrhe i obima projekta, bez specificiranja njegovih internih detalja,
8. Testiranje treba da bude integralni deo razvoja u svim fazama,
9. Saradnja svih uključenih strana je od suštinske važnosti za uspeh projekta.

2.1.2.5. RAD alati

Kako bi se proces razvoja učinio što efikasnijim, RAD metodologija se u velikom meri oslanja na alate za razvoj softvera, kreiranje softverskih prototipova, alate za modelovanje, CASE (Computer-*Aided Software Engineering*) alate i slično.
Upotreba ovakvih alata ima dve osnovne namene:

- Automatizaciju procesa razvoja
- Ponovnu upotreba (*reusability*) razvijenih specifikacija i modela, u cilju izbegavanja rasipanja vremena i energije.

Iako RAD metodologija danas ne predstavlja standard za razvoj informacionih sistema, principi na kojima ona počiva, kao što su naglasak na uključivanju korisnika, inkrementalni i iterativni razvoj i oslonac na alate, predstavljaju osnovu za nastanak niza „modernih“ metodologija koje su danas vrlo zastupljene.

2.1.3. Agilne metodologije razvoja

Agilne metodologije predstavljaju skup metodologija razvoja softvera nastalih u cilju prevazilaženja problema tradicionalnih metodologija da na vreme isporuče softver koji odgovara zahtevima korisnika [4, 17, 98]. Agilne metodologije počivaju na RAD principima, ali teže ka dodatnom poboljšanju efikasnosti procesa razvoja i prilagođavanju modernim tržišnim uslovima. Osnovni principi i vrednosti agilnog pristupa su definisani od strane grupe stručnjaka posebno oformljene sa ovim ciljem, pod nazivom Alijansa za agilni razvoj softvera (*Agile Software*

Development Alliance) ili samo Agilna alijansa (*Agile Alliance*) [78, 100]. Rezultat njihovog rada je Manifest agilnog razvoja [18] u kojem su definisane 4 osnovne vrednosti i 12 principa agilnog pokreta.

Vrednosti su:

1. **Osobe i njihove interakcije vrede više od procesa i alata**
   Agilni timovi treba da akcenat stave na učesnike u procesu razvoja i njihove interakcije u odnosu na metodološke procese i alate [77]. Oni se ne zalažu za ukidanje metodologija i alata, već stavljaju naglasak na stvaranju atmosfere u kojoj je razvoj upravljan kreativnim idejama koje dolaze kroz interakciju motivisanih pojedinaca u sklopu tima [27].
2. **Funkcionalan softver vredi više od opširne dokumentacije**
   Konstantno isporučivanje funkcionalnih delova softvera održava relacije sa klijentom i podstiče njihovo učestvovanje u projektu i dobijanje povratnih informacija u toku razvoja.
3. **Saradnja sa klijentima vredi više od pregovora oko ugovora**
   Postojanje ugovora je bitno, ali se ugovor ne može uzeti kao zamena za kontinuiranu komunikaciju sa korisnicima i pristajanje na promene, jer se samo na taj način može steći realna slika o tome šta krajnji korisnici očekuju od sistema koji se razvija [78].
4. **Reagovanje na promene vredi više od strogog praćenja plana**
   Promene su nešto što karakteriše proces razvoja softvera i u skladu sa tim, agilne metodologije stavljaju naglasak na kreiranju procesa razvoja koji unapred računa na njih i koji je u stanju da se na njih adaptira.

Vrednosti Agilnog manifesta predstavljaju nešto sa čime će se složiti većina softverskih inženjera i klijenata, mada, neke od navedenih praksi se ipak zanemaruju u većini timova [78]. Kako bi se detaljnije objasnio agilni razvoj, u Agilnom manifestu definisano je i sledećih 12 principa:

1. Najveći prioritet je zadovoljiti korisnika kroz brzu i kontinuiranu isporuku kvalitetnog softvera.
2. Prihvatati promenljive zahteve, čak i u kasnim fazama razvoja. Agilni razvoj podržava promene u korist klijenata.
3. Što češće isporučivati funkcionalan softver, u intervalima od nekoliko nedelja do nekoliko meseci preferirajući kraće intervale.
4. Domenski stručnjaci (korisnici) i razvojni tim moraju sarađivati na dnevnoj bazi u toku celokupnog trajanja projekta.
5. Organizovati projekat uz oslonac na motivisane članove tima. Obezbediti im odgovarajuće radno okruženje, pružiti im potrebnu podršku i imati poverenja da će posao biti urađen.
6. Najefikasniji način prenosa informacija u okviru tima, kao i između tima i korisnika, je putem direktne komunikacije “licem-u-lice”.
7. Funkcionalan softver je primarna mera napretka projekta.
8. Agilni procesi se zalažu za održivu stopu razvoja. Naručioci, razvojni tim i korisnici bi trebalo da budu u mogućnosti da održavaju isti ritam razvoja koliko god je potrebno.
9. Konstantan fokus na tehničku savršenost rešenja poboljšava njegovu agilnost.
10. Težnja ka jednostavnosti rešenja je esencijalna.
11. Najbolja rešenja dolaze od strane timova kojima je dozvoljeno da se samoorganizuju.

12. Potrebno je da tim, na bazi prethodnih iskustava, u redovnim vremenskim intervalima razmatra kako da bude efikasniji i prilagođava svoje ponašanje u skladu s tim.

Kao što se može videti na osnovu vrednosti i principa Agilnog manifesta, agilne metodologije se oslanjaju na iterativni i inkrementalni razvoj koji je organizovan u vremenske faze i strogo orijentisan ka komunikaciji između uključenih strana. Iako imaju različite tehnike i uloge, sve dele vrednosti i principe koji su gore izloženi. Neke od tih metodologija su: Ekstremno programiranje (*Extreme programming* - XP) [41, 78], SCRUM [98, 99], Agilno modelovanje (Agile *modelling* - AM) [78], Adaptivni razvoj softvera (*Adaptive Software Developm*ent - ASD) [77, 101] i Razvoj vođen funkcionalnostima (*Feature Driven Development* – FDD) [77, 102]. U daljem tekstu će biti predstavljeni XP i SCRUM.

2.1.3.1. Ekstremno programiranje (XP)

Ekstremno programiranje, sa stanovišta potrebnih artefakata razvoja, spada u grupu "lakših" agilnih metodologija [105]. Pogodno je za primenu unutar malih i srednje velikih timova [27, 77]. XP definiše skup vrednosti na kojima se metodologija zasniva ali ima i definisane procese kako treba sprovesti jedan XP projekat. Metodologija je nazvana *Ekstremno* programiranje jer je fokus na isporuci i razvoju, bez opterećivanja projekta kreiranjem opširnih modela i dokumentacije i razvojem komplikovanih generičkih rešenja.

Osnovne vrednosti XP-a su:

1. **Jednostavnost** – rešenja moraju da budu što jednostavnija i fokusirana na trenutne potrebe projekta, bez opterećivanja potencijalnim budućim potrebama (YAGNI – *You aren't gonna need it* princip).
2. **Komunikacija** – podstiče se tako što članovi XP tima rade zajedno, kako bi se znanje i iskustvo što brže razmenjivali i kako bi se podstakle diskusije vezane za potencijalne prepreke. Predstavnik korisnika je sve vreme sa razvojnim timom radi specificiranja zahteva i razrešavanja nedoumica (*user on site*). Razvoj se odvija u parovima (*Pair programming*) umesto da svaki član tima radi sam. U okviru programerskog para fokus jedne osobe je na tehničkim aspektima implementacije, dok je zadatak druge osobe da strateški planira što optimalnije rešenje. Dve osobe menjaju ove uloge u što češćim vremenskim intervalima.
3. **Povratne informacije** – konkretne i brze povratne informacije se očekuju od svih učesnika (razvojnog tima i korisnika), kao i od sistema, izvršavanjem raznih tipova testova.
4. **Hrabrost** – od članova XP tima se očekuje da u svakom trenutku budu spremni da odbace stare ideje i odreknu se urađenog posla ukoliko je to potrebno. Ovakav pristup i pogled na projekat se očekuje od svih uključenih strana.

U skladu sa predlozima Agilnog manifesta, XP projekti se realizuju u vremenskim inkrementima koji se u sklopu XP-a zove *Release*. Oni mogu da traju od nekoliko nedelja do nekoliko meseci i u sklopu njih projekat prolazi kroz pet faza (broj faza varira od konkretne implementacije XP pristupa od 4 do 6). Te faze su:

1. **Istraživanje** - u ovoj inicijalnoj fazi XP projekta cilj je sticanje što je moguće šire slike o projektu na osnovu koje bi učesnici mogli da baziraju dalji plan razvoja. Prikupljanje zahteva se sprovodi tako što korisnici pišu „korisničke priče" (*User stories*) koje imaju tačno

definisani format. Informacije dobijene analizom korisničkih priča služe za definisanje obima projekta, broja i trajanja vremenskih inkremenata, finansijskih detalja itd. U zavisnosti od obima projekta, ova faza može potrajati od nekoliko nedelja do nekoliko meseci [41].

2. **Planiranje** - cilj ove faze je izdvajanje podskupa korisničkih priča koje će biti implementirane u toku tekućeg inkrementa. Svaka odabrana korisnička priča se analizira i za nju se definiše skup pojedinačnih zadataka, na osnovu kojih se određuje njena "težina" (procena potrebnih resursa za implementaciju). Za svaki zadatak se određuje tim ili pojedinačna osoba zadužena za njegovu implementaciju. Za priče se definiše prioritet.
3. **Iterativni razvoj** - u ovoj fazi se odvija konkretna implementacija svih zadataka definisanih tokom planiranja trenutnog inkrementa. Ukoliko je faza planiranja dovoljno pažljivo izvedena, ovde bi svaki član razvojnog ima trebalo da ima dovoljno zadataka na kojima će raditi ceo inkrement. Međutim, ovde je uvek potrebno ostaviti prostora za re-evaluaciju zadataka i korisničkih priča jer je čest slučaj da se u toku razvoja identifikuju nove korisničke priče koje moraju da se implementiraju ili se zaključi da neke od priča ne mogu ili ne moraju da budu završene u trenutnom inkrementu. Kako bi svi članovi razvojnog tima bili u toku sa trenutnim stanjem razvoja, svakodnevno se održavaju kratki sastanci na kojima se izveštava o napretku i diskutuje o potencijalnim problemima [78]. Kako bi se interativni razvoj što efikasnije odvijao, ovde je veliki naglasak stavljen na podršku od strane alata koji služe za automatizaciju bitnih koraka kao što su jedinično testiranje i kontinuirana integracija [106, 107].
4. **Produkciona faza** - predstavlja poslednju fazu razvoja jednog XP projekta u sklopu koje se vrši testiranje celokupnog razvijenog sistema. Bitno je napomenuti da ulazak u produkcionu fazu ne znači nužno i prestanak inkrementalnog razvoja, jer je tu i dalje moguće raditi na nekim manje bitnim zadacima koji ne donose značajnije promene u sistem, ali je tempo razvoja značajno smanjen [78]. Uobičajena praksa je da se u ovoj fazi sistem testira u produkcionom okruženju i od strane korisnika, kao i da se deo razvojnog tima fokusira na sastavljanje različitih tipova dokumentacije.
5. **Održavanje** - predstavlja nastavak inkrementalnog razvoja sistema koji je prošao produkcionu fazu. U ovoj fazi se implementiraju dodatne korisničke priče koje su nastale kao dodatni korisnički zahtevi tokom vremena testiranja ili korišćenja sistema ili su preostale iz faza razvoja (uglavnom se ovde radi o zahtevima vrlo niskog prioriteta).

Neki XP timovi eksplicitno uvode i poslednju fazu koja se zove Faza terminacije i koja nastupa kada se realizuju sve korisničke priče ili kada projekat više ne može da se nastavi iz bilo kojeg razloga.

2.1.3.2. Scrum

Scrum predstavlja metodologiju agilnog razvoja koja naglasak stavlja na upravljanje procesom. Iako se termin Scrum pojavio i koristio tokom 80-ih i 90-ih godina 20. veka [108, 110], ova metodologija svoju veliku ekspanziju doživljava, zajedno sa celim agilnim pokretom, tokom prethodne dekade i ta popularnost još traje. Prema poslednjem godišnjem izveštaju o stanju agilnih metodologija [103] u kojem je učestvovalo preko 1000 ispitanika iz različitih industrijskih oblasti širom sveta, 75% je prijavilo da koristi Scrum ili neki hibridni pravac koji uključuje Scrum. Slično kao i u XP-u, razvoj Scrum projekta je organizovan u jasno definisane faze, pri čemu se glavni razvoj odvija u vremenskim inkrementima koji se u Scrum-u nazivaju sprintovi. Za razliku od XP-a, Scrum jasno definiše uloge i aktivnosti članova projekta i preporuke za trajanje Scrum sprintova.

Članovi jednog Scrum tima su podeljeni po sledećim ulogama:

1. ***Product owner*** – predstavlja most između korisnika i razvojnog tima i zadužen je za definisanje vizije i ciljeva projekta i brigu da se ta vizija realizuje i jasno iskomunicira sa svim učesnicima [109]. U skladu sa tim, član sa ovom ulogom ima mogućnost da odredi koji zadaci moraju da se implementiraju i sa kojim prioritetom. U Scrum timu postoji samo jedan član sa ulogom product owner-a i to je obično neko sa iskustvom i posedovanjem tehničkog i domenskog znanja, kao i veština upravljanja.
2. **Scrum *master*** – bavi se primenom Scrum metodologije i prilagođavanjem Scrum principa procesima konkretnog tima kako bi razvoj bio što efikasniji. U timu postoji samo jedan član sa ulogom Scrum master-a i to je obično posebno obučeni član za primenu date metodologije koji ne mora nužno da ima programerske veštine.
3. **Razvojni tim** – ostali članovi razvojnog tima sa različitim programerskim i projektantskim veštinama.

Osnovne aktivnosti i pojmovi vezani za Scrum proces su:

1. **Lista zahteva** (*backlog list*) - skup korisničkih priča koje moraju da budu implementirane u sklopu projekta. Osnovnu listu sastavlja *product owner* prikupljanjem zahteva na početku projekta u saradnji sa svim uključenim stranama. Na početku svakog sprinta se kreira trenutna lista zahteva izborom korisničkih priča koji će se pokušati implementirati u toku tog sprinta. Samo Scrum tim ima mogućnost da menja trenutnu listu zahteva – nikom spolja to nije dozvoljeno.
2. **Sprint** – vremenski inkrement u razvoju Scrum projekta. Scrum sprintovi obično traju do 3 nedelje i imaju isto trajanje sa unapred poznatim datumom početka i kraja [109].
3. **Planiranje sprinta** – odvija se na sastanku na početku sprinta na kojem *product owner* u saradnji sa svim zainteresovanim stranama određuje podskup korisničkih priča koje će se implementirati u toku sprinta.
4. **Dnevni sastanci** – kratki sastanci koji se održavaju jednom dnevno vođeni od strane *Scrum master*-a, na kojima svaki član tima u kratkim crtama treba da iznese šta je postigao od poslednjeg sastanka, šta planira da uradi do sledećeg i da li ima nekih poteškoća [109]. Cilj ovih sastanaka je obezbeđivanje uvida u napredak razvoja, mogućnost brzog otklanjanja prepreka i uzajamno podsticanje članova tima.
5. **Zatvaranje i analiza sprinta** - po završetku svakog sprinta održava se sastanak u kojem učestvuju sve strane uključene u projekat i na kojem se prezentuje šta je urađeno. Predstavlja kontrolnu tačku za proveru da li projekat ide u dobrom smeru. Ukoliko neke korisničke priče nisu implementirane, prenose se u naredni sprint.
6. **Retrospektiva sprinta** - poslednja aktivnost u jednom sprintu predstavlja retrospektivni sastanak unutar članova Scrum tima, pod vođstvom Scrum mastera. Cilj ovoga sastanka je analiza i prilagođavanje Scrum procesa na osnovu iskustava stečenih tokom sprinta [109].

Scrum projekat se završava kada se iscrpi lista zahteva ili kada se postigne saglasnost da je softver spreman za isporuku. Kako bi se Scrum što efikasnije primenjivao, preporuka je da timovi nemaju više od 9 članova. Ukoliko je neophodno, velike timove je moguće podeliti u više manjih od kojih svaki koristi Scrum, dok se njihovo funkcionisanje redovno sinhronizuje po Scrum principima (*Scrum of Scrums*) [27].

2.2 Inženjerstvo softvera vođeno modelima

Kod inženjerstva softvera vođenog modelima, model sistema zapisan dobro definisanim jezikom se, kroz jednu ili više automatizovanih transformacija, prevodi na programski kod koji se može izvršavati na ciljnoj platformi [163]. Za razliku od pristupa baziranih na modelima, gde se model obično razvija u početnim fazama i zatim koristi kao osnova za implementaciju koja često vremenom odstupi od njega, kod inženjerstva vođenog modelima on predstavlja okosnicu implementacije.

U ovom kontekstu jezik je dobro definisan ako poseduje:

- Apstraknu sintaksu (meta-model)
- Konkretnu sintaksu (notaciju)
- Semantiku.

Apstraktna sintaksa definiše koncepte domena, njihove osobine i međusobne odnose (relacije) [161]. Svi elementi modela kreirani na bazi datog jezika predstavljaju instance koncepata apstraktne sintakse.

Konkretna sintaksa (notacija), određuje na koji će način elementi jezika biti prezentovani (vizualizovani) korisniku. Jedan jezik može da ima više konkretnih sintaksi (grafičke, tekstualne, bazirane na formama itd.). Različite konkretne sintakse čine jezik lakšim za korišćenje različitim tipovima korisnika.

Tekstualne konkretne sintakse uveliko podseća na programske jezike opšte namene gde se program kreira zadavanjem tekstualnih komandi koje se prevode u mašinski jezik. Tekstualna notacija je uglavnom preferirani način korišćenja DSL-a od strane članova razvojnog tima koji su naviknuti na ovaj način korišćenja jezika zbog velikog iskustva sa različitim programskim jezicima opšte namene. Međutim, krajnji korisnici i domenski eksperti mogu preferirati neke druge vidove notacija, kao što je grafička sintaksa. Grafička sintaksa predstavlja grafičke oblike kako bi predstavila jezičke izraze i veze između njih. Ova sintaksa je vrlo pogodna za DSML jezike ali može biti korisna i za bilo koji tip jezika specifičnog za domen. Grafičke sintakse su uglavnom pogodne za prikazivanje i razumevanje strukture projekta, uključenih entiteta i veza između njih a pokazale su se i lakše za učenje [11, 112]. Zbog svega ovoga, grafičke sintakse se uglavnom smatraju prilagođenijima za korišćenje od strane krajnjih korisnika. Pored tekstualne i grafičke, postoje i drugi oblici konkretnih sintaksi kao što su simbolička i tabelarna.

Sekundarna notacija predstavlja upotrebu elemenata koji nisu deo jezika, ali utiču na razumljivost programa, odnosno modela. Elementi sekundarne notacije mogu biti prostorni raspored elemenata modela, boje, stilovi itd. [161]

Semantika definiše značenje sintaksno ispravnih iskaza na nekom jeziku [161]. Semantika se može definisati formalnim jezicima, ali i u vidu operativne semantike koja podrazumeva mapiranja iskaza datog jezika na druge jezike koji već imaju definisanu semantiku, što se u okviru MDE pristupa obično sprovodi generatorima koda ili interpreterima.

2.2.1 Model Driven Architecture - MDA

Najpoznatiji predstavnik inženjerstva vođenog modelima je MDA (*Model Driven Architecture*), čiji razvoj vodi OMG (*Object Management Group*) grupa[3]. Znanje o sistemu se u okviru MDA definiše u modelu na visokom nivou apstrakcije, koji ne zavisi od konkretne

[3] http://www.omg.org/

implementacione platforme (*Platform Indepedent Model* – PIM). Na osnovu PIM-a se, primenom Model-na-Model (Model-to-Model – M2M) transformacija dobijaju modeli zavisni od platforme (*Platform Specific Model* – PSM). PSM-ovi se mogu ručno menjati radi specificiranja implementacionih detalja određene platforme, posle čega se na osnovu njih generiše programski kod korišćenjem Model-na-Tekst (Model-to-Text – M2T) transformacija. Definicija transformacije predstavlja skup pravila koja opisuju kako se iskazi početnog jezika preslikavaju na iskaze ciljnog jezika [70, 148]. Cilj postojanja modela nezavisnih i zavisnih od platforme je pokušaj rešavanja problema prenosivosti stečenog znanja o sistemu na različite platforme, uz podršku za specifikaciju detalja koji su potrebni za uspešno generisanje koda za ciljne platforme [70].

Osnova za MDA je niz standardizovanih specifikacija definisanih od strane OMG grupe: *Meta Object Facility* (MOF), *Unified Modeling Languange* (UML), *Object Constraint Language* (OCL), XMI (*XML Metadata Interchange)*, itd.

MOF je OMG standard koji specificira meta-podatke i servise koji su podloga za razvoj meta-modela jezika. Zajednička osnova koji na ovaj način meta-modeli dobijaju je osnova za lakšu integraciju i implementaciju transformacija između modela na datim jezicima [150].

UML, poznat i kao jezik za objedinjeno modelovanje, je realizovan sa idejom da softverskoj zajednici omogući unificirano sredstvo za komunikaciju projektantskih ideja. Poseduje različite vrste dijagrama kojima se mogu modelovati statičke i dinamičke osobine sistema [13].

OCL je formalni jezik koji se koristi za specifikaciju različitih izraza u okviru MOF baziranih meta-modela i UML dijagrama klasa. Njime se definišu ograničenja, upiti, inicijalne vrednosti obeležja, preduslovi koji moraju biti zadovoljeni pre poziva metode i posledice koje moraju postojati posle poziva metode [149].

XMI je standard koji omogućava razmenu modela čiji su meta-modeli baziranih na MOF-u između alata različitih proizvođača [159].

U UML je ugrađen mehanizam profilisanja, tj. podrška za proširivanje UML meta-modela specifičnim konceptima koji su potrebni za modelovanje u nekom domenu primene koji nije predviđen postojećim UML dijagramima. Proširenje se naziva UML profil i sastoji se od stereotipa koji su proširenja postojećih meta-klasa UML-a, tagova (obeležja stereotipa) i OCL ograničenja. Neki primeri UML modela koji su postali industrijski standardi su SysML, razvijen od strane OMG grupe [142] ili UML profil za modelovanje XML šema predstavljen u [141].

Prednosti kreiranja UML profila u odnosu na realizaciju DSL-a od nule su što se kao editor može koristiti bilo koji alat za UML modelovanje, umesto razvoja sopstvenog okruženja, što značajno ubrzava razvoj. Mana je nedostatak slobode pri kreiranju proširenja – ne mogu se isključiti meta-klase koje nisu potrebne, niti se može kreirati nova meta-klasa. Za uspešnu realizaciju UML profila je potrebno detaljno poznavanje UML meta-modela koji je veoma kompleksan [13, 160].

2.2.2 Jezici specifični za domen

Jezici specifični za domen (*Domain-Specific Language* - DSL) predstavljaju vrstu računarskih jezika koji su specijalizovani za upotrebu u određenom uskom domenu primene [10, 112]. Nasuprot njima su jezici opšte namene (*General Purpose Language* - GPL) kojima se može rešavati proizvoljan skup problema.

Glavna prednost upotrebe jezika specifičnih za domen je što se njegovi koncepti direktno mapiraju na koncepte domena, koji može biti poslovni ili tehnički. Korišćenjem jezika opšte namene značajan napor se troši na prevazilaženje „apstrakcionog jaza“: mapiranje koncepata realnog sistema na koncepte jezika koji su na nižem nivou apstrakcije [162]. Korišćenjem DSL-ova apstrakcioni jaz ne postoji (pod uslovom da je DSL dobro isprojektovan). Na ovaj način se podstiče i učestvovanje domenskih eksperata u razvoju, jer su im pojmovi koji se koriste poznati i lako

razumljivi. Domenski eksperti su krajnji korisnici ili predstavnici naručioca softvera koji obično nemaju tehničko znanje, dok je njihovo domensko znanje od velikog značaja za projekat.

Prema načinu konstrukcije DSL-ovi mogu biti [113]:

- **Interni**: kreiraju se u okviru postojećih jezika, pri čemu nasleđuju razvojno okruženje i pripadajuće alate (editor, parser, itd). Oslonac na postojeći jezik značajno ubrzava razvoj, ali je ekspresivnost DSL-a i njegova fleksibilnost ograničena sintaksom jezika domaćina.
- **Eksterni**: razvijaju se „od nule" kao nezavisni jezici. Projektanti imaju punu slobodu prilikom kreiranja sintakse jezika, ali moraju razviti i prateće alate da bi jezik mogao da se koristi.

Postoje dva glavna pristupa definisanju sintakse DSL-a:

- Na bazi gramatičkih pravila (*grammar-based*) **–** gramatika se iskazuje u tekstualnoj formi, kao što je proširena Bakus-Naurova forma (EBNF) [117].
- Na bazi modela (*model-based*) **–** apstraktna sintaksa se specificira na bazi meta-meta-modela kao što je MOF [118].

2.2.3 Struktura ciljne aplikacije

Ciljna aplikacija kreirana na osnovu modela pisanog na jeziku specifičnom za domen u opštem slučaju se sastoji od tri celine koje je potrebno uklopiti [163]:

- Automatski generisanog koda
- Ručno pisanog koda
- Razvojnog okvira specifičnog za domen (*framework*).

Automatski generisani kod je rezultat M2T transformacija koje se obično implementiraju generatorima koda na bazi šablona. Za svaku vrstu datoteke koja se generiše je potrebno implementirati minimalno jedan šablon i pripadajući generator koda [27].

Preporuka je da se, pored generatora koda, implementira i razvojni okvir čije komponente implementiraju gradivne elemente aplikacije koji odgovaraju pojmovima iz domena. U tom slučaju, generisani kod je značajno manjeg obima jer se oslanja na već pripremljenu podlogu, pa generisanje koda, kompilacija i pokretanje kraće traje. U slučaju tehničkih domena, mogu se naći i odgovarajući gotovi okviri za razvoj.

Ciljne aplikacije koje se ne oslanjaju na okvire za razvoj obično imaju veliki broj linija generisanog koda koji je jednostavan za razumevanje i lak za održavanje, ali generisanje i pokretanje traje značajno duže u odnosu na prethodno rešenje.

Veoma kompleksni okviri za razvoj koji su implementirani kao samostalne aplikacije koje dinamički prilagođavaju svoj izgled i ponašanje na osnovu podataka iz modela se zovu generičke aplikacije (*engine*). One mogu da direktno čitaju model (ponašaju se kao interpreteri), ili im se podaci iz modela mogu pripremiti u formi pogodnoj za efikasno izvršavanje. U tom slučaju, generatori proizvode datoteke sa podacima o modelu, umesto programskog koda.

Generičke aplikacije su komplikovane za implementaciju, ali je pokretanje ciljne aplikacije skoro trenutno jer nema potrebe za kompilacijom generisanog programskog koda.

Nezavisno od izabranog pristupa, potrebno je podržati i uključivanje ručno pisanog koda za implementaciju funkcionalnosti koje nisu podržane jezikom, na način da ponovno generisanje ne ugrozi ručne izmene. MDE rešenja koja ovo ne podržavaju se ne mogu uključiti u iterativni i

inkrementalni način rada koji zahtevaju agilne metodologije. Ako je moguće, arhitekturu ciljne aplikacije treba tako organizovati da je ručno pisani i generisani kod fizički odvojen u posebnim datotekama [164].

2.2.4. Postojeća softverska rešenja za brzo kreiranje prototipova na bazi MDE tehnika

U ovoj sekciji su prezentovani i analizirani radovi i softverski alati koji se bave brzim razvojem prototipova korišćenjem MDSD ili MDA pristupa. Naglasak je na alatima koji koriste skice za dobijanje prototipa, mada su prikazani i alati koji kao ulaz koriste različite vrste grafičkih i tekstualnih modela.

Proučavanje postojećih alata i njihovih ograničenja je omogućilo sticanje jasnije slike kakve karakteristike MDSD alata su potrebne da bi se mogao efikasno uključiti počevši od inicijalnih faza životnog ciklusa projekta.

2.2.4.1. Umple

Na univerzitetu u Otavi (Kanada) je razvijen jezik UMLE (UML *Programming Language*) sa pripadajućim editorima i generatorima koda [35]. UMPLE jezik ima grafičku i tekstualnu notaciju koje se koriste za specifikaciju UML dijagrama klasa i dijagrama prelaza stanja. Cilj postojanja obe notacije je bio približavanje modelovanja programerima koji nisu skloni razvoju modela.

Tekstualna notacija je dizajnirana da podseća na Java programski jezik, dok grafička notacija odgovara UML dijagramu klasa [124]. Kreiranje modela se može vršiti na oba načina u grafičkom i tekstualnom editoru, pri čemu je moguće u svakom trenutku preći iz jednog u drugi, uz čuvanje izmena. Ova funkcionalnost je podržana i u Kroki alatu i njena osnovna namena je podrška za kreiranja modela od strane više vrsta korisnika koji mogu da preferiraju različite notacije i načine rada.

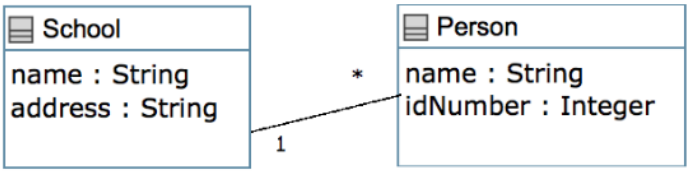
Na Listingu 1 se može videti primer modelovanja jednostavnog sistema pomoću UMPLE tekstualne notacije.

```
class Person {
       String name;
       Integer idNumber;
       key { idNumber }
}

class School {
       String name;
       String address;
       1 -- * Person student;
       key { name }
}
```

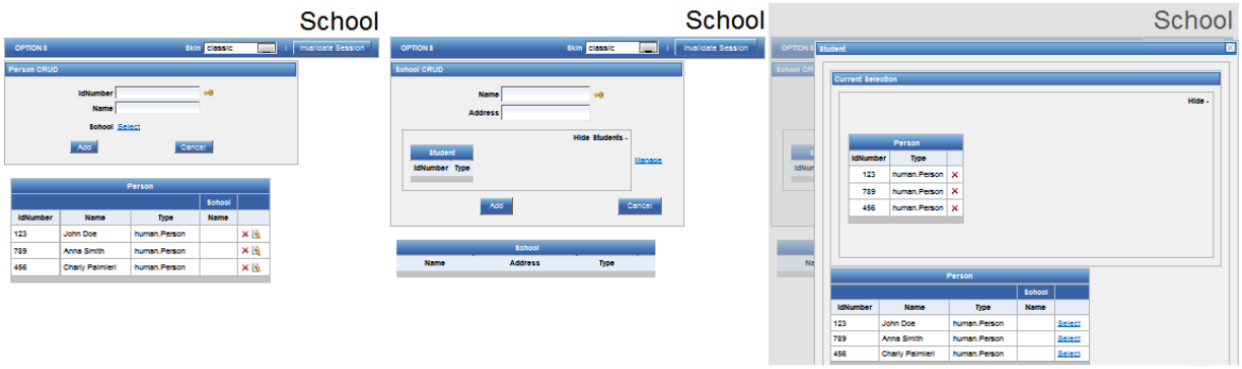
Listing 1 Primer tekstualne notacije UMPLE jezika

U primeru su modelovane dva klase, `Person` i `School` sa svojim atributima i njihovim tipovima. Kao identifikator klase `Person` se koristi atribut `idNumber` dok se klasa `School` identifikuje atributom `name`. Klase su povezane asocijacijom jedan-na-više koja specificira da jedna škola može imati jednog ili više polaznika koji su tipa `Person` (kraj ove asocijacije je u modelu sa primera nazvana `student`). UMPLE grafički dijagram koji odgovara ovoj tekstualnoj specifikaciji je dat na Slici 2.



Slika 2 Primer grafičkog UMPLE dijagrama koji odgovara tekstualnoj notaciji sa listinga 1

UMPLE paket poseduje više generatora koji podržavaju generisanje programskog koda za programske jezike Java, Ruby i PHP. Za ovo istraživanje je interesantan UIGU (User Interface Generator for UMPLE) [36] koji automatizuje razvoj prototipova korisničkog interfejsa web aplikacija. Prototip se generiše kao JSF (Java Server Faces) web aplikacija koja pruža mogućnost obavljanja osnovnih CRUD (Create, Read, Update, and Delete) operacija nad instancama modelovanih klasa. Primer korisničkog interfejsa generisanog na osnovu listinga 1 je dat na Slici 3.



Slika 3 Primer UI prototipa generisanog pomoću UMPLE alata

Prema [124] moguće je razviti namenske generatore i za druge tehnologije i programske jezike kao. Postojeći generatori podržavaju integraciju ručno pisanog i generisanog programskog koda. Rezultat generisanja u prikazanoj implementaciji je Java WAR (*Java Web Archive*) datoteka koja se može pokrenuti u nekom od standardnih aplikativnih servera (Apache Tomcat, JBoss i sl).

Prilikom implementacije ovog generatora posebna pažnja je posvećena lakoći korišćenja i brzini generisanja prototipova koji su upotrebljivi bez potrebe za detaljnim konfigurisanjem. Ovo je omogućeno korišćenjem predefinisanih vrednosti ukoliko neki od podataka nije specificiran od strane projektanta.

UMPLE programski paket predstavlja značajan korak u rešavanju problema kreiranja i korišćenja brzih prototipova korisničkog interfejsa koji deli nekoliko značajnih osobina sa Kroki alatom. Za razliku od Kroki alata, obe notacije UMPLE jezika su namenjene za projektante i programere, tako da uključivanje korisnika u proces razvoja nije direktno podržano. Pokretanje prototipa iz UMPLE okruženja takođe nije podržano, već je dobijenu web aplikaciju potrebno ručno pokrenuti u sklopu web servera, što zahteva dodatno tehničko znanje i troši vreme koje bi se moglo iskoristiti za modelovanje i evaluaciju sa korisnikom.

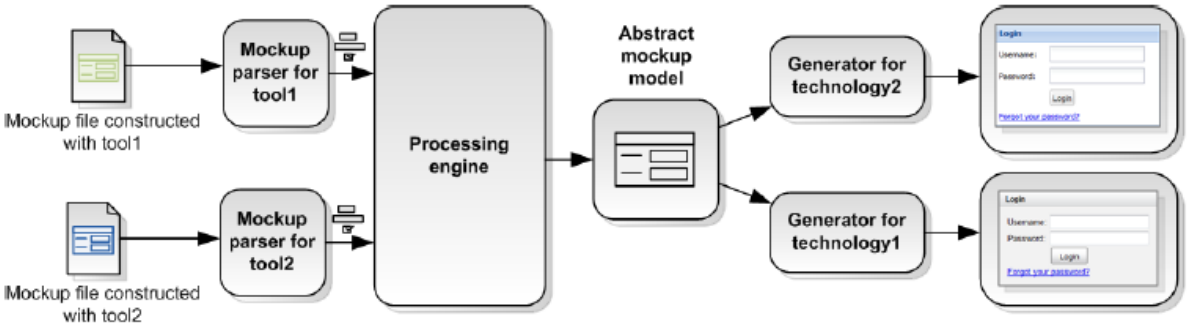
Pored prikazanih funkcionalnosti, UMPLE alati podržavaju niz korisnih dodataka koji omogućavaju napredno modelovanje i generisanje prototipova čiji se detaljniji opis može naći u originalnim publikacijama [35, 124, 125].

2.2.4.2. Oslonac na postojeće alate za crtanje skica

U [37] je predložen pristup korišćenju skica korisničkog interfejsa koje se crtaju nekim od postojećih alata opšte namene, kao što su npr. Axure, Balsamiq ili Pencil. Osnovni cilj ovog pristupa

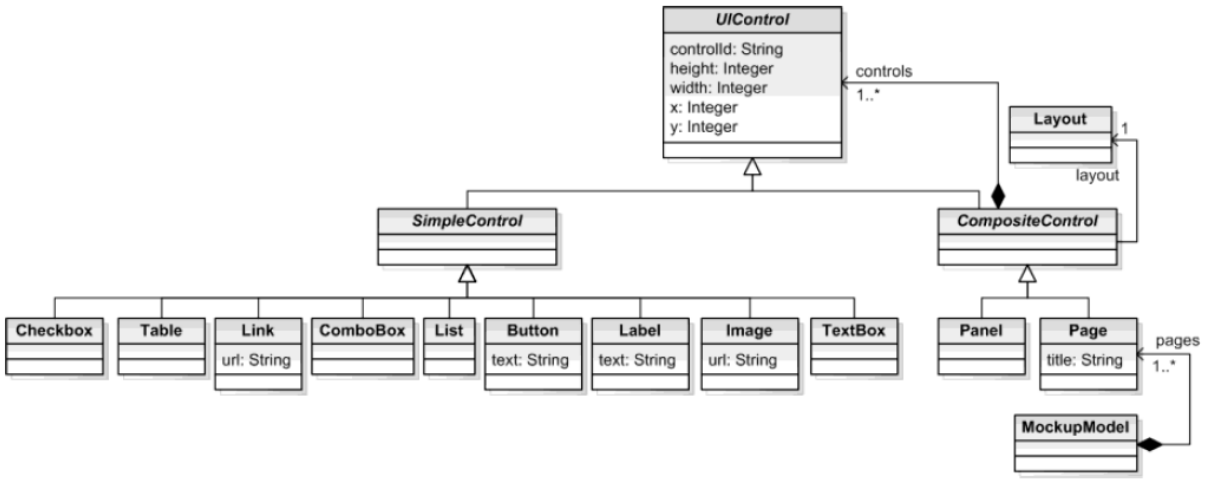
je uvođenje skica, koje su lako razumljive i mogu se kreirati od bilo koje stranke uključene u razvoj softverskog proizvoda, korišćenjem alata sa kojima su od ranije upoznati, bez potrebe za dodatnom obukom. Date skice se parsiranjem izlaznih datoteka navedenih alata prevode u model baziran na razvijenom meta-model-u prikazanom na slici 5. Dobijeni model se zatim koristi za generisanje prototipa ili kao osnova za ručnu implementaciju programskog koda. Razvojni proces je prikazan na slici 4.

Izbor podržanih alata kao i razvoj parsera za svaki od njih je ostavljen na implementaciju razvojnom timu koji koristi ovaj pristup. U [37] je navedeno da je osnovni zadatak datih parsera identifikacija elemenata korisničkog interfejsa i njihovo grupisanje, kako bi se mogli mapirati na instance elemenata meta-modela.



Slika 4 Proces baziran na parsiranju skica kreiranih alatima za crtanje opšte namene [37]

Dobijeni model predstavlja platformski nezavisnu specifikaciju korisničkog interfejsa koja služi za generisanje konkretne implementacije na odabranim platformama. Kako bi se ovo omogućilo, potrebno je razviti odgovarajuće generatore koda za svaku od željenih platformi i formata. U [37] se ne bave ovim delom, već je implementacija generatora ostavljena razvojnim timovima koji se odluče da koriste predloženi pristup. Autori tvrde da je prezentovani meta-model dovoljan za specifikaciju prostornog rasporeda velike grupe korisničkih interfejsa i kao dokaz ovog koncepta prilažu svoje uspešne implementacije parsera i generatora za više popularnih alata i platformi.



Slika 5 Meta-model za specifikaciju skica [37]

Kao prednost ovog pristupa autori navode mogućnost jednostavnog uključivanja u postojeći proces razvoja, jer učesnici ne moraju da uče korišćenje novih alata, kao i korišćenje skica u kasnijim fazama razvoja, čime se sprečava rasipanje uloženog vremena i energije. Nažalost,

model se dobija parsiranjem skica, što uvodi dodatni korak koji usporava razvoj i povećava mogućnost pojave grešaka prilikom mapiranja elemenata skice na elemente meta-modela.

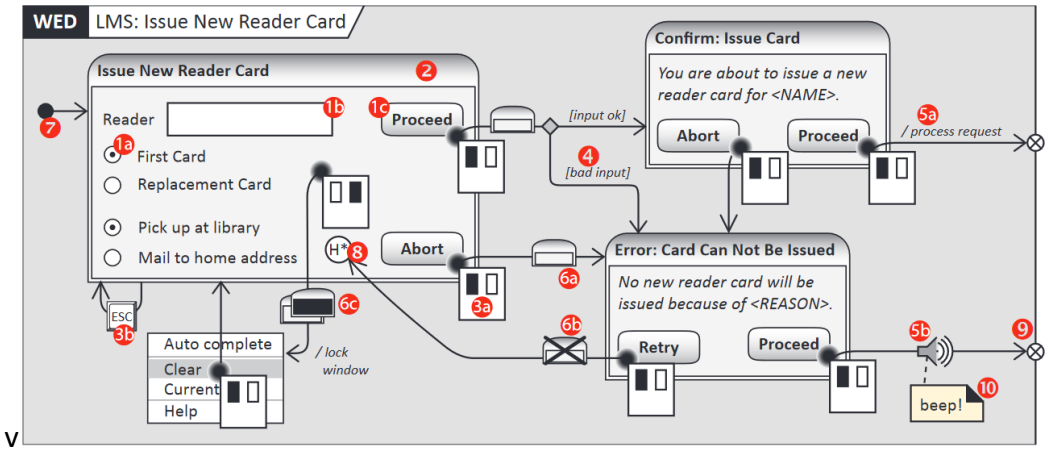
2.2.4.3. Window/Event dijagrami

U [38] je predstavljeno integrisano okruženje za specifikaciju izgleda i ponašanja korisničkog interfejsa uz pomoć grafičkog jezika za razvoj Window/Event – WED dijagrama. WED dijagrami predstavljaju kombinaciju skica korisničkog interfejsa i UML dijagrama prelaza stanja [126].

Svaki od prozora ili dijaloga specificirane aplikacije na WED dijagramu predstavlja jedno stanje, a događaji nad njihovim elementima (npr. klik mišem na dugme, unos teksta u polje i sl.) mogu da iniciraju prelaze stanja. Primer WED dijagrama je dat na Slici 6.

Cilj razvojnog okruženja je da obezbedi sledeće:

1. **Kontinuiran tok razvoja** - pretvaranje inicijalne specifikacije u formalne modele i zatim u programski kod.
2. **Sveobuhvatni dizajn** - objedinjavanje vizuelnih aspekata sa specifikacijom željenog ponašanja korisničkog interfejsa.
3. **Skalabilan nivo apstrakcije** - održavanje dovoljno visokog nivoa apstrakcije kako bi specifikacija bila što jasnija, a da se pri tome ne izgube tehnički detalji potrebni za implementaciju.



Slika 6 Primer WED dijagrama [38]

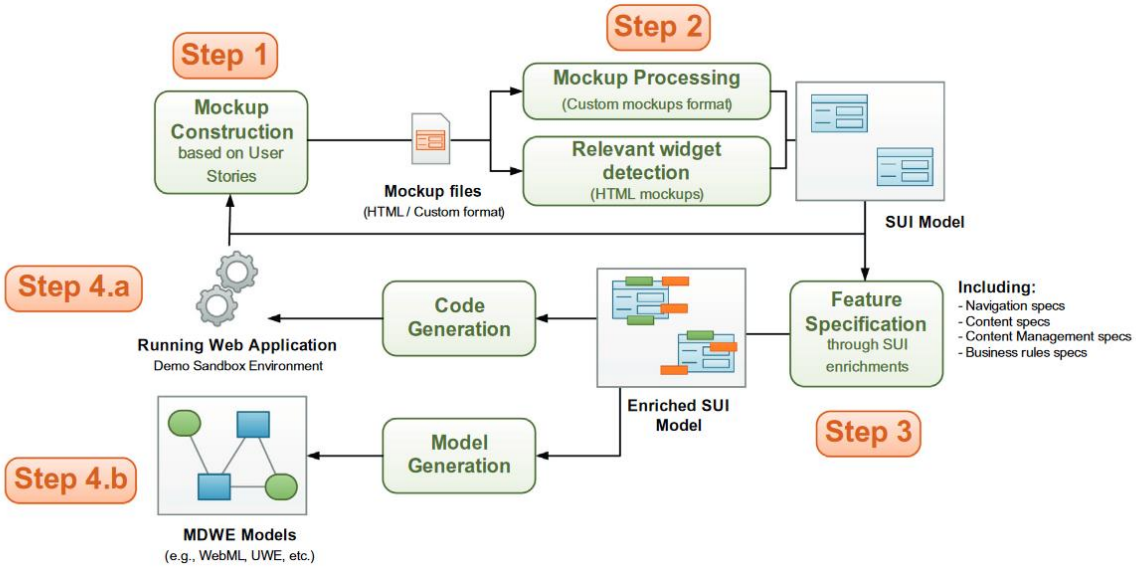
Konkretna implementacija predloženog pristupa se sastoji od grafičkog jezika za kreiranje WED dijagrama, okruženja za njegovo kreiranje – AIDE (Advanced Interaction Design Environment) [127] i generatora programskog koda [38]. Nažalost, u [38] i [127] se ne mogu naći detaljnije informacije o načinu implementacije i pokretanju generatoru koda, osim da generiše korisnički interfejs na XUL (XML User Interface Language) [128] jeziku.

U radu je opisan i kratak eksperiment sproveden sa ciljem evaluacije ove metode koji će biti opisan u posebnom poglavlju.

2.2.4.4. MockupDD

U [20] je prikazan MockupDD pristup (skraćeno od *Mockup-Driven Dev*elopment) koji razvoj web aplikacija vođenih modelima (*Model-Driven Web Engineering* - MDWE) proširuje crtanjem skica (*mockup*) u početnim fazama specifikacije zahteva. MockupDD se sastoji od sledećih koraka (slika 7):

1. **Kreiranje skica** – proces započinje prikupljanjem zahteva u saradnji sa korisnikom u vidu korisničkih priča, na osnovu kojih se zatim kreiraju skice korisničkog interfejsa, crtanjem na papiru ili uz pomoć nekog od alata za skiciranje.
2. **Procesiranje skica** – skice se importuju i automatski mapiraju na model baziran na SUI (*Structured User Interface*) meta-modelu [129]. Ovaj meta-model je razvijen od strane istih autora i sadrži koncepte za opis korisničkog interfejsa tzv. *data-entry* web aplikacija. Zadatak razvojnog tima je da obezbedi alate za procesiranje svih tipova skica koje želi da podrži.
3. **Specifikacija funkcionalnosti** – SUI modeli se proširuju specifikacijom funkcionalnosti modelovane aplikacije, dodavanjem oznaka (*tag*) koje treba da obezbede dodatne ulazne podatke za generatore koda. MockupDD ima kompleksan skup oznaka koji je detaljno opisan u [20]. Oznake se dodaju uz pomoć namenski razvijenog alata.
4. **Generisanje programskog koda i modela** – Na osnovu proširenih SUI dijagrama se generišu modeli i funkcionalni prototipovi implementirani na željenim programskim jezicima i platformama. Na razvojnom timu je da obezbedi potrebne generatore. Generisani prototipovi se koriste za evaluaciju od strane korisnika.



Slika 7 MockupDD proces [20]

MockupDD je uključen u Scrum i podržava iterativni i inkrementalni razvoj.

U [58] je opisan DataMock koji predstavlja specijalizaciju MockupDD procesa za generisanje UML modela podataka na osnovu skica korisničkog interfejsa. UML modeli podataka [130] predstavljaju vrstu UML dijagrama klasa koji služe za modelovanje i generisanje perzistentnog sloja softverskih rešenja i najveću primenu imaju u razvoju aplikacija koje naglasak stavljaju na podatke i njihove veze. DataMock proces je domenski orijentisan na web informacione sisteme i pogodan je za uključivanje u agilne metodologije kao što su MDWE i UWE (U*ML-Based Web Engineering*) [130].

U [58] su predstavljena i proširenja u odnosu na prethodnu verziju za proveru ispravnosti i automatsko ispravljanje grešaka u specifikaciji zahteva, kao i primeri integrisanja DataMock procesa u postojeće agilne metodologije.

2.2.4.5. Prototizer

Prototizer je alat otvorenog koda koji se instalira u vidu dodatka za Eclipse[4] platformu i dostupan za preuzimanje sa [134]. Kreiran je sa ciljem da bude pogodan za uključivanje u agilne metodologije [39]. Kao uzrok zašto MDSD alati obično nisu odgovarajući za agilan razvoj autori navode:

1. **Nefleksibilnost generatora koda** – teški su za prilagođavanje određenom razvojnom timu ili domenu; često ne podržavaju iterativni i inkrementalni razvoj.
2. **Komplikovan proces učenja** usled preširokog skupa funkcija koje obično podržavaju.

Osnovna funkcija Prototizer-a je generisanje programskog koda na osnovu UML modela. Koristi se samo u fazi implementacije, tako da se lako uključuje u bilo koji proces i metodologiju razvoja. Generisanje programskog koda je bazirano na MOFScript [135] tehnologiji i podržava inkrementalni razvoj i uključivanju ručno pisanog koda u generisano rešenje.

U [39] je predstavljena studija slučaja razvoja informacionog sistema nazvanog CODIFIX koji predstavlja pojednostavljenu verziju sistema za upravljanje resursima. U predstavljenom primeru su generisani sledeći programski artefakti:

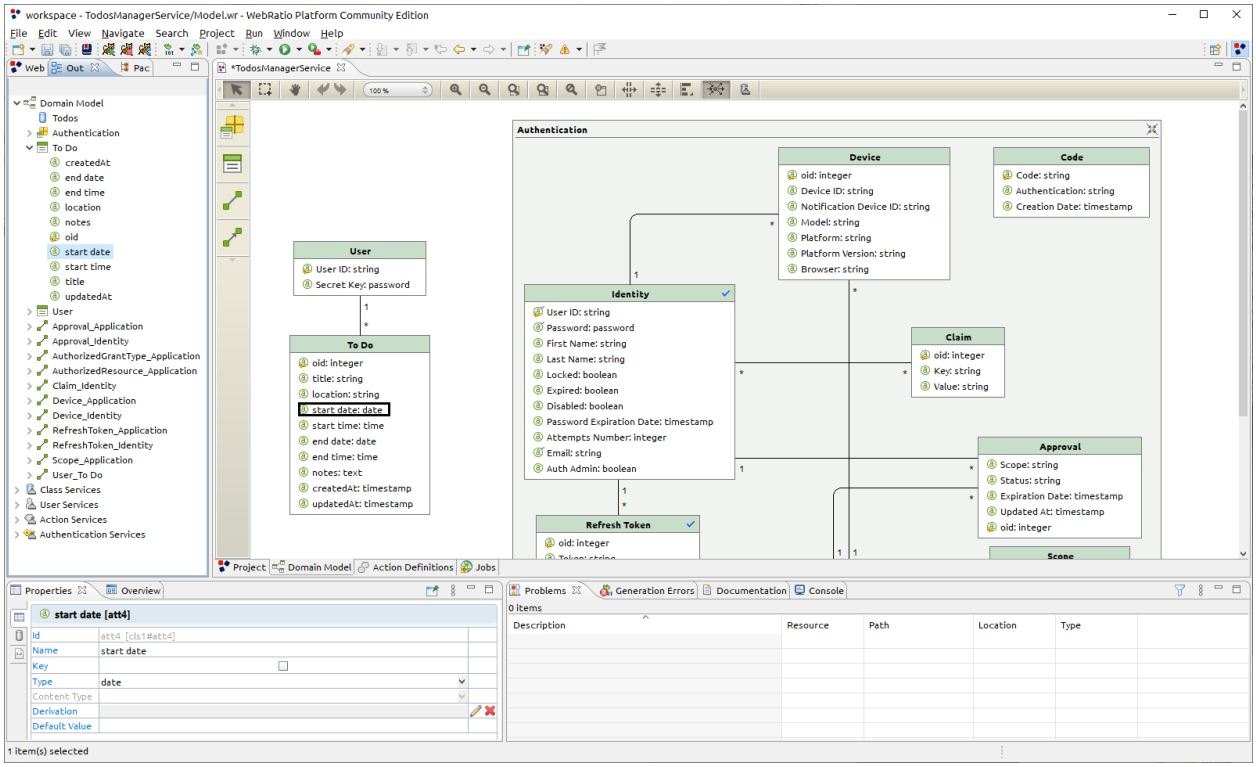
1. Šema relacione baze podataka i API (*Application Programming Interface*) za rad sa bazom,
2. Modelske PHP[5] klase koje predstavljaju entitete u sistemu,
3. Prezentacioni sloj web aplikacije u vidu HTML (*HyperText Markup Language*) stranica sa JavaScript[6] podrškom.

Procenat generisanog koda u prikazanom primeru je 78%. Sistem je implementiran u 22281 liniji programskog koda, od kojih je generisano 17546. Neki delovi generisanog programskog koda nisu iskorišćeni u finalnom rešenju.

2.2.4.6. WebRatio

Jedan od najkompleksnijih alata koji je proizašao iz ove grupe je WebRatio [61, 68, 69] koji se trenutno distribuira kao komercijalna platforma za razvoj koju čine tri osnovna podsistema: alat za razvoj mobilnih aplikacija, alat za razvoj web aplikacija i BPM (*Business Process Modelling*) alat. Incijalno, WebRatio je nastao kao MDSD alat baziran na Eclipse platformi i WebML [63] jeziku specifičnom za domen.

[4] https://www.eclipse.org
[5] https://www.php.net/
[6] https://www.javascript.com/

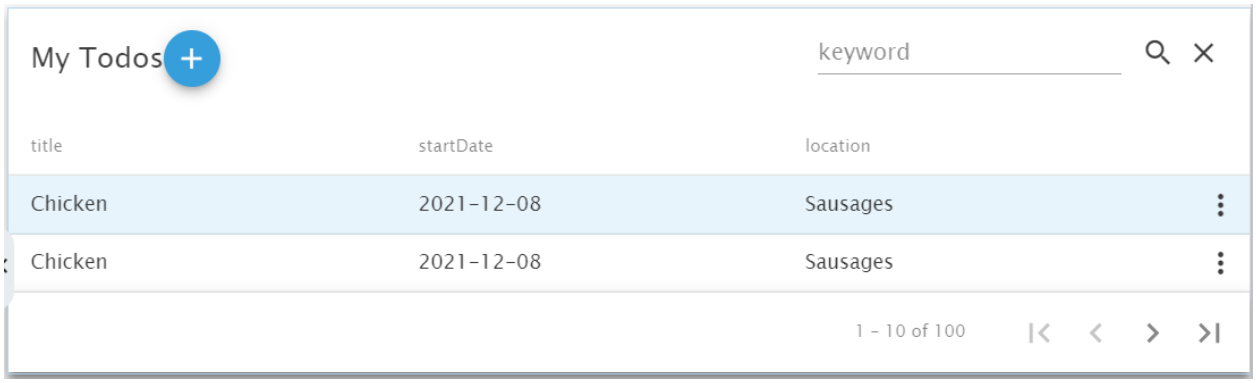


Slika 8 Glavni prozor WebRatio alata [68]

WebML, kreiran od strane grupe istraživača sa Univerziteta u Milanu, je jedan od prvih namenskih jezika za modelovanje web aplikacija koji je obuhvatao modelovanje svih bitnih aspekata kao što su: organizacija podataka, organizacija HTML stranica, modelovanje navigacije i podrška za personalizaciju. Od verzije 7 (u trenutku pisanja ove disertacije aktuelna je verzija 9), kao jezik se koristi IFML (In*teraction Flow Modelling Language*) [64] koji predstavlja nadogradnju WebML-s u kome je podržano i modelovanje korisničkog ponašanja prilikom korišćenja aplikacija. Ovde je takođe naglasak stavljen na web aplikacije, ali je podržano i modelovanje softverskih proizvoda koji rade na drugim platformama. Trenutno je WebRatio glavni alat koji podržava IFML, mada postoji nekoliko alata otvorenog koda koji ga takođe koriste. IFML je priznat kao standardni jezik za modelovanje korisničkog interfejsa od strane OMG grupe [165].

