Univerzitet u Beogradu
Fakultet organizacionih nauka

Završni rad

Tema: Generator korisničkog interfejsa korišćenjem XML i CSS tehnologije

Mentor:
dr.Xxxxx Xxxxx

Student:
Xxxxx Xxxxxxx

Beograd, 20 XX

Apstrakt

Predmet ovog rada je korisnički interfejs, kao sredstvo interakcije čoveka i računara. Cilj rada jeste da ukaže na nedostatke postojećih tehnika, tehnologija i alata za razvoj korisničkog interfejsa, sa posebnim osvrtom na Java platformu, kao i da pruži rešenje u vidu alternativnog postupka koji je lišen uočenih nedostataka.

Rešenje je zasnovano na upotrebi jednostavnog platformski nezavisnog modela korisničkog interfejsa definisanog pomoću XML i CSS sintakse i softverskog okvira koji generiše korisnički interfejs na osnovu tog modela. Dokazano je da je prezentovano rešenje bolje od ranije dostupnih i prikazani su principi na kojima se zasniva njegova uspešnost.

Ključne reči: Grafički korisnički interfejs, XML, CSS, Java, Swing

Abstract

The subject of this paper is user interface, as a mean of human-computer interaction. The main goal is to indicate the weaknesses of existing methods, technologies and tools for user interface development, with special focus on Java platform, and to offer a solution in form of an alternative method, which lacks the identified weaknesses.

The solution is based on use of a simple platform-independent user interface model defined in XML and CSS syntax and a software framework which generates user interface from that model. It was proven that the presented solution is better than previously available ones and the principles which were applied to achieve its success are shown.

Keywords: Graphical user interface, XML, CSS, Java, Swing

Sadržaj

1)Uvod....1
2)Šira tematika predmeta rada....3
2.1)Definicija....3
2.2)Vrste i razvoj korisničkih interfejsa....3
2.2.1)Komandna linija....4
2.2.2)Tekstualni korisnički interfejs....4
2.2.3)Grafički korisnički interfejs....5
2.2.4)Web korisnički interfejs....7
2.3)WIMP paradigma....8
2.4)Aktuelne tehnike i tehnologije....9
3)Pravila dobre prakse....13
3.1)MVC uzor....13
3.2)Razdvajanje pojedinačnih problema....14
4)Kratak opis rada....16
4.1)Konceptualno rešenje....16
4.1.1)Kontekst....16
4.1.2)Problem....16
4.1.3)Rešenje....18
4.2)Softversko rešenje....21
4.2.1)Apstrakcija....22
4.2.2)Implementacija....23
5)Uporedni pregled....30
5.1)Zahtevi zadatka....30
5.2)Izrada primera....31
5.2.1)Pomoću direktnog kodiranja....31
5.2.2)Pomoću NetBeans alata....34
5.2.3)Pomoću okvira....39
5.2.4)Pomoću Simple View pristupa....42
5.3)Analiza rezultata poređenja....45
5.3.1)Prednosti XML sintakse....46
5.3.2)Prednosti CSS sintakse....47
5.3.3)Prednosti višeg nivoa apstrakcije....48
6)Zaključak....49
6.1)Mogućnosti za proširenje....50
Dodatak A: Primer napredne upotrebe....52
Dodatak B: Kompletna Simple View specifikacija....70
Pogovor....76
Literatura....77

Indeks slika

Komandna linija....4
Turbo okruženje za programski jezik Pascal....4
Macintosh operativni sistem, 1984. godina [3]....6
MS Windows 95, 1995. godina [2]....6
KDE4 grafičko okruženje za Linux, 2008 godina....6
Korisnički interfejs web aplikacije Gmail....7
Primer jednostavne forme....19
Uporedni primer – Zahtev zadatka....30
Uporedni primer – Direktno kodiranje....34
Hijerarhija komponenata primera u NetBeans-u....35
Forma za lokalizovanje u NetBeans-u....36
Uporedni primer - korišćenjem NetBeans alata....36
Uporedni primer - korišćenjem okvira....41
Uporedni primer – Simple View....44
Napredni primer - Prvo pokretanje....60
Napredni primer - Unošenje podataka....62
Odzivna poruka....63
Napredni primer - Lokalizovana verzija....64
Napredni primer – Verzija sa promenjenim bojama....66
Napredni primer - Verzija sa web pretragom....68
Napredni primer - Rezultati web pretrage....69

Indeks dijagrama

Dijagram klasa MVC uzora....13
Sistemski prikaz generatora korisničkog interfejsa....20
Konceptualni dijagram klasa softverskog rešenja....21
Osnovni sekvencni dijagram SwingGuiBuilder projekta....24
Dijagram klasa podsistema kreiranja komponenata....25
Dijagram klasa podsistema dodavanja komponenata....26
Sekvencni dijagram algoritma formiranja strukture korisničkog interfejsa....26
Sekvencni dijagram algoritma za lokalizovanje komponenata....27
Dijagram klasa podsistema za primenu stilova....28
Sekvencni dijagram algoritma za primenu stilova....29
Napredni primer - Dijagram klasa domenskog modela....52

Indeks tabela

Neki od poznatijih jezika za označavanje namenjeni za izradu korisničkog interfejsa.[10] [11].....11
Poređenje tehnika za izradu korisničkog interfejsa.....18
Ključne prednosti i odgovarajući nedostaci.....50
Specifikacija - Spisak svih elemenata.....70
Specifikacija - CSS svojstva iz kategorije 'Pozadina'.....71
Specifikacija - CSS svojstva iz kategorije 'Okvir'.....71
Specifikacija - CSS svojstva iz kategorije 'Klasifikacija'.....72
Specifikacija - CSS svojstva iz kategorije 'Dimenzije'.....72
Specifikacija - CSS svojstva iz kategorije 'Font'.....73
Specifikacija - CSS svojstva iz kategorije 'Pozicioniranje'.....74
Specifikacija - CSS svojstva iz kategorije 'Tekst'.....74
Preslikavanje Simple View elemenata u Swing komponente.....75

1) Uvod

Prilikom razvoja softvera redovno rešavamo probleme veoma različite prirode. Izrada jednog tipičnog informacionog sistema podrazumeva projektovanje baze podataka, podsistema za komunikaciju sa bazom, zasebnog dela koji sadrži aplikacionu logiku i napokon projektovanje korisničkog interfejsa.

Za izradu nekih podsistema, dobra rešenja su pronađena i implementirana u vidu raznih alata, programskih biblioteka i okvira. Tako je moguće nabaviti softver za objektno-relaciono preslikavanje (kao što je *Hibernate*), za upravljanje životnim ciklusom domenskih objekata (kao što je *Spring*), kao i za obavljanje mnogih drugih poslova unutar informacionog sistema.

Nažalost, bar kada su Java tehnologije u pitanju, razvoj korisničkog interfejsa je slabo podržan, a mali broj veoma kvalitetnih alata i okvira za pravljenje korisničkog interfejsa najčešće je usko vezan za izvršnu platformu. Primera radi, *Tapestry* okvir je namenjen samo za web korisnički interfejs, a većina alata generiše kôd samo za *Swing* paket.

Posledica toga je apsurdna činjenica da je vreme koje programeri troše na razvoj korisničkog interfejsa nesrazmerno veliko u odnosu na vreme potrošeno na rešavanje daleko bitnijih pitanja tokom razvoja softverskog sistema. Smatramo da dalji razvoj softverskog inženjerstva kao teorijske i praktične oblasti računarstva nije moguć bez adekvatnog rešenja dosadašnjih problema prilikom projektovanja korisničkog interfejsa.

U ovom radu su ti problemi detaljno objašnjeni i prezentovano je naše rešenje u vidu softverskog sistema koji generiše korisnički interfejs korišćenjem XML (eXtensible Markup Language) i CSS (Cascading Style Sheets) tehnologije. Model zasnovan na ovim tehnologijama omogućava upotrebu pogodnije sintakse za projektovanje korisničkog interfejsa, lakše održavanje tokom vremena i nezavisnost korisničkog interfejsa od konkretnih platformi kao što su operativni sistemi, programske biblioteke i okviri.

Softversko rešenje se sastoji od platformski nezavisne specifikacije korisničkog interfejsa zasnovane na XML-u i CSS-u (*Simple View*), i generatora Swing korisničkog interfejsa (*SwingGuiBuilder*).

Rad se sastoji od šest poglavlja i dva dodatka:

U prvom poglavlju je dat uvod u kome su definisani predmet i cilj rada.

U drugom poglavlju je definisan pojam korisničkog interfejsa, prikazan istorijat njegovog razvoja i objašnjena je WIMP paradigma na kojoj se zasniva većina grafičkih korisničkih interfejsa. Nakon toga su nabrojane i objašnjene postojeće tehnike za kreiranje grafičkog korisničkog interfejsa.

U trećem poglavlju su objašnjena pravila dobre prakse koja se primenjuju prilikom projektovanja korisničkog interfejsa.

U četvrtom poglavlju je dat prikaz generatora korisničkog interfejsa, najpre konceptualno kao rešenje za problem u odgovarajućem kontekstu, a potom kao detaljan opis pratećeg softverskog rešenja.

U petom poglavlju je data komparativna analiza tehnika za razvoj korisničkog interfejsa na konkretnom primeru. Analiza je izvedena tako što je korisnički interfejs sličnog izgleda i funkcionalnosti izrađen pomoću sve četiri identifikovane tehnike i dobijeni rezultati su međusobno upoređeni.

U šestom poglavlju dat je sažet pregled celog diplomskog rada, prikazani su zaključci koji su izvedeni prilikom analize nove tehnike razvoja korisničkog interfejsa i nabrojane neke ideje za mogući dalji razvoj ovog projekta.

Dodatak A sadrži primer napredne upotrebe opisanog generatora za izradu korisničkog interfejsa za jedan informacioni sistem.

Dodatak B sadrži kompletnu *Simple View* specifikaciju.

Predmet rada je korisnički interfejs kao sredstvo interakcije čoveka i računara. Posebna pažnja je posvećena aspektima korisničkog interfejsa koji su bitni projektantima i softverskim inženjerima, a koji se odnose na modularnost, fleksibilnost i nezavisnost korisničkog interfejsa od platforme.

Cilj rada jeste da ukaže na nedostatke postojećih tehnika, tehnologija i alata za razvoj korisničkog interfejsa, sa posebnim osvrtom na Java platformu, kao i da pruži rešenje u vidu alternativnog postupka koji je lišen uočenih nedostataka.

2) Šira tematika predmeta rada

U ovom poglavlju opisani su bitni pojmovi u vezi sa predmetom rada:

- Definisan je pojam korisničkog interfejsa.
- Dat je sažet pregled istorije interakcije čoveka i računara posredstvom korisničkog interfejsa.
- Opisana je zajednička koncepcija na kojoj se zasnivaju gotovo sve tehnologije grafičkog korisničkog interfejsa.
- Identifikovane su najčešće tehnike za izradu grafičkog korisničkog interfejsa i nabrojane su neke tehnologije koje podržavaju te tehnike.

2.1) Definicija

Budući da je cilj rada pronaženje boljih načina za razvoj korisničkog interfejsa, najpre je neophodno definisati pojam korisničkog interfejsa.

Korisnički interfejs predstavlja realizaciju ulaza i/ili izlaza softverskog sistema.[1]

Dakle, korisničkim interfejsom možemo da smatramo svaki predviđen način komunikacije između korisnika i softverskog sistema. Budući da ovaj rad prati moderne trendove, najveći naglasak biće na grafičkom korisničkom interfejsu, kao trenutno najrasprostranjenijem načinu za ostvarivanje te komunikacije. Ipak, biće dat i kratak istorijat korisničkih interfejsa koji prethode grafičkom, radi prikazivanja kontinuiteta u razvoju tehnologije i izolovanja postojanih principa od promenjivih detalja trenutnog tehnološkog razvoja, izbora proizvođača ili aktuelnih trendova.

2.2) Vrste i razvoj korisničkih interfejsa

Istorija interakcije čoveka i računara deli se na tri perioda:

- period paketne obrade, tj. *batch computing* (1945-1968.)
- komandne linije (1969-1983.)
- grafičkog korisničkog interfejsa (1984. do danas).[2]

Pre komandne linije za unos podataka u računar su korišćene bušene kartice, a izlazni podaci su najčešće bili štampani na štampaču. Ovaj period karakteriše način interakcije sa računarom koji je veoma različit od trenutno aktuelnog, tako da on nije razmatran u ovom radu.

Radi prikazivanja ideja koje su dovele do nastanka grafičkog korisničkog interfejsa, dat je kratak istorijat počev od konzolnog interfejsa.

2.2.1) Komandna linija

Jedan od najstarijih načina komunikacije između korisnika i softverskog sistema jeste korišćenjem komandne linije. Ova praksa vodi poreklo iz vremena konzolnih operativnih sistema kada su se naredbe zadavale preko konzole, tj. preko komandne linije.

Slika 1: Komandna linija

Primer sa slike 1 pokazuje kako se instalira novi program (u ovom slučaju *Subversion*) u Ubuntu distribuciji Linux operativnog sistema.

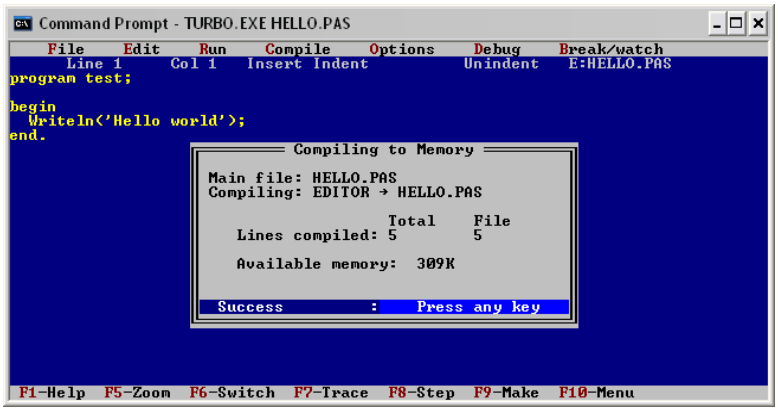
Ovakav korisnički interfejs se koristi tako što korisnik kuca naredbu operativnog sistema ili naziv korišćenog programa praćen parametrima. Korisnik koristi konzolu za pokretanje programa i unos podataka, a sistem je koristi za ispisivanje izlaznih poruka. Uglavnom nije moguća istovremena interakcija korisnika i sistema sa konzolom, što znači da se unos podataka i naredbi od strane korisnika i ispisivanje izlaznih podataka i poruka od strane sistema odigravaju naizmenično.

Svaki novi podatak se ispisuje u novom redu na dnu konzole, a kada se na ekranu popune svi redovi briše se najstarija linija (koja se nalazi u prvom redu) i sve linije se pomeraju za po jedan red naviše.

Ovakav korisnički interfejs je još uvek u velikoj meri zastupljen upravo u softveru za upravljanje operativnim sistemom, dok se u ostalim oblastima primene najčešće smatra zastarelim.

2.2.2) Tekstualni korisnički interfejs

Tekstualni korisnički interfejs (*Textual User Interface – TUI*), kao i komandna linija koristi samo alfanumeričke znakove, međutim način interakcije je potpuno drugačiji. Naredbe su predstavljene tekstualnim oznakama koje su raspoređene širom ekrana, a korisnik ih aktivira pritiskanjem naznačenog tastera na tastaturi ili klikom miša na odgovarajuću oznaku. Ovakav pristup se smatra mnogo produktivnijim, jer korisnik ne mora da zna napamet sve moguće naredbe i njihove parametre, već ih ima predložene na ekranu. Još jedna bitna razlika je ta što se odziv sistema najčešće ne ispisuje dnu ekrana, već se ispisuje unutar posebno označenog prostora za prikazivanje poruke (kao što je pravougaonik sa obaveštenjem o kompajlovanju *Pascal* programa prikazan na slici 2)



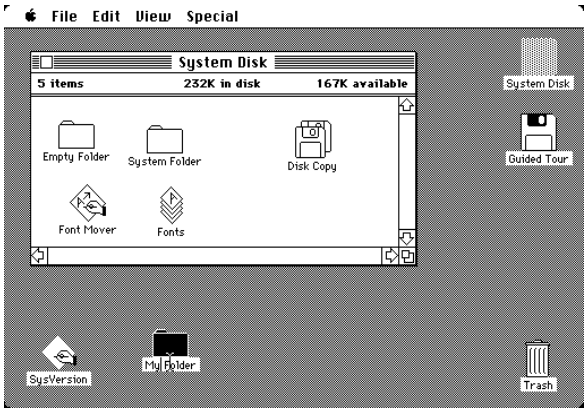
Slika 2: Turbo okruženje za programski jezik Pascal

Prelazak na korišćenje ovakvih korisničkih interfejsa predstavlja ključni trenutak, jer menja način komunikacije sa softverskim sistemom. Jedino što je nedostajalo tekstualnim interfejsima jeste postojanje grafičkih elemenata poput slika i ikonica, a uvođenjem tih inovacija nastali su grafički korisnički interfejsi.

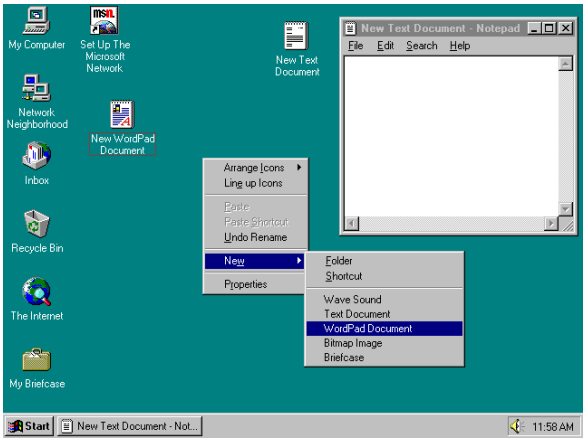
2.2.3) Grafički korisnički interfejs

Grafički korisnički interfejs (*Graphical User Interface – GUI*) predstavlja logičnu evoluciju novog načina interakcije korisnika i sistema uvedenog pojavom tekstualnog interfejsa. Inovacije koje su uvedene u ovoj etapi razvoja uglavnom su samo "kozmetičke" prirode, jer koriste postojeći model interakcije sa ulepšanim komponentama. Prostor na kome se izvršavaju programi više nije konzola, već površina, najčešće nazvana Desktop, na kojoj se nalaze grafički elementi. Interaktivni grafički elementi su prvobitno bili crno-beli crteži u niskoj rezoluciji, da bih ih zamenile komponente u boji bogato ukrašene slikama, koje se nalaze na Desktopu visoke rezolucije.

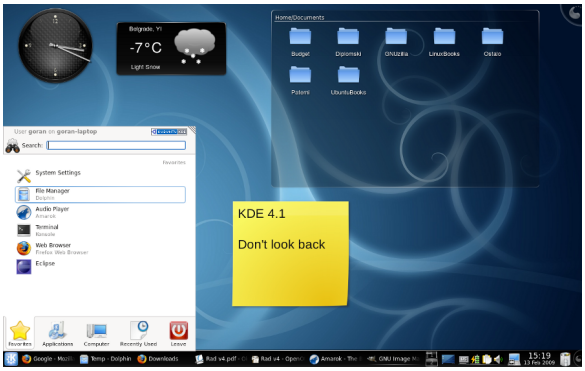
Na slikama 3, 4 i 5 prikazano je nekoliko Desktop okruženja popularnih operativnih sistema na kojima mogu da se vide ikonice, meniji, tasteri i ostali grafički elementi (slike preuzete iz izvora: [3] i [2]).



Slika 3: Macintosh operativni sistem, 1984. godina [3]



Slika 4: MS Windows 95, 1995. godina [2]



Slika 5: KDE4 grafičko okruženje za Linux, 2008 godina

2.2.4) Web korisnički interfejs

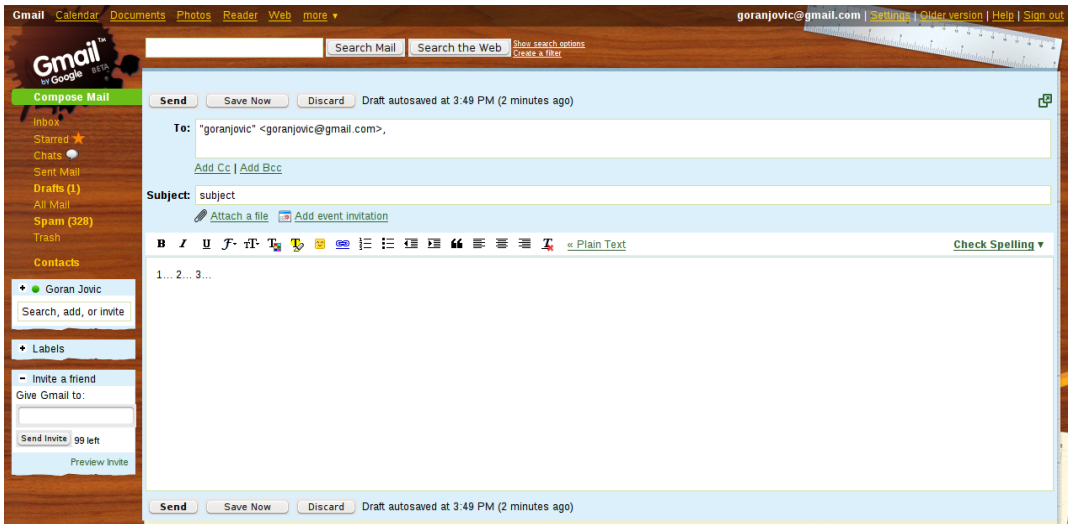
WWW ili *World Wide Web* je bez ikakve sumnje najpoznatiji Internet servis koji je ikada postojao. Isprva je to bio samo skup struktuiranih tekstualnih dokumenata, koji zahvaljujući mehanizmu hiperlinkova postižu nelinearnost teksta[1] i u kombinaciji sa svim ostalim dokumentima prisutnim na Internetu čine jednu ogromnu mrežu struktuiranih podataka (na osnovu čega je i nastao naziv).

Pojavom prvog grafičkog web čitača (*Mosaic*) postaje moguće dokumente predstaviti na dopadljiviji način, ali i usvojiti već prisutne načine interakcije sa korisnikom korišćenjem grafičkih komponenata, kao što je to bio slučaj i prilikom pojave prvih desktop korisničkih interfejsa.

Tokom vremena podrška za multimedije ubrzano raste i web, zahvaljujući velikom izboru mogućih boja, slika i zvukova postaje veoma atraktivan medij, a zahvaljujući osnovnim kontrolama za interakciju za korisnikom, i veoma popularna zamena za klasični grafički korisnički interfejs, usled čega nastaju web aplikacije.

Neke od početnih mana ovakvog pristupa su bile preterana zavisnost od čitača (pogotovo pre uvođenja web standarda) i ugrađena podrška samo za najosnovnije komponente. Sa druge strane, prednosti su bile: mogućnost projektovanja veoma atraktivnog korisničkog interfejsa, nezavisnost od operativnog sistema i hardvera sa klijentske strane i dostupnost velikom broju korisnika.

Većina ovih nedostataka je do danas gotovo potpuno eliminisana, a pojavom novih tehnologija kao što su *JavaScript*, *AJAX* (Asynchronous Javascript And XML), *XHTML* (eXtensible HyperText Markup Language), *DOM* (Document Object Model) grafički korisnički interfejs zasnovan na WWW-u postaje jednako moćan alat kao što je i njegov Desktop prethodnik.