WebRatio se distribuira kao komercijalno rešenje namenjeno korišćenju od strane obučenih profesionalaca sa ciljem implementacije kompletnih sistema. Cena generisanja kompletnog sistema je strma kriva učenja usled složenosti jezika i pripadajućih alata.

Na slikama 8 i 9 je prikazan je glavni prizor WebRatio alata i primer jedne stranice generisane web aplikacije, respektivno.



Slika 9 Primer generisane web stranice korišćenjem WebRatio platforme

2.2.4.7. Marama AI

Marama AI [40] predstavlja alat za generisanje prototipova korisničkog interfejsa na osnovu tekstualne specifikacije korisničkih zahteva. Dati alat kroz tri transformacije, ručno pisanu specifikaciju zahteva prevodi do funkcionalnih UI prototipova.

U prvom koraku, na osnovu tekstualne specifikacije zahteva se generišu *Essential Use Case* (EUC) dijagrami. EUC predstavljaju polu-formalne modele koji se baziraju na dijagramima slučajeva korišćenja i koji mogu da se validiraju u odnosu na definisane EUC šablone.

U sledećem koraku predloženog procesa, EUC dijagrami se transformišu u modele korisničkog interfejsa nazvane *Essential User Interface* (EUI) modeli na osnovu kojih se generišu konkretni UI prototipovi koji služe za korisničku evaluaciju specificiranih zahteva.

Kako bi se na osnovu tekstualne specifikacije korisničkih zahteva dobili elementi spomenutih modela, svaki od zahteva se analizira na osnovu predefinisanog skupa pravila i šablona kako bi se utvrdilo koji UI elementi su potrebni kako bi se ovaj zahtev realizovao. Ovo pojednostavljuje implementaciju transformacionih mehanizama i ubrzava proces transformacije, ali mogućnosti ovog pristupa su ograničene sadržajem i kvalitetom skupa šablona. Osim toga, izvršivi prototipovi se mogu generisati tek nakon ručne dorade dobijenih EUC modela pomoću Marama AI razvojnog okruženja.

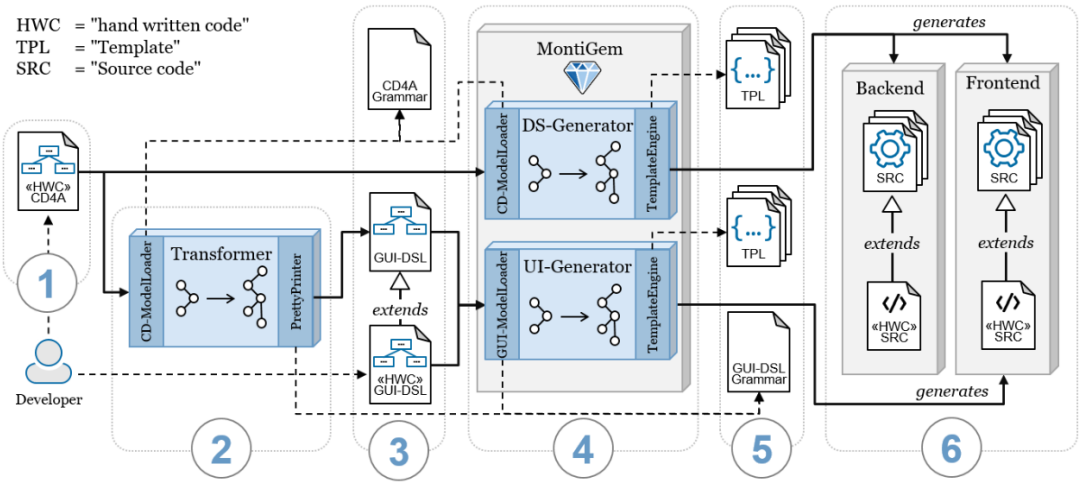
Dato okruženje poseduje grafički editor za prikaz EUC dijagrama putem kojeg se za svaki element može videti od kojeg tekstualnog korisničkog zahteva je potekao i upravljati skupom UI kontrola na koje će se zahtev mapirati u prototipu. Prototipovi su u vidu jednostavnih HTML formi putem kojih korisnici mogu manipulisati podacima u modelovanim entitetima. Nažalost, u publikaciji nije navedeno koji je stepen funkcionalnosti ovih prototipova postignut, kao ni na koji način je vršena evaluacija specificiranih korisničkih zahteva.

2.2.4.8. MontiGEM

U [185] je predstavljen alat pod nazivom MontiGEM za generisanje funkcionalnih prototipova poslovnih aplikacija, sa ciljem podrške za agilni evolutivni način razvoja. Kao polazna tačka za generisanje se koristi model podataka kreiran na bazi *Class Diagram for Analysis* (CD4A) DSL-a, razvijenog za potrebe datog alata. U [185] je navedeno da je moguće koristiti i druge vidove ulaznih modela, pod uslovom da se za njih obezbede odgovarajući generatori.

Na Slici 10 je prikazan proces rada i faze u okviru opisanog pristupa. Verzija predstavljena u ovoj publikaciji sadrži skup generatorskih komponenti koje na osnovu CD4A modela generišu programski kod koji realizuje poslovnu logiku i UI modele na osnovu kojih se dalje generiše implementacija korisničkog interfejsa. Razdvajanje programske logike od UI specifikacije omogućava laku zamenu nekog od delova generisane aplikacije bez uticaja na ostale, što pospešuje fleksibilnost celokupnog pristupa.

Pored navedenih modela, ulazne podatke za generatore čini i skup konfiguracionih datoteka na osnovu kojih se mogu specificirati dodatni detalji koji utiču na rezultat generisanja. Za rukovanje konfiguracionim parametrima je zadužen obučeni član razvojnog tima, s obzirom da su bazirani na DSL sintaksi. Generisano rešenje je lako proširivo ručno pisanim programskim kodom, kako bi se na osnovu prototipa dobila konačna verzija sistema.



Slika 10 MontiGEN pristup [185]

2.2.4.9. SocietySoft

Istraživanje prikazano u [186] je sprovedeno sa ciljem da se izbegne odbacivanje prototipova korisničkog interfejsa koji su na početku razvoja usaglašeni sa korisnicima i spreči ponavljanje već obavljenog posla fazi implementacije. Predstavljeno rešenje pod nazivom SocietySoft omogućava kreiranje direktne veze između prototipa i kasnijih faza softverskog proizvoda, generisanjem modela koji će služiti kao osnova za dalju implementaciju.

Prototipovi se kreiraju pomoću alata Microsoft PowerPoint i koriste za specifikaciju grafičkog rasporeda elemenata na formama i navigacije između formi. Na osnovu datih prototipa se generiše model korisničkog interfejsa koji se zatim transformiše u model korisničkih zahteva koji se može koristiti okviru alata NDT-Suite razvijenog od strane istih autora [186].

Validacija predloženog pristupa je izvedena u saradnji sa dve softverske kompanije. Obe kompanije su ocenile da je prezentovani pristup koristan, ali da odabir Microsoft PowerPoint alata za kreiranje skica nije odgovarajući za njih.

2.2.4.10. XIS-Web

U [189] se predlaže rešenje za automatizaciju razvoja modernih web aplikacija, koje se sastoji od domenski specifičnih jezika i integrisanog skupa alata čiji je zadatak generisanje UI modela na osnovu specifikacije korisničkih zahteva. Motivacija za implementaciju ovog rešenja je bila pojava sve većeg broja različitih tipova uređaja za koje je potrebno kreirati web aplikacije, što znatno utiče na povećanje kompleksnosti i cene razvoja ovog tipa softverskih proizvoda.

XIS-Web predstavlja web verziju XIS rešenja istih autora predstavljenog u [189]. U skladu sa tim, DSL razvijen za potrebe XIS-Web platforme se bazira na konceptima jezika opisanih u [189] koji podržava razvoj dve grupe modela: *Entity Views*, koji služe za specifikaciju poslovne logike i *User-Interface Views*, putem kojih se definiše korisnički interfejs web aplikacije. Svaka od spomenutih grupa se sastoji od nekoliko tipova pod-modela koji se bave specifičnim aspektima razvijanog sistema.

XIS-Web pruža dva pristupa modelovanju: osnovni (u publikaciji označen kao *dummy*) i pametni (*smart*) režim. Korišćenjem osnovnog režima, osoba zadužena za modelovanje treba da kreira sve modele potrebne za uspešno obavljanje transformacija. Ovaj režim je namenjen osobama koje žele da imaju pristup svim mogućnostima jezika sa ciljem detaljne specifikacije svih aspekata modelovane aplikacije. Sa druge strane, korišćenjem pametnog režima moguće je na osnovu ručno kreiranih modela iz grupe *Entity Views* automatski generisati prezentacione modele

koji spadaju u *User-Interface Views* grupu. Smanjivanjem broja modela koje je potrebno ručno kreirati se skraćuje vreme potrebno za generisanje programskog koda.

U oba režima XIS-Web obezbeđuje razvojno okruženje u vidu grafičkog editora koji je razvijen kao MDG[7] (*Model Driven Generation*) dodatak u okviru Enterprise Architect[8] platforme za UML modelovanje. M2M transformacije su realizovane na bazi Enterprise Architect transformacione platforme, dok se za M2T generisanje koriste tekstualni šabloni kreirani uz pomoć Eclipse Acceleo[9] platforme.

Pored opisa predloženog pristupa i pripadajućih alata, u publikaciji je predstavljena i evaluaciona studija koja će biti opisana u nastavku.

2.2.4.11. Application Specification Language

Application Specification Language (ASL) [187] predstavlja jezik specifičan za domen koji omogućava specifikaciju korisničkih zahteva softverskog proizvoda. Ovaj rad je nastavak istraživanja iz [188].

ASL spada u grupu kontrolisanih prirodnih jezika, što znači da koristi iskaze bliske prirodnim ljudskim jezicima ali koji moraju da poštuju stroga formalna pravila. Ovi jezici, održavanjem pogodnog odnosa između formalnosti i čitljivosti, su prilagođeni korišćenju od strane ljudi (razvojnog tima i krajnjih korisnika) i mašina (različite parserske ili generatorske softverske komponente).

ASL se koristi za specifikaciju zahteva vezanih za korisnički interfejs aplikacije na osnovu koje se može transformisati u softverske modele na različitim nivoima apstrakcije kao i na programski kod.

ASL je nastao kao kombinacija dva jezika: ITLingo RSL[10] (*Requirements and Tests*. *Specification Language*) i IFML. Osnovi pojam, preuzet iz RSL jezika, je `DataEntity` koji služi za definisanje domenskih koncepata ili informacionih entiteta kao što su poslovna dobra, zaposleni ili poslovne transakcije. `DataEntity` predstavlja poseban strukturni entitet koji uključuje specifične atribute, strane ključeve i ograničenja nad podacima. `DataEntity` instance se grupišu u klastere (entitet tipa `DataEntityCluster`). Pored ovih statičkih tipova entiteta, ASL omogućava definisanje slučajeva korišćenja i učesnika koji služe za specifikaciju interakcija unutar sistema.

Predloženi pristup je prikazan na Slici 11. Proces započinje specifikacijom sloja podataka definisanjem navedenih ASL izraza. Nakon specifikacije, moguće je uz pomoć ugrađenih alata validirati kreirani ASL model i, ukoliko je sve u redu, generisati ASL UI specifikaciju koja predstavlja varijantu ASL jezika specijalizovanu za opis korisničkog interfejsa.

Ova specifikacija je otvorena za ručne izmene od strane članova razvojnog tima. Nakon svake ručne izmene, proces validacije i generisanja nove UI specifikacije se ponavlja. Na osnovu UI modela se dalje može generisati programski kod za željenu ciljnu platformu. U procesu se koriste i M2M i M2T transformacije kako bi se na osnovu početne specifikacije dobio upotrebljiv programski kod.

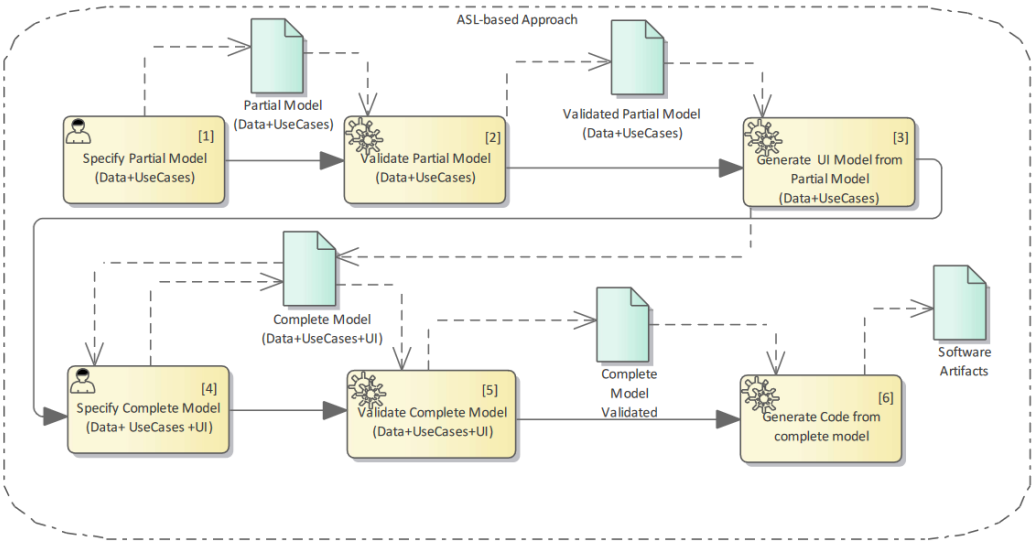
---

[7] https://sparxsystems.com/resources/mdg_tech

[8] https://sparxsystems.com/

[9] https://www.eclipse.org/acceleo/

[10] http://itbox.inesc-id.pt/ITLingo/RSL



Slika 11 ASL pristup specifikaciji korisničkih zahteva [187]

Kao primer primene ovog koncepta, implementirana je generatorska komponenta koja na osnovu ASL UI specifikacije generiše Django administratorski web panel[11].

2.2.4.12. E-WAE

U [194] se predlaže proširenje UML-a pod nazivom *Enterprise Web Application Extension* (E-WAE) koje omogućuje modelovanje prezentacionog sloja i korisničkih prava u okviru poslovnih web aplikacija. E-WAE se sastoji od 22 stereotipa koji su namenjeni za modelovanje prezentacionog sloja. Za modelovanje ostalih delova web aplikacije predlaže se upotreba standardnog UML-a. E-WAE projektantima omogućava rad u poznatom okruženju, ali i mogućnost korišćenja koncepata na višem nivou apstrakcije, kako bi se modeli mogli lakše iskomunicirati sa krajnjim korisnicima.

Polazni PIM, proširen E-WAE stereotipima, se može transformisati u jedan od dva dostupna platformska modela. Podržano je generisanje modela za Java JSF [12]i za ASP.NET MVC [13] radne okvire. Ovi modeli na nižem nivou apstrakcije omogućavaju detaljniji uvid u aspekte implementacije. Na osnovu PSM-ova je moguće generisati programski kod za navedene platforme. Generisane web aplikacije mogu poslužiti kao prototipovi za brzu evaluaciju od strane korisnika.

Upotreba ovog pristupa demonstrirana je na tri primera projektovanja i implementacije realnih web aplikacija.

U prvom primeru kreirana je aplikacija po uzoru na sistem kupovine artikala na web stranici firme Amazon. Tu je prezentiran celokupni E-WAE pristup, počevši sa kreiranjem PIM modela, preko generisanja platformskog modela za obe podržane tehnologije i na kraju prototipa web aplikacije. Generisane web stranice sadrže samo osnovnu HTML strukturu, bez uključivanja dodatnih tehnologija kao što su CSS i Javascript. Ručna intervencija je neophodna kako bi se omogućile određene navigacione funkcije.

Drugi primer je baziran na edukativnoj web aplikaciji, gde je demonstrirano modelovanje korisničkih prava pristupa po RBAC (*Role Based Access Control*) [195] modelu.

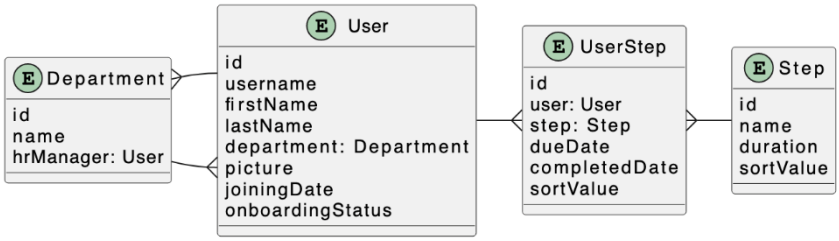
[11] https://docs.djangoproject.com/en/4.2/ref/contrib/admin/
[12] https://www.oracle.com/java/technologies/javaserverfaces.html
[13] https://dotnet.microsoft.com/en-us/apps/aspnet/mvc

U poslednjem primeru u kojem su demonstrirane mogućnosti modelovanja navigacione strukture web aplikacije, kreiran je sistem katalogizacije i pretrage baziran na web stranici Kongresne biblioteke Sjedinjenih Američkih Država.

2.2.4.13. Jmix

Jmix[14] (ranije objavljivan kao *CUBA Platform*) je skup alata za razvoj web poslovnih aplikacija upravljan modelima. Sastoji se od razvojnog okruženja (Jmix Studio) baziranog na IntelliJ IDEA[15] platformi. Web aplikacije se generišu nad Spring Boot [16] web razvojnim okvirom. Izvorni kod platforme je dostupan na GitHub repozitorijumu[17].

Kreiranje aplikacije upotrebom Jmix platforme započinje specifikacijom modela podataka (primer je prikazan na Slici 12).



Slika 12 Primer Jmix modela podataka

Korišćenje Jmix platforme i njenog potpunog skupa mogućnosti nije besplatno. Besplatni paket uključuje licencu za jednog inženjera i obuhvata osnovno razvojno okruženje i mogućnost generisanja aplikacija. Za pristup dodatnim mogućnostima platforme, kao što su podrška za različite DBMS sisteme, alate za modelovanje prava pristupa ili poslovnih procesa, potrebno je kupiti jedan od dva dostupna (RAD ili Enterprise) komercijalna paketa. Određeni dodaci iz galerije (interaktivni grafici, mape, mogućnost slanje email poruka itd.) se takođe naplaćuju.

U odnosu na Kroki, ova platforma nije predviđena za kolaborativni razvoj sa korisnicima u ranim fazama softverskog projekta. Korisnici mogu evaluirati prototip tek kada je završeno generisanje koda.

2.2.4.14. Ostali alati

Pored rešenja obrađenih u prethodnim sekcijama, za potrebe doktorske disertacije su analizirani i stariji radovi [40, 42 - 45] u kojima su predstavljene različite varijacije uključivanja modela i skica korisničkog interfejsa kao podloge za generisanje određenog dela programskog koda. Nijedan od pomenutih radova ne koristi skice kao deo konkretne sintakse jezika na kojem je bazirano modelovanje, već se skice importuju, obrađuju i prevode na model primenom različitih heuristika, što produžava proces i pogoduje nastanku grešaka.

2.3 Evaluacija metoda i tehnologija

U nastavku ovog poglavlja biće prikazani modeli i tehnike za evaluaciju tehnologija ili metoda, koji su poslužili kao podloga ili inspiracija za kreiranje eksperimenta koji je opisan u petom poglavlju.

[14] https://www.jmix.io/
[15] https://www.jetbrains.com/idea/
[16] https://spring.io/projects/spring-boot
[17] https://github.com/jmix-framework/jmix

2.3.1 Technology Acceptance Model i Method Evaluation Model

Problem evaluacije prihvatanja određene tehnologije ili metode već decenijama predstavlja značajan predmet interesovanja naučnika iz raznih oblasti. Što se tiče informacionih tehnologija, vrlo rano je uočeno kako potencijalne prednosti korišćenja određenog sistema ne dolaze do izražaja ukoliko se on ne koristi dovoljno u praksi [47]. Na osnovu ovog zapažanja, predloženo je nekoliko modela za evaluaciju praktičnog prihvatanja tehnoloških rešenja. Jedan od najranijih i najprihvaćenijih je postao *Technology Acceptance Model* – TAM, predstavljen u [46] 1989. godine. Prema TAM-u, namera korišćenja određene tehnologije (*Intention to Use*) direktno zavisi od dva subjektivna parametra:

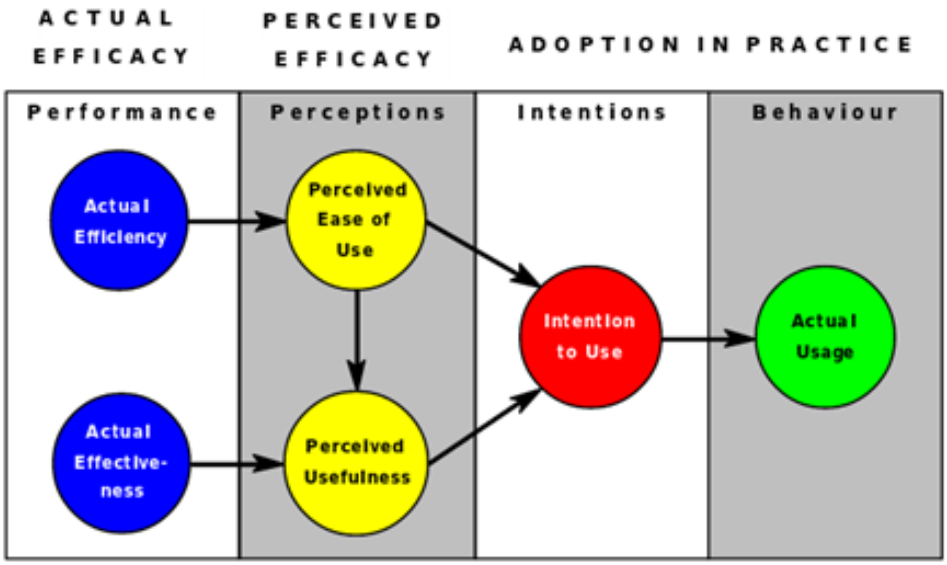
1. Procenjena korisnost - mera u kojoj osoba veruje da će korišćenje posmatranog sistema poboljšati njene performanse na poslu.
2. Procenjena lakoća korišćenja - mera u kojoj osoba smatra da će upotreba posmatranog sistema biti bez napora.

Prema ovom modelu, evaluacija određene tehnologije se sprovodi kreiranjem upitnika čija pitanja su kreirana tako da učesnici mogu da izraze svoj stepen slaganja sa iskazima koji testiraju ova dva parametra, na osnovu čega se kreira ocena uspeha korišćenja posmatranog sistema.

Jedan od široko prihvaćenih teorijskih modela baziranih na TAM-u je *Method Evaluation Model* - MEM, prezentovan u [47]. Osnovna premisa MEM-a je da se validnost određenog metoda može odrediti jedino njegovom primenom u praksi.

U skladu sa tim, MEM proširuje elemente TAM-a. Slika 13 predstavlja konceptualni pregled MEM elemenata prihvatanja tehnologija. Sa slike je vidljivo da realna efikasnost i efektivnost metoda utiču na procenjenu lakoću korišćenja i procenjenu korisnost, što dovodi do namere za korišćenjem metoda i na kraju do realnog korišćenja u praksi.

Ovaj model jasno razdvaja koncept namere korišćenja (*Intention to Use*) na kojoj se TAM zaustavlja i stvarnog korišćenja metoda (*Actual Usage*).



Slika 13 Osnovni koncepti MEM-a [47]

U [47] se, pored predstavljanja MEM-a, nalazi i detaljan opis eksperimenta za evaluaciju tehnologije po opisanom modelu, koji je poslužio kao polazna tačka za mnoga slična istraživanja [52]. Dati eksperiment je bio inspiracija i za naš eksperiment prezentovan u petom poglavlju.

2.3.2. Okvir za kvalitativnu evaluaciju DSL-ova: FQUAD

S obzirom da se tehnički deo istraživanja bavi razvojem EUIS DSL-a i njegove konkretne sintakse koja treba da bude pogodna za skiciranje formi zajedno sa korisnicima, u nastavku je prikazan okvir za kvalitativnu analizu jezika specifičnih za domen (*Framework for qualitative assessment of domain-specific languages* - FQUAD) prezentovan u [53], čiji delovi su korišćeni pri projektovanju i evaluaciji EUIS DSL-a i njegovih pripadajućih editora. Pored njega, postoje i stariji okviri za testiranje upotrebljivosti DSL-ova prezentovani u [119] i [120].

FQUAD radni okvir definiše metrike i parametre na osnovu kojih se može izvršiti formalna evaluacija kvaliteta posmatranog jezika specifičnog za domen. Koristeći smernice definisane u [121] i karakteristike kvalitetnog softvera definisane standardom ISO/IEC 25010:2011 [122], FQUAD definiše 10 karakteristika i 26 pod-karakteristika koje je potrebno evaluirati kako bi se procenila uspešnost implementacije određenog DSL-a. Navedene karakteristike se nalaze u Tabeli 1, u prevodu autora disertacije. Originalni nazivi se mogu naći u [53].

Tabela 1 Kvalitativne karakteristike DSL jezika definisane FQUAD okvirom.

Karakteristika	Pod-karakteristike	Opis
Funkcionalna pogodnost	Opisuje u kojoj meri DSL podržava razvoj rešenja koja zadovoljavaju navedene potrebe aplikativnog domena.	
	Kompletnost	Svi koncepti i scenariji iz domena se mogu predstaviti DSL-om.
	Prikladnost	DSL je prikladan za upotrebu unutar domena.
Upotrebljivost	Upotrebljivost DSL-a je mera u kojoj se jezik može koristiti radi ostvarenja željenih ciljeva.	
	Razumljivost	Elementi DSL-a su razumljivi (mogu se razumeti nakon čitanja opisa i uputstava koji dolaze sa jezikom).
	Lakoća učenja	Koncepti i simboli koji se koriste u jeziku se lako uče i pamte.
	Broj potrebnih aktivnosti po zadatku	Jezik pomaže korisnicima da obave svoje zadatke uz minimalan broj potrebnih aktivnosti.
	Korisnička percepcija dopadljivosti	Korisnici mogu da prepoznaju da li je DSL prikladan za njihove potrebe.
	Operabilnost	Jezik sadrži elemente koji omogućavaju laku upotrebu i kontrolu nad samim DSL-om (elementi se mogu lako odabrati i koristiti, akcije se mogu poništavati, poruke o greškama sadrže opise procedura za oporavak, itd.).
	Vizuelna dopadljivost	Elementi jezika su vizuelno dopadljivi.
	Kompaktnost	Odštampana verzija programa (modela) razvijenog upotrebom DSL-a ne zauzima više od jedne stranice.
Pouzdanost	Pouzdanost DSL-a je svojstvo jezika da pomaže u kreiranju pouzdanih programa (modela): poseduje funkcije za proveru modela, sprečava kreiranje nepredviđenih scenarija, itd.	
	Provera modela	DSL sprečava projektante da prave greške.
	Tačnost	Jezik definiše ispravne odnose između elemenata modela.
	Stepen u kojem jezik promoviše lakoću njegovog održavanja.	

Mogućnost održavanja	Mogućnost izmene	DSL je dizajniran tako da ga je moguće proširiti dodatnim funkcionalnostima bez degradiranja postojećih.
	Minimalna zavisnost komponenti	DSL se sastoji od nezavisnih komponenti tako da promena u jednoj ima minimalan uticaj na druge komponente.
Produktivnost	Produktivnost je karakteristika vezana za količinu resursa koji su potrebni projektantu kako bi obavio željene zadatke upotrebom jezika.	
	Vreme razvoja	Vreme provedeno u razvoju programa koji zadovoljava datu specifikaciju je umanjeno korišćenjem jezika.
	Količina ljudskih resursa	Količina ljudskih resursa potrebnih da se razvije željeni program upotrebom jezika je smanjena,
Proširivost	Mera posedovanja mehanizama putem kojih korisnici mogu da dodaju nove funkcionalnosti u jezik.	
	Mehanizmi za dodavanje novih funkcionalnosti	DSL poseduje mehanizme putem kojih korisnici mogu dodati nove funkcionalnosti u jezik.
Kompatibilnost	Stepen kompatibilnosti DSL-a sa domenom problema i procesom razvoja.	
	Kompatibilnost sa domenom	DSL je kompatibilan sa domenom problema. DSL može da sarađuje sa ostalim elementima iz problemskog domena bez dodatnih modifikacija.
	Kompatibilnost sa procesom razvoja	Kreiranje modela uz pomoć DSL-a se uklapa u postojeći proces razvoja.
Ekspresivnost	U kojoj meri se strategija rešavanja problema može prirodno predstaviti programom.	
	Mapiranje ideja u program	Strategija rešavanja problema se može predstaviti u jeziku.
	Jedinstvenost	DSL omogućava tačno jedan dobar način za predstavljanje svakog odabranog koncepta.
	Ortogonalnost	Svaki element u sklopu DSL-a predstavlja tačno jedan koncept iz problemskog domena.
	Predstavljanje bitnih domenskih koncepata	Elementi DSL-a odgovaraju bitnim konceptima iz domena problema. DSL ne sadrži domenske koncepte koji nisu bitni.
	Konfliktni elementi	DSL ne sadrži konfliktne elemente.
	Odgovarajući nivo apstrakcije	Nivo apstrakcije DSL-a je takav da on nije komplikovaniji ili ne sadrži više detalja nego što je potrebno.
Ponovna upotrebljivost	Stepen ponovne upotrebljivosti jezičkih iskaza u drugim jezicima.	
	Simboli i ostali elementi DSL-a se mogu upotrebiti u više od jednog DSL-a ili u građenju drugih elemenata istog jezika.	
Lakoća integrisanja	DSL se može integrisati sa ostalim jezicima koji se koriste u procesu razvoja.	

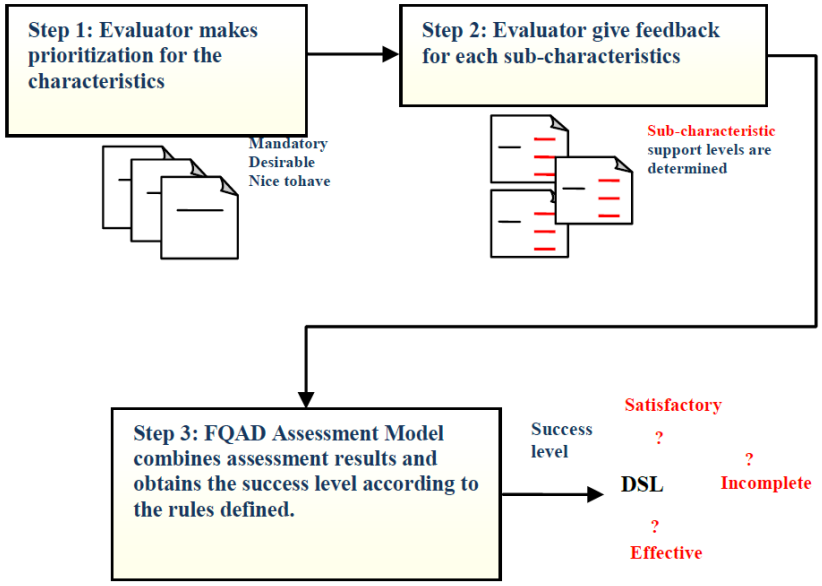
Kako bi se uspeh određenog DSL-a evaluirao uz pomoć ovoga okvira, njegovi autori predlažu evaluacioni proces koji se odvija u 3 faze:

1. **Faza 1** – Osoba koja evaluira DSL u ovoj fazi ima zadatak da, putem upitnika, grupiše funkcionalne karakteristike jezika iz Tabele 1 prema sledećim kategorijama:
    a. Obavezne (*Mandatory*),
    b. Poželjne (*Desirable*),
    c. Dobrodošle (*Nice to have*).

Grupisanje karakteristika po važnosti omogućava kreiranje profila svakog od evaluatora ponaosob, na osnovu kojeg se može steći uvid u to šta on očekuje od posmatranog DSL-a i kakva je generalna percepcija važnosti navedenih karakteristika među korisnicima jezika.

2. **Faza 2** – Osoba koja evaluira daje odgovore na pitanja koja se tiču svake od funkcionalnih karakteristika ponaosob, kako bi se odredilo u kojoj meri je svaka od njih podržana u jeziku. Na osnovu ovoga se definiše stepen podržanosti za svaku od karakteristika i on može biti:
    a. Ne podržava
    b. Podržava u manjoj meri
    c. Podržava u većoj meri
    d. Potpuno podržava
3. **Faza 3** – U ovoj fazi se, na osnovu odgovora ispitanika, formira ocena posmatranog DSL-a, koja može biti:
    a. **Nepotpun** (*Incomplete*) – DSL u nedovoljnoj meri zadovoljava namenjenu svrhu i treba da se unapredi,
    b. **Zadovoljavajući** (*Satisfactory*) – DSL uglavnom zadovoljava namenjenu svrhu ali može biti unapređen,
    c. **Efektivan** (*Effective*) – DSL zadovoljava namenjenu svrhu.

Na Slici 14 je dat grafički prikaz procesa evaluacije DSL-a na osnovu FQUAD modela:



Slika 14 Model evaluacije DSL-a na osnovu FQUAD okvira [53]

2.3.3 Studija slučaja

Studija slučaja je kvalitativna istraživačka metoda kojom se određeni procesi ili pojave ispituju u njihovom stvarnom okruženju [76], kako bi se stekao detaljan uvid u predmet istraživanja i mehanizme koji pokreću uzročno-posledične veze unutar njega [75, 123]. S obzirom na naglasak koji se u ovakvim istraživanjima postavlja na kontekst i na konkretne posmatrane situacije, studije slučaja su korisne za istraživanje specifičnih situacija i oblasti, ali se ne primenjuju na istraživanje nekih globalnih fenomena. Osnovna podela studija slučaja je na:

1. Eksplorativne studije slučaja – vrše se u početnim fazama nekog istraživanja kao baza za formiranje novih hipoteza i teorija o predmetu posmatranja.
2. Konfirmatorne studije slučaja – služe za testiranje postojećih teorija.

Prvi korak u sprovođenju studije slučaja predstavlja definisanje predloga studije (*Study proposition*) koji jasno opisuje šta studija treba da pokaže. Na osnovu definisanog predloga studije, potrebno je odabrati predmet posmatranja koji se u studijama slučaja naziva *jedinica analize.* Preduslov za sprovođenje studije slučaja je dobro definisana ciljna jedinica analize.

Kao istraživačka metoda, studija slučaja vuče poreklo iz psihoanalize gde su, u sklopu studija slučaja, posmatrani pojedinačni pacijenti. U zavisnosti od prirode istraživanja, ovo se može promeniti, pa predmet posmatranja mogu biti i grupe ljudi ili neke pojave koje nisu osobe [123].

U softverskoj industriji jedinica analize može biti pojedinačni softverski inženjer, krajnji korisnik, kompanija, razvojni tim, određena epizoda ili događaj iz procesa razvoja ili neki softverski proizvod [75]. Za razliku od statističkih eksperimenata ili nekih drugih kvantitativnih (ali i kvalitativnih) metoda, predmeti posmatranja se u studijama slučaja ne biraju nasumično već vrlo ciljano [75], pri čemu je dovoljan vrlo mali broj (često i samo jedan) dobro definisanih i odabranih slučajeva kako bi studija bila uspešna. Kao sredstva prikupljanja podataka u studijama slučaja se koriste pisane napomene istraživača, audio i video snimci, kao i podaci dobijeni direktno od posmatranih učesnika putem intervjua i upitnika.

Osnovni nedostatak studija slučaja je podložnost rezultata interpretacijama, kao i uticaj pristrasnosti istraživača koja može da utiče na njegove utiske koji su u ovakvim istraživanja vrlo bitni [75]. Kako bi se ovo prevazišlo, potrebno je ustanoviti dobro definisan okvir za odabir slučajeva i prikupljanje i analizu podataka. Pored ovoga, veći broj slučajeva korišćenih u jednoj studiji slučaja ili ponovljene studije slučaja mogu doprineti validnosti studije, pogotovo kada je reč o konfirmatornim studijama.

2.3.4 Evaluacioni eksperimenti

U nastavku su prezentovani načini evaluacije nekih od alata analiziranih u sekciji 2.2.4. radi mogućnosti poređenja sa eksperimentom koji je prezentovan u petom poglavlju.

2.3.4.1 MockupDD evaluacija

Publikacija u kojoj je predstavljan MockupDD proces [20] sadrži detaljan opis eksperimenta koji za cilj ima validaciju MockupDD-a u odnosu na tradicionalne pristupe razvoju web aplikacija baziranih na modelima. Eksperiment je dizajniran koristeći GQM (Goal-Question Metric) model [137]. GQM predstavlja metodu evaluacije softverskih proizvoda u okviru koje je potrebno definisati cilj projekta, zatim taj cilj pretočiti u skup pitanja čiji odgovori će omogućiti procenu (uz definisanje odgovarajućih metrika) u kojoj meri proizvod odgovara svom cilju. U skladu sa ovim, autori su cilj svoje studije definisali na sledeći način:

- **Analizirati:** Razvijenu web aplikaciju i proces razvoja
- **Kako bi se**: Razumeo MockupDD pristup
- **Sa stanovišta**: Efikasnosti i efektivnosti
- **Iz ugla**: Razvojnog ima
- **U kontekstu**: razvoja web aplikacije sa simuliranim klijentima i korisnicima

U odnosu na ovaj cilj, prateći GQM model, definisana su sledeća pitanja:

1. Da li MockupDD oznake sadrže semantiku potrebnu za efektivno predstavljanje i implementaciju web aplikacija koje rukuju velikim količinama podataka?
2. Da li je generisanje MDWE modela uz pomoć MockupDD procesa efikasnije (zahteva manje napora i daje manju mogućnost greške) u odnosu na kreiranje tih modela koristeći standardne MDWE jezike i alate?

Metrike koje su definisane prema ovim pitanjima su usmerene na merenje efikasnosti MockupDD pristupa u odnosu na modelovanje korišćenjem WebML jezika. Cilj je istražiti da li MockupDD modeli pružaju isti nivo ekspresivnosti, ali uz manji napor potreban da se dođe do modela. Kako bi se ovo ispitalo, definisane su sledeće metrike:

1. **Stepen poklapanja (CR[18])** – Predstavlja postotak elemenata WebML modela koji se mogu generisati na osnovu MockupDD oznaka, kako bi se dobila željena semantika. Ova metrika predstavlja meru efektivnosti procesa modelovanja.
2. **Vreme provedeno ispravljajući greške (TSCE[19])** – Greške predstavljaju određene situacije u kojima sučesnici nisi uspešno obavili modelovanje, zbog čega je potrebna nova specifikacija. Do ove mere se dolazi analizom modela generisanih MockupDD procesom, u odnosu na ručno kreirane modele sa ciljem provere grešaka (kao što su nedostajući elementi i/ili semantičke greške), kako bi se utvrdilo koji pristup je imao bolje performanse sa manjim brojem grešaka. Ova metrika predstavlja meru efikasnosti procesa modelovanja.
3. **Stepen kompletnosti u odnosu na srednje vreme provedeno na zadacima (CRMT[20])** – Ova metrika predstavlja odnos vremena potrebnog za modelovanje iste aplikacije u oba posmatrana pristupa i takođe daje meru efikasnosti procesa modelovanja.

Učesnici u eksperimentu su bili studenti završnih godina računarstva koji su upoznati sa tehnikama razvoja web aplikacija kao i sa WebML-om. Podeljeni su u 10 tročlanih timova, pri čemu je pola timova koristilo MockupDD (eksperimentalna grupa), dok je druga polovina (kontrolna grupa) koristila WebML korišćenjem alata WebRatio [68]. Ulogu korisnika i klijenata su imali zaposleni na univerzitetu, od kojih su neki i autori navedene studije i MockupDD procesa. Eksperimentalne grupe su koristile MockupDD ukomponovan u agilni proces koji je trajao tri nedelje, podeljen u jednonedeljne Scrum sprintove, pri čemu je u svakom sprintu jedan deo razvijenog modela unapređivan i proširivan MockupDD oznakama. Kontrolne grupe su koristile WebRatio bez određenog metodološkog procesa.

Vrlo detaljna analiza procesa, rezultata kao i diskusija o samoj studiji su dati u [20]. Ovde ćemo se samo ukratko osvrnuti na dobijene rezultate. Kako bi se dobila vrednost za CR metriku, izračunat je stepen poklapanja za svaki razvijeni model i zatim je nađena srednja vrednost. Rezultati su pokazali da je prosečna vrednost CR metrike za WebML modele između 5.44% i 11.14%. S obzirom na ove brojke i na činjenicu da se delovi koji nisu podržani WebML-om mogu modelovati pomoću MockupDD tagova, autori su zaključili da se može pozitivno odgovoriti na prvo pitanje.

Kako bi proverili drugi kriterijum, autori su dodelili određenu vrednost svim vrstama grešaka koje su učesnici pravili tokom modelovanja. Ova vrednost odgovara vremenu provedenom

[18] Coverage Ratio

[19] Time Spent on Correcting Errors

[20] Completion Rate/Mean Time-on-task

na ispravljanje greške i spisak predstavlja meru semantičke podudarnosti generisanih modela u odnosu na ručno kreirane. Koristeći ovu metriku, model koji savršeno odgovara onome što autori nazivaju idealan WebML model, imaće vrednost jednaku nuli, što znači da nije potrebno provesti dodatno vreme kako bi se ispravile greške na njemu. „Idealan model“ su pažljivo razvili istraživači koji su radili na datoj publikaciji. Primenom navedenih vrednosti i korišćenjem Cliff's delta metrike [138], autori su došli do podataka o tome da je MockupDD proces malo bolji od standardnog načina modelovanja, kada je vreme provedeno na ispravljanje grešaka u pitanju.

Na osnovu ovoga rezultata autori zaključuju da MockupDD proces ne uvodi nove greške u odnosu na tradicionalne pristupe modelovanju i predstavlja (u nekoj meri) efikasniji način modelovanja. Pored ovoga, autori uočavaju da su korisnici iz kontrolne grupe bili skloniji pravljenju grešaka sa većom dodeljenom vrednošću, odnosno grešaka za čije ispravljanje je potrebno više vremena. Prilikom merenja poslednje metrike (CRMT), računato je vreme provedeno na svakom koraku prilikom modelovanja od strane svakog učesnika. Za MockupDD pristup ovi koraci su inicijalne instrukcije i dodavanje oznaka na model, dok je WebML pristup imao samo jedan korak a to je modelovanje. Koristeći istu statističku metodu kao i za TSCE metriku, autori su došli do saznanja da je za modelovanje putem MockupDD procesa potrebno značajno manje vremena. Uzimajući u obzir pozitivne zaključke za metrike TSCE i CRMT, autori smatraju da se i na drugo istraživačko pitanje može dati pozitivan odgovor i tako konačno zaključuju da je MockupDD pristup brži i manje sklon greškama u modelovanju u odnosu na tradicionalne MDWE procese.

Bitno je osvrnuti se i na potencijalne nedostatke i ograničenja koji mogu ugroziti validnost istraživanja:

1. Mali uzorak modelovanih aplikacija – u istraživanju je modelovana samo jedna web aplikacija što se može smatrati malim uzorkom za izvlačenje generalizovanih rezultata. U vezi sa ovim autori navode kako su pažljivo odabrali model aplikacije tako da ona pokrije što je više moguće standardnih funkcionalnosti i ponašanja koje se sreću u web aplikacijama.
2. Iskustvo učesnika u industriji – uključivanje učesnika koji imaju iskustva na radu u industrijskim projektima, umesto studenata, bi omogućilo zrelije razumevanja prednosti MockupDD pristupa u odnosu na tradicionalne. Autori navode kako su, da bi umanjili uticaj neiskustva učesnika, birali studente koji su učestvovali u razvoju barem dve web aplikacije u toku studija.
3. Poznavanje domena – učesnici u eksperimentu mogu imati inicijalno poznavanje domena i poslovne logike koje može uticati na modelovanje. Uzimajući ovo u obzir, autori navode kako su birali učesnike koji nisu imali dobro poznavanje domena modelovane aplikacije na osnovu intervjua.
4. Greške u kontrolnom WebML modelu – s obzirom da je kontrolni model ključan za izračunavanje postavljenih metrika, eventualne greške u njemu bi znatno uticale na validnost studije. Kao što je navedeno ranije u tekstu, ovaj model je iz tog razloga pažljivo razvijen od strane stručnjaka sa minimalno pet godina iskustva u WebML jeziku i MDWE polju.
5. Procena vrednosti grešaka – vrednosti koje su dodeljene greškama prilikom računanja druge metrike su određene od strane autora ovog eksperimenta i one su konstruisane tako da se uzme u obzir vreme koje je potrebno za ispravljanje grešaka u ručnom načinu rada, u odnosu na generisanje pomoću MockupDD procesa.

2.3.4.2. FQUAD studije slučaja

Kako bi validirali i unapredili svoje rešenje, autori FQUAD okvira su u [53] sproveli i predstavili dve studije slučaja u kojima su autori različitih jezika specifičnih za domen evaluirali svoje jezike koristeći FQUAD metod. Ove studije slučaja predstavljaju korisne konkretne primere evaluacije DSL-ova koji su poslužili prilikom dizajna eksperimenta koji je deo ove disertacije.

Prvi primer predstavlja eksplorativnu studiju slučaja čiji je cilj finalizacija liste karakteristika DSL-a predloženog FQUAD okvirom na primeru jednog realnog jezika. Nakon ovoga sprovedena je validaciona studija čiji je zadatak bio testiranje finalnog oblika FQUAD okvira na jeziku koji se koristi na jednom velikom softverskom projektu. Jezici testirani u ovim studijama su kreirani i koriste se u sklopu iste kompanije, ali autori navode da se radi o dva vrlo različita jezika koja se koriste od strane različitih, fizički udaljenih odeljenja.

Učesnici u eksperimentu su bili članovi tima zaduženog za kreiranje evaluiranih jezika, odabrani tako da predstavljaju različite pozicije u sklopu tih timova (menadžeri, programeri, domenski eksperti i slično), pri čemu je zadatak svakog od njih evaluacija jezika sa stanovišta njihove pozicije u timu. Nakon sprovedenih studija, učesnici su ispunjavali tekstualne upitnike putem kojih su ocenjivali FQUAD okvir. U daljem tekstu će biti predstavljene ove studije.

Eksplorativna studija

U ovoj studiji evaluiran je DSL kompanije koja se bavi razvojem radarskog softvera za vojnu industriju. Autori navode kako razvojni tim ove kompanije koristi agilne metodologije razvoja, uz oslonac na razvoj upravljan modelima i ponovnu upotrebljivost softverskih komponenti. Posmatrani jezik specifičan za domen je razvijen 2011. godine (4 godine pre objavljivanja razmatrane studije) i redovno se unapređuje u vidu novih verzija koje izlaze svake godine.

Cilj jezika je podrška brzom razvoju i evoluciji namenskih softverskih modula, nazvanih MDF (*Mission Data File*), koji se generišu na osnovu konceptualnih modela. Pored MDF modula, generiše se i programski API sloj koji omogućava korisnicima sistema da vrše upite nad uskladištenim podacima. U studiji je učestvovalo pet članova tima koji je razvijao DSL, od čega je jedan imao menadžersku ulogu, dva su bili domenski eksperti iz oblasti razvoja jezika, a preostala dva su koristili jezik za razvoj.

Konfirmatorna (validaciona) studija

DSL posmatran u ovoj studiji razvijen je i koristi se u sklopu iste kompanije, tako da su svi detalji vezani za kompaniju, timove i način rada istovetni onima u prvoj studiji. Ovaj jezik se koristi sa ciljem podrške brzom razvoju softverskih modula koji se koriste u sistemima za kontrolu paljbe u oružanoj industriji. Kao primer koncepata iz domena ovog jezika navode se: senzori za brzinu vetra, GPS uređaji i antene kao i njihovi namenski konektori. Zadatak generisanih modula je omogućavanje integracije različitih fizičkih platformi, kao što su ratna vozila i oružje koje se može na njih ugraditi.

U ovoj studiji učestvovala su tri člana DSL tima. Jedan je bio menadžer, dok su preostala dva člana delila poslove implementiranja i korišćenja jezika. Zadatak ove studije je validacija FQUAD pristupa nakon prikupljanja podataka i rafinisanje kriterijuma koji su objavljeni u sklopu eksplorativne studije.

Sakupljenje i validacija podataka

Kao tehnike sakupljanja podataka u predstavljenim studijama su korišćeni intervjui i upitnici. Na početku studije održan je sastanak sa svih učesnicima, na kojem su im predstavljeni detalji svakog koraka u sklopu studije, kao i njihovi zadaci. Nakon toga, obavljeni su intervjui sa

svakim učesnikom ponaosob. Na ovim sesijama, koje su ograničene na 2 sata, je učesnicima predstavljen i objašnjen FQUAD okvir i svakom učesniku je uručen dokument sa specifikacijom svih karakteristika jezika koje se njime evaluiraju, sa pripadajućim pod-karakteristikama i primerima.

Nakon sesija korisnicima su putem email-a prosleđeni upitnici. Prvi upitnik, koji služi za rangiranje karakteristika DSL-a, omogućava učesniku da rangira po važnosti (po sopstvenom mišljenu) karakteristike DSL-a predviđene FQUAD okvirom. Ovde je bitno napomenuti da je korisnicima omogućeno da navedu koje bi karakteristike ili pod-karakteristike oni dodali u okvir.

Putem drugog upitnika korisnicima se omogućava ocenjivanje koncepata FQUAD okvira, davanjem odgovora o tome da li se slaže ili ne slaže sa tvrdnjama koje se tiču ovih koncepata. Pored ova dva upitnika, dodatni podaci su prikupljani omogućavanjem korisnicima da u poseban dokument unesu svoja mišljena o FQUAD okviru u slobodnoj formi. Na osnovu povratnih informacija dobijenih u sklopu eksplorativne studije, upitnici u drugoj studiji se kreirani pomoću Microsoft Excel tabela i formula umesto Word dokumenta, za koje su korisnici rekli da im oduzimaju više vremena za popunjavanje.

Analiza rezultata

Sprovedene studije imaju za cilj evaluaciju FQUAD okvira u odnosu na tri osnovna kriterijuma koja su postavili autori po uzoru na smernice date u [136]. Ti kriterijumi su:

1. Osnovna evaluacija – ovaj kriterijum predstavlja stav korisnika o tome da li bi mogli da koriste FQUAD za svoje realne potrebe ili ne.
2. Evaluacija korišćenja – ispituje da li korisnici ocenjuju FQUAD kao koristan ili ne.
3. Evaluacija dobiti – Ovaj kriterijum ocenjuje da li korisnici smatraju da je FQUAD bolji od rešenja koja su im bila na raspolaganju do tada.

Za svaki od navedenih kriterijuma su definisani i pod-kriterijumi koji će ovde biti izuzeti. Za upoznavanje sa njima čitaoci se upućuju na peto poglavlje originalne publikacije [53]. Vezano za analizu rezultata eksplorativne studije, autori navode da su na osnovu odgovora na pitanja za prvi kriterijum zaključili da su učesnici ocenili da je FQUAD okvir lako razumljiv i jednostavan za primenu. Na osnovu komentara se može zaključiti da su neki učesnici imali problema sa razumevanjem nekih pod-karakteristika definisanih okvirom, što predstavlja jedan od pravaca za njegovo unapređenje. Pored ovoga, učesnici su izrazili potrebu za mehanizmom automatskog generisanja rezultata koji nedostaje u FQUAD-u. Na osnovu pitanja koja se tiču drugog kriterijuma, zaključuje se da su učesnici prve studije imaju poverenje u korisnost procene DSL jezika uz pomoć FQUAD okvira. Pored ovoga, FQUAD je preporučen za upotrebu i ostalim timovima unutar kompanije, što potvrđuje i treći evaluacioni kriterijum.

Učesnici druge studije su jednoglasno procenili FQUAD kao jasan proces evaluacije, navodeći da nisu imali problema sa interpretacijom značenja njegovih iskaza, što ukazuje na efikasnost poboljšanja koja su uvedena nakon prve studije. Pored ovoga, korisnici su imali predloge o reorganizaciji karakteristika i pod-karakteristika od kojih su neke uvažene i uvedene u finalnu verziju okvira. Jedna zamerka je bila što je namena FQUAD okvira evaluacija već gotovih jezika, dok bi pristup ovakvom metodu evaluacije bio korisniji u ranim fazama razvoja DSL-a. Osim navedenog, mišljenja učesnika druge studije su potvrdila zaključke izvedene u prvoj studiji.

Opisano istraživanje, kao i sam FQUAD okvir predstavljaju veliki doprinos evaluacionoj metodologiji primenjenoj u ovoj disertaciji. Sam okvir je korišćen za procenu Kroki alata dok su iskustva i smernice autora prilikom izvođenja studija poslužili za dizajn i izvođenje studija slučaja opisanih u tekstu ove disertacije.

2.3.4.3. Evaluacija WED dijagrama

Kratak pregled evaluacionog eksperimenta Window-Event dijagrama i pripadajućeg AIDE alata dat je u [38]. Spomenuti alati su u trenutku objavljivanja rada bili još na nivou prototipa, tako da autori nemaju konkretne rezultate evaluacije rešenja u industriji, ali navode rezultate istraživanja sprovedenog sa studentima kao neke utiske rada sa kompanijama.

U prvom eksperimentu, nekoliko grupa studenata je dobilo dva zadatka kreiranja istog korisničkog interfejsa koristeći olovku i papir, Microsoft PowerPoint i tablu za crtanje, posle čega su iznosili svoje utiske istraživačima. Učesnici koji su koristili papir i olovku kreirali najobuhvatnije modele interfejsa, pri čemu su se najviše zabavili i prijavili da se osećaju najmanje ograničenim alatom koji su koristili. Modeli kreirani pomoću alata PowerPoint su ocenjeni kao vizuelno najverniji i najprivlačniji. Prema rečima učesnika, glavna prednost PowerPoint-a je mogućnost predstavljanja rešenje pomoću slajdova prezentacije, što nije bilo dostupno drugim grupama. Kao najveći nedostaci korišćenja PowerPoint-a se navodi potreban trud, nedostatak fleksibilnosti i neprirodan način rada u timovima. Na osnovu ovoga, autori zaključuju da njihov AIDE alat spaja prednosti ova dva načina rada tako što omogućava uvoz ručno kreiranih skica i njihovu kombinaciju sa dijagramima stanja, kako bi se dobio model koji predstavlja i strogo vizuelne aspekte interfejsa, ali i specifikaciju ponašanja korisnika prilikom njegovog korišćenja. Kao primer iz industrije, autori navode neimenovan tim u kojem su WED dijagrami korišćeni i od strane dizajnera, koji su uvozili svoje skice interfejsa kreirane preferiranim alatima, nakon čega su dobijene modele i generisan kod koristili članovi razvojnog tima.

Nažalost, metod za evaluaciju u ovom radu ne poseduje dovoljno podataka, niti prati određenu metodologiju koja bi mogla biti korisna za dizajn i sprovođenje budućih eksperimenata, ali predstavlja koristan uvid u jedan primer konkretne metodologije za razvoj softvera baziran na skicama korisničkog interfejsa sa pratećim evaluacionim primerom, što predstavlja dobru polaznu tačku za izučavanje ove oblasti.

2.3.4.4. XIS-Web evaluacija

Evaluacija XIS-Web pristupa, opisana u [188], sastoji se od dve studije slučaja u okviru kojih se ovaj pristup koristio za razvoj realnih sistema i jedne probne sesije sa korisnicima koja bi omogućila evaluaciju korisnosti XIS-Web platforme.

U okviru sprovedenih studija slučaja, posmatrane web aplikacije su razvijene tradicionalnim pristupom ručnog modelovanja i implementacije i pomoću XIS-Web pristupa, kako bi se uporedila efikasnost predloženog pristupa u odnosu na tradicionalne metodologije rada. U prvoj studiji slučaja (Studija slučaja A) kreirana je aplikacija za upravljanje vremenskim okvirima koji se rezervišu od strane studenata. Svaki od studenata, kao korisnik ovog sistema, treba da je u mogućnosti da pregleda rezervisane i slobodne vremenske okvire, rezerviše za sebe okvir u vremenu koje je slobodno i da upravlja svojim okvirima.

Aplikacija razvijena u okviru druge studije slučaja (Studija B) ima za cilj da omogući građanima digitalizaciju i upravljanje svojim ličnim dokumentima (lična karta, vozačka dozvola, itd.). Pored građana koji imaju pristup pregledu i upravljanju svojim dokumentima, korisnici ovog sistema su i administrativni radnici kojima sistem omogućava kreiranje šablona ličnih dokumenata koje njihova ustanova izdaje i na osnovu kojih građani mogu da kreiraju digitalne kopije svojih ličnih dokumenata.

Autori navode da je uslov za pozitivnu evaluaciju predložene tehnologije da se korišćenjem XIS-Web pristupa može generisati više od 70% programskog koda opisanih aplikacija. Prema rezultatima Studije A, odnos ručnog i generisanog programskog koda iznosi 80.8%, dok je u slučaju

Studije B taj odnos 69.4%. U publikaciji se navodi da je razlog ovog rezultata Studije B činjenica da aplikacija kreirana u okviru njene sesije sadrži određeni deo kompleksne poslovne logike čije modelovanje nije predviđeno XIS-Web konceptima. Nažalost, u radu nije navedeno po kojem kriterijumu su birane spomenute aplikacije. U radu su izostavljeni i neki značajni detalji evaluacije kao što je opis manuelnog procesa modelovanja i implementacije, kao i koji režim modelovanja XIS-Web metodologije je korišćen u studijama.

Drugi deo evaluacione studije uključivao je sesiju sa korisnicima koja je bila fokusirana na tri aspekta: (i) da je korišćeni jezik dovoljno dobar za primenu u domenu specifikacije web aplikacija i kolikoj je meri težak za učenje, (ii) evaluacija pratećih softverskih alata (grafički editor, validator modela, generator modela i generator programskog koda) i (iii) generalna evaluacija pristupa.

U sesijama je učestvovalo ukupno 12 korisnika starosti između 23 i 30 godina koji su posedovali minimalno diplomu osnovnih studija računarstva. Autori navode kako je pola učesnika imalo iskustva u razvoju web aplikacija, dok je četvoro učesnika imalo i profesionalno iskustvo na istom polju. Zadatak svakog od učesnika je bio da, nakon 15-minutne prezentacije XIS-Web metodologije, razviju aplikaciju modelovanu u okviru Studije slučaja A. Nakon obavljenog posla, ispitanici su ispunjavali upitnike na osnovu kojih je vršena evaluacija, čiji rezultati su navedeni u publikaciji u vidu prosečnih ocena koje su učesnici davali evaluiranim aspektima ove metodologije. Na osnovu prosečne ocene jezika, koja iznosi 4.1 od mogućih 5, autori smatraju da jezik nije ocenjen kao lak za učenje i da tome planiraju da se posvete u budućnosti.

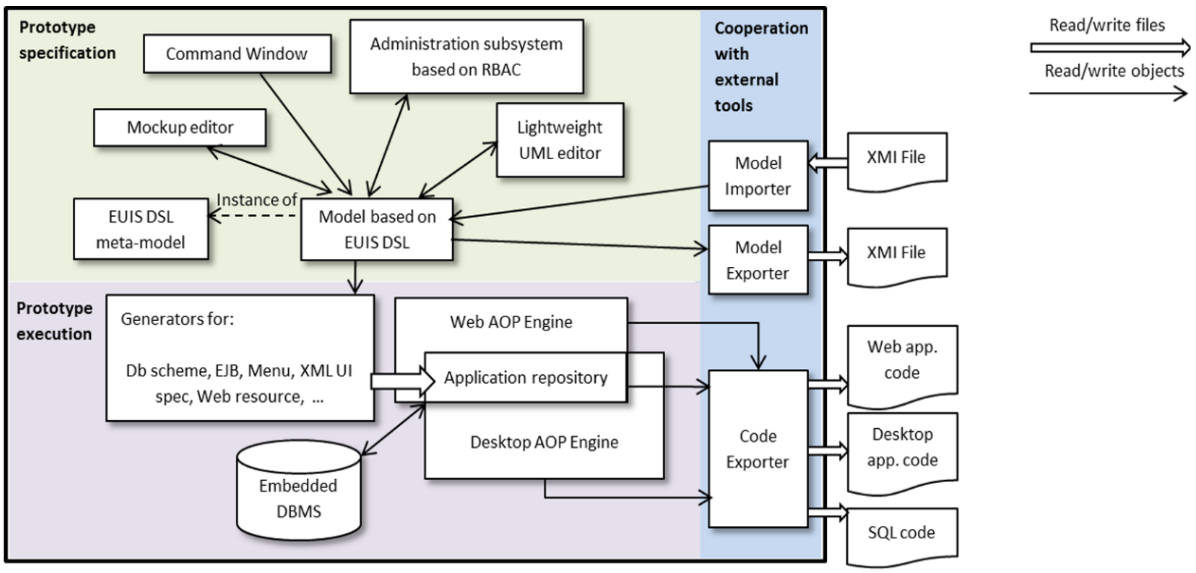
Sa druge strane, korišćeni alati koji su ocenjeni prosečnom ocenom 4.53 predstavljaju najbolje ocenjen aspekt metodologije. Ukupna korisnost metodologije je ocenjena prosečnom ocenom 4.33. Zaključak autora je da je da ispitanici smatraju kako XIS-Web pristup donosi značajna poboljšanja produktivnosti, ali da postavljena ograničenja (npr. zavisnost od Enterprise Architect platforme) mogu uticati na njegovu primenu u realnim projektima. Pored ovoga, navodi se da se ukupan broj od 12 ispitanika može smatrati nedovoljnim za statističku analizu, mada se autori pozivaju na izvore [193] koji tvrde da se 80% problema vezanih za upotrebljivost može evaluirati na malim grupama od 5 ispitanika.

POGLAVLJE 3

PROJEKTOVANJE I IMPLEMENTACIJA SOFTVERSKOG REŠENJA

U ovom poglavlju će biti predstavljeno projektovanje i implementacija softverskog rešenja pod nazivom Kroki, kao rezultat višegodišnjeg rada i istraživanja na ovom polju. Kroki potiče od francuske reči *croquis* što znači skica. Arhitektura Kroki alata (slika 15) je projektovana da obezbedi:

- kolaborativni razvoj sa korisnicima koji nemaju znanje projektovanja i programiranja softverskih sistema,
- efikasno pokretanje direktno iz razvojnog okruženja alata, dajući mogućnost korisniku da može da isproba prototip tokom modelovanja kad god poželi,
- ponovno korišćenje informacija dobijenih prilikom razvoja prototipova u kasnijim fazama razvoja, kako bi se smanjilo nepotrebno trošenje resursa na razvoj „od nule“.



Slika 15 Arhitektura Kroki alata

Specifikacija poslovnih aplikacija koja se kreira u okviru Kroki alata je bazirana na internom UI standardu [23] i zadaje se korišćenjem EUIS DSL-a (nezavisnim jezikom implementiranim na bazi EUIS UML profila [27] u okviru Kroki alata). Za specifikaciju poslovne aplikacije implementirane su tri konkretne sintakse:

- grafička sintaksa u vidu skica korisničkog interfejsa koju implementira editor skica (*Mockup* editor),
- grafička sintaksa koja podseća na UML dijagram klasa sa stereotipima, koju implementira pojednostavljeni UML editor (*Lightweight* UML editor) i
- konkretna sintaksa tekstualnog komandnog jezika čiji iskazi se unose u okviru komandne konzole.

S obzirom da svi editori rade nad istim modelom, izmene specifikacije izvršene u okviru jednog editora su trenutno vidljive i u ostalim.

Pokretanje razvijene specifikacije se obavlja korišćenjem web ili desktop generičke aplikacije (*engine*), za koje generatori obezbeđuju programski kod i parametre za pokretanje koje preuzimaju iz modela. Rukovanjem pokrenutom aplikacijom, korisnici mogu da isprobaju osnovne funkcije budućeg sistema, pri čemu je, ukoliko se ustanovi da postoje nedostaci, uvek moguć povratak u režim modelovanja kako bi se oni ispravili i generisao se novi prototip. Na ovaj način se inkrementalno dolazi do izvršive specifikacije zahteva u vidu Kroki modela koja je evaulirana na osnovu funkcionalnog prototipa i potvrđena od svih uključenih strana. Ohrabrivanjem uključivanja krajnjih korisnika u proces modelovanja i omogućavanje brzog pokretanja izvršivih prototipova visokog stepena funkcionalnosti se doprinosi premošćavanju komunikacionog jaza između naručilaca i razvojnog tima [1, 2, 3, 19].

Razvijeni model baziran na EUIS DSL-u se može eksportovati u XMI format radi korišćenja od strane UML alata za modelovanje opšte namene. Programski kod razvijenog prototipa se takođe može eksportovati u Eclipse projekat. Na ovaj način se postiže ponovno korišćenje informacija dobijenih prilikom razvoja prototipova za potrebe kasnijih faza razvoja softvera.

Dijagram klasa razvijen UML alatima opšte namene i snimljen u XMI format se može učitati u Kroki alat i pokrenuti. Na ovaj način se Kroki može uključiti i u kasnijim fazama razvoja samo za evaluaciju prototipa, ukoliko određenom timu više odgovara drugačiji skup softverskih alata.

Kroki alat[21] je razvijen na programskom jeziku Java i predstavlja rešenje otvorenog koda, tako da je njegov programski repozitorijum[22] javno raspoloživ. Način korišćenja njegove ranije verzije je dostupan u okviru snimljenog videa[23].

Deo Kroki alata je razvijen u okviru disertacije [93]. U [93] je implementiran editor za pojednostavljeni UML dijagram klasa kojim je podržana konkretna grafička sintaksa EUIS DSL-a čiji izgled podseća na UML dijagram klasa sa stereotipima.

Inicijalne verzije različitih funkcija Kroki alata su implementirane u više master radova koji su testirani, nadograđivani, integrisani u Kroki alat i dalje održavani od strane autora ove disertacije, radi dobijanja arhitekture kao na slici 28.

U nastavku ovog poglavlja je ukratko, radi kompletnosti teksta, predstavljen interni UI standard i EUIS UML profil (detalji se mogu naći u [27]). Zatim je prikazana implementacija EUIS DSL-a i njegovih sintaksi u okviru pripadajućih editora, implementacija generičke web aplikacije koja obezbeđuje efikasno pokretanje razvijene specifikacije i podsistemi za saradnju sa alatima za modelovanje i programiranje opšteg tipa. Na kraju poglavlja se nalazi poređenje Kroki alata sa postojećim rešenjima predstavljenim u sekciji 2.2.4.

3.1. Interni standard korisničkog interfejsa poslovnih aplikacija

Internim UI standardom su definisane funkcionalne i vizuelne karakteristike minimalno potrebnog broja UI komponenti poslovne aplikacije. Projektovan je sa namerom da obezbedi jednostavno korišćenje aplikacije, brzu obuku korisnika i automatizaciju izrade korisničkog interfejsa. Njegovi osnovni elementi su aplikativni podsistemi i više vrsta ekranskih formi [23].

[21] https://www.kroki-mde.net/
[22] https://github.com/KROKIteam/KROKI-mockup-tool
[23] https://youtu.be/r2eQrl11bzA

Aplikativni podsistemi predstavljaju softverske podsisteme (module) u sklopu poslovne aplikacije koji služe kao podrška grupi poslova koji se obavljaju na određenom radnom mestu. Sa aplikativnog stanovišta, ovi podsistemi služe za grupisanje funkcija kojima se pristupa u okviru glavnog menija poslovne aplikacije.

Ekranske forme podržane standardom su:

1. Standardna forma,
2. „Parent-Child" forma,
3. „Many-to-Many" forma,
4. Forma za prikaz izveštaja podsistema,
5. Forma za preuzimanje parametara od korisnika.

Standardna forma služi da korisniku pruži prikaz podataka jednog perzistentnog entiteta i pristup svim operacijama koje je u tom trenutku moguće izvršiti nad njim. Operacije zajedničke za sve entitete nazvane su „standardne operacije" i nalaze se u gornjem delu forme u vidu dugmića sa odgovarajućim ikonicama. Operacije specifične za pridruženi entitet smeštene su sa desne strane forme u vidu dugmadi sa natpisima.

Standardne operacije su: pretraga po svim poljima, pregled, unos, izmena, brisanje i promena vrste prikaza, dok specifične operacije predstavljaju složene procedure za manipulisanje podacima (npr. knjiženje, fakturisanje ili štampanje izveštaja). Operacije se uvek odnose na izabrani red (instanca klase ili slog tabele u bazi podataka kojim se manipuliše) putem standardne forme.

Standardne forme podržavaju tabelarni i pojedinačni (tzv. „jedan ekran – jedan red") prikaz koje korisnik može da menja po potrebi. U tabelarnom prikazu podataka korisniku se prikazuju svi redovi entiteta pridruženog formi, pri čemu korisnik može da izabere željeni red i aktivira operaciju nad njim. Pojedinačni prikaz omogućava korisniku da vidi, dodaje i menja podatke u okviru panela koji se sastoji od polja (UI komponenti) pridruženih atributima datog entiteta ili izvedenim vrednostima: kalkulisanim, agregiranim i *lookup*.

Sadržaj kalkulisanih polja se formira primenom određene formule nad izabranim obeležjima datog ili sa njim povezanih entiteta (npr. suma, proizvod) u okviru jednog reda. Sadržaj agregiranih polja se formira korišćenjem funkcije za agregaciju nad vrednostima jednog obeležja datog ili vezanih entiteta, nad više redova (npr. prosek, minimum, maksimum). Sadržaj *lookup* polja predstavlja vrednost reprezentativnih obeležja *parent* entiteta. Parent entitet je entitet povezan sa datim asocijacijom „više-prema-1", gde je kardinalitet 1 na strani *parent*-a a „više" na strani *child*-a.

Relacije između standardnih ekranskih formi se definišu pomoću mehanizama *zoom*, *next* i *activate*.

Pod *zoom*-om se podrazumeva aktiviranje forme pridružene *parent* entitetu putem zoom dugmeta sa tri tačke, iz koje korisnik može izabrati željeni red i preuzeti njegove vrednosti u polja forme pridružene *child* entitetu. *Combozoom* je varijanta ovog mehanizma, gde se zoom dugme nalazi pored combobox komponente i pokreće se radi mogućnosti dodavanja reda u parent formi, ukoliko traženi podatak ne postoji u okviru podataka date *combobox* komponente.

Next mehanizam se koristi za prelazak sa forme pridružene *parent* entitetu na formu pridruženu *child* entitetu, tako da se u *child* formi nalaze samo filtrirani podaci u odnosu na vrednost ključa tekućeg reda iz *parent* forme.

Activate mehanizam omogućava aktiviranje forme od strane druge forme bez ikakvih ograničenja. Aktivirana forma ne mora biti u vezi sa formom koja ju je pokrenula.

Standardni paneli realizuju izgled i funkcionalnost standardne forme, ali nisu namenjeni da se prikazuju samostalno, već služe za građenje složenih formi. Složene forme su „Parent-Child“ i „Many-to-Many”.

„Parent-Child” forme prikazuju hijerarhijski organizovane standardne panele čiji entiteti su povezani asocijacijama „1-prema-više“. Panel na n-tom nivou hijerarhije filtrira svoj sadržaj u skladu sa vrednošću odabranog reda panela na n-1-om nivou hijerarhije.

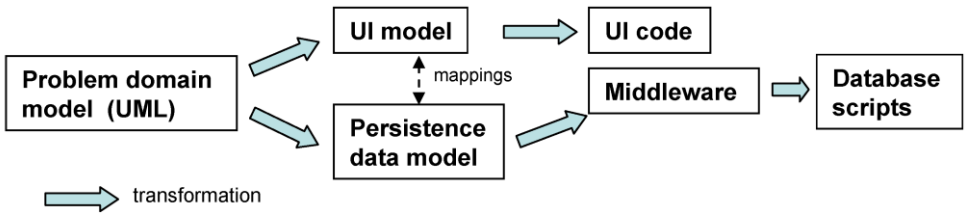
„Many-to-Many“ forme su složene forme koje omogućavaju manipulaciju i pregled podataka o entitetima koji su nastali kao posledica asocijacija „više-na-više”, sa ili bez pridruženih asocijativnih klasa. S obzirom da se ovaj tip asocijacija u relacionim bazama podataka prevodi u vezne tabele koje sadrže ključeve povezanih entiteta i kolone koje odgovaraju atributima iz asocijativne klase (ako postoji), ove forme omogućavaju brz i lak način za kreiranje novih redova u datim tabelama.

U poslovnim aplikacijama je potrebno omogućiti štampanje različitog broja namenskih izveštaja koji se koriste u sklopu svakodnevnog poslovanja preduzeća**. Forma za prikaz izveštaja** se aktivira sa menija aplikativnog podsistema. Svojim izgledom podseća na standardnu formu koja sadrži prikaz trenutno dostupnih izveštaja datog poslovnog podsistema. Odabirom željenog izveštaja korisniku se omogućava unos parametara putem **forme za preuzimanje parametara** i njegov prikaz na ekranu ili štampu.

F**orma za preuzimanje parametara** se aktivira prilikom pokretanja izveštaja ili složenih poslovnih obrada koje od korisnika zahtevaju unos vrednosti potrebnih za obavljanje željenih aktivnosti.

3.2. UML profil za specifikaciju korisničkog interfejsa poslovnih aplikacija

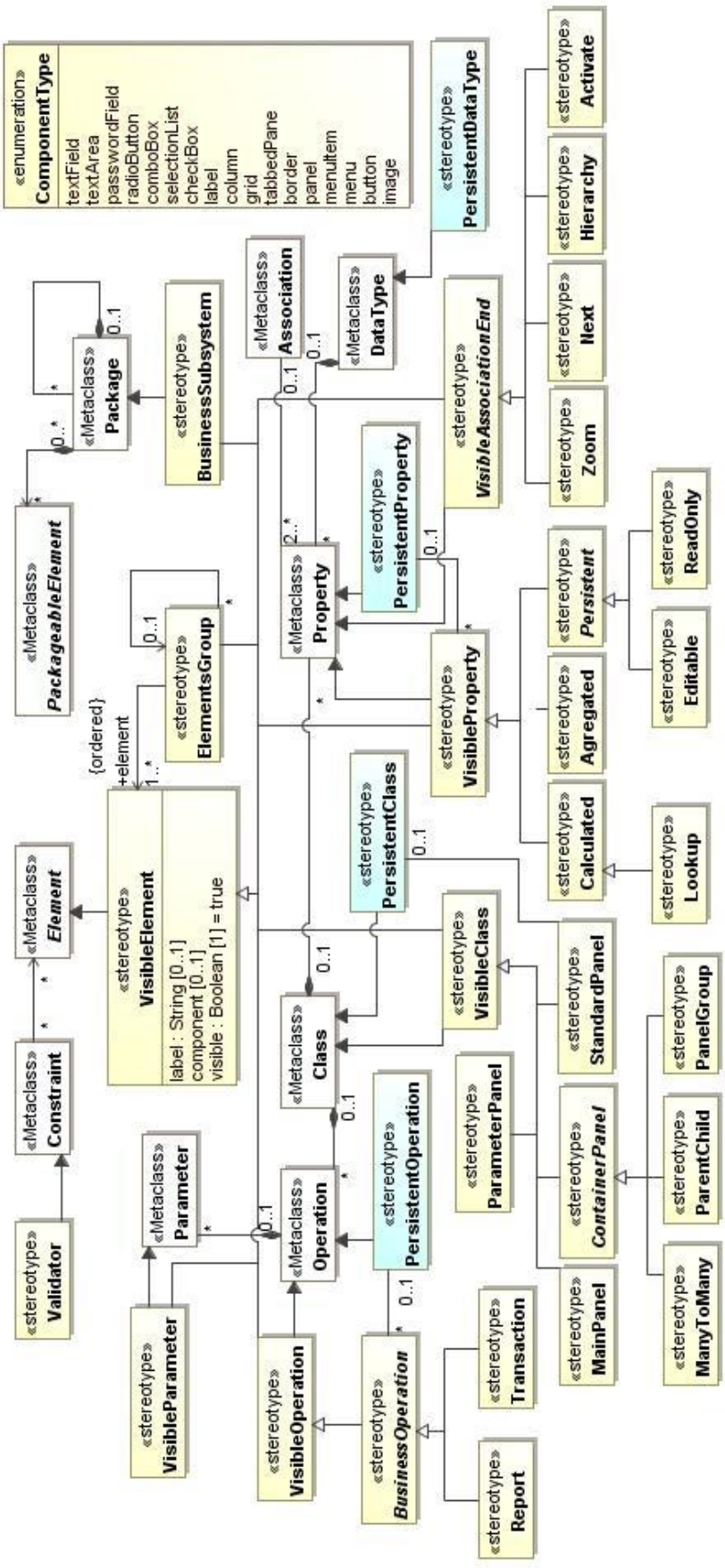
EUIS UML profil je projektovan sa ciljem da obezbedi brzo modelovanje korisničkog interfejsa poslovnih aplikacija i jednostavnu integraciju modela korisničkog interfejsa sa modelom podataka (perzistentnim modelom), korišćenjem UML dijagrama klasa kao zajedničke osnove [27]. Njime se proširuju meta-klase iz paketa UML::Kernel: `Class`, `Property`, `Operation`, `Parameter`, `Constraint` i `Package` (slika 17).



Slika 16 Razvoj poslovne aplikacije na bazi EUIS UML profila

Modelovanje ciljnog sistema korišćenjem EUIS UML profila podrazumeva (slika 16):

- kreiranje modela podataka nezavisnog od platforme korišćenjem dijagrama klasa,
- njegovu automatsku transformaciju u model korisničkog interfejsa i u model podataka specifičan za određenu implementacionu platformu,
- ručnu popravku i dopunu modela korisničkog interfejsa i
- generisanje koda za ciljnu platformu.

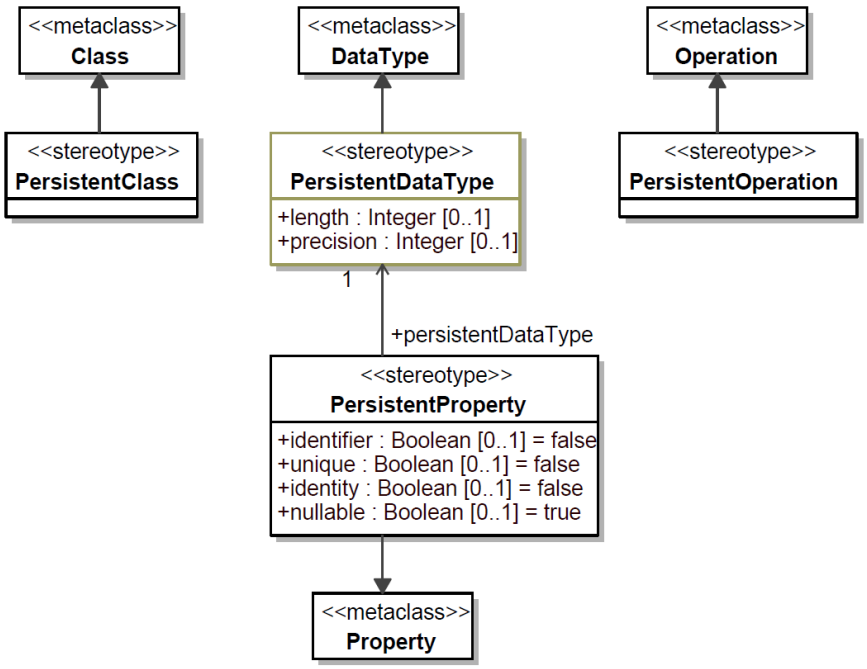


Slika 17 Struktura EUIS UML profila

Model podataka se oslanja na manji, pomoćni UML profil za modelovanje perzistentinih koncepata koji je dostupan u većini alata za modelovanje. U nastavku će biti ukratko prikazan perzistentni profil i svi stereotipovi EUIS UML profila koji su bitni za implementaciju Kroki alata. Kompletna specifikacija svih stereotipa, njihovih tagova i OCL ograničenja se može naći u [27].

3.2.1. Perzistentni profil

Profil za definisanje perzistentnih aspekata (srednji sloj, baza podataka) poslovnih sistema u okviru EUIS profila sadrži stereotipove koji su prikazani na slici 18.



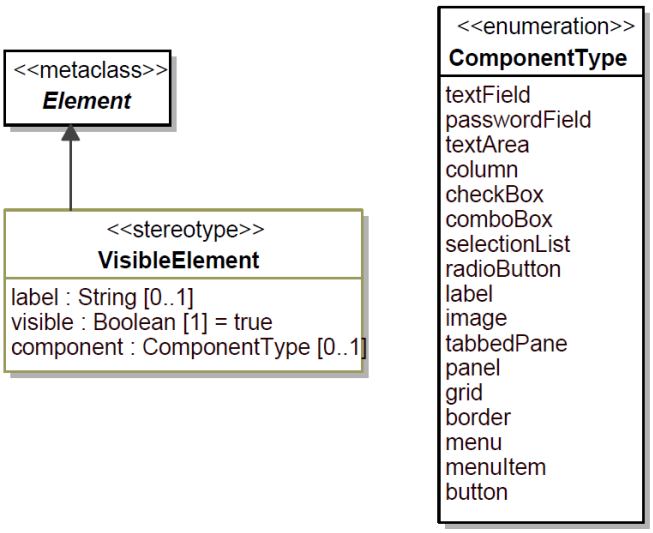
Slika 18 Struktura perzistentnog profila [27]

Osnovni stereotipovi ovog profila su [27]:

- `PersistentClass` – klasa čija instanca se trajno čuva u bazi podataka poslovne aplikacije.
- `PersistentProperty` – obeležje (atribut) perzistentne klase koje se trajno čuva (perzistentna klasa može imati i obeležja koja nisu perzistentna). Poseduje obeležje kojim se označava da li je obeležje identifikator (`identifier`), da li je njegova vrednost jedinstvena na nivou skupa instanci klase kojoj pripada (`unique`), da li mu se vrednost povećava za 1 za svaku novu instancu klase (`identity`) i da li je unos vrednosti obavezan prilikom kreiranja nove instance klase (`nullable`).
- `PersistentDataType` – koristi se za modelovanje tipa poqdataka kolone u bazi podataka na koju se određeno obeležje mapira. Za svaki perzistentni tip se može definisati dužina (`length`) i preciznost (`precision`), ukoliko se radi o numeričkim obeležjima.
- `PersistentOperation` – procedura koja služi za manipulaciju podacima u perzistentnom sloju. U zavisnosti od konkretne implementacije može biti programska funkcija u srednjem sloju aplikacije ili uskladištena procedura u relacionoj bazi podataka.

3.2.2. Vidljivi elementi

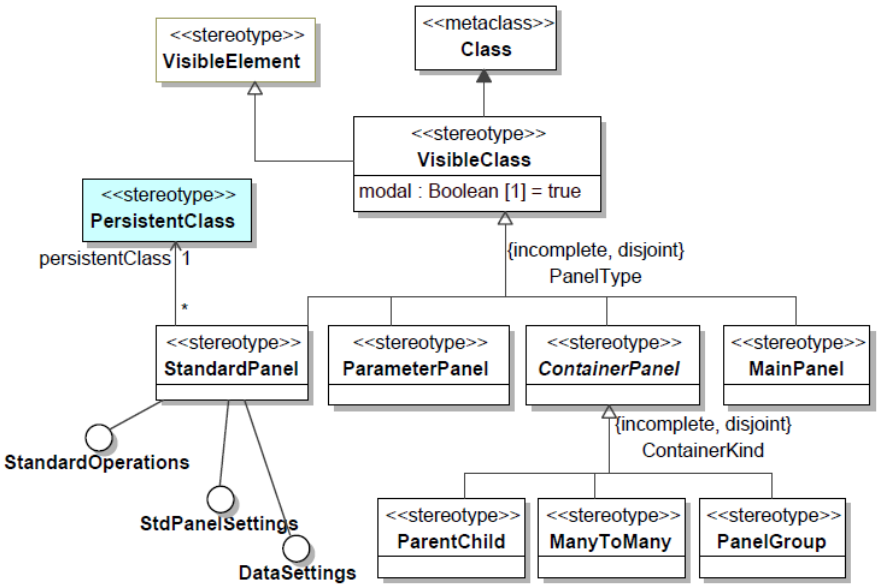
Stereotip `VisibleElement` označava element modela koji se preslikava na element korisničkog interfejsa (Slika 19) [27]. Pošto je meta-klasa `Element` predak svih UML meta-klasa, na ovaj način je omogućeno preslikavanje svih elemenata modela na elemente korisničkog interfejsa poslovne aplikacije. Svaki vidljivi element ima labelu (`label`) i UI komponentu čiji su tipovi predstavljeni nabrojanim tipom `ComponentType`.



Slika 19 „Vidljivi“ element [27]

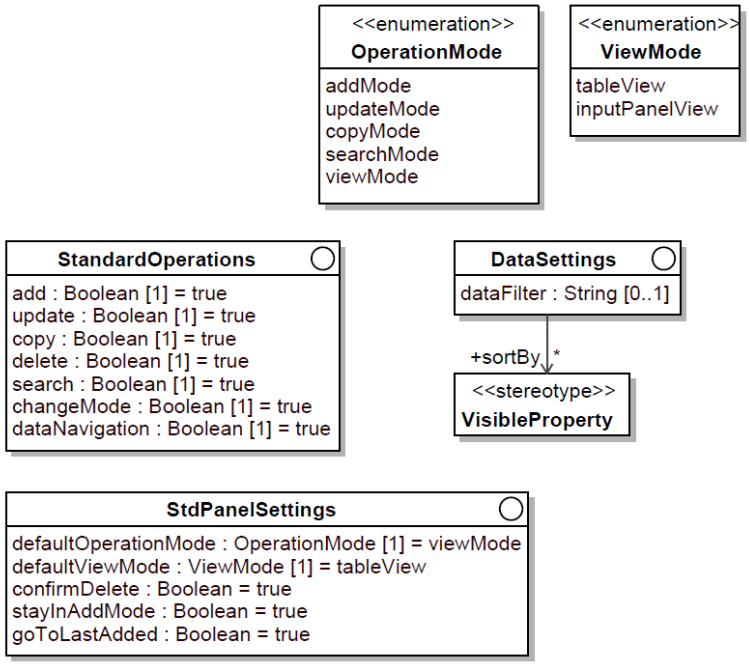
3.2.3. Vidljive klase

Stereotip `VisibleClass` omogućava preslikavanje klase iz modela na panel odgovarajuće vrste u okviru korisničkog interfejsa aplikacije (slika 20). Ukoliko se panelu pridruži prozor, postaje ekranska forma koja se može samostalno pokretati. U suprotnom se koristi kao element složenog panela.



Slika 20 „Vidljive“ klase - paneli [27]

Tag `label`, koji je nasleđen od stereotipa `VisibleElement`, se koristi kao naslov prozora. Ukoliko se ne unese prilikom modelovanja, kao naslov prozora koristiće se ime klase kojoj je pridružen stereotip.



Slika 21 Stereotipovi i nabrojane vrednosti vezane za stereotip `VisibleClass` [27]

Stereotip `StandardPanel` se koristi da se klasi iz modela pridruži standardni panel prilikom generisanja koda korisničkog interfejsa. Implementira tri interfejsa (slika 21):

1. `StandardOperations` – omogućava definisanje standardnih operacija (predviđenih UI standardom) koje su dozvoljene/zabranjene u okviru panela. Isključivanje određene operacije ovim putem je trajno - nije moguća naknadna dozvola te operacije podešavanjem korisničkih prava.
2. `StdPanelSettings` – putem ovog interfejsa se definiše skup podešavanja standardnog panela kojima se specificira njegovo ponašanje:
    a. `defaultViewMode`: režim prikaza podataka u kojem se panel otvara,
    b. `defaultOperationMode`: režim rada u kojem se panel otvara,
    c. `confirmDelete`: određuje da li je potrebno tražiti od korisnika potvrdu brisanja u okviru panela,
    d. `stayInAddMode`: određuje da li se po uspešnom unosu novog reda panel vraća u režim pregleda ili ostaje u režimu dodavanja,
    e. `goToLastAdded`: ovim podešavanjem se specificira da se nakon napuštanja režima za unos korisniku automatski označava poslednje uneti red.
3. `DataSettings` – interfejs koji omogućava definisanje filtera za podatke i način njihovog sortiranja prilikom prikaza.

`ParentChild` stereotip modeluje „Parent-Child“ formu opisanu UI standardom koja se sastoji od standardnih panela organizovanih u hijerarhiju. Klasa sa ParentChild stereotipom se povezuje sa klasama sa stereotipom `StandardPanel` asocijacijama na čijem kraju se nalazi stereotip `Hierarchy` kojim se definiše nivo hijerarhije za dati panel.

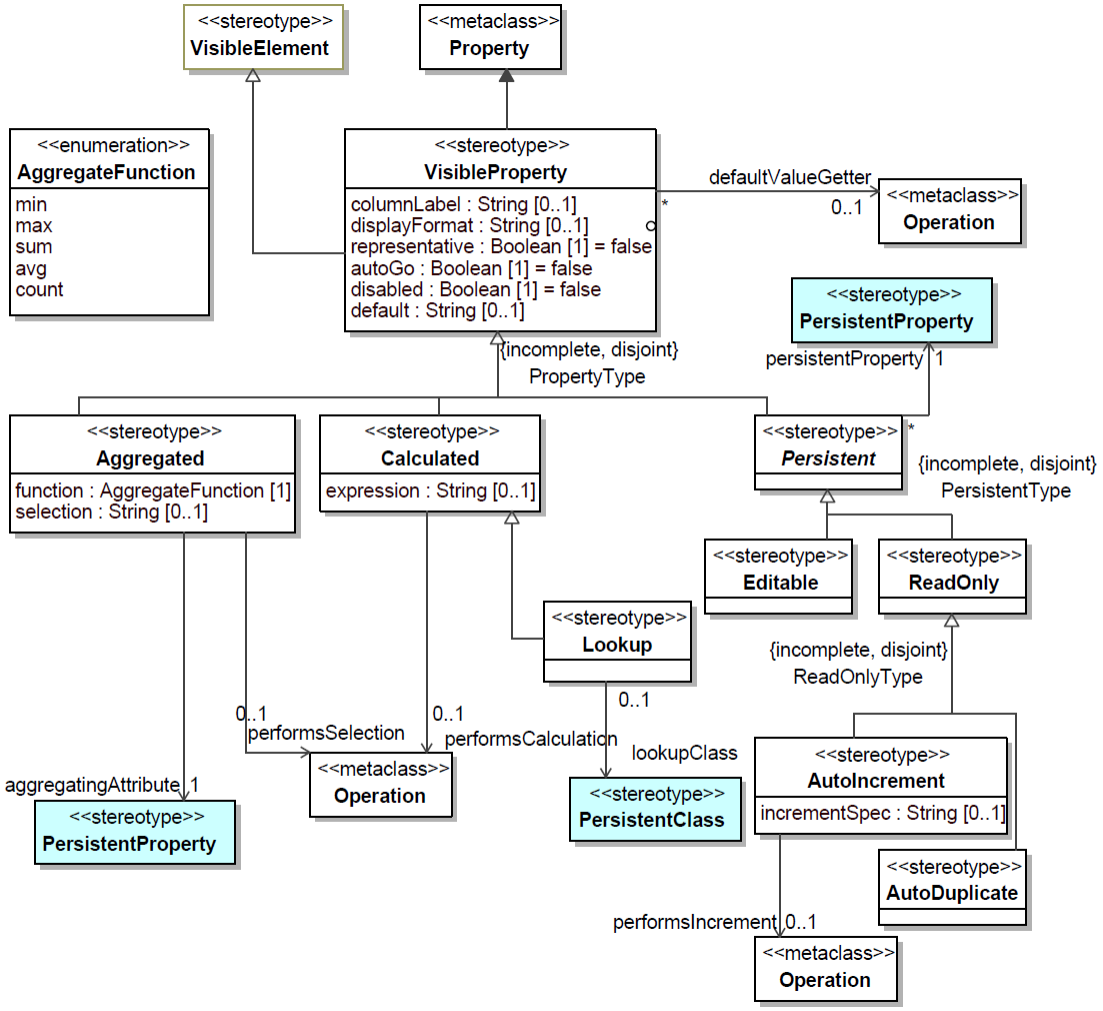
Klase kojima je pridružen `ManyToMany` stereotip služe za modelovanje složenih „Many-to-Many“ formi, na način kako je opisano internim UI standardom.

3.2.4. Vidljiva obeležja

Stereotip `VisibleProperty` je korenski stereotip za sva vidljiva obeležja koja ga nasleđuju (Slika 22). Vidiljivo obeležje predstavlja obeležje vidljive klase kojem je pridružena odgovarajuća komponenta korisničkog interfejsa.

Tagovi ovog stereotipa su:

- `columnLabel`: naslov kolone u tabelarnom prikazu standardnog panela.
- `displayFormat`: format u kom se vrednosti obeležja prikazuju i unose. Ako se ne navede, predviđeno je da se koristi podrazumevani format za tip obeležja ili prezistentni tip podatka.
- `representative` – označava da li je dato obeležje reprezentativno za svoju klasu. Reprezentativna obeležja se koriste za popunjavanje vrednosti *lookup* polja u klasama koje referenciraju datu.
- `autoGo` – označava da li se, nakon popunjavanja komponente vezane za dato obeležje, fokus automatski premešta na sledeću komponentu za unos u okviru istog panela.
- `disabled` – označava da li je polje vezano za obeležje onemogućeno za unos.
- `default` – OCL izraz kojim se definiše način dobavljanja inicijalne vrednosti obeležja.



Slika 22 „Vidljiva“ obeležja [27]

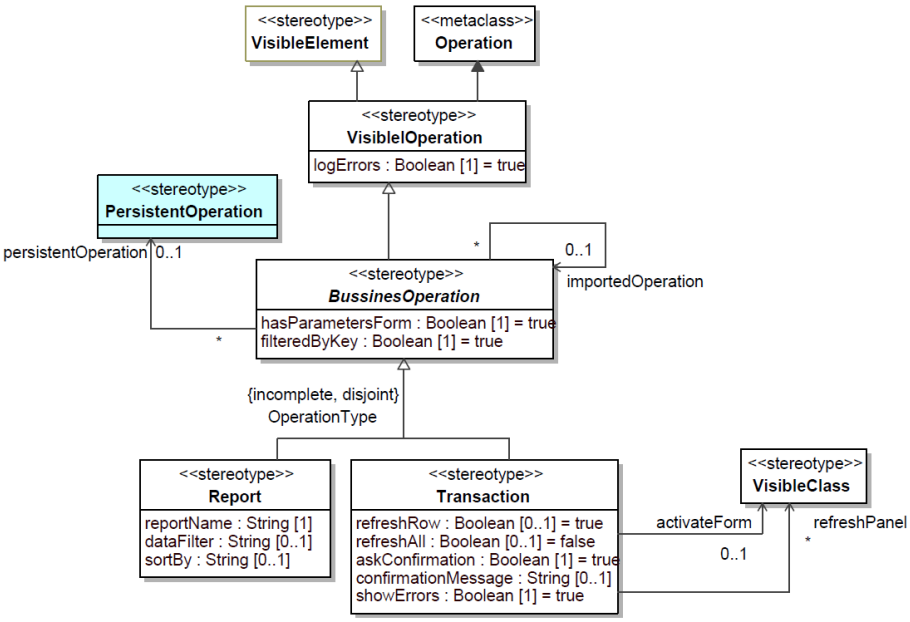
Tag `label` koji je nasleđen od stereotipa `Property` predstavlja tekstualni natpis (labelu) koji se prikazuje pored komponente korisničkog interfejsa koja je pridružena vidljivom obeležju. Ukoliko se vrednost ovog taga ne unese, koristiće se ime obeležja.

Nasleđeni tag `component` predstavlja tip grafičke komponente koja se pridružuje vidljivom obeležju. Ako se ne unese, predviđeno je da se koristi podrazumevani tip komponente za dati tip obeležja.

Stereotipi `Aggregated`, `Calculated` i `Lookup` modeluju respektivno agregirano, kalkulisano i lookup polje, na način definisan UI standardom. Stereotip `Calculated` ima tag `expression` u koji se unosi formula za računanje vrednosti u vidu OCL izraza.

Apstraktni stereotip `Persistent` označava vidljivo obeležje čija se vrednost trajno čuva u bazi podataka i može se pregledati i ažurirati putem pridružene komponente korisničkog interfejsa. Njegovi naslednici su `Editable` i `ReadOnly`.

3.2.5. Vidljive operacije



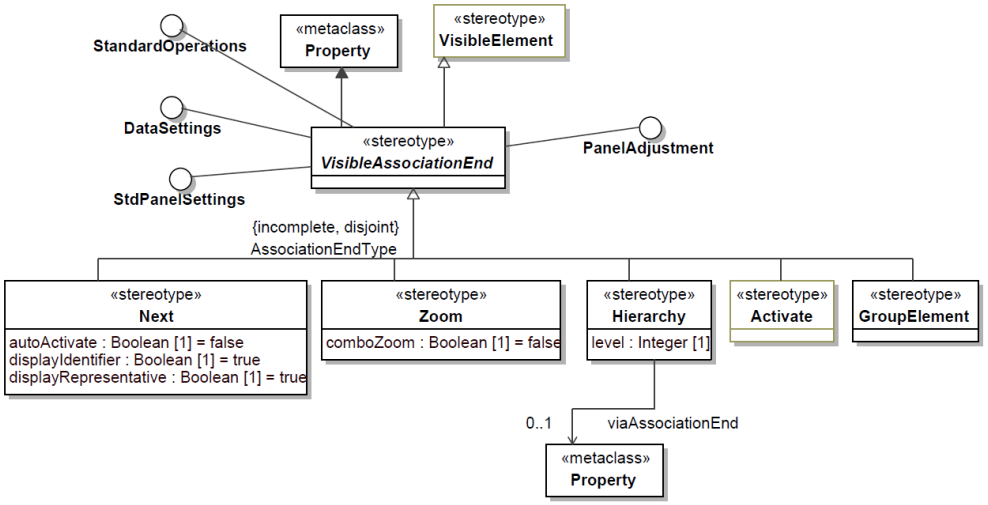
Slika 23 „Vidljive“ operacije [27]

Vidljiva operacija predstavlja metodu klase `VisibleClass` koja se aktivira od strane korisnika putem pridružene UI komponente (programskog dugmeta ili stavke menija). Kao natpis za ovu komponentu se koristi nasleđeni tag `label`, ukoliko je definisan, ili ime metode, ukoliko to nije slučaj. Korenski stereotip koji predstavlja vidljivu metodu je `VisibleOperation` (Slika 23).

`BussinesOperation` predstavlja metodu koja implementira neku poslovnu logiku u perzistentnom sloju. Može biti `Report` ili `Transaction`. Stereotip `Report` predstavlja operaciju koja kreira i prikazuje poslovni izveštaj koji se implementira nekim od alata za kreiranje izveštaja, na primer iReport[24] ili ReportServer[25]. Stereotip `BusinessTransaction` predstavlja metodu koja pokreće složenu poslovnu transakciju i koja je implementirana na nivou baze podataka, kao uskladištena procedura, ili u srednjem sloju, kao deo programskog rešenja poslovnog sistema.

[24] https://community.jaspersoft.com/project/ireport-designer
[25] https://reportserver.net

3.2.6. Vidljivi krajevi asocijacije



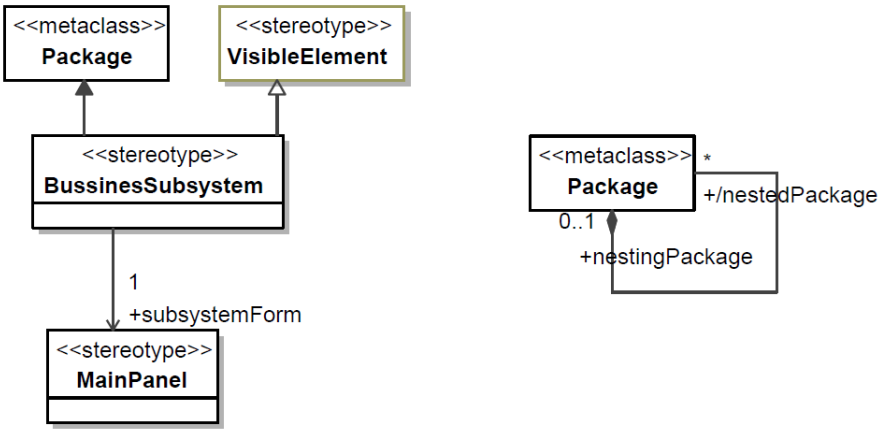
Slika 24 „Vidljivi“ krajevi asocijacije [27]

Svrha stereotipa `VisibleAssociationEnd` (Slika 24) je definisanje odnosa između aktivacionog panela (pridruženog klasi kojoj obeležje pripada) i odredišnog panela, koji pripada klasi na drugom kraju asocijacije [27]. `VisibleAssociationEnd` je apstraktni stereotip koji specificira zajednička svojstva konkretnih naslednika:

1. `Next` – označava da se redovi odredišnog panela filtriraju u odnosu na ključ izabranog reda aktivacionog panela.
2. `Zoom` – označava da se odredišni panel prikazuje kao forma koja omogućava izbor i preuzimanje podataka radi popunjavanja linija za unos aktivacione forme, kako bi se korisniku olakšao unos vrednosti povezanih obeležja.
3. `Hierarchy` – Pridružuje se krajevima asocijacije koji učestvuju u hijerarhijskom prikazu panela u okviru „Parent-Child“ i „Many-To-Many“ formi.
4. `Activate` – Označava da aktivacioni panel pokreće odredišni bez ikakvih restrikcija. U web implementacijama poslovnih sistema ovo je najčešće HTML link koji prikazuje povezanu stranicu.

3.2.7. Poslovni podsistemi

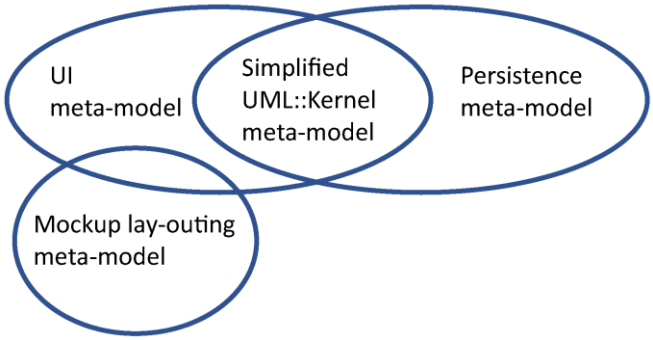
Kao što je predviđeno UI standardom, forme u okviru modelovane poslovne aplikacije se organizuju u podsisteme. U zavisnosti od prava korisnika, svaki podsistem može da ima svoju glavnu formu sa stavkama menija putem kojih se pristupa formama podsistema (ovo je slučaj u kojem korisnik ima pravo da radi samo sa jednim podsistemom) ili se poslovni podsistemi prikazuju kao podmeniji u glavnom meniju osnovne forme čitavog sistema, ukoliko korisnik ima pravo da radi sa više njih. U EUIS profilu poslovni podsistemi su definisani stereotipom `BussinessSubsystem` koji proširuje meta-klasu `Package`. Kao što se može videti sa Slike 25, svaki UML paket može imati svoje pod-pakete.



Slika 25 Stereotip `BussinesSubsystem` i metaklasa `Package` [27]

3.3 EUIS DSL

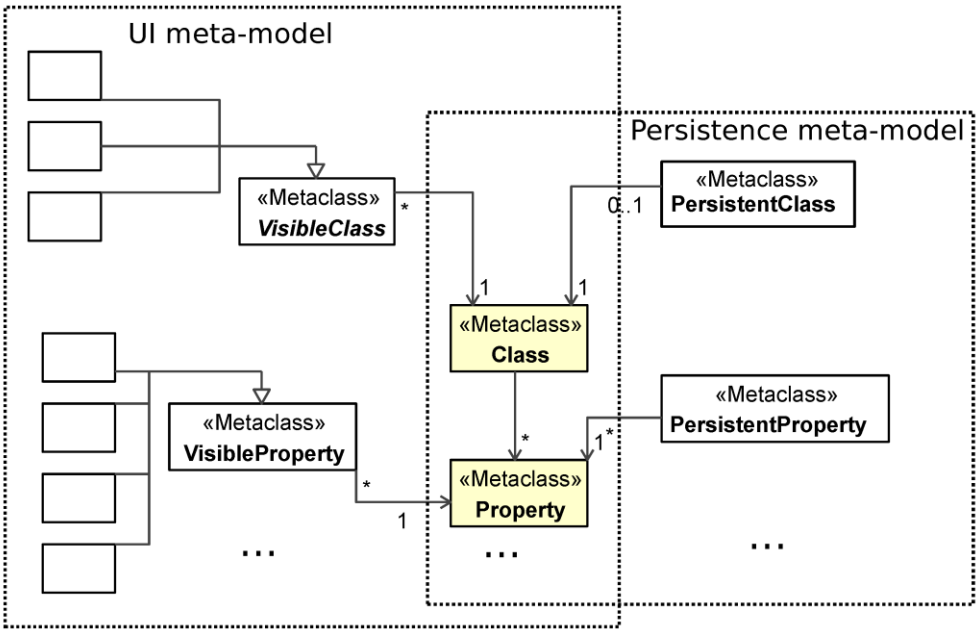
Osnova na kojoj je izgrađen Kroki alat je EUIS DSL— samostalan jezik kreiran na bazi EUIS UML profila. Njegov meta-model se sastoji od više manjih meta-modela integrisanih zajedničkim meta-klasama (slika 26). Meta-klase iz UI i perzistentnog meta-modela referenciraju meta-klase koje implementiraju pojednostavljeni `UML::Kernel` (slika 27). UI meta-model je integrisan sa meta-modelom za raspoređivanje UI komponenti u okviru skica korisničkog interfejsa, kojim se specificira izgled formi, redosled i uzajamni odnos njihovih elemenata (slika 28). Ovaj meta-model ima ulogu sekundarne sintakse za osnovni EUIS DSL.



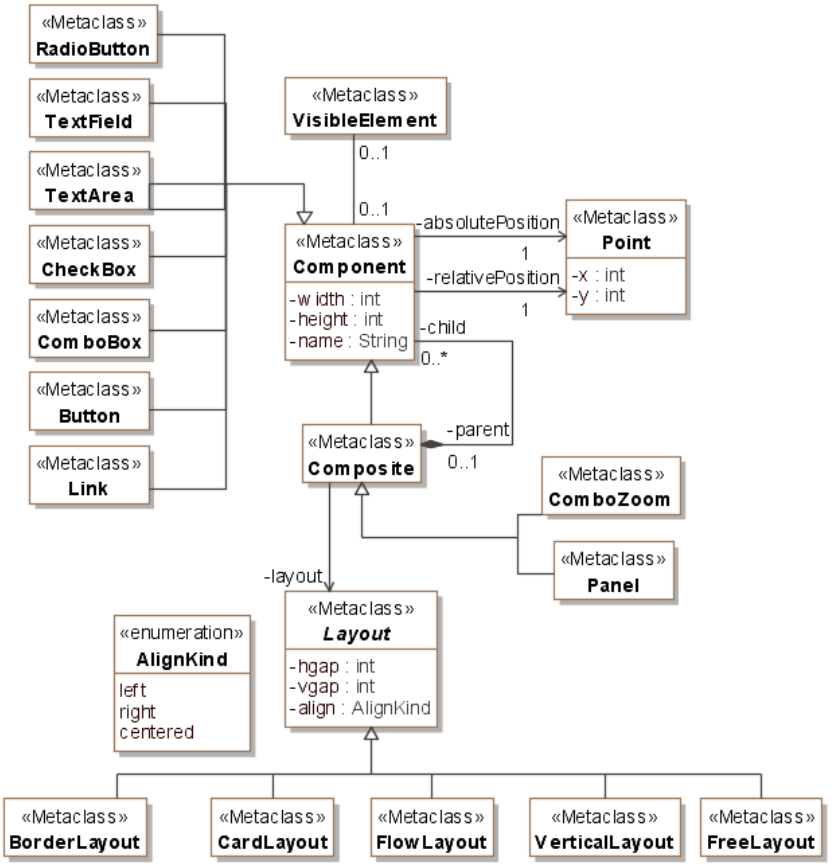
Slika 26 Struktura EUIS meta-modela

Cilj načina na koji je meta-model dizajniran je da omogući da se elementi UI modela posmatraju kao skice kroz editor skica i kao „obične“ klase sa stereotipima kroz pojednostavljeni UML editor. Potreba za M2M transformacijama, koje su bile deo ranijeg pristupa na bazi EUIS UML profila, kao i većine drugih pristupa opisanih u sekciji 2.2.4., na ovaj način je izbegnuta, jer je model sve vreme integrisan i spreman za pokretanje. S obzirom da su kao osnova jezika zadržane `UML::Kernel` meta-klase, mogućnost saradnje sa alatima za modelovanje opšte namene i dalje postoji, kroz uvoz i izvoz kreiranih dijagrama klasa.

Meta-klase iz UI meta-modela su spregnute i sa osnovnim meta-klasama RBAC meta-modela, radi mogućnosti definisanja korisničkih uloga i podešavanja prava nad elementima korisničkog interfejsa u okviru administrativnog podsistema Kroki alata. Inicijalna implementacija ovog podsistema je obavljena u okviru [190] i prezentovana u [166].