Slika 6: Korisnički interfejs web aplikacije Gmail

Moderne web aplikacije postaju sve popularnije. U njima je moguće koristiti gotovo sve komponente kao i u desktop ekvivalentima, a dostupne su sa bilo kog računara koji ima pristup

1 Običan tekst ima linearnu strukturu, što znači da se čita redom - od početka do kraja. Hipertekst na određenim mestima sadrži naročito označene reči (hiperlinkove), koje ukazuju na neki drugi hipertekst. Akcijom nad hiperlinkom, čitalac prelazi na čitanje tog drugog hiperteksta, koji može da se nalazi na proizvoljnoj poziciji u odnosu na prvi hipertekst, zbog čega kažemo da hiperlinkovi stvaraju efekat koji čini strukturu hiperteksta nelinearnom.

Internetu i po pravilu su mnogo atraktivnije.

2.3) WIMP paradigma

U prethodnom hronološkom pregledu imali smo priliku da vidimo da su komponente vremenom vizuelno poboljšavane, ali da su već više od dvadeset godina bazirane na identičnoj koncepciji.[4]

Ta koncepcija se zove **WIMP,** što je akronim od reči ***W****indow*, ***I****con*, ***M****enu* i ***P****ointer*, [5] koje označavaju osnovne koncepte preko kojih korisnik vrši interakciju sa sistemom. Tokom vremena dodate su mnoge konkretne komponente za različite platforme poput tastera, radio dugmića, kontekst menija, *toolbar*-ova, *combo box*-ova, dijaloga i mnogih drugih. Do današnjih dana, skup osnovnih komponenata koje podržava većina platformi je konvergirao u jedan zajednički spisak koji se u velikoj meri može smatrati konačnim.

Upravo zahvaljujući zajedničkoj koncepciji na kojoj su zasnovani i sada već istom skupu podržanih komponenata, može se reći da su grafički interfejsi na svim modernim platformama konceptualno istovetni. Najveću korist od te istovetnosti imaju sami korisnici - zbog toga što grafički korisnički interfejs izgleda slično na svim izvršnim platformama moguće je brže i lakše naučiti kako se softver koristi. Sa druge strane, programerima posao nije olakšan, jer tehnologije pomoću kojih se realizuje WIMP korisnički interfejs se često, uprkos površnoj sličnosti, značajno razlikuju. Razlike su najuočljivije prilikom poređenja programskog koda desktop korisničkog interfejsa u bilo kom objektno-orijentisanom programskom jeziku sa HTML (HyperText Markup Language) dokumentom koji opisuje web korisnički interfejs.

Takva situacija sugeriše da postoji potreba za postojanjem modela kojim bi bilo moguće na apstraktnom nivou opisati grafički korisnički interfejs, bez ulaženja u detalje implementacije koji se tiču konkretne platforme. Ukoliko bi postojao formalni model zasnovan na WIMP paradigmi, tim modelom bi bilo moguće opisati korisnički interfejs za svaku dostupnu modernu platformu koja je zasnovana na istoj paradigmi. Osim toga, formalni model bi mogao da posluži kao specifikacija na osnovu koje je moguće automatsko generisanje korisničkog interfejsa u proizvoljnoj konkretnoj tehnologiji.

Uzevši u obzir da je u pitanju prilično stara koncepcija (pogotovo prema kriterijumima inače veoma brzog razvoja računarstva), opravdano se postavlja pitanje koliko će još dugo WIMP paradigma da bude aktuelna. Naime, postoje neke oblasti primene za koje interakcija sa računarom preko WIMP interfejsa nije dovoljno dobra i za koje je neophodno tražiti drugačija rešenje. Neke od njih su 3D modelovanje, softver za obuku vozača i računarske simulacije komplikovanih hirurških zahvata. Činjenica je da ova paradigma nije univerzalno primenjiva na korisnički interfejs za svaki tip softvera, međutim primećujemo i da su navedene oblasti veoma specifične. Dok je za softverske sisteme koji rešavaju probleme iz tih oblasti neophodno pronalaziti i koristiti drugačije načine interakcije sa korisnikom, u većini preostalih slučajeva korisnički interfejs zasnovan na WIMP paradigmi ipak predstavlja sasvim adekvatno rešenje.

2.4) Aktuelne tehnike i tehnologije

U prethodnom poglavlju razmotrili smo evolutivni razvoj korisničkog interfejsa pretežno sa stanovišta usluge koju pruža korisniku. U ovom poglavlju ograničićemo se samo na trenutno aktuelne platfome za grafički korisnički interfejs i to sa stanovišta načina na koji ga programer razvija.

Istraživanja iz devedesetih godina pokazuju da je kreiranje korisničkog interfejsa tada obuhvatalo oko 50% vremena razvoja celog softverskog projekta i približno 48% ukupnog programskog koda. [6] Novije istraživanje pokazuje da su vreme potrebno za implementaciju korisničkog interfejsa, kao i količina koda koju programer mora da napiše u međuvremenu smanjeni korišćenjem različitih softverskih alata, ali da problem nije u potpunosti uklonjen. [7]

Do sada je nastao velik broj tehnologija za izradu grafičkog korisničkog interfejsa koje podržavaju različite tehnike izrade.

Identifikovane tehnike su:

- Direktno kodiranje
- Korišćenje alata za crtanje
- Korišćenje okvira (*framework*)
- Kodiranje pomoću jezika za označavanje (*markup language*)

Direktno kodiranje je najstarija tehnika za izradu korisničkog interfejsa. Ova tehnika je ujedno i najmoćnija zbog toga što je njom moguće postići sve funkcionalnosti koje data tehnologija pruža. Nažalost, njena najveća mana je što po pravilu podrazumeva pisanje velike količine koda i za postizanje najelementarnijih efekata.

Tehnika je često primenjivana kod biblioteka za korisnički interfejs za starije programske jezike, a doskora je predstavljala jedini način za kreiranje korisničkog interfejsa pomoću Javinog standardnog Swing paketa.

Korišćenje alata za crtanje je tehnika koja ne rešava probleme direktnog kodiranja, ali ih u određenoj meri zaobilazi. Naime, velika količina koda i dalje postoji, ali o tome više ne vodi računa programer, već specijalizovani softverski alat. Programer pomoću alata crta korisnički interfejs koji želi da napravi, a alat na osnovu toga generiše kôd. Nedostaci su što velika količina koda i dalje postoji, a funkcionalnosti korisničkog interfejsa su ograničene na one koje alat podržava, dok je za sve dodatno potrebno ručno menjati generisani kôd.

Među trenutno popularnim tehnologijama, Majkrosoftova .NET platforma ubedljivo prednjači u podržanosti alatima. *Visual Studio*, zvanično razvojno okruženje za .NET aplikacije, sadrži odličan editor za grafički korisnički interfejs, uključujući i desktop i web aplikacije. Neki od drugih popularnih alata su *NetBeans* i *Eclipse Visual Editor plugin* (Java), *Delfi* (Object Pascal) i *Macromedia Dreamweaver* (HTML).

Korišćenje okvira (*framework*) predstavlja rešenje većine problema prethodna dva pristupa koje je postignuto korišćenjem veoma generalizovanog koda za grafički korisnički interfejs i njegovim prilagođavanjem u skladu sa željenim parametrima konkretnog problema. Cilj je postizanje istog broja mogućnosti kao i prilikom direktnog kodiranja, ali sa manje koda. Generalizovan kôd se nalazi unutar samog okvira i izvedeni korisnički interfejsi se dinamički povezuju sa njim umesto da se generiše novi kôd, kao što alati rade. Zbog toga je kod ovog pristupa povećana uređenost i iskorišćenost koda, ali korišćeni okvir mora da uđe u sastav softverskog proizvoda, što može da ima pravne posledice vezane za licence.

Ovaj pristup je naročito popularan među web aplikacijama za koje je razvijen velik broj okvira. Neki od poznatijih su *Struts*, *Tapestry*, *Java Server Faces* (Java), *ASP.NET* (.NET) i drugi.

Posebni kategoriju čine CMS okviri (*Content Management System* – sistemi za upravljanje sadržajima) koji služe za izradu celokupnih web aplikacija (uključujući korisnički interfejs) za neku često veoma usku namenu poput web portala ili elektronske trgovine. Poznatiji primeri su *Drupal* i *Joomla!* (PHP).

Kodiranje pomoću jezika za označavanje (*markup language*) je najnoviji postupak koji predstavlja unapređenje tehnike zasnovane na okvirima. Korisnički interfejs se definiše u određenom jeziku za označavanje najčešće (ali ne obavezno) zasnovanom na XML-u. Potom se te definicije ubacuju u prateći okvir koji ih izvršava kao interpreter ili ih transformiše u izvorni kôd konkretne tehnologije (na sličan način kao što kompajleri prevode izvorni kôd u mašinski). Ovaj pristup je uzeo maha u svetu i desktop i web programiranja, ali još uvek nije široko prihvaćen.

Bitne prednosti ovog pristupa su platformska i tehnološka nezavisnost projektovanog korisničkog interfejsa, mogućnost ponovne upotrebe grafičkih komponenata između različitih implementacionih tehnologija [8], kao i jedinstven način izrade interfejsa za različite vrste uređaja od kojih neki mogu da budu namenski računari poput mobilnih telefona najnovije generacije. [9]

Dodatna prednost uvođenja apstraktnog modela korisničkog interfejsa jeste mogućnost lakšeg praktikovanja razvoja vođenog slučajevima korišćenja (*Use Case Driven Development*), gde svakoj realizaciji slučaja korišćenja odgovara određeni korisnički interfejs. Budući da metodologije zasnovane na slučajevima korišćenja u prvi plan stavljaju korisnikovu interakciju sa softverskim sistemom, korisno je rano razmatranje rešenja korisničkog interfejsa na apstraknom nivou, pomoću nekog dovoljno nezavisnog modela. Ukoliko bi model korisničkog interfejsa bio potpuno apstraktan i sveobuhvatan bilo bi moguće doći do njega preslikavanjem iz modela slučaja korišćenja bez vezivanja za konkretnu tehnologiju.

Naziv	Proizvođač	Godina	Okruženje	Platforma	Licenca
GladeXML	GNOME	1998	Desktop	GTK+	LGPL
GNUStep Renaissance	GNUStep	2001	Desktop	GNUStep	LGPL
Gul2/Xul	redsofa	2005	Web/Desktop	PHP-GTK 2	GPL
LZX	Laszlo Systems	2003	Web	Flash Player 5	CPL
MXML	Adobe	2004	Web	Flash Player 9	MPL
QuiX	inno:script	2005	Web	browser	Komercijalna
UIML	OASIS	1997	Web/Desktop	/	/
XAML	Microsoft	2005	Web/Desktop	.NET ili Silverlight	Komercijalna
XForms	W3C	2006	Web	browser plugin	W3C
XUL	Mozilla Foundation	1998	Web/Desktop	Gecko	GPL/LGPL/MPL
ZUML	Potix	2005	Web	ZK Ajax Framework	GPL

Tabela 1: Neki od poznatijih jezika za označavanje namenjeni za izradu korisničkog interfejsa.[10] [11]

U tabeli 1 možemo da vidimo da postoji velik broj softverskih proizvoda namenjenih generisanju korisničkog interfejsa na osnovu jezika za označavanje. Neki od zaključaka koje možemo da izvedemo jesu da:

- Uprkos konceptualnoj sličnosti desktop korisničkog interfejsa i web ekvivalenta, mali broj nabrojanih jezika podržava oba tipa.
- Štaviše, neki od njih su tesno vezani za specifičan operativni sistem ili čak određeni softverski paket. Na primer:
  - Desktop korisnički interfejs definisan pomoću XAML specifikacije zahteva .NET okruženje a samim tim i *Windows* operativni sistem
  - *GladeXML* je razvijan za pravljenje korisničkog interfejsa pomoću GTK biblioteka Gnome desktop okruženja Linux operativnog sistema.
  - *Gul2/Xul* pored GTK-a podržava rad i sa PHP bibliotekama
  - *LZX* i *MXML* zavise ne samo od *Flash* izvršnog okruženja, već i od njegove verzije.

UIML (User Interface Markup Language) je opisan u jednom od prvih objavljenih radova na ovu temu. Ideja o tehnološki nezavisnom korisničkom interfejsu generisanom iz XML-a je opisana u izvorima [12] i [13]. Nažalost, prevelika složenost predložene specifikacije opisane u izvoru [14] dovela je do veoma slabo izražene upotrebe. Ipak, ideja prezentovana u ovim radovima podstakla je većinu kasnijih radova i projekata.

UIML predviđa da se korisnički interfejs definiše pomoću podkomponenata koje određuju strukturu, stil, sadržaj, i ponašanje. Za definisanje svih podkomponenata koristi se XML sintaksa.

XForms i **XUL** (XML User Interface Language) su specifikacije koje su objavili W3C (*World Wide Web Consortium*) i Mozilla fondacija, respektivno. Obe specifikacije su prvobitno namenjene za definisanje funkcionalno i vizuelno bogatog web korisničkog interfesa. Komparativna analiza pokazuje da ne postoje značajne razlike u upotrebnoj vrednosti ova dva modela. [15]

Razlika između njih nastaje tek sa pojavom implementacija XUL korisničkog interfejsa za desktop okruženje (kao što je Luxor implementacija za Java Swing), što daje veliku prednost XUL specifikaciji.

Majkrosoftov odgovor na prethodne specifikacije je **XAML** (Extensible Application Markup Language), koji je za razliku od XUL-a, od početka projektovan kao jezik za definisanje i web i desktop korisničkog interfejsa. Web varijanta se izvršava unutar *Silverlight* dodatka za Internet Explorer ili *Moonlight* dodatka Mozilla Firefox, dok se desktop varijanta izvršava unutar *Windows Presentation Foundation* komponente .NET platforme.

U poređenju sa prethodnicima XAML donosi odličnu inovaciju u vidu kvalitetno podržanog projektovanja korisničkog interfejsa i za web i za desktop okruženje *Windows* operativnog sistema. Sa druge strane, kao ključnu manu treba navesti da je malo verovatno da će biti podržana desktop okruženja ostalih operativnih sistema.

Postupak koji je prezentovan u ovom radu i koji čini njegovu centralnu komponentu spada u kategoriju izrade korisničkog interfejsa *markup* sintaksom i kao što naslov teme sugeriše u pitanju je sintaksa zasnovana na XML-u.

Ono što bitno razlikuje novi pristup od drugih iz iste kategorije jeste:

- Strogo razdvajanje **nezavisne specifikacije** jezika i platformski **zavisne implementacije**[2], kako bi se omogućio kasniji razvoj drugih generatora za različite platforme.
- Strogo poštovanje **pravila dobre prakse** razvoja softvera kao što su primena softverskih uzora (*software patterns*) i razdvajanje pojedinačnih problema (*Separation of Concerns*).

2 Priložena implementacija je okvir koji u vreme izvršenja programa interpretira nezavisni model i na osnovu njega generiše forme pomoću Javinog Swing paketa.

3) Pravila dobre prakse

Tokom višegodišnjeg razvoja računarstva i softverskog inženjerstva kao njegove discipline, kao i same upotrebe stečenih znanja u praksi, otkriven je i dokumentovan čitav niz pravila dobre prakse. Ova pravila predstavljaju dovoljno opšta rešenja za neku klasu problema koja se mogu više puta primeniti i formulisana su u vidu softverskih uzora.

Definicija koju je dao Aleksander glasi:
Uzor je trodelno pravilo, koje uspostavlja relaciju između nekog problema, njegovog rešenja i njihovog konteksta.[16]

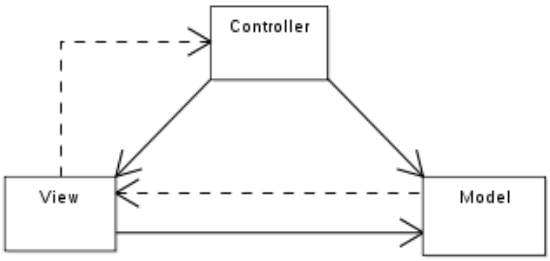
Često je korišćena i jednostavnija definicija:
Uzor je rešenje za problem u nekom kontekstu.[17]

Dokumentovanje oprobano dobrih postupaka i tehnika za rešavanje čestih problema veoma je bitno za svaki dalji razvoj bilo koje inženjerske discipline, pogotovo ukoliko je kod nje zabeležen tako vrtoglav razvoj, kakav je pratio istoriju softverskog inženjerstva.

Dve dobre prakse su od posebnog značaja za ovaj rad jer se bliže bave tematikom strukture korisničkog interfejsa i njegovim mestom u arhitekturi softvera.

3.1) MVC uzor

Model-View-Controler (MVC) je jedan od uzora arhitekture, koji se oslanja na princip tronivojske arhitekture aplikacije. Tronivojska arhitektura podrazumeva razdvajanje aplikacije na prezentacioni sloj, sloj poslovne logike i sloj podataka. MVC ide korak dalje razdvojivši prezentacioni sloj na *View* i *Controller*.[3] [19]



Dijagram 1: Dijagram klasa MVC uzora

Model obuhvata sloj poslovne logike koji obmotava sloj podataka i odgovoran je za poslovno područje aplikacije, njenu strukturu i operacije. [20]

View predstavlja sam način prikaza rezultata obrade i odgovoran je za vizuelnu reprezentaciju modela.

3 Dijagram preuzet sa [18]

Controller je koncept koji povezuje prethodna dva i odgovoran je za kontrolu toka i stanja korisničkog ulaza. Dodatni zadatak, koji predstavlja neophodan uslov osnovnog jeste vršenje prevođenja (konverzije, kastovanja) ulaznih tipova podataka u one koje sistem predstavljen modelom očekuje i obrnuto.

Budući da MVC uzor predstavlja nezvanični standard za projektovanje softverskih sistema, postupak realizovan u ovom radu je projektovan tako da korisnički interfejs dobijen njegovom upotrebom bude u skladu sa MVC arhitekturom.

3.2) Razdvajanje pojedinačnih problema

MVC uzor je specijalni slučaj opštijeg principa poznatog pod nazivom *Separation of Concerns*. Prema ovom principu softver je radi lakšeg projektovanja i kasnijeg održavanja potrebno podeliti na manje celine. Podela se obavlja prema problemima koje ceo softverski sistem treba da reši, tako da se rešavanje svakog pojedinačnog problema izdvoji u zasebnu celinu.[21] Osim MVC uzora, još jedan veoma poznat specijalni slučaj ovog principa je i sama tronivojska arhitektura softvera, prema kojoj se softverski sistem deli na komponente koje zasebno rešavaju probleme čuvanja podataka, aplikacione logike i interakcije sa korisnikom.

Kada su pogodnosti upotrebe tronivojske arhitekture i MVC uzora postale očigledne, usledio je logičan nastavak primene ovog principa u vidu daljeg raslojavanja dobijenih komponenata. U skladu sa predmetom proučavanja ovog rada posmatraćemo dalje raslojavanje *View* komponente. Zanimljivo je, kao što ćemo videti u ovom poglavlju, da su za web i desktop izvršno okruženje primenjena razdvajanja po različitim kriterijumima.

Ranije je za pisanje web stranica korišćen stari HTML standard koji je obuhvatao definisanje i strukture dokumenta[4] i njegove prezentacije[5]. Struktura stranice određuje skup elemenata koji se nalaze na njoj, njihovu logičku organizaciju i podatke koji se nalaze u njima, dok prezentacija (ili stil) stranice određuje samo način na koji će ti elementi da budu prikazani u čitaču i njihove vizuelne osobine poput boje, veličine ili fonta. Rešavanje oba zadatka u istom dokumentu dovodilo je do ozbiljnih problema prilikom održavanja, jer su bogato ukrašene stranice postajale prilično složene, a za promene različite prirode trebalo je menjati isti kôd.

Rešenje koje je preporučio W3C bilo je zamena zastarelog HTML-a novim standardima koji bi dva različita problema rešili na različite načine. Tako su nastali XHTML, kao prepravljena i poboljšana varijanta HTML-a, i CSS – novi jezik za definisanje izgleda web stranica. Primenom novih standarda *View* komponenta je efektivno podeljena na deo zadužen za strukturu i deo za prezentaciju.

XHTML za razliku od svog prethodnika predstavlja varijantu XML jezika posebno namenjenu za web stranice, što između ostalog znači da poštuje stroga pravila dobre oformljenosti koja moraju da važe za sve XML dokumente.

4 tagovima <head>, <body>, <h1>...
5 tagovima <font>, <b>, <i>...

Dokument je dobro oformljen ako:

- postoji deklaracija XML dokumenta
- postoji samo jedan koreni element unutar koga se nalaze ostali elementi
- poštuje pravila XML sintakse
  - za svaki otvoren tag postoji odgovarajući zatvoren tag
  - poslednji otvoreni tag mora prvi da bude zatvoren
  - vrednosti svih atributa taga moraju da budu okružene znacima navoda [22]

Dobro oformljeni dokumenti imaju precizno definisanu hijerarhijsku strukturu, što olakšava izradu web čitača ali i bilo kog drugog softvera koji služi za njihovu obradu.

CSS je specijalizovani jezik za jednostavan opis prezentacije bilo kog XML dokumenta. Upotreba ovog jezika donosi neke nove prednosti:

- Izdvajanjem opisa vizuelnih osobina web stranice u CSS dokument, u osnovnom dokumentu ostaje samo struktura, što ga čini mnogo jednostavnijim.
- Moguće je opisati izgled većeg broja stranica (npr. celog sajta) na samo jednom mestu.
- Specijalizovani jezik pruža više različitih mogućnosti za definisanje izgleda stranice.

Za to vreme su na polju desktop aplikacija struktura i prezentacija i dalje čvrsto spojeni u istoj programskoj celini. Međutim, i na desktop platformi se pojavljuje novo razdvajanje aspekata prilikom primene eksternih tekstualnih datoteka sa lokalizovanim tekstovima radi lakšeg prevođenja aplikacije na drugi jezik. U Java aplikacijama je za lokalizovanje najčešće korišćen format parova ključ-vrednost koji se nalaze u *properties* datotekama.

Veoma brzo se upotreba lokalizacionih datoteka našla i u nekim web aplikacijama i to koristeći gotovo identičan postupak kao kod desktop programa, što znači da je pokazano dobra praksa uspešno prenesena sa jedne platforme na drugu. Usled toga, razumno je pretpostaviti da je moguće prenošenje i u suprotnom smeru, odnosno da bi trebalo uvesti raslojavanje korisničkog interfejsa na strukturu i prezentaciju i u desktop svetu.

Projekat kod koga je dugoročni cilj objedinjavanje razvoja korisničkog interfejsa za web i za desktop okruženje, u skladu sa navedenim principom (*Separation of Concerns*), morao bi primenjuje razdvajanje po oba kriterijuma, dakle da podržava razdvajanje korisničkog interfejsa na zasebne delove za definisanje **strukture**, **prezentacije** i **lokalizacije**.

4) Kratak opis rada

Kao programski deo ovog rada priložen je okvir namenjen za generisanje Java Swing korisničkog interfejsa na osnovu platformski nezavisnog modela definisanog pomoću XML i CSS sintakse. U ovom poglavlju je dat opis priloženog rada.

Na konceptualnom nivou je razmotrena potreba za postojanjem generatora korisničkog interfejsa, identifikovani su ključni problemi postojećih tehnika i na jednostavnom primeru je pokazano kakvo rešenje treba da se primeni da bi problemi bili prevaziđeni.

Softversko rešenje je realizovano na osnovu zaključaka izvedenih u konceptualnom rešenju. Zbog potrebe za nezavisnošću od platforme, rešenje je projektovano tako da su apstraktna specifikacija i konkretna implementacija strogo razdvojene.

4.1) Konceptualno rešenje

U ovom delu rada je predstavljeno konceptualno rešenje problema formulisano u vidu softverskog uzora – kao rešenje problema projektovanja grafičkog korisničkog interfejsa prilikom razvoja Java aplikacija.

4.1.1) Kontekst

Već duže vreme je aktuelno pitanje kreiranja korisničkog interfejsa za programe pisane u Javi. Naime, postojeća rešenja su značajno slabija od svojih ekvivalenata za druge popularne programske jezike, što je posebno primetno kod alata za Swing. Situacija je znatno bolja na polju web aplikacija, što je doprinelo reputaciji Jave kao "serverskog jezika", tradicionalno jakog u web svetu, ali manje primenjivog za desktop aplikacije.