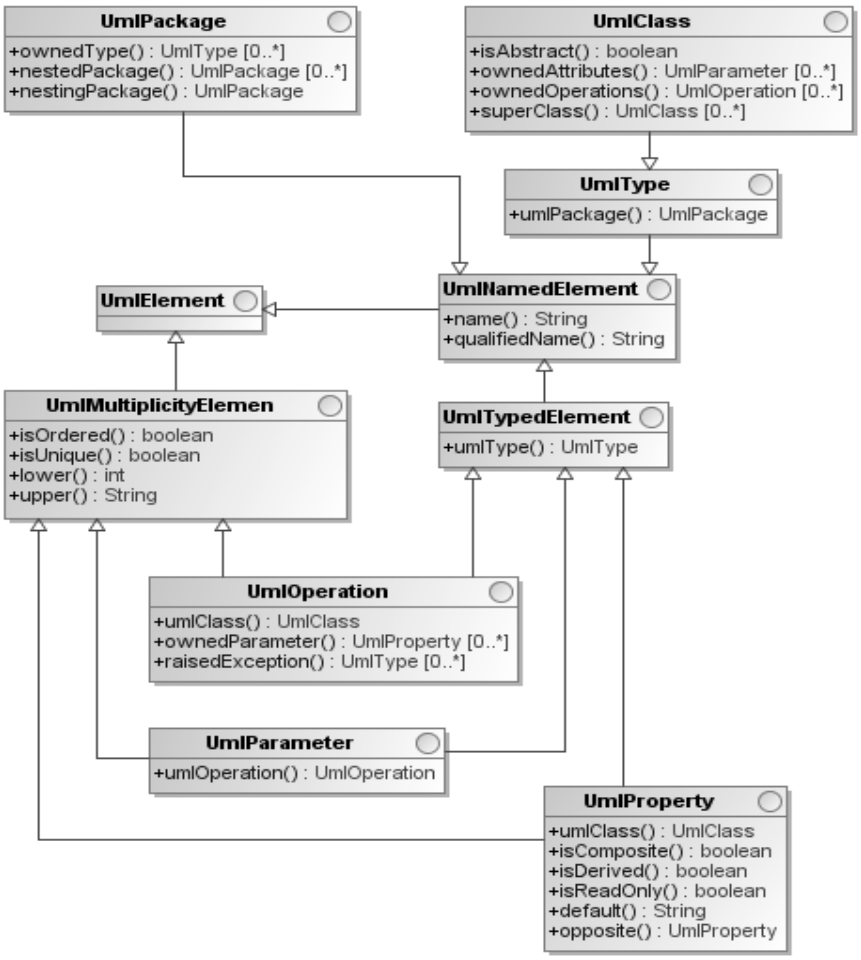
Slika 27 Skica integracije UI i perzistentnog meta-modela preko zajedničkih meta-klasa iz pojednostavljene implementacije `UML::Kernel`-a



Slika 28 Meta-model za raspoređivanje UI elemenata u okviru skica

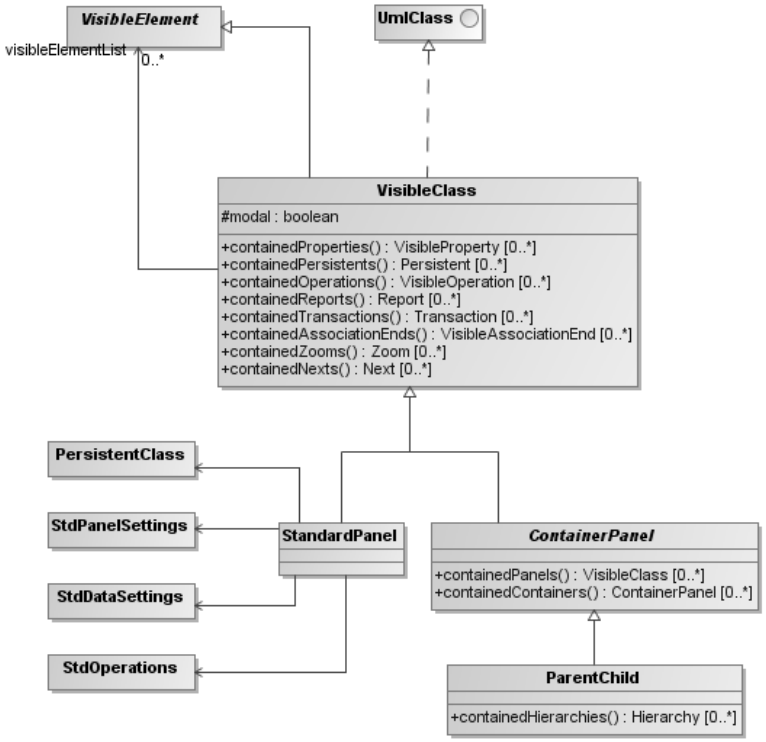
EUIS meta-model je implementiran na programskom jeziku Java, kao i ostatak Kroki alata. Java ne podržava višestruko nasleđivanje meta-klasa koje je veoma zastupljeno u okviru UML

meta-modela [13]. Iz ovog razloga je uveden skup interfejsa koji se u Javi mogu višestruko nasleđivati (slika 29), a koji obezbeđuju ponašanje meta-klasa iz `UML::Kernel`-a.

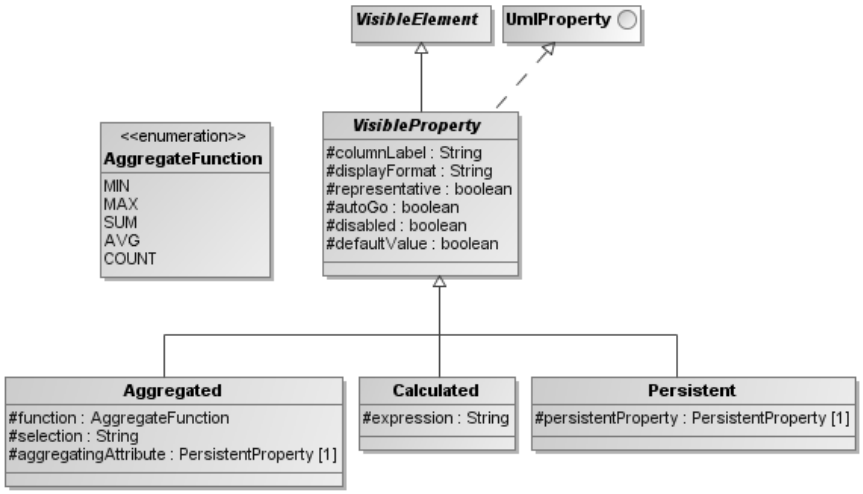


Slika 29 Java interfejsi koji specificiraju ponašanje UML::Kernel meta-klasa

Meta-klase koje odgovaraju stereotipima iz EUIS UML profila su implementirane tako da poseduju atribute koji odgovaraju tagovima stereotipa, na način kako je prikazano u sekciji 3.2, ali i da implementiraju odgovarajući interfejs sa slike 29, radi podrške osobina UML meta-klasa koje su u podlozi UML profila. Na slici 31 je kao primer prikazan deo implementacije meta-klasa za podršku „vidljivih“ obeležja – elemenata formi, a na slici 30 „vidljivih“ klasa – različitih vrsta panela. Ostatak meta-klasa je takođe implementiran po ugledu na odgovarajuće elemente EUIS UML profila. Detalji implementacije se mogu naći u [167, 168].



Slika 30 Deo Java implementacije „vidljivih“ klasa



Slika 31 Deo Java implementacije „vidljivih“ obeležja

S obzirom da se elementi modela mogu kreirati korišćenjem dva grafička editora, tekstualnim komandnim jezikom i importom iz alata za modelovanje opšte namene, implementiran je API koji kreira, briše i ažurira instance meta-modela, tako da se svakom novom elementu dodeli optimalno mesto u okviru skice korisničkog interfejsa, odnosno dijagrama klasa u okviru pojednostavljenog grafičkog editora (primer je dat na listingu 2).

```
public class ApiCommons {
	public static VisibleProperty makeVisibleProperty(String label, boolean visible,
ComponentType type, VisibleClass panel) {…}
	public static void removeProperty(VisibleClass visibleClass, int classIndex) {…}

	public static VisibleOperation makeVisibleOperation(String label, boolean visible,
ComponentType componentType, VisibleClass panel, OperationType operationType) {…}
	public static void removeOperation(VisibleClass visibleClass, int classIndex) {…}

	public static void addHierarchyElement(VisibleClass visibleClass, Hierarchy
hierarchy, VisibleClass targetPanel) {…}
	…
}
```

Listing 2 Deo API metoda za kreiranje instanci elemenata EUIS DSL meta-modela

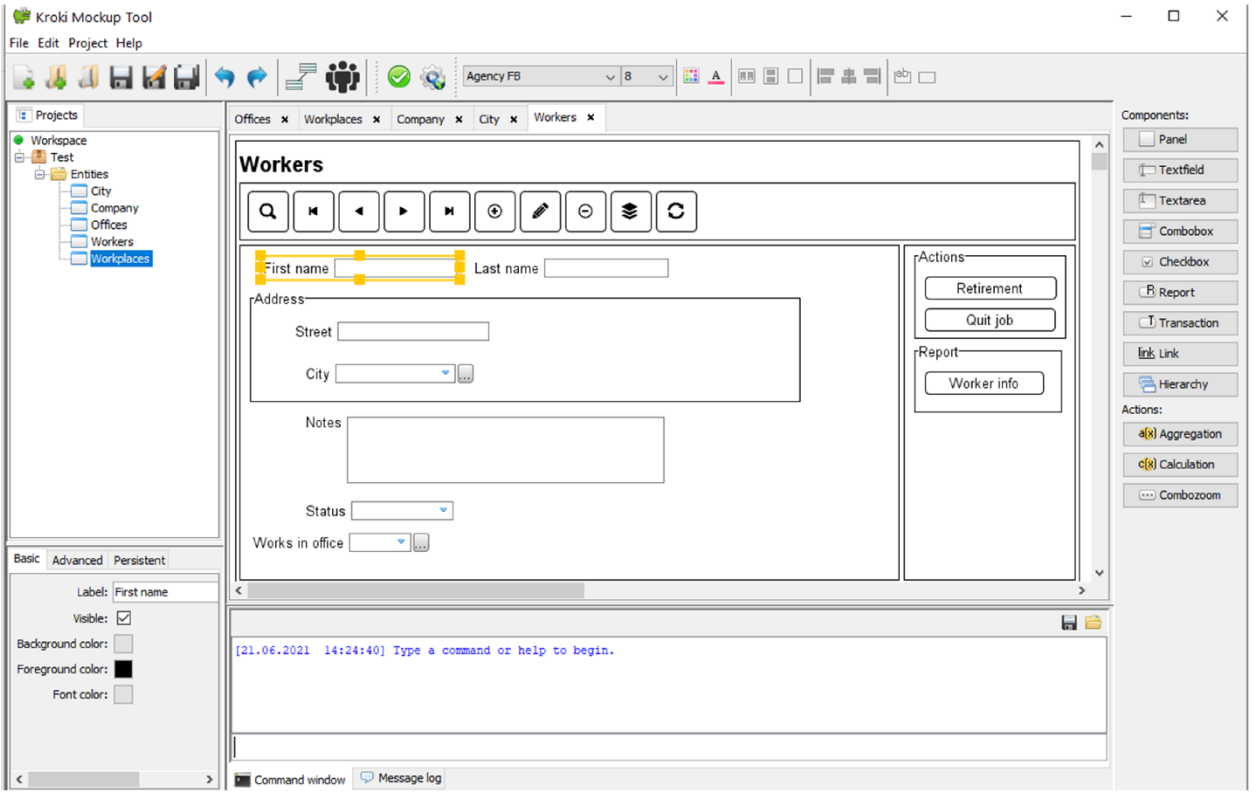
Za parsiranje formula u okviru izvedenih polja koristi se Dresden OCL parser. Dresden OCL [33] je alat otvorenog koda za pisanje i analizu OCL izraza, kao i generisanje Java, AspectJ i SQL koda na osnovu njih. Može se koristiti kao samostalna Java biblioteka ili kao razvojno okruženje u okviru Eclipse platforme. U okviru Kroki alata je iskorišćen samo OCL parser. Detalji implementacije su opisani u [28].

3.4 Grafičke konkretne sintakse EUIS DSL-a

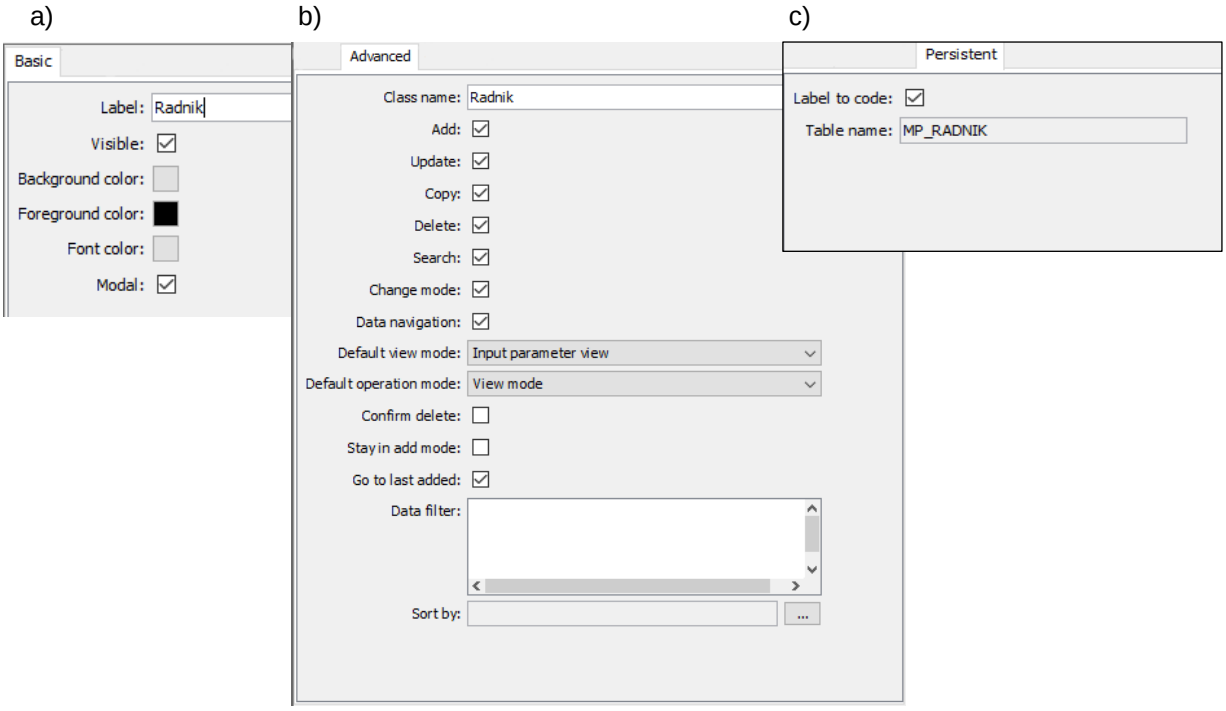
Editor za kreiranje skica (slika 32) implementira grafičku sintaksu EUIS DSL-a koja je namenjena kolaborativnom radu u sesijama zajedničkog modelovanja. Projektovan je tako da podseća na alate za crtanje skica korisničkog interfejsa, koje se smatraju kao pogodno sredstvo za komunikaciju sa korisnicima [1, 20, 31, 58].

Editor se sastoji od:

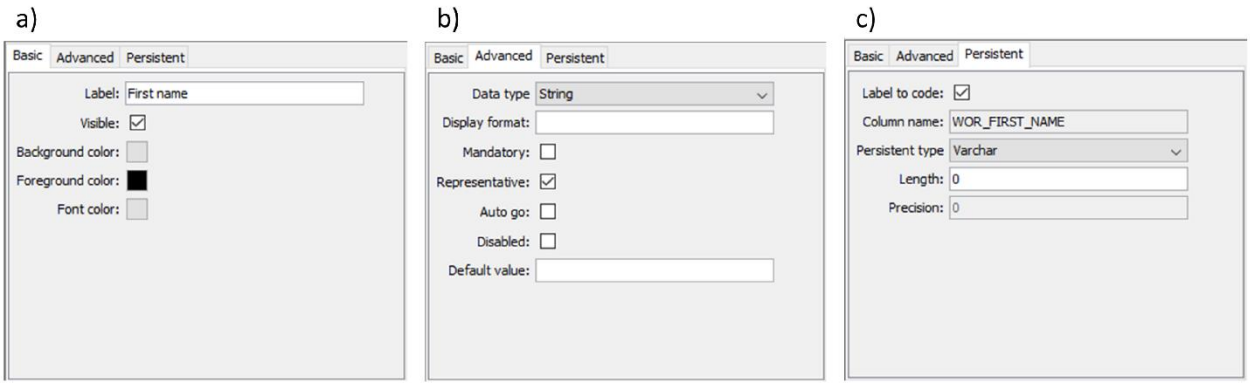
- Panela za prikaz strukture svih projekata u vidu stabla.
- Radne površine za skiciranje formi koja se sastoji od više jezičaka, po jedan za svaku aktivnu skicu na kojoj se radi. Elementi koji su bazirani na EUIS DSL „vidljivim“ obeležjima i operacijama se na skicu dodaju iz palete sa desne strane grafičkog editora.
- Panela za podešavanje osobina skica formi i njihovih elemenata. Ove osobine se mapiraju na atribute odgovarajuće meta-klase EUIS DSL-a čija instanca je trenutno selektovana u okviru radne površine. Panel je organizovan u tri jezička, za podešavanje osnovnih, naprednih i osobina perzistentnog sloja. Osnovna podešavanja se odnose na specifikaciju labela i boja, što bi mogao da obavi i korisnik koji učestvuje u kolaborativnoj sesiji sa razvojnim timom. Napredna i perzistentna podešavanja se odnose na implementacione detalje korisničkog interfejsa i perzistentnog sloja, za koje je zadužen razvojni tim (npr. imena klasa i atributa, tipovi podataka, dužina i preciznost, format za unos i sl). Ukoliko se ne popune, koriste se podrazumevane vrednosti, tako da se prototip uvek može pokrenuti. Podešavanja za standarnu formu i za „vidljivo“ obeležje su data na slikama 33 i 34, respektivno.
- Komandnog prozora koji implementira tekstualnu sintaksu EUIS DSL-a.
- Prozora za prikaz poruka koji ispisuje status izvršenih operacija i poruke o greškama, ako nastanu prilikom kreiranja modela ili pokretanja prototipa.



Slika 32 Editor skica Kroki alata

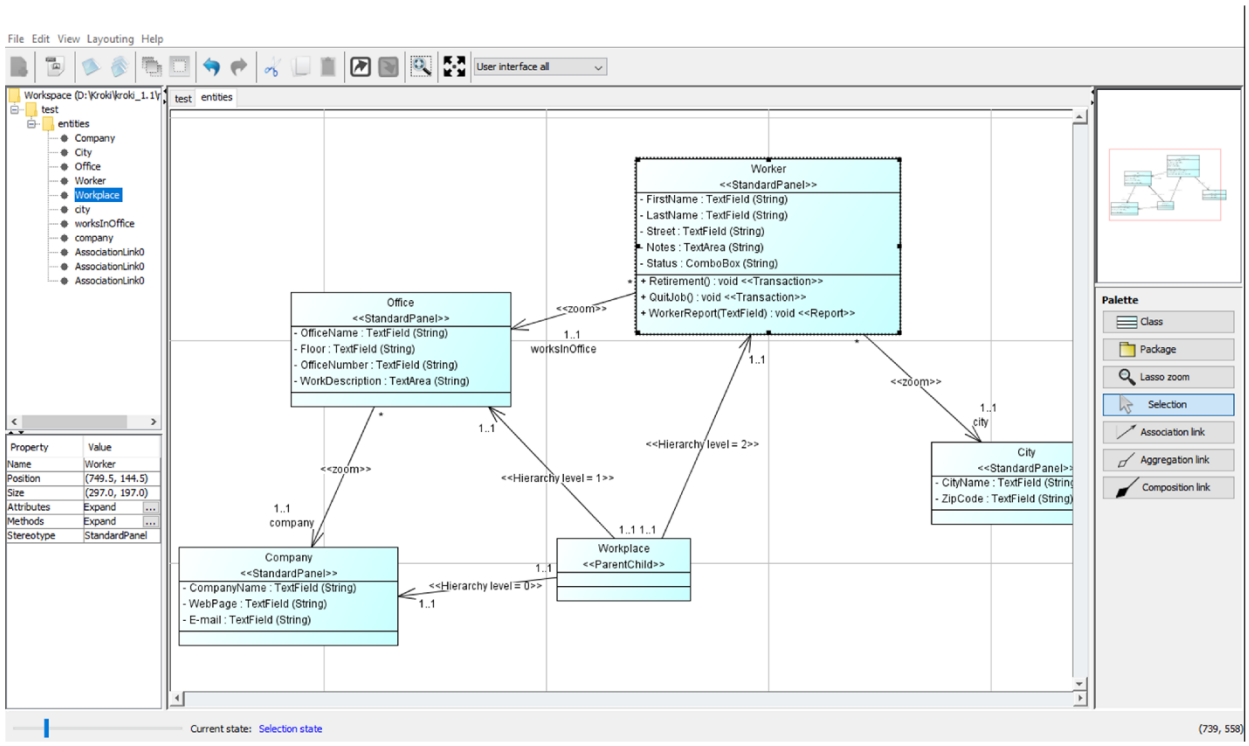


Slika 33 Panel za podešavanje a) osnovnih b) naprednih i c) perzistentnih osobina skica formi



Slika 34 Panel za podešavanje a) osnovnih b) naprednih i c) perzistentnih osobina „vidljivih“ obeležja – UI komponenti na formama

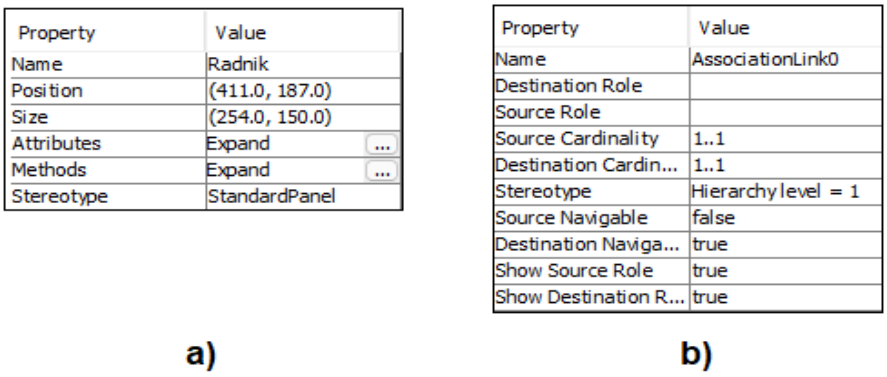
Pored editora u kojem se model kreira putem skica korisničkog interfejsa, Kroki poseduje i pojednostavljeni UML editor u kojem se isti model može kreirati korišćenjem sintakse koja podseća na dijagram klasa sa stereotipima koji odgovaraju konceptima EUIS DSL-a (slika 35). Prelazak iz jednog editora u drugi je moguć u svakom trenutku, uz automatsko raspoređivanje novounetih elemenata.



Slika 35 Prozor pojednostavljenog UML editora u okviru Kroki alata

Osnovni delovi pojednostavljenog UML editora su:

1. Panel za prikaz strukture trenutno otvorenog projekta.
2. Radna površina za modelovanje koja prikazuje klase selektovanog paketa iz stabla projekta. Na radnu površinu se dodaju elementi koji su raspoloživi u paleti sa desne strane radne površine.
3. Panel za podešavanje osobina selektovanog elementa modela (klase i veze) – slika 36.
4. Mapa UML dijagrama za podršku brze navigacije kroz modele koji nadilaze veličinu ekrana.



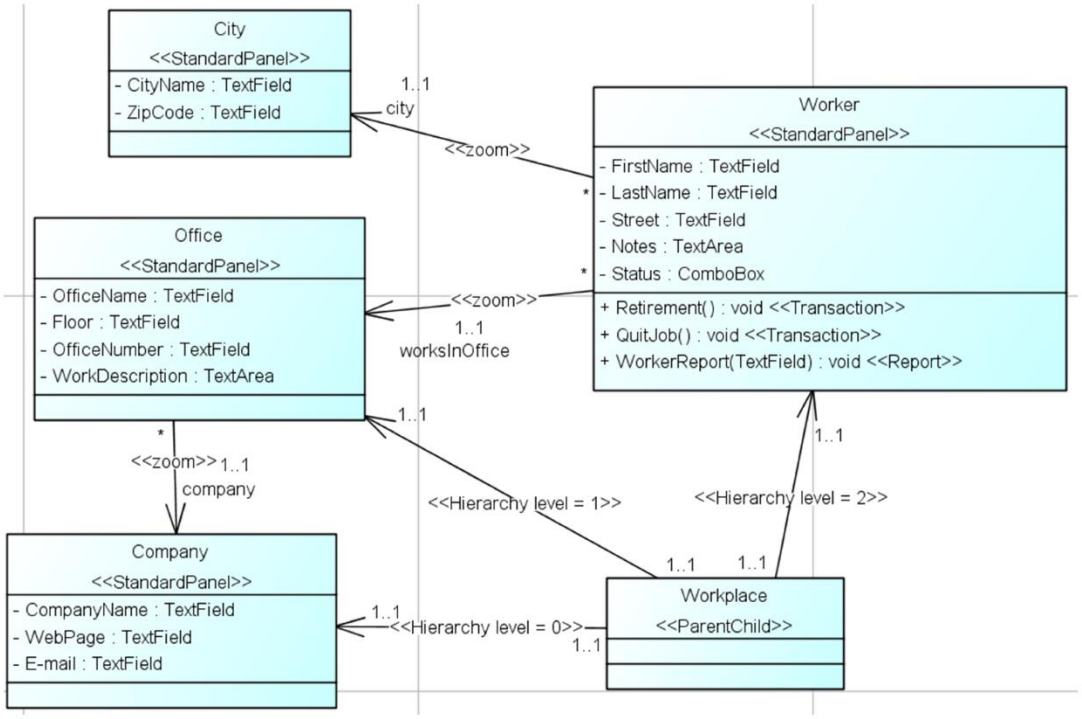
Slika 36 Paneli za podešavanje osobina a) UML klase i b) veze asocijacije u okviru UML editora Kroki alata

UML editor podržava automatsku prostornu raspodelu elemenata dijagrama, koji se koristi prilikom importa dijagrama kreiranih UML alatima opšte namene i prilikom raspoređivanja klasa unetih editorom skica ili komandnim okvirom. Na raspolaganju je više implementacija algoritama za prostorni raspored koji organizuju model tako da su klase i njihove veze jasno vidljive, bez obzira na veličinu modela. Više o implementiranim algoritmima se može naći u [153].

Način korišćenja datih editora i pokrenutog prototipa je prezentovan u četvrtom poglavlju, gde se može naći više detalja o elementima grafičkih sintaksi EUIS DSL-a i steći uvid u način funkcionisanja Kroki alata.

3.4.1 Primer korišćenja grafičkih sintaksi

Na slici 37 je prikazana specifikacija poslovne aplikacije za raspoređivanje radnika (klasa Worker) po kancelarijama (klasa `Office`) u okviru jedne kompanije (klasa `Company`) u okviru pojednostavljenog UML editora.



Slika 37 Specifikacija raspoređivanje radnika po kancelarijama u kompaniji u okviru pojednostavljenog UML editora

Slika 38 Standardni panel za unos podataka o radnicima kojem odgovara klasa Worker sa slike 37

Na Slici 38 se može videti standardni panel za evidenciju podataka o radnicima. Na osnovu skice alatne trake koja se nalazi odmah ispod naziva panela, možemo videti da su omogućene sve standardne operacije predviđene EUIS jezikom. Za svakog radnika je omogućena evidencija imena, prezimena, adrese, kancelarije u kojoj radi, statusa radnika i zapažanja o njemu. Adresa je prikazana u okviru posebnog panela i sastoji od ulice i mesta stanovanja. Mesto stanovanja i kancelarija se popunjavaju podacima iz povezanih entiteta i pribavljaju putem *Combozoom* komponenti kojima je ciljni panel (*Target panel*) `City` i `Office`, respektivno. Kreiranjem *Combozoom* komponente koja pokazuje na drugu standardnu formu uspostavlja se veza asocijacije između ova dva entiteta u UML modelu koji se automatski kreira u pozadini ovog procesa (videti sliku 37).

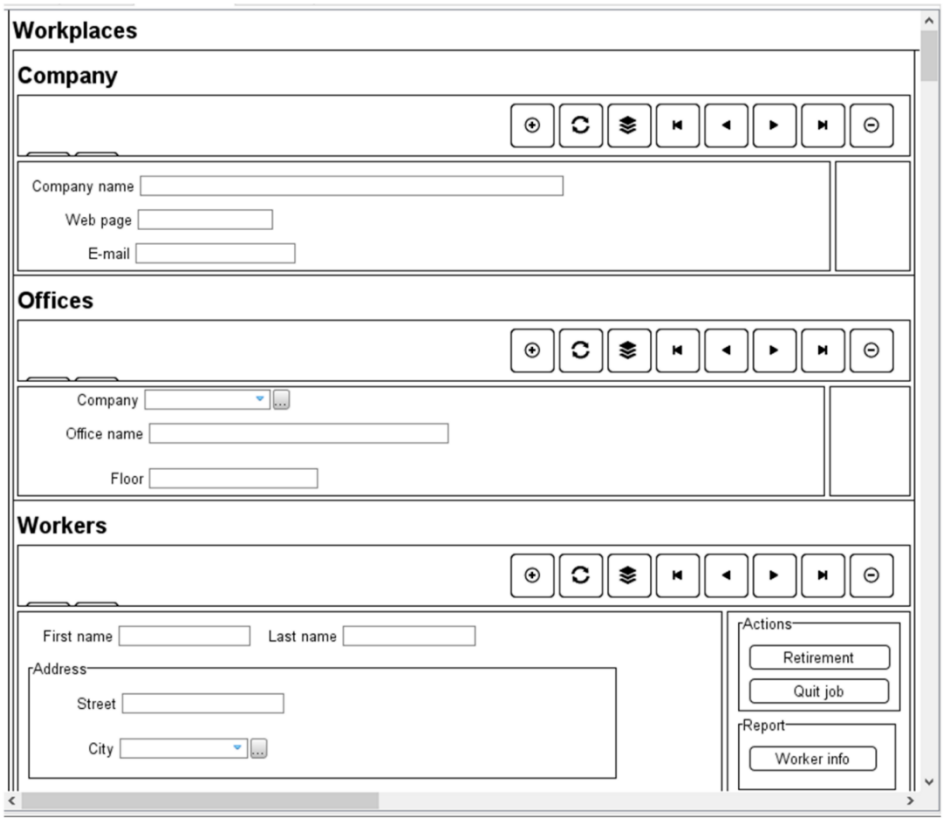
Sa desne strane skice se može videti da ova forma sadrži jedan poslovni izveštaj za prikaz podataka o radniku i dve transakcije, za penzionisanje radnika i prekid zaposlenja. Oni su prikazani kao metode klase Worker sa stereotipima `Report` i `Transaction`, respektivno, na slici 37.

Na slici 39 se nalazi skica „Parent-Child“ forme koja odgovara klasi `Workplace` sa stereotipom `ParentChild` sa slike 37. Na prvom nivou se nalaze podaci o kompaniji, na drugom kancelarije u okviru kompanije i na trećem nivou su radnici raspoređeni po kancelarijama. Na UML dijagramu se može primetiti da datim nivoima odgovaraju asocijacije sa stereotipom `Hierarchy` čiji su nivoi redom 0, 1 i 2.

Kada se pokrene izvršavanje prototipa na bazi ove specifikacije, forma za manipulaciju podataka o kancelarijama izgleda kao na slici 40. Kao što se može videti, forme u okviru prototipa kao i forme na skicama u okviru Kroki alata, odgovaraju EUIS specifikaciji i sastoje se od glavne alatne trake, dela za manipulaciju podacima i dela sa specifičnim operacijama.

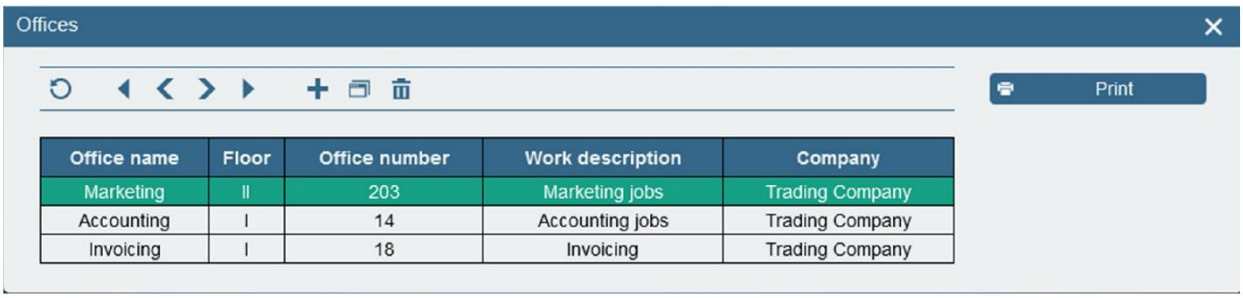
Prema internom UI standardu, standardne forme poseduju dva režima prikaza: režim unosa (input view) i tabelarni režim (table view), pri čemu se u okviru Kroki alata za svaku formu može specificirati podrazumevani režim prikaza. Ukoliko podrazumevani režim nije specificiran, sve forme će se otvarati u tabelarnom režimu. Na slici 40 su prikazana oba režima. Pored polja za

unos kompanije se vidi zoom dugme koje aktivira formu pridruženu klasi `Company`, radi preuzimanja željenog reda.

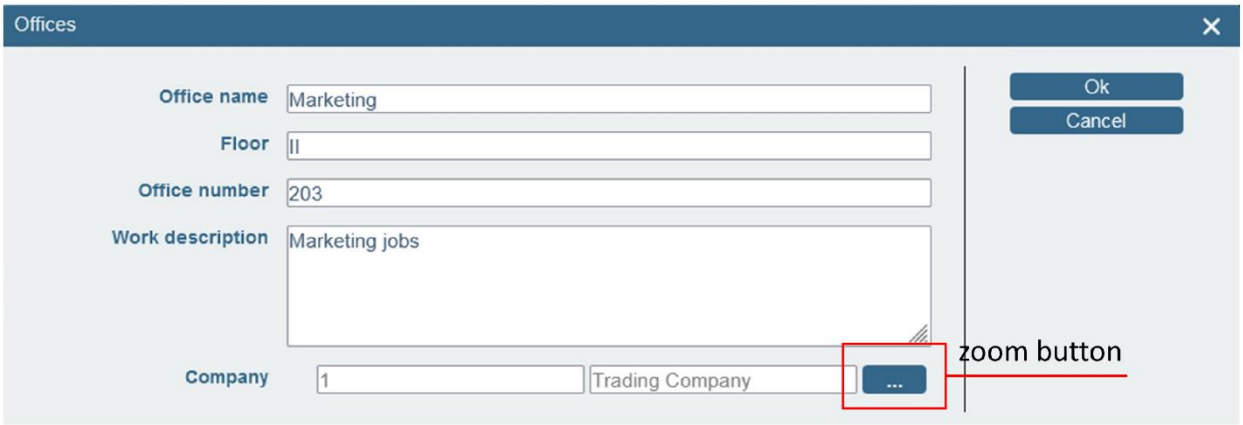


Slika 39 Primer Parent-Child forme na tri nivoa koja prikazuje radnike u okviru kancelarija koje pripadaju kompaniji

a)

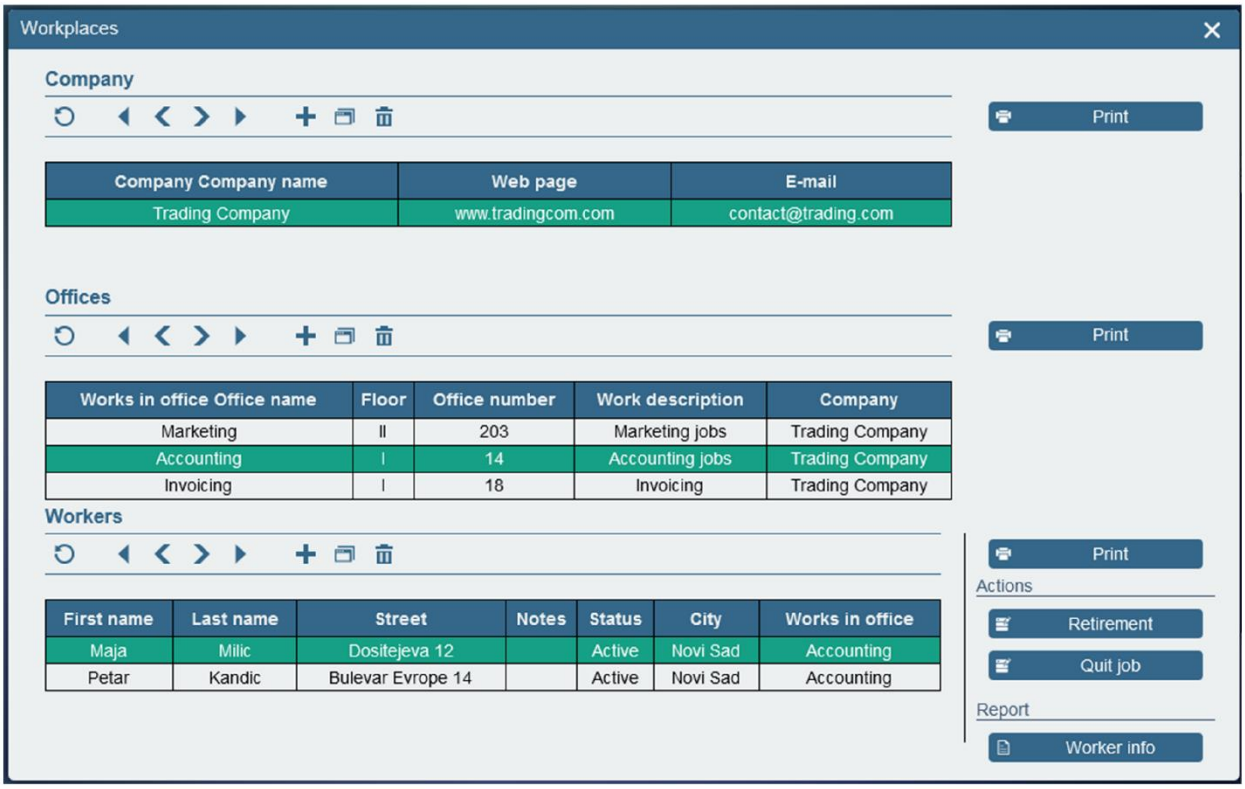


b)



Slika 40 Pokrenuta web forma vezana za klasu `Office` u a) tabelarnom prikazu b) u režimu unosa

Na slici 41 se nalazi pokrenuta „Parent-Child“ forma sa slike 39. U sklopu web prototipa „Parent-Child“ forme su implementirane tako da prate smernice definisane UI standardom i izgled specificiran na skicama. Podržan je rad *„Parent-Child“* formi sa proizvoljnim brojem nivoa hijerarhije koje podržavaju automatsko filtriranje podataka u panelima u odnosu na odabrane redove na višem nivou hijerarhije.



Slika 41 Pokrenuta „Parent-Child“ web forma koja odgovara skici sa slike 39

U sekciji 4.5 je dato više detalja o korišćenju pokrenutog prototipa. Tehničke karakteristike generičke web aplikacije za pokretanje prototipa su prezentovane u nastavku.

3.5. Tekstualni komandni jezik

Tekstualna sintaksa EUIS DSL-a omogućava kreiranje elemenata modela unošenjem i izvršavanjem tekstualnih komandi u okviru komandnog okvira koji se nalazi na donjoj strani radne površine editora skica. Ovaj režim rada je prilagođen korisnicima kojima više odgovara rad sa tastaturom i tekstualnim interfejsom u odnosu na korišćenje miša i grafičkih editora. Motivacija za implementaciju potiče od istraživanja koja ukazuju na veću efikasnost tekstualnih sintaksi u odnosu na grafičke kod određenih tipova korisnika [154, 155], kao i od prakse uključivanja tekstualnog režima u alate za modelovanje [156].

```
number ::= "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9" ;
letter ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" |
           "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" |
           "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" |
           "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;
whiteSpace ::= " ";
elementName ::= letter, { letter | number | whiteSpace } ;
textfieldDataType = String | Integer | Long | BigDecimal | Date ;
persistentTypes ::= Char | Varchar | Text | Integer | Number | Float | Decimal | Boolean  |
Date | Time | DateTime ;
components ::= {panelComponent | textfiledComponent | textareaComponent | comboboxComponent
| checkboxComponent | reportComponent | transactionComponent | linkComponent }
panelComponent ::= panel, Label, elementName ;
textfiledComponent ::= textfield, Label, elementName, DataType, textfieldDataType ;
textareaComponent ::= textarea, Label, elementName;
comboboxComponent ::= combobox, Label, elementName;
checkboxComponent ::= checkbox, Label, elementName;
reportComponent ::= report, Label, elementName;
transactionComponent ::= transaction, Label, elementName;
linkComponent  ::=  link,  Label,  elementName,  [TargetPanel,  elementName],  [TargetZoom,
elementName] ;
hierarchyComponent  ::=  hierarchy,  Label,  elementName,  [ActivationPanel,  elementName],
[TargetPanel, elementName];
combozoomComponent ::= combozoom, Label, elementName, [TargetPanel, elementName];
makeCommand  ::=  makeProject  |  makePackage  |  makeStdPanel  |  makeParentChildPanel  |
makeComponent | makeCombozoom | makeHierarchy | makeManyToManyPanel;
makeProject ::= make, project, '"', elementName, '"' ;
makePackage  ::=  make,  package,  '"',  elementName,  '"',  in,  '"',  elementName,  [{'/',
elementName}], '"' ;
makeStdPanel  ::=  make,  std-panel,  '"',  elementName,  '"',  in,  '"',  elementName,  [{'/',
elementName}], ['{', components, ',', '}'] ;
makeParentChieldPanel ::= make, parent, child, '"', elementName, '"', in, '"', elementName, [{'/', elementName}], '"'
makeManyToManyPanel ::= make, mtm-panel, '"', elementName, '"', in, '"', elementName, [{'/', elementName}], '"'
makeComponent ::= make, component, '{',{components, [',']} '}', in, '"', elementName, [{'/',
elementName}], '"' ;
makeCombozoom ::= make, combozoomComponent, in, '"', elementName, {'/', elementName}, '"' ;
makeHierarchy ::= make, hierarchyComponent, in, '"', elementName, [{'/', elementName}], '"'
;
```

Listing 3 Tekstualna sintaksa EUIS DSL-a za kreiranje elemenata modela

Parser za prvu verziju tekstualne sintakse EUIS DSL-a namenjene za kreiranje instanci EUIS meta-modela je implementirana u okviru [171] korišćenjem ANTLR [170] parser generatora. Listing 3 prikazuje datu sintaksu iskazanu proširenom Bakus-Naurovom formom. Parser je kreiran kao nezavisna komponenta, a za pisanje programa na datom jeziku je korišćen proizvoljni tekstualni editor.

U okviru ove disertacije je razvijen komandni okvir koji se koristi za unos i izvršavanje komandi u okviru Kroki alata. Sastoji se od:

1. linije za unos i pokretanje komandi,
2. prozora za prikaz poruka i rezultata izvršavanja,
3. alatne trake koja omogućava snimanje i učitavanje datoteke sa komandama.

Efikasnost rada je podržana putem pamćenja istorije komandi koje se mogu ponovo ispisati putem navigacionih strelica na tastaturi i automatskim dopunjavanjem komandi korišćenjem tastera Tab.

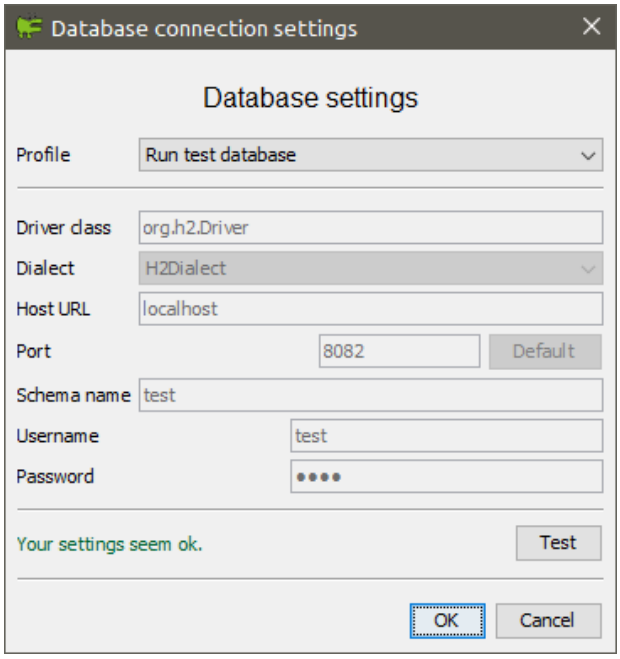
Komandni okvir omogućava spremanje svih unesenih komandi u tekstualnu datoteku sa ekstenzijom *.parse*, kao i učitavanje ovakve datoteke. Spremljena datoteka će sadržavati sve komande koje su unete od pokretanja Kroki alata, po jednu u redu. Prilikom učitavanja, svaka snimljena komanda će biti izvršena redom kako je navedena, tako da će Kroki projekat biti rekonstruisan. Prednost snimanja projekta u tekstualnoj datoteci je sledeća:

1. Čitljiva je za ljude.
2. Olakšana je analiza i otklanjanje grešaka u modelu. Projektanti mogu, uvidom u sadržaj ove datoteke, steći sliku o toku procesa modelovanja i dobiti informacije o svakom preduzetom koraku.
3. Pogodna je za deljenje i predstavlja materijal za učenje i razmenu delova modela.
4. Pogodna je za korišćenje u sistemima za kontrolu verzija, što može omogućiti paralelan rad više grupa projektanata na istom modelu.

3.6. Podrška za izvršavanje prototipa

Radi mogućnosti kolaborativnog razvoja sa korisnikom gde se u svakom trenutku može proveriti uzajamno razumevanje na bazi izvršivog prototipa, efikasno pokretanje je od ključne važnosti. Podrška za efikasno pokretanje je implementirana na više nivoa:

- Zahvaljujući dizajnu EUIS DSL-a koji integriše UI i perzistentne elemente modela, izbegnuto je trošenje vremena na M2M transformacije.
- Korišćenjem adaptivnih generičkih aplikacija koje dinamički prilagođavaju svoj izgled i funkcionalnost na osnovu podataka preuzetih iz modela, u velikoj meri je skraćeno vreme koje se troši na M2T transformacije. Adaptivne generičke aplikacije su unapred implementirane, tako da se većinom generišu konfiguracione datoteke koje diktiraju izgled i funkcionalnosti prototipa. Ovi podaci se smeštaju u aplikativni repozitorijum koji je optimizovan za brz pristup. Podrška za unos ručnih izmena je obezbeđena putem aspekt-orijentisanog programiranja.
- H2 sistem za rukovanje bazama podataka koji je uključen u Kroki alat obezbeđuje perzistenciju unetih podataka tokom korišćenja prototipa, bez potrebe za dodatnim konfigurisanjem. Međutim, podržano je i korišćenje proizvoljnog sistema za rukovanje bazama podataka, ukoliko razvojni tim tako odabere. Na slici 42 je prikazan dijalog za podešavanje konekcije za proizvoljnu bazu podataka.



Slika 42 Dijalog za podešavanje konekcije za proizvoljnu bazu podataka

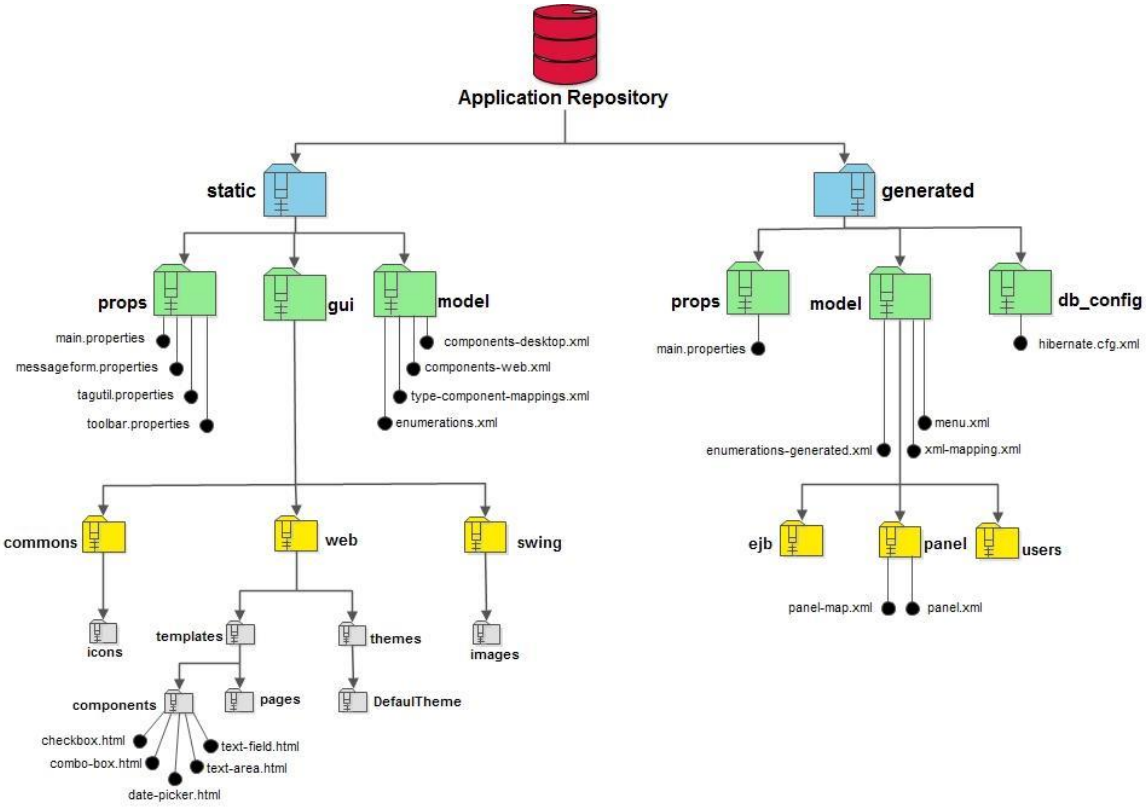
U nastavku su prikazani detalji implementacije aplikativnog repozitorijuma i generisanja potrebnih datoteka, dat je kratak pregled aspekt-orijentisanog programiranja i ApectJ platforme i objašnjen je način funkcionisanja generičke web aplikacije za izvršavanje prototipa.

3.6.1. Aplikativni repozitorijum

Aplikativni repozitorijum sadrži aplikativne i administrativne podatke adaptivne aplikacije i predstavlja podlogu za njeno dinamičko izvršavanje. U aplikativne podatke spadaju opis strukture poslovnih entiteta i njihovih međusobnih veza, kao i opis prostornog rasporeda grafičkih elemenata na panelima. Administrativne podatke čine opis korisničkih uloga i grupa u sklopu poslovnog sistema i njihovih prava nad elementima aplikacije, struktura radnog okruženja za svaku grupu i slično. Aplikativni repozitorijum može biti implementiran kao skup datoteka, baza podataka, ili neki drugi vid organizacije podataka kojem adaptivne aplikacije imaju pristup. U okviru Kroki alata je implementiran kao folder koji je podeljen na dva dela (slika 43):

1. Statički deo – sadrži resurse koji su zajednički za sve Kroki projekte.
2. Generisani deo – koristi se za smeštanje resursa koji se kreiraju od strane Kroki generatora na osnovu trenutno aktivnog Kroki projekta. Sadržaj ovog dela repozitorijuma će biti obrisan i ponovo kreiran sa svakim pokretanjem prototipa.

Sa slike se može videti da je veći deo repozitorijuma statički, tako da su resursi potrebni na generisanje prototipa svedeni na minimum, u cilju efikasnijeg pokretanja prototipa.



Slika 43 Struktura aplikativnog repozitorijuma

3.6.1.1. Statički deo repozitorijuma

Datoteke u statičkom delu repozitorijuma služe za konfigurisanje generičkih adaptivnih web i desktop aplikacija. Specifične su za konkretnu tehnološku platformu na kojoj su date aplikacije razvijene. Statički deo repozitorijuma nije predviđen da se menja, osim u cilju poboljšavanja izgleda ili funkcionisanja prototipa u toku izvršavanja. Promena ovog dela bi trebalo da je uslovljena izdavanjem nove verzije generičkih aplikacija ili cele Kroki platforme i ne bi trebalo da utiče na rad modela razvijenih sa starijim verzijama.

Statički deo repozitorijuma je podeljen na tri osnovna direktorijuma:

1. `props` – sadrži Java *properties*[26] datoteke sa različitim podešavanjima generičkih aplikacija. Osnovne datoteke u ovom direktorijumu su:
    a. `main.properties` – sadrži osnovna podešavanja generisanog prototipa kao što su podaci za prijavu podrazumevanog korisnika i ime vizuelne teme,
    b. `message.properties` – sadrži tekstualne resurse koji se koriste u okviru korisničkog interfejsa generičkih aplikacija,
    c. `tagutil.properties` – sadrži interna imena (ključeve) za sve XML tagove koji se koriste u sistemu, tako da se XML datoteke parsiraju u odnosu na unose u ovoj datoteci, a ne na imena konkretnih tagova u njima. Ovaj pristup omogućava laku izmenu XML specifikacije ukoliko za time ima potrebe,
    d. `toolbar.properties` – Sadrži tekstualne resurse koji se koriste na alatnim trakama formi u generisanim prototipovima.

[26] https://docs.oracle.com/javase/tutorial/essential/environment/properties.html

2. `gui` – sadrži statičke resurse koji definišu izgled generičkih aplikacija. Na Slici 43 se može videti da se sastoji od tri pod-direktorijuma: *commons*, *web* i *swing*. Direktorijum *commons* sadrži grafičke resurse zajedničke za obe generičke aplikacije, dok *swing* i *desktop* sadrže resurse specifične samo za date platforme. Direktorijum *web* se sastoji od sledećih pod-direktorijuma:
    a. `templates` – sadrži Java Freemarker[27] šablone za sve komponente i HTML stranice koje se dinamički kreiranju prilikom pokretanja prototipa. Pod-direktorijum `components` sadrži šablone za UI komponente koja se nalaze na formama. Generička aplikacija pronalazi odgovarajući šablon za svaku komponentu iz Kroki modela na osnovu mapiranja koja su navedena u *model* direktorijumu, koji će biti predstavljen u nastavku.
    b. `themes` – Sadrži pod-direktorijume za svaku vizuelnu temu koja je na raspolaganju za korišćenje u web aplikaciji. Ime trenutno korišćene teme se navodi u konfiguracionoj datoteci *main.properties* i taj unos mora da odgovara imenu direktorijuma teme. U trenutnoj verziji Kroki alata na raspolaganju je samo jedna web tema, nazvana `DefaultTheme`. Uključivanje nove vizuelne teme bi podrazumevalo kreiranje novog pod-direktorijuma i odgovarajućih datoteka (prateći strukturu osnovne teme) i ručnog menjanja unosa u `main.properties` datoteci. Kroki alat trenutno ne poseduje mogućnost odabira vizuelne teme u toku kreiranja modela, ali se to planira kao jedno od unapređenja alata.
3. `model` – Statička verzija ovog direktorijuma sadrži XML specifikacije različitih artefakata i gradivnih elemenata generičkih aplikacija kao i konteksta u okviru kojeg se oni koriste. Osnovne datoteke koje ovo omogućuju su:
    a. `type-component-mapping.xml` – specificira koje UI komponente u generisanom prototipu će biti korišćene u zavisnosti od tipa podataka obeležja definisanog u Kroki modelu. Kako bi se omogućilo korišćenje izabranih UI komponenti za isti tip podatka, neka od imena tipova su dopunjena podatkom o prirodi polja (npr. *java.lang.String:ComboBox* ili *java.lang.String:TextArea*) a takođe postoje i imena tipova koji su specifični za Kroki okruženje, kao što su *kroki.joinColumn*. Primer ove datoteke je dat na Listingu 4 na kojem se može videti da se za komponente koriste interna imena koja se zatim mapiraju na konkretne nazive komponenti u datotekama za web i desktop aplikacije koje su navedene u nastavku.
    b. `components-web.xml` – specificira na koji način će se, u okviru web generičke aplikacije, imena UI komponenti definisana u datoteci *type-component-mapping.xml* dalje mapirati na konkretne HTML šablone u pokrenutoj aplikaciji. Ovo predstavlja poslednji korak u kreiranju odgovarajuće HTML komponente u odnosu na tip podatka kojim ta komponenta manipuliše. Sadržaj ove datoteke je prikazan na Listingu 5.
    c. `components-desktop.xml` – služi za mapiranje internog imena UI komponenti na imena Java Swing klasa koje će se koristiti za instanciranje odgovarajuće Java UI komponente na formama desktop generičke aplikacije.
    d. `enumerations.xml` – sadrži XML specifikaciju nabrojanih tipova definisanih EUIS DSL-om koji se koriste u okviru prototipova.

[27] https://freemarker.apache.org/

```
<map>
    <property language-type="java.lang.String" component-id="textField" />
    <property language-type="java.lang.String:TextArea" component-id="textArea" />
    <property language-type="java.lang.String:ComboBox" component-id="comboBox" />
    <property language-type="java.lang.Long" component-id="textField" />
    <property language-type="java.lang.Integer" component-id="numberField" />
    <property language-type="java.math.BigDecimal" component-id="textField" />
    <property language-type="java.util.Date" component-id="datePicker" />
    <property language-type="java.lang.Boolean" component-id="checkBox" />
    <property language-type="kroki.joinColumn" component-id="zoomFiled" />
    <property language-type="java.lang.String:Email" component-id="emailField" />
</map>
```

Listing 4 Sadržaj datoteke *type-component-mappings.xml*

```
<components>
    <component id="textField"   template-file="text-field.html"/>
    <component id="textArea"    template-file="text-area.html"/>
    <component id="checkBox"    template-file="checkbox.html"/>
    <component id="comboBox"    template-file="combo-box.html"/>
    <component id="datePicker"  template-file="date-picker.html"/>
    <component id="zoomFiled"   template-file="zoom-field.html"/>
    <component id="numberField" template-file="number-field.html"/>
    <component id="emailField"  template-file="email-field.html"/>
</components>
```

Listing 5 Sadržaj datoteke *components-web.xml*

3.6.1.2. Generisani deo repozitorijuma

Generisani deo Kroki aplikativnog repozitorijuma podseća na strukturu njegovog statičkog dela i sadrži dopune statičkih datoteka koje zavise od trenutnog Kroki projekta. Ovaj deo ne sadrži *gui* direktorijum, s obzirom da je izgled poslovnih sistema definisan internim UI standardom. Osnovni elementi generisanog dela Kroki aplikativnog repozitorijuma su:

1. `props` – sadrži generisanu verziju *main.properties* datoteke u koju se smešta ime i opis Kroki projekta koji se trenutno pokreće.
2. `db_config` – sadrži generisanu *hibernate.cfg.xml* datoteku koja služi za konfigurisanje Java Hibernate[28] alata za objektno-relaciono mapiranje koji se koristi u obe verzije adaptivne aplikacije. Parametri za generisanje ove datoteke se preuzimaju iz Kroki dijaloga za konfigurisanje baze podataka (Slika 42).
3. `model` – Sadrži XML specifikacije svih gradivnih elemenata trenutnog Kroki projekta. Osnovni direktorijumi i datoteke koji se ovde generišu su:
    a. `ejb` – XML specifikacije svih modelovanih entiteta na osnovu kojih se generišu Java EJB [34] perzistentne klase. Za svaki entitet se generiše po jedna XML datoteka.
    b. `panel` – sadrži XML specifikaciju svake modelovane forme.
    c. `enumerations-generated.xml` – sadrži specifikaciju svih nabrojanih tipova koji su kreirani prilikom modelovanja *Combobox* komponenti na Kroki formama. Na

[28] https://hibernate.org/

osnovu ovih podataka će biti generisane Java enumeracije koje se koriste u generičkim aplikacijama.

d. `menu-generated.xml` – specifikacija glavnog menija generisanog prototipa. Struktura glavnog menija odgovara strukturi poslovnih podsistema (paketa) iz Kroki projekata, ali se može izmeniti koristeći ugrađeni Kroki alat za dizajn glavnog menija.
e. `users` – sadrži podatke o modelovanim korisnicima sistema i njihovim pravima, ukoliko je to specificirano Kroki alatom za upravljanje administracijom korisnika.

Kao što se može videti iz prikazane strukture, Kroki aplikativni repozitorijum ne sadrži programski kod, već samo konfiguracione datoteke, koje se učitavaju prilikom pokretanja generičkih aplikacija, koje na osnovu njih dinamički prilagođavaju svoj izgled i funkcionalnost.

3.6.2. Proces generisanja podataka za izvršavanje prototipa

Proces dobijanja izvršivog prototipa započinje preuzimanjem podataka iz Kroki modela i prevođenjem u format pogodan za upotrebu od strane FreeMarker obrađivača šablona. Ovaj deo posla se odvija u okviru klase `ProjectExporter`. Kada korisnik pokrene akciju generisanja prototipa, rekurzivno se prolazi kroz instancu odabranog Kroki projekta i popunjavaju se prikazane lista podacima o odgovarajućim delovima modela i pozivaju generatorske klase koja će popuniti nedostajuće delove aplikativnog repozitorijuma i generičke aplikacije na osnovu prosleđenih podataka.

Implementacija konstruktora ove klase je prikazana na Listingu 6 na kojem se mogu videti sve strukture podataka u koje se smeštaju podaci iz modela.

```
public ProjectExporter() {
	// Data structures that contain Kroki project data
	classes = new ArrayList<EJBClass>();
	menus = new ArrayList<Menu>();
	elements = new ArrayList<VisibleElement>();
	enumerations = new ArrayList<Enumeration>();
	rootMenu = new kroki.app.menu.Submenu("Menu");
	// Generator class
	appRepoGenerator = new ApplicationRepositoryGenerator();
}
```

Listing 6 Konstruktor klase `ProjectExporter`

Za sprovođenje procesa generisanja konfiguracionih datoteka i programskog koda koji je potreban generičkim aplikacijama, zadužena je glavna generatorska klasa nazvana `ApplicationRepositoryGenerator`. Proces generisanja započinje pozivom `generate` metode (Listing 7).

```
public void generate(
        ArrayList<EJBClass> classes,
        ArrayList<Menu> menus,
        ArrayList<VisibleElement> elements,
        ArrayList<Enumeration> enumerations,
        Submenu rootMenu) {
        DBConfigGenerator.geneateHibernateConfigXML(
                        "ApplicationRepository\generated\db_config\hibernate.cfg.xml"
        );
        DBConfigGenerator.generatePersistenceXMl();
        EJBGenerator.generateEJBClasses(classes);
        ConstraintGenerator.generateConstraints(classes);
        EJBGenerator.generateEJBXmlFiles(
                        classes, "ApplicationRepository\generated\model\ejb"
        );
        EJBGenerator.generateXMLMappingFile(classes);
        menuGenerator.generateMenu(menus);
        menuGenerator.generateNewMenu(rootMenu);
        panelGenerator.generate(elements);
        enumGenerator.generateXMLFiles(enumerations);
}
```

Listing 7 Metoda **generate** klase **ApplicationRepositoryGenerator**

Ova metoda se poziva iz klase `ProjectExporter` i kao parametri joj se prosleđuju odgovarajuće Java liste popunjene podacima iz Kroki modela. Možemo videti i da klasa `ApplicationRepositoryGenerator` služi kao centralno mesto za upravljanje ostalim generatorskim klasama - njen osnovni zadatak je pozivanje odgovarajućih metoda generatorskih klasa i prosleđivanje adekvatne strukture podataka. U trenutnoj verziji Kroki alata implementirane su sledeće generatorske klase:

1. `EJBGenerator` - zadužena je za generisanje podataka o perzistentnim entitetima iz Kroki projekta. Ona na osnovu liste `EJBClass` objekata kreira sledeće datoteke:
    a. *Eterprise Java Bean*[29] klasu za svaku perzistentnu klasu iz modela.
    b. XML datoteku sa opisom svake od perzistentnih klasa.
    c. XML datoteku *`xml-mapping.xml`* koja sadrži podatke o tome koja XML datoteka odgovara kojoj EJB Java klasi.
2. `DatabaseConfigGenerator` – Generiše datoteke *`persistance.xml`* i *`hibernate.cfg.xml`* na osnovu konfigurisanih podataka za konekciju na bazu podataka koja se specificira putem dijaloga na Slici 42.
3. `PanelGenerator` – Za svaki vidljivi element u Kroki modelu generiše XML unos u datotekama *`panel.xml`* i *`panel-map.xml`* u direktorijumu *model* generisanog dela aplikativnog repozitorijuma (Slika 43).
4. `MenuGenerator` – Generiše *`menu.xml`* datoteku koju generičke aplikacije koriste za konstruiranje glavnog menija za pristup formama. Kroki alat omogućava korisnicima da specificiraju željenu organizaciju menija kako bi generisani prototip u što većoj meri odgovarao ciljnom sistemu. Ukoliko korisnička struktura menija nije specificirana, struktura glavnog menija će odgovarati strukturi paketa iz Kroki projekta.
5. `EnumerationGenerator` – Ova generatorska klasa je zadužena za generisanje XML specifikacije Java nabrojanih tipova koji će se koristiti u okviru generisanog prototipa na osnovu navedenih vrednosti za *Combobox* komponente na Kroki modelu.

[29] https://docs.oracle.com/javaee/7/api/javax/ejb/EJB.html

6. `ConstraintGenerator` – Za svaku generisanu EJB klasu generiše pridruženu Java klasu koja sadrži Java implementaciju modelovanih OCL ograničenja za dati entitet.

Kao što se može videti, jedini programski kod koji se generiše u ovom procesu su Java EJB klase, dok se ostali ulazni podaci specificiraju u XML obliku. Ovo čini generičke aplikacije nezavisnim od načina specifikacije ulaznih podataka i, ako bi se klasa `EJBGenerator` izmestila tako da radi u okviru generičkih aplikacija, one bi bile nezavisne od Kroki alata. Ovakav način implementacije čini generičke aplikacije lako zamenjivima u slučaju potrebe.

3.6.3. Aspekt-orijentisano programiranje

Aspekt-orijentisano programiranje (AOP) [177] predstavlja programsku paradigmu koja ima za cilj razdvajanje nadležnosti unutar programskog rešenja uvođenjem novih programskih jedinica – aspekata. Ima jasno definisanu strukturu i pravila i koristi se kao nadgradnja nekog objektno-orijentisanog (OO) programskog jezika.

Potreba za uvođenjem aspektne paradigme javila se usled potrebe za boljim razdvajanjem nadležnosti koje objektna paradigma nije u nekim slučajevima mogla da postigne na jednostavan način. Iako se u OO programiranju teži razdvajanju nadležnosti unutar klasa, postoji određena funkcionalnost koju je teško delegirati konkretnoj klasi, već se provlači kroz celu aplikaciju (praćenje događaja i njihovo evidentiranje unutar aplikacije, obrada grešaka, perzistencija objekata i sl).

AOP predstavlja rešenje za ovakav tip problema. Ideja je da omogući da se određena funkcionalnost (npr. obrada grešaka) implementira na jednom mestu, u okviru aspekta, a da postoji mogućnost definisanja načina na koji se data funkcionalnost kombinuje sa ostatkom koda aplikacije (npr. svaki put kada se negde pojavi greška, nezavisno od njenog mesta).

Osnovni koncept u AOP su aspekti. Aspekti predstavljaju osnovnu jedinicu modularizacije, pri čemu se jedan aspekt odnosi na jednu ili više srodnih funkcija u sklopu sistema u kojem je definisan. Aspekti se pišu nezavisno od ostatka programskog koda aplikacije, ali se kasnije komponuju zajedno sa njim pomoću korišćenjem *weaver-a*. Na aspekte možemo gledati kao na klase u OO programiranju u smislu da predstavljaju osnovne jedinice programskog koda. Međutim, za razliku od OO paradigme, AOP se ne zasniva na razmeni poruka između aspekata, već na izdvajanju ponavljajućih zadataka koji bi inače bili rasuti po objektnom modelu. Aspekti se, za razliku od klasa, ne mogu instancirati, već za svaki aspekat postoji tačno jedna instanca u toku životnog ciklusa aplikacije.

Svaki aspekt definiše tzv. “tačke spajanja” (*joinpoints*) sa osnovnim programom. Tačka spajanja može da bude bilo koji trenutak u toku izvršavanja programa. Uglavnom su to pozivi nekih funkcija, inicijalizacija promenjivih, instanciranje klase, aktiviranje grešaka itd. Kada se desi neki događaj koji odgovara tački spajanja određenog aspekta, na tom mestu će se, uz kod samog programa izvršiti naredbe definisane u „advice” delu aspekta. *Advice* delovi predstavljaju blokove programskih instrukcija koje će biti izvršene u tački životnog ciklusa programa na koju se odnosi pripadajuća tačka spajanja. Pored samog koda koji će biti izvršen, *advice* delovi definišu i vreme izvršenja u odnosu na pripadajuću tačku spajanja, tako da se instrukcije mogu izvršiti pre, umesto ili posle izvršenja koda iz osnovnog programa na koji se odnose. Pored ovih konstrukcija specifičnih za AOP, aspekt može da sadrži i atribute i funkcije, slično kao i klasa.

Aspekt-orijentisani pristup programiranju je podržan u gotovo svim programskim jezicima u vidu proširenja ili biblioteka. Najpoznatija implementacija AOP za Java programski jezik je AspectJ biblioteka, koja je korišćena i u ovom rešenju. AspectJ [178] predstavlja proširenje Java jezika za razvoj aspekt-orijentisanih programskih rešenja. Sintaksa AspectJ odgovara sintaksi Jave,

tako da se programski kod napisan pomoću AspectJ biblioteke može izvršavati na svakoj Java virtuelnoj mašini. Kod iz aspekata se komponuje sa Java kodom pomoću *weaver* komponente koja vrši povezivanje (uplitanje) aspekt koda sa kodom java programa.

Povezivanje se može vršiti pre kompajliranja koda ili na nivou Java byte-koda, što zavisi od implementacije AOP rešenja. Zahvaljujući ovome, pristup izvornom kodu nije uvek neophodan prilikom razvoja aspekata, što može biti korisno kod povezivanja sa aplikacijama za koje ne postoji izvorni kod. Pored osnovnih koncepata iz Jave, AspectJ podržava definisanje svih gore navedenih konstrukcija AOP pomoću posebnih programskih konstrukcija

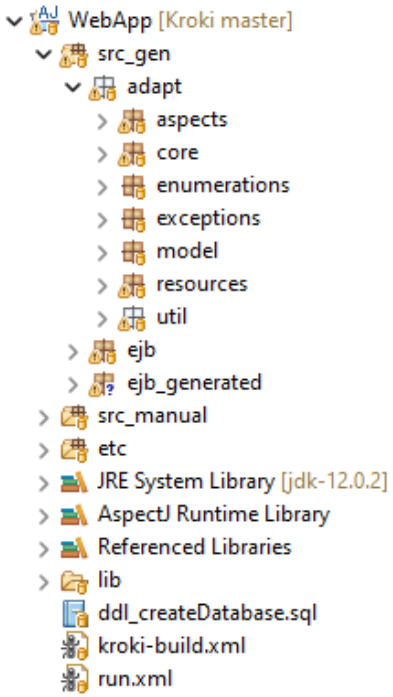
3.6.4. Generička web aplikacija

Generičke aplikacije predstavljaju softverska rešenja koja dinamički prilagođavaju svoj izgled i funkcionalnost u zavisnosti od konfiguracionih parametara. S obzirom na njihovu generičku prirodu, razvoj ovakvih aplikacija je složeniji u odnosu na razvoj konkretnih rešenja, tako da je potrebno obratiti pažnju kako bi se razvila kvalitetna podloga u vidu generičkih gradivnih elemenata koji odgovaraju pojedinačnim elementima meta-modela jezika koji podržavaju.

Kroki izvršivi prototip se dobija pokretanjem generičke java web aplikacije koja koristi podatke iz Kroki modela kako bi prilagodila svoj izgled i ponašanje u skladu sa razvijenom specifikacijom poslovne aplikacije. Kroki generičke aplikacije su projektovane da podrže standardizovani skup funkcionalnosti i vizuelnih aspekata iz domena poslovnih aplikacija specificiranih EUIS jezikom i internim UI standardom. Za potrebe Kroki platforme implementirane su dve adaptivne poslovne aplikacije: web verzija koja je trenutno u upotrebi i desktop verzija koja se više ne koristi, tako da će u ovoj disertaciji izvršavanje prototipa biti predstavljeno uz pomoć adaptivne web aplikacije.

Implementacija generičke adaptivne aplikacije je bazirana na Restlet[30] tehnologiji koja dolazi u vidu Eclipse projekta koji se distribuira zajedno sa Kroki programskim kodom. Ona koristi AspectJ [178] kao podlogu za uključivanje ručno pisanog koda i primenu prava korisnika. Slika 44 prikazuje strukturu Eclipse projekta koji sadrži programski kod ove aplikacije.

[30] https://restlet.talend.com/



Slika 44 Struktura Eclipse projekta generičke web aplikacije

U strukturi prikazanoj na Slici 44 može se videti da je projekat na osnovnom nivou podeljen na dva paketa: `src_gen` i `src_manual`. Paket `src_gen` sadrži kod koji je generisan u toku pokretanja Kroki prototipa, dok je paket `src_manual` inicijalno prazan i rezervisan je za ručne intervencije ukoliko za njima bude potrebe. Kao što je navedeno ranije, većina programskog koda ove aplikacije je već razvijena, tako da se sadržaj `src_gen` paketa većinski dobija kopiranjem unapred implementiranih Java direktorijuma i datoteka, dok se jedino generiše sadržaj paketa `ejb_generated`. Osim toga, bitno je skrenuti pažnju na nekoliko ključnih Java paketa i datoteka:

1. Paket `adapt.core`: sadrži inicijalnu logiku koja se izvršava prilikom pokretanja generičke web aplikacije. U ovom paketu se nalaze klase koje su zadužene za preuzimanje konfiguracionih podataka iz aplikativnog repozitorijuma, kao i za uspostavljanje i pokretanje web infrastrukture (baza podataka, interni web server) i automatsko otvaranje početne strane u web čitaču korisnika.
2. Paket `adapt.aspects`: U ovom paketu se nalaze ugrađeni Java aspekti koji su zaduženi za upravljanje korisničkim pravima i podacima za vreme trajanja njegove sesije u okviru web aplikacije. Postojanje ovih aspekata je posledica toga što odabrani Restlet radni okvir ne poseduje podršku za administraciju korisnika i održavanje korisničke sesije. Ukoliko se za izradu generičke aplikacije odabere neki drugi radni okvir koji podržava navedeno (npr. Java Spring[31] ili Django[32]), ručna implementacija ovih mehanizama nije potrebna.
3. Paket `adapt.model`: sadrži modelske klase koje implementiraju elemente EUIS DSL-a, kao i neke entitete bitne za pravilno funkcionisanje konkretne implementacije generičke aplikacije.

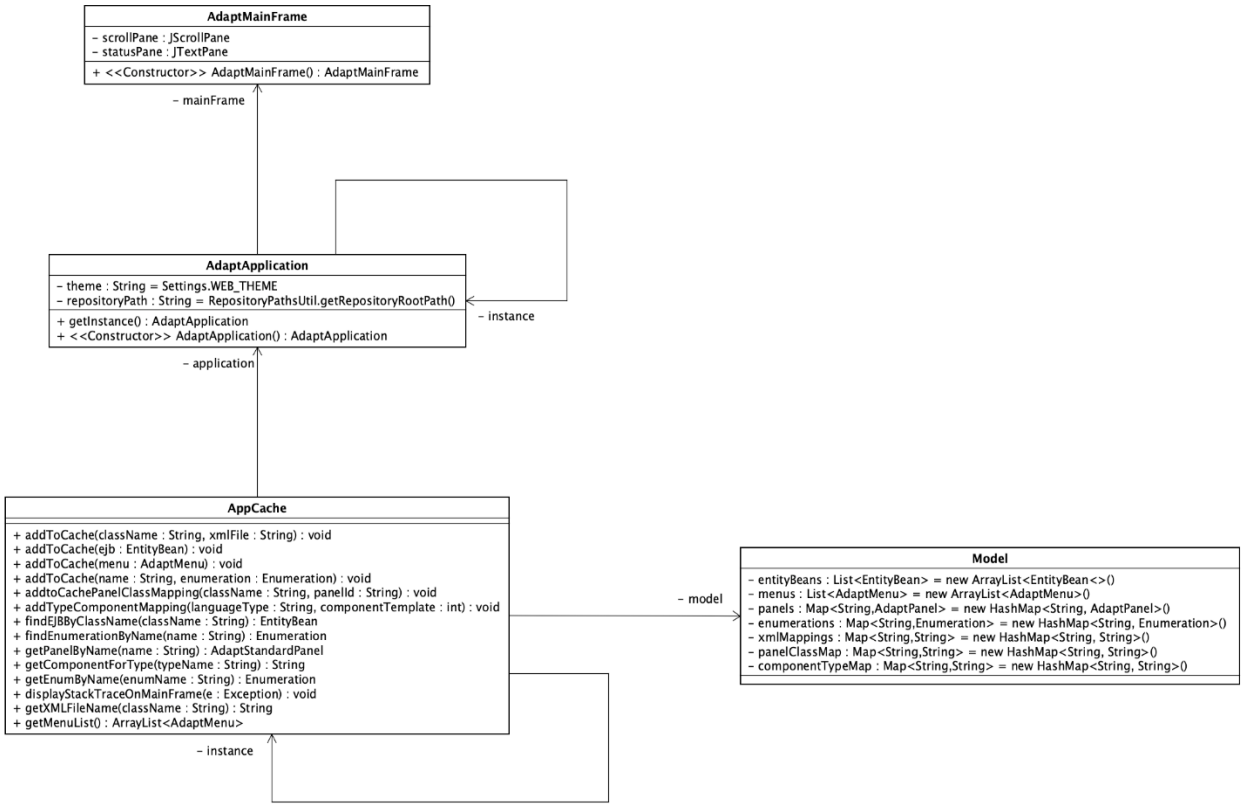
[31] https://spring.io/
[32] https://www.djangoproject.com/

4. Paketi `ejb` i `ejb_generated`: Sadrže EJB klase koje se koriste u projektu. U paketu `ejb` se nalaze klase koje predstavljaju ugrađene entitete koji su nezavisni od podataka u konfiguracionim datotekama, dok paket `ejb_generated` sadrži EJB klase koje predstavljaju entitete trenutno modelovane aplikacije koji se generišu na osnovu podataka iz *model* direktorijuma iz gen dela aplikativnog repozitorijuma.
5. Datoteka `kroki-build.xml`: Apache Ant[33] skripta koja vrši kompajliranje i pakovanje web projekta sa svim resursima iz aplikativnog repozitorijuma u Java arhivu koja je spremna za deljenje ili pokretanje
6. Datoteka `run.xml`: Skripta za pokretanje date Java arhive. Poziva se prilikom pokretanja prototipa iz Kroki alata.

UML dijagram klasa paketa `adapt.core` je dat na Slici 45. Osnovna Java klasa za ovu aplikaciju je klasa `AdaptMainFrame` iz ovog paketa koja ima dvojaku ulogu:

1. Realizuje Java prozor (instancu) klase `JFrame` koja će korisniku pružati osnovne informacije i prikaz poruka u toku rada web aplikacije. S obzirom da *Restlet* radni okvir ne obezbeđuje korisnički interfejs, ovaj prozor je kreiran po uzoru na konzolne prozore koji se prikazuju pokretanjem nekih serverskih aplikacija opšte namene (npr. *Apache Tomcat*). Primer ovog prozora je dat na Slici 59,
2. Inicijalizuje sve potrebne komponente za rad web aplikacije: Učitava podatke iz konfiguracionih datoteka aplikativnog repozitorijuma, uspostavlja sistem za rukovanje bazom podataka i pokreće Restlet serversku aplikaciju.

[33] https://ant.apache.org/

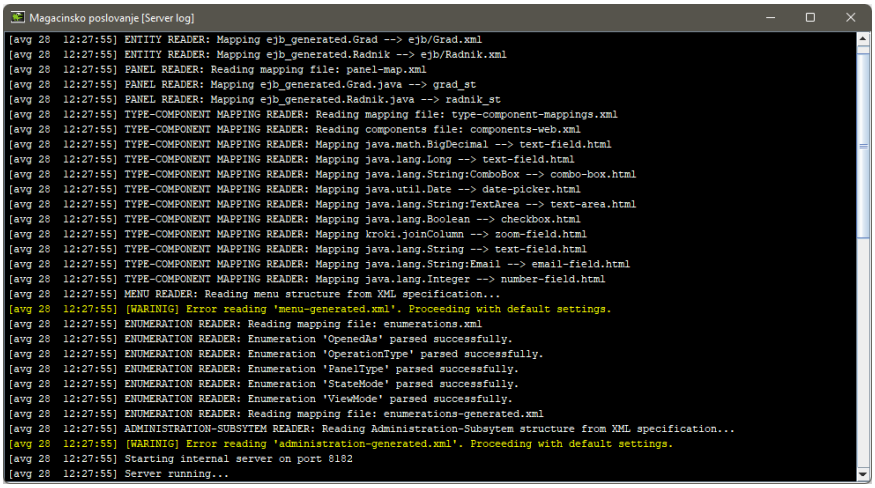


Slika 45 Klase iz adapt.core paketa generičke web aplikacije

Učitavanje podataka iz aplikativnog repozitorijuma je implementirano po principu *lazy loading*, što znači da se u trenutku pokretanja čitaju samo podaci o mapiranju (XML datoteke iz `model` direktorijuma) i specifikacija glavnog menija, dok se konkretni podaci o panelima i entitetima čitaju u trenutku kada se željena standardna forma prvi put aktivira. Ovaj način ubrzava pokretanje prototipa i pomaže u kolaborativnom radu sa korisnikom. U operativnoj memoriji se čuvaju samo oni podaci kojima je korisnik imao potrebe da pristupi. Jednom učitani podaci se čuvaju u instanci klase `AppCache` tako da nema potrebe za dodatnom komunikacijom sa serverom i čitanjem XML datoteka kada je potreban ponovni pristup. `AppCache` klasa je implementirana koristeći *Singleton* dizajn šablon [60] tako da je njena instanca jedinstvena na nivou cele aplikacije.

U okviru generičke web aplikacije implementirane su posebne klase za učitavanje podataka iz aplikativnog repozitorijuma koje se nalaze u okviru `adapt.util.xml_readers` paketa. Svaka od ovih klasa poseduje informaciju o tome za koju datoteku iz aplikativnog repozotorijuma je zadužena, kao i statičku metodu koja će izvršiti zadano učitavanje. Može se videti da svaka od ovih klasa odgovara jednoj Kroki generatorskoj klasi u smislu da je zadužena za učitavanje podataka iz datoteke koju ona generiše.

Kada se pokrene web aplikacija, zadatak `AdaptMainFrame` klase je da, u okviru svoje metode `loadMappings`, pozove odgovarajuću metodu svake od ovih klasa, kako bi se podaci učitali u odgovarajuće strukture podataka klase `AppCache`. Metoda `loadMappings` klase `AdaptMainFrame` je prikazana na Listingu 8.

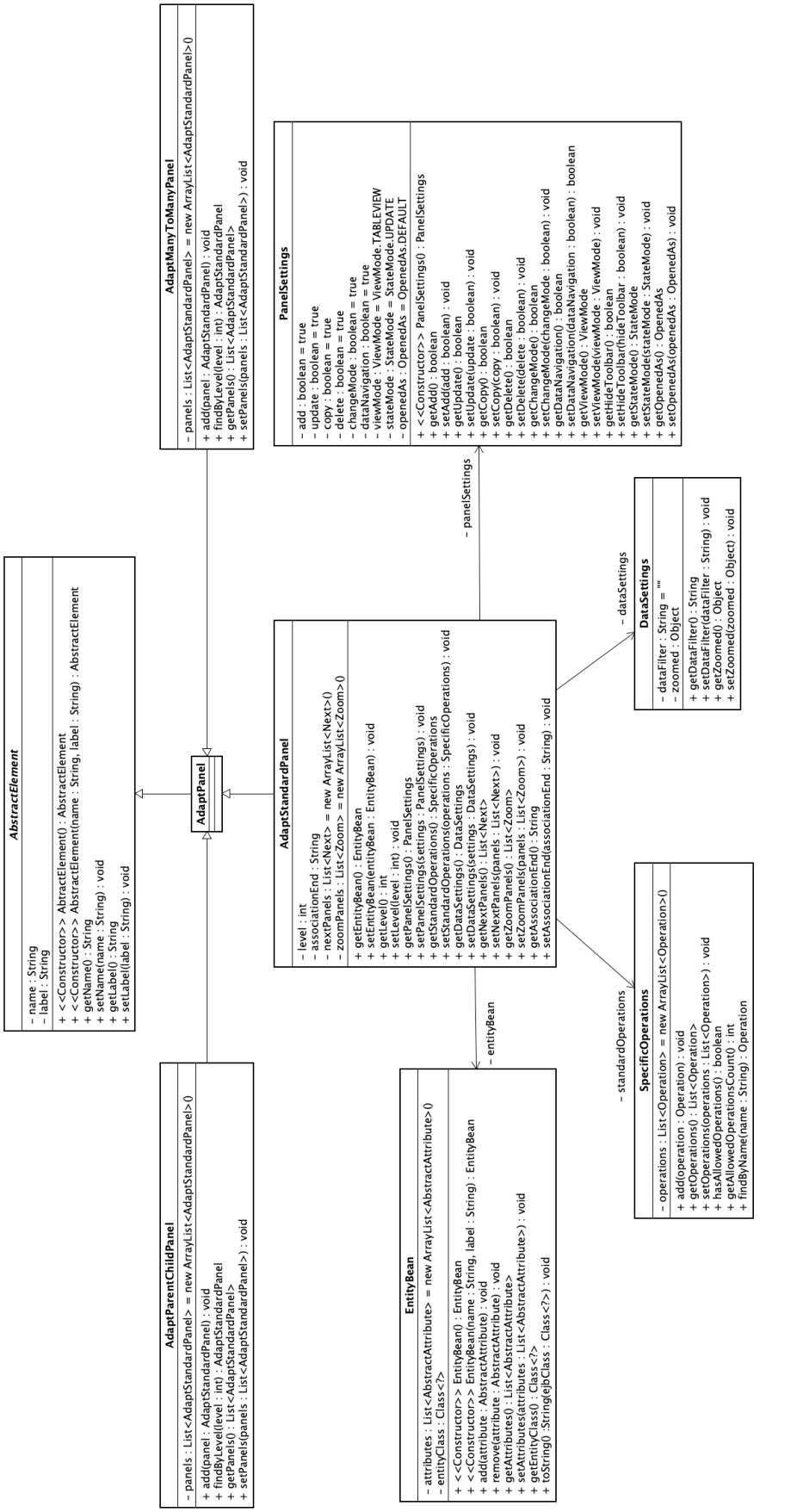


Slika 46 Primer prozora AdaptMainFrame

```
private void loadMappings() {
        EntityReader.loadMappings();
        PanelReader.loadMappings();
        TypeComponenMappingReader.mapTypesToComponents();
        MenuReader.load();
        EnumerationReader.loadEnumerations();
}
```

Listing 8 Metoda **loadMappings klase AdaptMainFrame**

U trenutku kada korisnik aktivira neku standardnu formu u toku rada web prototipa, podaci o toj formi i povezanim entitetima i panelima će se učitati iz odgovarajućih konfiguracionih datoteka u AppCache instancu i na osnovu njih će se kreirati instance odgovarajućih modelskih klasa generičke web aplikacije. Ove klase su smeštene `adapt.model` Java paketu i predstavljaju implementaciju meta-klasa definisanih EUIS DSL-om.



Slika 47 UML dijagram paketa `adapt.model`

UML dijagram modelskih klasa koje se koriste u okviru web aplikacije je prikazan na Slici 47. Osnovna klasa u ovoj implementaciji je `AdaptStandardPanel` koja predstavlja implementaciju EUIS standardnog panela prilagođenu konkretnoj upotrebi u okviru web generičke aplikacije. Specifikacija panela je upotpunjena povezanim klasama `SpecificOperations`, `DataSettings` i `PanelSettings` dok su složene forme modelovane klasama `AdaptParentChildPanel` i `AdaptManyToManyPanel` (sve ove klase nasleđuju klasu `AdaptPanel`). Na osnovu podataka o svakom od poslovnih entiteta u sistemu kreira se po jedna instanca klase `EntityBean` koja je povezana sa odgovarajućim panelom kako bi se omogućila direkta manipulacija podacima u bazi podataka sa pokrenute forme.

Prilikom aktiviranja željene standardne forme, poziva se `loadPanel` metoda klase `PanelReader` koja će učitati podatke o ovoj formi iz datoteke *`panel.xml`* ili iz `AppCache` instance, ukoliko se ovoj formi ne pristupa prvi put. Rezultat izvršavanja pomenute metode je instanca odgovarajućeg naslednika `AdaptPanel` klase. U toku obavljanja ovog zadatka pozivaju se procedure za učitavanje povezanih podataka, kao što je metoda `load` klase `EntityReader` koja će konstruisati instancu `EntityBean` klase i povezati je sa instancom trenutnog standardnog panela. Sakupljeni podaci se u JSON formatu šalju klijentskom delu aplikacije koji će u okviru web stranice (uz pomoć logike implementirane u JavaScript jeziku) formirati traženu formu.

3.7. Saradnja sa alatima za modelovanje i razvoj opšte namene

Kako bi se olakšalo uključivanje Kroki alata u postojeće procese razvoja koji podrazumevaju oslonac na različite alate za modelovanje i razvoj koji su dostupni na tržištu i prihvaćeni od strane razvojnih timova, u okviru Kroki platforme implementirani su mehanizmi koji omogućavaju razmenu Kroki projekata sa ovim alatima.

Saradnja sa alatima za modelovanje implementirana je kroz izvoz i uvoz UML dijagrama klasa zapisanih u XMI formatu. Ovo omogućava razvojnim timovima da svoje UML modele uvezu u Kroki, obave validaciju korišćenjem izvršivih prototipa, unesu potrebne izmene i izvezu popravljen model. Takođe, razvijene Kroki specifikacije se mogu izvesti u vidu UML dijagrama klasa, kako bi bili dostupni za pregled i izmenu od strane profesionalnih alata za UML modelovanje [151].

Mogućnost izvoza u XMI format daje razvojnim timovima veću slobodu, jer ih ne ograničava na pojednostavljeni UML editor i programsku platformu u okviru Kroki alata. Na ovaj način, razvojni tim može proverenu specifikaciju poslovnog sistema dobijenu Kroki alatom dalje koristiti putem odabranog UML alata opšte namene i na osnovu njega generisati željeni programski kod ili dokumentaciju.

Generisani Kroki prototip moguće je izvesti u obliku Java projekta za *Eclipse* razvojno okruženje, što omogućava razvojnom timu pristup programskom kodu radi implementacije ručnih izmena i dodataka, kako bi se unapredio generisani prototip ili dobila finalna implementacija. U nastavku su predstavljeni tehnički detalji implementacije podloge za saradnju sa alatima za modelovanje i programiranje i data je ilustracija njene upotrebe na realnom primeru.

3.7.1. Saradnja sa alatima za modelovanje

Prva implementacija podloge za saradnju sa alatima za modelovanje je implementirana u okviru [191] I publikovana u [151]. Ona je ostvarena na bazi EMF[34] (E*clipse Modeling Framework*) [172] radnog okvira i UML2 [152] implementacije UML-a. EMF je deo Eclipse platforme

[34] https://www.eclipse.org/modeling/emf/

specijalizovan za implementaciju jezika i alata za kreiranje i upravljanje softverskim modelima. U osnovi EMF radnog okvira je implementacija MOF meta-meta-modela pod nazivom *Ecore*, kao i skup biblioteka za prikaz i manipulaciju Ecore baziranih modela i generisanje programskog koda. Eclipse UML2 je implementacija OMG UML 2.x specifikacije bazirane na EMF okviru i namenjene za korišćenje u okviru *Eclipse* platforme.

Prilikom implementacije Kroki podloge za konverziju projekata u UML specifikaciju korišćen je podskup Eclipse UML2 meta-modela čiji su elementi odabrani tako da se mogu lako preslikati na odgovarajuće elemente EUIS meta-modela. Spisak ovih elemenata sa odgovarajućim EUIS elementima je dat u Tabeli 2. EMF platforma omogućava podršku za više formata spremanja modela koji se prepoznaju po različitim ekstenzijama datoteka u kojima se čuvaju. Kroki podloga za uvoz i izvoz modela je implementirana tako da podržava samo datoteke sa *.uml* ekstenzijom koje koriste specifikaciju Eclipse UML2 meta-modela počevši za verzijom 2.1.

Tabela 2 Mapiranje UML2 elemenata na elemente EUIS meta-modela

Eclipse UML2 elementi	Elementi EUIS metamodela
Model	BusinessSubsystem
Package	BusinessSubsystem
Class	StandardPanel
Property	VisibleProperty
Operation	VisibleOperation
Association	Zoom

Element Model predstavlja korenski element UML2 meta-modela i kao takav se preslikava na korenski element odgovarajućeg Kroki projekta. S obzirom da u EUIS jeziku ne postoji koncept projekta, za ovaj korenski element se koristi BusinessSubsystem, s obzirom da je ovaj element projektovan tako da može da sadrži druge poslovne podsisteme. Proces mapiranja Model elemenata na BusinessSubsystem je sledeći:

- Svojstvo label klase BusinessSubsystem se mapira na svojstvo name klase Model, uz svođenje na *CamelCase* notaciju radi dobijanja ispravnog Java imena,
- Svojstvo visible se uvek postavlja na vrednost true, a svojstvo owner na null za korenski podsistem,
- Vrednost svojstva componentType klase BusinessSubsystem se postavlja na ComponentType.MENU.

Svi ostali poslovni podsistemi koji nisu korenski se mapiraju na UML2 elemente Package. Prilikom uvoza UML modela i njegovog transformisanja u Kroki projekat, rekurzivno se na osnovu svih paketa UML modela kreira po jedna instanca Kroki BusinessSubsystem meta-klase. Svojstvo label se mapira na svojstvo name UML paketa, dok se vrednost svojstva owner postavlja kao referenca na roditeljski paket (odnosno poslovni sistem). Isti princip se koristi i prilikom izvoza Kroki projekata.

Na sličan način, rekurzivnim prolaskom, sve UML2 klase (elementi tipa Class) se mapiraju na EUIS standardne panele, pri čemu se za svaki od njih postavlja naziv (labela) i referenca na roditeljski element, nakon čega se prelazi na obradu njihovih atributa i operacija.

Kao što je prikazana u Tabeli 2, vidljiva obeležja EUIS profila (elementi tipa VisibleProperty) se mapiraju na elemente tipa Property UML2 meta-modela. Prilikom uvoza UML modela u Kroki alat i procesiranja elemenata tipa Property, potrebno je utvrditi da li se radi

o obeležju koje je kraj asocijacije. Ukoliko jeste, element se mapira na `Zoom` element. U suprotnom, za svaki `Property` element iz UML modela se kreira po jedan `VisibleProperty` u okviru Kroki standardne forme. Vrednost atributa `componentType` svakog kreiranog vidljivog obeležja se određuje na osnovu tipa podatka elementa `Property`. Mapiranje UML2 `Association` elemenata na `Zoom` elemente se obavlja tek nakon što su sva vidljiva obeležja procesirana, kako bi se reference na njih mogle ispravno postaviti. `Zoom` polja se kreiraju samo za asocijacije tipa *jedan-na-više*, što se proverava čitanjem kardinaliteta krajeva asocijacije.

3.7.2. Saradnja sa programskim razvojnim okruženjima opšte namene

Saradnja Kroki platforme sa alatima za razvoj softvera je implementirana u vidu podrške za jednosmernu komunikaciju sa Eclipse razvojnim okruženjem koja omogućava izvoz Kroki prototipova u obliku Eclipse Java projekata. Ovako izvezeni prototipovi se mogu koristiti nezavisno od Kroki platforme. Programski kod generičkih aplikacija na kojima se baziraju prototipovi je strukturiran tako da omogućava dodavanje ručnih izmena korišćenjem AspectJ platforme. Izdvajanjem ručno pisanih proširenja u okviru aspekata postignuta su dva cilja:

1. Uvođenje aspekata ne utiče na rad i kvalitet programskog rešenja generičkih poslovnih aplikacija,
2. Eventualne izmene na Kroki modelima i ponovni izvoz prototipa u vidu Eclipse projekata neće poništiti već napisane aspekte i proširenja implementirana u okviru njih. Ovo podrazumeva da su sve ručne izmene implementirane u okviru direktorijuma *src_manual*.

3.7.3. Testiranje podloge za saradnju Kroki alata na realnom primeru

Realna primena uvoza UML dijagrama u Kroki alat je testirana i prikazana u [174]. U radu je opisan uvoz CERIF (*Common European Research Information Format*) [175, 169] modela koji predstavlja standard za opis naučnih publikacija koji se koristi kao podloga u CRIS (*Current Research Information System*) informacionom sistemu na nivou Evropske unije. CERIF model podataka predstavlja detaljnu specifikaciju svih relevantnih aspekata katalogizacije publikacija i kao takav sadrži veliki broj entiteta i veza, što otežava čitanje i unapređivanje modela. Korišćenje Kroki alata sa pripadajućim editorima i mogućnošću generisanja izvršivog prototipa omogućilo je lakše uvođenje izmena u model i sagledavanje posledica tih izmena na CRIS informacionom sistemu koji ga koristi.

CERIF model je preuzet sa euroCRIS[35] web stranice i uvezen u Kroki alat. Više od 280 entiteta iz ovog modela je organizovano u 31 paket kako bi se poboljšala preglednost modela. Izvršivi prototip je izvezen u vidu Eclipse Java projekta i u njega je ručno dodat aspekt za podršku prikaza podataka o povezanim entitetima na više jezika, radi prilagođavanja izgleda prototipa potrebama CRIS sistema. Aspekt ima dve funkcije:

1. Pronalaženje odgovarajućeg entiteta na osnovu opisanih pravila imenovanja i injektovanje njegovih podataka u HTML tabelu za prikaz i formu za izmenu i dodavanje. Podaci o entitetu se dobavljaju pronalaženjem odgovarajuće EJB klase na osnovu imena upotrebnom Java refleksije,
2. Učitavanje konkretne vrednosti odgovarajućeg atributa pronađenog povezanog entiteta i njegovo injektovanje u skup vrednosti koji se prikazuje u odgovarajućim HTML

[35] https://eurocris.org/

elementima. Vrednost ovog atributa se učitava slanjem upita u bazu podataka, pošto aspekti, kao ravnopravni članovi Java projekta, imaju pristup svim delovima aplikacije kao i klase.

Uvezeni model, u vidu Kroki datoteke, zajedno sa prototipom se može preuzeti sa GitHub platforme[36].

Na osnovu opisanog se može videti da je preduslov za razvoj ručnih proširenja poznavanje detalja implementacije generičke Java aplikacije. Za definisanje aspekata je potrebno poznavati konkretne tačke u okviru programskog koda aplikacije u okviru kojih je potrebno intervenisati. S obzirom da dinamička priroda izvršavanja generičke Java aplikacije može biti teška za učenje od strane osoba koje nisu učestvovale u njenoj implementaciji, planira se razvoj programskog interfejsa (API) koji bi omogućio uključivanje ručno pisanih proširenja bez potrebe detaljnog poznavanja njene arhitekture.

3.8. Zaključak

U prethodnim sekcijama su prezentovani tehnički aspekti Kroki alata koji su implementirani sa ciljem da se postigne:

- kolaborativni razvoj specifikacije poslovne aplikacije sa korisnicima koji nemaju znanje projektovanja i programiranja softverskih sistema,
- efikasno pokretanje prototipa direktno iz sopstvenog razvojnog okruženja, dajući mogućnost korisniku da isproba prototip tokom modelovanja kad god poželi,
- ponovno korišćenje informacija dobijenih prilikom razvoja prototipova u kasnijim fazama razvoja, kako bi se smanjilo nepotrebno trošenje resursa.

Provera uspešnosti ovih ciljeva je obavljena eksperimentom opisanim u okviru petog poglavlja. U okviru narednog poglavlja je dato uputstvo za korišćenje grafičkih editora, radi sticanja uvida u funkcionalnost alata i procene pogodnosti za realan razvoj.

Radi poređenja sa postojećim rešenjima analiziranim u sekciji 2.2.4 kreirana je sumarna tabela 3. Može se uočiti sledeće:

- Kreiranje skica korisničkog interfejsa kod većine alata ne predstavlja inicijalnu fazu u razvoju, već se ono sprovodi tek nakon već obavljene faze prikupljanja zahteva, uvodeći redundantne korake.
- Alati koji pokušavaju da automatizuju korišćenje skica za kreiranje izvršivih prototipova importuju skice crtane alatima opšte namene, a zatim ih dodatno anotiraju, što usložnjava i usporava ceo proces i povećava mogućnost unosa greške.
- Alati jednostavni za korišćenje, u velikom broju slučajeva, proizvode samo programski kod koji realizuje izgled aplikacije, pri čemu se programska logika mora ručno implementirati, tako da korisnik ne može da stekne osećaj realnog korišćenja.
- U slučaju alata koji obezbeđuju značajan procenat generisanog koda (npr. WebRatio), notacija za modelovanje je obično kompleksna i teška za učenje i ne pogoduje zajedničkom radu sa korisnikom.

[36] https://github.com/KROKIteam/KROKI-CERIF-Model

- Do izvršive aplikacije ili njenog prototipa se dolazi kroz minimalno jednu M2M transformaciju
- Nijedan od razmatranih alata ne koristi skice kao deo konkretne sintakse jezika.

Tabela 3 Poređenje karakteristika Kroki alata sa razmatranim alatima iz sekcije 2.2.4

Alat	Dizajn započinje kreiranjem	Način skiciranja	Potrebno dodatno ručno procesiranje skica?	Potrebno parsiranje skica?	Potrebna minimalno jedna M2M transformacija?	Koji delovi finalne aplikacije se generišu?	Podržava trenutno izvršavanje prototipa?
MockupDD [20]	Skica	Alati za skiciranje opšte namene	Da	Da	Da	Izvršivi prototip/MDWE modeli (npr. WebML)	Ne
DataMock [58]	Skica	Alati za skiciranje opšte namene	Da	Da	Da	UML dijagram klasa	Ne
AIDE [38]	Skica	Papir i olovka	Da	Ne	Da	XUL (XML User Interface Language) XML datoteke	Ne
Marama AI [40]	EUI (Essential UI) modela upotrebom MaramaEUI Editor alata	-	-	-	Da	EUC (Essential Use Case) model i jednostavan HTML prototip aplikcaije	Ne
MontiGEM [185]	Modela struktura podataka	-	-	-	Da	Proširivi izvršivi prototip koji podržava CRUD operacije	Ne
SocietySoft [186]	Skica	PowerPoint	Ne	Da	Da	Navigabilni prototip korisničkog interfejsa za Enterprise Architect alat	Ne
Umple [35]	Tekstualnog modela podataka	-	-	-	Da	Prototip korisničkog interfejsa koji podržava CRUD operacije	Da
WebRatio [61]	WebML modela	-	-	-	Da	Izvršivi prototip	Ne
Protoizer [39]	MagicDraw dijagrama klasa	-	-	-	Ne	SQL procedure, *model* (klase) i *view* (UI programski kod) slojevi aplikacije	Ne
Alati baziriani na ASL jeziku [187]	Tekstualnog modela i Modela slučajeva korišćenja	-	-	-	Da	Django aplikacija koja podržava CRUD operacije	Ne
XIS Web [188]	Domenskog modela, Modela poslovnih entiteta, Modela arhitekture i Modela slučajeva korišćenja	-	-	-	Da	Web protoip koji podržava CRUD operacije	Da
E WAE [194]	UML dijagrama sa implementiranim	-	-	-	Da	JSF ili .NET prototip	Ne

	WAE UML profilom						
Jmix	Modela aplikacije upotrebom alata CUBA Studio	-	-	-	Ne	Izvršiva JavaScript Web aplikacija	Da
APPGen [23]	Modela podataka	-	-	-	Da	Desktop aplikacija, SQL procedure	Ne
Kroki	Skica	Kroki Mockup Editor alat	Ne	Ne	Ne	Desktop i Web aplikacija, XMI model podataka	Da

POGLAVLJE 4

KORIŠĆENJE RAZVIJENOG SOFVERSKOG REŠENJA

U ovom poglavlju je pokazano korišćenje Kroki alata: rad sa projektom, kreiranje specifikacije poslovne aplikacije korišćenjem dve grafičke sintakse, pokretanje razvijene aplikacije i korišćenje pokrenutog prototipa.

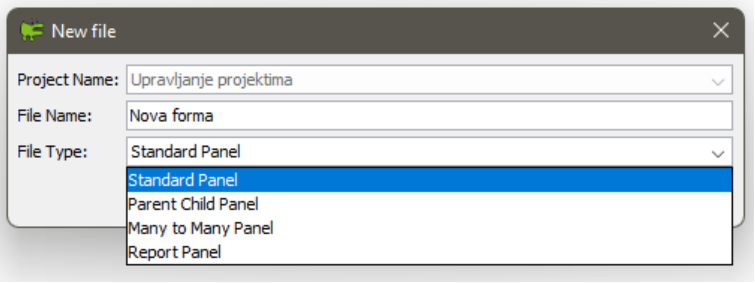
4.1. Rad sa projektom (`File` podmeni)

Pre dodavanja novih skica, potrebno je da postoji minimalno jedan aktivan projekat u okviru razvojnog okruženja. Kreiranje novog projekta se vrši putem opcije glavnog menija *File -> New Project.* Prethodno sačuvani Kroki projekat se otvara putem opcije menija *File -> Open Project*. Kroki projekti se čuvaju u binarnim datotekama sa ekstenzijom .kroki.

Uvoz UML dijagrama klasa kreiranim nekim od alata za modelovanje opšte namene koji podržavaju Eclipse UML2 format (datoteka sa ekstenzijom .uml) se obavlja opcijom *File -> Import.*

4.2 Kreiranje standardne forme

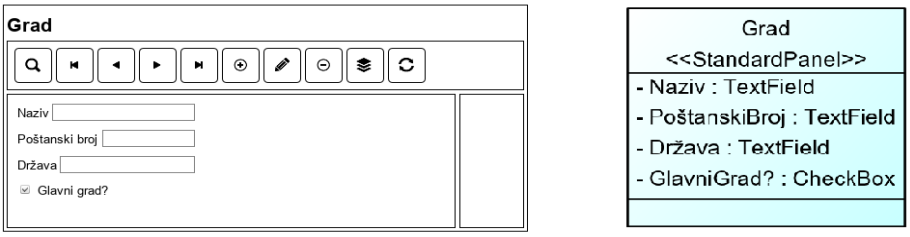
Kreiranje nove standardne forme u editoru skica se vrši pritiskom na desni taster projekta ili paketa u okviru panela za prikaz strukture projekta i izborom opcije *New -> File* kojom se aktivira dijalog na slici 48.



Slika 48 Dijalog za kreiranje forme u okviru editora skica

Standardna forma, u skladu sa internim UI standardom, sadrži naslovni deo, deo za standardne operacije, centralni deo za manipulaciju podacima i desni deo za specifične operacije (transakcije i izveštaje). Dodavanjem UI elemenata na skicu standardne forme definišemo koje podatke želimo da evidentiramo o perzistentnom entitetu za koji kreiramo standardnu formu.

Prilikom korišćenja pojednostavljenog UML editora, kreiranje novih standardnih formi se vrši dodavanjem UML klase na dijagram odabirom *Class* alata iz palete i klikom na željeno mesto na dijagramu. Sve novokreirane klase u okviru pojednostavljenog UML editora imaju pridružen stereotip `StandardPanel`, koji se može promeniti putem panela za podešavanje osobina odabrane klase. UML klasa koja predstavlja formu sa Slike 49a je prikazana pored nje u b) delu iste slike.



Slika 49 a) Kroki skica standardne forme b) Prikaz iste forme u pojednostavljenom UML editoru

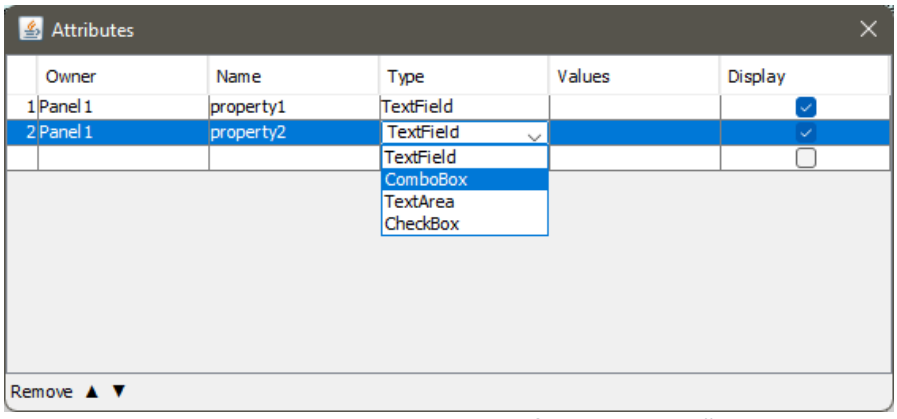
4.2.1 Vidljivi elementi

Prilikom rada sa skicama, potrebno je iz palete sa desne strane editora skica odabrati željeni tip UI elementa i kliknuti na formu na koju želimo da ga dodamo. U Tabeli 4 su navedene UI komponente dostupne u okviru editora skica sa navedenim tipovima podataka koje podržavaju.

Tabela 4 UI komponente dostupne u okviru editora skica i njihovi tipovi podataka

Slika	Opis	Tipovi podataka
Textfield	Jednolinijsko tekstualno polje	Bilo koji
Textarea	Višelinijsko tekstualno polje	Bilo koji, preferirano tekst
Combobox	Padajuća lista	Bilo koji, preferirano tekst
Checkbox	Checkbox	Logički tip (boolean)

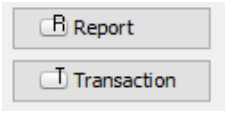
Dodavanje elemenata na standardnu formu putem pojednostavljenog UML editora se obavlja korišćenjem dijaloga koji se poziva klikom na *Attributes* dugme u panelu za podešavanje osobina trenutno odabrane klase. Dijalog za dodavanje atributa prikazuje tabelu u koju se novi atributi mogu dodavati klikom levim tasterom miša na prazan red. Projektant može efikasno dodati onoliko redova koliko mu treba atributa na formi, pri čemu će oni biti popunjeni predefinisanim podacima i nakon toga krenuti u njihovo detaljno konfigurisanje. Dijalog za dodavanje i podešavanje atributa forme je prikazan na Slici 50.