4.1.2) Problem

Problem projektovanja grafičkog korisničkog interfejsa ogleda se u ulaganju nesrazmerno puno vremena i truda za postizanje elementarnih efekata.

Osnovni problem može da se razloži na više podproblema:

- Programski kôd zadužen za korisnički interfejs je uglavnom preobiman:[6]
  - Više pojedinačnih problema rešava se u istoj celini (kôd za strukturu forme, njen izgled, a često i ponašanje nalazi se u jednoj klasi).
  - Korisnički zahtevi postaju sve složeniji, što povećava obim koda.

6 Što je naročito izraženo upravo kod Swing tehnologije

- Velik broj različitih tehnologija:
  - Ukoliko isti softverski sistem treba da ima korisnički interfejs za više platformi, neophodno ga je više puta implementirati (za svaku platformu po jednom).
  - Programer mora ponovo da uči svaku novonastalu tehnologiju.
- Primena odgovarajuće tehnike može da reši jedan problem, ali uvede drugi:
  - Direktnim kodiranjem mogu da se iskoriste sve mogućnosti tehnologije, po cenu ogromne količine koda.
  - Korišćenjem alata za crtanje se olakšava prvobitno kreiranje formi, ali se otežava njihovo kasnije održavanje.
  - Korišćenjem alata za crtanje je teško napraviti korisnički interfejs koji se dinamički menja u vreme izvršenja programa.
  - Korišćenjem okvira je moguće napraviti dinamički promenjiv korisnički interfejs, ali okviri su najčešće vezani za određenu platformu.
  - Korišćenje jezika za označavanje može da uvede i nezavisnost od konkretne platforme, međutim dobijeni *markup* dokument može da bude jednako obiman kao i programski kôd kojim bi bila implementirana ista forma korišćenjem direktnog kodiranja.

Iz navedenih problema, možemo da identifikujemo nekoliko ključnih osobina od kojih zavisi upotrebna vrednost tehnologije i tehnike za izradu grafičkog korisničkog interfejsa:

- Jednostavnost – mogućnost lakog kreiranja forme, pomoću relativno malo programskog koda.
- Dinamika – mogućnost kreirane forme da se menja u toku rada programa.
- Nezavisnost od platforme – mogućnost upotrebe iste forme na različitim platformama.[7]
- Prilagodljivost – mogućnost lake izmene forme, prilikom održavanja.
- Lokalizacija – mogućnost lakog prevođenja forme na određeni jezik (specijalni slučaj prilagodljivosti).

U tabeli 2 prikazan je nivo u kome postojeće tehnike izrade korisničkog interfejsa poseduju nabrojane osobine.

7 Pod platformom se podrazumeva kombinacija hardvera i softvera koja omogućava rad softverskih aplikacija. Softverska platforma je operativni sistem (npr. Windows, Linux) ili programsko okruženje (npr. Java, .NET). [23]

\	**Jednostavnost**	**Dinamika**	**Nezavisnost**	**Prilagodljivost**	**Lokalizacija**
Sam kôd	-	+	-	?	?
Alat	+	-	-	?	+
Okvir	+	+	-	+	+
???	+	+	+	+	+

Legenda:

+	Tehnika poseduje datu osobinu.
-	Postizanje osobine u toj tehnici nemoguće ili izrazito teško.
?	Tehnika ne poseduje datu osobinu, ali ne onemogućava njeno postizanje.

Tabela 2: Poređenje tehnika za izradu korisničkog interfejsa

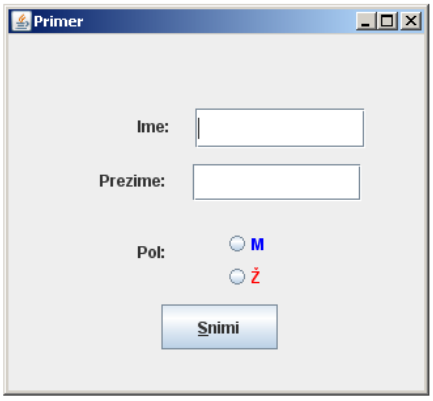
Ukoliko pretpostavimo da bi tehnika koja poseduje sve opisane osobine mogla da ukloni navedene poteškoće, problem se svodi na nepostojanje takve tehnike koja omogućava jednostavno projektovanje i održavanje dinamički izmenjivog, platformski nezavisnog, potpuno prilagodljivog (uključujući lokalizaciju) korisničkog interfejsa.

4.1.3) Rešenje

Tehnika koja bi rešila sve opisane probleme mora da bude u skladu sa MVC uzorom, što znači da mora da podržava, a poželjno i forsira, razdvajanje prezentacionog sloja na deo koji je zadužen za sam izgled stranice (*View*) i deo koji definiše ponašanje, tj. povezivanje sa aplikacionom logikom (*Controller*). Potom, u skladu sa pravilima dobre prakse, *View* sloj bi morao da bude strogo razdvojen na deo zadužen za definisanje strukture korisničkog interfejsa, deo za definisanje njegovog izgleda i zaseban deo sa lokalizovanim tekstovima za specifičan jezik.

Svaki deo treba da bude formulisan korišćenjem one sintakse koja najviše odgovara prirodi zadatka koji rešava. Tako bi struktura bila definisana u XML-u, prezentacija u CSS-u, a lokalizacija u bilo kom formatu koji ima parove ključ-vrednost, kao što su npr. *properties* datoteke, korišćene u mnogim Java aplikacijama.

Sledi primer kako bi mogao da izgleda formalni opis jedne jednostavne forme.



Slika 7: Primer jednostavne forme

Da bismo napravili formu kao na slici 7, treba prvo da definišemo njenu strukturu. U strukturi određujemo koji se elementi nalaze u kojim kontejnerima na formi. U našem primeru imamo tri labele, dva tekstualna polja, dva radio dugmeta i jedno obično dugme. Dakle XML koji opisuje strukturu izgledao bi ovako:

```
<form id="form">
	<label id="firstName"/>
	<textfield id="firstNameField"/>
	<label id="lastName"/>
	<textfield id="lastNameField"/>
	<label id="gender"/>
	<group id="genderRadios">
		<radio id="male"/>
		<radio id="female"/>
	</group>
	<button id="save"/>
</form>
```

Dalje, CSS model, kojim je opisan izgled forme, bi sadržao skup pravila koji bi bio primenjen nad prethodno definisanim elementima. Pravila bi bila primenjena nad pojedinačnim elementima, pomoću jedinstvenog identifikatora *id*, ili nad svim elementima odgovarajućeg tipa.

```
label{ text-align: right }

#male{ color:blue }

#female{ color:red } [8]
```

Napokon, lokalizovanje se svodi na formiranje parova identifikatora elemenata i njihovih lokalizovanih naziva, kao u sledećem primeru:

8 Napomena: u primeru je dat nekompletan CSS opis koji ne obuhvata dimenzije i pozicije elemenata. Ovaj primer služi samo za ilustrovanje ideje, a potpuni CSS opis će biti dat kasnije okviru detaljne analize ovog postupka, prilikom predstavljanja krajnjeg rešenja.

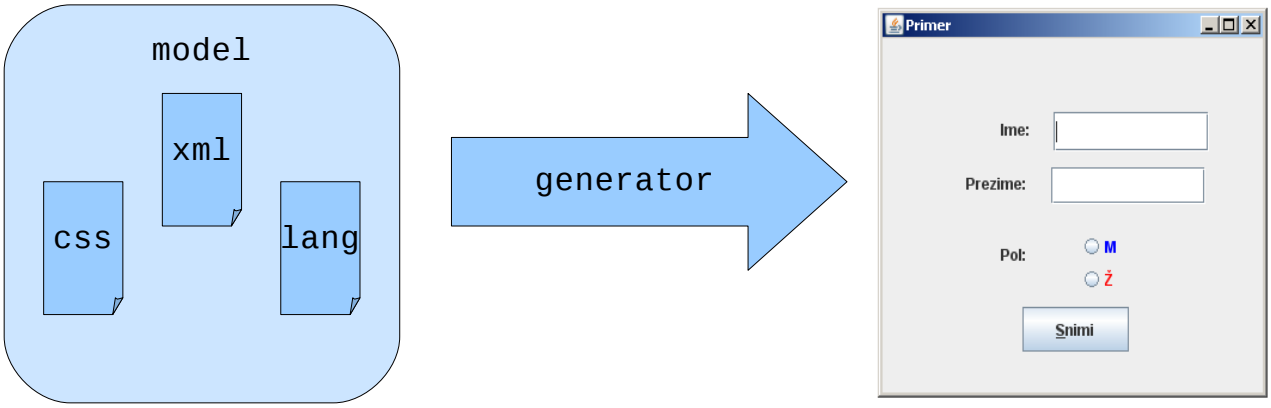
```
form=Proba
firstName=Ime
lastName=Prezime
gender=Pol
male=M
female=Z
save=Snimi
```

Korisnički interfejs definisan ovakvim modelom ispunjava sve uslove koji su razmatrani ranije:

- ✔ Lako se održava, jer su ključni problemi potpuno razdvojeni u pojedinačne jednostavne celine.
- ✔ Omogućava dinamičko menjanje korisničkog interfejsa, budući da sami opisi mogu da se menjaju u toku rada programa.
- ✔ Potpuno je nezavisan od prezentacione tehnologije ili čak Java platforme.
- ✔ U skladu je sa pravilima dobre prakse korišćenja MVC uzora i razdvajanja strukture i prezentacije.
- ✔ U osnovi podržava lokalizaciju.

Uvođenjem nezavisnog modela u prezentacioni sloj stvorene su ogromne mogućnosti, ali da bi on radio neophodno ga je prethodno transformisati u platformski zavisan oblik, pomoću softvera koji kao ulaz koristi model i na osnovu njega generiše odgovarajući korisnički interfejs. U zavisnosti od toga koji generator je odabran, forme bivaju generisane u Swing-u, HTML-u, JSP-u (Java Server Pages), SWT-u (Standard Widget Toolkit) ili bilo kom drugom paketu za izradu korisničkog interfejsa, dokle god za taj paket postoji generator.

Konceptualna skica preslikavanja modela u konkretan korisnički interfejs data je na dijagramu 2.



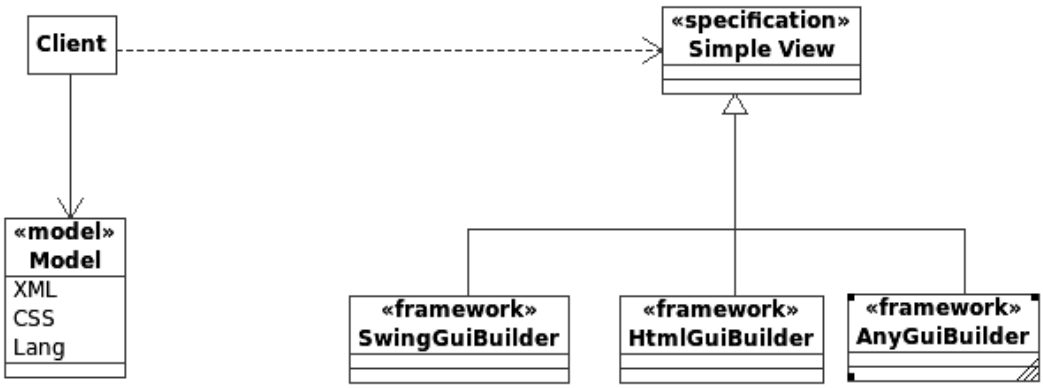
Dijagram 2: Sistemski prikaz generatora korisničkog interfejsa

4.2) Softversko rešenje

Ovo poglavlje sadrži opis softverskog dela ovog rada i predstavlja detaljnu razradu prethodnog opisa neophodnog rešenja. Uzevši u obzir potrebu za nezavisnošću u odnosu na konkretnu tehnologiju grafičkog korisničkog interfejsa, apstrakcija i implementacija su striktno razdvojene, pa će da budu i opisane u odvojenim poglavljima.

Za apstrakciju ovog softverskog rešenja sastavljena je specifikacija koji definiše model korisničkog intefejsa, dok je njena implementacija bilo koji softver koji generiše odgovarajući konkretan korisnički interfejs na osnovu tog modela.

Specifikacija softverskog dela rada zove se *Simple View*, a implementacija za Java Swing paket zove se *SwingGuiBuilder*.



Dijagram 3: Konceptualni dijagram klasa softverskog rešenja

Na dijagramu 3 je prikazan odnos između uvedenih pojmova. Koncept *Simple View* predstavlja specifikaciju za generatore korisničkog interfejsa, kao što su *SwingGuiBuilder* ili hipotetički *HtmlGuiBuilder*. Zbog uvođenja apstrakcije u vidu formalne specifikacije, struktura i sadržaj prethodno opisanih deskriptora moraju da budu formalno definisani. Deskriptori se takođe nalaze na dijagramu smešteni zajedno u jedan koncept po imenu *Model*.

Ključno pitanje koje se postavlja u ovom slučaju jeste: *Šta je zapravo Klijent?*

Klijent je može da bude programer, koji zahvaljujući ovakvom pristupu može da kreira forme u različitim tehnologijama na istovetan način ili alat za crtanje formi koji nije ograničen samo na jednu primenjivu tehnologiju. U svakom slučaju *Klijent* zahvaljujući korišćenju apstraktnog modela ne zavisi od konkretne tehnologije.

4.2.1) Apstrakcija

Simple View specifikacija je skup pravila koji definiše:

- skup podržanih komponenata grafičkog korisničkog interfejsa
- podržana svojstva za određivanje izgleda korisničkog interfejsa
- sintakse za određivanje strukture, prezentacije i lokalizacije korisničkog interfejsa
- ograničenja unutar kojih treba da rade generatori koji je implementiraju.

Prva reč u nazivu - **S**im**pl**e potiče od početnih slova reči **S**truktura, **P**rezentacija i **L**okalizacija, a druga reč – View bliže objašnjava za koju komponentu korisničkog interfejsa je specifikacija namenjena, u terminologiji MVC uzora.

Prema specifikaciji, View komponenta korisničkog interfejsa definiše se pomoću tri dokumenta: XML dokument za strukturu, CSS dokument za prezentaciju i Properties dokument za lokalizaciju. Svaki dokument je formulisan u skladu sa sintaksom koja najbolje odgovara problemu koji rešava.

Struktura korisničkog interfejsa jeste hijerarhijska predstava imenovanih elemenata, što je idealna struktura za predstavljanje u XML formatu. Potom, XML je jezik koji je veoma široko prihvaćen u svetu softvera i za koji postoje brojni parseri, programske biblioteke i editori.

Prezentacija se prilikom izrade web korisničkog interfejsa definiše u CSS formatu, dok kod izrade desktop korisničkog interfejsa to nije raširena praksa. Smatra se da je popularizovanje CSS-a u velikoj meri doprinelo razvoju web-a, tako da odabir CSS-a za propisanu sintaksu opisa prezentacije unutar *Simple View* modela predstavlja prenošenje dobre prakse razvijene za potrebe projektovanja web korisničkog interfejsa u oblast projektovanja korisničkog interfejsa za desktop aplikacije ili čak za teorijski proizvoljnu platformu.

Doduše, neophodno je napomenuti da je CSS korišćen u Simple View-u samo podskup celokupnog CSS standarda definisanog od strane W3C konzorcijuma koji se odnosi na aspekte relevantne za razvoj korisničkog interfejsa (npr. CSS pruža način za zasebno formatiranje prvog slova u paragrafu, što jeste veoma korisno za postavljanje enciklopedijskih članaka na sajt biblioteke, ali nije relevantno za razvoj korisničkog interfejsa). Takođe je neophodno napomenuti i da je upotrebljen pravi podskup, što znači da u Simple View CSS nisu dodavana nikakva svojstva koja ne postoje u osnovnom CSS standardu.

Sintaksa CSS jezika je definisana je preko tri koncepta: selektora, svojstva i vrednosti zapisanih u formatu:

```
selektor {
svojstvo1: vrednost1;
svojstvo2: vrednost2;
svojstvo3: vrednost3...
}
```

Simple View podskup CSS-a podržava tipske i id selektore, što znači da je moguće formatirati sve elemente istog tipa ili pojedinačne elemente referencirane po id atributu.

Klasni selektori nisu podržani, ali sličnu funkcionalnost moguće je postići pomoću grupnih selektora koji predstavljaju listu drugih selektora razdvojenu zarezima:

```
selektor1, selektor2, selektor3 ... {
svojstvo1: vrednost1
}
```

Kompletan CSS nudi nekoliko načina definisanja elemenata: statički, relativni, apsolutni i fiksni. Korišćeni podskup koristi isključivo apsolutno pozicioniranje, što u terminologiji CSS-a znači da je položaj svakog elementa određen preko udaljenosti od ivica svog neposrednog roditelja.

Što se lokalizacije tiče, koristi se format Javinih *properties* datoteka, sa dodatnim ograničenjem da su ključ i vrednost razdvojeni isključivo znakom jednakosti, a njihovi parovi znakom za novi red:

```
id1=Lokalizovani tekst1
id2=Lokalizovani tekst2
```

Lokalizacione datoteke su obavezne, međutim nije obavezno da svi elementi budu lokalizovani. Ne lokalizuju se elementi koji nemaju tekstualnu reprezentaciju (npr. *menu-bar*), a ukoliko element ima tekstualnu reprezentaciju, a nije lokalizovan, treba da bude upotrebljen njegov *id* iz XML opisa. Ove datoteke moraju da podržavaju *Unicode* enkripciju.

U ovom poglavlju prikazan je samo kratki opis *Simple View* specifikacije kao apstrakcije generatora grafičkog korisničkog interfejsa. Spisak svih podržanih komponenata, CSS svojstava, tipova vrednosti, funkcija i konstanti moguće je videti u kompletnoj specifikaciji u Dodatku B.

4.2.2) Implementacija

Za sada postoji samo jedna konkretna tehnologija korisničkog interfejsa za koju postoji *Simple View* implementacija. U pitanju je Javin standardni *Swing* paket. Generator *Swing* formi na osnovu modela definisanih u skladu sa *Simple View* specifikacijom se zove *SwingGuiBuilder* i čini sastavni deo softverskog dela ovog rada.

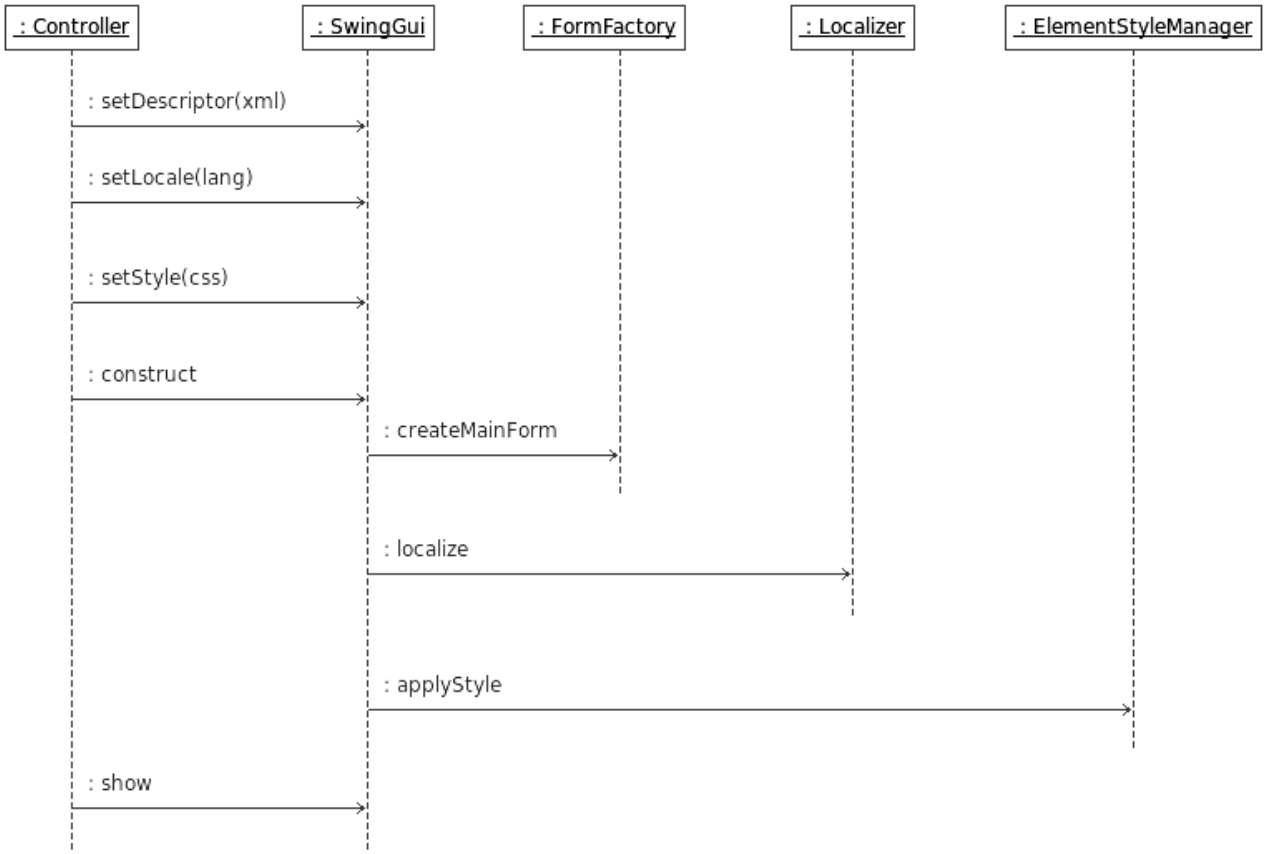
U nastavku je dat konceptualni opis implementacije, arhitektura softverskog rešenja i značaj pojedinih paketa i klasa. Interes za čitanje ovog poglavlja mogu da imaju svi čitaoci koje zanima kako rešenje funkcioniše, kao i oni čitaoci koji traže uputstva i smernice za kreiranje sopstvene *Simple View* implementacije za proizvoljnu tehnologiju.

Centralna klasa u celom softverskom paketu jeste *SwingGui*, koji predstavlja apstrakciju forme *Swing* korisničkog interfejsa. Ova klasa je ujedno i jedina klasa iz *SwingGuiBuilder* okvira koju programer aplikacije mora da koristi. Nakon kreiranja, objektu klase *SwingGui* se prosleđuju datoteke sa strukturom, prezentacijom i lokalizacijom napisane u skladu sa *Simple View* specifikacijom. Datoteke su prilikom prosleđivanja (pozivom odgovarajućih *set* metoda klase *SwingGui*) automatski parsirane u odgovarajuće objektne modele i na osnovu njih se generiše spreman za prikazivanje *Swing* korisnički interfejs.

Naravno, parsiranje, generisanje, primenu stila, lokalizovanje i sve ostale zadatke ne obavlja sama klasa *SwingGui*, već ih prosleđuje pomoćnim klasama u okviru paketa. Pomoćne klase su grupisane

po nameni i hijerarhijski su organizovane. Klasa *SwingGui* koristi tri takve klase:

- *FormFactory* - zadužena za kreiranje komponenata na osnovu XML opisa;
- *Localizer* - zadužena za lokalizovanje kreiranih komponenata;
- *ElementStyleManager* - zadužena za primenjivanje stilova definisanih u CSS opisu.