Slika 50 Dijalog za upravljanje atributima forme u UML režimu rada

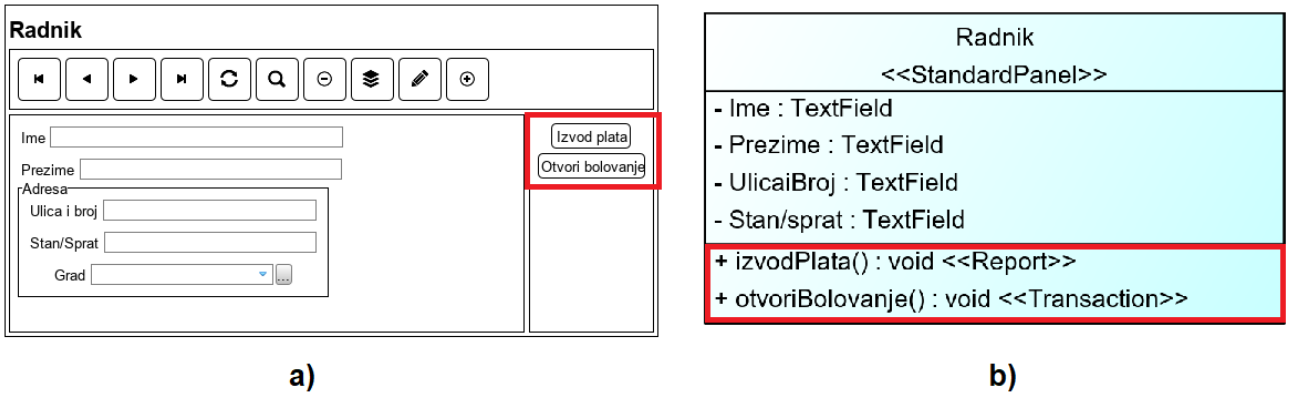
4.2.2 Poslovne operacije

Poslovne operacije definisane EUIS jezikom, *Transakcija* i *Izveštaj*, takođe imaju svoju implementaciju u okviru editora skica i dostupne su na paleti (slika 51). Operacije se dodaju u desni deo forme, u deo rezervisan za njih. Onemogućeno je dodavanje operacije na neko drugo mesto, što se korisnicima signalizira promenom izgleda kursora miša.



Slika 51 Poslovne operacije u paleti editora skica

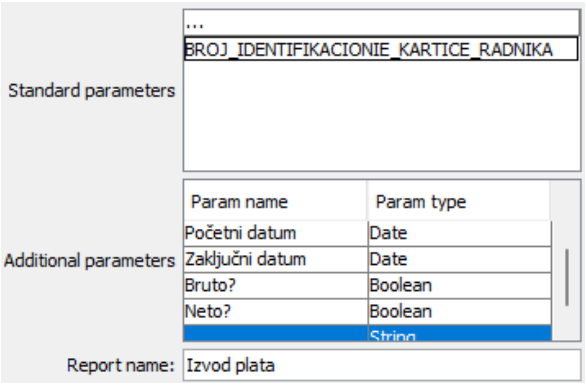
Dodavanjem i odabirom operacije na skici forme, u sklopu panela za podešavanje naprednih osobina će biti omogućeno podešavanje svih svojstava definisanih EUIS DSL-om. Skica forme, kao i njena reprezentacija u UML sintaksi, sa pridruženim izveštajem pod nazivom "Izvod plata" i transakcijom nazvanom "Otvori bolovanje" je prikazana na slici 52.



Slika 52 Izgled poslovnih operacija na a) skici standardne forme i b) njenom UML prikazu

Za izveštaj koji je dodat na standardnu formu moguće je specificirati parametre putem jezička za definisanje naprednih osobina. Ovaj deo je prikazan na Slici 53 gde se može videti da je za svaki izveštaj moguće definisati dve grupe parametara:

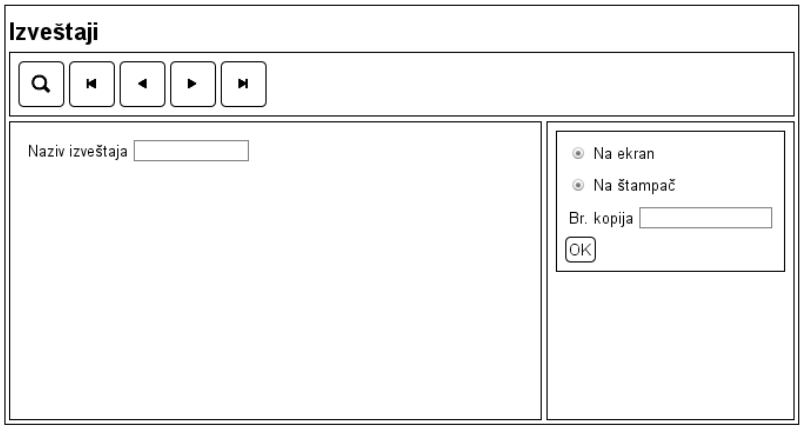
1. Standardni parametri (Standard parameters) – predstavljaju elemente standardne forme koji učestvuju u kreiranju izveštaja, odnosno koriste se za filtriranje podataka na njemu. U ovoj listi je moguće navesti imena kolona bilo kojeg perzistentnog elementa sa forme na kojoj se nalazi izveštaj. Ukoliko se radi o polju koje predstavlja strani ključ, u okviru generisanog prototipa će biti prikazana komponenta za prikaz povezane forme i preuzimanje željenog reda kako bi se popunila vrednost stranog ključa.
2. Dodatni parametri (Additional parameters) - Predstavljaju dodatne parametre koji se mogu koristiti na izveštaju. Ovi parametri se mogu koristiti samo za prikaz dodatnih informacija (npr. datum kreiranja) ili mogu učestvovati u filtriranju podataka zajedno sa standardnim parametrima.



Slika 53 Podešavanje naprednih osobina izveštaja

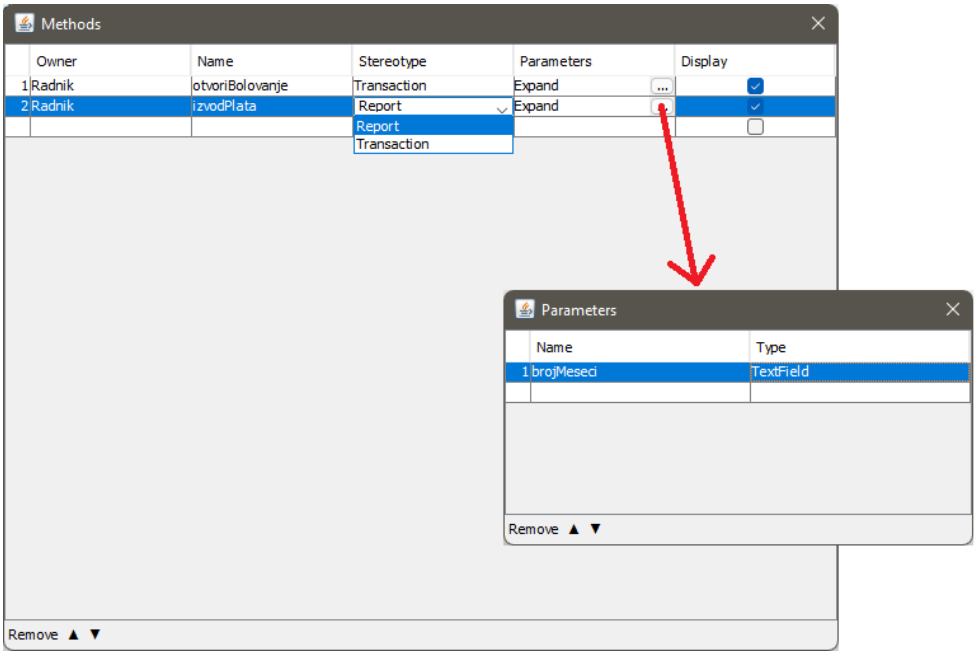
Kako bi se omogućila bolja preglednost izveštaja u sistemu, Kroki omogućava kreiranje posebne vrste forme, nazvane *Report Panel*, sa koje je moguće prikazati bilo koji izveštaj iz trenutnog Kroki projekta uz mogućnost definisanja nekoliko brzih parametara kao što su izlaz na kojem želimo da prikažemo izveštaj (ekran ili štampač) i broj kopija koji nam je potreban.

Skica ove forme je prikazana na Slici 54. Na slici se može videti da ona liči na standardnu Kroki formu, ali sadrži samo jedno predefinisano polje „Naziv izveštaja“ i deo sa definisanjem opcija prikaza izveštaja. U generisanom prototipu, u tabelarnom režimu prikaza, ova forma će sadržavati sve izveštaje iz sistema. Odabirom izveštaja u tabeli i klikom na dugme Ok, pokreće se odabrani izveštaj što uključuje prikaz forme za preuzimanje parametara od korisnika (ukoliko izveštaj to zahteva) i prikaz samog izveštaja.



Slika 54 Skica forme za prikaz izveštaja

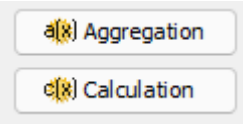
U UML režimu rada, operacije se nazivaju metodama i njima se, slično kao i atributima, upravlja odabirom željene klase na dijagramu. Klikom na dugme *Methods* u delu prozora za podešavanje osobina odabrane forme, otvara se dijalog za dodavanje novih i upravljanje postojećim metodama klase, pri čemu tip metode određuje tip poslovne operacije koji će biti instanciran. Rad sa ovim dijalogom je istovetan radu sa dijalogom za upravljanje atributima klase, osim što je ovde moguće podesiti i parametre odabrane operacije. Klikom na dugme u koloni *Parameters*, otvara se dijalog za definisanje parametara metode. Dati dijalozi su prikazani na Slici 55.



Slika 55 UML dijalog za podešavanje osobina metode i dijalogom za upravljanje njenim parametrima

4.2.3 Izvedena polja

Vrednosti izvedenih polja, kao što je definisano EUIS jezikom, korisnik ne unosi direktno, već se izvode iz vrednosti drugih polja na istoj ili na povezanim formama. Kroki alat omogućava modelovanje agregiranih i kalkulisanih polja.



Slika 56 Izvedena polja u Kroki paleti

Radeći u režimu za manipulaciju UI skicama, alatke za specifikaciju agregiranih i kalkulisanih polja se nalaze u posebnoj sekciji palete, nazvanoj *Actions*, i označene su kao *Aggregation* i *Calculation*, respektivno (slika 56). Smeštanje izvedenih polja putem ovih alatki direktno na formu nije moguće, već se one koriste za konvertovanje „običnih“ tekstualnih polja u izvedena. Tekstualno polje na koje je primenjena alatka Aggregation ili Calculation postaje izvedeno polje, pri čemu zadržava svoja vizuelna svojstva, ali dobija mogućnost unosa formule (obeležje `Expression`) putem OCL izraza. Na skici izvedena polja imaju sivu pozadinu, u odnosu na polja za slobodni unos koja podrazumevano imaju belu, kako bi se naglasilo da nisu omogućena za unos vrednosti.

Trenutna verzija pojednostavljenog UML editora ne omogućava pretvaranje tekstualnih polja u izvedena, tako da je ovo moguće samo iz režima kreiranja skica, dok izvedena polja u UML režimu prikazuju kao polja tipa TextField.

4.2.4 Povezivanje formi

Veze između entiteta poslovnog sistema, odnosno njima pridruženih standardnih formi se na skicama modeluju na sledeće načine:

1. Korišćenjem *Combozoom* komponente

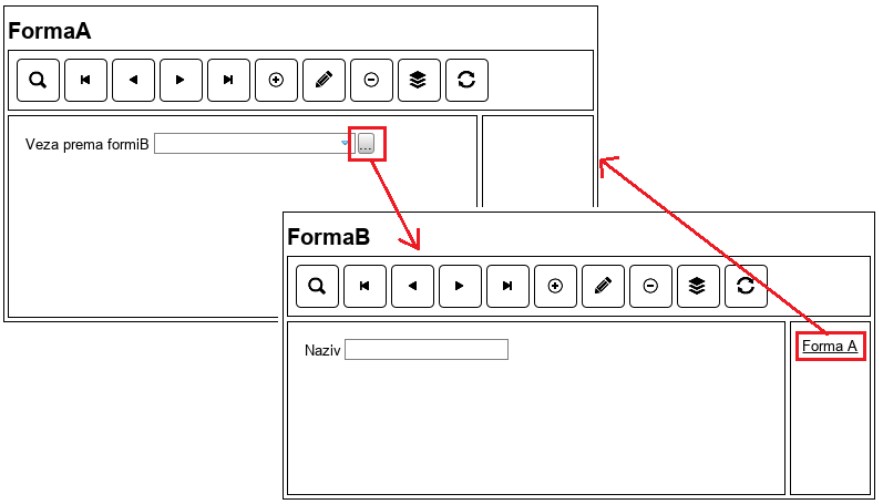
2. Korišćenjem *Link* komponente

Combozoom alatka se ne može direktno dodati na formu, već se pridružuje postojećoj *Combobox* komponenti, nakon čega ona postaje *Combozoom*. Dodavanjem *combozoom* komponente na formu i podešavanjem ciljne forme, kreiraće se veza asocijacije između entiteta koji se generišu prilikom generisanja prototipa.

Link UI komponenta se na formu dodaje iz Kroki palete, gde je predstavljena istoimenim alatom, i smešta se u desni deo forme rezervisan za specifične akcije. *Link* komponenta omogućava aktiviranje neke druge standardne forme koja se postavlja kao ciljni panel u okviru dela za podešavanje naprednih osobina odabranog linka.

Između ciljne i aktivacione forme (forme na kojoj se nalazi *Link* komponenta), odnosno njihovih pridruženih entiteta, ne mora da postoji veza asocijacije i u tom slučaju Link predstavlja implementaciju *Activate* stereotipa iz EUIS DSL-a. Ukoliko ciljna forma sadrži vezu prema aktivacionoj u vidu *Combozoom* komponente, *Link* omogućava modelovanje *Next* mehanizma pri čemu se podaci ciljne forme automatski filtriraju u odnosu na odabrani red u aktivacionoj formi.

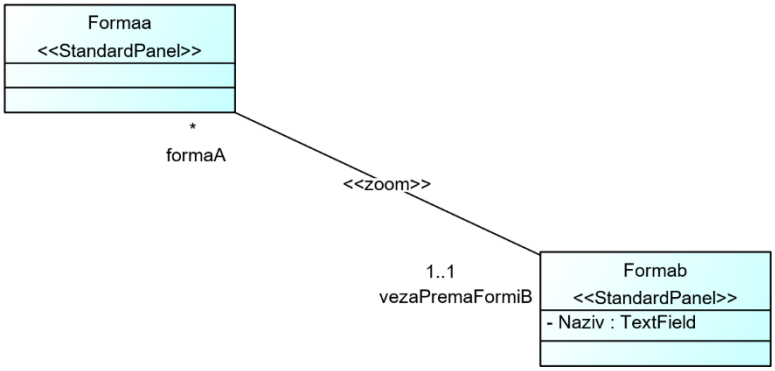
Slika 57 prikazuje skice dve forme, A i B, između kojih postoji veza asocijacije. Na skici forme A se vidi *Combozoom* komponenta koja aktivira formu B sa ciljem unosa podataka koji nedostaju u combobox komponenti na formi A. Forma B sadrži *Link* komponentu putem koje se može aktivirati forma A sa filtriranim podacima u odnosu na trenutno odabrani red forme B.



Slika 57 Skice formi povezanih putem *combozoom* i *link* komponenti

Povezivanje klasa u okviru UML režima obavlja se odabirom alatke koja predstavlja asocijaciju i prevlačenjem miša, držeći pritisnut levi taster, od odredišne prema ciljnoj klasi. Kada se dve klase povežu asocijacijom, na skici aktivacione forme će se kreirati *Combozoom* komponenta, dok će se na skici ciljne forme kreirati *Link* komponenta sa adekvatno podešenim osobinama.

Na Slici 58 su prikazane forme sa Slike 57 u pojednostavljenom UML editoru sa vezom asocijacije između njih.



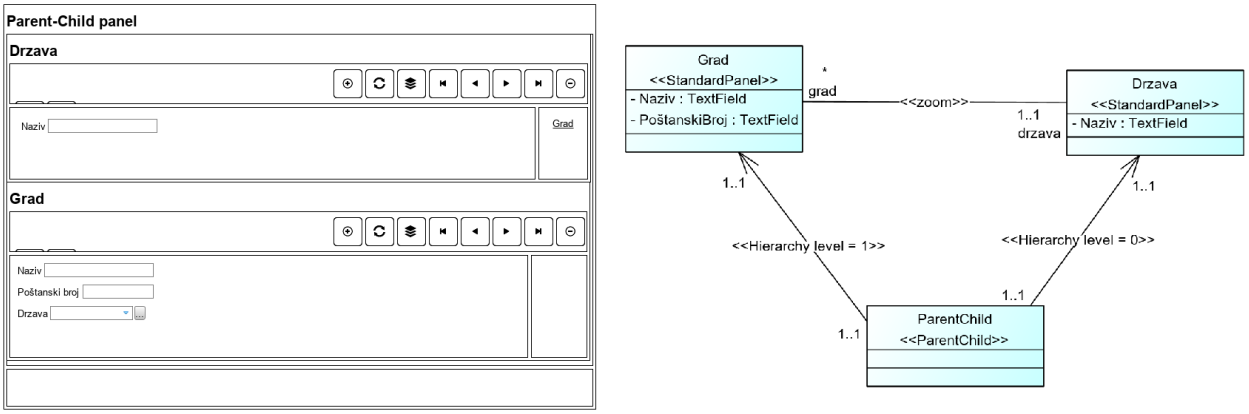
Slika 58 Povezane forme u pojednostavljenom UML editoru koje odgovaraju skicama sa slike 57

4.3 Parent-Child forme

Kreiranje „Parent-Child“ forme u okviru Kroki režima za manipulaciju UI skicama se vrši odabirom *Parent Child Panel* opcije iz dijaloga sa Slike 48. Nakon kreiranja prazne forme, moguće je dodavanje jedino komponenti tipa *Hierarchy*. Svaka *Hierarchy* komponenta služi za modelovanje jednog nivoa hijerarhije koju želimo da prikažemo i na njemu se prikazuje standardni panel koji je podešen kao ciljni panel za tu komponentu. Podešavanje ciljnog panela za odabranu *Hierarchy* kompoentu se vrši iz dela za podešavanje njenih naprednih osobina. Prilikom određivanja ciljnog panela za prvi nivo hijerarhije, Kroki alat će omogućiti korisniku da odabere bilo koju standardnu formu iz modela. Na svim ostalim nivoima, opcije za odabir će biti filtrirane tako da se mogu odabrati samo one forme koje imaju vezu asocijacije prema formi na nivou iznad trenutnog, tako da se pravilno kreiranje i funkcionisanje prototipa obezbeđuje od strane editora skica.

Broj i tip veza je jedini faktor koji ograničava broj mogućih nivoa hijerarhije na jednoj formi, ali treba obratiti pažnju da se povećavanjem broja prikazanih formi smanjuje preglednost podataka na ekranu. Slika 59a prikazuje primer skice „Parent-Child“ forme sa dva nivoa hijerarhije.

Kreiranje ovog tipa forme u pojednostavljenom UML editoru započinje dodavanjem UML klase sa stereotipom `ParentChild`, što je ekvivalentno kreiranju prazne forme u režimu kreiranja skica. Dodavanje pod-formi se vrši povlačenjem veza asocijacije između ove klase i UML klasa koje predstavljaju standardne panele koje želimo da uključimo. Slično kao i kada kreiramo UI skicu, prilikom definisanja prvog nivoa hijerarhije, UML alat će omogućiti kreiranje ove veze između „Parent-Child“ klase i bilo koje klase na dijagramu, pri čemu će nova veza automatski biti nazvana *Hierarchy Level=0*. Nakon toga, kreiranje veza u cilju dodavanja nižih nivoa hijerarhije će biti omogućeno samo prema klasama koje imaju vezu asocijacije sa klasom na nivou koji je viši od nivoa trenutne veze. Na slici 59 je prikazana jedna „Parent-Child“ forma prikazana korišćenjem obe grafičke notacije.



Slika 59 „Parent-Child“ forma u okviru editora skica i pojednostavljenog UML editora

4.4 Many-to-Many forme

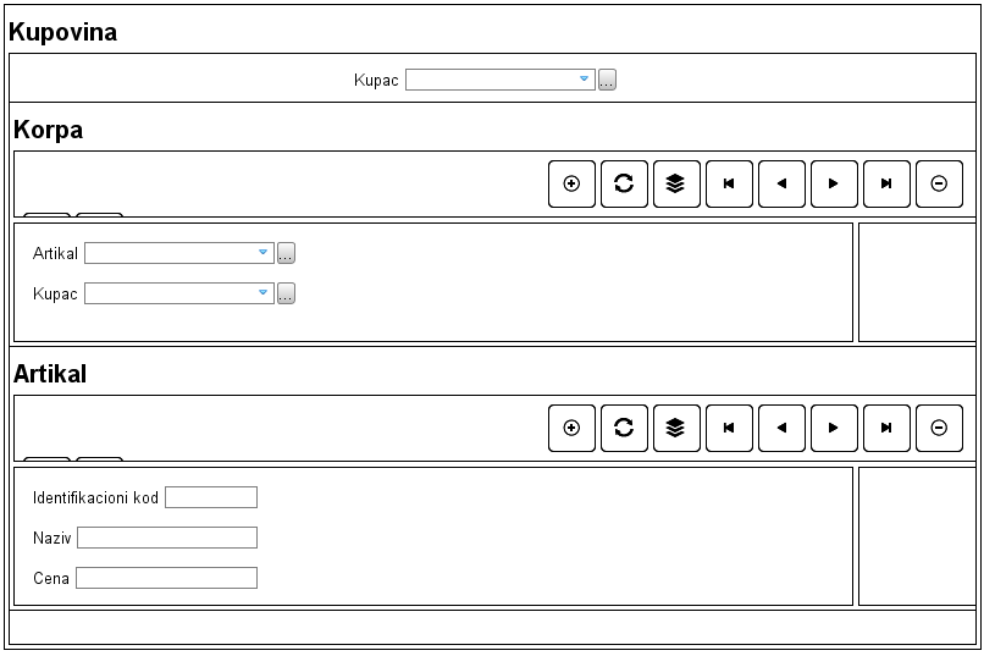
„Many-to-Many“ forme su složeni tip formi koji je internim UI standardnom predviđen za lakši unos podataka entiteta koji su nastali kao rezultat veza *„više-na-više“*, sa ili bez asocijativnih klasa. U skladu s tim, Kroki omogućava kreiranje „Many-to-Many“ forme, ali pre nego što se krene sa kreiranjem same forme u projektu je potrebno kreirati ostale forme koje učestvuju u vezi. To su:

1. Standardne forme koje predstavljaju entitete na krajevima *many-to-many* veze,
2. Standardna forma koja sadrži *combozoom* komponente koje kao ciljne panele imaju navedene forme.

Kroki skice „Many-to-Many“ formi, sadrže sledeće elemente:

1. Combozoom komponentu koja pokazuje na standardnu formu koja predstavlja entitet koji se nalazi na jednom kraju asocijacije. Ova komponenta će u generisanom prototipu služiti za odabir konkretnog reda entiteta koji učestvuje u vezi sa jedne strane.
2. Hierarchy komponentu koja kao ciljni panel ima formu pridruženu entitetu nastalom kao posledica veze „više-na-više“.
3. Hijerarchy komponentu koja kao ciljni panel ima standardnu formu čiji pridruženi entitet se nalazi na drugom kraju veze.

Dodavanje više od dva nivoa hijerarhije na ovaj tip forme je onemogućeno od strane Kroki alata (slika 60).



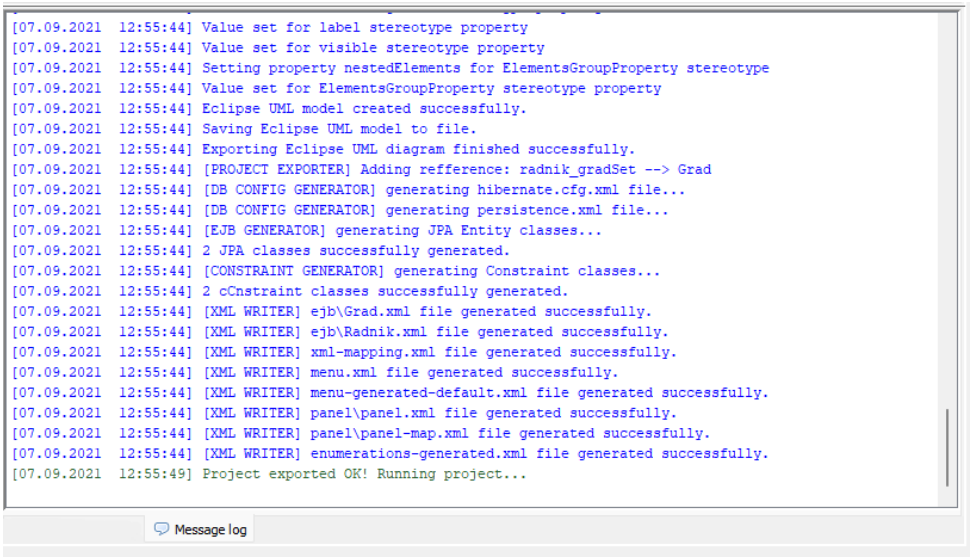
Slika 60 Skica „Many-to-Many“ forme u editoru skica

Kreiranje ovog tipa složene forme u okviru pojednostavljenog UML editora se vrši slično kao i „Parent-Child“ forme. Prvo se kreira klasa sa stereotipom ManyToMany, a zatim se dodaju sledeće veze:

1. Veza asocijacije (zoom) između „Many-to-Many“ klase i klase koja predstavlja entitet na jednom kraju ove veze,
2. Veza asocijacije između „Many-to-Many“ klase i klase koja predstavlja entitet na drugom kraju asocijacije, ova veza predstavlja prvi nivo hijerarhije,
3. Veza asocijacije između „Many-to-Many“ klase i klase koja predstavlja povezni entitet, ova veza predstavlja drugi nivo hijerarhije.

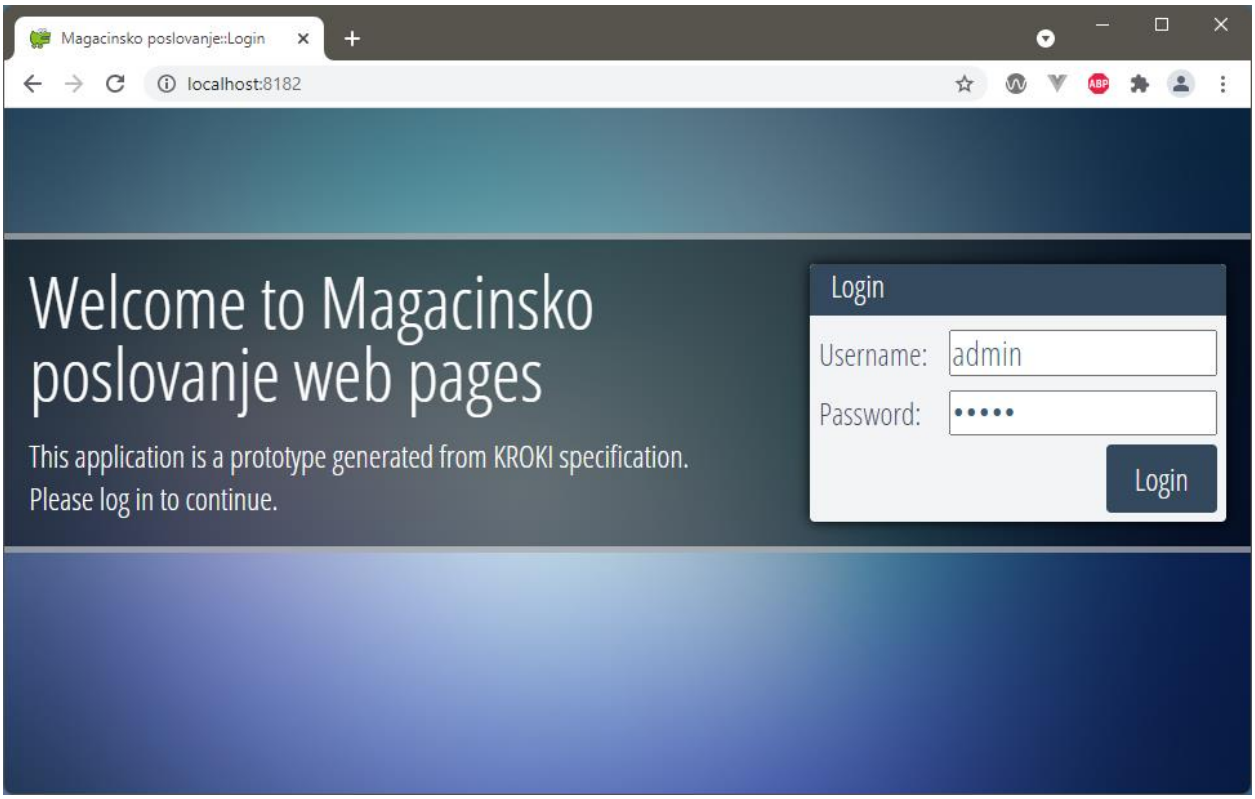
4.5 Korišćenje web prototipa

U nastavku će biti prikazan primer korišćenja izvršivog prototipa dobijenog pokretanjem generičke web aplikacije nad specifikacijom sa slike 59. Nakon kreiranja specifikacije korišćenjem bilo kojeg Kroki editora, proces pokretanja prototipa se inicira klikom na dugme za pokretanje u alatnoj traci glavnog Kroki prozora. Informacije o svakom koraku ovog procesa se prikazuju u vidu tekstualnih poruka u okviru panela nazvanog *Message log* koji se nalazi u donjem delu editora skica. Primer ovog panela sa porukama o uspešnom generisanju koda i pokretanju prototipa je prikazan na Slici 61.



Slika 61 Panel za prikaz poruka tokom pokretanja prototipa

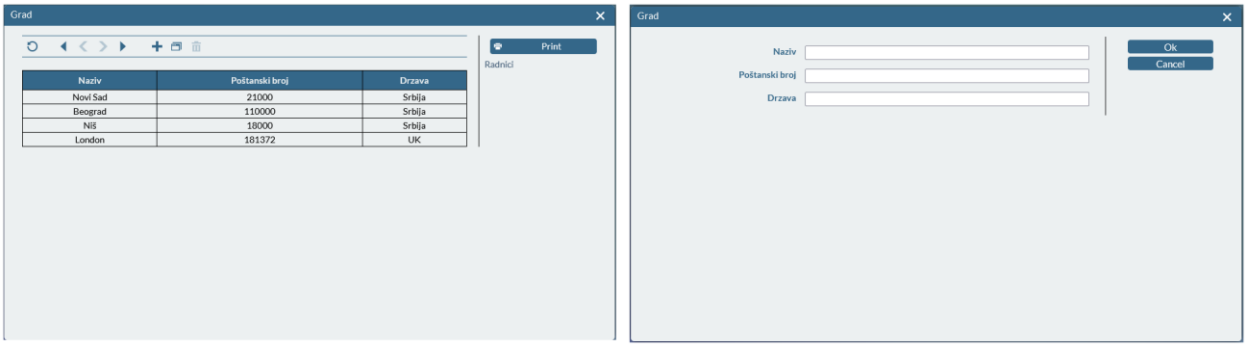
Nakon uspešnog generisanja podataka u aplikativnom repozitorijumu, automatski se pokreće generička web aplikacija koja će, nakon inicijalnog učitavanja podataka, otvoriti web čitač sa naslovnom stranom aplikacije koja sadrži formu za prijavu. Ova forma će biti popunjena podacima za prijavu predefinisanog administrativnog korisnika, ukoliko nije specificirano drugačije. Web čitač sa otvorenom stranicom za prijavu je prikazan na Slici 62.



Slika 62 Početna stranica generičke web aplikacije popunjena imenom projekta i podacima predefinisanog korisnika

Nakon prijave, otvara se glavna stranica aplikacije koja sadrži samo glavni meni putem kojeg se mogu pokrenuti sve specificirane forme. Ukoliko nije drugačije modelovano, struktura ovog menija oslikava strukturu Kroki projekta. Sa leve strane slike 63 je prikazana forma za

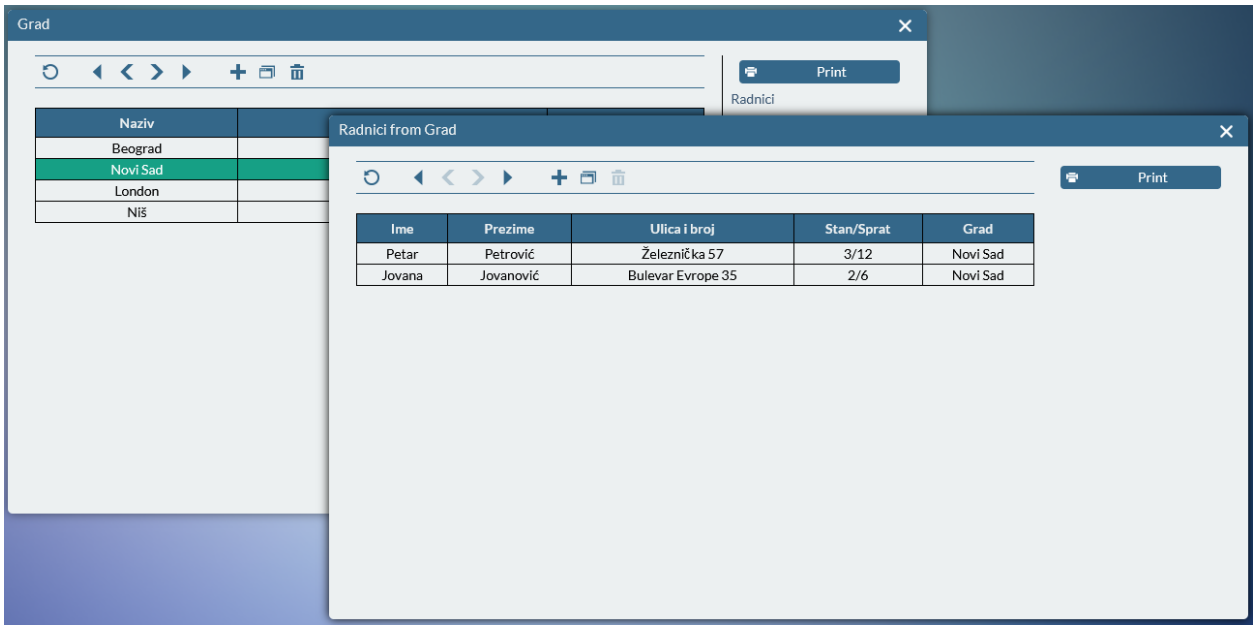
evidenciju podataka o gradovima u tabelarnom režimu prikaza, dok je pored nje prikazana ista forma u režimu za unos podataka.



Slika 63 Web prototip forme za evidenciju podataka o gradovima u tabelarnom prikazu i u režimu unosa

Kao što se može videti, forme u okviru prototipa kao i forme na skicama u okviru Kroki alata, odgovaraju EUIS specifikaciji i sastoje se od glavne alatne trake, dela za manipulaciju podacima i dela sa specifičnim operacijama. Kroki skica ove forme je prikazana na Slici 49. U delu za specifične operacije se nalazi veza prema formi *Radnici* koja predstavlja web implementaciju Kroki *Link* komponente. Klikom na ovu vezu aktivira se forma *Radnici* koja prikazuje podatke o radnicima filtrirane u odnosu na trenutno odabrani red u tabeli na formi *Gradovi*. Opisani način aktivacije predstavlja realizaciju *Next* mehanizma definisanog EUIS standardom.

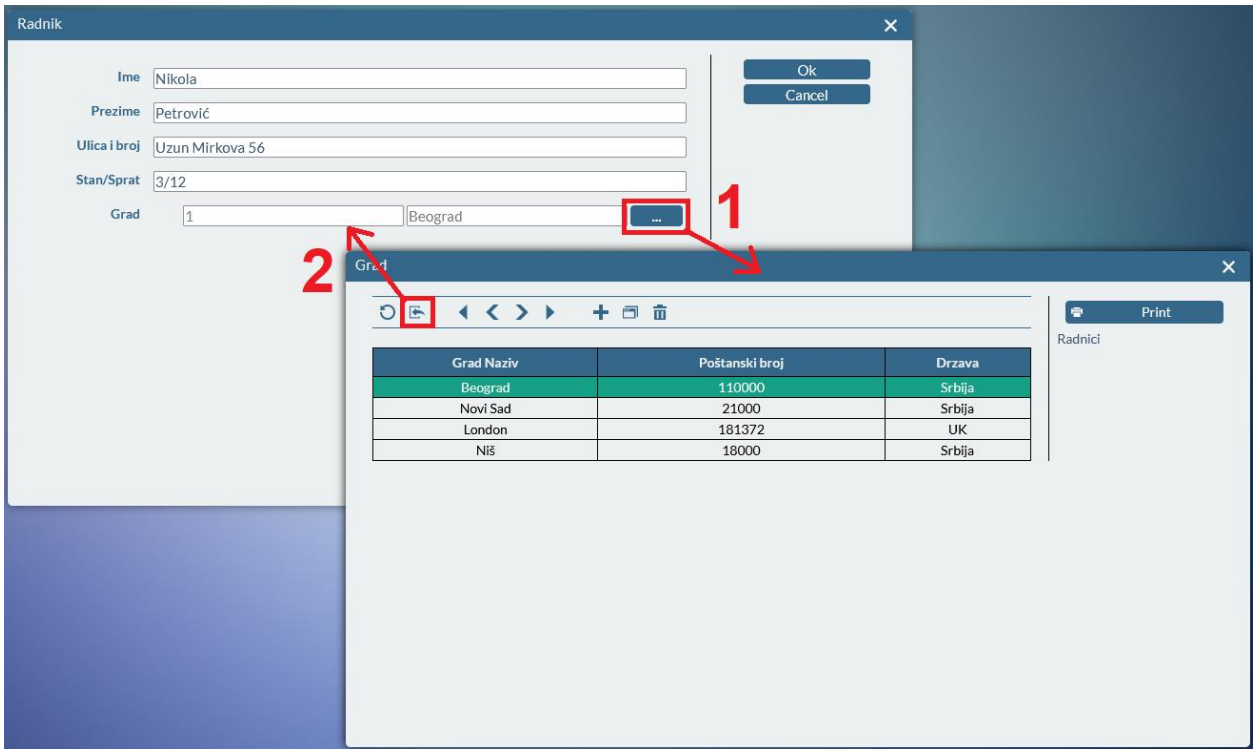
Na Slici 64 je prikazana forma *Radnici* aktivirana linkom sa forme *Grad* koja se, zajedno sa odabranim redom, može videti u pozadini slike. U koloni *Grad* forme *Radnici* se prikazuje ime grada, jer je to obeležje označeno kao reprezentativno na skici. Ukoliko se ne specificiraju reprezentativna obeležja za neki entitet, u povezanim kolonama će se prikazivati automatske identifikacione oznake konkretnih redova.



Slika 64 Prikaz implementacije *Next* mehanizma u okviru web prototipa

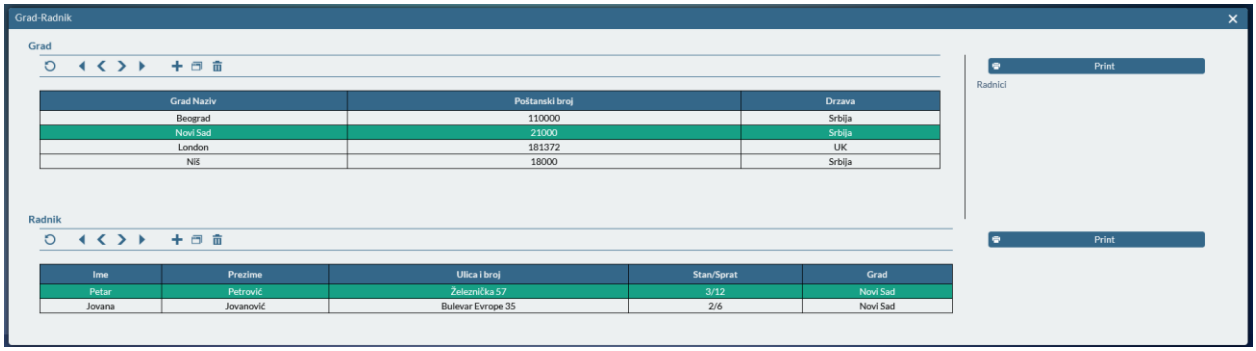
Forma *Radnici* je modelovana tako da se podaci o gradu koji je povezan sa odabranim radnikom unose i prikazuju pomoću *Combozoom* komponente (skica sa slike 52). Izgled ove web forme je prikazan na Slici 65. Slično kao i prilikom implementacije povezane kolone, *Combozoom*

komponenta prikazuje identifikacionu oznaku kao i sva obeležja koja su označena kao reprezentativna na modelu povezanog entiteta. Klikom na dugme pored Combozoom komponente (označeno brojem 1 na Slici 65) aktivira se povezana forma u režimu za preuzimanje podataka (glavna alatna traka sadrži dugme, označenom brojem 2, za prenos podataka u polaznu formu). Na aktiviranoj formi korisnici mogu odabrati red u tabeli koji predstavlja instancu entiteta koju žele da povežu ili mogu uneti novu, ukoliko željena instanca ne postoji. Aktivirana forma je potpuno funkcionalna, tako da se mogu vršiti sve standardne operacije nad njenim podacima. Odabirom reda u tabeli, dugme za prenos podataka postaje aktivno i klikom na njega zatvara se aktivirana forma, a *Combozoom* komponenta se popunjava podacima iz odabranog reda.



Slika 65 Prikaz implementacije *Zoom* mehanizma u okviru web prototipa

Slika 66 prikazuje web implementaciju „Parent-child“ forme za manipulaciju podacima o radnicima i gradovima čija je skica prikazana na slici 59.



Slika 66 Primer implementacije složene forme u okviru web prototipa

Svi standardni paneli na složenoj formi su potpuno funkcionalni, tako da je moguće aktiviranje svih podržanih operacija nad podacima uključenih entiteta u okviru jedne ekranske forme. Rukovanje ovako implementiranom aplikacijom omogućava korisnicima evaluaciju

funkcionisanja modelovanog sistema tako što, kroz unos i pregled svojih realnih podataka, mogu primetiti da neka polja za unos nedostaju ili da navigacija nije odgovarajuća.

POGLAVLJE 5

EVALUACIONI EKSPERIMENT

Kako bi se istražila mogućnost korišćenja izvršivih prototipova u inicijalnoj fazi specifikacije zahteva u saradnji sa korisnicima, sproveden je eksperiment koji je uključivao seriju od 10 eksplorativnih studija slučaja u kojima su učestvovali autor disertacije i njegov mentor u ulozi projektanata (u nastavku teksta još nazvani članovi Kroki tima, odnosno istraživači) i po jedan učesnik (participant) u ulozi korisnika. U sklopu svake od studija organizovana je dvočasovna sesija na kojoj se radilo na specifikaciji poslovne aplikacije po izboru učesnika. Nakon obavljene sesije, učesnik je ispunjavanjem upitnika davao mišljenje o procesu rada kojem je prisustvovao i o dobijenom prototipu aplikacije. Kvalitativni i kvantitativni podaci prikupljeni tokom sesija i posle njih, putem upitnika, korišćeni su za izvođenje zaključaka o efikasnosti predloženog pristupa i pravcima daljih unapređenja. Razvijeni izvršivi modeli aplikacija, slike modela i pokrenutih prototipa u različitim fazama razvoja i popunjeni upitnici su javno dostupni i mogu se preuzeti sa repozitorijuma u okviru platforme Mendeley Data [179].

U prvom delu ovog poglavlja je predstavljen opisani eksperiment i dati su upitnici koji su korišćeni za dobijanje kvantitativnih podataka o studijama. U drugom delu su opisane pojedinačne sesije. Dati su podaci o učesniku, domenu aplikacije na čijoj specifikaciji zahteva je rađeno i zapažanja članova Kroki tima i učesnika koja predstavljaju izvor kvalitativnih informacija. Poslednji deo poglavlja sadrži diskusiju o rezultatima i o validnosti studije.

5.1. Dizajn eksperimenta

Evaluacioni eksperiment je sproveden kao niz od deset studija slučaja. Studije slučaja su izabrane kao tehnika istraživanja koja se oslanja pre svega na kvalitativne podatke. U skladu sa ciljem evaluacije, definisano je sledeće istraživačko pitanje i predlog studije:

- **Istraživačko pitanje**: Da li bi organizovanje kolaborativnih razvojnih sesija uz oslonac na izvršive prototipove bilo korisno u ranim fazama specifikacije zahteva poslovnih aplikacija?
- **Predlog studije**: Izvršivi prototipovi razvijeni u saradnji sa korisnicima mogu pospešiti međusobno razumevanje, korisničko zadovoljstvo i skratiti trajanje faze specifikacije zahteva.

5.1.1. Odabir učesnika

Prilikom dizajna eksperimenta je važno definisati kriterijume za odabir učesnika u cilju dobijanja validnih rezultata. Izdvojili smo grupe obaveznih i poželjnih kriterijuma. Obavezni kriterijumi za učesnike su bili:

1. Dolaze iz različitih poslovnih domena (grana industrije).
2. Dati domeni nisu poznati članovima razvojnog tima.
3. Ne bave se razvojem softvera.

4. Nemaju prethodna iskustva sa Kroki alatom.
5. Nisu ranije sarađivali sa članovima Kroki tima u ulozi naručioca softvera.

Poželjne osobine učesnika:

1. Zaposleni su u preduzećima u kojima koriste IS za obavljanje svakodnevnih poslova.
2. Bili su uključeni u razvoj IS-a u svojoj kompaniji u ulozi korisnika ili naručioca, da bi mogli da porede svoje ranije iskustvo sa onim iz sesije kolaborativnog razvoja.

Prvi i drugi obavezan kriterijum su imali za cilj izbegavanje korišćenja ranije stečenog znanja razvojnog tima o poslovnom domenu, što bi onemogućilo proveru efektivnosti komunikacije korišćenjem Kroki alata. S obzirom da članovi Kroki tima imaju iskustvo u razvoju različitih podsistema ERP (*Enterprise Resource Planning*) sistema, kao što su podsistemi za podršku rada računovodstva, upravljanja ljudskim resursima, magacinskim poslovanjem i sl., učesnici iz ovih domena nisu birani.

Treći, četvrti i peti obavezan kriterijum služe da bi se izbegao uticaj ekspertize učesnika na tok razvojne sesije.

Prethodno iskustvo u korišćenju aplikacija i njihovom razvoju u ulozi korisnika je poželjna osobina, da bi korisnici znali šta razvijamo i da bi mogli da porede svoje iskustvo u radu na sesiji sa svojim prethodnim iskustvom.

Iako smo uključivanjem studenata u eksperiment mogli da lakše dođemo do učesnika, smatramo da studenti računarstva na predstavljaju adekvatnu zamenu za realne domenske eksperte. Ova pretpostavka je bazirana na prethodnim iskustvima sprovođenja eksperimenata sa studentima u okviru naše institucije ali i šire naučne zajednice [180]. Studenti su u odvojenom istraživanju korišćeni za testiranje funkcionalnosti Kroki alata u ulozi projektanata za potrebe odvojenog istraživanja [146, 181].

U svim sesijama učestvovala su dva ista istraživača kako bi uticaj njihovih ličnosti na sesije bio uniforman. Jedan od članova tima je rukovao Kroki alatom, dok je drugi bio više uključen u proces donošenja odluka prilikom modelovanja. Oba člana razvojnog tima su ravnopravno učestvovala u komunikaciji sa učesnikom.

Učesnici u eksperimentu su zamoljeni da na sesije dođu sa idejom o poslovnoj aplikaciji koju će razvijati. To je mogla da bude nova aplikacija ili poboljšanje postojeće, ali je bilo neophodno da bude iz domena kojim se učesnik bavi. Preporučeno im je da na sesiju ponesu dokumentaciju (izveštaje, formulare, slike i sl.) koju koriste na svom poslu i koja bi mogla da pomogne u komunikaciji sa razvojnim timom. Komunikacija između istraživača i učesnika o željenoj aplikaciji pre sesije nije bila dozvoljena da bi se na svim sesijama radilo na ideji sa kojom se razvojni tim prvi put sreće. Trajanje sesija je ograničeno na dva sata da bi se mogao porediti broj razvijenih elemenata modela iz različitih sesija.

Sesije bi započinjale kratkim razgovorom u kojem bi učesnici opisivali svoj domen poslovanja i željenu aplikaciju, a članovi razvojnog tima postavljali pitanja u cilju dobijanja više detalja i razrešavanja nejasnoća. U trenutku kada bi članovi tima procenili da imaju dovoljno informacija, prešli bi na zajedničko modelovanje sistema. Modelovanje bi započinjalo skiciranjem formi u Kroki alatu koje podržavaju obavljanje jedne dogovorene funkcionalnosti sistema. Korišćen je editor za rad sa skicama korisničkog interfejsa za koji je trebalo da dokažemo hipotezu da se može iskoristiti za zajedničko modelovanje sa korisnicima.

5.1.2. Evaluacioni instrument

S obzirom da je cilj istraživanja traženje odgovora na pitanje da li korišćenje predloženog pristupa specifikaciji zahteva poslovnih aplikacija može pospešiti međusobno razumevanje i korisničko zadovoljstvo, odabrana je pretežno kvalitativna metoda istraživanja „posmatranje uz učešće“ (*Participant observation*) [179] u toku sesija. Posmatranje učesnika tokom zajedničkog rada nam omogućava da zabeležimo njihove reakcije, emocije i način ponašanja koji bi možda ostali nezabeleženi upotrebnom samo kvantitativnih metoda. Ipak, da bismo izbegli oslanjanje samo na impresije istraživača, kreirani su i upitnici za učesnike istraživanja kao tehnika kvantitativnih metoda istraživanja. Različiti uticaji koji dolaze kao posledica korišćene tehnologije ali i interakcija i emocija svih uključenih u razvojni proces zahteva uključivanje kvalitativnih i kvantitativnih metoda istraživanja, radi povećanja validnosti studije [75].

Kako bi utvrdili da li upotreba izvršivih prototipova može skratiti trajanje faze specifikacije, planirano je poređenje vremena potrebnog za sticanje međusobnog razumevanja prilikom rada na projektima na kojima nije korišćen Kroki alat, opisanim u [23]. Smatramo da je ovo poređenje moguće s obzirom da su sistemi razvijani na istim UI smernicama i uz oslonac na generatore koda, uz prisustvo istraživača koji je bio direktno uključen u razvoj korišćenjem oba navedena pristupa.

Prilikom pripreme studije, konsultovano je više stručnjaka iz različitih polja (softversko inženjerstvo, statistika, eksperimentalna psihologija) kako bi se došlo do predstavljenog evaluacionog instrumenta. Testiranje instrumenata i načina izvođenja sesije je obavljeno izvođenjem probne (nulte) sesije sa učesnikom koji je zaposlen u odeljenju za marketing jednog velikog preduzeća. Prema početnom planu, sesija je trebalo da traje četiri sata, jer smo smatrali da je to minimalno vreme za koje se može napraviti specifikacija zahteva. Nakon izvođenja nulte sesije zaključeno je da je dva sata dovoljno. Preformulisana su i neka od pitanja u upitnicima i odrađene su manje izmene na Kroki alatu radi lakšeg praćenja aktivnosti tokom sesija.

Za učesnike su razvijeni sledeći upitnici:

1. **Podaci o ispitaniku (A1)** – Služi za prikupljanje osnovnih informacija o ispitaniku (Tabela 5).
2. **Evaluacija kolaborativnog razvoja (A2)** – Upitnik putem kojeg učesnici mogu da izraze svoje mišljenje o sesijama zajedničkog razvoja i korišćenom alatu (Tabela 6). Rezultati ovog upitnika će služiti za proveru hipoteze H1.
3. **Evaluacija zajednički razvijenog prototipa (A3)** – Upitnik vezan za kvalitet i lakoću korišćenja generisanog prototipa (Tabela 7) čiji rezultati će biti korišćeni za proveru hipoteze H2.

Tabela 5 Upitnik o podacima *o ispitaniku*

Podaci o ispitaniku (A1)			
Opšti podaci	Pol, godine, zanimanje, obrazovanje, radno mesto		
Pitanja		**Skala**	
aQ1	Koliko ste vešti u korišćenju računara?	5-stepeni	Neiskusan - Vešt
aQ2	Koliko ste iskusni u korišćenju grafičkih programa na računaru?	5-stepeni	Neiskusan - Vešt
aQ3	Da li Vaše preduzeće poseduje informacioni sistem koji koristite u okviru svog radnog mesta? Ako je odgovor da, koliko ste vešti u njegovom korišćenju?	5-stepeni	Neiskusan - Vešt

aQ4	Da li ste u toku školovanja imali predmet ili ste pohađali kurs koji se bavio projektovanjem baza podataka ili modelovanjem softverskih sistema?	Da/Ne	
aQ5	Koliko ste puta imali priliku da kao korisnik ili naručilac učestvujete u razvoju softvera za Vaše preduzeće ili da sarađujete sa nekim softverskim timom?	5-stepeni	Nikad – Mnogo puta

Tabela 6 Upitnik za evaluaciju kolaborativnog razvoja

Evaluacija kolaborativnog razvoja (A2)			
Pitanja		**Skala**	
bQ1	Zajednički razvoj sa projektantom mi je pomogao da izrazim svoje zahteve.	5-stepeni	Ne slažem se – Slažem se
bQ2	Vreme koje sam proveo sa projektantom je trajalo predugo.	5-stepeni	Ne slažem se – Slažem se
bQ3	Tokom tog vremena je urađeno puno.	5-stepeni	Ne slažem se – Slažem se
bQ4	Ovakav način rada mi je bio iscrpljujući.	5-stepeni	Ne slažem se – Slažem se
bQ5	Bilo mi je teško da razumem pitanja projektanta.	5-stepeni	Ne slažem se – Slažem se
bQ6	Projektantu je bilo teško da razume šta želim.	5-stepeni	Ne slažem se – Slažem se
bQ7	Osećao sam se neprijatno i nekompetentno tokom rada sa projektantom.	5-stepeni	Ne slažem se – Slažem se
bQ8	Bilo bi mi lakše da svoje zahteve dostavim u pisanoj formi, bez direktne komunikacije.	5-stepeni	Ne slažem se – Slažem se
bQ9	Smatram da bi skiciranje ekrana na papiru bilo bolje sredstvo za komunikaciju sa razvojnim timom.	5-stepeni	Ne slažem se – Slažem se
bQ10	Pokretanje programa kojeg smo razvijali mi je pomoglo da lakše uočim šta mi je sve potrebno.	5-stepeni	Ne slažem se – Slažem se
bQ11	Unos podataka u pokrenutom programu olakšava otkrivanje grešaka.	5-stepeni	Ne slažem se – Slažem se
bQ12	Unos podataka u pokrenutom programu troši puno vremena.	5-stepeni	Ne slažem se – Slažem se
bQ13	Voleo bih da nastavim da radim na razvoju programa za moje preduzeće na ovaj način.	5-stepeni	Ne slažem se – Slažem se
bQ14	Smatram da bih specifikaciju mog programa korišćenjem Kroki alata mogao da izvršim i samostalno, bez prisustva projektanta.	5-stepeni	Ne slažem se – Slažem se
bQ15	Ako ste nekad učestvovali u razvoju programa sa nekim drugim projektantima ili programerima, molim Vas da napišete kakvi su Vaši utisci u poređenju sa tim iskustvom.	Slobodan unos	
bQ16	Komentari i sugestije.	Slobodan unos	

Tabela 7 Upitnik za evaluaciju razvijenog prototipa

Evaluacija zajednički razvijenog prototipa (A3)			
Pitanje		**Skala**	
cQ1	Koncepti programa su laki za razumevanje.	5-stepeni	Ne slažem se – Slažem se

cQ2	Elementi programa su laki za korišćenje.	5-stepeni	Ne slažem se – Slažem se
cQ3	Standardizovani izgled programa mi je intuitivno jasan.	5-stepeni	Ne slažem se – Slažem se
cQ4	Kretanje kroz program je olakšano zbog elemenata koji povezuju ekrane.	5-stepeni	Ne slažem se – Slažem se
cQ5	Smatram da ne bih mogao da uspešno koristim ovaj program bez prethodne obuke.	5-stepeni	Ne slažem se – Slažem se
cQ6	Smatram da ne bih mogao da uspešno koristim program bez pisanog uputstva.	5-stepeni	Ne slažem se – Slažem se
cQ7	Smatram da bi mi ovakav program olakšao zadatke na radnom mestu.	5-stepeni	Ne slažem se – Slažem se
cQ8	Izgled programa mi se sviđa.	5-stepeni	Ne slažem se – Slažem se
cQ9	Komentari i sugestije.	Slobodan unos	

Svaki od upitnika se sastoji od niza tvrdnji sa kojima korisnik treba da iskaže svoj stepen slaganja na petostepenoj Likertovoj skali [75], gde prvi stepen odgovara stavu *Slažem se*, dok peti stepen predstavlja stav *Ne slažem se*. Stavovi koji odgovaraju svim vrednostima skale su:

1. *Ne slažem se*
2. *Uglavnom se ne slažem*
3. *Nemam stav*
4. *Uglavnom se slažem*
5. *Slažem se*

Na kraju upitnika je ostavljena sekcija u kojoj ispitanici u slobodnoj formi mogu da unesu svoje komentare i sugestije. Prilikom izrade upitnika uzeti su u obzir poznati fenomeni koji se javljaju kao posledica korišćenja Likertove skale i preduzete su mere kako bi se minimizovao njihov efekat na kvalitet podataka [75][179]:

1. **Uticaj redosleda pitanja** – Redosled kojim su stavke u upitnicima poređane može imati uticaja na odgovore ispitanika. Kako bi se ovaj uticaj smanjio, pitanja su poslagana po nasumičnom redosledu.
2. **Sistematično (šablonsko) odgovaranje** – Usled raznih faktora, kod ispitanika se može javiti šablonsko odgovaranje na pitanja, pri čemu oni odgovore daju rutinski, bez razmišljanja o postavljenim tvrdnjama. Sa ciljem izbegavanja ove pojave, ispitanici su upitnike ispunjavali u svojim domovima nakon obavljene sesije kako bi se umanjio efekat umora i dosade. Pored ovoga, neke od izjava u upitnicima su negirane kako bi ispitanici morali dobro da ih pročitaju pre davanja odgovora.
3. **Pozitivna tendencija** – Predstavlja težnju ispitanika da se slažu sa postavljenim tvrdnjama bez obzira na njihov sadržaj [180]. Smatramo da jedan od faktora koji mogu dovesti do ovakvog ponašanja ispitanika predstavlja pritisak koji stvara prisustvo istraživača, tako da je mera koja podrazumeva ispunjavanje upitnika na mestu i u vreme koje odabere ispitanik uvedena kako bi se smanjila i ova tendencija.