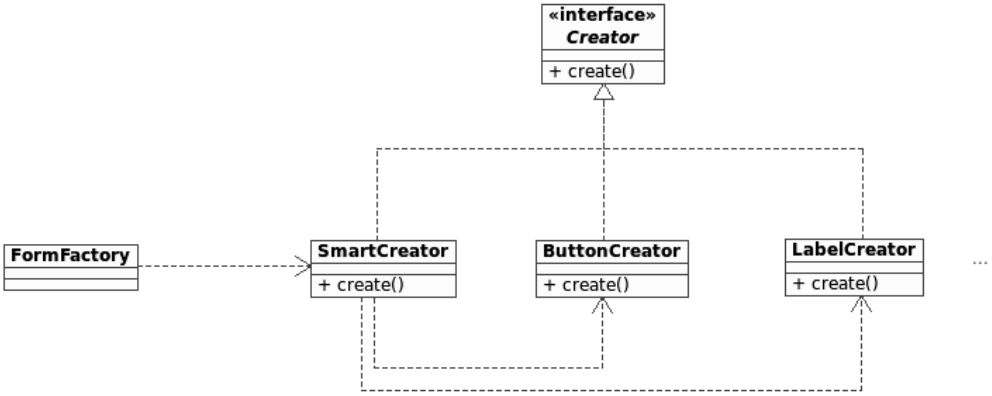


Dijagram 4: Osnovni sekvencni dijagram SwingGuiBuilder projekta

Na dijagramu 4 je prikazan predviđen način korišćenja klase SwingGui. Redosled poziva metoda je veoma bitan. Najpre treba postaviti sve tri datoteke modela, potom pokrenuti konstrukciju korisničkog interfejsa i tek onda ga prikazati. Redosled metoda je bitan i unutar metode *construct*: najpre se kreiraju sve komponente, potom se lokalizuju i napokon se primenjuju stilovi. Veoma je bitno da se lokalizacija izvršava pre primene stilova zbog toga što postoje neka svojstva koja se odnose na izgled tekstualnog sadržaja komponente (npr. *text-transform:capitalize*).

Kreiranje komponenata

Klasa *FormFactory* takođe obmotava čitavo mnoštvo drugih klasa koje imaju različita zaduženja. U osnovnoj klasi se kreira samo pojavljivanje klase *JFrame* i potom se rekurzivno popunjava objektima ugneždenih elemenata. Ostali elementi se kreiraju u svojim odgovarajućim *Creator* klasama. Klasa *FormFactory* od svih kreatorskih klasa poziva samo jednu – *SmartCreator*, koja sama određuje od kog konkretnog kreatora treba tražiti objekat. Dijagram klasa podsistema kreiranja komponenata dat je na dijagramu 5.



Dijagram 5: Dijagram klasa podsistema kreiranja komponenata

U slučaju potrebe da se doda novi tip komponente (usled izmene specifikacije, na primer) potrebno je dopisati kreatorsku klasu za novu komponentu i dodati par linija koda u *SmartCreator*. Ova prilagodljivost je postignuta zahvaljujući primeni *Proxy* uzora.

Kreiranje pojedinačnih komponenata različitih tipova je rešeno relativno jednostavno. Ipak, postoji još jedan detalj koji dodatno komplikuje projektovanje klasa. Naime, korišćeni algoritam pretpostavlja da se sve komponente dodaju u roditeljsku komponentu na isti način. U većini slučajeva način dodavanja je isti:

```
parent.add(child)
```

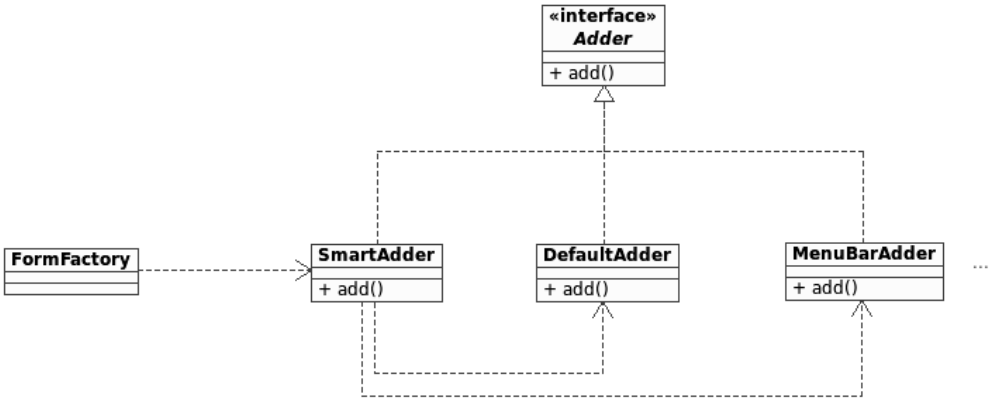
Međutim, u nekoliko slučajeva nije tako. Na primer, linija sa menijima (*menu-bar*) se u formu dodaje na sledeći način:

```
frame.setJMenuBar(jMenuBar)
```

A bilo koja komponenta se ubacuje u skrol (*scroll*) pomoću:

```
scroll.setViewportView(component)
```

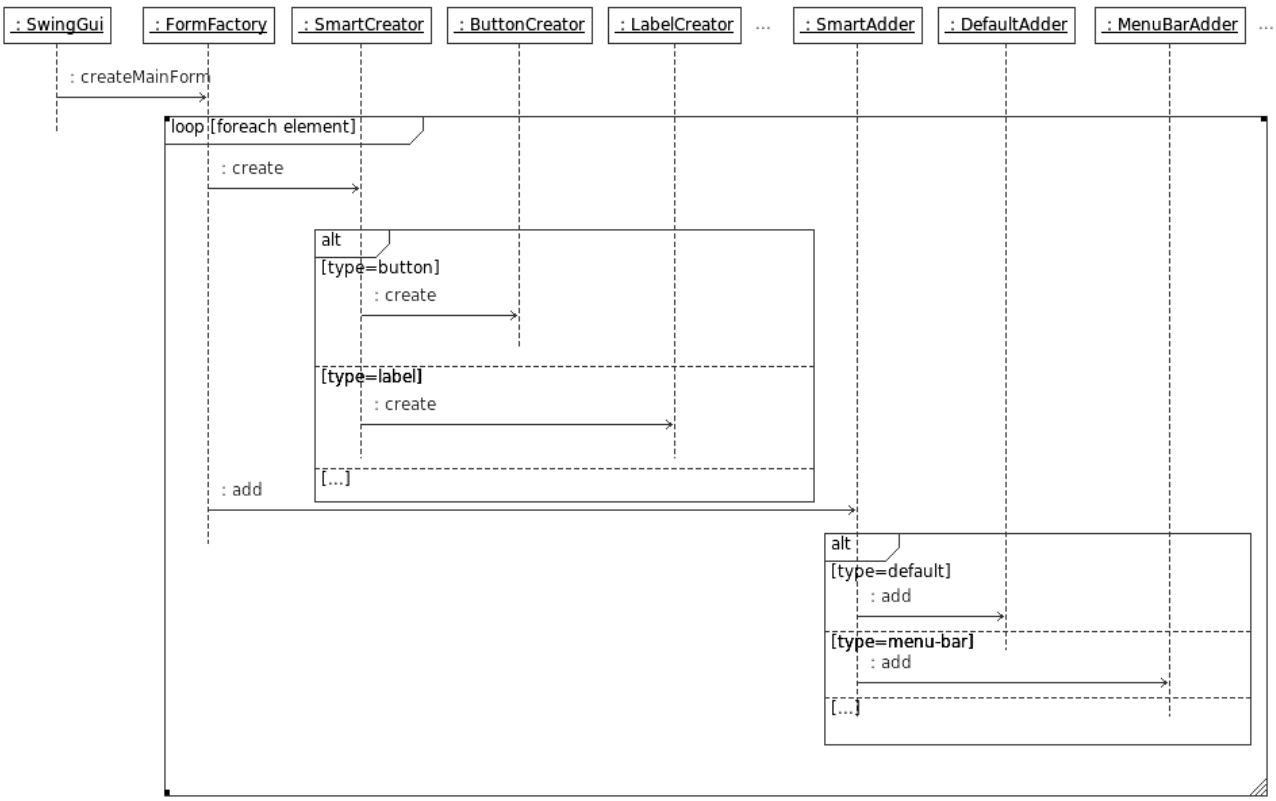
Na osnovu toga zaključujemo da neku vrstu apstrakcije treba uvesti i prilikom dodavanja komponente u roditeljski objekat. Problem dodavanja rešavamo ponovnom primenom *Proxy* uzora, na istovetan način kao ranije. *FormFactory* poziva samo klasu *SmartAdder*, a ona određuje koji će konkretan *Adder* biti upotrebljen. Dijagram klasa podsistema dodavanja komponenata dat je na dijagramu 6.



Dijagram 6: Dijagram klasa podsistema dodavanja komponenata

U slučaju takve izmene specifikacije kojom bi bio dodat takav novi tip komponente koji se u roditeljsku komponentu ubacuje na potpuno drugačiji način, u postojeći softver bi osim koda za kreatorske klase bilo potrebno dodati i novu klasu za dodavanje, kao i par linija koda u klasi *SmartAdder*.

Nakon što je trenutna komponenta dodata u roditeljsku, njen *ElementDescription* omotač objekat se dodaje u *hash* mapu kojom se elementi indeksiraju da bi im se kasnije pristupalo pomoću metode *findComponent(id)* iz klase *SwingGui*. Algoritam formiranja strukture korisničkog interfejsa, kreiranja i dodavanja komponenata prikazan je na dijagramu 7.



Dijagram 7: Sekvencni dijagram algoritma formiranja strukture korisničkog interfejsa

Lokalizovanje komponenata

Lokalizovanje obavlja klasa *Localizer* po veoma jednostavnom algoritmu. Najpre se lokalizuje koreni element (*form*), a potom se postupak rekurzivno ponavlja za sve potomke tog elementa i sve potomke njihovih potomaka. Lokalizovanje pojedinačnih elemenata se obavlja tako što se u objektnoj reprezentaciji lokalizacione datoteke pronađe član koji ima isti identifikator kao i trenutni element, a onda se elementu za vrednost teksta postavi lokalizovani string iz datoteke.

Da bi takav algoritam radio neophodno je da se svim elementima vrednost teksta postavlja na isti način. Nažalost, nije tako. Većini komponenata se vrednost teksta postavlja jednostavno:

```
component.setText(text)
```

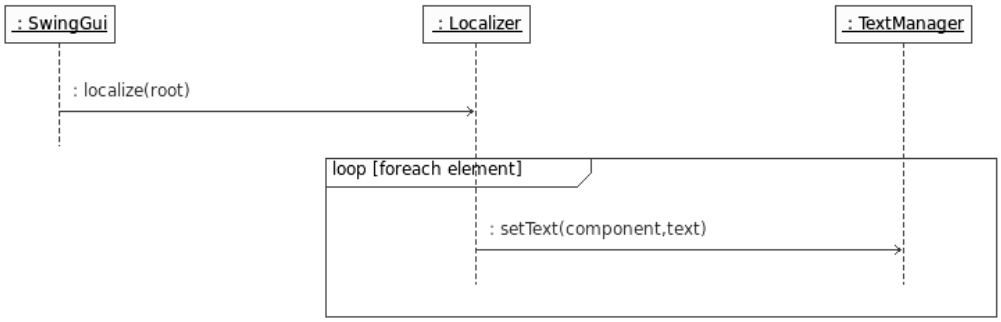
Ali postoji nekoliko tipova komponenata, za koje to ne važi. Na primer, za korenu komponentu (*form*) odgovarajući kôd glasi:

```
frame.setTitle(text)
```

A za ikonicu (*icon*):

```
icon.setToolTipText(text)
```

Rešenje za uočeni problem je uvođenje nove klase *TextManager* koja poseduje metodu kojoj se kao parametar proslede komponenta i tekst. Potom *TextManager* objekat proverava kog tipa je komponenta i u zavisnosti od toga poziva odgovarajuću metodu za promenu teksta. Algoritam za lokalizovanje komponenata pomoću *TextManager* klase prikazan je na dijagramu 8.

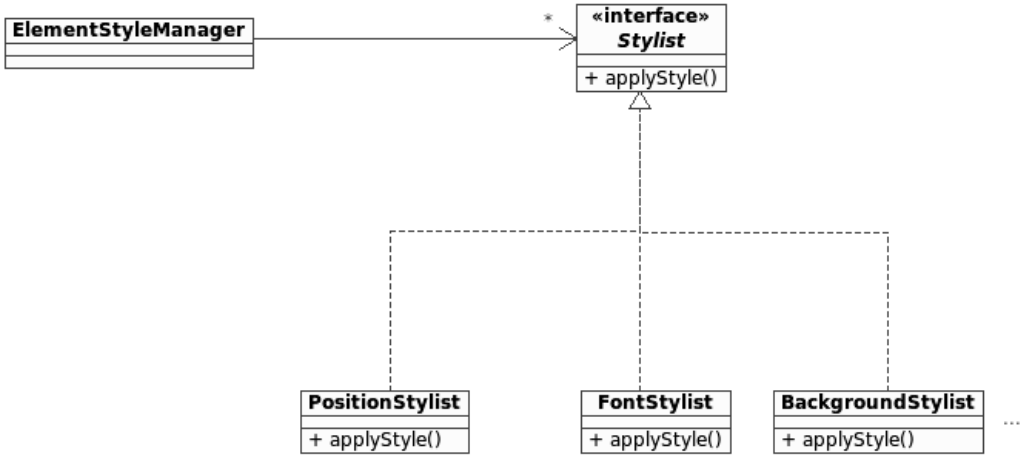


Dijagram 8: Sekvencni dijagram algoritma za lokalizovanje komponenata

Primenjivanje stilova

Za primenjivanje stilova nad komponentama je zadužena klasa *ElementStyleManager*. Klasa *SwingGui* joj, prilikom poziva metode *applyStyle*, prosleđuje koreni element, kao i objektni model CSS fajla.[9]

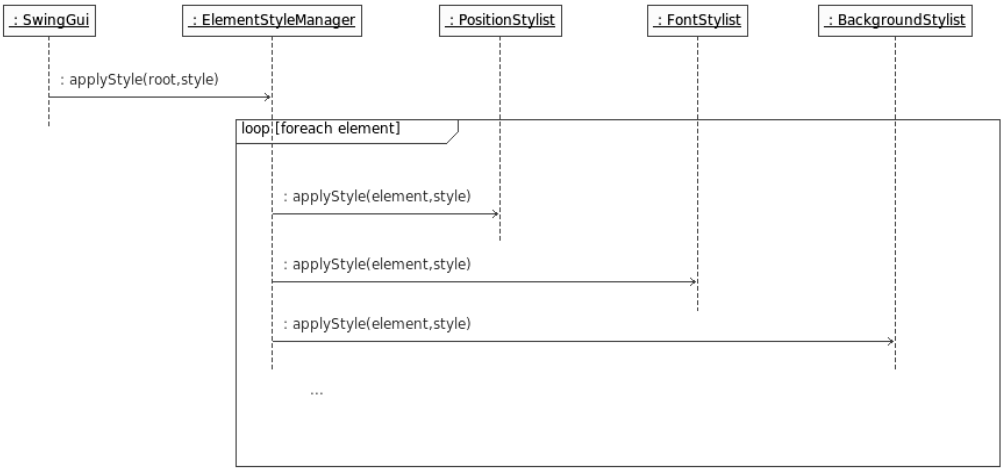
Primenjivanje stilova je grupisano po kategorijama (pozicija, pozadina, font, dimenzije...) i za svaku grupu napisana je jedna klasa. Sve klase zadužene za primenjivanje stilova iz određene kategorije implementiraju interfejs *Stylist.* Klasa *ElementStyleManager* sadrži kolekciju objekata tipa *Stylist*, kao što je prikazano na dijagramu 9. Elementi ove kolekcije su po jedan objekat svake klase koja implementira taj interfejs.



Dijagram 9: Dijagram klasa podsistema za primenu stilova

Sa dobro definisanim modelom klasa, algoritam se definiše relativno lako. Klasa *ElementStyleManager* poziva sve objekte iz kolekcije da primene odgovarajuća formatiranja prvo nad korenim elementom, a potom rekurzivno nad svim njegovim potomcima. Algoritam za primenu stilova je prikazan na dijagramu 10.

9 Projekat *SwingGuiBuilder* za parsiranje CSS dokumenata koristi pomoćni projekat – *CSSToolkit*, koji sadrži parser i paket klasa koje reprezentuju objektni model CSS dokumenta. Celokupan kôd parsera se takođe nalazi na pratećem disku.



Dijagram 10: Sekvencni dijagram algoritma za primenu stilova

U slučaju da je potrebno implementirati još neku kategoriju CSS stilova, dovoljno je napisati dodatnu implementaciju *Stylist* interfejsa i dodati je u *ElementStyleManager*.

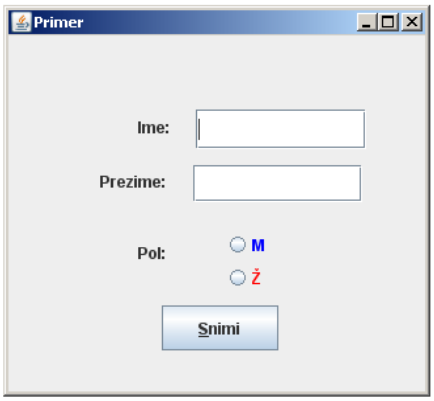
Stilovi se prvi put primenjuju prilikom poziva metode *construct*. Da bi bilo moguće ostvariti promenjive pozicije i veličine koje zavise od veličine prozora, neophodno je primeniti stilove iz tih grupa još po jednom svaki put kada dođe do promene veličine prozora. Za to je zadužena klasa *ResizeListener*, koja implementira interfejs *ComponentListener* i definiše metodu *componentResized* u kojoj poziva na ažuriranje onih grupa stilova koji zavise od veličine prozora. Objekat ove klase je registrovan na koreni *JFrame* objekat, tako da će svaki put kada je prozoru promenjena veličina biti obavljeno ažuriranje stilova nad svim njegovim elementima.

5) Uporedni pregled

Radi temeljne analize sličnosti i razlika između ranije opisanih tehnika, u ovom poglavlju je dat njihov uporedni pregled na konkretnom primeru. Data su rešenja za isti zadatak dobijena pomoću svake tehnike.

5.1) Zahtevi zadatka

Potrebno je napraviti korisnički interfejs kao na slici 8:



Slika 8: Uporedni primer – Zahtev zadatka

Tekst u labelama treba da bude poravnat uz desnu ivicu. Boja teksta uz prvo radio dugme treba da bude plava, a uz drugo crvena. Neophodno je omogućiti lokalizaciju cele forme izmenom samo jedne datoteke.

Formu treba realizovati u Java Swing tehnologiji.

Dodatni uslov je da veličina i pozicija elemenata zavise od dimenzija celog prozora, tj. da bude moguće menjati veličinu prozora tako da svi njegovi elementi ostanu proporcionalno raspoređeni.

5.2) Izrada primera

Korisnički interfejs opisan prethodnim zahtevima je realizovan pomoću sve četiri identifikovane tehnike:

1. Direktnim kodiranjem
2. Pomoću alata za crtanje – Odabrano je *NetBeans* okruženje za razvoj.
3. Pomoću okvira – U nedostatku dobrog okvira za *Swing* tehnologiju, namenjenog rešavanju problema kreiranja i formatiranja grafičkih komponenata forme, korišćene su pomoćne klase koje igraju ulogu okvira.
4. Pomoću jezika za označavanje – Korišćen je *Simple View* model, zasnovan na XML i CSS jezicima, sa *SwingGuiBuilder* implementacijom.

5.2.1) Pomoću direktnog kodiranja

Uobičajeni postupak za rešavanje ovakvog problema koristeći Swing kôd, obuhvata korišćenje nekog *LayoutManager*-a za upravljanje rasporedom komponenata. Budući da je najmoćniji i najpodesniji za direktno kodiranje svih netrivijalnih formi *GridBagLayout*, klasa koja predstavlja traženu formu napisana je pomoću njega. Klasa je data u nastavku sa komentarima koji dodatno pojašnjavaju pojedine delove koda:

```
public class PrimerSamKod extends JFrame{

	//potrebno je deklarisati sve komponente kao polja
	private JLabel firstNameLabel;
	private JTextField firstNameField;
	private JLabel lastNameLabel;
	private JTextField lastNameField;
	private JLabel genderLabel;
	private JPanel genderRadios;
	private JRadioButton male;
	private JRadioButton female;
	private ButtonGroup genderGroup;
	private JButton save;

	//i spisak lokalizovanih naziva
	private Properties locale;

	public static void main(String[] args) {
		PrimerSamKod psk = new PrimerSamKod();
		psk.init();
		psk.setVisible(true);
	}

	private void init(){

		//početna podešavanja forme
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(new Dimension(400,400));
		Container pane = getContentPane();

		//instanciranje svih komponenata
		firstNameLabel = new JLabel("firstName");
```

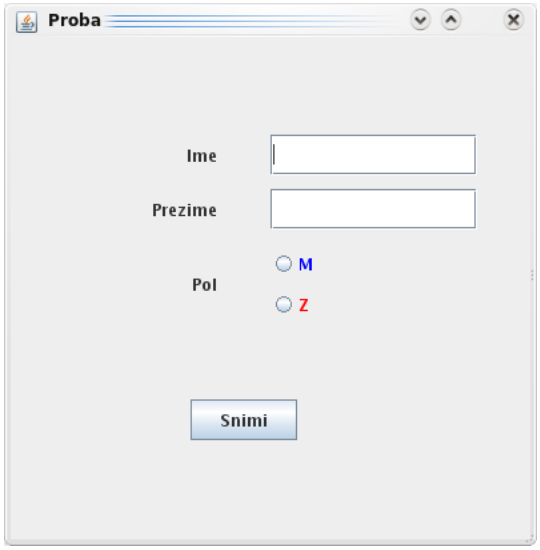
```
firstNameField = new JTextField();
lastNameLabel = new JLabel("lastName");
lastNameField = new JTextField();
genderLabel = new JLabel("gender");
genderRadios = new JPanel();
male = new JRadioButton("male");
female = new JRadioButton("female");
genderGroup = new ButtonGroup();
save = new JButton("save");

//punjenje lokalizovanih naziva
try {
       File f = new File("src/proba.lang");
       locale = new Properties();
       locale.load(new FileReader(f));
       //ukoliko fajl postoji izuzetak nije iskocio
       //i nazivi se primenjuju
       firstNameLabel.setText(locale.getProperty("firstName"));
       lastNameLabel.setText(locale.getProperty("lastName"));
       genderLabel.setText(locale.getProperty("gender"));
       male.setText(locale.getProperty("male"));
       female.setText(locale.getProperty("female"));
       save.setText(locale.getProperty("save"));
       this.setTitle(locale.getProperty("form"));
} catch (FileNotFoundException e) {
} catch (IOException e) {
}

//postavljanje odgovarajućih LayoutManager-a
LayoutManager layout = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();
pane.setLayout(layout);
genderRadios.setLayout(new BoxLayout(genderRadios, BoxLayout.Y_AXIS));

//postavljanje dimenzija komponenata
Dimension labelAndTextSize = new Dimension(100, 30);
Dimension genderGroupSize = new Dimension(100, 60);
Dimension radioSize = new Dimension(60,30);
Dimension buttonSize = new Dimension(80,30);

firstNameLabel.setPreferredSize(labelAndTextSize);
firstNameField.setPreferredSize(labelAndTextSize);
lastNameLabel.setPreferredSize(labelAndTextSize);
lastNameField.setPreferredSize(labelAndTextSize);
genderLabel.setPreferredSize(labelAndTextSize);
genderRadios.setPreferredSize(genderGroupSize);
male.setPreferredSize(radioSize);
female.setPreferredSize(radioSize);
save.setPreferredSize(buttonSize);

//... i svih ostalih formatiranja kao što su
//poravnanje, boja teksta ili font
//ukoliko želimo da su sve labele poravnate uz desnu ivicu,
//moramo da pozovemo odgovarajuću metodu za svaku labelu posebno
firstNameLabel.setHorizontalAlignment(JLabel.RIGHT);
lastNameLabel.setHorizontalAlignment(JLabel.RIGHT);
genderLabel.setHorizontalAlignment(JLabel.RIGHT);
male.setForeground(Color.BLUE);
female.setForeground(Color.RED);

//i napokon, GridBagLayout zahteva puno koda za
//definisanje ograničenja sa kojima postavlja komponente u formu
c.weightx = 0.5;
c.weighty = 0;
c.insets = new Insets(0,0,10,40);

c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
pane.add(firstNameLabel,c);

c.fill = GridBagConstraints.HORIZONTAL;
```

```
                c.gridx = 1;
                c.gridy = 0;
                pane.add(firstNameField,c);

                c.fill = GridBagConstraints.HORIZONTAL;
                c.gridx = 0;
                c.gridy = 1;
                pane.add(lastNameLabel,c);

                c.fill = GridBagConstraints.HORIZONTAL;
                c.gridx = 1;
                c.gridy = 1;
                pane.add(lastNameField,c);

                c.fill = GridBagConstraints.HORIZONTAL;
                c.gridx = 0;
                c.gridy = 2;
                pane.add(genderLabel,c);

                c.fill = GridBagConstraints.HORIZONTAL;
                c.gridx = 1;
                c.gridy = 2;
                pane.add(genderRadios,c);

                //da bi radio dugmići bili međusobno isključivi
                //moramo da ih dodamo u isti ButtonGroup.
                //ukoliko osim toga želimo i da su grupisani na istom mestu
                //unutar forme, treba da ih smestimo u isti JPanel.
                //tada se dodavanje vrši dvaput!
                genderRadios.add(male);
                genderRadios.add(Box.createVerticalGlue());
                genderRadios.add(female);
                genderGroup.add(male);
                genderGroup.add(female);

                c.fill = GridBagConstraints.NONE;
                c.gridx = 0;
                c.gridy = 3;
                c.gridwidth = 2;
                c.insets = new Insets(45,0,0,40);
                c.anchor = GridBagConstraints.PAGE_END;
                pane.add(save,c);

        }

}
```

Osim klase neophodna je i lokalizaciona datoteka *proba.lang* čiji sadržaj je:

```
form=Proba
firstName=Ime
lastName=Prezime
gender=Pol
male=M
female=Z
save=Snimi
```

Kada se program tek pokrene dobija se forma sa slike 9.



Slika 9: Uporedni primer – Direktno kodiranje

Primećujemo da je potrebno puno koda za relativno jednostavan primer sa svega nekoliko komponenata, kakav se često sreće u praksi. Koristeći isti postupak za izradu korisničkog interfejsa za neki znatno složeniji zadatak broj linija koda drastično raste, čineći pritom svaku naknadnu izmenu veoma teškim zadatkom.

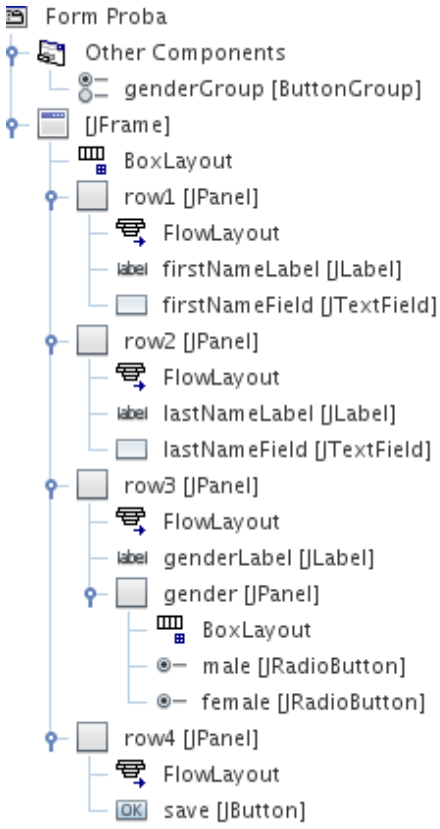
5.2.2) Pomoću NetBeans alata

Zaobilazno rešenje za problem ogromne količine koda koji treba napisati već postoji u vidu alata za generisanje grafičkog korisničkog interfejsa. Ovi alati nude programeru mogućnost da grafičkim putem, najčešće u vidu crtanja, definiše kako želi da forme izgledaju, a alati su ti koji generišu izvorni kôd koji omogućava takve forme. Na taj način je omogućen mnogo brži razvoj korisničkog interfejsa, ali broj linija koda nije smanjen, štaviše moguće je da, zbog generalizovanog algoritma za generisanje, koda čak ima i više. To nije problem prilikom samog kreiranja, i može da se desi da ne predstavlja problem ni prilikom održavanja, ukoliko je potrebno izvršiti izmenu koju je moguće izvesti preko alata. Međutim, ukoliko alat ne nudi željenu mogućnost, potrebno je ručno prepravljati generisani kôd.

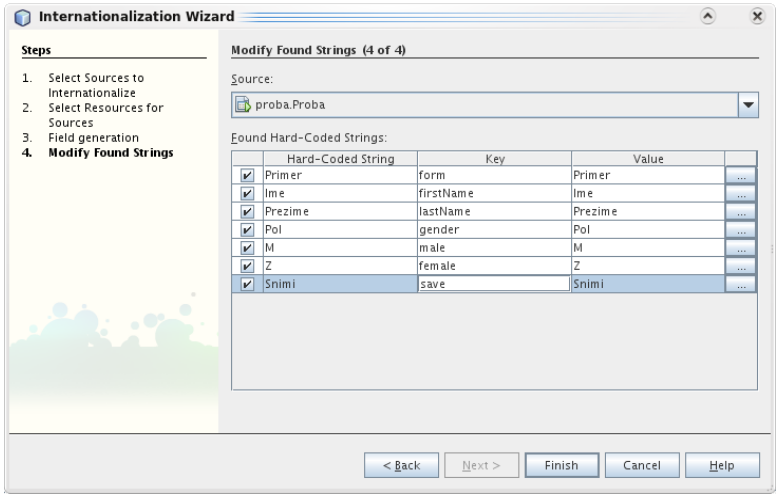
Još jedan nedostatak je što malo koji alat ima dobru podršku za složenije *LayoutManager*-e kao što je *GridBagLayout.* Zbog toga se često koriste alternative poput apsolutnog pozicioniranja ili kombinovanja jednostavnijih raspoređivača.

Pošto je jedan od uslova zadatka da bude omogućena promena veličine forme koja ima efekta nad svim njenim komponentama, apsolutno pozicioniranje nije dobro rešenje.

Drugo moguće rešenje dobija se kombinovanjem raspoređivača *BoxLayout* i *FlowLayout*. Prvi raspoređuje komponente jednu pored druge vertikalno ili horizontalno, zavisno od izbora, a drugi samo horizontalno. Da bismo ostvarili željeni raspored, potrebno je da korenom panelu postavimo vertikalni *BoxLayout*, u njega ubacimo četiri pomoćna panela koji igraju ulogu redova, potom svakom od tih panela postavimo *FlowLayout*. U prva dva panela ubacujemo labelu i tekstualno polje, tim redom, u treći panel labelu i dodatni panel koji grupiše radio dugmad, a u četvrti samo dugme. Takođe, potrebno je panelu u trećem redu postaviti vertikalni *BoxLayout* i u njega ubaciti radio dugmiće. Dobijena struktura forme predstavljena pomoću alatke za pregled hijerarhije komponenata razvojnog okruženja *NetBeans 6.1* prikazana je na slici 10.



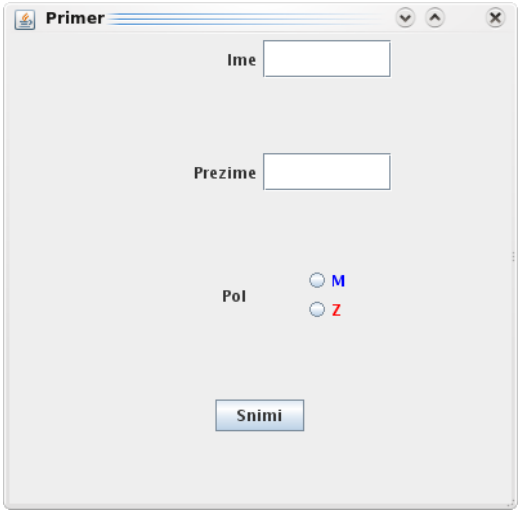
Slika 10: Hijerarhija komponenata primera u NetBeans-u



Slika 11: Forma za lokalizovanje u NetBeans-u

Nakon kreiranja forme, iz Tools menija biramo opciju *Internationalization>Internationalization Wizard* i lokalizujemo je korišćenjem ponuđenog dijaloga (na slici 11) koji umesto programera kreira lokalizacionu datoteku.

Nakon pokretanja programa pojavljuje se forma sa slike 12.



Slika 12: Uporedni primer - korišćenjem NetBeans alata

Korišćenjem alata, može da se stvori isti korisnički interfejs po ceni daleko manje uloženog vremena i truda. Jedan od nedostataka je taj što svi programeri koji rade na projektu moraju da koriste isti ili kompatibilan alat da bi vršili izmene pomoću vizuelnog editora. Npr. forma generisana u *NetBeans*-u, unutar *Eclipse* okruženja vidljiva je samo kao običan kôd. Ovo stvara određenu zavisnost razvojnog tima od konkretne tehnologije što bi trebalo izbegavati.

Takođe, kao što je već spomenuto, postoji mogućnost da se prilikom održavanja pojavi potreba da se korisnički interfejs menja u samom kodu, najčešće zbog nemogućnosti editora da izvrši izmenu. U tom slučaju bilo bi potrebno ručno menjati generisani kôd, što može da dovede do uvođenja grešaka jer programer menja mali deo koda, često bez sagledavanja celokupnog koda i uticaja izmene koju je uneo na ostale funkcionalnosti. Generisani kôd za korisnički interfejs iz ovog primera izgleda ovako:

```
/*
 * Proba.java
 *
 * Created on December 11, 2008, 1:31 PM
 */

package proba;

import java.util.ResourceBundle;

/**
 *
 * @author  goran
 */
public class Proba extends javax.swing.JFrame {
    private static final ResourceBundle bundle = java.util.ResourceBundle.getBundle("proba/Bundle");

    /** Creates new form Proba */
    public Proba() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        genderGroup = new javax.swing.ButtonGroup();
        row1 = new javax.swing.JPanel();
        firstNameLabel = new javax.swing.JLabel();
        firstNameField = new javax.swing.JTextField();
        row2 = new javax.swing.JPanel();
        lastNameLabel = new javax.swing.JLabel();
        lastNameField = new javax.swing.JTextField();
        row3 = new javax.swing.JPanel();
        genderLabel = new javax.swing.JLabel();
        gender = new javax.swing.JPanel();
        male = new javax.swing.JRadioButton();
        female = new javax.swing.JRadioButton();
        row4 = new javax.swing.JPanel();
        save = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle(bundle.getString("form")); // NOI18N
        setMinimumSize(new java.awt.Dimension(400, 400));
        getContentPane().setLayout(new javax.swing.BoxLayout(getContentPane(),
javax.swing.BoxLayout.Y_AXIS));

        firstNameLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        firstNameLabel.setText(bundle.getString("firstName")); // NOI18N
        firstNameLabel.setPreferredSize(new java.awt.Dimension(100, 30));
        row1.add(firstNameLabel);

        firstNameField.setPreferredSize(new java.awt.Dimension(100, 30));
        row1.add(firstNameField);

        getContentPane().add(row1);
```

```
        lastNameLabel.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
        lastNameLabel.setText(bundle.getString("lastName")); // NOI18N
        lastNameLabel.setPreferredSize(new java.awt.Dimension(100, 30));
        row2.add(lastNameLabel);

        lastNameField.setPreferredSize(new java.awt.Dimension(100, 30));
        row2.add(lastNameField);

        getContentPane().add(row2);

        genderLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        genderLabel.setText(bundle.getString("gender")); // NOI18N
        genderLabel.setPreferredSize(new java.awt.Dimension(100, 30));
        row3.add(genderLabel);

        gender.setMaximumSize(new java.awt.Dimension(100, 100));
        gender.setMinimumSize(new java.awt.Dimension(100, 100));
        gender.setLayout(new javax.swing.BoxLayout(gender, javax.swing.BoxLayout.Y_AXIS));

        genderGroup.add(male);
        male.setForeground(new java.awt.Color(0, 0, 255));
        male.setText(bundle.getString("male")); // NOI18N
        gender.add(male);

        genderGroup.add(female);
        female.setForeground(new java.awt.Color(255, 0, 0));
        female.setText(bundle.getString("female")); // NOI18N
        gender.add(female);

        row3.add(gender);

        getContentPane().add(row3);

        save.setText(bundle.getString("Snimi")); // NOI18N
        row4.add(save);

        getContentPane().add(row4);

        pack();
    }// </editor-fold>

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Proba().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JRadioButton female;
    private javax.swing.JTextField firstNameField;
    private javax.swing.JLabel firstNameLabel;
    private javax.swing.JPanel gender;
    private javax.swing.ButtonGroup genderGroup;
    private javax.swing.JLabel genderLabel;
    private javax.swing.JTextField lastNameField;
    private javax.swing.JLabel lastNameLabel;
    private javax.swing.JRadioButton male;
    private javax.swing.JPanel row1;
    private javax.swing.JPanel row2;
    private javax.swing.JPanel row3;
    private javax.swing.JPanel row4;
    private javax.swing.JButton save;
    // End of variables declaration
}
```

Jednostavnim poređenjem sa ručno napisanim kodom primećujemo razliku u broju linija i uređenosti. Isto tako valja obratiti pažnju na generisani komentar u kome je upozorenje programeru da ne menja automatski generisani kôd u označenoj sekciji, jer će sve unesene izmene biti izgubljene prilikom sledećeg ažuriranja između alata i koda. Dakle, čak i kada bi programer precizno poznavao ceo generisani kôd, mogu da postoje tehničke poteškoće na koje bi naišao prilikom održavanja.

5.2.3) Pomoću okvira

Za razliku od alata za crtanje, okviri zaista rešavaju problem velike količine i slabe uređenosti koda, tako što sadrže veoma generalizovan kôd za korisnički interfejs, koji podešavanjem određenih parametara treba prilagoditi u skladu sa potrebama.

Nažalost, i korišćenje okvira nosi sa sobom određene probleme, od kojih je najizraženiji problem zavisnosti od softverske platforme. Takođe treba istaći da su okvirima pokrivene uglavnom tehnologije za web korisnički interfejs, dok se za programiranje desktop formi češće koriste alati za crtanje. Kada je u pitanju *Swing* tehnologija, ne postoji zvanično podržani okvir koji rešava probleme kreiranja formi i komponenata[10], tako da je potrebno tražiti rešenje na drugom mestu.

U članku [25] predstavljene su dve klase namenjene za jednostavnije pravljenje formi:

- *LabelledItemPanel* – omogućava lako kreiranje panela sa parovima labele i polja za unos.
- *StandardDialog* – omogućava lako kreiranje forme sa jednim dugmetom za potvrdu unosa (*OK*) i jednim za poništavanje (*Cancel*).

U nedostatku pravog okvira, a budući da pomoću njih može da se lako napravi zadata forma, navedene klase će u ovom primeru igrati ulogu okvira. Teško da možemo dve klase da nazovemo okvirom, međutim uklapaju se u definiciju generalizovanog koda za korisnički interfejs, koji podešavanjem određenih parametara treba prilagoditi u skladu sa potrebama, tako da će adekvatno ilustrovati ideju korišćenja okvira, kao i prednosti i nedostatke.

Rezultati uporedne analize između korišćenja alata za crtanje i ove dve klase, prikazani u istom članku, pokazuju da je prosečno vreme kreiranja standardnih formi za unos podataka značajno smanjeno. Dodatna prednost je u tome što se izgled svih dijaloga i panela kreiranih korišćenjem ovih klasa može izmeniti samo na jednom mestu, što ne bi bilo moguće sa korišćenjem alata, budući da oni svaki put generišu novi kôd.

Da bi klasa *StandardDialog* bila primenjiva za izradu korisničkog interfejsa iz našeg zadatka, potrebno je izmeniti je tako da postoje odgovarajuće *get* i *set* metode za *JButton* objekte koji predstavljaju *OK* i *Cancel* tastere.[11]

10 Zvanični *Swing Application Framework* rešava probleme životnog ciklusa aplikacije, obrade događaja, upravljanja nitima i lokalizovanja resursa, ali ne bavi se problemom kreiranja i postavljanja komponenata na formu. [24]

11 Izmenjene klase su priložene na pratećem disku. Originalne klase je moguće preuzeti iz samog članka.

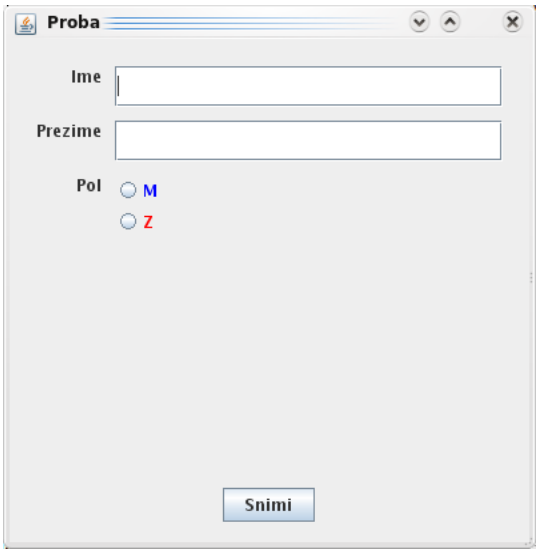
Sledeća klasa pokazuje kako se pravi korisnički interfejs prateći ovaj postupak:

```
public class PrimerOkvir {

	public static void main(String[] args) {

		try {
			//učitavanje lokalizovanih naziva
			File f = new File("src/proba.lang");
			Properties locale = new Properties();
			locale.load(new FileReader(f));

			//kreiranje svih komponenata (osim labela)
			//i postavljanje njihovih svojstava
			Dimension dimension = new Dimension(100,30);
			JTextField firstName = new JTextField();
			firstName.setPreferredSize(dimension);
			JTextField lastName = new JTextField();
			lastName.setPreferredSize(dimension);
			ButtonGroup group = new ButtonGroup();
			JPanel gender = new JPanel();
			gender.setLayout(new BoxLayout(gender, BoxLayout.Y_AXIS));
			JRadioButton male = new JRadioButton(locale.getProperty("male"));
			male.setForeground(Color.BLUE);
			JRadioButton female = new JRadioButton(locale.getProperty("female"));
			female.setForeground(Color.RED);
			group.add(male);
			group.add(female);
			gender.add(male);
			gender.add(female);

			//kreiranje panela i ubacivanje polja za unos
			//panel automatski pravi labele
			LabelledItemPanel panel = new LabelledItemPanel();
			panel.addItem(locale.getProperty("firstName"), firstName);
			panel.addItem(locale.getProperty("lastName"), lastName);
			panel.addItem(locale.getProperty("gender"), gender);

			//kreiranje dijaloga i ubacivanje panela
			StandardDialog dialog = new StandardDialog("PrimerOkvir");
			dialog.setContentPane(panel);

			//dodatno podešavanje panela i pokretanje
			dialog.getOkButton().setText(locale.getProperty("save"));
			dialog.getCancelButton().setVisible(false);
			dialog.setPreferredSize(new Dimension(400,400));
			dialog.pack();
			dialog.setVisible(true);

		} catch (Exception e) {
		}

	}

}
```

I lokalizaciona datoteka:

```
form=Proba
firstName=Ime
lastName=Prezime
gender=Pol
male=M
female=Z
save=Snimi
```

Nakon pokretanja pojavljuje se forma prikazana na slici 13.



Slika 13: Uporedni primer - korišćenjem okvira

Ukoliko uporedimo programski kôd koji je neophodan da bi se postiglo rešenje zadatka na ovaj način sa kodom potrebnim za postizanje istog rešenja pomoću direktnog kodiranja ili alata, primećujemo ogromnu razliku.

Korišćenjem okvira očigledno postižemo veliku prednost u odnosu na korišćenje alata:

- Kôd je manjeg obima i bolje uređen
- Specifičan kôd (klasa *PrimerOkvir*) se dinamički povezuje sa generalizovanim (*LabelledItemPanel* i *StandardDialog*), zbog čega je:
  - veća iskorišćenost postojećeg koda
  - moguća izmena izgleda svih dijaloga i panela na samo jednom mestu.[12]

Doduše, neki nedostaci i dalje postoje, kao što su velika vezanost za softversku platformu i nepostojanje dobrih okvira za sve tehnologije, pre svega za *Swing* paket.

Nedostatak okvira može relativno lako da se reši pravljenjem većeg broja okvira za različite tehnologije. Sa druge strane, zavisnost od platforme može da se ukloni tek upotrebom jezika za označavanje.

12 Npr. ukoliko želimo da promenimo način na koji su raspoređeni redovi labela i polja za unos, treba da promenimo klasu *LabelledItemPanel* i promena će imati efekta nad svim formama koje koriste tu klasu. To nije moguće prilikom korišćenja alata, zbog toga što oni svaki put generišu novi kôd.

5.2.4) Pomoću *Simple View* pristupa

Koristeći *Simple View* način za projektovanje korisničkog interfejsa razdvajamo željenu formu na njenu strukturu, prezentaciju (ili stil) i lokalizaciju na jezik po želji.

Počinjemo od strukture. Identifikujemo potrebne elemente i smeštamo ih u XML dokument:

```
<form id="form">
	<label id="firstName"/>
	<textfield id="firstNameField"/>
	<label id="lastName"/>
	<textfield id="lastNameField"/>
	<label id="gender"/>
	<group id="genderRadios">
		<radio id="male"/>
		<radio id="female"/>
	</group>
	<button id="save"/>
</form>
```

U ovom dokumentu je definisano samo koje se komponente pojavljuju na formi i koja je njihova međusobna hijerarhija. Identifikatori *id* su obavezni, jer služe sa jednoznačno referenciranje bilo kog elementa iz drugog modela (css, lokalizacija) ili iz kontrolera i moraju da budu jedinstveni.

Nakon što smo odredili koje komponente želimo, za definisanje njihovog izgleda koristimo CSS:

```
form{
height:400px;
width:400px
}

label{
height:30px;
width:25%;
left:15%;
text-align:right
}

textfield{
height:30px;
width:40%;
left:45%
}

#genderRadios{
height:25%;
width:25%;
left:45%
}

#firstName,#firstNameField{
top:15%
}

#lastName,#lastNameField{
top:30%
}

#gender, #genderRadios{
top:45%
}

radio{
height:30px;
width:60px;
```

```
left:25%
}

#male{
color:blue;
top:0px
}

#female{
color:red;
top:20px
}

#save{
height:30px;
width:80px;
top:75%;
left:45%
}
```

Element se formatira tako što se kao selektor napiše njegov *id* (kome prethodi znak #) iza koga sledi lista pravila, koja su data u obliku parova svojstvo-vrednost. Moguće je umesto identifikatora koristiti naziv tipa elementa (tada se znak # izostavlja), u kom slučaju se pravila primenjuju nad svim pojavljivanjima tog tipa. Videli smo u prethodnim primerima da korišćenjem bilo samog koda bilo alata za generisanje, primenjivanje pravila nad svim objektima jednog tipa nije moguće izvesti u samo jednom izrazu.

Unutar CSS dokumenta definišemo veličinu svih komponenata i to fiksno - izraženo u pikselima, ili relativno – izraženo u procentu od vrednosti istog svojstva neposrednog roditelja date komponente. Pozicija se definiše kao udaljenost od gornje ili donje i leve ili desne ivice dodeljenog prostora (unutrašnji prostor roditeljske komponente) u pikselima ili procentima. Samo ova funkcionalnost je sasvim dovoljna da omogući sve što nude *LayoutManager*-i, korišćenjem mnogo jednostavnijeg koda.