Upitnik A1, *Podaci o ispitaniku*, sadrži sekciju za unos opštih podataka i niz pitanja putem kojih korisnici mogu da ocene svoje veštine u rukovanju informacionim sistemima i iskustva

prilikom razvoja istih (ukoliko ih imaju). Prilikom analize rezultata, odgovori su kodirani brojevima na skali od 1 do 5, gde broj 1 odgovara tvrdnji *Ne slažem se*, a broj 5 tvrdnji *Slažem se*.

5.2. Studije slučaja

U nastavku će biti pojedinačno predstavljene studije slučaja sprovedene u okviru istraživanja koje su bazirane na sesijama kolaborativnog razvoja sa korisnicima. U sesijama se radilo na kreiranju specifikacije korisničkih zahteva ciljne aplikacije uz oslonac na Kroki alat i izvršive prototipove. U Tabeli 8 je naveden spisak sprovedenih sesija zajedno sa naslovima projekata koji su u okviru njih specificirani i koji ukratko oslikavaju domen problema kojim se aplikacija bavi.

Tabela 8 Lista sprovedenih sesija

Sesija	Naslov projekta
S1	Procena rizika otplate kredita
S2	Baza stručnog usavršavanja
S3	IS pozorišnog muzeja
S4	Otkrivanje prevara u osiguranju
S5	Aplikacija za pomoć na putu
S6	Upravljanje projektima u štampariji
S7	IS servisa računara
S8	Registar dokumentacije poreskih obveznika
S9	Evidencija nastupa i takmičenja učenika muzičkih škola
S10	IS agencije za selidbe

Za svaku sesiju je navedeno vreme trajanja, broj skica različitih formi u kreiranom Kroki modelu kao i broj iteracija, pri čemu jedna iteracija predstavlja period kreiranja Kroki modela koji se završava pokretanjem prototipa. Nakon evaluacije izvršivog prototipa i povratkom u Kroki editor, može započeti nova iteracija. Broj iteracija nam može poslužiti kao mera koliko se korisnici oslanjanju na izvršivi prototip i unos podataka prilikom razvoja.

5.2.1. S1 - Procena rizika

Učesnica u ovoj sesiji zaposlena je na poziciji srednjeg menadžmenta u banci u okviru koje se bavi upravljanjem kreditima. Za obavljanje svog svakodnevnog posla koristi informacioni sistem razvijen za banku u kojoj radi. Za potrebe učešća u razvojnoj sesiji imala je predlog za unapređenjem postojećeg informacionog sistema kako bi se olakšala navigacija i pospešila preglednost podataka kojima svakodnevno manipuliše. Datom informacionom sistemu nedostaje mogućnost objedinjenog prikaza podataka o svim kreditima vezanim za određenog klijenta, tako da je fokus sesije bio na modelovanju ove funkcionalnosti u okviru Kroki alata.

Nakon kratkog uvoda, u kojem je učesnica predstavila svoju ideju projektantima, vođena je diskusija čiji je cilj sakupljanje svih potrebnih podataka za implementaciju date funkcionalnosti. U toku diskusije kreirano je nekoliko skica standardnih formi koji bi omogućile rukovanje podacima o ključnim entitetima u sistemu kao što su Klijent, Kredit, Račun, Valuta i sl. Sledeći korak u modelovanju je bio kreiranje „Parent-Child“ formi radi povezivanja podataka na način koji bi odgovarao potrebama učesnice. Posle ovoga je aktivirano generisanje koda i pokretanju prototipa.

Nakon nekoliko minuta koje je učesnica provela koristeći razvijeni prototip, kroz unos i pregled podataka, ustanovila je da elementi na nekim formama nisu organizovani adekvatno, kao i da nedostaju određeni podaci za neke od entiteta, što je zahtevalo povratak u Kroki editor formi, modifikaciju modela u skladu sa zahtevima i ponovno pokretanje prototipa. Nakon kratke evaluacije, učesnica je potvrdila da trenutna implementacija aplikacije zadovoljava njene potrebe.

S obzirom da je fokus ove sesije bio na modelovanju i implementaciji jedne nedostajuće funkcionalnosti, ona je trajala relativno kratko i okončana je u roku od 70 minuta, od kojih je veći deo (oko 45) potrošeno na uvodno upoznavanje sa domenom i diskusiju, pri čemu je kreirano ukupno 7 skica formi. U odnosu na prosečan broj formi razvijenih na sesijama (videti diskusiji rezultata), sistem modelovan na ovoj sesiji se može smatrati vrlo jednostavnim. Učesnica je izrazila veliko zadovoljstvo načinom rada i razvijenim prototipom, što se može videti i u odgovoru na pitanje *bQ16* u okviru upitnika *A2*, koje se tiče poređenja sa prethodnim iskustvom u razvoju softvera:

> *Ne može da se poredi. Prethodni razvoj je trajao mesecima a u projekte je bilo uključeno mnogo ljudi sa različitim stepenom razumevanja i razmišljanja. Ovo je bilo fantastično u poređenju sa bilo čime iz prethodnog iskustva.*

5.2.2. S2 - Baza stručnog usavršavanja

U ovom slučaju, učesnica istraživanja je bila 51-godišnja nastavnica koja je zaposlena i kao koordinator nastave u osnovnoj školi, što uključuje sakupljanje i katalogizaciju podataka o profesionalnom usavršavanju nastavnika, kao što su učestvovanje na seminarima i radionicama i objavljivanje stručnih publikacija. Ustanova u kojoj je učesnica zaposlena za ovaj tip posla ne obezbeđuje adekvatnu softversku podršku, tako da ona koristi informacije koje joj svako od nastavnika dostavlja u slobodnoj formi, kako bi sastavila godišnji izveštaj za svakog od njih korišćenjem alata Microsoft Word.

Dinamika rada na ovoj sesiji je bila slična onoj na sesiji S1, pri čemu je inicijalno upoznavanje sa domenom rada učesnice i prikupljanje podataka o ciljnoj aplikaciji trajalo 55 minuta. Nakon diskusije, modelovane su standardne forme koje bi omogućavale manipulaciju potrebnim podacima o svakom od nastavnika (entitet *Nastavnik*) kao i o njihovim usavršavanjima (entitet *Program stručnog usavršavanja*) koje su zatim objedinjene u *Parent-Child* formu na kojoj se mogu videti sva postignuća jednog nastavnika u toku željene školske godine. Nakon inicijalne evaluacije generisanog prototipa, popravljene su uočene greške te je, posle testiranja izmenjenog prototipa, ispitanica potvrdila da razvijeni sistem poseduje sve funkcije za podršku poslu koji je do tada obavljala ručno. Rezultat ove sesije, koja je trajala 95 minuta, je prototip sistema koji sadrži 8 skica različitih formi, tako da su obim rada i kompleksnost rezultujućeg sistema slični onima sa prethodne sesije (S1).

U odnosu na S1, u ovom slučaju je bilo potrebno više vremena za modelovanje manjeg broja formi. Na osnovu utisaka i beleški istraživača se može utvrditi da potrošeno dodatno vreme nije rezultat poteškoća u komunikaciji sa korisnikom, već je ispitanica bila mnogo detaljnija u svom inicijalnom izlaganju od prethodne. Slično kao i u prethodnom slučaju, iskazano zadovoljstvo učesnice je bilo na vrlo viskom nivou, pri čemu su posebne pohvale bile upućene grafičkim aspektima razvijenog prototipa, što se može videti iz komentara ostavljenog u okviru upitnika *A3* (pitanje *cQ9*) koji glasi:

> *Sviđa mi se što je interfejs jednostavan i funkcionalan bez opterećujućih grafičkih elemenata.*

Smatramo da je ovaj komentar vrlo koristan za evaluaciju UI standarda, s obzirom da je ispitanica po zanimanju akademska slikarka, pa je bilo očekivano da izrazi mišljenje o grafičkom delu korišćenih alata. Pored ovoga, izjasnila se da poseduje prethodna iskustva u razvoju softvera u ulozi klijenta i njen odgovor na pitanje *bQ15* glasi:

> *Kao klijent/korisnik sam učestvovala u razvoju nekoliko aplikacija. Ovo je daleko najbolje iskustvo. Ovako korisnik mnogo bolje sagledava potrebne funkcionalnosti i uviđa direktnu odgovornost za sve aspekte od strukture do dizajna ekrana. Definitivno je ovaj „tailor made" koncept odgovor na mnoge nedoumice i klijenta i projektanta. Verujem da će se mnogi budući klijenti hvaliti da su to sve sami isprojektovali, jedino se bojim da neće biti svesni radnih sati u bekgraundu.*

5.2.3. S3 – IS pozorišnog muzeja

Učesnik u ovoj sesiji je bio 43-godišnji arhivar zaposlen u Pozorišnom muzeju. U trenutku učestvovanja u eksperimentu, ispitanik je bio uključen na projektu digitalizacije muzejske kolekcije koji je tada bio u fazi planiranja, tako da je cilj sesije bila specifikacija budućeg informacionog sistema ovog muzeja. Ispitanik je zbog svega navedenog pokazao veliki entuzijazam za učestvovanje u eksperimentu.

Nakon uvodne diskusije, koja je trajala 50 minuta, procenjeno je da se može započeti za kreiranjem skica. Ustanovljeno je da željeni informacioni sistem treba da podrži upravljanje podacima o pozorištima (osnovni podaci, postojeće scene u okviru pozorišta i dr.), događajima koji se u njima izvode i učesnicima u tim događajima (glumcima, režiserima, scenaristima i ostalom osoblju). S obzirom da učesnici u evidentiranim događajima mogu da dolaze iz različitih zemalja, bilo je potrebno omogućiti višejezičnu katalogizaciju događaja.

Pored evidencije osnovnih podataka o spomenutim entitetima, bilo je potrebno omogućiti uvid u bibliografiju vezanu za svakog učesnika, događaj i pozorište. Ova bibliografija uključuje pregled istorije vezanih publikacija kao što su razni članci (najave, kritike, reklame) i monografije.

Finalni prototip je razvijen za 75 minuta, pri čemu su bile potrebne 4 iteracije kako bi se rešenje izbrusilo do svog finalnog oblika koji je uključivao 14 skica različitih ekranskih formi. Ispitanik je rekao da je uživao u radu na sesiji i izneo zadovoljstvo postignutim rezultatom, što je dodatno potkrepljeno njegovim interesovanjem za savlađivanje samostalnog korišćenja Kroki alata.

5.2.4. S4 – Otkrivanje prevara u osiguranju

Cilj ove sesije je bio unapređenje postojeće aplikacije koju učesnica koristi na svom radnom mestu u odeljenju za otkrivanje prevara jedne osiguravajuće kuće. Njen svakodnevni posao se u velikoj meri oslanja na efikasno organizovane podatke sa naprednim mogućnostima navigacije, što nije bilo na zadovoljavajućem nivou u njenoj trenutnoj aplikaciji.

Prilikom diskusije uočeno je da je osnovni entitet u ovom domenu prijava štete, pri čemu je potrebno voditi evidenciju i o polisama osiguranja, vlasnicima polisa i ostalim učesnicima u prijavi. Za prijavu štete može biti vezan proizvoljan broj pratećih dokumenata koji treba da se klasifikuju po tipu. Prijava služi kao osnova za računanje štete, uključujući i periodičnu finansijsku nadoknadu. Polise osiguranja u sistemu sadrže podatke o riziku i tarifi i potrebno je da mogu da se klasifikuju po predmetu osiguranja. Pored ovoga, sistem treba da podržava evidenciju o nadležnostima u okviru osiguravajuće kompanije koje se određuju na osnovu geografske lokacije i organizacione hijerarhije.

Nakon izlaganja i diskusije (50 minuta), zajedničkim radom su kreirani standardni paneli za rukovanje podacima o predstavljenim konceptima. Dati paneli su zatim organizovani u složene forme, kako bi se pospešila preglednost podataka i navigabilnost, što je bio jedan od osnovnih zahteva učesnice. U finalnoj verziji, razvijeni prototip se sastojao od 17 skica standardnih formi do kojih se došlo u pet iteracija. Ukupno vreme trajanja ove sesije je bilo 70 minuta.

Izražavajući svoje mišljenje putem upitnika, ispitanica je naglasila da je do sada učestvovala kao korisnik u razvoju softvera, pri čemu se prethodna iskustva ne mogu porediti sa iskustvom rada na sesiji (u korist kolaborativnog pristupa uz oslonac na Kroki alat).

5.2.5. S5 – Aplikacija za pomoć na putu

U izvođenju ove sesije učestvovao je 43-godišnji rukovodilac takođe zaposlen u osiguravajućoj kompaniji. Učesnik je na sesiju došao sa idejom o razvoju mobilne aplikacije koja bi omogućila kreiranje i kupovinu paketa osiguranja automobila u bilo kom trenutku. Pošto Kroki u svojoj trenutnoj verziji ne podržava specifikaciju korisničkog interfejsa mobilnih aplikacija, odlučeno je da će zadatak ove sesije biti specifikacija prototipa sistema koji bi služio kao serverski deo (back-end) željene aplikacije.

U okviru aplikacije je potrebno evidentirati podatke o karakteristikama podržanih vozila koje su relevantne sa stanovišta osiguranja kao i o kupcima polisa. Potrebna je i evidencija dostupnih paketa osiguranja, njihovih tarifa kao i vremenske komponente koja bi omogućila specifikaciju perioda važenja kupljenih paketa i polisa.

Učesnik, diplomirani ekonomista po struci, je u okviru svog univerzitetskog obrazovanja položio jednosemestralni predmet iz oblasti razvoja informacionih sistema. U upitniku A1 je uneo podatak da je učestvovao u razvoju informacionih sistema u ulozi naručioca više puta (izjavu aQ5 ocenio je sa maksimalnom vrednošću 5). Zahvaljujući ovom iskustvu, rad u sesiji nije uključivao često pokretanje izvršivog prototipa kako bi se evaluirala Kroki specifikacija, s obzirom da je bio u stanju da shvati sve potrebne detalje samo na osnovu UI skica. Pored ovoga, zahtevi ovog učesnika su bili vrlo jasno koncipirani i saopšteni na način da je razvojnom timu bilo lako da ih razume i implementira.

Prvih 40 minuta ove sesije je provedeno u izlaganju i diskusiji, dok je radni deo sesije trajao 50 minuta. Rezultat je specifikacija koja se sastoji od 20 skica ekranskih formi do kojih se došlo u dve iteracije, što čini ovu sesiju najproduktivnijom u ovom eksperimentu. Ocenjujući kolaborativni način rada, korisnik je izjavio da mu je u prethodnim projektima pisanje formalnih zahteva predstavljalo problem i ocenio pristup radu koji je praktikovan na sesiji kao kreativan, jednostavan i efikasan.

5.2.6. S6 – Upravljanje projektima u štampariji

Ulogu klijenta u ovoj eksperimentalnoj sesiji imala je 30-godišnja učesnica koja je imala ulogu u radu departmanske štamparije i taj angažman je bio povod za učestvovanje u eksperimentu. Naime, uočeno je da veliki broj štamparskih projekata ima problema sa izvođenjem faze pripreme štampe jer se komunikacija između klijenata i radnika obavlja neformalno, najčešće telefonom ili putem elektronske pošte. Učesnica je izrazila potrebu za poslovnom aplikacijom koja bi omogućila upravljanje štamparskim projektima koji bi im pomogao u prevazilaženju pomenutih problema.

U toku diskusije, zaključeno je da pomenuta aplikacija treba da omogući specifikaciju namenskog procesa koji bi bio prilagođen za svaki pojedinačni projekat. Identifikovan je skup faza koje bi služile kao gradivni blokovi za štamparske projekte. Prilikom specifikacije procesa rada na svakom projektu, korisnicima je omogućena specifikacija faza koje čine projekat, njihovog

redosleda izvršavanja kao i uslova koji su potrebni da budu ispunjeni kako bi se jedna faza smatrala kompletiranom i kako bi se moglo preći na sledeću.

Deo rezervisan za diskusiju je, prilikom izvođenja ove sesije, trajao gotovo 60 minuta što je posledica otežane komunikacije, jer su članovi razvojnog tima i učesnica imali problema da prenesu svoje ideje jedni drugima. Međutim, kada je postignuto osnovno razumevanje domena problema i pravac razvoja željene aplikacije i nakon što je započeo proces kreiranja skica, vrlo brzo je modelovana osnova za upravljanje potrebnim fazama rada na projektu. Pored ovoga, u toku kreiranja skica, zajedničkim radom se došlo do ideje za novu korisnu funkcionalnost koju ispitanica inicijalno nije imala na umu: da se podrži kreiranje šablonskih projekata sa predefinisanim skupom faza.

Iako je pojavljivanje ideja o novim korisnim dodacima u toku razvoja podrške za postojeći sistema vrlo često, smatramo da pristup modelovanju i razvoju koji je primenjen u ovom istraživanju pospešuje ovaj način kolaboracije i pruža tehničku osnovu da se nove ideje brzo implementiraju i evaluiraju.

Rezultat rada na ovoj sesije je Kroki model koji se sastoji od 16 skica različitih ekranskih formi, razvijen u ukupno tri iteracije pri čemu je faza modelovanja trajala 60 minuta. Na osnovu izloženog se može uočiti da je ovo jedina sesija na eksperimentu u okviru koje je iskorišćen ceo rezervisan vremenski okvir od 120 minuta.

5.2.7. S7 – Informacioni sistem servisa računara

Ulogu klijenta u ovoj studiji slučaja imao je 28-godišnji radnik jednog servisa računara i računarske opreme. Motivacija za učestvovanje je bila nezadovoljstvo ispitanika trenutnom aplikacijom koja se koristi kao podrška za evidenciju i naplaćivanje usluga koje servis u kojem je zaposlen koristi. Kao glavni nedostatak trenutne aplikacije učesnik je naveo nepostojanje strukturiranog i kontrolisanog unosa podataka, jer se informacije uglavnom unose kao slobodan tekst. Pored ovoga, postojeća aplikacija ne poseduje mogućnost kreiranja i korišćenja cenovnika, već su zaposleni primorani da cene usluga čitaju sa štampanih dokumenata i ručno unose prilikom svake naplate.

Kako bi se prevazišli navedeni problemi i unapredila efikasnost rada, modelovana je poslovna aplikacija koja podržava vođenje evidencije o zaposlenima, klijentima, uslugama koje pruža servis, izdatim računima, itd.

Ova sesija je trajala relativno kratko i završena je za 80 minuta. Uvodni deo sesije trajao je 45 minuta, dok je za razvoj modela utrošeno 35 minuta, pri čemu je kreiran model sa ukupno 13 skica formi u dve iteracije. Rezultat ove efikasnosti je prvenstveno familijarnost članova razvojnog tima sa domenom kompjuterske opreme kao i generička priroda dela informacionog sistema koji se odnosi na rukovanje cenovnicima i uslugama. Osim toga, učesnik je imao jasnu viziju željenog sistema pri čemu je bio vrlo jasan i koncizan u iznošenju svojih zahteva.

5.2.8. S8 – Registar dokumentacije poreskih obveznika

Ispitanica, 62-godišnja poreska inspektorka, je u ovoj sesiji učestvovala sa idejom razvoja poslovne aplikacije koja bi omogućila poreskim obveznicima katalogizaciju poreske dokumentacije. Ona je u toku svoje karijere radila sa velikim brojem poreskih obveznika koji nisu imali kompletiranu i urednu poresku dokumentaciju, uglavnom kao posledicu neobaveštenosti i otežanog pristupa informacijama. Željena aplikacija bi bila dostupna svim građanima i omogućavala bi im prijavu na svoj poreski profil i upravljanje aktivnostima vezanim za plaćanje poreza u skladu sa svojom poreskom kategorijom. Pored ovoga, aplikaciju bi mogli da koriste i poreski inspektori kako bi lakše pratili ove aktivnosti.

Članovi razvojnog tima su ovu sesiju ocenili kao vrlo zahtevnu. Veliki deo vremena je utrošen na pokušaje razumevanja detalja problemskog domena i korisničkih zahteva, pri čemu su istraživači imali utisak da je učesnica imala poteškoće da prenese svoje znanje o poreskom sistemu. S obzirom da je ispitanica, u trenutku izvođenja eksperimenta, bila već dve godine u penziji i bavila se drugačijom profesijom, postojala je potreba za proveravanjem informacija na službenim stranicama Poreske uprave, što je dodatno usporavalo rad. U pokušaju da se dođe do međusobnog razumevanja, modelovanje se radilo u kratkim iteracijama, tako što su kreirane skice koje bi podržale funkcionalnost za koju se smatralo da je razumevanje postignuto, nakon čega je učesnica pokušavala da koristi generisani prototip. Ona često nije mogla da odredi koje podatke treba da unese na kom mestu, što je bio znak da prethodna diskusija nije bila uspešna.

Kao rezultat opisanih poteškoća, razvojni tim je ocenio ovu sesiju kao neuspešnu, jer nisu bili u stanju da razumeju i implementiraju željene korisničke zahteve u zadatom vremenskom okviru. Uprkos ovome, ispitanica je putem upitnika unela podatke da razvijeni sistem odgovara onome što je imala na umu pre sesije i da je zadovoljna radom. Pretpostavka razvojnog tima, bazirana na iskustvima sa sesije, je bila da je ovakav odgovor ostavljen kako se ne bi povredila njihova osećanja i da ne predstavlja stvaran utisak ispitanice.

Za razliku od ostalih sesija, u ovom slučaju je diskusija trajala kraće od modelovanja što je posledica želje da se što pre započne sa radom na modelu kako bi se dalje funkcionalnosti sistema mogle ispregovarati uz oslonac na funkcionalni prototip. Inicijalni deo sesije trajao je 40 minuta dok je na modelovanje i isprobavanje utrošeno 80 minuta. Rezultat rada na ovoj sesiji je Kroki model koji sadrži ukupno 9 skica ekranskih formi razvijen u 14 iteracija.

5.2.9. S9 – Evidencija nastupa i takmičenja učenika muzičkih škola

Učesnica u ovoj studiji, 46-godišnja profesorica violine, imala je potrebu za razvojem poslovne aplikacije koja bi joj omogućila evidentiranje i praćenje nastupa učenika u njenoj klasi. Želela je da, uz pomoć ovog sistema, bude u mogućnosti da kreira poseban program rada za svakog učenika, u odnosu na njihove sposobnosti i ambicije. Program za jednog učenika se sastoji od liste muzičkih dela koje učenik treba da savlada u toku školske godine.

Pored ovoga, željeni sistem bi trebalo da omogući evidentiranje učestvovanja na muzičkim takmičenjima kao i ostalih vrsta nastupa. Jedna grupa koja učestvuje u nekom nastupu može biti sačinjena od učenika i profesora iz različitih muzičkih škola. Učenici i profesori mogu biti u različitim ulogama i svirati različite instrumente. Za evidentiranje nastupa bitno je zabeležiti vreme, mesto, učesnike i izvedena dela.

Nakon inicijalne diskusije koja je trajala 45 minuta prešlo se na izradu Kroki modela. Rezultat rada u okviru ove sesije je bio model sa skicama 18 formi razvijen u 6 iteracija. Ukupno trajanje sesije je bilo 70 minuta.

Prilikom svakog pokretanja izvršivog prototipa, učesnica je dobijala ideje o novim korisnim funkcionalnostima, tako da je prototip postepeno rastao kako je sesija odmicala. Po okončanju sesije, učesnica je zaključila da je razvijeni prototip funkcionalan u dovoljnoj meri da podrži navedene zahteve i pospeši njenu efikasnost na poslu. Izjavila je kako je uživala u vremenu provedenom na sesiji i ocenila proces rada kao vrlo kreativan. Članovi razvojnog tima su se složili sa njenim utiscima.

5.2.10. S10 – Informacioni sistem agencije za selidbe

U ulozi klijenta u ovoj studiji bila je 41-godišnja direktorka firme za pružanje usluga selidbe. Njen cilj je bio razvoj podsistema za upravljanje radnicima drugih firmi angažovanih od strane njene kompanije, kao i zadacima na koje su raspoređeni (radnim nalozima). U okviru ovog

podsistema je potrebno evidentirati podatke o radnicima kao što su ime, adresa, kontakt telefon, dostupnost, radne veštine, veličina uniforme i sl. Pored ovoga, jedan od osnovnih zahteva je omogućavanje raspoređivanja radnika na radne naloge i beleženje njihovog angažovanja (vremena provedenog obavljajući različite vrste posla u okviru jednog radnog naloga).

Uvodni deo sesije, posvećen upoznavanju domena i potreba učesnice, trajao je 50 minuta. Prlikom izrade Kroki modela, prvo je modelovano nekoliko jednostavnih formi koje bi omogućile upravljanje podacima o radnicima kao i jednu složenu formu koja bi rukovodiocima omogućila jednostavan i objedinjen uvid u angažovanje svih radnika. Nakon sprovedene evaluacije ovog dela pokretanjem prototipa, uočeni su nedostaci na nekima od standardnih formi, kao i potreba za drugačijom organizacijom složene forme. Model je prepravljen u skladu sa ovim primedbama i nakon ponovnog pokretanja, učesnica je zaključila da je ova verzija prototipa odgovarajuća.

Sledeći korak u kreiranju Kroki modela je predstavljao modelovanje radnih naloga i angažovanja radnika na zadacima. Ovo je obavljeno za mnogo kraće vreme jer su svi učesnici sesije bili u ovom trenutku dobro upoznati sa problemskim domenom i procesom razvoja modela.

Finalna verzija prototipa se sastojala od 11 ekranskih formi razvijenih u 2 iteracije, pri čemu je ukupno vreme trajanja sesije bilo 60 minuta. Ovi podaci pokazuju da je ova sesija bila vrlo efikasna, što je velikim delom posledica dobrog poznavanja posla i jasne vizije od strane učesnice, kao i laka komunikacija između nje i članova razvojnog tima. Ispitanica je izjavila da smatra kako je ovaj način rada i efikasnost razvoja mnogo bolji od onoga koji se sprovodi u njenoj kompaniji, gde se od trenutka unosa pisane specifikacije korisničkog zahteva do konačne implementacije funkcionalnosti čeka nekoliko meseci.

5.3. Analiza rezultata upitnika

U okviru opisane studije sprovedeno je ukupno 10 dvočasovnih sesija u kojima su učestvovali ispitanici u ulozi korisnika iz različitih poslovnih domena. Svaka od sesija predstavljala je slučaj specifikacije i evaluacije zahteva u saradnji sa učesnikom. U 9 od 10 sesija uspešno je realizovana inicijalna ideja u vidu modela kreiranog u okviru Kroki platforme i evaluiranog pomoću izvršivog prototipa. Broj skica ekranskih formi, koji je u ovom eksperimentu korišćen kao jedna od mera kompleksnosti modela, u specificiranim aplikacijama se kretao od 7 do 20 (Tabela 12). Prema zapažanjima članova istraživačkog tima i odgovorima datim u ponuđenim upitnicima, može se zaključiti da su iskustva učesnika većinom pozitivna, što je uticalo na efikasnost rada i prijatnu atmosferu na sesijama. Rezultati upitnika su dati u nastavku.

U tabeli 9 su dati rezultati upitnika A1, podaci o učesnicima.

Tabela 9 Rezultati upitnika *A1, Podaci o učesnicima*

Sesija	Pol	Godine	Zanimanje	Stručna sprema	aQ1	aQ2	aQ3	aQ4	aQ5
S1	Ž	44	Diplomirani ekonomista	bachelor	4	2	5	NE	4
S2	Ž	51	Akademski slikar - likovni pedagog	master	5	5	5	NE	4
S3	M	43	Arhivar - dokumentarista	SSS	5	5	5	DA	3
S4	Ž	42	Diplomirani ekonomista	bachelor	4	3	5	NE	3
S5	M	43	Diplomirani ekonomista	bachelor	4	2	4	DA	5
S6	Ž	30	Inženjer grafičkog inženjerstva i dizajna	Doktor nauka	5	5	5	DA	3
S7	M	28	Serviser računara	VSS	5	4	5	DA	4
S8	Ž	62	Poreski inspektor	bachelor	5	5	5	NE	3

S9	Ž	46	Profesor violine	master	3	3	5	NE	1
S10	Ž	41	Diplomirani pravnik	bachelor	4	2	5	NE	3

U tabeli 10 su prikazani rezultati upitnika A2, *Evaluacija kolaborativnog razvoja*, putem kojeg su ispitanici iznosili svoje utiske o načinu rada primenjenom na sesijama. Upitnik se sastoji od 16 izjava sa kojima su ispitanici izražavali stepen svog slaganja, odnosno neslaganja, na skali od 1 do 5. U tabeli su prikazane ocene učesnika, sa prosečnim ocenama prikazanim u u poslednjem redu.

Većina izjava u upitniku su afirmativnog karaktera, ali su neke izjave i negirane kako bi se izbegao efekat sistematskog odgovaranja. Prilikom analize ocena u upitnicima, pod pojmom „pozitivan stav“ smatraćemo stav koji se pozitivno odnosi na pristup koji se ispituje. Imajući ovo u vidu, afirmativne izjave su bliže pozitivnom stavu što je prosečna vrednost njihovih ocena bliža vrednosti 5. Negirane izjave su bliže pozitivnom stavu što su vrednosti njihovih ocena bliže vrednosti 1 (neslaganjem sa negativnim stavom učesnici iskazuju svoje slaganje sa evaluiranim pristupom).

Na osnovu rezultata prikazanih u Tabeli 10 može se zaključiti da je stepen korisničkog zadovoljstva kolaborativnim razvojem podržanim Kroki alatom vrlo visok, što se ogleda u prosečnim ocenama afirmativnih izjava čije se vrednosti kreću oko 5 (maksimalan stepen slaganja sa izjavom) dok se ocene negiranih izjava kreću oko vrednosti 1, koja predstavlja minimalni stepen slaganja.

Tabela 10 Rezultati upitnika *A2, Evaluacija kolaborativnog razvoja*

Sesija	bQ1	bQ2	bQ3	bQ4	bQ5	bQ6	bQ7	bQ8	bQ9	bQ10	bQ11	bQ12	bQ13	bQ14
S1	5	1	5	1	1	1	1	1	1	4	5	1	5	2
S2	5	1	5	1	1	1	1	1	1	5	5	1	5	1
S3	5	1	5	1	1	1	1	1	1	5	5	1	5	2
S4	5	1	5	1	1	1	1	1	1	5	5	1	5	5
S5	5	1	5	1	1	2	1	1	1	5	5	1	5	1
S6	5	1	5	2	2	1	1	1	1	5	4	2	4	5
S7	5	1	5	2	1	1	2	1	2	5	5	1	5	1
S8	5	1	5	1	1	1	1	1	1	5	3	1	5	1
S9	5	1	5	1	1	2	2	1	1	5	5	1	5	1
S10	5	1	5	1	1	1	1	1	1	5	5	1	5	1
Prosek	**5**	**1**	**5**	**1.2**	**1.1**	**1.2**	**1.2**	**1**	**1.1**	**4.9**	**4.7**	**1.1**	**4.9**	**2**

Sa izuzetkom tvrdnje *bQ14*, odstupanja od maksimalne ocene, u slučaju afirmativnih izjava odnosno minimalne ocene, u slučaju negiranih, ne prelaze vrednost 0.3. Prosečno odstupanje od pozitivnog stava iznosi svega 0.17, tako da možemo smatrati da su učesnici jednoglasno ocenili pristup uspešnim.

Izjava bQ14 čija ocena odstupa za 3 poena glasi: *Smatram da bih specifikaciju programa korišćenjem Kroki alata mogao da izvršim i samostalno, bez prisustva projektanta.* Ona je uvrštena u upitnik radi provere da li bi se Kroki u svom tekućem obliku mogao iskoristiti i za korisnički razvoj (*end-user development*) [147]. Na osnovu ovog rezultata možemo smatrati da Kroki trenutno nije prilagođen samostalnom korišćenju od strane krajnjih korisnika, što nije ni bio cilj ovog istraživanja.

Ocene ostalih izjava upitnika A2 jasno pokazuju ujednačen pozitivan stav, što govori da je primarna misija razvijenog alata, a to je podrška korisničkog učešća u najranijim fazama specifikacije poslovnih aplikacija, postignuta. Ovaj zaključak se ogleda i u prosečnoj oceni tvrdnje

bQ13 (*Voleo/la bih da nastavim da radim na razvoju programa za moje preduzeće na ovaj način)* koja iznosi 4.9.

Tabela 11 Rezultati upitnika *A3, Evaluacija zajednički razvijenog prototipova*

Sesija	cQ1	cQ2	cQ3	cQ4	cQ5	cQ6	cQ7	cQ8
S1	5	5	5	5	3	5	5	5
S2	5	5	4	4	5	5	5	5
S3	5	5	5	5	3	3	3	4
S4	5	5	5	5	1	1	1	5
S5	5	5	5	5	5	5	5	5
S6	5	4	4	3	2	2	4	4
S7	5	5	4	4	1	1	1	4
S8	5	5	5	5	1	1	1	5
S9	5	5	5	5	1	1	1	5
S10	5	5	5	5	1	1	1	5
Prosek	**5**	**4.9**	**4.7**	**4.6**	**2.3**	**2.3**	**3**	**4.7**

Tabela 11 prikazuje rezultate upitnika A3, putem kojeg su korisnici mogli da iskažu svoje stavove o pokrenutom prototipu. U odnosu na rezultate upitnika A2, stavovi po pitanju ovih tvrdnji nisu u tolikoj meri usaglašeni. Ovo se može zaključiti na osnovu činjenice da maksimalno odstupanje prosečne ocene od pozitivnog stava u ovom upitniku iznosi čak 2 (tvrdnja cQ6), dok prosečno odstupanje svih prosečnih ocena u ovom slučaju ima vrednost 0,71. Imajući ovo u vidu, sa stanovišta evaluacije razvijenog prototipa najzanimljivije su tvrdnje cQ5, cQ6 i cQ7 koje su ocenjene prosečnim ocenama koje najviše odstupaju od pozitivnog stava, kao i tvrdnja cQ1 koja je jednoglasno pozitivno ocenjena.

Tvrdnje cQ5 i cQ6 se odnose na lakoću samostalnog korišćenja razvijenog prototipa. Putem cQ5 ispitanici iskazuju stav prema mogućnosti korišćenja prototipa bez prethodne obuke, a putem cQ6 bez pisanog priručnika. Prosečna ocena ovih tvrdnji ima vrednost 2.3 i odgovara stavu *Ne slažem se*. S obzirom da su dali pozitivne ocene za karakteristike prototipa kao što su razumljivost elemenata, lakoća korišćenja, jednostavnost navigacije i sl., zaključujemo da se korisnici gneralno ne osećaju sigurno bez obezbeđene obuke i uputstava. Pri prvom pokretanju je istraživač rukovao pokrenutim prototipom i demonstrirao način korišćenja, dok je većina učesnika koristila prototip samostalno odmah u sledećoj iteraciji. Za aplikacije implementirane korišćenjem istog UI standarda bilo je dovoljno do pola sata obuke [27].

Prosečan stav iskazan prema izjavi cQ7 (*Smatram da bi mi ovakav program olakšao zadatke na radnom mestu*) ima prosečnu vrednost 3 i odgovara stavu *Nemam stav*. Iz ovog zaključujemo da izvršivi prototip u datom obliku nije pogodan za instaliranje kod korisnika, što mu i nije bila namena. Akcenat je bio na jednostavnosti i izostavljanju vizuelnih detalja koji bi odvlačili pažnju od bitnijih aspekata aplikacije. Za potrebe implementiranja realne aplikacije bilo bi potrebno kreirati odvojene generatore koda kod kojih akcenat ne bi bio na brzini generisanja i pokretanja, već na dobijanju kvalitetne web aplikacije.

5.4 Zapažanja istraživača

Prema oceni istraživača, većina razvojnih sesija (9 od 10) su bile veoma uspešne sa stanovišta efikasnosti rada, lakoće razumevanja i brzine otklanjanja grešaka. U nekim situacijama su istraživači i korisnici imali poteškoće u razumevanju poslovnog domena i zahteva, ali je kreiranje i pokretanje prve skice omogućilo da se barijera u komunikaciji prevaziđe. Kada bi učesnici videli na koji način se njihove ideje realizuju u vidu izvršivog prototipa, bili su u stanju da daju preciznije

zahteve ili da revidiraju prethodno izrečene. Pokretanje prototipa omogućilo im je navigaciju kroz forme i manipulaciju podacima, što im je dalo priliku da uoče greške u specifikaciji (pogrešno raspoređene ili nedostajuće UI komponente, neadekvatno kombinovanje formi, neodgovarajuća ili nedostajuća navigacija) koje su prošle neopaženo tokom diskusije. Mogućnost da u roku od nekoliko minuta provere svoje ideje, unesu ispravke i ponovo sve provere, uticala je vrlo pozitivno na njihovo samopouzdanje i volju za nastavkom rada. Iako su u pitanju bile simulacije sesija a ne realan razvoj, učesnici bi počeli da vrlo angažovano i sa velikim entuzijazmom daju svoj doprinos.

Povremeno nerazumevanje je bilo očekivana posledica činjenice da su poslovni domeni birani tako da budu nepoznati istraživačima. Smatramo da ovo predstavlja jaku stranu istraživanja jer verno simulira stvarne situacije u kojima se razvojni timovi i klijenti nalaze prilikom specifikacije korisničkih zahteva.

Korisnici koji su ranije više sarađivali sa razvojnim timovima imali su manje potrebe za pokretanjem prototipova u toku sesije u odnosu na manje iskusne učesnike. Oni su bili precizniji u izlaganju i mogli su da predvide većinu bitnih koncepata i detalje implementacije već na osnovu Kroki skica.

Moguća negativna strana ovakvih sesija je da su bile veoma intenzivne za razvojni tim. Bio je potreban značajan napor kako bi se komuniciralo sa učesnicima, evidentirali njihovi zahtevi i kreirale skice na osnovu njih. Na sesijama je učestvovalo dvoje istraživača što je donekle omogućilo podelu posla, ali bi brojniji razvojni timovi omogućili efikasniji i lakši rad. Ipak, izvođenje više od dve ovakve sesije sa realnim korisnicima dnevno bi verovatno predstavljalo preveliko opterećenje za članove razvojnog tima, bez obzira na njihov broj.

5.5 Merenje efikasnosti razvoja

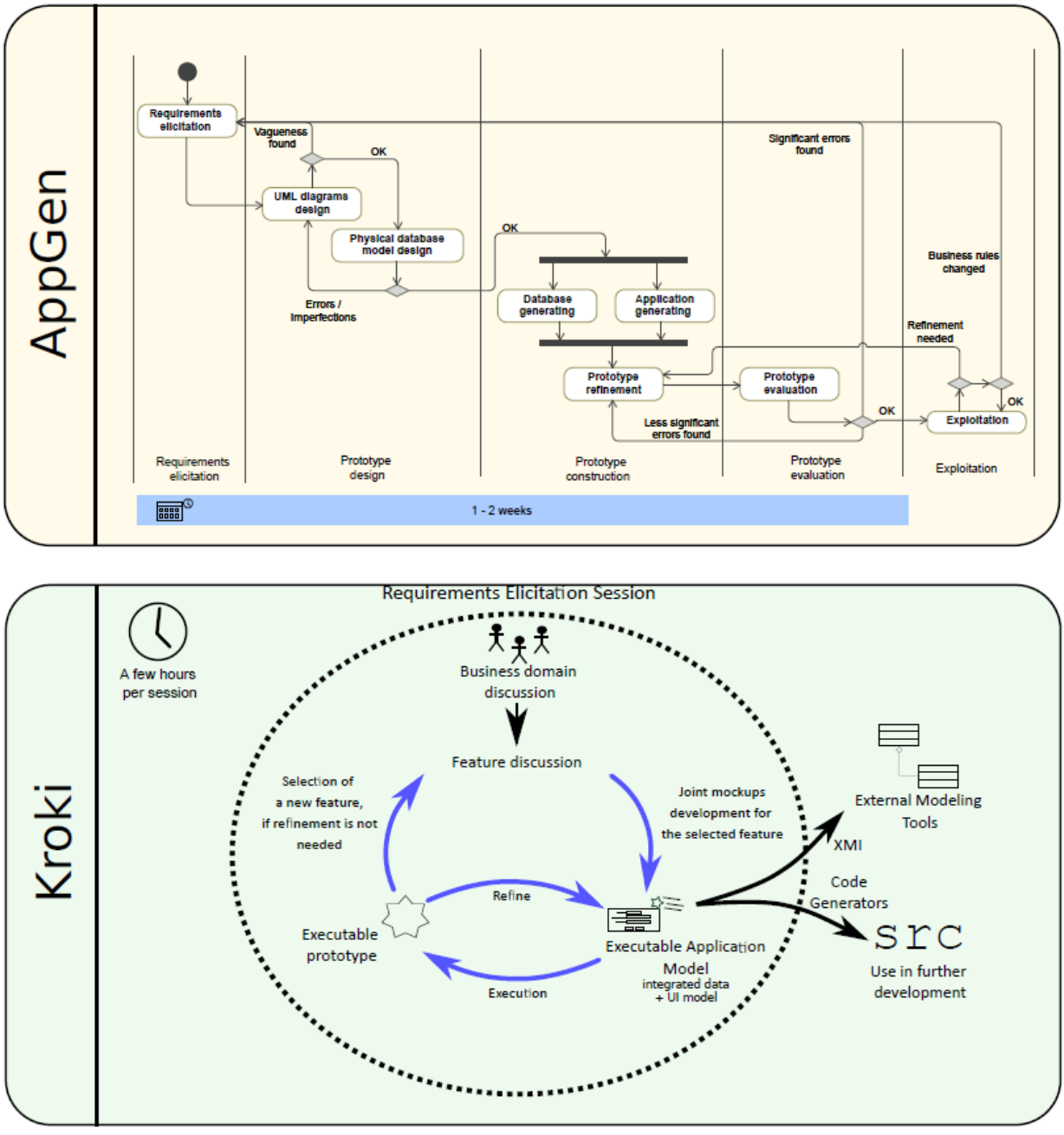
Tabela 12 prikazuje sumarne podatke sprovedenih sesija sa stanovišta trajanja i broja razvijenih skica. Prosečan broj skica ekranskih formi na razvijenim prototipovima iznosi 13.3, što je očekivano s obzirom na vremensko ograničenje sesija. Prosečno vreme provedeno u kreiranju jedne ekranske skice je 4.67 minuta.

Tabela 12 Sumarni podaci sa sesija

Sesija	Naziv	Broj skica	Broj iteracija	Trajanje sesije	Trajanje modelovanja	skica/min.	Uspešno?
S1		7	2	70	25	3.57	Da
S2		8	2	95	40	5	Da
S3		14	4	125	75	5.36	Da
S4		17	5	120	70	4.12	Da
S5		20	2	130	90	4.5	Da
S6		16	3	120	60	3.75	Da
S7		13	2	80	35	2.69	Da
S8		9	14	120	80	8.89	Ne
S9		18	6	105	45	2.5	Da
S10		11	2	110	70	5.36	Da
		13.3	**4.2**	**107.5**	**59**	**4.67**	

Kako bi utvrdili da li ovakav način rada može dovesti do efikasnijeg dobijanja proverene specifikacije zahteva, izvršićemo poređenje sa sesijama na kojima je korišćeno UML modelovanje i AppGen generator koda [23]. U oba slučaja su sistemi razvijani na istim UI smernicama, na bazi MDSD alata, uz učešće istog projektanta.

Slika 67 daje uporedni prikaz ova dva pristupa. Za generisanje izvršivog prototipa primenom AppGen alata bio je potreban period od jedne do dve sedmice. Proces se sastojao od posebne sesije prikupljanja zahteva sa korisnicima, nakon koje je razvojni tim pristupao fazi samostalnog modelovanja sa ciljem razvoja UML dijagrama klasa. Nedostaci i nedorečenosti prikupljenih zahteva su najčešće otkrivani u ovoj fazi, ali se moralo čekati do sledeće iteracije specifikacije zahteva kako bi se dobilo razjašnjenje od strane korisnika. Svaka od opisanih sesija trajala je u proseku nekoliko časova. Tek posle nekoliko dana je dijagram klasa bio spreman za generisanje prototipa koji bi korisnici mogli da testiraju.



Slika 67 Poređenje razvojnih sesija na bazi AppGen i Kroki alata

U odnosu na ovaj tempo, pristup baziran na Kroki izvršivim prototipovima donosi dramatično unapređenje efikasnosti jer omogućava izvršavanje nekoliko iteracija prikupljanja korisničkih zahteva, modelovanja korišćenjem konkretne sintakse koja je razumljiva korisnicima i evaluacije upotrebnom izvršivog prototipa u okviru jedne sesije.

Iako bi za razvoj realnog sistema istog obima putem Kroki alata trebalo više vremena od jedne dvočasovne sesije, povećanje bi se merilo u satima ili danima, ne u sedmicama. Za realan razvoj bilo bi potrebno sprovesti pripremnu sesiju za sticanje generalnog uvida u strukturu i nefunkcionalne zahteve tražene aplikacije. Na sesijama predstavljenim u ovom istraživanju prototipovi su kreirani specifikacijom samo bazičnih podešavanja u okviru Kroki alata, dok bi za specifikaciju naprednih i perzistentnih atributa trebalo dodatno vreme, u trajanju još jedne sesije u kojoj bi učestvovali samo projektanti (ovo smo testirali na sesiji S3 na aplikaciji za pozorišni muzej).

5.6 Provera hipoteza

Hipoteza H0, koja glasi:

> *Korišćenje izvršivih modela u saradnji sa korisnicima u najranijim fazama razvoja dovodi do boljeg razumevanja između razvojnog tima i korisnika i skraćuje vreme specifikacije zahteva.*

je evaluirana na osnovu rezultata upitnika A2 i A3 i poređenjem sa sesijama baziranim na AppGen alatu. Afirmativne izjave upitnika A2 su ocenjene prosečnom ocenom 4.41 što odgovara stavu *Uglavnom se slažem*, dok negirane izjave imaju prosečnu ocenu koja iznosi 1.11 i odgovaraju stavu *Ne slažem se*. Kako bismo dobili konačan rezultat, potrebno je normalizovati ocene ova dva tipa izjava. U tu svrhu potrebno je invertovati vrednosti ocena negiranih izjava, tako da budu u rasponu od 1 do 5, sa održavanjem odstupanja od maksimalne ocene. Normalizovane ocene su prikazane u Tabeli 13.

Tabela 13 Normalizovane ocene upitnika A2

Izjava	bQ1	bQ2	bQ3	bQ4	bQ5	bQ6	bQ7	bQ8	bQ9	bQ10	bQ11	bQ12	bQ13	bQ14
Prosek	5	5	5	4.8	4.9	4.8	4.8	5	4.9	4.9	4.7	4.9	4.9	3

Prosečna normalizovana ocena upitnika A2 (*Evaluacija kolaborativnog razvoja*) ima vrednost 4.68 što odgovara stavu *Slažem se*.

Prosečan stav ispitanika u upitniku A3 se može dobiti računanjem prosečne ocene svih izjava, bez potrebe za njihovom normalizacijom. Ona iznosi 3.93, što odgovara stavu *Uglavnom se slažem*. Iako je ova ocena nešto niža od prosečne ocene upitnika A2, i dalje pokazuje pozitivan odnos ispitanika prema izgledu i funkcionalnosti pokrenutih prototipova.

Smatramo da ove dve ocene, u kombinaciji sa zapažanjima učesnika i istraživača tokom sesija, pokazuju da je prvi deo hipoteze, koji se tiče boljeg razumevanja između razvojnog tima i korisnika, dokazan. Drugi deo hipoteze koji se tiče skraćenja vremena specifikacije zahteva je dokazan u sekciji 5.4, poređenjem sa sesijama baziranim na AppGen alatu.

Ovim možemo smatrati hipotezu H0 potvrđenom.

Rezultati upitnika A2 u kombinaciji sa zapažanjima istraživača potvrđuju i hipotezu H1:

> *Jezik specifičan za domen modelovanja poslovnih aplikacija čija konkretna sintaksa izgledom oponaša skice korisničkog interfejsa je pogodan kao sredstvo komunikacije između projektanata i naručioca softvera*

Hipoteza H2, koja glasi:

Moguće je razviti alat koji omogućava modelovanje na jeziku iz hipoteze H1 i direktno izvršavanje modela tako da korisnici i članovi razvojnog tima, u okviru sesija zajedničkog razvoja (tzv. kolaborativni dizajn), mogu da efikasno testiraju kvalitet specifikacije i stepen uzajamnog razumevanja.

dokazana je implementacijom i korišćenjem Kroki alata na opisanim sesijama. Zaključno sa ovom ocenom, smatramo da je eksperiment dao pozitivnu ocenu o predloženom načinu rada i razvijenim alatom.

5.7 Validnost studije

Kako bi se osigurao zadovoljavajući nivo validnosti prikazane studije, korišćena je tehnika triangulacije, koja podrazumeva kombinovanje kvalitativnih i kvantitativnih izvora podataka na osnovu kojih bi se steklo temeljno razumevanje posmatranog fenomena [192]. U sprovedenoj studiji, ovi izvori podataka uključuju: opservacije istraživačkog tima i učesnika, rezultate upitnika i poređenje sa prethodnim pristupom razvoja informacionih sistema opisanim u [23]. Oba člana istraživačkog tima su u toku sesija pisala beleške kako bi mogli da uporede zapažanja nakon sesija. Za merenje vremena korišćen je ispis u okviru prozora sa porukama Kroki alata, kako bi se precizno evidentiralo izvršavanje značajnih akcija, kao što je generisanje i pokretanje prototipa.

Potencijalna slabost ove studije je broj učesnika - može se smatrati da deset učesnika nije dovoljno za sprovođenje značajne statističke analize. Iako je ovo tačno kada je reč o kvantitativnim tehnikama istraživanjima, za kvalitativne tehnike, kakve su studije slučaja i „posmatranje uz učestvovanje“, nekada je i samo jedan, dobro odabran slučaj dovoljan da dokaže ili opovrgne neku tvrdnju [75]. Većina alata razmatranih u sekciji 2.2.4 je testirana na samo jednom slučaju. Za učesnike su često birani studenti, pošto je do realnih korisnika teško doći. U ovoj studiji je velika pažnja posvećena odabiru učesnika koji su realni domenski eksperti iz različitih poslovnih domena, kako bi eksperimentalne sesije predstavljale što verniju simulaciju realnih sesija specifikacija korisničkih zahteva. Studija je pažljivo pripremljena, uz konsultovanje više stručnjaka iz različitih oblasti i izvođenje nulte sesije, radi provere i podešavanja metode rada i upitnika.

U pojedinačnim sesijama je obavljano inicijalno upoznavanje sa domenom, prikupljanje i specifikacija zahteva i njihova provera na bazi izvršivih prototipova, za šta je u realnom razvoju potrebna barem još jedna pripremna sesija. Nisu razmatrani ni nefunkcionalni zahtevi. Prilikom modelovanja su korišćeni samo osnovni parametri, ne i oni potrebni za detaljnu specifikaciju perzistentnog sloja i naprednih osobina aplikacije. Uključivanje pripremne sesije i dodavanje detalja potrebnih za razvoj funkcionalne aplikacije bi povećalo vreme razvoja, ali bi ono i dalje bilo značajno kraće nego na bazi pristupa opisanog u [23].

Kvalitet kreiranog modela i brzina razvoja zavisi od veština projektanata koji učestvuju u sesijama. Neko ko poseduje iskustvo u analizi zahteva i konceptualnom modelovanju će kreirati kvalitetno rešenje koje se može koristiti za dalji razvoj. Neiskusan projektant će verovatno kreirati rešenje koje zahteva dodatno refaktorisanje, ali i dalje dovoljno dobro da posluži kao osnova za komunikaciju sa korisnicima.

S obzirom da se radi o eksplorativnoj studiji čiji zadatak je da ispita mogućnost ovakvog načina rada, dobijeni rezultati su ohrabrujući i otvaraju prostor za nastavak istraživanja. Analizirajući rezultate upitnika, može se uočiti gotovo jednoglasno iskazano korisničko zadovoljstvo sprovedenim sesijama i predloženom načinom rada. Iako ovo nedvosmisleno govori u korist korišćenog alata i pristupa razvoju, moramo uzeti u obzir da je u svakoj od sesija ulogu

klijenta imao tačno jedan učesnik. Ovaj nedostatak suprotstavljenih mišljenja, sloboda izražavanja i fokusiranost članova razvojnog tima na želje i zahteve jedne osobe potencijalno predstavlja faktor koji je pozitivno uticao na iskazano zadovoljstvo. Kako bi se ispitao ovaj uticaj, potrebno je sprovesti sličnu studiju koja bi uključivala više učesnika iz iste kompanije koji bi imali ulogu klijentskog tima.