Osim dimenzija i položaja, u ovom delu modela definišemo i sva ostala formatiranja kao što su font, boja teksta ili pozadine i slično. U našem primeru neophodno je da sve labele imaju tekst koji je poravnat sa desne strane i da tekst pored radio dugmića bude odgovarajuće boje. Linije u kojima su određena ova svojstva su podebljane.

Treći korak je pisanje lokalizacione datoteke za proizvoljni jezik:

```
firstName=Ime
lastName=Prezime
gender=Pol
male=M
female=Z
save=Snimi
```

Identifikatoru svake komponente koja sadrži tekst pridružujemo njen lokalizovan naziv, koji će biti prikazan, ako je odabran taj jezik.

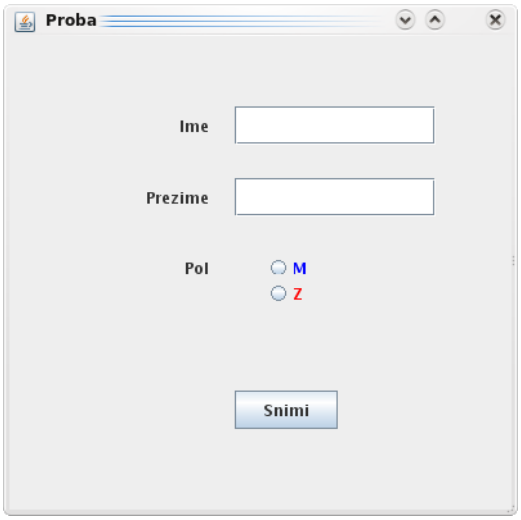
Ova tri deskriptora zajedno predstavljaju *View* komponentu MVC uzora, zbog toga što predstavljaju različite aspekte samog izgleda korisničkog interfejsa. Za njegovo povezivanje sa ostatkom aplikacije neophodan je *Controller*, koji je zadužen za kreiranje *View* komponente i njeno ponašanje. Kontroler treba da bude napisan kao Java klasa.

```
public class PrimerSimpleView{

	private SwingGui gui;

	public static void main(String[] args) throws Exception{
		PrimerSimpleView controller = new PrimerSimpleView();
	}

	public PrimerSimpleView() throws Exception {

		gui = new SwingGui();
		gui.setDescriptor("src/proba.xml");
		gui.setStyle("src/proba.css");
		gui.setLocale("src/proba.lang");

		gui.construct();
		gui.show();

		JButton save = (JButton) gui.findComponent("save");
	}

}
```

U ovom kontroleru ima relativno malo koda, zato što napisana forma za sada ništa ne radi. Postupak predviđa pravljenje objekta klase *SwingGui*, postavljanje datoteka koji sadrže prethodna tri modela, pozivanje metode *construct()* koja priprema, postavlja, formatira i lokalizuje sve elemente i pozivanje metode *show()* koja prikazuje formu.

Do bilo koje komponente moguće je doći pozivom metode *findComponent* koja kao parametar prima *id* te komponente iz XML opisa. Nakon toga je moguće dinamički menjati svojstva te komponente, izvaditi i parsirati sadržaj, dodeliti joj *event handler*, popuniti vrednostima iz baze (npr. ako je u pitanju *combo box*) i vršiti sve ostale akcije koje logički potpadaju pod domen kontrolera.

Nakon pokretanja programa prikazuje se ekranska forma data na slici 14.



Slika 14: *Uporedni primer – Simple View*

Ukoliko želimo da forma nešto radi treba da izmenimo kontrolersku klasu. Primera radi, naša forma bi mogla da na konzoli ispiše ime i prezime unete osobe kad korisnik klikne na dugme. Izmenjeni kontroler koji ovo omogućava izgleda ovako:

```
public class ProbaController implements ActionListener{

	private SwingGui gui;

	public static void main(String[] args) throws Exception{
		ProbaController controller = new ProbaController();
	}

	public ProbaController() throws Exception {

		gui = new SwingGui();
		gui.setDescriptor("src/proba.xml");
		gui.setStyle("src/proba.css");
		gui.setLocale("src/proba.lang");

		gui.construct();
		gui.show();

		JButton save = (JButton) gui.findComponent("save");
		save.addActionListener(this);
	}

	@Override
	public void actionPerformed(ActionEvent event) {

		JTextField firstNameField = (JTextField) gui.findComponent("firstNameField");
		JTextField lastNameField = (JTextField) gui.findComponent("lastNameField");
		String firstName = firstNameField.getText();
		String lastName = lastNameField.getText();
		System.out.println(firstName + " " + lastName);

	}
```

5.3) Analiza rezultata poređenja

Ukoliko je kriterijum poređenja obimnost koda, *Simple View* način ima ubedljivu prednost. Takođe, prateći isti postupak dobijeni kôd nije samo kraći, već i uređeniji, jer je podeljen na delove u zavisnosti od svrhe i zbog toga što forsira primenu MVC uzora i drugih pravila dobre prakse.

Primećujemo da korišćenje okvira u dobroj meri rešava problem, međutim postignuto rešenje je i dalje zavisno od platforme. Nezavisnost od platforme se postiže tek uvođenjem *markup* sintakse, kakva se koristi u *Simple View* pristupu.

Dobar kontraargument *Simple View* pristupu jeste da je, uprkos svim inovacijama, korišćenje dobrog alata za crtanje i dalje lakše i brže. To jeste tačno kada je reč o prvobitnom razvoju aplikacije, međutim korisnički interfejs izrađen praćenjem predloženog postupka je, zbog njegove izrazito modularne strukture, mnogo lakše održavati.[13]

Još jedna prednost korisničkog interfejsa kreiranog na predloženi način jeste da je moguće

13 U Dodatku A je dat primer izrade korisničkog interfejsa za nešto složeniju aplikaciju, kao i primeri primene nekoliko čestih tipova izmene prilikom održavanja.

realizovati sve izmene koje se odnose samo na View sloj bez ponovnog kompajlovanja programa. Neophodno je samo promeniti odgovarajuće tekstualne datoteke i ponovo pokrenuti aplikaciju.

Glavne prednosti *Simple View* pristupa mogu se svesti na:

- Nezavisnost korisničkog interfejsa od softverske platforme
- Rešavanje različitih problema u zasebnim celinama, zahvaljujući poštovanju pravila dobre prakse (struktura forme, njena prezentacija i lokalizacija se nalaze u odvojenim datotekama)
- Potrebno je manje programskog koda

Uporedni pregled nam pokazuje i razloge usled kojih je manji broj linija koda u svakoj programskoj celini:

- Upotreba adekvatne sintakse za svaki problem
- Programiranje na višem nivou apstrakcije

5.3.1) Prednosti XML sintakse

Primenom XML sintakse postignuto je značajno smanjivanje broja linija koda u odnosu na ekvivalent pisan Java sintaksom. Razlog se nalazi u činjenici da jedna XML predstava grafičke komponente reprezentuje i njeno kreiranje i dodavanje u komponentu predstavljenu roditeljskim elementom.

Kreiranje elemenata i dodavanje u roditeljske elemente pomoću Java sintakse (izvor – *Primer#1*):

```
genderRadios = new JPanel();
male = new JRadioButton("male");
female = new JRadioButton("female");
genderGroup = new ButtonGroup();
save = new Jbutton("save");

genderRadios.add(male);
genderRadios.add(female);

genderGroup.add(male);[14]
genderGroup.add(female);
```

Kreiranje elemenata i dodavanje u roditeljske elemente pomoću XML sintakse (izvor – *Primer#3*):

```
<group id="genderRadios">
        <radio id="male"/>
        <radio id="female"/>
</group>
```

U izdvojenim odlomcima koda primećujemo da je za postizanje istog efekta korišćenjem Java sintakse potrebno 7 linija koda (poslednje dve linije se ne računaju), a korišćenjem XML-a samo 4 linije. Ova razlika je još drastičnija ukoliko uzmemo u obzir *import* naredbe i deklaracije polja.

14 Isto tako, u poslednje dve linije Java primera vidimo još jednu prednost korišćenja generatora na osnovu formalnog opisa, a to je uvođenje novog nivoa apstrakcije, o čemu će još biti reči.

5.3.2) Prednosti CSS sintakse

Da bi se primenilo neko svojstvo svim elementima istog tipa na tradicionalan način (kôd ili alat) potrebno je pozvati odgovarajuću *set* metodu pojedinačno nad svim elementima tog tipa, a isti efekat se u CSS sintaksi postiže jednim blokom naredbi. U analiziranom primeru je bilo potrebno svim labelama postaviti desno poravnanje teksta.

Postavljanje poravnanja svim labelama pomoću Java sintakse (izvor – *Primer#1*):

```
firstNameLabel.setHorizontalAlignment(JLabel.RIGHT);
lastNameLabel.setHorizontalAlignment(JLabel.RIGHT);
genderLabel.setHorizontalAlignment(JLabel.RIGHT);
```

Postavljanje poravnanja svim labelama pomoću CSS sintakse (izvor – *Primer#3*):

```
label{
text-align:right
}
```

Da je trebalo da napraviti formu sa npr. 34 labele, u Java kodu bismo morali da imamo isto toliko linija samo za postavljanje poravnanja, a u CSS kodu bismo koristili potpuno isti blok.

Štaviše, razlike u sintaksi imaju efekta i na održavanje. Moguć je slučaj da je potrebno izmeniti kôd tako da tekst u svim labelama bude poravnat udesno, a da samo jedna labela, u našem primeru bi to mogla da bude ona koja označava prezime, ima centriran tekst.

U Java kodu bi bilo potrebno tražiti liniju koja postavlja poravnanje toj konkretnoj labeli i promeniti je:

```
firstNameLabel.setHorizontalAlignment(JLabel.RIGHT);
lastNameLabel.setHorizontalAlignment(JLabel.CENTER);//promena
genderLabel.setHorizontalAlignment(JLabel.RIGHT);
```

A u CSS kodu je dovoljno dodati još jedan blok:

```
label{
text-align:right
}

//novi blok
#lastNameLabel{
text-align:center
}
```

Prednost je u tome što je minimizovana potreba ručnog menjanja starog koda. Ako primer zaista ima 34 labele, mnogo je lakše dodati mali blok CSS koda nego prolaziti kroz Java klasu, tražeći jednu liniju koju treba izmeniti.

5.3.3) Prednosti višeg nivoa apstrakcije

Prilikom analize dodavanja radio dugmića u grupu, primetili smo da je Java kôd sadržao duplo dodavanje:

```
genderRadios = new JPanel();
male = new JRadioButton("male");
female = new JRadioButton("female");
genderGroup = new ButtonGroup();
save = new Jbutton("save");

//dodavanje u panel
genderRadios.add(male);
genderRadios.add(female);

//dodavanje u ButtonGroup
genderGroup.add(male);
genderGroup.add(female);
```

Bilo je potrebno jednom dodati dugmiće u panel koji ih vizuelno grupiše, a potom ih ponovo dodati u *ButtonGroup* objekat koji omogućava njihovu međusobnu isključivost. U Simple View kodu nije tako:

```
<group id="genderRadios">
        <radio id="male"/>
        <radio id="female"/>
</group>
```

Element *group* kreira i panel i mehanizam za ostvarivanje međusobne isključivosti, tako da su dodavanjem u taj element radio dugmići implicitno dodati i u istu grupu. Doduše, povezivanje vizuelnog i logičkog grupisanja nije obavezujuće. Ukoliko programer (iz ma kog razloga) želi da radio dugmiće grupiše samo vizuelno, ali da ne budu međusobno isključivi, umesto *group* elementa treba da koristi *panel*. Opet, ukoliko želi da radio dugmići ne budu vizuelno grupisani, ali da budu međusobno isključivi, treba da ih proizvoljno rasporedi u modelu i ručno doda u isti *ButtonGroup* iz kontrolera.

6) Zaključak

Na početku rada postavljeni su ciljevi:

1. Ukazati na nedostatke postojećih tehnika, tehnologija i alata za razvoj korisničkog interfejsa.
2. Pružiti rešenje u vidu alternativnog postupka koji je lišen uočenih nedostataka.

Najpre je definisan pojam korisničkog interfejsa. Nakon toga je prikazan istorijat korisničkog interfejsa počev od perioda komandne linije, zaključno sa grafičkim korisničkim interfejsom (desktop i web varijanta). Pokazano je da je većina tehnologija grafičkog korisničkog interfejsa u skladu sa WIMP paradigmom i da je zahvaljujući tome moguć njihov platformski nezavisan opis.

Potom je dat pregled postojećih tehnika za razvoj korisničkog interfejsa i njihovih odgovarajućih tehnologija:

- Direktno kodiranje
- Korišćenje alata
- Korišćenje okvira
- Korišćenje jezika za označavanje

Za svaku tehnologiju su nabrojane prednosti i nedostaci:

→ Direktno kodiranje pruža velike mogućnosti, po cenu mnogo koda.

→ Alati za crtanje olakšavaju pravljenje formi, ali mogu da otežaju kasnije održavanje.

→ Forma napravljena korišćenjem alata za crtanje ne može da se dinamički menja u vreme izvršenja programa.

→ Okviri su najčešće vezani za određenu platformu.

→ Jezici za označavanje korisničkog interfejsa mogu da budu jednako složeni (ili složeniji) nego običan programski kôd.

Ovim je ostvaren prvi cilj rada

Na osnovu saznanja prikupljenih detaljnom analizom postojećih tehnika i njihovih nedostataka, projektovani su:

Simple View – platformski nezavisna specifikacija korisničkog interfejsa zasnovanog na WIMP paradigmi projektovana u skladu sa pravilima dobre prakse.

SwingGuiBuilder – implementacija iste specifikacije za Java Swing tehnologiju.

Realizacijom ova dva projekta pruženo je rešenje koje otklanja probleme prethodnih tehnika, čime je ostvaren drugi cilj rada.

Nakon opisa softverskog rešenja, dat je uporedni pregled postojećih tehnika i *Simple View* pristupa na primeru izrade jednostavne ekranske forme. Analiziran je programski kôd dobijen svakom tehnikom, utvrđeno je da *Simple View* primer ima ubedljivu prednost u odnosu na primere dobijene korišćenjem ostalih tehnika i ustanovljeni su principi na kojima se zasniva uspešnost postignutog rešenja:

1. Platformska nezavisnost korisničkog interfejsa
2. Poštovanje pravila dobre prakse
3. Upotreba adekvatne sintakse za svaki problem
4. Uvođenje novog nivoa apstrakcije

Ipak, pored mnogih prednosti novi postupak ima i određene nedostatke koje, zbog kompletnosti rada, valja istaći.

++	→←	--
Lak za upotrebu	ALI	Dodatna cena po performanse
Prilagodljivo projektovanje		Nedovoljna podrška za dinamiku
Jedan jezik za sve GUI tehnologije		Za sad podržan samo Swing

Tabela 3: Ključne prednosti i odgovarajući nedostaci

Nadamo se da će se pokazati da *Simple View* programerima donosi mnogo više pogodnosti nego nedostataka, kao i da će svi postojeći nedostaci biti prevaziđeni u dogledno vreme.

6.1) Mogućnosti za proširenje

Svaki poduhvat, uključujući i softverske projekte, ma koliko dobro realizovan u svom početnom stanju, da bi bio dugoročno uspešan mora da bude otvoren ka promenama i poboljšanjima. Projekat opisan u ovom radu ima puno mogućnosti za proširenje i još više potreba za poboljšanjem i popravkom. Neke od mogućnosti su nabrojane u nastavku.

Potrebno je:

- Još podržanih komponenata – Spisak podržanih komponenata još uvek nije sveobuhvatan, tako da može da se očekuje pojava dodatnih elemenata u novijim verzijama *Simple View* specifikacije.
- Još podržanih stilova – Ukoliko se pokaže da su potrebne još neke funkcionalnosti ili svojstva za bogatije ukrašavanje formi, moguće je proširivanje podskupa podržanog CSS-a.
- Još implementacija – Ne postoje generatori za ostale prezentacione tehnologije kao što su SWT, JSP i mnoge druge. Bilo bi posebno korisno imati i web implementaciju *Simple View*-a.

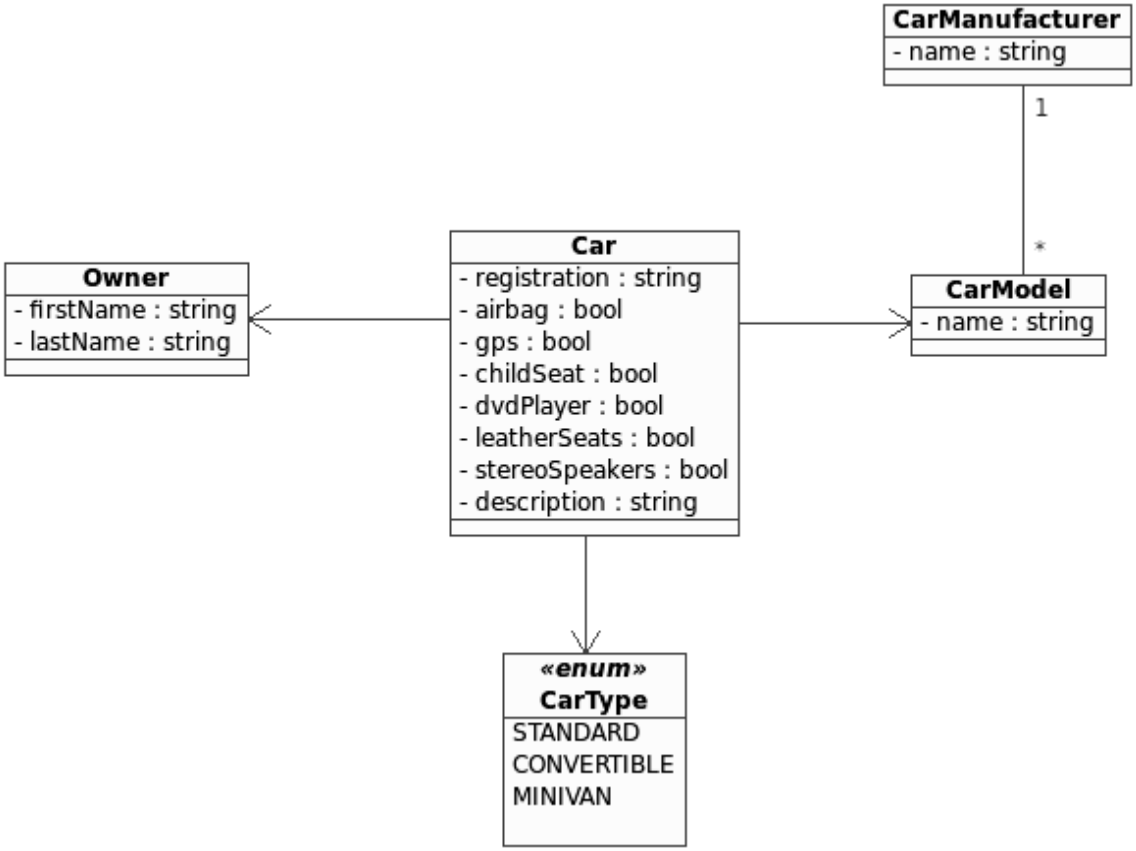
- Poboljšavanje dinamike formi – Dodavanje posebnih elemenata za upravljanje dinamikom forme bi u velikoj meri povećalo upotrebnu vrednost projekta.
- Obuhvatanje problema korisničkog interfejsa i šire od same *View* komponente – Hipotetički *Simple Controller* bi bio odličan komplement postojećem projektu.
- Postojanje alata za crtanje – Glavni argument protiv trenutno postojećih alata za crtanje korisničkog interfejsa je taj da oni samo zaobilaze problem ogromne količine koda. Ukoliko bi alat umesto programskog koda generisao *Simple View* model, taj argument više ne bi bio važeći. Štaviše, takvim alatom bi bilo moguće kreirati korisnički interfejs za bilo koju tehnologiju sa dostupnom *Simple View* implementacijom.

Dodatak A: Primer napredne upotrebe

Svrha primera iz uporednog pregleda bila je da prikaže razlike između postojećih postupaka i Simple View-a. Ipak, taj primer nije dovoljno složen da bi prikazao pune mogućnosti ove tehnike. Zbog toga je u ovom dodatku dat primer koji bi trebalo da detaljnije ilustruje Simple View pristup.

Primer obuhvata realizaciju jednog slučaja korišćenja tipičnog za informacione sisteme. Potrebno je omogućiti čuvanje podataka o automobilima u skladištu podataka. Primer treba da bude izgrađen u skladu sa troslojnom arhitekturom – neophodni su sloj podataka, domenske logike i prezentacioni sloj. Radi jednostavnosti, a budući da nam je cilj da se fokusiramo na prezentacioni sloj, skladište podataka će biti simulirano običnom kolekcijom u memoriji.

Domenski model primera je dat na dijagramu 11.



Dijagram 11: Napredni primer - Dijagram klasa domenskog modela

Najpre je potrebno napisati sve identifikovane klase. Budući da su u pitanju jednostavne Java klase koje poseduju samo polja prikazana na dijagramu i odgovarajuće *get* i *set* metode, kôd za njih će zbog jednostavnosti biti izostavljen iz ovog dodatka.

Klasa *Car* sadrži broj registarskih tablica (*registration*), veze ka svom vlasniku i modelu (a preko modela i tranzitivnu vezu ka proizvođaču) i polje nabrojivog tipa *CarType*, koje definiše da li je u pitanju običan automobil, kabriolet ili kombi. Polja *boolean* tipa označavaju postojanje dodataka

kod tog automobila.

Potom, potrebno je implementirati sloj podataka. Klasa *DBBroker* obmotava skladište podataka koje je predstavljeno klasom *SimulatedDataBase*. Metode koje pruža su *getAllCars()*, *getAllManufacturers()* i *getAllOwners()*. Simulirana baza podataka sadrži kolekcije sa "perzistiranim" objektima i kôd koji ubacuje početne vrednosti u te kolekcije.

Nakon što su sloj aplikacione logike i sloj podataka završeni, pristupamo izradi korisničkog interfejsa. Budući da slučaj korišćenja koji implementiramo obuhvata unos novog automobila, potrebna nam je forma za editovanje objekata klase *Car*.

Zahtevi vezani za sadržaj forme:

- Potrebno je da veze automobila sa vlasnikom, proizvođačem i modelom predstavimo *combo-box* komponentama i to tako da je ponuđeni izbor modela određen odabranim proizvođačem.
- Broj registarskih tablica može da bude unet pomoću običnog tekstualnog polja.
- Dodatke predstavljene boolean poljima treba uneti pomoću *check-box* komponenata. Ove komponente u formi treba grupisati na one koje određuju dodatke vezane za bezbednost (vazdušni jastuk, GSP i dečije sedište) i dodatke namenjene zabavi vozača (DVD plejer, kožna sedišta i ozvučenje). Ovo grupisanje treba ostvariti pomoću tabova.
- Tip automobila (standardni, kabriolet ili kombi) treba uneti odabirom između tri radio dugmeta.
- Dodatni opis automobila treba uneti pomoću višelinijskog tekstualnog polja koje mora da podržava skrolovanje u slučaju da je uneto više teksta nego što ima mesta.

Sada je potrebno implementirati korisnički interfejs prema definisanim zahtevima. U okviru tronivojske arhitekture koristimo *Model-View-Controller* uzor. Komponentu *Model* smo već napravili u prethodna dva koraka. Sada je red na *View* komponentu koju pravimo koristeći *Simple View* specifikaciju.