POGLAVLJE 6

ZAKLJUČAK

Problemi koji mogu da ugroze projekat dolaze u velikoj meri iz faze specifikacije zahteva, često usled uzajamnog nerazumevanja korisnika i razvojnog tima. Široko prihvaćen način za rešavanje ovog problema u okviru različitih metodologija razvoja softvera je kreiranje ranih prototipova, na osnovu kojih bi korisnici mogli da provere da li su njihovi zahtevi pravilno shvaćeni. Međutim, jeftini prototipovi sa stanovišta utroška vremena i energije razvojnog tima često ne uspevaju da pobude dovoljnu pažnju korisnika i pomognu da se otkriju omaške u specifikaciji sistema. Detaljniji prototipovi daju bolje rezultate, ali mogu dovesti do iscrpljivanja razvojnog tima, posebno ako se na kraju odbace [76].

Istraživanje predstavljeno u okviru ove disertacije imalo je za cilj unapređenje procesa specifikacije korisničkih zahteva poslovnih aplikacija na bazi detaljnih, izvršivih prototipova koji se mogu kreirati uz minimalan utrošak vremena i energije. Radi postizanja ovog cilja je implementiran alat otvorenog koda pod nazivom Kroki (fr. *croquis* – skica) čija je arhitektura projektovana tako da obezbedi:

- kolaborativni razvoj specifikacije poslovne aplikacije sa korisnicima koji nemaju znanje projektovanja i programiranja softverskih sistema,
- efikasno pokretanje prototipa direktno iz sopstvenog razvojnog okruženja, dajući mogućnost korisniku da isproba prototip tokom modelovanja kad god poželi,
- ponovno korišćenje informacija dobijenih prilikom razvoja prototipova u kasnijim fazama razvoja, kako bi se smanjilo nepotrebno trošenje resursa.

Specifikacija poslovnih aplikacija koja se kreira u okviru Kroki alata je bazirana na internom UI standardu i EUIS (*Enterprise User Interface Specification*) UML profilu, razvijenim u okviru ranijeg istraživanja u okviru disertacije [27]. Internim UI standardom su definisane funkcionalne i vizuelne karakteristike minimalno potrebnog broja UI komponenti poslovne aplikacije, sa namerom da se obezbedi jednostavno korišćenje aplikacije, brza obuka korisnika i automatizacija izrade korisničkog interfejsa. EUIS UML profil je projektovan sa ciljem da se postigne brzo modelovanje korisničkog interfejsa poslovnih aplikacija opisanih UI standardom i jednostavna integracija modela korisničkog interfejsa sa modelom podataka (perzistentnim modelom), korišćenjem UML dijagrama klasa kao zajedničke osnove. Modelovanje ciljnog sistema korišćenjem EUIS UML profila je podrazumevalo: kreiranje modela podataka nezavisnog od platforme korišćenjem dijagrama klasa, njegovu automatsku transformaciju u model korisničkog interfejsa i u model podataka specifičan za određenu implementacionu platformu, ručnu popravku i dopunu modela korisničkog interfejsa i generisanje koda za ciljnu platformu. Zbog potrebe za M2M i M2T transformacijama, kao i specifikacije aplikacije koja je počinjala razvojem dijagrama klasa, razvojni proces nije bio prilagođen radu sa korisnikom, tako da je evaluacija prototipa bila moguća u razmaku od nekoliko dana od datuma prikupljanja zahteva [27].

U okviru Kroki alata je implementiran EUIS DSL— samostalan jezik na bazi EUIS UML profila. Njegov meta-model se sastoji od meta-modela korisničkog interfejsa i perzistentnog meta-modela koji su integrisani zajedničkim meta-klasama koje implementiraju pojednostavljeni `UML::Kernel`.

Zahvaljujući tome, model aplikacije je sve vreme integrisan i ne postoji više potreba za M2M transformacijama koje su bile prisutne u ranijem rešenju [27] i u većini analiziranih rešenja opisanih u drugom poglavlju.

Za specifikaciju poslovne aplikacije obezbeđene su tri konkretne sintakse EUIS DSL-a koju implementiraju odgovarajući editori Kroki alata:

- grafička sintaksa u vidu skica korisničkog interfejsa prilagođena radu sa korisnikom, koju implementira editor skica,
- grafička sintaksa koja podseća na UML dijagram klasa sa stereotipima, prilagođena za efikasan rad projektanata, koju implementira pojednostavljeni UML editor (*Lightweight* UML editor) i
- konkretna sintaksa tekstualnog komandnog jezika, namenjena članovima razvojnog tima kojima ne odgovaraju grafičke sintakse.

Podrška za efikasno pokretanje je implementirana na više nivoa:

- Zahvaljujući integrisanom meta-modelu EUIS DSL-a izbegnuto je trošenje vremena na M2M transformacije.
- Grafički editori omogućavaju specifikaciju poslovnih aplikacija uz unos samo osnovnih obeležja.
- Korišćenjem adaptivnih generičkih aplikacija koje dinamički prilagođavaju svoj izgled i funkcionalnost na osnovu podataka preuzetih iz modela je u velikoj meri skraćeno vreme koje se troši na M2T transformacije. Adaptivne generičke aplikacije su unapred implementirane, tako da se većinom generišu konfiguracione datoteke koje diktiraju izgled i funkcionalnosti prototipa. Ovi podaci se smeštaju u aplikativni repozitorijum koji je optimizovan za brz pristup. Podrška za unos ručnih izmena je obezbeđena putem aspekt-orijentisanog programiranja.
- H2 sistem za rukovanje bazama podataka koji je uključen u Kroki alat obezbeđuje perzistenciju unetih podataka tokom korišćenja prototipa, bez potrebe za dodatnim konfigurisanjem. Međutim, podržano je i korišćenje proizvoljnog sistema za rukovanje bazama podataka, ukoliko razvojni tim tako odabere.

S obzirom da su kao osnova jezika zadržane `UML::Kernel` meta-klase, mogućnost saradnje sa alatima za modelovanje opšte namene je očuvana, kroz uvoz i izvoz specifikacije u vidu dijagrama klasa zapisanim u XMI formatu. Programski kod razvijenog prototipa se takođe može eksportovati u Eclipse projekat. Na ovaj način se postiže ponovno korišćenje informacija dobijenih prilikom razvoja prototipova za potrebe kasnijih faza razvoja softvera i uključivanje Kroki alata u proizvoljno mesto u razvojnom procesu.

U disertaciji su definisane tri hipoteze koje su proverene eksperimentom opisanim u petom poglavlju:

- **H0**: Korišćenje izvršivih modela u saradnji sa korisnicima u najranijim fazama razvoja dovodi do boljeg razumevanja između razvojnog tima i korisnika i skraćuje vreme specifikacije zahteva.
- **H1**: Jezik specifičan za domen modelovanja poslovnih aplikacija čija konkretna sintaksa izgledom oponaša skice korisničkog interfejsa je pogodan kao sredstvo komunikacije između projektanata i korisnika.

- **H2**: Moguće je razviti alat koji omogućava modelovanje na jeziku iz hipoteze H1 i direktno izvršavanje modela tako da korisnici i članovi razvojnog tima, u okviru sesija zajedničkog razvoja (tzv. kolaborativni dizajn), mogu da efikasno testiraju kvalitet specifikacije i stepen uzajamnog razumevanja.

Eksperiment je uključivao 10 eksplorativnih studija slučaja u kojima su učestvovali autor disertacije i njegov mentor u ulozi projektanata i po jedan učesnik u ulozi korisnika. U sklopu svake od studija organizovana je dvočasovna sesija na kojoj se radilo na specifikaciji poslovne aplikacije po izboru učesnika. Nakon obavljene sesije, učesnik je ispunjavanjem upitnika davao mišljenje o procesu rada kojem je prisustvovao i o dobijenom prototipu aplikacije. Kvalitativni i kvantitativni podaci prikupljeni tokom sesija i posle njih, putem upitnika, iskorišćeni su za izvođenje pozitivnih zaključaka i potvrde hipoteza. Razvijeni izvršivi modeli aplikacija, slike modela i pokrenutih prototipa u različitim fazama razvoja i popunjeni upitnici su javno dostupni i mogu se preuzeti sa repozitorijuma u okviru platforme Mendeley Data [179]. Dizajn istraživanja je baziran na konceptima MEM-a (*Method Evaluation Model*), a upitnici koji evaluiraju jezik za modelovanje i Kroki alat su formulisani tako da odgovaraju izabranim konceptima FQUAD (*Framework for qualitative assessment of domain-specific languages*) okvira za evaluaciju DSL-ova [53].

Kako bi se osigurao zadovoljavajući nivo validnosti prikazane studije, korišćena je tehnika triangulacije, koja podrazumeva kombinovanje kvalitativnih i kvantitativnih izvora podataka na osnovu kojih bi se steklo temeljno razumevanje posmatranog fenomena [192]. U sprovedenoj studiji, ovi izvori podataka uključuju: opservacije istraživačkog tima i učesnika, rezultate upitnika i poređenje sa prethodnim pristupom razvoja informacionih sistema opisanim u [23]. Oba člana istraživačkog tima su u toku sesija pisala beleške kako bi mogli da uporede zapažanja nakon sesija. Za merenje vremena korišćen je ispis u okviru prozora sa porukama Kroki alata, kako bi se precizno evidentiralo izvršavanje značajnih akcija, kao što je generisanje i pokretanje prototipa.

Studije su uključivale realne domenske eksperte, za razliku od eksperimenata analiziranih u drugom poglavlju, gde su kao ispitanici korišćeni studenti. Analizirani eksperimenti su se sastojali od najviše tri studije slučaja, dok je eksperiment opisan u petom poglavlju imao deset studija.

Merenje efikasnosti dobijanja proverene specifikacije je izvršeno poređenjem sa sesijama na kojima je korišćeno UML modelovanje i prethodno rešenje, AppGen generator koda iz [23]. U oba slučaja su sistemi razvijani na istim UI smernicama, na bazi MDSD alata, uz učešće istog projektanta. Za generisanje izvršivog prototipa primenom AppGen alata bio je potreban period od jedne do dve sedmice, dok je korišćenjem Kroki alata bilo dovoljno nekoliko sati. Pristup baziran na Kroki izvršivim prototipovima donosi dramatično unapređenje jer omogućava izvršavanje nekoliko iteracija prikupljanja korisničkih zahteva, modelovanja korišćenjem konkretne sintakse koja je razumljiva korisnicima i evaluacije upotrebnom izvršivog prototipa u okviru jedne sesije.

Realna primena saradnje Kroki alata sa alatima za modelovanje i programiranje opšte namene je prikazana u [174] za potrebe implementacije izvršivog CERIF (*Common European Research Information Format*) [175, 169] modela koji predstavlja standard za opis naučnih publikacija koji se koristi kao podloga u CRIS (*Current Research Information System*) informacionom sistemu na nivou Evropske unije. CERIF model je preuzet sa euroCRIS[37] web stranice i uvezen u Kroki alat. Više od 280 entiteta iz ovog modela je organizovano u 31 paket kako bi se poboljšala preglednost modela. Izvršivi prototip je izvezen u vidu Eclipse Java projekta i u njega je ručno dodata logika za podršku prikaza podataka o povezanim entitetima na više jezika,

[37] https://eurocris.org/

radi prilagođavanja izgleda prototipa potrebama CRIS sistema. Uvezeni model, u vidu Kroki datoteke, zajedno sa prototipom se može preuzeti sa GitHub platforme[38].

U [146, 181] je Kroki alat iskorišćen kao podloga za razvoj platforme za dinamičko prilagođavanje meta-podataka koji opisuju obrazovne resurse u digitalnim repozitorijumima. Platforma omogućava kreiranje modela koji se dinamički transformišu u web aplikaciju za upravljanje obrazovnim resursima. Testiranje korišćenja Kroki alata u svojstvu projektanata je tom prilikom obavljeno od strane korisnika koji poseduju tehničko znanje (studenata računarstva) a njihovi predlozi su iskorišćeni za poboljšanje funkcionisanja grafičkih editora.

Sve navedeno pokazuje da je Kroki alat implementiran u skladu sa ciljevima da obezbedi kolaborativni razvoj specifikacije poslovne aplikacije, efikasno pokretanje prototipa i ponovno korišćenje informacija dobijenih prilikom razvoja prototipova u kasnijim fazama razvoja.

Programski kod Kroki alata je javno dostupan za preuzimanje putem GitHub[39] platforme, tako da se može koristiti za potrebe drugih istraživača ili razvojnih timova iz industrije. Trenutna verzija Kroki alata je razvijena radi podrške ovde predstavljenom istraživanju i ne obezbeđuje implementaciju poslovnih aplikacija na industrijskom nivou kvaliteta. Pravci daljeg razvoja bi išli u tom smeru:

- stvaranja podrške za uključivanje generatora koda za generisanje poslovnih aplikacija za proizvoljnu platformu od strane timova koji bi želeli da koriste Kroki alat za realan razvoj.
- razvoj programskog interfejsa (API) koji bi omogućio uključivanje ručno pisanih proširenja bez potrebe detaljnog poznavanja arhitekture generičke Java aplikacije, radi lakše modifikacije postojećih izvršivih prototipa.

Takođe, planiraju se dodatne studije gde bi u sesijama učestvovalo više predstavnika korisnika, a razvijana poslovna aplikacija bila većeg obima.

[38] https://github.com/KROKIteam/KROKI-CERIF-Model
[39] https://github.com/KROKIteam/KROKI-mockup-tool

Literatura i reference

[1] Hussain, Z., Slany, W., & Holzinger, A. (2009, November). Current state of agile user-centered design: A survey. In Symposium of the Austrian HCI and Usability Engineering Group (pp. 416-427). Springer, Berlin, Heidelberg.

[2] Asaro, P. M. (2000). Transforming society by transforming technology: the science and politics of participatory design. Accounting, Management and Information Technologies, 10(4), 257-290.

[3] Martin, J., & Kleinrock, L. (1985). Excerpts from: An Information Systems Manifesto. Communications of the ACM, 28(3), 252-255.

[4] Highsmith, J., & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *IEEE Computer, 34*, 120-122.

[5] Cohen, D., Lindvall, M., & Costa, P. (2003). Agile software development: A dacs state-of-the-art report. Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland.

[6] Constantine, L. L., & Lockwood, L. (2002). Process agility and software usability: Toward lightweight usage-centered design. Information Age, 8(8), 1-10.

[7] Kelly, S., & Tolvanen, J. P. (2008). Domain-specific modeling: enabling full code generation. John Wiley & Sons.

[8] Schmidt, D. C. (2006). Model-driven engineering. Computer-IEEE Computer Society-, 39(2), 25.

[9] France, R. B., Ghosh, S., & Turk, D. E. (2001). Towards a model-driven approach to reuse. In OOIS 2001 (pp. 181-190). Springer, London. A. van Deursen and J. Visser, “Domain-specific languages: an annotated bibliography,”

[10] Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. ACM Sigplan Notices, 35(6), 26-36.

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

[12] Dejanović, I. (2011). Prilog metodama brzog razvoja softvera na bazi proširivih jezičkih specifikacija (Doctoral dissertation, University of Novi Sad, Faculty of Technical Science).

[13] Object Management Group (2015). Unified Modeling Language (UML) V2.5, https://www.omg.org/spec/UML/2.5.

[14] Bézivin, J., & Gerbé, O. (2001, November). Towards a precise definition of the OMG/MDA framework. In Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001) (pp. 273-280). IEEE.

[15] Siegel, J. M. (2014). Model driven architecture (MDA)–MDA Guide rev. 2.0. Object Management Group, Tech. Rep. ORMSC/14-06-0.

[16] Sewall, S. J. (2003). Executive Justification for Adopting Model Driven Architecture (MDA). Object Management Group, Tech. Rep. A. Cockburn, Agile Software Development, Addison-Wesley, Boston, 2002.

[17] Cockburn, A. (2006). Agile software development: the cooperative game. Pearson Education.

[18] Fowler, M., & Highsmith, J. (2001). The agile manifesto. Software development, 9(8), 28-35.

[19] Dominguez, J. (2009). The curious case of the chaos report 2009. Project Smart, 1-16.

[20] Rivero, J. M., Grigera, J., Rossi, G., Luna, E. R., Montero, F., & Gaedke, M. (2014). Mockup-driven development: providing agile support for model-driven web engineering. Information and Software Technology, 56(6), 670-687.

[21] Martin, A., Biddle, R., & Noble, J.W. (2004). The XP customer role in practice: three studies. *Agile Development Conference*, 42-54.

[22] Rivero, J.M., Grigera, J., Rossi, G., Luna, E.R., & Koch, N. (2011). Improving Agility in Model-Driven Web Engineering. *CAiSE Forum*.

[23] Milosavljević, G., & Perišić, B. (2004). A method and a tool for rapid prototyping of large-scale business information systems. Computer Science and Information Systems, 1(2), 57-82.

[24] Perišić, B., Milosavljević, G., Dejanović, I., & Milosavljević, B. (2011). UML profile for specifying user interfaces of business applications. Computer Science and Information Systems, (18), 405-426.

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

[26] Filipović, M., Vuković, Ž., Dejanović, I., & Milosavljević, G. (2021). Rapid Requirements Elicitation of Enterprise Applications Based on Executable Mockups. Applied Sciences, 11(16), 7684.

[27] Milosavljević, G. (2010). Prilog metodama brzog razvoja adaptivnih poslovnih informacionih sistema, doktorska disertacija, Univerzitet u Novom Sadu, Fakultet tehničkih nauka.

[28] Rackov, M., Kaplar, S., Filipović, M., & Milosavljević, G. (2016). Java code generation based on OCL rules. In *6th International Conference on Information Society and Technology*. Kopaonik. (pp. 191-196).

[29] Vijayasarathy, L., & Turk, D. (2012). Drivers of agile software development use: Dialectic interplay between benefits and hindrances. Information and Software Technology, 54(2), 137-148.

[30] The practical adoption of agile methodologies - APM. (n.d.). Retrieved Jun 29, 2023, from https://www.apm.org.uk/media/1185/practical-adoption-of-agile-methodologies.pdf

[31] Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., & Astesiano, E. (2010, September). On the effectiveness of screen mockups in requirements engineering: results from an internal replication. In Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement (p. 17). ACM.

[32] Kalyuga, S. (Ed.). (2008). Managing cognitive load in adaptive multimedia learning. IGI Global.

[33] Demuth, B. (2004, September). The Dresden OCL toolkit and its role in Information Systems development. In *Proc. of the 13th International Conference on Information Systems Development (ISD'2004)* (Vol. 7).

[34] Oracle America, Inc., (2013), JSR-000345 Enterprise JavaBeans Specification, v3.2, Final Release, April 30, 2013

[35] Forward, A., Badreddin, O., Lethbridge, T. C., & Solano, J. (2012). Model-driven rapid prototyping with Umple. Software: Practice and Experience, 42(7), 781-797.

[36] Solano, J. (2010). Exploring how model oriented programming can be extended to the UI level. (Doctoral dissertation, University of Ottawa (Canada).

[37] Rivero, J. M., Rossi, G., Grigera, J., Burella, J., Luna, E. R., & Gordillo, S. (2010, July). From mockups to user interface models: an extensible model driven approach. In International Conference on Web Engineering (pp. 13-24). Springer, Berlin, Heidelberg.

[38] Störrle, H. (2010, August). Model driven development of user interface prototypes: an integrated approach. In Proceedings of the Fourth European Conference on Software Architecture: Companion Volume (pp. 261-268).

[39] Hovsepyan, A., & Van Landuyt, D. (2015). Prototizer: Agile on Steroids. In FlexMDE@ MoDELS (pp. 51-60).

[40] Kamalrudin, M., & Grundy, J. (2011, November). Generating essential user interface prototypes to validate requirements. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (pp. 564-567). IEEE Computer Society.

[41] Beck, K. (2000). Extreme programming explained: embrace change. addison-wesley professional.

[42] Buchmann, T. (2012, October). Towards tool support for agile modeling: sketching equals modeling. In Proceedings of the 2012 Extreme Modeling Workshop (pp. 9-14).

[43] Coyette, A., Schimke, S., Vanderdonckt, J., & Vielhauer, C. (2007, September). Trainable sketch recognizer for graphical user interface design. In IFIP Conference on Human-Computer Interaction (pp. 124-135). Springer, Berlin, Heidelberg.

[44] Coyette, A., & Vanderdonckt, J. (2005, September). A sketching tool for designing anyuser, anyplatform, anywhere user interfaces. In IFIP Conference on Human-Computer Interaction (pp. 550-564). Springer, Berlin, Heidelberg.

[45] Plimmer, B., & Apperley, M. (2004, April). INTERACTING with sketched interface designs: an evaluation study. In CHI'04 extended abstracts on Human factors in computing systems (pp. 1337-1340).

[46] Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly, 319-340.

[47] Moody, D. L. (2003). The method evaluation model: a theoretical model for validating information systems design methods.

[48] Moody, D. (2003). Dealing with complexity in information systems modeling: Development and empirical validation of a method for representing large data models. ICIS 2003 Proceedings, 18.

[49] Moody, D. (1997, November). A multi-level architecture for representing enterprise data models. In International Conference on Conceptual Modeling (pp. 184-197). Springer, Berlin, Heidelberg.

[50] Feldman, P., & Miller, D. (1986). Entity model clustering: Structuring a data model by abstraction. The Computer Journal, 29(4), 348-360.

[51] Simsion, G. C. (1989). A structured approach to data modelling. Australian Computer Journal, 21(3), 108-117.

[52] Abrahão, S., Insfran, E., Carsí, J. A., & Genero, M. (2011). Evaluating requirements modeling methods based on user perceptions: A family of experiments. Information Sciences, 181(16), 3356-3378.

[53] Kahraman, G., & Bilgen, S. (2015). A framework for qualitative assessment of domain-specific languages. Software & Systems Modeling, 14(4), 1505-1526.

[54] Savić, G., Segedinac, M., & Konjović, Z. (2012). Automatic generation of E-Courses based on explicit representation of instructional design. Computer Science and Information Systems, 9(2), 839-869.

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

[56] Barišic, A., Amaral, V., Goulao, M., & Aguiar, A. (2014). Introducing usability concerns early in the DSL development cycle: FlowSL experience report. In *MD²P² 2014–Model-Driven Development Processes and Practices Workshop Proceedings* (p. 8).

[57] Challenger, M., Kardas, G., & Tekinerdogan, B. (2016). A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. Software Quality Journal, 24(3), 755-795.

[58] Rivero, J.M., Grigera, J., Distante, D., Simarro, F.M., & Rossi, G. (2017). DataMock: An Agile Approach for Building Data Models from User Interface Mockups. *Software & Systems Modeling, 18*, 663-690.

[59] Luna, E.R., Mayo, F.J., Rivero, J.M., García-García, J.A., Sánchez-Begínes, J.M., Cuaresma, M.J., & Rossi, G. (2017). Challenges for the Adoption of Model-Driven Web Engineering Approaches in Industry. *J. Web Eng.*.

[60] Joshi, B., & Joshi, B. (2016). Creational Patterns: Singleton, Factory Method, and Prototype. *Beginning SOLID Principles and Design Patterns for ASP. NET Developers*, 87-109.

[61] Brambilla, M., & Butti, S. (2014). Fifteen Years of Industrial Model-Driven Development in Software Front-End: from WebML to WebRatio and IFML. Novática.

[62] Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., & Fraternali, P. (2008). Web Applications Design and Development with WebML and WebRatio 5.0. *TOOLS*.

[63] Ceri, S., Fraternali, P., & Bongio, A. (2000). Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks, 33*, 137-157.

[64] Brambilla, M., & Fraternali, P. (2014). The interaction flow modeling language (IFML). In Technical Report. version 1.0. Object Management Group (OMG).

[65] Bernaschina, C., Comai, S., & Fraternali, P. (2017, May). IFML Edit. org: model driven rapid prototyping of mobile apps. In 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft) (pp. 207-208). IEEE.

[66] Kotonya, G., & Sommerville, I. (1998). Requirements engineering: processes and techniques. John Wiley & Sons, Inc..

[67] LaRoche, C., & Traynor, B. (2010). User-centered design (UCD) and technical communication: The inevitable marriage. *2010 IEEE International Professional Comunication Conference*, 113-116.

[68] Low-code platform for Digital Transformation. WebRatio. (n.d.). Retrieved Jun 29, 2023, from https://www.webratio.com/site/content/en/home

[69] Acerbis, R., Bongio, A., Brambilla, M., & Butti, S. (2007, July). Webratio 5: An eclipse-based case tool for engineering web applications. In International Conference on Web Engineering (pp. 501-505). Springer, Berlin, Heidelberg.

[70] Kleppe, A. G., Warmer, J. B., & Bast, W. (2003). *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.

[71] Kautz, K. (2010, March). Participatory design activities and agile software development. In IFIP Working Conference on Human Benefit through the Diffusion of Information Systems Design Science Research (pp. 303-316). Springer Berlin Heidelberg.

[72] Chamberlain, S., Sharp, H., & Maiden, N. (2006, June). Towards a framework for integrating agile development and user-centred design. In International Conference on Extreme Programming and Agile Processes in Software Engineering (pp. 143-153). Springer Berlin Heidelberg.

[73] Walsham, G. (1995). Interpretive case studies in IS research: nature and method. European Journal of information systems, 4(2), 74-81.

[74] Gould, J. D., & Lewis, C. (1985). Designing for usability: key principles and what designers think. Communications of the ACM, 28(3), 300-311.

[75] Shull, F., Singer, J., & Sjøberg, D. I. (Eds.). (2007). Guide to advanced empirical software engineering. Springer Science & Business Media.

[76] Bourque, P., Dupuis, R., Abran, A., Moore, J. W., & Tripp, L. (1999). The guide to the software engineering body of knowledge. IEEE software, 16(6), 35-44.

[77] Avison, D., & Fitzgerald, G. (2003). Information systems development: methodologies, techniques and tools. McGraw-Hill.

[78] Ambler, S. (2002). Agile Modelling: Effective Practices for Extreme Programming and the Unified Process. 2002.

[79] Thayer, R. H., Bailin, S. C., & Dorfman, M. (1997). Software requirements engineerings. IEEE Computer Society Press.

[80] Boehm, B. W. (1984). Verifying and validating software requirements and design specifications. IEEE software, (1), 75-88.

[81] Leffingwell, Dean, and Don Widrig. Managing software requirements: a unified approach. Addison-Wesley Professional, 2000

[82] Lauesen, S. (2002). Software requirements: styles and techniques. Pearson Education.

[83] Deguzis, B., Harris, A., Matos, E., & Bullock, Z. (2018). Software Requirements Specification.

[84] Van Lamsweerde, A. (2000, June). Requirements engineering in the year 00: a research perspective. In Proceedings of the 22nd international conference on Software engineering (pp. 5-19).

[85] Brooks, F. P. (1974). The mythical man-month. Datamation, 20(12), 44-52.

[86] Ross, D. T., & Schoman, K. E. (1977). Structured analysis for requirements definition. IEEE transactions on Software Engineering, (1), 6-15.

[87] IEEE Computer Society. Software Engineering Technical Committee. (1983). IEEE standard glossary of software engineering terminology. Institute of Electrical and Electronics Engineers.

[88] De Gea, J. M. C., Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C., & Vizcaíno, A. (2012). Requirements engineering tools: Capabilities, survey and assessment. Information and Software Technology, 54(10), 1142-1157.

[89] Hoffmann, M., Kuhn, N., Weber, M., & Bittner, M. (2004, September). Requirements for requirements management tools. In Proceedings. 12th IEEE International Requirements Engineering Conference, 2004. (pp. 301-308). IEEE.

[90] Smith, M. F., & Smith, M. F. (1991). Software prototyping: Adoption, practice, and management (p. 216). London: McGraw-Hill.

[91] Hardgrave, B. C. (1995). When to prototype: Decision variables used in industry. Information and Software technology, 37(2), 113-118.

[92] Alter, S. (1998). Information systems. Addison-Wesley Longman Publishing Co., Inc.

[93] Vaderna R., (2018). Algoritmi i jezik za podršku automatskom raspoređivanju elemenata dijagrama, doktorska disertacija, Univerzitet u Novom Sadu, Fakultet tehničkih nauka

[94] Martin, J. (1991). Rapid application development. Macmillan Publishing Co., Inc.

[95] Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): an empirical review. European Journal of Information Systems, 8(3), 211-223.

[96] Mackay, H., Carne, C., Beynon-Davies, P., & Tudhope, D. (2000). Reconfiguring the user: Using rapid application development. Social studies of science, 30(5), 737-757.

[97] Stapleton, J. (1997). DSDM, dynamic systems development method: the method in practice. Cambridge University Press

[98] Martin, R. C. (2002). Agile software development: principles, patterns, and practices. Prentice Hall.

[99] Schwaber, K., & Beedle, M. (2002). Agile software development with Scrum (Vol. 1). Upper Saddle River: Prentice Hall.

[100] Agile Alliance, (n.d.). Retrieved Jun 29, 2023, from https://www.agilealliance.org/

[101] Highsmith, J. (2013). Adaptive software development: a collaborative approach to managing complex systems. Addison-Wesley.

[102] Palmer, S. R., & Felsing, M. (2001). A practical guide to feature-driven development. Pearson Education.

[103] 14th annual State of Agile Report (2019), https://explore.digital.ai/state-of-agile/14th-annual-state-of-agile-report

[104] Sharma, S., & Hasteer, N. (2016, April). A comprehensive study on state of Scrum development. In 2016 International Conference on Computing, Communication and Automation (ICCCA) (pp. 867-872). IEEE.

[105] Rumpe, B., & Schröder, A. (2014). Quantitative survey on extreme programming projects. arXiv preprint arXiv:1409.6599.

[106] Fowler, M., & Foemmel, M. (2006). Continuous integration.

[107] Stolberg, S. (2009, August). Enabling agile testing through continuous integration. In 2009 agile conference (pp. 369-374). IEEE.

[108] Schwaber, K. (1997). Scrum development process. In Business object design and implementation (pp. 117-134). Springer, London.

[109] Rubin, K. S. (2012). Essential Scrum: A practical guide to the most popular Agile process. Addison-Wesley.

[110] Takeuchi, H., & Nonaka, I. (1986). The new new product development game. Harvard business review, 64(1), 137-146.

[111] Iyengar, S. S. (2005). U.S. Patent No. 6,874,146. Washington, DC: U.S. Patent and Trademark Office.

[112] Dimitrieski, V. (2018). Model-Driven Technical Space Integration Based on a Mapping Approach. Doctoral Dissertation. University of Novi Sad, Faculty of Technical Sciences

[113] Fowler, M. (2010). Domain-specific languages. Pearson Education.

[114] Kosar, T., Mernik, M., Črepinšek, M., Henriques, P. R., da Cruz, D., Pereira, M. J. V., & Oliveira, N. (2009, October). Influence of domain-specific notation to program understanding. In 2009 International Multiconference on Computer Science and Information Technology (pp. 675-682). IEEE

[115] S Wile, D. (2001). Supporting the DSL spectrum. Journal of Computing and Information Technology, 9(4), 263-287.

[116] Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM computing surveys (CSUR), 37(4), 316-344.

[117] Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L. C., ... & Wachsmuth, G. (2013). DSL engineering: Designing, implementing and using domain-specific languages (p. 558). dslbook. org.

[118] Alanen, M., & Porres, I. (2004). A relation between context-free grammars and meta object facility metamodels. Turku Centre for Computer Science.

[119] Poltronieri, I., Zorzo, A. F., Bernardino, M., & de Borba Campos, M. (2018, April). Usa-DSL: usability evaluation framework for domain-specific languages. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing (pp. 2013-2021).

[120] Bariic, A., Amaral, V., & Goulao, M. (2012, September). Usability evaluation of domain-specific languages. In 2012 Eighth International Conference on the Quality of Information and Communications Technology (pp. 342-347). IEEE.

[121] Cameron, K. S., & Whetten, D. A. (1983). Some conclusions about organizational effectiveness. In Organizational effectiveness (pp. 261-277). Academic Press.

[122] ISO: ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models

[123] Flyvbjerg, B. (2006). Five misunderstandings about case-study research. Qualitative inquiry, 12(2), 219-245.

[124] Forward, A., Lethbridge, T. C., & Brestovansky, D. (2009, May). Improving program comprehension by enhancing program constructs: An analysis of the Umple language. In 2009 IEEE 17th International Conference on Program Comprehension (pp. 311-312). IEEE.

[125] Lethbridge, T. C., Forward, A., & Badreddin, O. (2010, October). Umplification: Refactoring to incrementally add abstraction to a program. In 2010 17th working conference on reverse engineering (pp. 220-224). IEEE.

[126] Lano, K. (Ed.). (2009). UML 2 semantics and applications. John Wiley & Sons.

[127] Störrle, H. (2014). From Pen-and-Paper Sketches to Prototypes: The Advanced Interaction Design Environment. In T. Yue, & B. Combemale (Eds.), Demonstrations at MODELS 2014: Proceedings of the Demonstrations Track of the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (DEMO-MODELS 2014), October 1st and 2nd, 2014 CEUR Workshop Proceedings Vol. 1255 http://ceur-ws.org/Vol-1255

[128] Feldt, K. C. (2007). Programming Firefox: Building rich internet applications with XUL. " O'Reilly Media, Inc.".

[129] Rivero, J. M., Rossi, G., Grigera, J., Luna, E. R., & Navarro, A. (2011, October). From interface mockups to web application models. In International conference on web information systems engineering (pp. 257-264). Springer, Berlin, Heidelberg.

[130] Naiburg, E., Naiburg, E. J., & Maksimchuck, R. A. (2001). UML for database design. Addison-Wesley Professional.

[131] Koch, N., Knapp, A., & Zhang, G. (2007). UML Based Web engineering: an approach based on standards. Web Engineering: Modelling and Implementing Web Applications, HCI Se-ries, vol. 12, No. 7.

[132] France, R., & Rumpe, B. (2007, May). Model-driven development of complex software: A research roadmap. In Future of Software Engineering (FOSE'07) (pp. 37-54). IEEE.

[133] Fieber, F., Regnat, N., & Rumpe, B. (2014). Assessing usability of model driven development in industrial projects. arXiv preprint arXiv:1409.6588.

[134] Prototizer, Retrieved Jun 29, 2023, from https://prototizer.com/

[135] Oldevik, J. (2006). MOF Script user guide. Version 0.6 (MOFScript v 1.1. 11), 21.

[136] Kitchenham, B. (1993). A methodology for evaluating software engineering methods and tools. In Experimental Software Engineering Issues: Critical Assessment and Future Directions (pp. 121-124). Springer, Berlin, Heidelberg.

[137] Caldiera, V. R. B. G., & Rombach, H. D. (1994). The goal question metric approach. Encyclopedia of software engineering, 528-53 2.

[138] Macbeth, G., Razumiejczyk, E., & Ledesma, R. D. (2011). Cliff's Delta Calculator: A non-parametric effect size program for two groups of observations. Universitas Psychologica, 10(2), 545-555.

[139] Alhir, S. S. (2006). Guide to Applying the UML. Springer Science & Business Media.

[140] Ristanović, G. (2001) Razvojno okruženje sa minimalno potrebnim brojem funkcija za razvoj složenih softverskih proizvoda, Magistarska teza, Fakultet tehničkih nauka.

[141] Carlson, D. (2001). Modeling XML applications with UML (Vol. 6). Upper Saddle River^ eNJ NJ: Addison-Wesley.

[142] Holt, J., & Perry, S. (2008). SysML for systems engineering (Vol. 7). IET.

[143] Apple Computer, Inc. (1992). Macintosh human interface guidelines. Addison-Wesley Professional.

[144] Lowney, G. C. (1995) The Microsoft Windows Guidelines for Accessible Software Design. Microsoft Corporation.

[145] Richters, M., & Gogolla, M. (2002). OCL: Syntax, semantics, and tools. In Object Modeling with the OCL (pp. 42-68). Springer, Berlin, Heidelberg.

[146] Alhaag Abdyssalam, A. (2018) Model-Driven Software Architecture for the Management of Educational Resources Metadata, doktorska disertacija, Univerzitet u Novom Sadu, Fakultet tehničkih nauka

[147] Barricelli, B. R., Cassano, F., Fogli, D., & Piccinno, A. (2019). End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, *149*, 101-137.

[148] Singh, Y., & Sood, M. (2009, July). Models and Transformations in MDA. In *2009 First International Conference on Computational Intelligence, Communication Systems and Networks* (pp. 253-258). IEEE.

[149] Object Management Group. (2014). Object Constraint Language (OCL) Version 2.4, https://www.omg.org/spec/OCL/2.4

[150] Object Management Group. (2016). Meta Object Facility (MOF) Core Specification, Version 2.5.1, https://www.omg.org/spec/MOF/2.5.1/PDF

[151] Ivković, Ž., Vaderna, R., Filipović, M., Milosavljević, G., & Dejanović, I. (2014). Implementacija podloge za saradnju kroki alata sa alatima za UML modelovanje opšte namene, YuInfo, Kopaonik

[152] MDT/UML2/UML2 4.0 Migration Guide (n.d.). Retrieved Jun 29, 2023, from http://wiki.eclipse.org/MDT/UML2/UML2_4.0_Migration_Guide

[153] Vaderna, R., Vuković, Ž., Dejanović, I., & Milosavljević, G. (2018). Graph Drawing and Analysis Library and Its Domain-Specific Language for Graphs' Layout Specifications. Scientific Programming, 2018.

[154] Petre, M. (1995). Why looking isn't always seeing: readership skills and graphical programming. Communications of the ACM, 38(6), 33-44.

[155] Engelen, L., & van den Brand, M. (2010). Integrating textual and graphical modelling languages. Electronic Notes in Theoretical Computer Science, 253(7), 105-120.

[156] Scheidgen, M. (2008, June). Textual modelling embedded into graphical modelling. In European Conference on Model Driven Architecture-Foundations and Applications (pp. 153-168). Springer, Berlin, Heidelberg.

[157] Filipović, M., Kaplar, S., Vaderna, R., Ivković, Ž., Milosavljević, G., & Dejanović, I. (2015, March). Aspect-oriented engines for Kroki models execution. In 5th International Conference on Information Society Technology and Management (ICIST) (pp. 502-507).

[158] Popovici, A., Gross, T., & Alonso, G. (2002, April). Dynamic weaving for aspect-oriented programming. In Proceedings of the 1st international conference on Aspect-oriented software development (pp. 141-147).

[159] Object Management Group (2015), XML Metadata Interchange (XMI), https://www.omg.org/spec/XMI/2.5.1/PDF

[160] Fuentes-Fernández, L., & Vallecillo-Moreno, A. (2004). An introduction to UML profiles. *UML and Model Engineering*, *2*(6-13), 72.

[161] Dejanović, I. (2021) Jezici specifični za domen, udžbenik, Fakultet tehničkih nauka, Novi Sad

[162] Dmitriev, S. (2004). Language oriented programming: The next programming paradigm. *JetBrains onboard*, *1*(2), 1-13

[163] Stahl, T., Völter, M. (2006), Model Driven Software Development: Tehnology, Engineering, Management, John Wiley & Sons, Ltd.

[164] Greifenberg, T., Hölldobler, K., Kolassa, C., Look, M., Nazari, P. M. S., Müller, K., ... & Wortmann, A. (2015, February). A comparison of mechanisms for integrating handwritten and generated code for object-oriented programming languages. In *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)* (pp. 74-85). IEEE.

[165] Object Management Group (2015). Interaction Flow Modeling Language (IFML), v1.0, February 2015

[166] Kaplar, S., Filipović, M., Milosavljević, G., & Sladić, G. (2015, March). Kroki Administration Subsystem Based on RBAC Standard and Aspects. In *5th International Conference on Information Society Technology and Management (ICIST)* (pp. 61-66).

[167] Marsenić V., (2012), Interaktivno razvojno okruženje za specifikaciju poslovnih aplikacija, Master rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka

[168] Filipović, M., Marsenić V., Milosavljević, G., & Dejanović, I. (2013). Kroki: Alat Za Interaktivni Razvoj Poslovnih Aplikacija Baziran Na Skicama, XIX konferencija YU INFO 2013, Kopaonik

[169] Jörg, B. (2010). CERIF: The common European research information format model. *Data Science Journal*, *9*, CRIS24-CRIS31.

[170] Parr, T. J., & Quong, R. W. (1995). ANTLR: A predicated-LL (k) parser generator. *Software: Practice and Experience*, *25*(7), 789-810.

[171] Trkulja Z., (2015), Implementacija tekstualne sintakse za EUIS DSL, Master rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka

[172] Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *EMF: eclipse modeling framework*. Pearson Education.

[173] Caprile, B., & Tonella, P. (2000, October). Restructuring Program Identifier Names. In icsm (pp. 97-107).

[174] Filipović, M., Vaderna, R., Ivković, Ž., Kaplar, S., Vuković, Ž., Dejanović, I., ... & Ivanović, D. (2017). Application of Kroki mockup tool to implementation of executable CERIF specification. Procedia Computer Science, 106, 245-252.

[175] Jeffery, K., Houssos, N., Jörg, B., & Asserson, A. (2014). Research information management: the CERIF approach. International Journal of Metadata, Semantics and Ontologies, 9(1), 5-14.

[176] Rackov, M., (2015), Implementacija izvedenih polja korišćenjem OCL izraza, Master rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka

[177] Kiczales, G., & Hilsdale, E. (2001, September). Aspect-oriented programming. In Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering (p. 313).

[178] Gradecki, J. D., & Lesiecki, N. (2003). Mastering AspectJ: aspect-oriented programming in Java. John Wiley & Sons.

[179] Filipović, M., Milosavljević, G. (2021), "Rapid Requirements Elicitation Sessions Based on Executable Mockups", Mendeley Data, V1, doi: 10.17632/kbh7hjttzy.1, https://data.mendeley.com/datasets/kbh7hjttzy/1

[180] Svahnberg, M., Aurum, A., & Wohlin, C. (2008, October). Using students as subjects-an empirical evaluation. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (pp. 288-290).

[181] Abdyssalam Alhaag, A., Savic, G., Milosavljevic, G., Tima Segedinac, M., & Filipovic, M. (2018). Executable platform for managing customizable metadata of educational resources. *The Electronic Library*, *36*(6), 962-978.

[182] Musante, K., & DeWalt, B. R. (2010). Participant observation: A guide for fieldworkers. Rowman Altamira.

[183] Lundstrom, W. J., & Lamont, L. M. (1976). The development of a scale to measure consumer discontent. Journal of Marketing Research, 13(4), 373-381.

[184] Schriesheim, C. A., & Hill, K. D. (1981). Controlling acquiescence response bias by item reversals: The effect on questionnaire validity. Educational and psychological measurement, 41(4), 1101-1114.

[185] Gerasimov, A., Michael, J., Netz, L., Rumpe, B., & Varga, S. (2020). Continuous Transition from Model-Driven Prototype to Full-Size Real-World Enterprise Information Systems. In AMCIS

[186] Sánchez-Villarín, A., Santos-Montaño, A., Koch, N., & Casas, D. L. (2020). Prototypes as Starting Point in MDE: Proof of Concept. In WEBIST (pp. 365-372).

[187] Gamito, I., & da Silva, A. R. (2020, September). From Rigorous Requirements and User Interfaces Specifications into Software Business Applications. In International Conference on the Quality of Information and Communications Technology (pp. 459-473). Springer, Cham.

[188] Seixas, J., Ribeiro, A., & da Silva, A. R. (2019, May). A Model-Driven Approach for Developing Responsive Web Apps. In ENASE (pp. 257-264).

[189] Da Silva, A. R., Saraiva, J., Silva, R., & Martins, C. (2007, March). XIS-UML profile for extreme modeling interactive systems. In *Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'07)* (pp. 55-66). IEEE.

[190] Kaplar, S., (2014), Podsistem za specifikaciju korisničkih prava u okviru Kroki alata, Master rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka

[191] Ivković, Ž., (2013), Implementacija podloge za saradnju Kroki alata za skiciranje poslovnih aplikacija sa alatima za UML modelovanje opšte namene, Master rad, Univerzitet u Novom Sadu, Fakultet tehničkih nauka

[192] Triangulation, D. S. (2014, September). The use of triangulation in qualitative research. In Oncol Nurse Forum (Vol. 41, No. 5, pp. 545-7).

[193] Nielsen, J., & Landauer, T. K. (1993, May). A mathematical model of the finding of usability problems. In Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems (pp. 206-213).

[194] Cortés, H., & Navarro, A. (2017). Enterprise WAE: A Lightweight UML Extension for the Characterization of the Presentation Tier of Enterprise Applications with MDD-Based Mockup Generation. International Journal of Software Engineering and Knowledge Engineering, 27(08), 1291-1331.

[195] Ferraiolo, D., Cugini, J., & Kuhn, D. R. (1995, December). Role-based access control (RBAC): Features and motivations. In Proceedings of 11th annual computer security application conference (pp. 241-48).

Indeks korišćenih skraćenica

AIDE	–	Advanced Interaction Design Environment
AM	–	Agile Modeling
AOP	–	Aspect-Oriented Programming
API	–	Application Programming Interface
ASD	–	Adaptive Software Development
ASL	–	Application Specification Language
CASE	–	Computer-Aided Software Engineering
CD4A	–	Class Diagram For Analysis
CERIF	–	Common European Research Information Format
CI	–	Continous Integration
CRIS	–	Current Research Information System
CRUD	–	Create, Retrieve, Update, Destroy
DSDM	–	Design Systems Development Method
DSL	–	Domain Specific Language
DSML	–	Domain-Specific Modeling Language
EBNF	–	Extended Bacus-Naur Form
EMF	–	Eclipse Modeling Framework
ERP	–	Enterprise Resource Planning
EUC	–	Essential Use-Case
EUIS	–	Enterprise User Interface Specification Language
FDD	–	Feature-Driven Development
FQUAD	–	Framework For Qualitative Assesment Of Domain-Specific Languages
GPL	–	General Purpose Language
GPML	–	General Purpose Modeling Language
GQM	–	Goal-Question Metric
UI	–	User Interface
IFML	–	Interaction Flow Modelling Language
IS	–	Information System
JAD	–	Joint Application Design
JSF	–	Java Server Faces
M2M	–	Model To Model
M2T	–	Model To Text
MDE	–	Model-Driven Engineering
MDF	–	Mission Data File
MDSD	–	Model-Driven Software Development
MDWE	–	Mode-Driven Web Engineering
MEM	–	Methodology Acceptance Model
MOF	–	Meta Object Facility
OCL	–	Object Constraint Language
OMG	–	Object Management Group
OOP	–	Object-Oriented Programming
PIM	–	Platform-Idependent Model
PSM	–	Platform-Specific Model
RAD	–	Rapid Application Development

RBAC	–	Role Based Access Control
RE	–	Requirements Engineering
SDLC	–	Information System Development Life Cycle
SP	–	Software Prototyping
TAM	–	Technology Acceptance Model
UIGU	–	User Interface Generator For UMPLE
UMPLE	–	UML Programming Language
WAR	–	Web Archive
WED	–	Window/Event Diagram
WebML	–	Web Modelling Language
XP	–	Extreme Pogramming
XUL	–	XML User Interface Language

Indeks slika

Index tabela

Indeks listinga

Biografija

Milorad Filipović rođen je 23.02.1988. U Osijeku, republika Hrvatska. Osnovnu školu (OŠ "Markušica") pohađao je u mestu Markušica nakon čega upisuje srednju školu "Tehnička škola Nikole Tesle", u Vukovaru. Po završetku srednjoškolskog obrazovanja, 2006. godine, stiče zvanje tehničara za računarstvo. Iste godine upisuje Fakultet tehničkih nauka u Novom Sadu, odsek za Elektrotehniku i računarstvo, studijski program Računarstvo i automatika. Osnovne studije završava u roku, te 2011. stiče zvanje diplomirani inženjer elektrotehnike i računarstva. Master studije na istom fakultetu završava naredne godine sa prosečnom ocenom 9.6 i odbranom master rada pod naslovom “Adaptivna arhitektura web aplikacije bazirana na aspektima” stiče zvanje master inženjer elektrotehnike i računarstva.

2012. godine zapošljava se na Katedri za informatiku Fakulteta tehničkih nauka u Novom Sadu kao stručni saradnik, a iste godine upisuje doktorske studije na studijskom programu za elektrotehničko i računarsko inženjerstvo. U toku svog angažmana na Katedri za informatiku aktivno učestvuje organizaciji raznih događaja koji za cilj imaju promociju računarskih nauka među decom i mladima, kao što su organizacija letnjih škola za učenike, radionice za decu u okviru Festivala nauke i konferencija za promovisanje slobodnog softvera u nastavi.

Sa istraživačkim radom na polju razvoja softvera upravljanog modelima započinje u toku master studija gde tokom rada na svom master radu, zajedno sa još nekoliko kolega, razvija osnovne elemente softverske platforme koja će kasnije postati okosnica istraživanja predstavljenog u ovoj disertaciji. U ovoj fazi učestvuje i u osnivanju MDE (*Model-Driven Egineering*) tima koji okuplja istraživače zaposlene na Katedri za informatiku oko zajedničke teme brzog razvoja softvera. U okviru samostalnog i grupnog naučnog rada učestvovao je na nekoliko međunarodnih naučnih konferencija i autor je 15 naučnih radova koji su objavljeni u zbornicima konferencija i međunarodnim i domaćim časopisima. Pored ovoga, bio je angažovan u održavanju nastave na Katedri za informatiku na više nastavnih predmeta, između kojih su Poslovna informatika, Web programiranje i Osnove računarstva.

Trenutno živi i radi u Berlinu.

Овај Образац чини саставни део докторске дисертације, односно докторског уметничког пројекта који се брани на Универзитету у Новом Саду. Попуњен Образац укоричити иза текста докторске дисертације, односно докторског уметничког пројекта.

План третмана података

Назив пројекта/истраживања
Унапређење спецификације захтева пословних апликација уз ослонац на извршиве моделе
Назив институције/институција у оквиру којих се спроводи истраживање
а) Факултет техничких наука, Универзитет у Новом Саду
Назив програма у оквиру ког се реализује истраживање
Рачунарство и аутоматика – докторска дисертација
1. Опис података

1.1 Врста студије

Укратко описати тип студије у оквиру које се подаци прикупљају

Серија од десет експлоративних студија случаја са учесницима који су доменски експерти из различитих пословних домена.

1.2 Врсте података
а) квантитативни
б) квалитативни

1.3. Начин прикупљања података
а) анкете, упитници, тестови
б) клиничке процене, медицински записи, електронски здравствени записи
в) генотипови: навести врсту _____
г) административни подаци: навести врсту _____
д) узорци ткива: навести врсту_____
ђ) снимци, фотографије: навести врсту_____
е) текст, навести врсту _____
ж) мапа, навести врсту _____
з) остало: **посматрањем уз учествовање (*participant observation*), мерењем времена**

1.3 Формат података, употребљене скале, количина података

1.3.1 Употребљени софтвер и формат датотеке:
а) **Excel фајл,** датотека _____
b) SPSS фајл, датотека _____

c) **PDF фајл**, датотека _____
d) Текст фајл, датотека _____
e) **JPG фајл**, датотека _____
f) Остало, датотека _____

1.3.2. Број записа (код квантитативних података)

а) број варијабли 3
б) број мерења (испитаника, процена, снимака и сл.) 10

1.3.3. Поновљена мерења
а) да
б) не

Уколико је одговор да, одговорити на следећа питања:
а) временски размак између поновљених мера је _____
б) варијабле које се више пута мере односе се на _____
в) нове верзије фајлова који садрже поновљена мерења су именоване као _____

Напомене: _____

Да ли формати и софтвер омогућавају дељење и дугорочну валидност података?
а) Да
б) Не
Ако је одговор не, образложити _____
_____

2. Прикупљање података

2.1 Методологија за прикупљање/генерисање података

2.1.1. У оквиру ког истраживачког нацрта су подаци прикупљени?
а) експеримент, навести тип **серија од 10 студија случаја**
б) корелационо истраживање, навести тип _____
ц) анализа текста, навести тип _____
д) остало, навести шта _____

2.1.2 Навести врсте мерних инструмената или стандарде података специфичних за одређену научну дисциплину (ако постоје).
_____

_____

2.2 Квалитет података и стандарди

2.2.1. Третман недостајућих података
а) Да ли матрица садржи недостајуће податке? Да **Не**

Ако је одговор да, одговорити на следећа питања:

а) Колики је број недостајућих података? _____

б) Да ли се кориснику матрице препоручује замена недостајућих података? Да Не

в) Ако је одговор да, навести сугестије за третман замене недостајућих података

_____

2.2.2. На који начин је контролисан квалитет података? Описати

Применом триангулације: комбиновањем техника квалитативног и квантитативног истраживања (упитници, посматрање уз учествовање, праћење времена од стране коришћеног софтверског алата).

2.2.3. На који начин је извршена контрола уноса података у матрицу?

3. Третман података и пратећа документација

3.1. Третман и чување података

3.1.1. Подаци ће бити депоновани у Mendeley Data репозиторијум.
3.1.2. URL адреса https://data.mendeley.com/datasets/kbh7hjttzy/1
3.1.3. DOI 10.17632/kbh7hjttzy.1

3.1.4. Да ли ће подаци бити у отвореном приступу?
а) Да
б) Да, али после ембарга који ће трајати до _____
в) Не

Ако је одговор не, навести разлог _____

3.1.5. Подаци неће бити депоновани у репозиторијум, али ће бити чувани. Образложење

_____

_____

3.2 Метаподаци и документација података
3.2.1. Који стандард за метаподатке ће бити примењен? _____

3.2.1. Навести метаподатке на основу којих су подаци депоновани у репозиторијум.

_____

_____

Ако је потребно, навести методе које се користе за преузимање података, аналитичке и процедуралне информације, њихово кодирање, детаљне описе варијабли, записа итд.

_____

_____

_____

_____

3.3 Стратегија и стандарди за чување података
3.3.1. До ког периода ће подаци бити чувани у репозиторијуму? _____
3.3.2. Да ли ће подаци бити депоновани под шифром? Да Не
3.3.3. Да ли ће шифра бити доступна одређеном кругу истраживача? Да Не
3.3.4. Да ли се подаци морају уклонити из отвореног приступа после извесног времена? Да Не
Образложити

_____

_____

4. Безбедност података и заштита поверљивих информација

Овај одељак МОРА бити попуњен ако ваши подаци укључују личне податке који се односе на учеснике у истраживању. За друга истраживања треба такође размотрити заштиту и сигурност података.
4.1 Формални стандарди за сигурност информација/података
Истраживачи који спроводе испитивања с људима морају да се придржавају Закона о заштити података о личности (*https://www.paragraf.rs/propisi/zakon_o_zastiti_podataka_o_licnosti.html*) и одговарајућег институционалног кодекса о академском интегритету.

4.1.2. Да ли је истраживање одобрено од стране етичке комисије? Да Не
Ако је одговор Да, навести датум и назив етичке комисије која је одобрила истраживање

_____

4.1.2. Да ли подаци укључују личне податке учесника у истраживању? Да Не
Ако је одговор да, наведите на који начин сте осигурали поверљивост и сигурност информација везаних за испитанике:
а) Подаци нису у отвореном приступу
б) Подаци су анонимизирани
ц) Остало, навести шта

_____

_____

5. Доступност података

5.1. Подаци ће бити
а) јавно доступни
б) доступни само уском кругу истраживача у одређеној научној области
ц) затворени

Ако су подаци доступни само уском кругу истраживача, навести под којим условима могу да их користе:

_____

_____

Ако су подаци доступни само уском кругу истраживача, навести на који начин могу приступити подацима:

_____

_____

5.4. Навести лиценцу под којом ће прикупљени подаци бити архивирани.

_____

6. Улоге и одговорност

6.1. Навести име и презиме и мејл адресу власника (аутора) података

Милорад Филиповић, mfili@uns.ac.rs

6.2. Навести име и презиме и мејл адресу особе која одржава матрицу с подацима

Милорад Филиповић, mfili@uns.ac.rs

6.3. Навести име и презиме и мејл адресу особе која омогућује приступ подацима другим истраживачима

_____