Prvim opisom (u datoteci *res/carsale.xml*) definišemo strukturu forme:

```
<form id="newcar">
	<label id="registration"/>
	<textfield id="registrationField"/>
	<label id="manufacturer"/>
	<combo id="manufacturerCombo"/>
	<label id="model"/>
	<combo id="modelCombo"/>
	<label id="owner"/>
	<combo id="ownerCombo"/>
	<label id="type"/>
	<group id="typeGroup">
		<radio id="standard"/>
		<radio id="convertible"/>
		<radio id="minivan"/>
	</group>
	<tabbed-panel id="accesories">
		<tab id="safety">
			<checkbox id="airbag"/>
			<checkbox id="gps"/>
			<checkbox id="childSeat"/>
		</tab>
		<tab id="leisure">
			<checkbox id="dvdPlayer"/>
			<checkbox id="stereoSpeakers"/>
			<checkbox id="leatherSeats"/>
		</tab>
	</tabbed-panel>
	<label id="description"/>
	<scroll id="descriptionScroll">
		<textarea id="descriptionArea"/>
	</scroll>
	<button id="save"/>
</form>
```

U ovom dokumentu smo odredili koje elemente želimo na formi, njihovu hijerarhiju i jedinstvene logičke nazive predstavljene identifikatorom *id*.

Od ranije korišćenih elemenata tu su koreni element *form*, kao i *label*, *textfield* i *button.* Novi elementi su *combo* koji očigledno predstavlja combo box, *radio* za radio dugme, *group* element koji omogućava međusobnu isključivost svih sadržanih radio dugmića, *tabbed-panel* i *tab* za grupisanje elemenata u tabove, a u svakom tabu po tri *checkbox* elementa. Za višelinijsko unošenje teksta koristi se *textarea* element, a ukoliko želimo funkcionalnost skrolovanja, treba da ga ubacimo unutar *scroll* elementa.[15]

Nakon što smo definisali strukturu korisničkog interfejsa, treba da zasebnim CSS modelom (u datoteci *res/carsale.css*) definišemo njen izgled.

Prednost CSS sintakse jeste što omogućava formatiranje grafičkih elemenata na veoma jednostavan način, što i nije neobično kada se uzme u obzir da je upravo za tu svrhu namenski i smišljena. Tabelarni raspored elemenata vrlo lako postižemo tako što svim labelama postavimo istu udaljenost od leve ivice i širinu, a svim ulaznim elementima (textfield, combo, group i scroll) postavimo istu tu udaljenost od leve ivice uvećanu za širinu labela i još neki razmak.

15 Upotrebljeni elementi čine samo podskup ukupnog broja podržanih elemenata. Ovaj primer je pravljen sa ciljem da što bolje ilustruje *Simple View* na konkretnom primeru, međutim ne postoji takav konkretan primer gde bi bilo moguće upotrebiti sve podržane elemente. Za ceo spisak videti kompletnu specifikaciju u Dodatku B.

```
label{
left:10%;
text-align:right;
width:30%;
height:50px
}


textfield,combo,group,scroll{
left:45%;
width:200px;
height:50px
}
```

Potom definišemo udaljenost od gornje ivice forme za parove labela – ulazno polje, npr:

```
#registration,#registrationField{
top:0%
}

#manufacturer,#manufacturerCombo{
top:10%
}
```

Sličnim postupkom definišemo položaj i za ostale parove.

Panel sa tabovima možemo da pozicioniramo nezavisno od upravo definisanog rasporeda:

```
#accesories{
top:60%;
left:45%;
height:120px;
width:200px
}
```

Ovde možemo da primetimo da za razliku od korišćenja *LayoutManager*-a za *Swing*, upotreba istog rasporeda nije obavezujuća na celoj komponenti.

Svaki tab predstavlja još jedan kontejner za *checkbox* elemente, tako da njihov razmeštaj unutar tabova možemo da realizujemo na isti način kao i raspored elemenata u glavnoj formi:

```
checkbox{
height:30px;
width:70%
}

#airbag,#dvdPlayer{
top:0px
}

#gps,#leatherSeats{
top:30px
}

#childSeat,#stereoSpeakers{
top:60px
}
```

Time smo pomoću relativno jednostavnog CSS dokumenta postigli veoma fleksibilnu alternativu *LayoutManager*-a koja je pritom platformski nezavisna, jer referencira samo opšte poznat CSS standard. U slučaju potrebe da se menja raspored elemenata na formi, neophodno je menjati samo ovaj dokument, što predstavlja još jednu prednost ovog pristupa.

Jedina svojstva koja nisu vezana za položaj i dimenzije su poravnanje labela i okvir oko grupe sa radio dugmićima:

```
label{
left:10%;
text-align:right;
width:30%;
height:50px
}

#typeGroup{
height:100px;
border-style:solid;
border-width:thin
}
```

Kada sastavimo prethodne odlomke i dopišemo svojstva za ostale elemente dobijamo ceo CSS opis:

```
#newcar{
width:700px;
height:700px
}

label{
left:10%;
text-align:right;
width:30%;
height:50px
}


textfield,combo,group,scroll{
left:45%;
width:200px;
height:50px
}

#registration,#registrationField{
top:0%
}

#manufacturer,#manufacturerCombo{
top:10%
}

#model,#modelCombo{
top:20%
}

#owner,#ownerCombo{
top:30%
}

#type,#typeGroup{
top:40%
}

#typeGroup{
height:100px;
border-style:solid;
border-width:thin
}

radio{
width:90%;
height:30px;
left:10px
}
```

```
#standard{
top:5px
}

#convertible{
top:35px
}

#minivan{
top:65px
}

#accesories{
top:60%;
left:45%;
height:120px;
width:200px
}

tab{
height:100%;
width:100%
}

checkbox{
height:30px;
width:70%
}

#airbag,#dvdPlayer{
top:0px
}

#gps,#leatherSeats{
top:30px
}

#childSeat,#stereoSpeakers{
top:60px
}

#description,#descriptionScroll{
top:80%
}

#descriptionScroll{
height:100px
}

#descriptionArea{
height:100%;
width:100%
}

#save{
height:30px;
width:80px;
left:52%;
top:95%
}
```

Sada je na redu pisanje lokalizacionog dokumenta (datoteka *res/carsale.lang*):

```
newcar=New Car
registration=Plate numbers
manufacturer=Manufacturer
model=Model
owner=Owner
type=Type
standard=Standard
convertible=Convertible
minivan=Mini-van
safety=Safety
leisure=Leisure
airbag=Airbag
gps=GPS
childSeat=Child seat
dvdPlayer=DVD player
leatherSeats=Leather seats
stereoSpeakers=Stereo speakers
description=Description
save=Save
```

Ukoliko je potrebno prevesti celu aplikaciju na neki drugi jezik, dovoljno je promeniti ovaj dokument.

Kao implementaciju koristimo za sada jedini generator koji podržava *Simple View*, a to je *SwingGuiBuilder*. Potrebno je smestiti *SwingGuiBuilder.jar* i *CSSToolkit.jar* na *classpath* projekta.

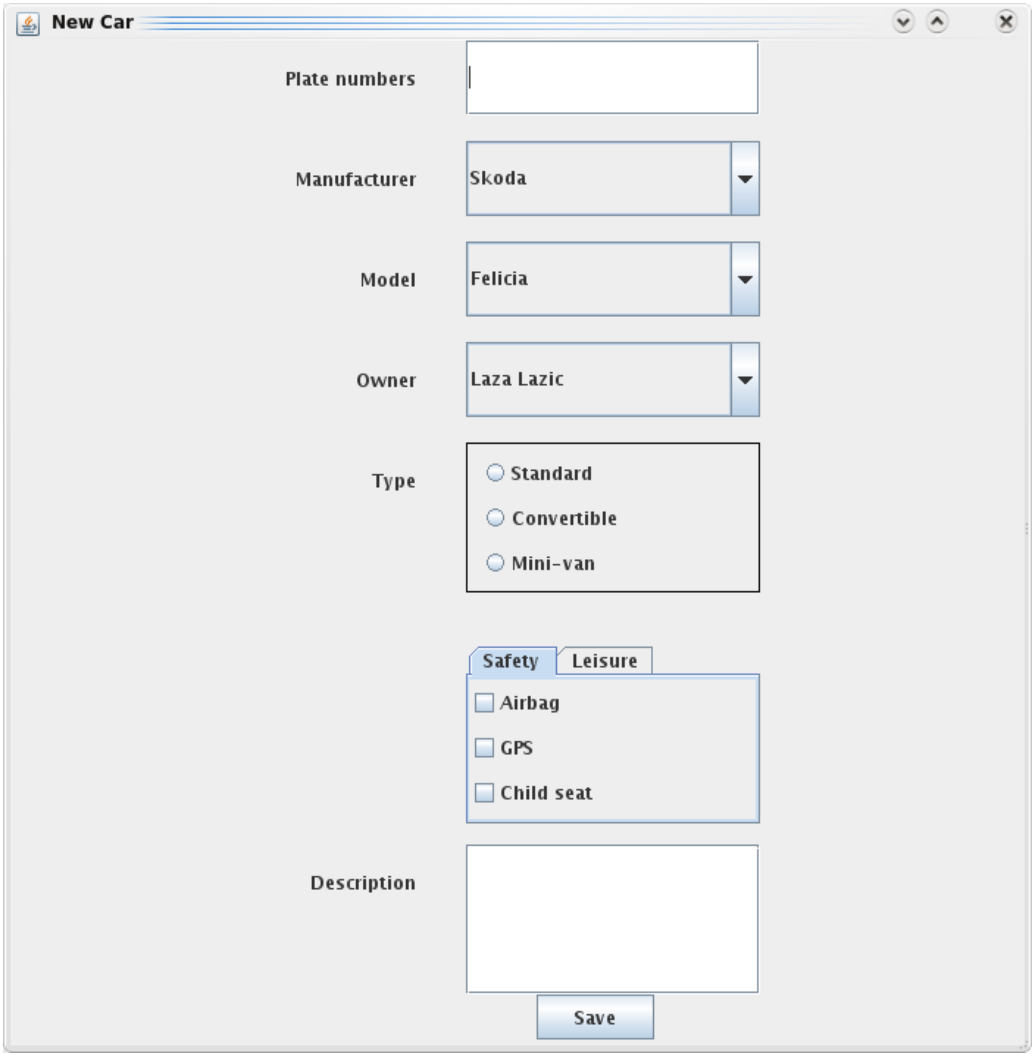
Napisana je *View* komponenta, tako da je preostao još samo *Controller*. Kontrolerska klasa je zadužena za instanciranje *SwingGui* klase na osnovu upravo sačinjenih dokumenata, popunjavanje elemenata početnim podacima, obradu događaja, konvertovanje vrednosti iz ulaznih elemenata, pozive metoda iz aplikacione logike i prikazivanje izlaznih podataka na *View* komponenti.

```
public class CarSaleController implements ItemListener, ActionListener {

	private SwingGui gui;
	private DBBroker dbb = new DBBroker();

	public CarSaleController(){

		// kreiranje gui objekta
		gui = new SwingGui();
		gui.setDescriptor("res/carsale.xml");
		gui.setStyle("res/carsale.css");
		gui.setLocale("res/carsale.lang");
		gui.construct();

		//popunjavanje početnih vrednosti dobijenih iz baze
		JComboBox manufacturerList = (JComboBox) gui.findComponent("manufacturerCombo");
		fillCombo(manufacturerList, dbb.getAllManufacturers());
		//dodavanje handlera za događaj
		manufacturerList.addItemListener(this);

		//opet popunjavanje početnih vrednosti dobijenih iz baze
		JComboBox ownerCombo = (JComboBox) gui.findComponent("ownerCombo");
		fillCombo(ownerCombo, dbb.getAllOwners());

		//još jedno dodavanje handlera za događaj
		JButton save = (JButton) gui.findComponent("save");
		save.addActionListener(this);

		selectModels();

		//prikazivanje forme
		gui.show();

	}

	//pomoćna metoda
	private void fillCombo(JComboBox combo, List<?> data){
		combo.removeAllItems();
		for(Object item : data){
			combo.addItem(item);
		}
	}

	//pomoćna metoda
	private void selectModels() {
		JComboBox manufacturerList = (JComboBox) gui.findComponent("manufacturerCombo");
		CarManufacturer cm = (CarManufacturer) manufacturerList.getSelectedItem();
		JComboBox modelList = (JComboBox) gui.findComponent("modelCombo");
		fillCombo(modelList, cm.getModels());
	}

	// ... ostale metode

}
```

Da bi kontroler mogao da se pokrene negde je potrebna *main* metoda. Zato pišemo posebnu klasu za pokretanje programa:

```
public class LaunchApp {

	public static void main(String[] args) {

		CarSaleController controller = new CarSaleController();
	}
}
```

Nakon pokretanja *main* metode, pojavljuje se forma sa slike 15.



Slika 15: Napredni primer - Prvo pokretanje

Možemo da vidimo da su svi elementi formatirani u skladu sa zahtevima i da prilikom promene veličine prozora ostaju raspoređeni prema istoj proporciji (što je moguće i proveriti pokretanjem ovog programa sa diska iz priloga)

Kontroler još uvek ne sadrži metode za obradu događaja, tako da klik na dugme *Save* ne stvara nikakav efekat i spisak ponuđenih modela automobila još uvek ne zavisi od odabranog proizvođača.

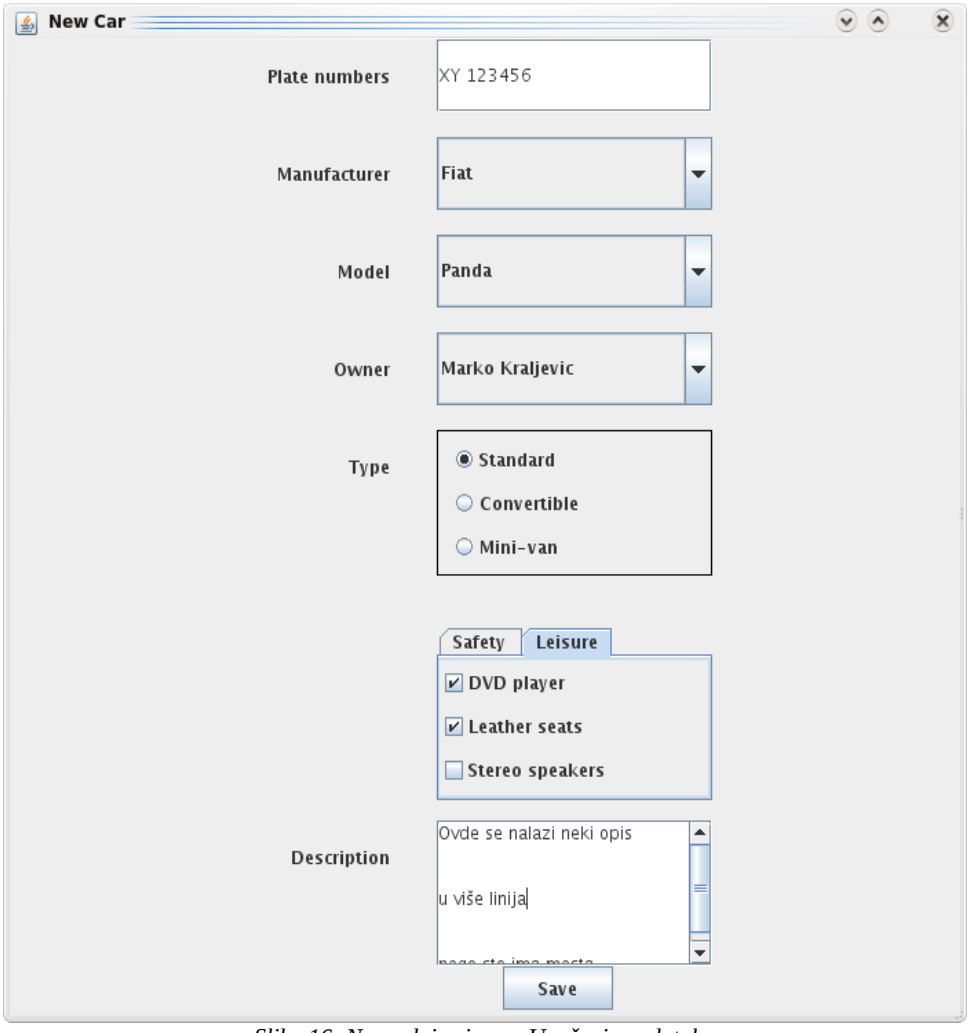
Da bismo to omogućili, dodajemo nove metode u klasu *CarSaleController*:

```
@Override
public void itemStateChanged(ItemEvent ie) {
	//kada se promeni selektovani proizvođač
	//menja se spisak ponuđenih modela
	selectModels();
}
@Override
public void actionPerformed(ActionEvent event) {
	Component source = (Component) event.getSource();
	if (source.getName().equals("save")) {
		handleSaveButtonAction();
	}

}

public void handleSaveButtonAction() {
	//instancira domenske objekte
	Car car = new Car();

	//vadi podatke iz komponenata i
	//ubacuje ih u svojstva objekta...
	JTextField reg = (JTextField) gui.findComponent("registrationField");
	car.setRegistration(reg.getText());

	JComboBox mod = (JComboBox) gui.findComponent("modelCombo");
	CarModel model = (CarModel) mod.getSelectedItem();
	car.setModel(model);

	JComboBox own = (JComboBox) gui.findComponent("ownerCombo");
	Owner owner = (Owner) own.getSelectedItem();
	car.setOwner(owner);

	CarType type = null;
	JRadioButton standard = (JRadioButton) gui.findComponent("standard");
	if(standard.isSelected()) type = CarType.STANDARD;
	JRadioButton convertible = (JRadioButton) gui.findComponent("convertible");
	if(convertible.isSelected()) type = CarType.CONVERTIBLE;
	JRadioButton minivan = (JRadioButton) gui.findComponent("minivan");
	if(minivan.isSelected()) type = CarType.MINIVAN;
	car.setType(type);

	JCheckBox airbag = (JCheckBox) gui.findComponent("airbag");
	car.setAirbag(airbag.isSelected());

	JCheckBox gps = (JCheckBox) gui.findComponent("gps");
	car.setGps(gps.isSelected());

	JCheckBox childSeat = (JCheckBox) gui.findComponent("childSeat");
	car.setChildSeat(childSeat.isSelected());

	JCheckBox dvdPlayer = (JCheckBox) gui.findComponent("dvdPlayer");
	car.setDvdPlayer(dvdPlayer.isSelected());

	JCheckBox stereoSpeakers = (JCheckBox) gui.findComponent("stereoSpeakers");
	car.setStereoSpeakers(stereoSpeakers.isSelected());

	JCheckBox leatherSeats = (JCheckBox) gui.findComponent("leatherSeats");
	car.setLeatherSeats(leatherSeats.isSelected());

	JTextArea descriptionArea = (JTextArea) gui.findComponent("descriptionArea");
	car.setDescription(descriptionArea.getText());

	//izvršava operacije aplikacione logike
	dbb.saveCar(car);

	//na formi prikazuje izlazne podatke
	JFrame frame = (JFrame) gui.findComponent("newcar");
	JOptionPane.showMessageDialog(frame, car.toString());
}
```

Za ispisivanje podataka o automobilu koristimo metodu *toString()* klase *Car* koja ispisuje podatke u formatu:

```
Reg: Broj tablica
Manuf: Naziv proizvođača
Model: Naziv modela
Owner: Ime i prezime vlasnika
Type: Tip vozila
---------------------------
Accessories:
spisak svih dodataka
---------------------------
Description:
opis vozila
---------------------------
```

Ponovo pokrećemo program, unosimo proizvoljne podatke (kao na slici 16) i pritiskamo taster *Save*, nakon čega se pojavljuje odzivna poruka sa slike 17.



Slika 16: Napredni primer - Unošenje podataka

Slika 17: Odzivna poruka

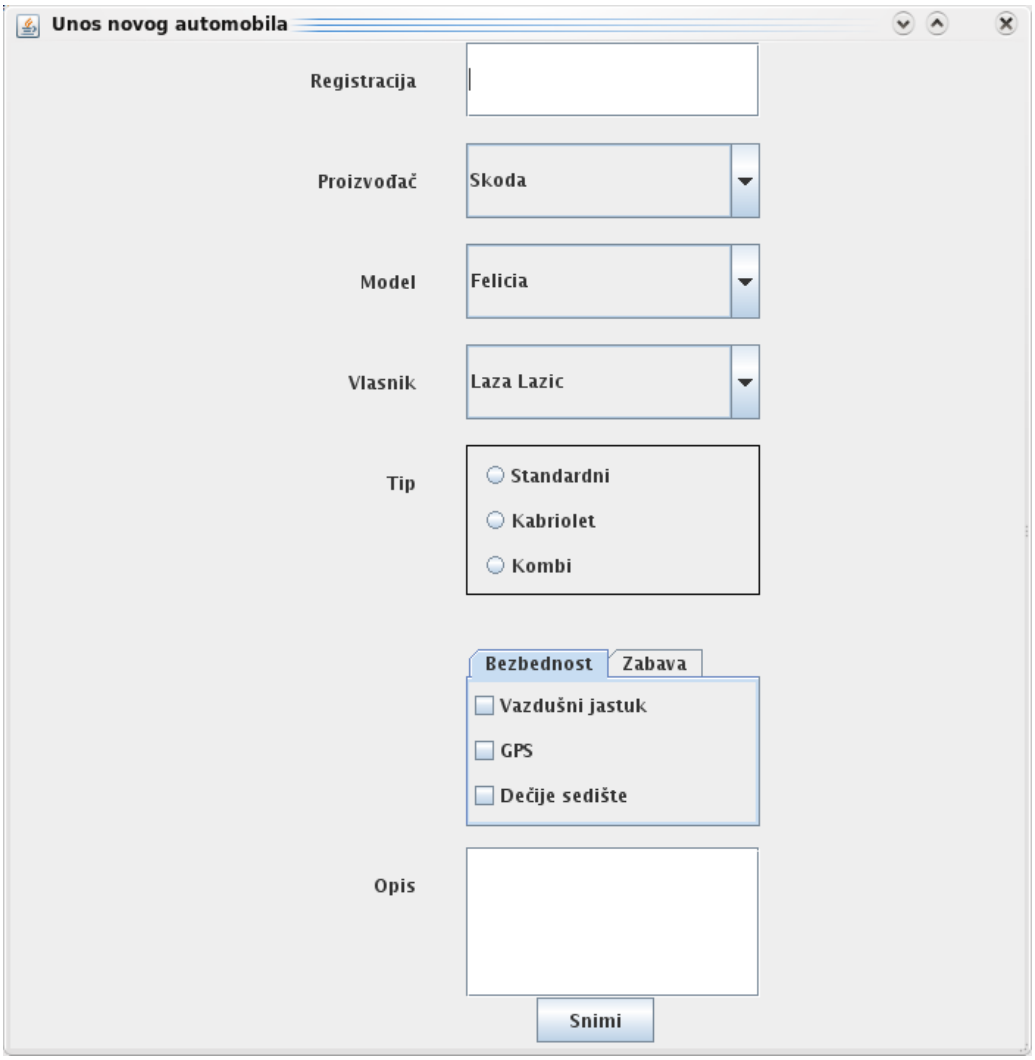
U svega nekoliko koraka smo napravili grafički korisnički interfejs za pojednostavljen, ali potpuno funkcionalan softverski sistem. Međutim, jednostavnost početnog kreiranja korisničkog interfejsa nije jedina prednost *Simple View* pristupa. Druga, verovatno mnogo bitnija prednost, je mogućnost lakog održavanja. U nastavku je dat primer mogućih izmena prilikom održavanja softvera kreiranog ovim postupkom.

Zahtev za izmenu #1:
Potrebno je celu aplikaciju prevesti na srpski jezik.

Kao što je ranije rečeno, da bismo preveli aplikaciju, treba da promenimo sadržaj lokalizacionog dokumenta. Nakon izmene ovaj dokument ima sledeći sadržaj:

```
newcar=Unos novog automobila
registration=Registracija
manufacturer=Proizvođač
model=Model
owner=Vlasnik
type=Tip
standard=Standardni
convertible=Kabriolet
minivan=Kombi
safety=Bezbednost
leisure=Zabava
airbag=Vazdušni jastuk
gps=GPS
childSeat=Dečije sedište
dvdPlayer=DVD plejer
leatherSeats=Kožna sedišta
stereoSpeakers=Stereo zvučnici
description=Opis
save=Snimi
```

A forma nakon izmene izgleda kao na slici 18.



Slika 18: Napredni primer - Lokalizovana verzija

Zahtev za izmenu #2:
Potrebno je postojeću formu vizuelno ukrasiti drugim skupom boja i fontova.

Forma treba da bude u nijansama zelene, tekst labela treba da bude plave boje, tekst u komboima žute, a u tekstualnim poljima crvene boje. Pozadinska boja tekstualnih polja treba da bude žuta, a dugme *Snimi* treba da ima tekst žute boje na plavoj pozadini. Takođe, font u polju za unos registracije treba da bude podebljan, a u polju za unos opisa u italik stilu.

Potrebne izmene izvršavaju se samo na jednom mestu, a to je CSS datoteka. Da bismo promenili skup boja treba da dodamo sledeći blok koda:

```
form{
background-color:#8ADB68
}

combo,group,tab,tabbed-panel{
background-color:#54AF23
}

radio,checkbox{
background-color:transparent
}

textfield,textarea{
background-color:rgb(226,245,54);
color:red
}

#save{
background-color:#2E7DCC
}

combo,radio,checkbox,button{
color:yellow
}

label{
color:blue
}
```

A za izmenu osobina fonta treba dodati:

```
#registrationField{
font-weight:bold
}

#descriptionArea{
font-style:italic
}
```

Izmenjena forma prikazana je na slici 19.

Slika 19: Napredni primer – Verzija sa promenjenim bojama

Pitanje umeća autora ovog rada da odabere prave boje ostaje diskutabilno. U svakom slučaju primer jasno pokazuje da primenom CSS-a dobijamo jednostavnu alternativu za postizanje proizvoljnog *look-and-feel*-a korisničkog interfejsa, a na projektantima je da te mogućnosti upotrebe na pravi način.

Zahtev za izmenu #3:
Potrebno je dodati novi element na formu. Dodavanje novog elementa je tip izmene koji je najkomplikovaniji za održavanje programa pisanog pomoću *Simple View*-a, zbog toga što je najčešće neophodno menjati sva tri opisa. Međutim, u narednom primeru ćemo da vidimo da uprkos tome promena predstavlja jednostavan zadatak, zato što se opisi lako menjaju.

Dodaćemo pomoćni meni koji se aktivira desnim klikom na polje za opis automobila. Meni sadrži samo jednu opciju koja se zove *Google pretraga*. Željena funkcionalnost novog menija je da pomoću Google pretraživača pronađe sve web stranice gde se pojavljuje trenutno selektovan tekst iz opisa.

Element koji predstavlja pomoćni meni zove se *popup-menu* i koristi se tako što se ugnezdi u element nad kojim treba da se poziva. Pošto želimo da se meni pojavi kada korisnik klikne desnim tasterom miša na tekstualno polje za unos opisa, *popup-menu* smeštamo unutar tog elementa i dodajemo mu jedan *menu-item* element, koji predstavlja njegovu jedinu opciju:

```
<label id="description"/>
<scroll id="descriptionScroll">
	<textarea id="descriptionArea">
		<popup-menu id="popup">
			<menu-item id="google"/>
		</popup-menu>
	</textarea>
</scroll>
```

Dodavanjem pomoćnog menija, automatski su postavljeni i *event listener*-i koji aktiviraju meni prilikom klika desnim tasterom miša na njegovu roditeljsku komponentu, tako da je programer oslobođen pisanja koda zaduženog za taj posao.

U ovom primeru nije neophodna izmena CSS dokumenta, zbog toga što meniji ne zahtevaju eksplicitno podešavanje pozicije i veličine. U drugim slučajevima, kada to jeste potrebno, CSS dokument menjamo na način koji je opisan ranije.

U lokalizacioni dokument za srpski jezik dodajemo još jedan ključ za novi element:

```
google=Google pretraga
```

I u odgovarajući dokument dodajemo isti ključ za engleski jezik:

```
google=Google search
```

Sada treba dodati željenu funkcionalnost u kontrolersku klasu. Treba promeniti *event handler* metodu:

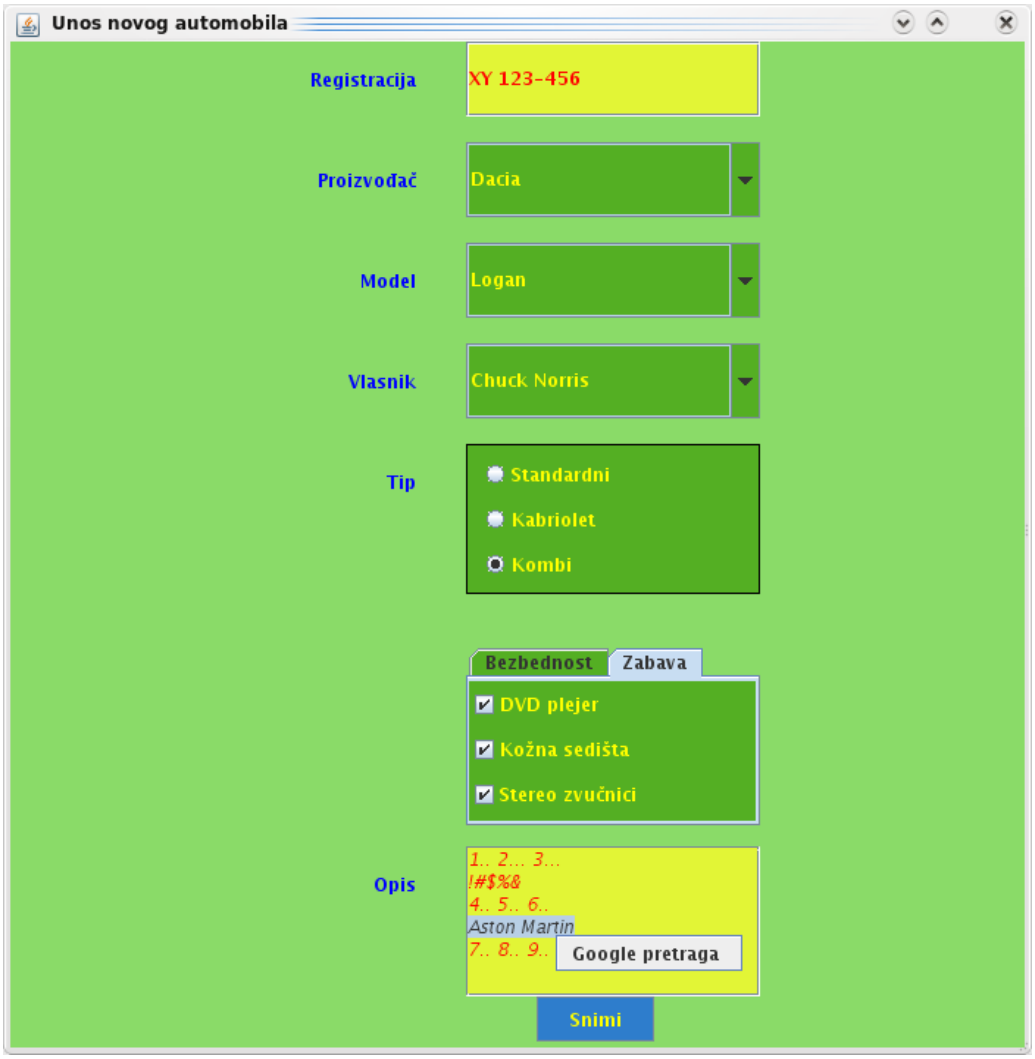
```
@Override
public void actionPerformed(ActionEvent event) {
	Component source = (Component) event.getSource();
	if (source.getName().equals("save")) {
		handleSaveButtonAction();
	}else if(source.getName().equals("google")){
		handleGoogleAction();
	}
}
```

I dodati novu metodu:

```
private void handleGoogleAction() {

	JTextArea descriptionArea = (JTextArea) gui.findComponent("descriptionArea");
	String selected = descriptionArea.getSelectedText();

	String browser = "firefox";[16]
	String url = "http://www.google.com/search?q="+selected;;

	try {
		Runtime.getRuntime().exec(new String[]{browser, url});
	} catch (IOException e) {}}
```

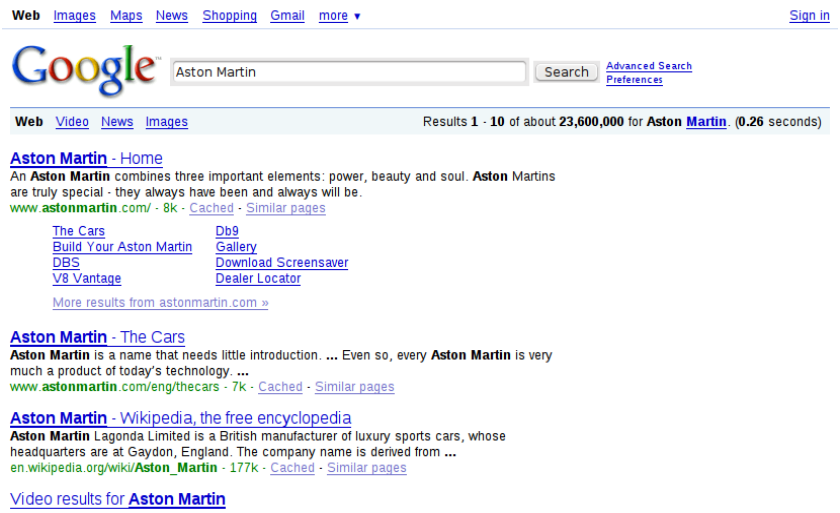
16 Zbog očuvanja jednostavnosti, primer je ograničen na Mozilla Firefox browser, čiji glavni izvršni program mora da se nalazi na sistemskoj PATH putanji.

Ponovo pokrećemo izmenjenu aplikaciju i testiramo novu funkcionalnost. Izmenjena forma je prikazana na slici 20.



Slika 20: Napredni primer - Verzija sa web pretragom

U polje za opis je unet probni tekst i selektovan deo celog teksta koji glasi “Aston Martin”. Klikom desnog tastera miša pojavljuje se pomoćni meni sa opcijom *Google pretraga*. Nakon odabira te opcije u Firefox čitaču se otvara stranica sa rezultatima pretrage prikazana na slici 21.



Slika 21: Napredni primer - Rezultati web pretrage

Na konkretnom primeru je ilustrovan ceo proces projektovanja, realizacije i održavanja grafičkog korisničkog interfejsa izrađenog prema *Simple View* specifikaciji, koristeći *SwingGuiBuilder* implementaciju.

Pokazano je da predloženi pristup omogućava jednostavnost projektovanja i prilagodljivost prilikom održavanja, kao i da je, zahvaljujući adekvatnoj sintaksi i visokom nivou apstrakcije na kome se definiše korisnički interfejs, potrebno relativno malo programskog koda za ostvarivanje željenog efekta.

U sledećem dodatku se nalazi potpuna *Simple View* specifikacija, gde je moguće pronaći sve potrebne informacije o podržanim elementima i stilovima.

Dodatak B: Kompletna Simple View specifikacija

Spisak svih podržanih komponenata i njihovih odgovarajućih XML elemenata:

Element	Opis	Napomena
form	Osnovni prozor	Isključivo koreni element
label	Labela	/
textfield	Jednolinijsko tekstualno polje	/
textarea	Višelinijsko tekstualno polje	/
button	Dugme	/
radio	Radio dugme	/
checkbox	Checkbox dugme	/
panel	Panel	Grupiše sadržane elemente
group	Grupa radio dugmića	Obezbeđuje međusobnu isključivost sadržanih radio dugmića
combo	Kombo	/
list	Lista	/
table	Tabela	/
passwordfield	Polje za unos šifre	/
image	Dekorativna slika	/
icon	Interaktivna slika	/
toolbar	Paleta alata	/
menu-bar	Linija sa menijima	Sadrži samo menu elemente
menu	Meni	Ubacuje se samo u menu-bar, menu ili popup-menu element
menu-item	Stavka menija	Ubacuje se samo u menu ili popup-menu element
popup-menu	Pomoćni (kontekst) meni	Ubacuje se u element nad kojim treba da se poziva
tabbed-panel	Panel sa tabovima	Sadrži samo tab elemente
tab	Tab	Ubacuje se samo u tabbed-panel element
scroll	Panel sa skrolovima	Obmotava element kome treba da omogući funkcionalnost skrolovanja

Tabela 4: Specifikacija - Spisak svih elemenata

Spisak podržanog podskupa CSS-a je kreiran izmenom specifikacije osnovnog standarda sa sajta W3C [26]. Svojstva koja nisu podržana su izbačena sa spiska. U slučaju da je neko svojstvo podržano, ali da nisu podržane sve njegove vrednosti, tada su nepodržane vrednosti prikazane kao precrtane.

Pozadina

Svojstvo	**Opis**	**Vrednosti**
background-color	Pozadinska boja elementa	*color-rgb* *color-hex* *color-name* transparent
background-image	Pozadinska boja elementa[17]	url(*URL*) none

Tabela 5: Specifikacija - CSS svojstva iz kategorije 'Pozadina'

Okvir

Svojstvo	**Opis**	**Vrednosti**
border-color	Boja okvira	*color*
border-style	Stil okvira	none hidden ~~dotted~~ ~~dashed~~ solid ~~double~~ groove ridge inset outset
border-width	Debljina okvira	thin medium thick *length*

Tabela 6: Specifikacija - CSS svojstva iz kategorije 'Okvir'

17 U aktuelnoj verziji *SwingGuiBuilder* implementacije radi samo na *icon* i *image* elementima.

Klasifikacija

Svojstvo	**Opis**	**Vrednost**
cursor	Tip kursora koji treba da se koristi dok se pointer nalazi unutar zone elementa	*url* auto crosshair default pointer move e-resize ne-resize nw-resize n-resize se-resize sw-resize s-resize w-resize text wait ~~help~~

Tabela 7: Specifikacija - CSS svojstva iz kategorije 'Klasifikacija'

Dimenzije

Svojstvo	**Opis**	**Vrednosti**
height	Visina elementa	auto *length* *%*
max-height	Maksimalna visina elementa	none *length* *%*
max-width	Maksimalna širina elementa	none *length* *%*
min-height	Minimalna visina elementa	*length* *%*
min-width	Minimalna širina elementa	*length* *%*
width	Širina elementa	auto *%* *length*

Tabela 8: Specifikacija - CSS svojstva iz kategorije 'Dimenzije'

Font

Svojstvo	**Opis**	**Vrednosti**
font-family	Naziv fonta	*family-name* *generic-family*
font-size	Veličina fonta	xx-small x-small small medium large x-large xx-large smaller larger *length* *%*
font-stretch	Vrši horizontalno širenje ili skupljanje trenutnog fonta elementa nad kojim se primenjuje	normal ~~wider~~ ~~narrower~~ ~~ultra-condensed~~ ~~extra-condensed~~ condensed semi-condensed semi-expanded expanded ~~extra-expanded~~ ~~ultra-expanded~~
font-style	Stil fonta	normal italic oblique
font-weight	Debljina fonta	normal bold ~~bolder~~ ~~lighter~~ 100 200 300 400 500 600 700 800 900

Tabela 9: Specifikacija - CSS svojstva iz kategorije 'Font'

Pozicioniranje

Svojstvo	**Opis**	**Vrednost**
bottom	Udaljenost od donje ivice roditeljskog elementa	auto *%* *length*
left	Udaljenost od leve ivice roditeljskog elementa	auto *%* *length*
right	Udaljenost od desne ivice roditeljskog elementa	auto *%* *length*
top	Udaljenost od gornje ivice roditeljskog elementa	auto *%* *length*
z-index	Dubina elementa, tj. prioritet za renderovanje	auto *number*

Tabela 10: Specifikacija - CSS svojstva iz kategorije 'Pozicioniranje'

Tekst

Svojstvo	**Opis**	**Vrednost**
color	Boja teksta	*color*
direction	Smer teksta	ltr rtl
text-align	Poravnanje teksta	left right center ~~justify~~
text-decoration	Ukrašavanje teksta	none underline ~~overline~~ line-through ~~blink~~
text-transform	Vrši transformaciju malih slova u velika ili obrnuto	none capitalize uppercase lowercase

Tabela 11: Specifikacija - CSS svojstva iz kategorije 'Tekst'

Odvojeno od *Simple View* specifikacije, za svaku implementaciju, ako postoji potreba za to, treba napisati tabelu preslikavanja elemenata iz specifikacije u komponente iz konkretne tehnologije.

Za *SwingGuiBuilder* implementaciju tabela pokazuje koji elementi se preslikavaju u objekte koje Swing klase, što daje odgovor na pitanje u koji tip treba da se prevede objekat dobijen prilikom poziva metode *findComponent(id)*.

Simple View element	**Swing klasa**
form	JFrame
label	JLabel
textfield	JTextField
textarea	JTextArea
button	JButton
radio	JRadioButton
checkbox	JCheckBox
panel	JPanel
group	JPanel[18]
combo	JComboBox
list	JList
table	JTable
passwordfield	JPasswordField
image	JLabel
icon	JButton
toolbar	JToolBar
menu-bar	JMenuBar
menu	JMenu
menu-item	JMenuItem
popup-menu	JPopupMenu
tabbed-panel	JTabbedPane
tab	JPanel[19]
scroll	JScrollPane

Tabela 12: Preslikavanje Simple View elemenata u Swing komponente

18 Može da se koristi i podklasa *org.goranjovic.guibuilder.ButtonGroupPanel* iz SwingGuiBuilder paketa
19 Može da se koristi i podklasa *org.goranjovic.guibuilder.Tab* iz SwingGuiBuilder paketa

Pogovor

Praktični primeri sadržani u ovom radu su pokazali da se naša tehnika odlikuje brojnim prednostima u odnosu na dosadašnje. Ipak, najbolji pokazatelj uspešnosti nije nikakav hipotetički primer, već zastupljenost u stvarnoj praksi.

Ohrabrujemo sve programere da koriste *Simple View* model i *SwingGuiBuilder* implementaciju u svojim projektima. *SwingGuiBuilder* projekat je objavljen kao slobodan softver pod LGPL (Lesser GNU General Publishing Licence) licencom, prema kojoj se određuju uslovi korišćenja i redistribuiranja.

Takođe, da *Simple View* ne bi bio tehnološki nezavisan samo u teoriji, a u praksi podržan samo za *Swing*, pozivamo sve programere, stručnjake za web, softverske inženjere i ostale zainteresovane da u duhu razvoja slobodnog softvera daju svoj doprinos i razviju *Simple View* implementaciju za svoju omiljenu tehnologiju grafičkog korisničkog interfejsa.

Nadamo se da će ovaj projekat podstaći razvoj mnoštva novih *Simple View* implementacija za razne prezentacione tehnologije, kako bismo svi imali koristi od toga.

Literatura

1: Dr. Siniša Vlajić, *Projektovanje programa (Skripta)*, 2006

2: Eric Steven Raymond, Rob W. Landley, *The Art of Unix Usability*, 2004

3: Nathan Lineback, *The Macintosh in 1984*, http://toastytech.com/guis/macos1.html

4: Andries van Dam, *Post-WIMP User Interfaces*, 1997

5: Ashley George Taylor, *Wimp Interfaces*, 1997

6: Brad Myers, *User Interface Software Tools*, 1994

7: Brad A. Myers, *Graphical User Interface Programming*, 2003

8: Sophie Lepreux, Jean Vanderdonckt, Benjamin Michotte, *Visual Design of User Interfaces by (De)composition*, 2006

9: Kris Luyten, Karin Coninx, Jan Meskens, Jo Vermeulen, *Generating Domain-specific Interface Builders for Multi-Device User Interface Creation*, 2008

10: Wikipedia, *List of user interface markup languages*, http://en.wikipedia.org/wiki/List_of_user_interface_markup_languages

11: Nathalie Souchon, Jean Vanderdonckt, *A Review of XML-compliant User Interface Description Languages*, 2003

12: Mir Farooq Ali, Manuel A. Pérez-Quiñones, Marc Abrams, Eric Shell, *Building Multi-Platform User Interfaces With UIML*, 2001

13: Constantinos Phanouriou, *UIML: A Device-Independent User Interface Markup Language*, 2000

14: Marc Abrams, James Helms, *User Interface Markup Language (UIML) Specification*, 2004

15: Mikko Pohja, Mikko Honkala, Miemo Penttinen, Petri Vuorimaa,Panu Ervamaa, *Web User Interaction - Comparison of Declarative Approaches*, 2006

16: Christopher Alexander, *The Timeless Way of Building*, 1979

17: James Coplien, Dick Gabriel, *Pattern Definition*, http://www.hillside.net/patterns/definition.html

18: Wikipedia, *Model-view-controller*, http://en.wikipedia.org/wiki/Model-view-controller

19: Trygve Reenskaug, *Models-Veiws-Controllers*, 1979

20: Miloš Milić, *Komparativna analiza Spring i EJB tehnologije*, 2007

21: E. W. Dijkstra, *On the role of scientific thought*, 1974

22: Siniša Vlajić, Dušan Savić, Vojislav Stanojević, Ilija Antović, Miloš Milić, *Projektovanje*

softvera - Napredne Java tehnologije, 2008

23: Marija Vidaković, *Platformski neutralne arhitektureza složene (Enterprise) softverske sisteme*, 2008

24: John O'Conner, *Using the Swing Application Framework (JSR 296)*, 2007

25: David Fraser, Michael Harris, *Speed up your Swing GUI construction with better building blocks*, http://www.javaworld.com/javaworld/jw-10-2002/jw-1004-dialog.html?page=8

26: W3C, *CSS2 Reference*, http://www.w3schools.com/CSS/CSS_reference.asp

BESPLATNI GOTOVI SEMINARSKI, DIPLOMSKI I MATURSKI RAD.

RADOVI IZ SVIH OBLASTI, POWERPOINT PREZENTACIJE I DRUGI EDUKATIVNI MATERIJALI.

WWW.SEMINARSKIRAD.ORG

WWW.MATURSKIRADOVI.NET

WWW.MATURSKI.NET

WWW.SEMINARSKIRAD.INFO

WWW.MATURSKI.ORG

WWW.ESSAYSX.COM

WWW.FACEBOOK.COM/DIPLOMSKIRADOVI

NA NAŠIM SAJTOVIMA MOŽETE PRONAĆI SVE, BILO DA JE TO SEMINARSKI, DIPLOMSKI ILI MATURSKI RAD, POWERPOINT PREZENTACIJA I DRUGI EDUKATIVNI MATERIJAL. ZA RAZLIKU OD OSTALIH MI VAM PRUŽAMO DA POGLEDATE SVAKI RAD, NjEGOV SADRŽAJ I PRVE TRI STRANE TAKO DA MOŽETE TAČNO DA ODABERETE ONO ŠTO VAM U POTPUNOSTI ODGOVARA. U BAZI SE NALAZE GOTOVI SEMINARSKI, DIPLOMSKI I MATURSKI RADOVI KOJE MOŽETE SKINUTI I UZ NJIHOVU POMOĆ NAPRAVITI JEDINSTVEN I UNIKATAN RAD. AKO U BAZI NE NAĐETE RAD KOJI VAM JE POTREBAN, U SVAKOM MOMENTU MOŽETE NARUČITI DA VAM SE IZRADI NOVI, UNIKATAN SEMINARSKI ILI NEKI DRUGI RAD RAD NA LINKU IZRADA RADOVA. PITANjA I ODGOVORE MOŽETE DOBITI NA NAŠEM FORUMU ILI NA MATURSKIRADOVI.NET@GMAIL.COM